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.

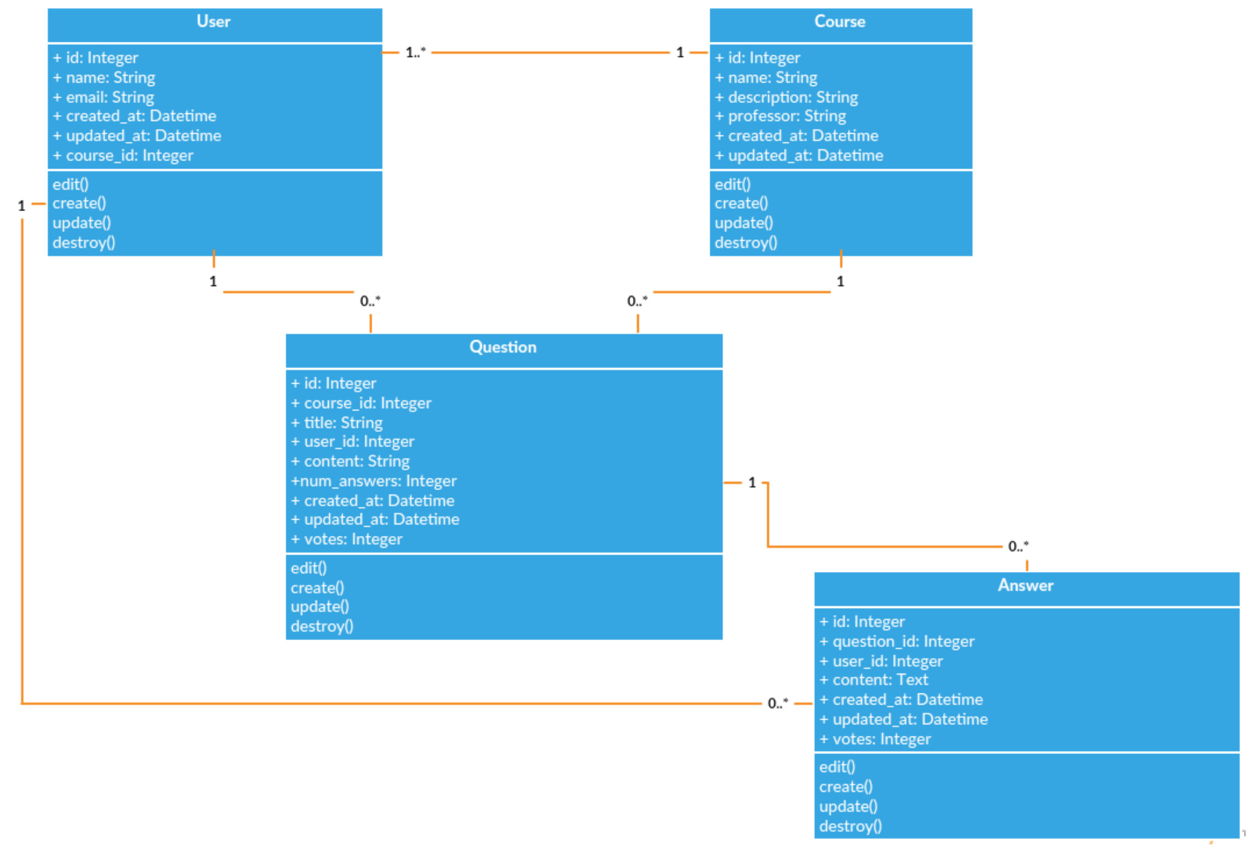
Iteration 2 Database Design Decisions:

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.

