DESIGN DOCUMENT

CSC 440 Semester Project

Brian Vach, Ethan Perry, Jared Bryant, Chris Fitzgerald

Table of Contents

Iteration 1 Design Decisions:

The CSC 440 Iteration 1 Demo App was built using Ruby on Rails, taking advantage of the scaffolding feature that is made available in Rails. Upon loading the website, the user is presented with a list of courses, each with a link to questions related to the course. The user can then click on one of these links, and view the questions, or submit a new one. If the user clicks on a question, they are show answers for that question, and a link to submit new answers. At the top of every page is a nav bar with a link to the home courses page, and to a user management page, where new users can be created and existing users edited. There is also a search bar with functionality that has not been implemented yet. It was decided to use Bootstrap as the CSS framework because it integrates with Rails extremely well, and is very easy to implement. The courses, questions, and answers are displayed as bootstrap "cards" because the card offers a built-in border with rounded edges that we liked. For the headers of the card lists on the questions and answers pages, we used a "jumbotron" object that complements the cards well. Overall we chose a blue/grey theme (except for the search button, which is green for a complement).

Iteration 1 Database Design Decisions:

The database for the CSC Iteration 1 Demo App consists of 4 tables. These tables are "Courses", "Users", "Questions, and "Answers". The decision was made to limit the database to these 4 tables for now for simplicity. Later in the project, more tables may be added. First, let's look at the "Courses" table. It was decided that this table only needed three fields for now, Name, Description, and Professor. This was enough to get meaningful data stored for a demo. Next is "Users". This table contains Name, Email, and Course_ID. It was decided that for now, a user will only belong to one course. Later this will be changed. Next is the table "Questions". This table contains Course_ID, Title, User_ID, and Content. It was decided that a question can only belong to one class, and one user. We do not anticipate this to change. The last table is "Answers". This table is very similar to "Questions". It contains Question_ID, User_ID, and Content. An answer should only belong to one question and one user. Again, we do not expect this to change.

Iteration 2 Design Decisions:

For this iteration, we have added more functionality to the website. We have added additional use cases which include add user, edit user, authenticate user, logout user, vote on question, and vote on answer. All the added functionality allows for users to interact with the system on a personal level. In the first iteration, users would select which user they would want to post as, which could obviously cause issues with building credibility on the website. The main functionality that was added to this iteration was the addition of voting, which will be most noticeable to the user. Adding and editing users is more of an extension of what was already present in the previous system. We have yet to include a search bar as was wanted from iteration 1.

Iteration 2 Database Design Decisions:

For this iteration, we have kept our database design decisions the same with 2 small changes. We now have votes added as a field for both questions and answers. Since users are now able to vote on questions and answers, the database needs to keep track of how many votes each has. We have also added an “is_admin” field to the user’s table, to separate admins from non-admins.

Iteration 3 Design Decisions:

In this iteration, we have added additional functionality to the application. We have added the ability for admins to edit or delete any question or answer, and for users to edit and delete their own questions and answers. We have also added a registrar user role that is allowed to view users, and upload a CSV file containing a list of new users. Admins can also upload users via CSV. We have made improvements to the voting, by preventing users from voting on a question or answer more than once. The vote button is hidden once a user votes, so they know they have already voted on that question or answer. The vote count now automatically updates when clicking the vote button, instead of having to refresh the page. This was accomplished by using JQuery. We have added profile pages for all users, where the user can go to view their information like which classes they are in, or what their total accumulated score is. Lastly, we have incorporated a search feature, that searches through the content and titles of questions, and displays them sorted by score.

Iteration 3 Database Design Decisions:

In this iteration, we have made 2 major changes to our database. We have added a table for votes, containing a record for each vote made by one user to one question or answer. This allows us to join this table with User, Question, and Answer to retrieve the number of votes a question or answer has or the number of votes on questions and answers posted by a certain users. It also lets us determine if a user has voted on a question or answer before, so we can hide the vote buttons to prevent voting again. We have also added a many to many relationship between User and Course. This relationship involves a dynamically generated join table called UserCourse that is not pictured in this diagram. This relationship is enforced by the HABTM pattern in the User and Course model. This stands for Has And Belongs To Many. This causes rails to dynamically enforce the relationship, and allow for things like selecting User.courses, even though that column is not defined in User. Rails automatically generates the query to include the join.

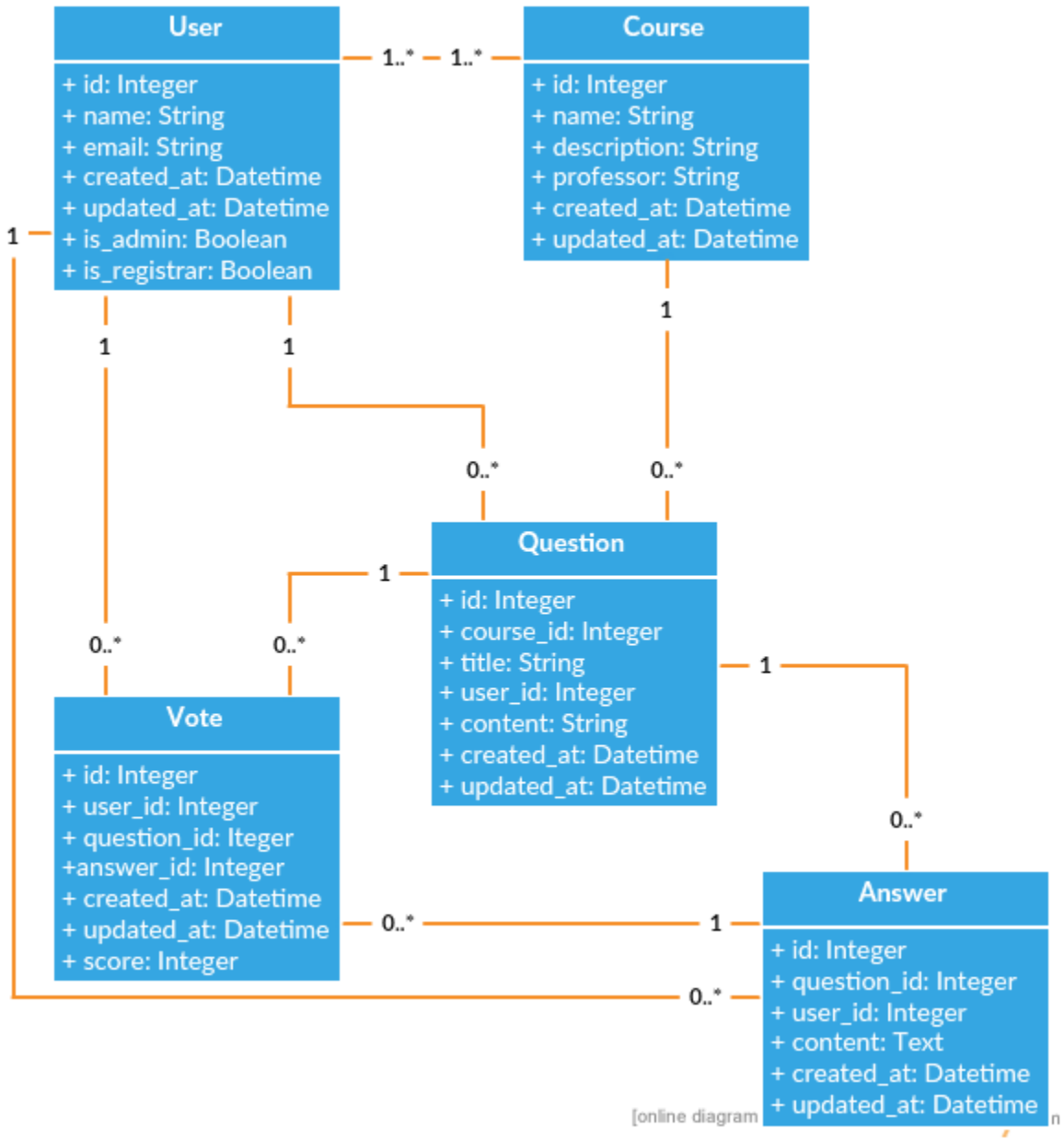
GRASP Patterns Identified:

