EDUvote Design Doc

Annie Tang, Casey O'Brien, Qui Nguyen, Kathryn Siegel

Overview

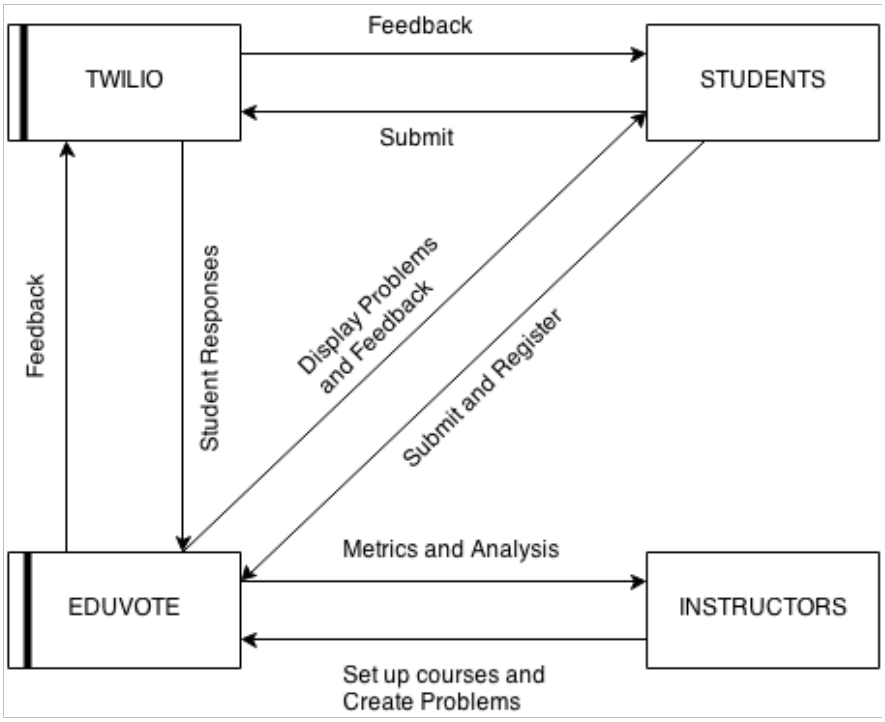
Purpose and Goals *(Annie)*

Description: EDUvote is an app that allows students to submit answers to questions asked in class and enables instructors to track and analyze student responses. Instructors will be able to create multiple choice or numeric questions through our app interface and display them to the class by projecting an app view onto a screen. Then, students can submit responses to those questions through text messages. The app registers these responses by utilizing the Twilio API and has the ability to determine whether they are right or wrong. Feedback is sent using the Twilio API. After responses are collected, instructors can view summaries and analysis of the responses.

Purpose & Goals: To facilitate participation and student/professor interaction within the classroom by creating a voting application with low costs, a simple interface and detailed analytics.

Motivation: Many classes at MIT try to foster interaction and keep track of attendance by posting a short question and having students use Turning Point Clickers to answer them. Clickers work as follows: there is a receiver system controlled by the professor or TA in the classroom. Questions are displayed via projector on a screen, and can be opened to responses or closed by the professor or TA via the receiver system. While a question is open, students submit responses by tuning their clicker to the class channel and clicking the appropriate answer to the displayed multiple choice question. However, clickers are unnecessarily expensive ($40) for both students and instructors, and they give no feedback directly to students after a submission. Additionally, students often cheat by giving other students their clickers. Instructors would also benefit from detailed analysis, which is not readily accessible with current clicker software.