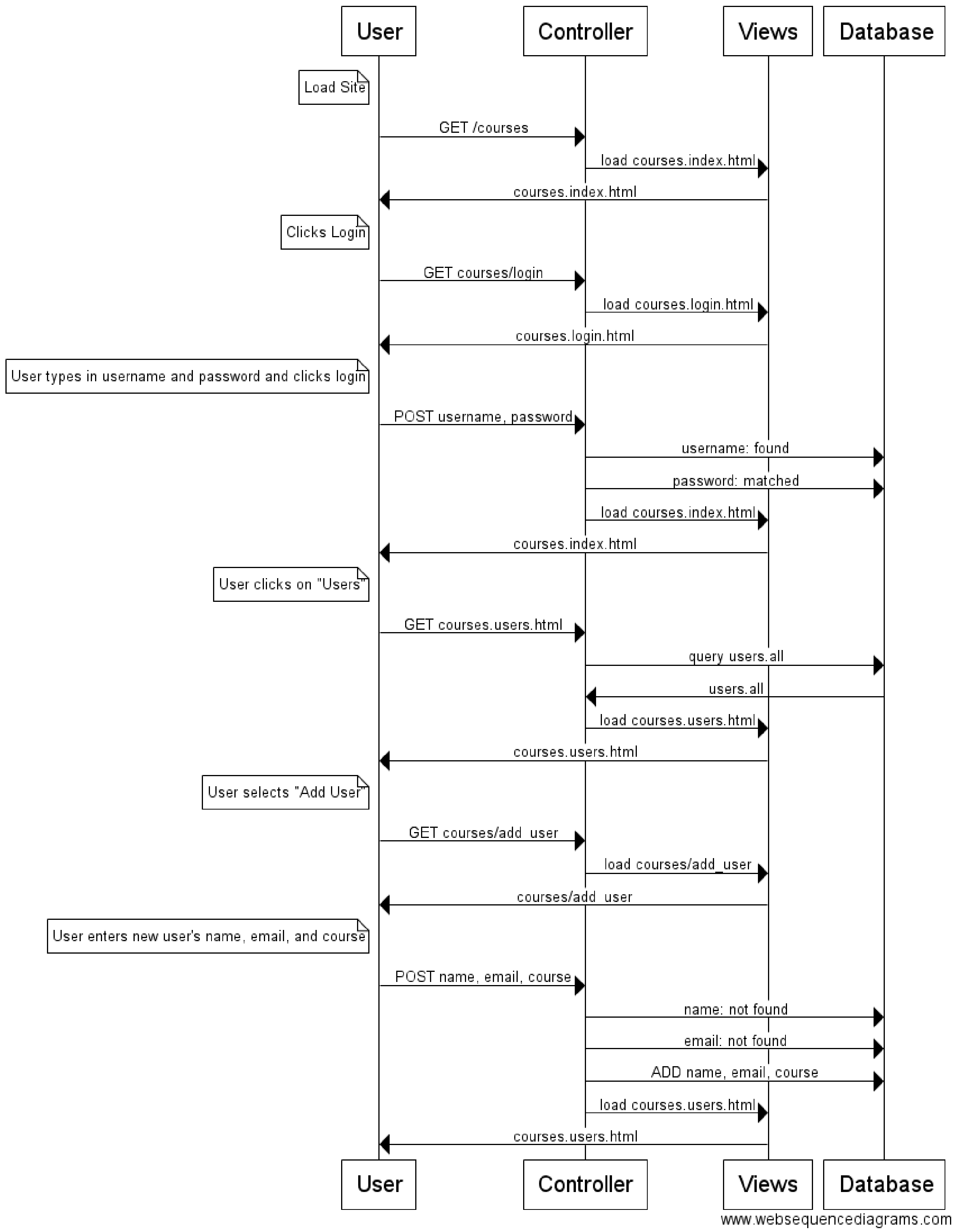
Iteration 2 GRASP Patterns Identified:

We have identified several GRASP patterns that were used in this iteration, and the previous iteration. 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.