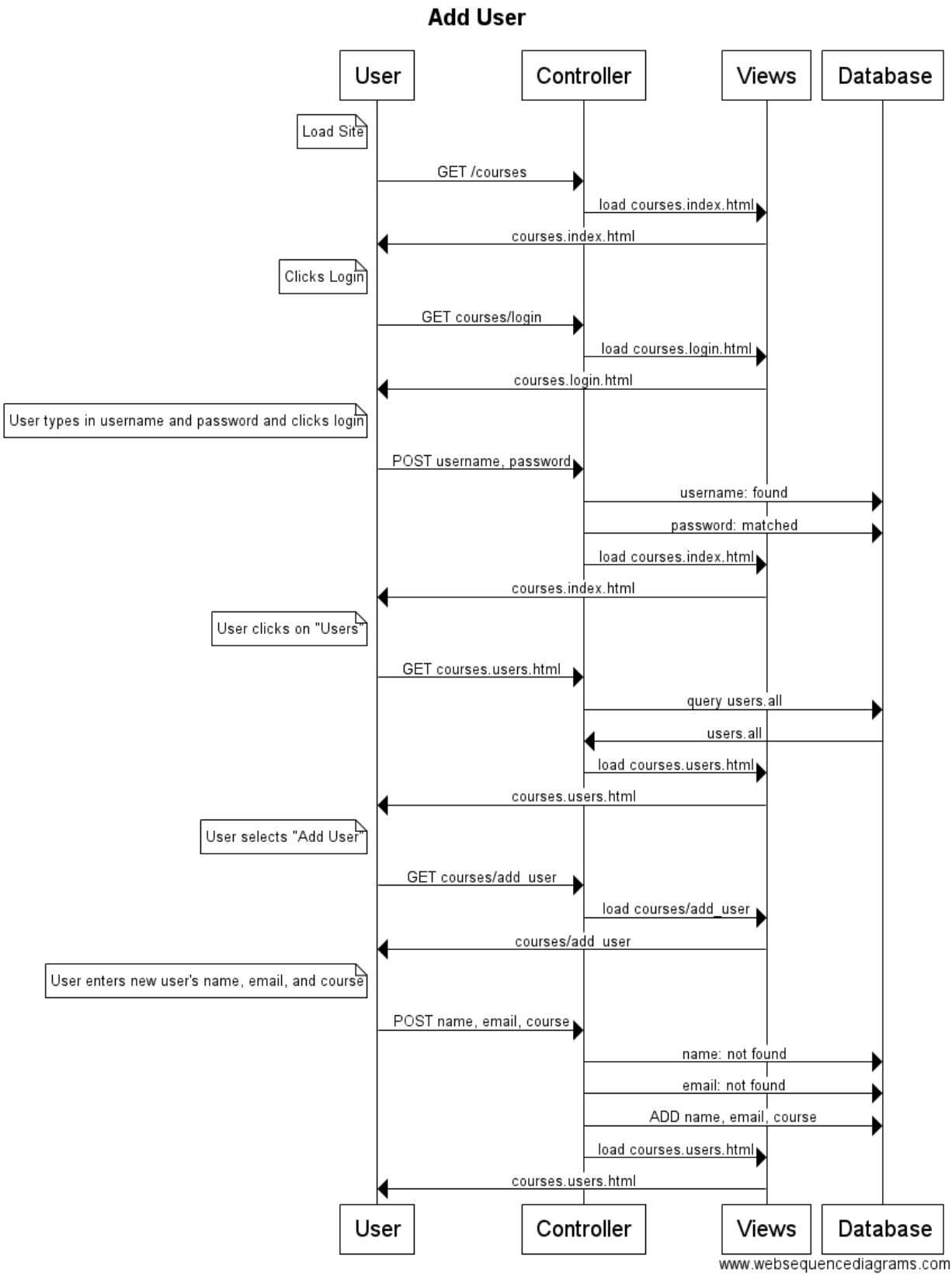
We have identified several GRASP patterns that were used in this project. The main pattern that we have identified is the Controller pattern. This pattern can be seen in almost every use case. This is likely because Rails is a Model, View, Controller style framework. We have also identified the Indirection pattern, which is similar to Controller, and also supports the MVC style of the Rails framework. Another pattern at work here, is High Cohesion. This pattern refers to how the application is broken down into related and highly focused components, which is again evidenced by almost all the use cases listed here. With the High Cohesion pattern comes Low Coupling. This pattern can be seen when we have a low dependency between classes, such as User and Course in the case of this application. A change to one does not strongly affect the other.

GOF Patterns Identified:

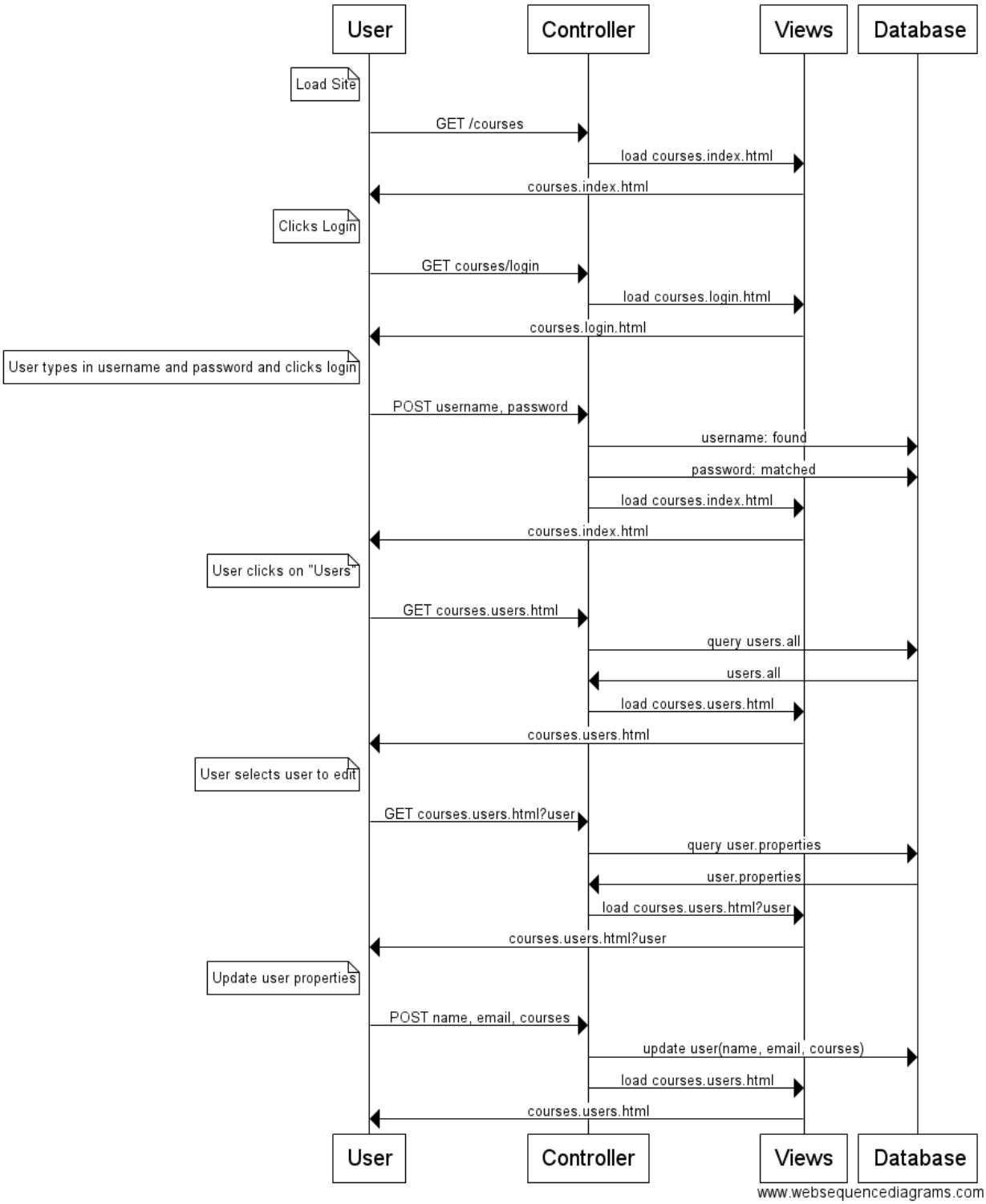
One of the GOF patterns we have identified is the “Chain of Responsibility”. This can be represented by the Rails routing configuration. The Router takes requests, and delegates them to the various controllers, similar to a class that delegates responsibilities to handlers. We can also see the Mediator pattern in action in the connection between User and Course. These classes communicate with eachother through the HABTM model in Rails. The UserCourse relationship is the mediator object itself.