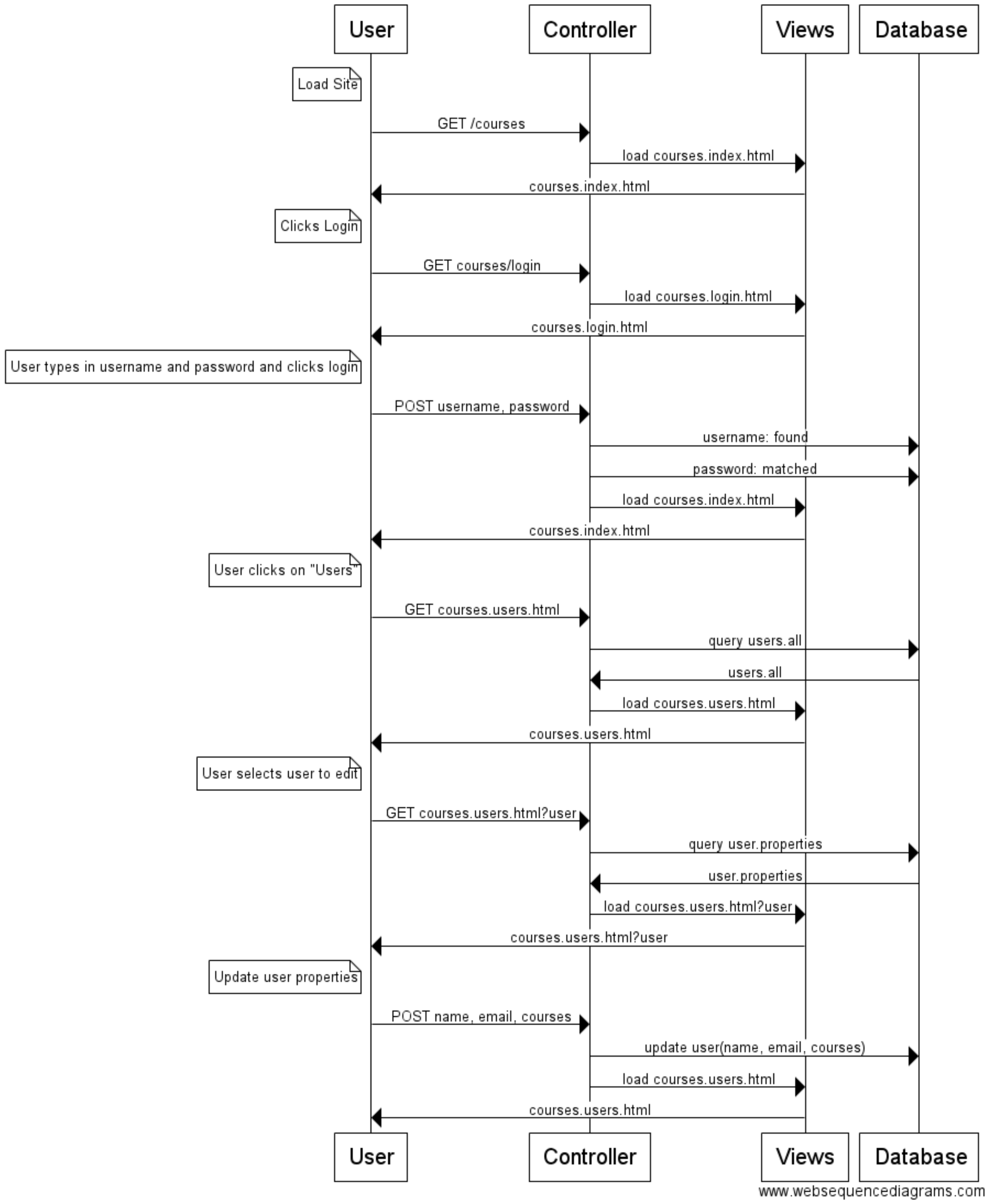
Design Class Diagram



Add User

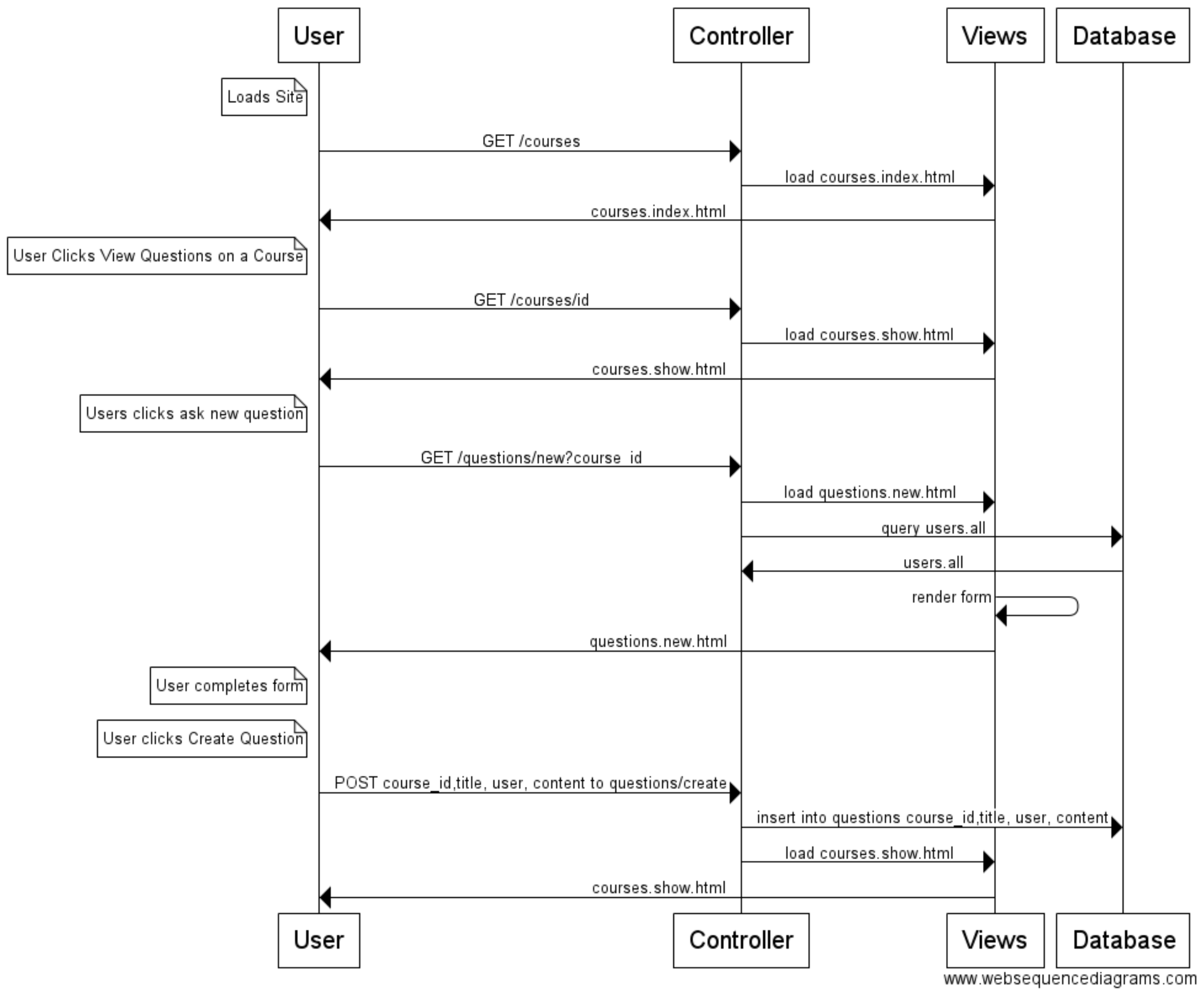

User
Controller
Views
Database
Load Site
GET /courses
load courses.index.html
courses.index.html
Clicks Login
GET courses/login
load courses.login.html
courses.login.html
User types in username and password and clicks login
POST username, password
username: found
password: matched
load courses.index.html
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

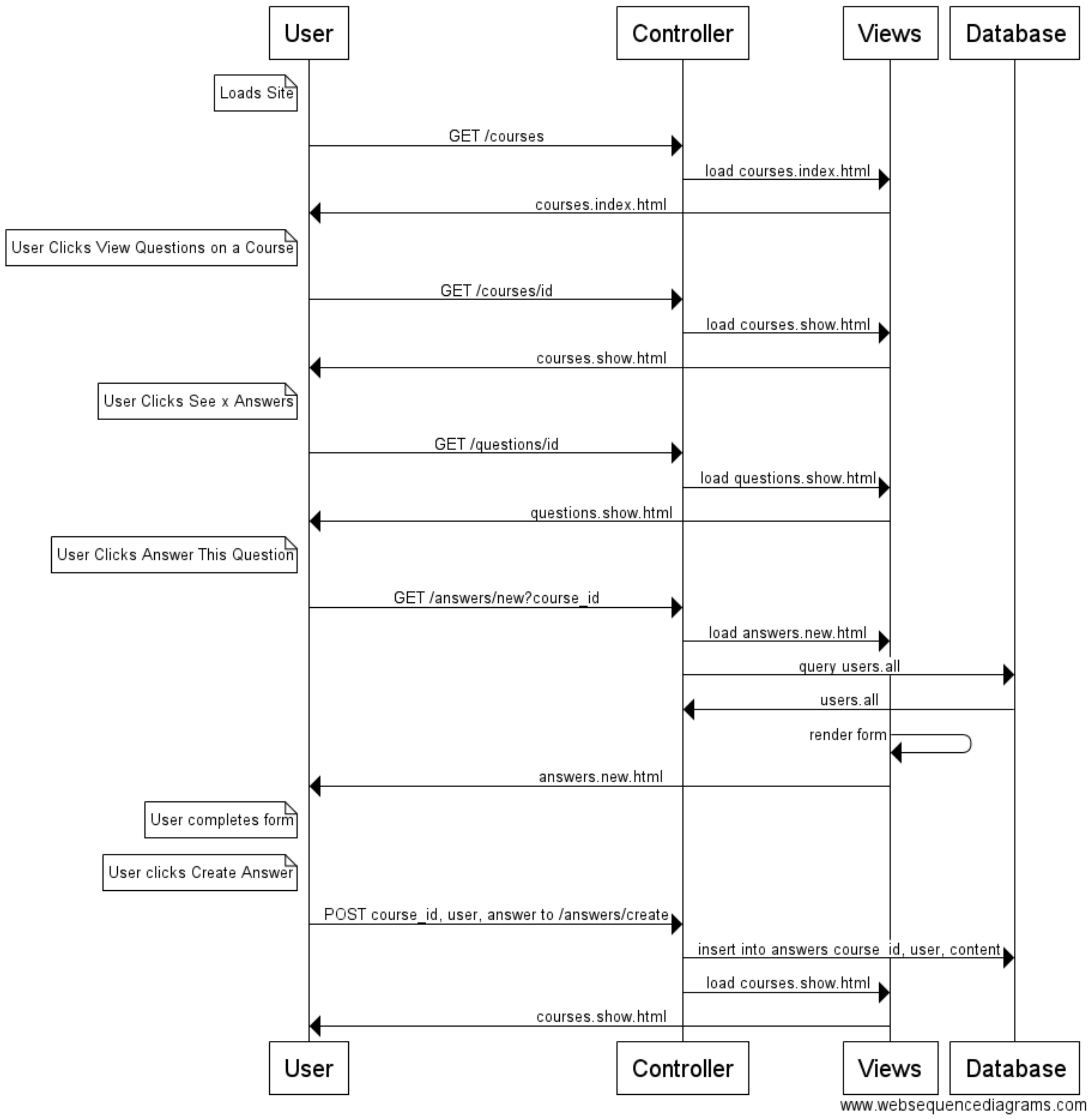


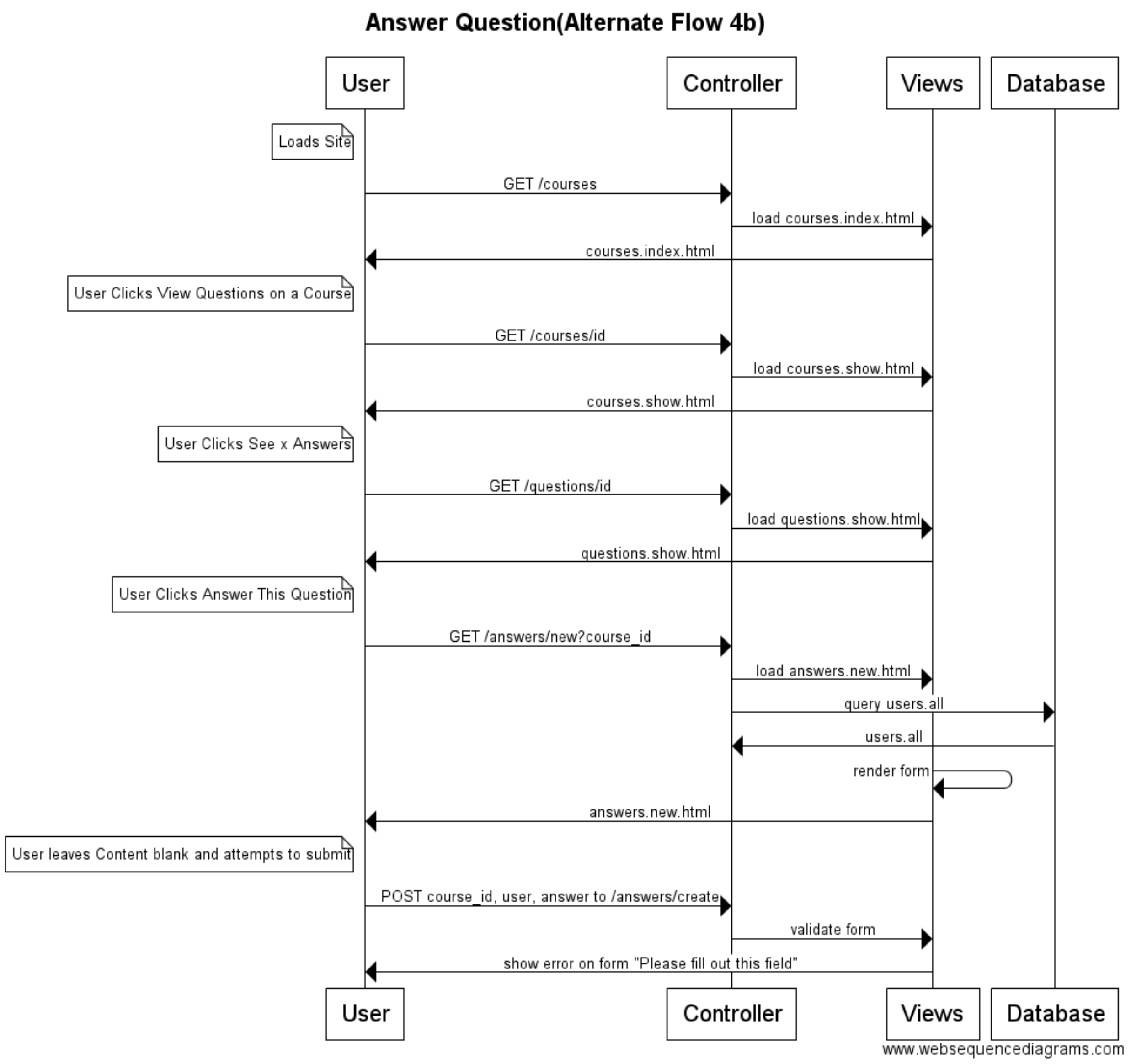
User
Controller
Views
Database
Load Site
GET /courses
load courses.index.html
courses.index.html
Clicks Login
GET courses/login
load courses.login.html
courses.login.html
User types in username and password and clicks login
POST username, password
username: found
password: matched
load courses.index.html
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