Design Class Diagram

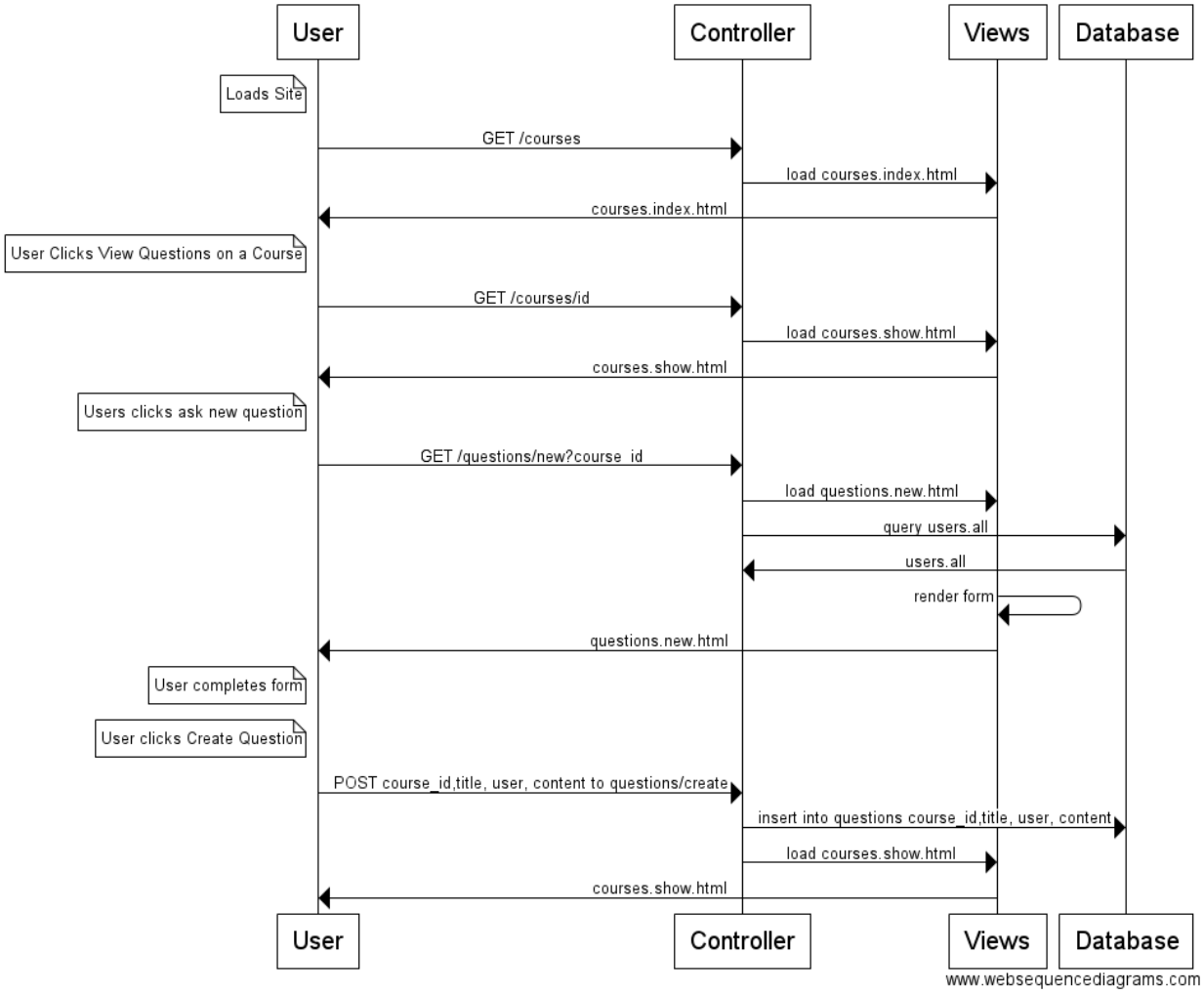


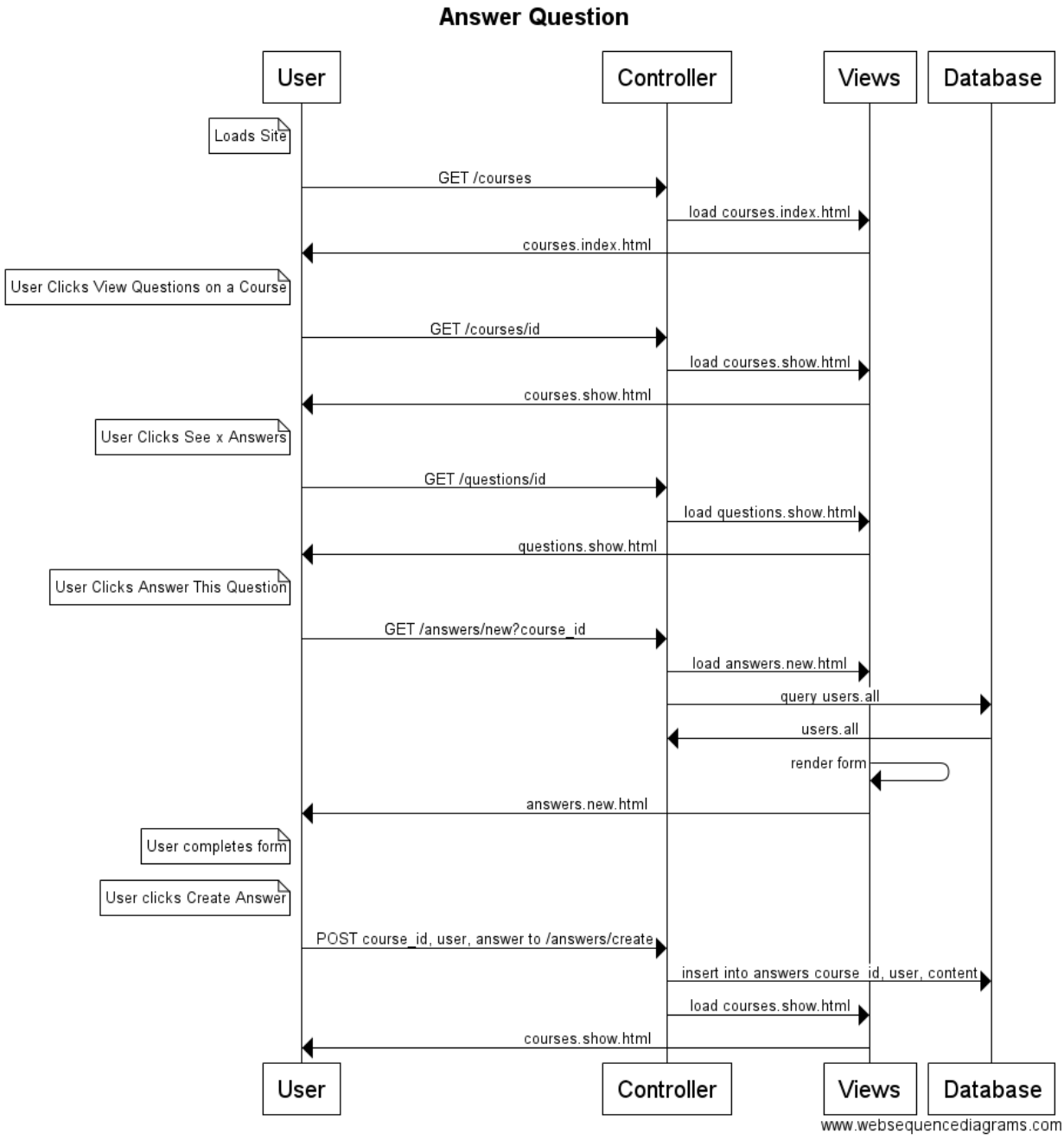

Add User
User
Controller
Views
Database
Load Site
GET /courses
load courses.index.html
courses.index.html
Clicks Login
GET courses/login
load courses.login.html
courses.login.html
User types in username and password and clicks login
POST username, password
username: found
password: matched
load courses.index.html
courses.index.html
User clicks on "Users"
GET courses.users.html
query users.all
users.all
load courses.users.html
courses.users.html
User selects "Add User"
GET courses/add_user
load courses/add_user
courses/add_user
User enters new user's name, email, and course
POST name, email, course
name: not found
email: not found
ADD name, email, course
load courses.users.html
courses.users.html
User
Controller
Views
Database