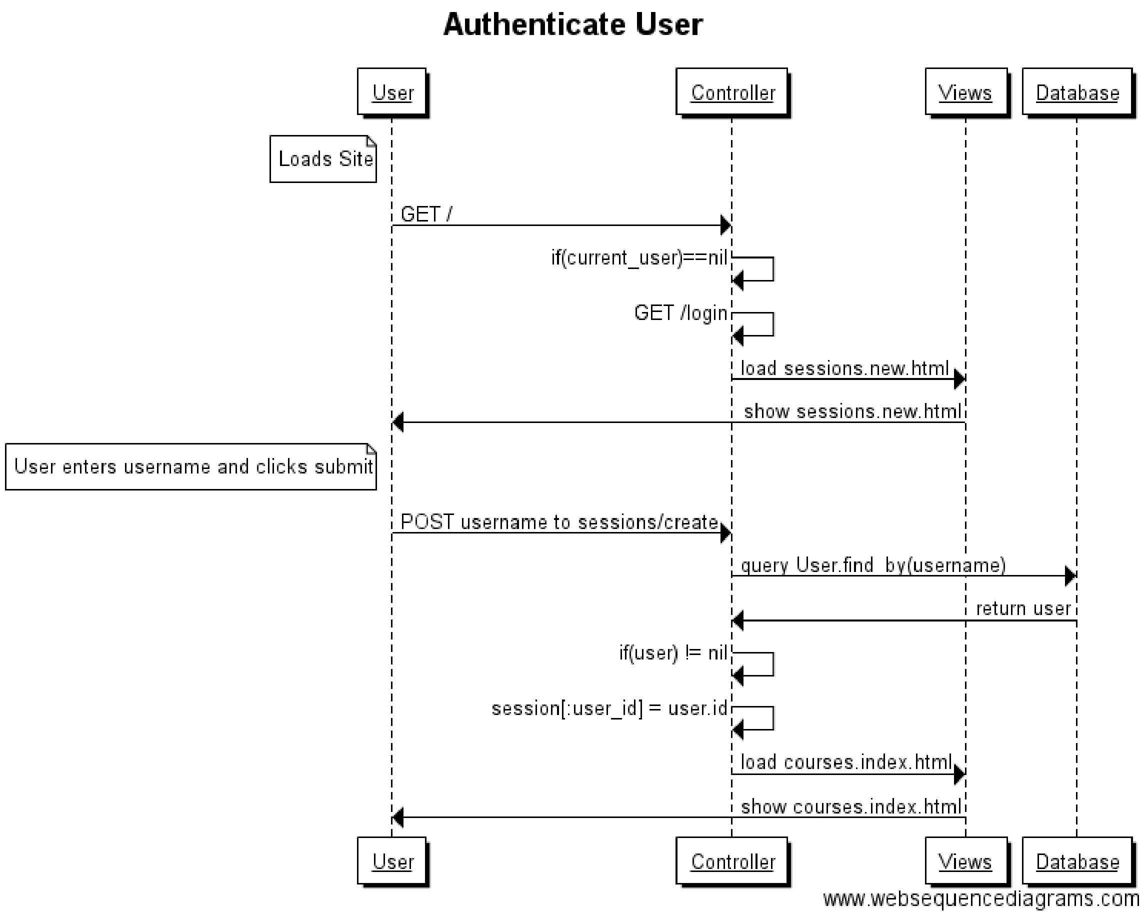
Post Question



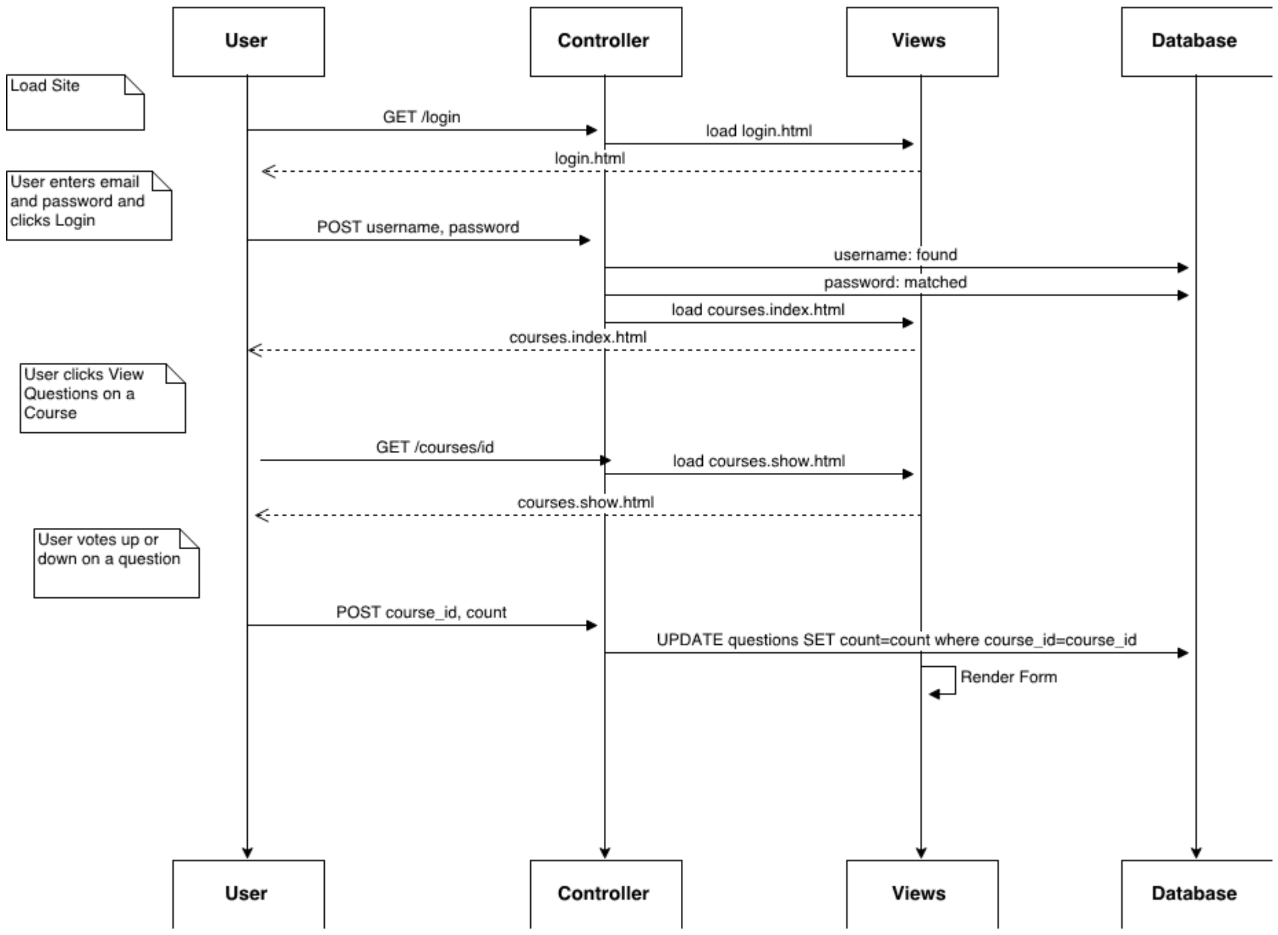
Answer Question



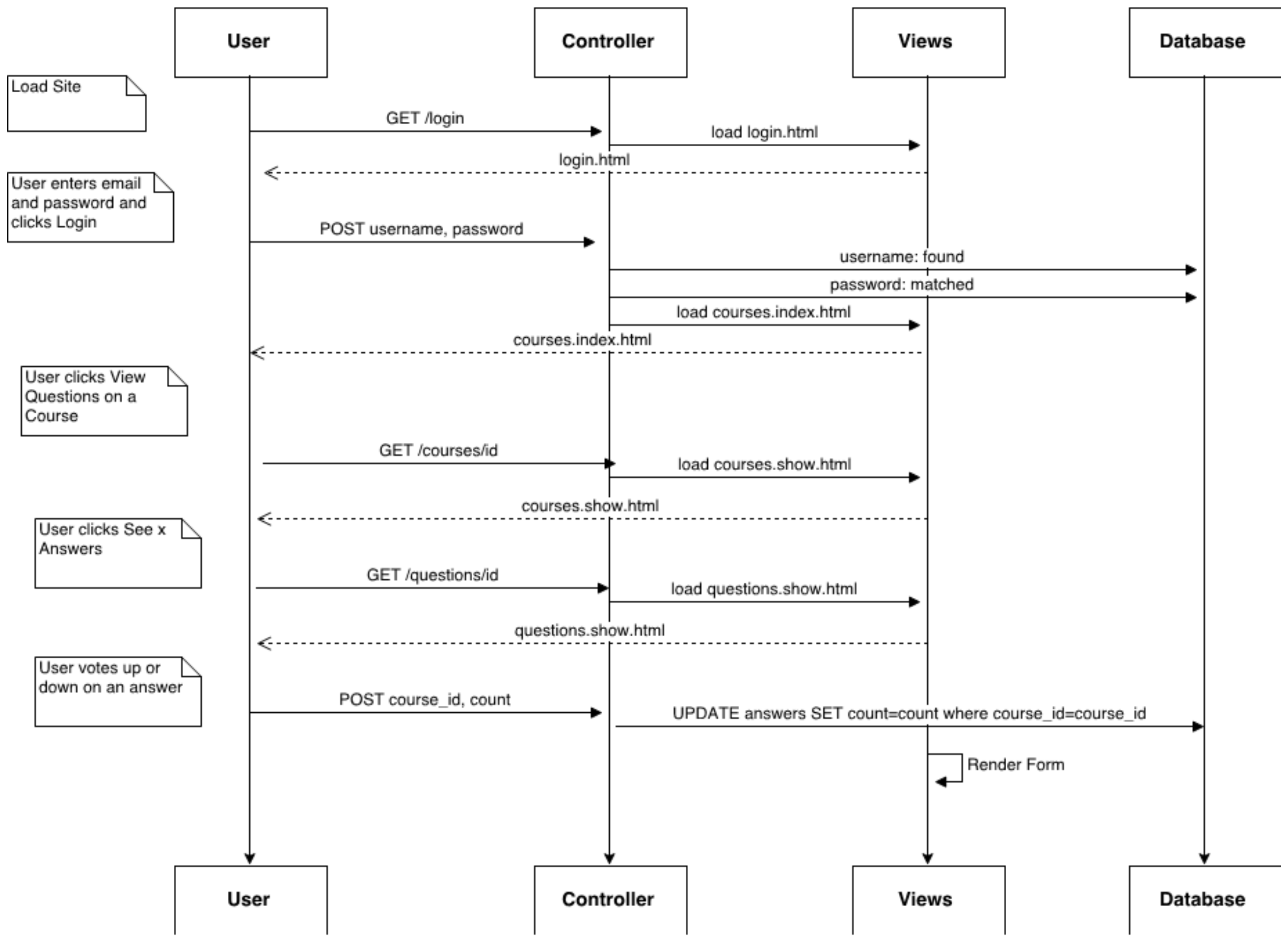

Answer Question(Alternate Flow 4b)
User
Controller
Views
Database
Loads Site
GET /courses
load courses.index.html
courses.index.html
User Clicks View Questions on a Course
GET /courses/id
load courses.show.html
courses.show.html
User Clicks See x Answers