www.websequencediagrams.com

Edit User

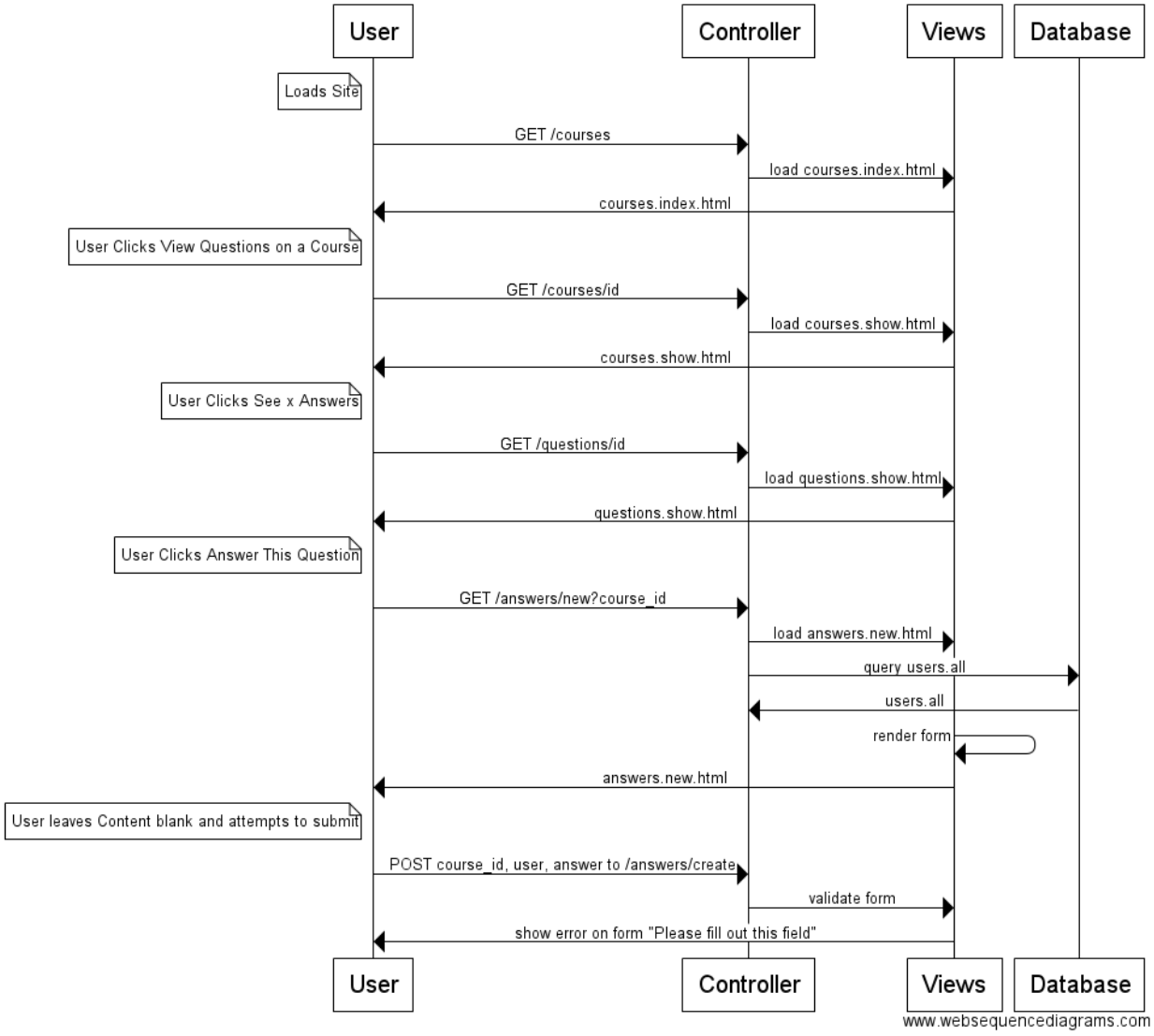

User
Controller
Views
Database
Load Site
GET /courses
load courses.index.html
courses.index.html
Clicks Login
GET courses/login
load courses.login.html
courses.login.html
User types in username and password and clicks login
POST username, password
username: found
password: matched
load courses.index.html
courses.index.html
User clicks on "Users"
GET courses.users.html
query users.all
users.all
load courses.users.html
courses.users.html
User selects user to edit
GET courses.users.html?user
query user.properties
user.properties
load courses.users.html?user
courses.users.html?user
Update user properties
POST name, email, courses
update user(name, email, courses)
load courses.users.html
courses.users.html
User
Controller
Views
Database
www.websequencediagrams.com

Post Question

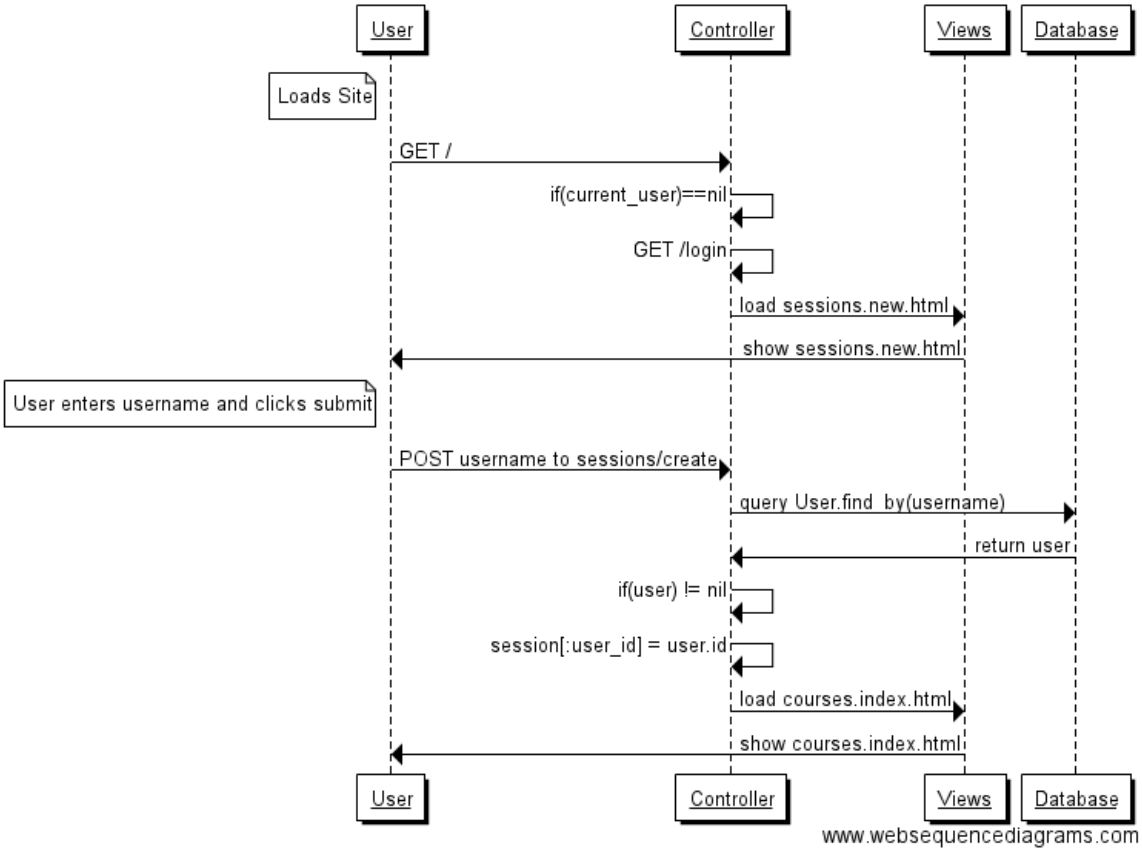



Answer Question
User
Controller
Views
Database
Loads Site
GET /courses
load courses.index.html
courses.index.html
User Clicks View Questions on a Course
GET /courses/id
load courses.show.html
courses.show.html
User Clicks See x Answers
GET /questions/id
load questions.show.html
questions.show.html
User Clicks Answer This Question
GET /answers/new?course_id
load answers.new.html
query users.all
users.all
render form
answers.new.html
User completes form
User clicks Create Answer
POST course_id, user, answer to /answers/create
insert into answers course_id, user, content
load courses.show.html
courses.show.html
User
Controller
Views
Database
www.websequencediagrams.com

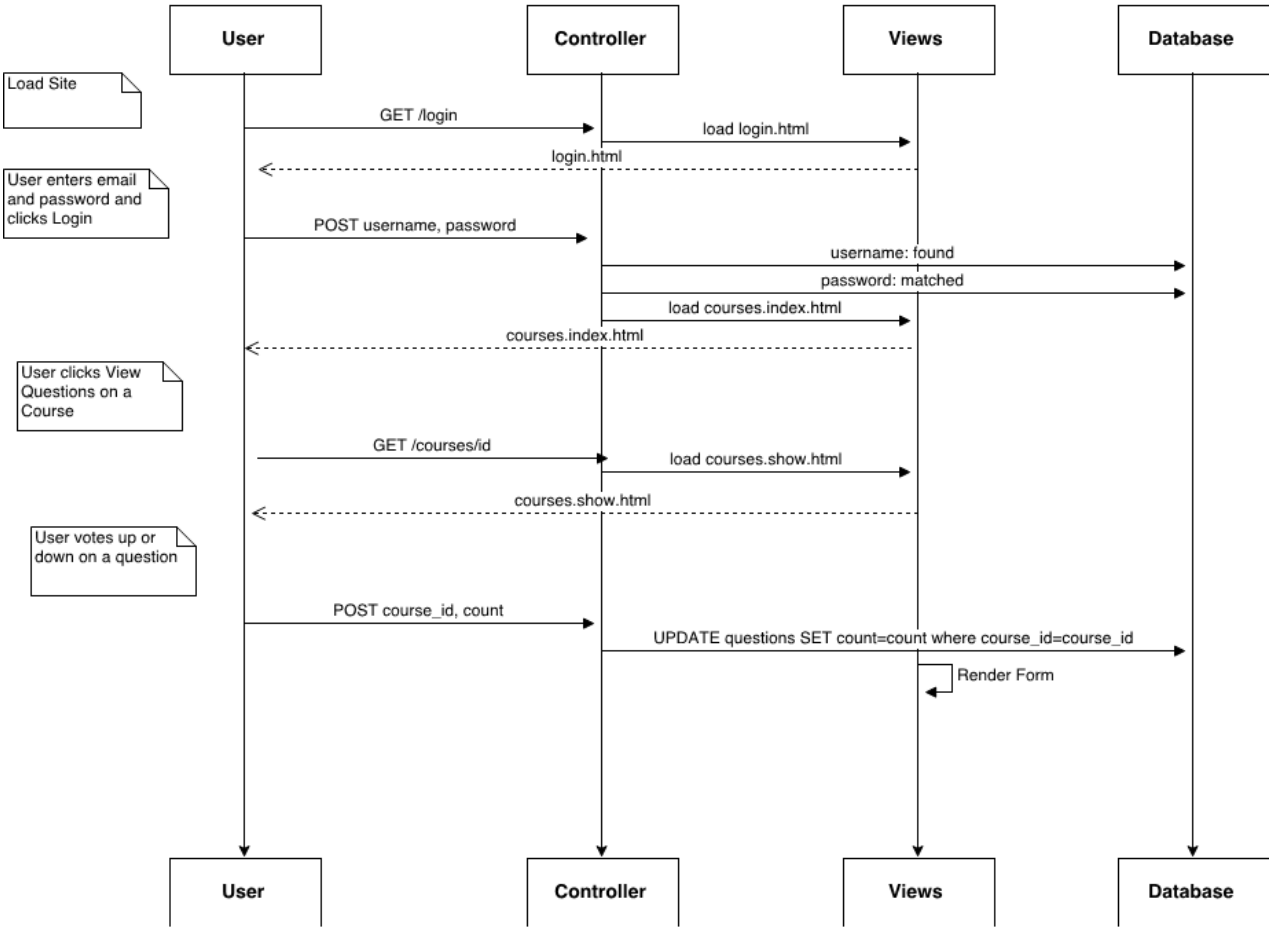
Answer Question(Alternate Flow 4b)