GET /questions/id
load questions.show.html
questions.show.html
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 leaves Content blank and attempts to submit
POST course_id, user, answer to /answers/create
validate form
show error on form "Please fill out this field"
User
Controller
Views
Database
www.websequencediagrams.com

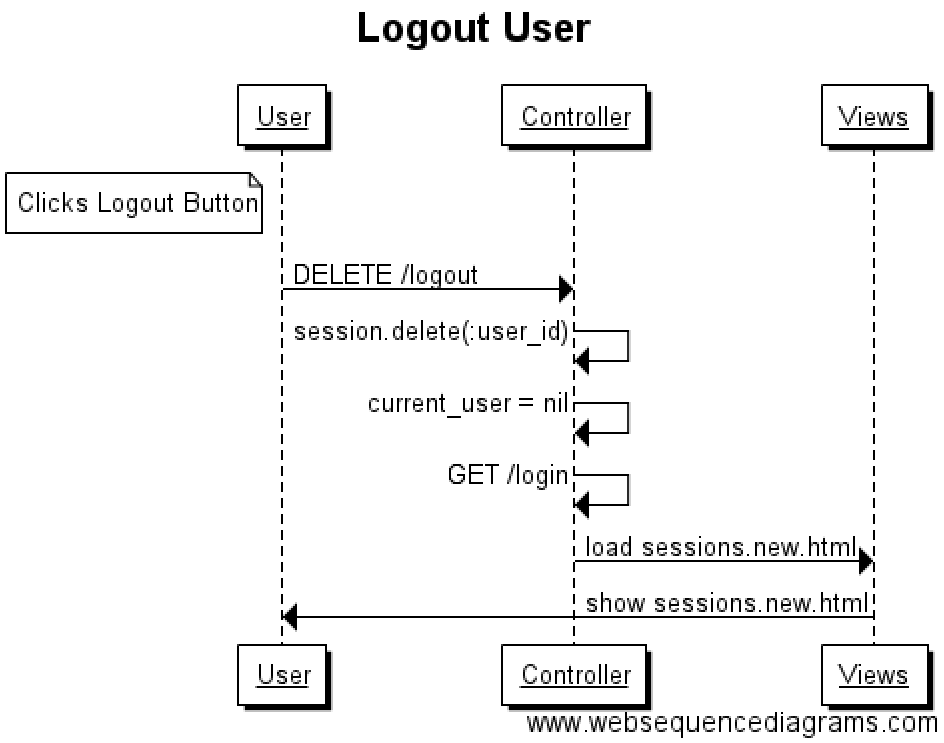

Authenticate User
User
Controller
Views
Database
Loads Site
GET /
if(current_user)==nil
GET /login
load sessions.new.html
show sessions.new.html
User enters username and clicks submit
POST username to sessions/create
query User.find_by(username)
return user
if(user) != nil
session[:user_id] = user.id
load courses.index.html
show courses.index.html
www.websequencediagrams.com

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