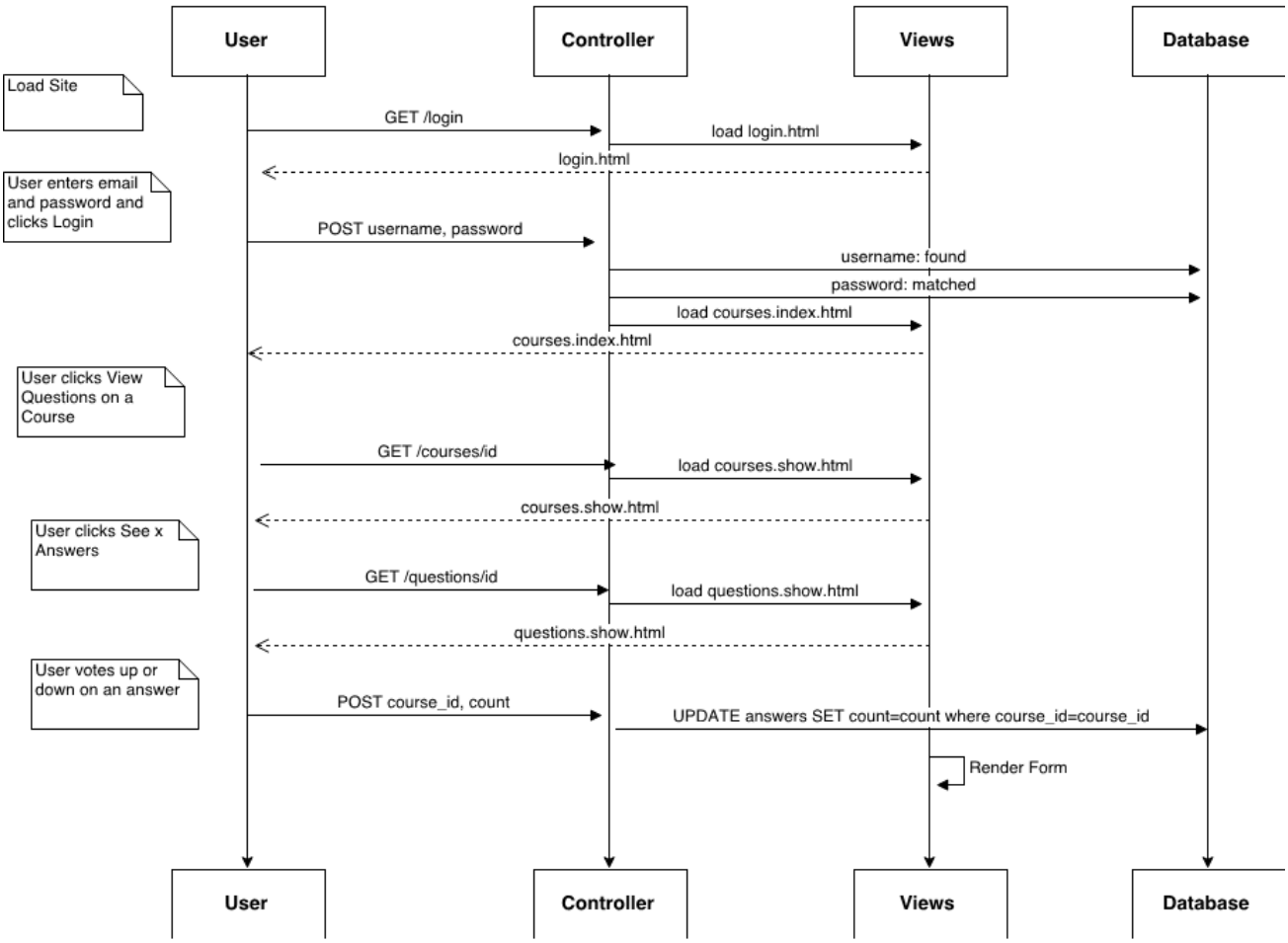
Authenticate User

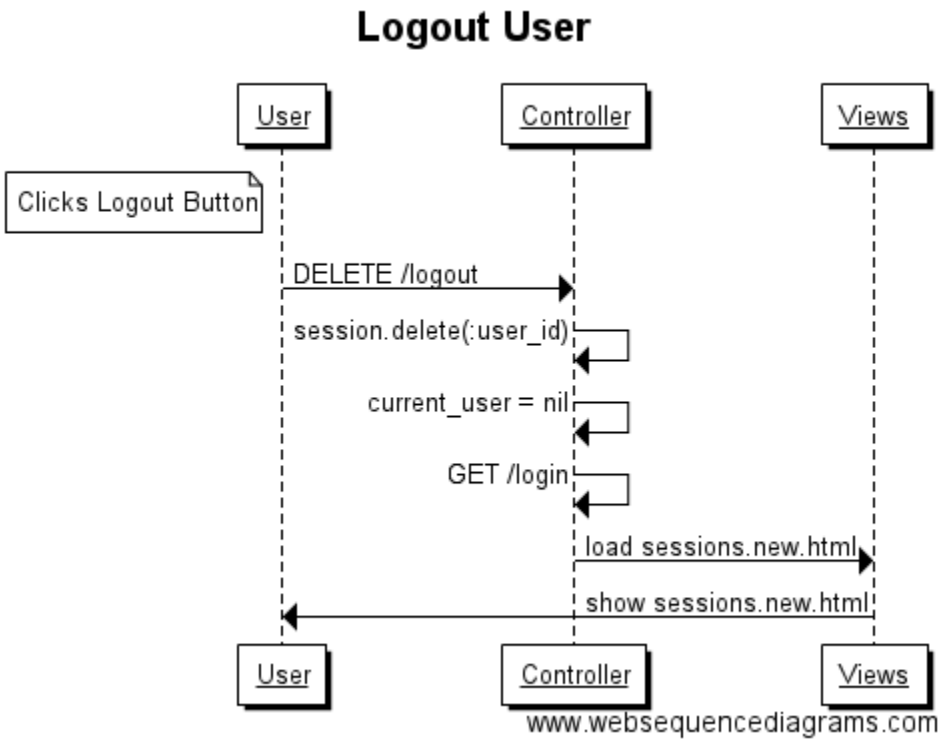


Vote on Question

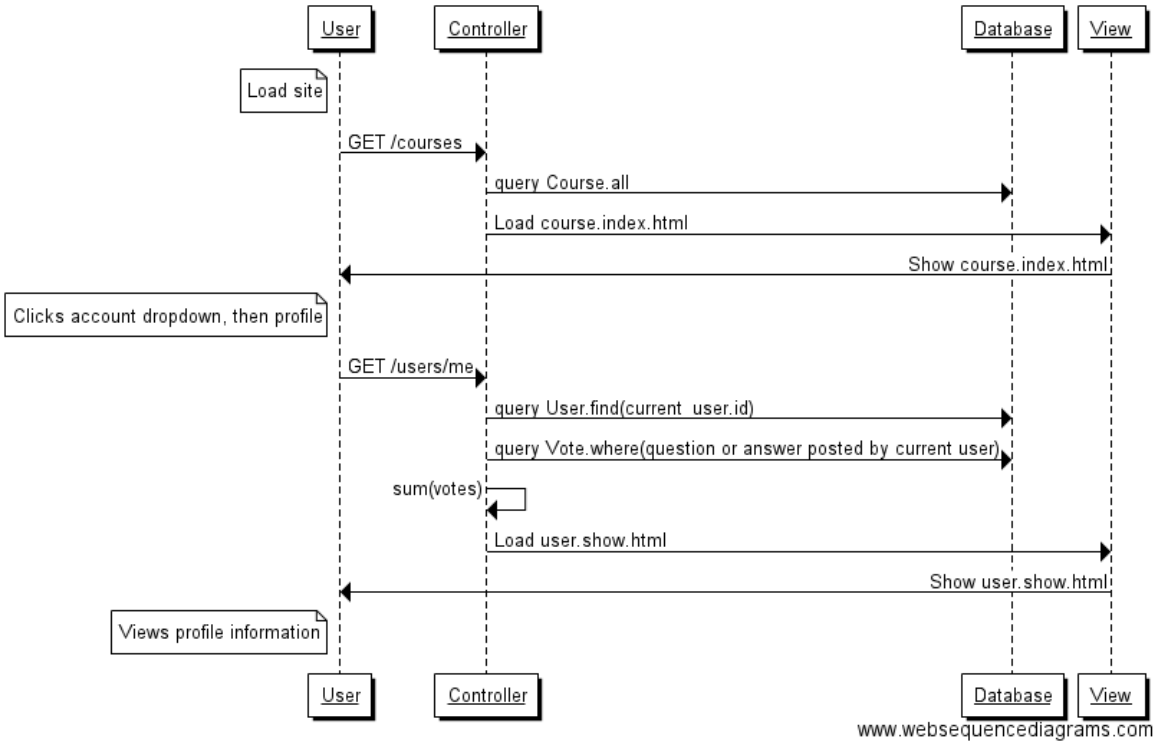


Vote on Answer

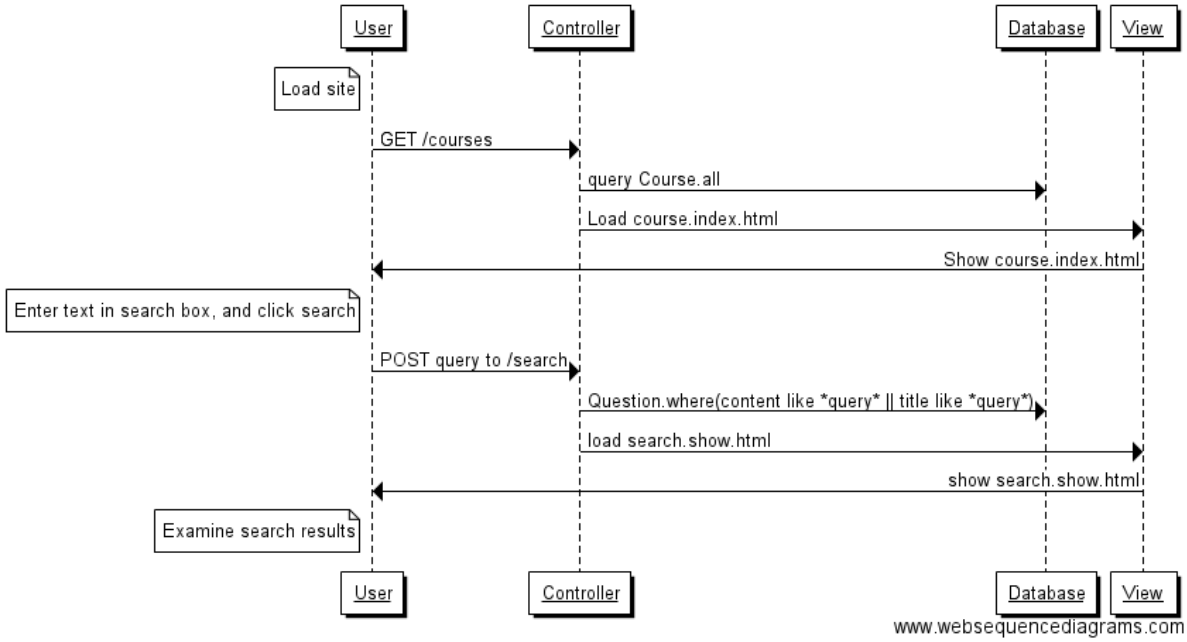


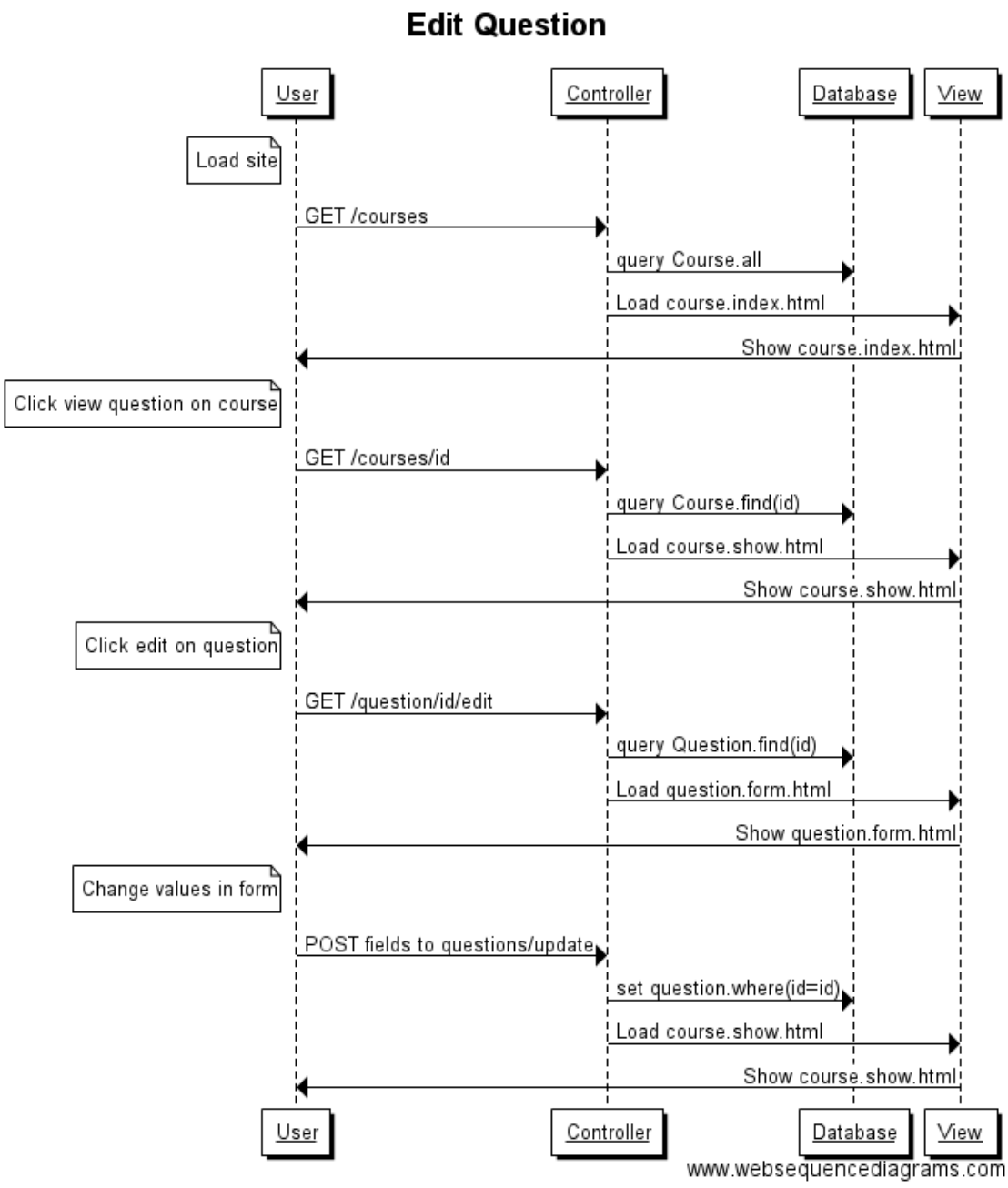

Logout User
User
Controller
Views
Clicks Logout Button
DELETE /logout
session.delete(:user_id)
current_user = nil
GET /login
load sessions.new.html
show sessions.new.html
User
Controller
Views
www.websequencediagrams.com

View Profile

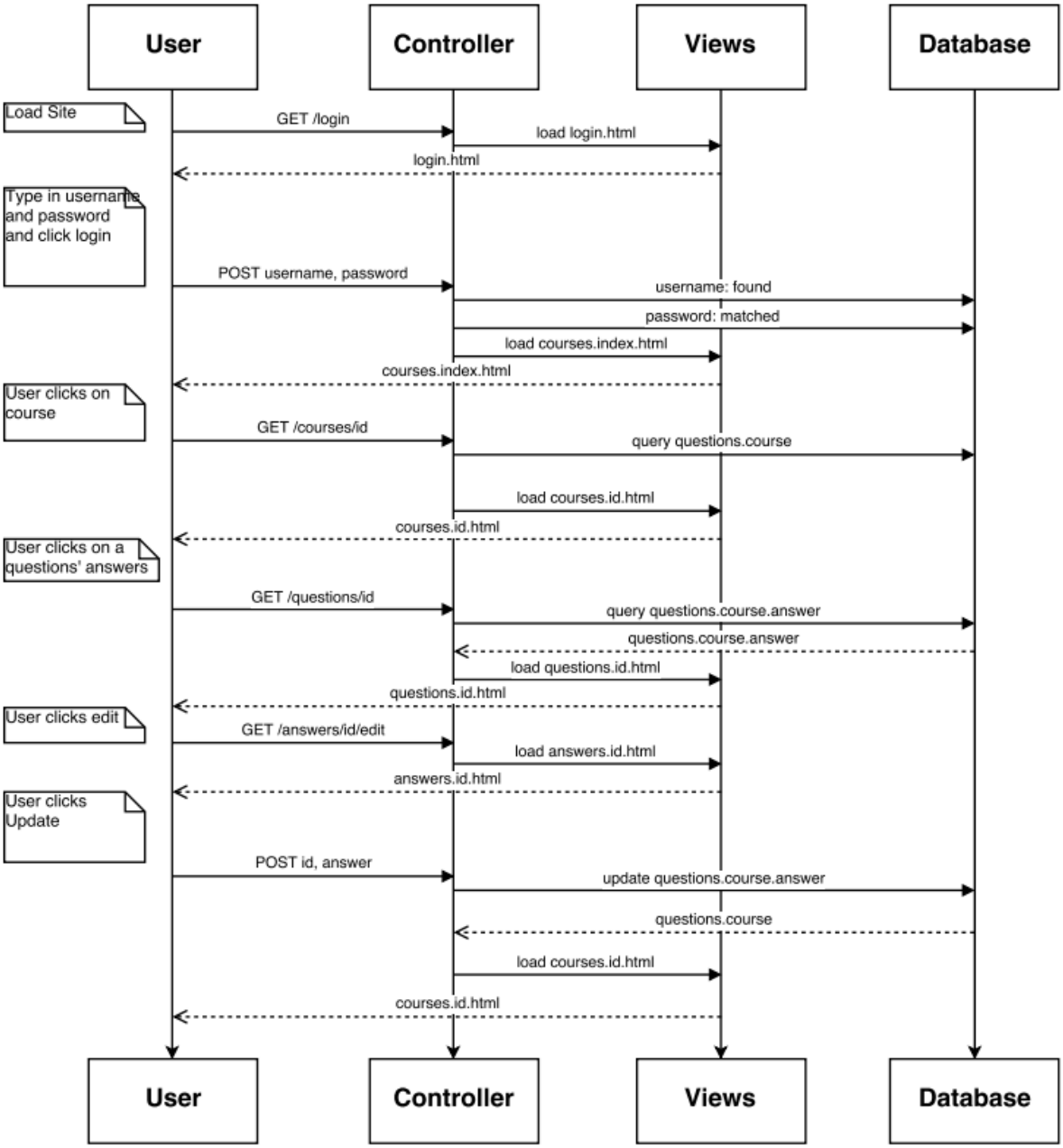


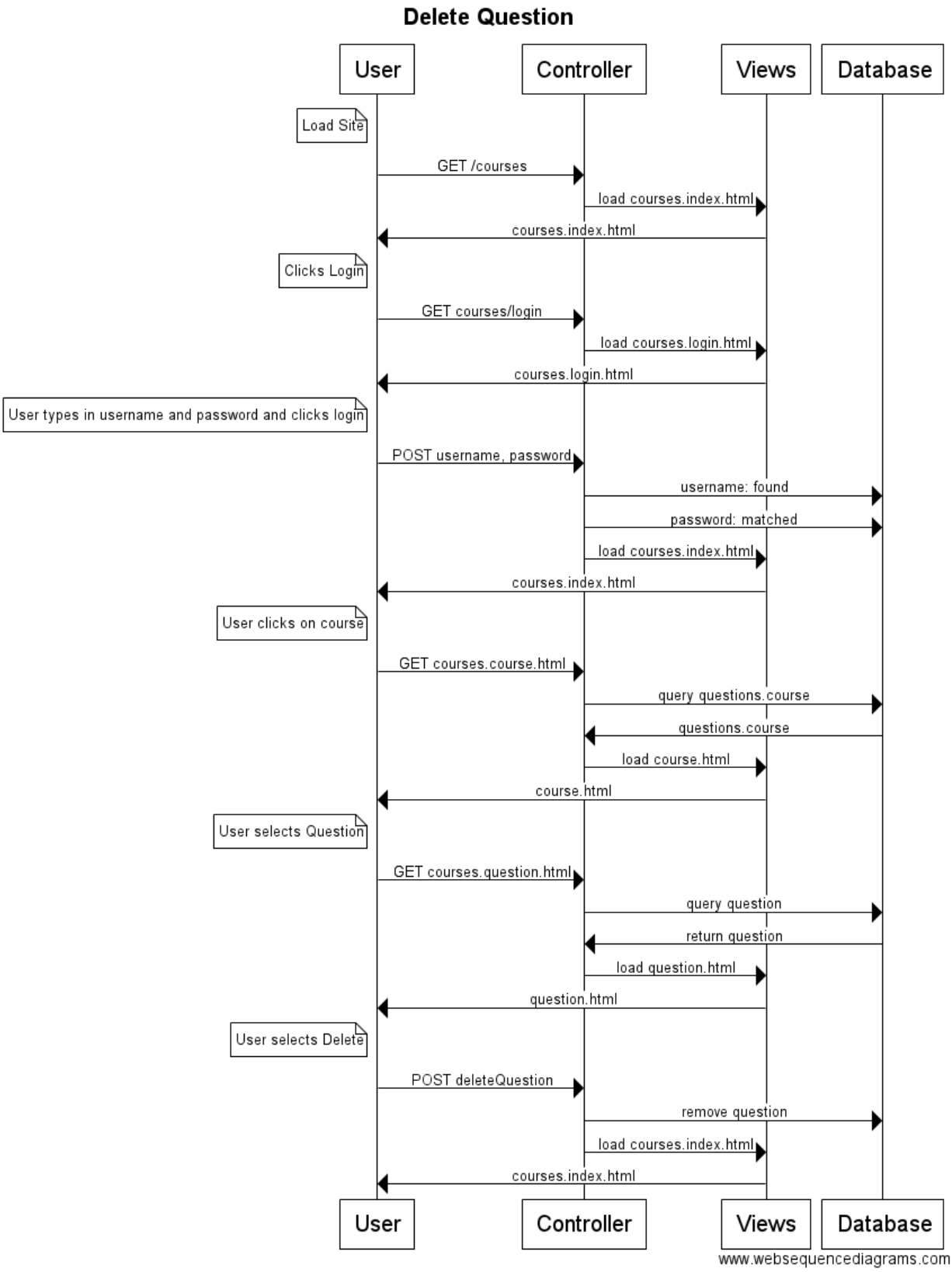
Search for Question




Edit Question
User
Controller
Database
View
Load site
GET /courses
query Course.all
Load course.index.html
Show course.index.html
Click view question on course
GET /courses/id
query Course.find(id)
Load course.show.html
Show course.show.html
Click edit on question
GET /question/id/edit
query Question.find(id)
Load question.form.html
Show question.form.html
Change values in form
POST fields to questions/update
set question.where(id=id)
Load course.show.html
Show course.show.html
User
Controller
Database
View
www.websequencediagrams.com

Edit Answer


User
Controller
Views
Database
Load Site
GET /login
load login.html
login.html
Type in username and password and click login
POST username, password
username: found
password: matched
load courses.index.html
courses.index.html
User clicks on course
GET /courses/id
query questions.course
load courses.id.html
courses.id.html
User clicks on a questions' answers
GET /questions/id
query questions.course.answer
questions.course.answer
load questions.id.html
questions.id.html
User clicks edit
GET /answers/id/edit
load answers.id.html
answers.id.html
User clicks Update
POST id, answer
update questions.course.answer
questions.course
load courses.id.html
courses.id.html
User
Controller
Views
Database


Delete Question
User
Controller
Views
Database
Load Site
GET /courses
load courses.index.html
courses.index.html
Clicks Login
GET courses/login
load courses.login.html
courses.login.html
User types in username and password and clicks login
POST username, password
username: found
password: matched
load courses.index.html
courses.index.html
User clicks on course
GET courses.course.html
query questions.course
questions.course
load course.html
course.html
User selects Question
GET courses.question.html
query question
return question
load question.html
question.html
User selects Delete
POST deleteQuestion
remove question
load courses.index.html
courses.index.html
User
Controller
Views
Database
www.websequencediagrams.com

Delete Answer