Context Diagram *(Annie)*



Key Concepts *(Qui)*

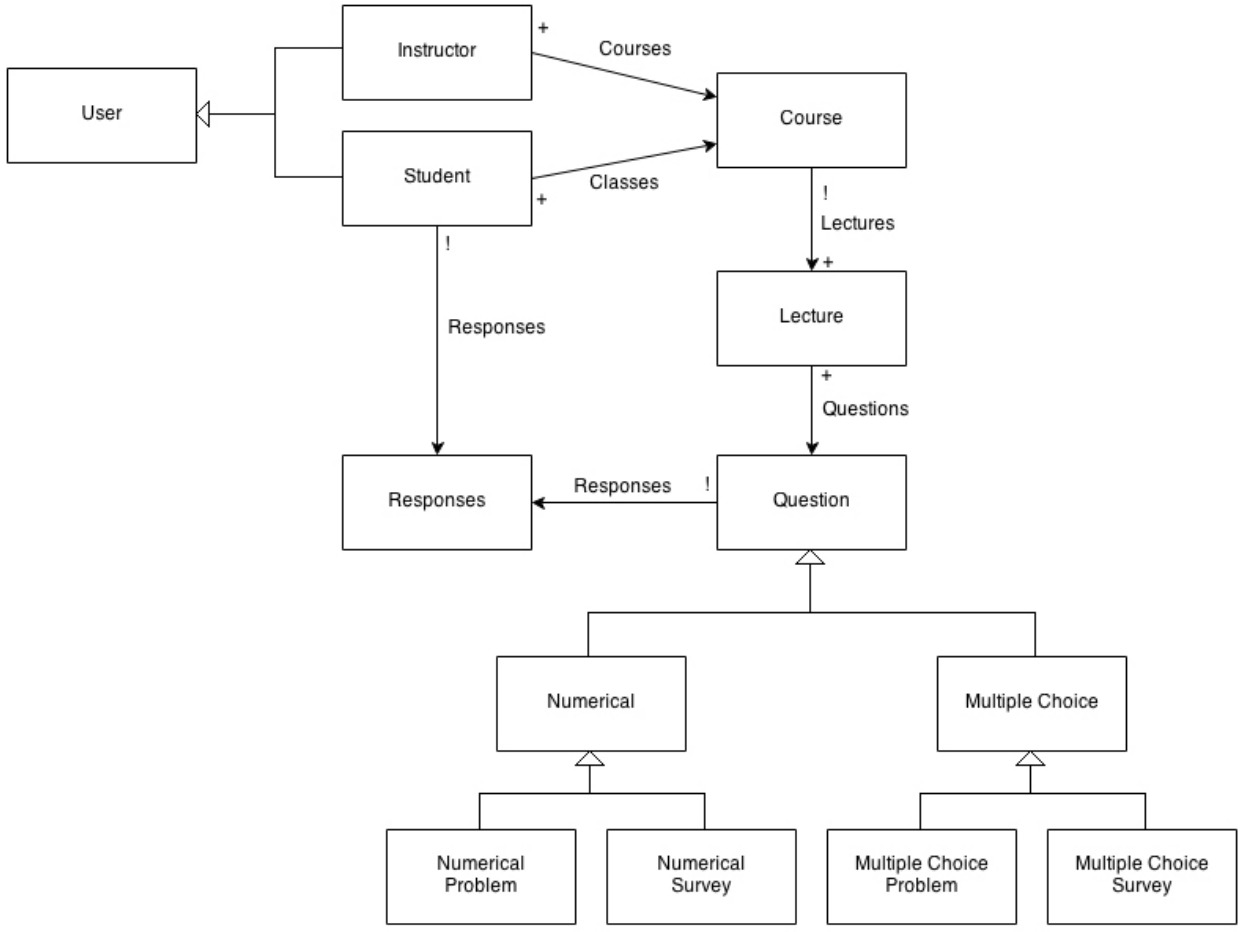
Question: something that a person can give a response to. It can be multiple-choice or free response, and it may or may not have a correct answer..

Response: an answer to a question. It must be either a number or a short string (like A, B, C, etc).

Student: a person who can enroll in courses. A student will be required to respond to the questions for each course he/she is enrolled in.

Instructor: a person who teaches course(s). For each course an instructor teaches, they can create questions.

Data Model *(Casey)*



Clarifying Notes:

- A *Course* is an overall class, something like 8.01
- A *Lecture* is a particular class session, something like Lecture 4 on 11/7
- A *Question* can be *Numerical* (single number solution) or *Multiple Choice*. It can also be a *Problem* (has a correct answer) or a *Survey* (no correct answer, just gathering information)
- Each student will have at most one response *per Question*.

Feature Descriptions *(Qui)*

Simple creation and display of questions: On the EDUvote website, instructors will be able to create questions and organize them by lecture. A question can either be multiple choice or allow numerical responses, and LaTeX will be supported for mathematical notation and figures. Once an instructor is ready to ask a question, he/she can also show the question in display view directly from the website.

Receive responses from students and send feedback via text message: To respond to a question asked during class, students can send text messages to a predetermined phone number. EDUvote will receive all these responses and store them for review by the course's instructor(s), ensuring that students can only respond once per question. EDUvote will also send a textual confirmation to students after it has received their responses.

Receive responses from students and send feedback directly: Students who cannot or do not want to text message can answer questions directly on the EDUvote website. The website will be optimized for mobile access, so the website can easily be accessed from phones as well as from computers.

Analysis of student responses and attendance: Instructors will be able to view a summary of student responses by question and analysis of responses over time. In addition, they will have access to a list of all students in the course; then, for each student, they will be able to see all the responses from that student, as well as that student's attendance records and percentage of questions correct.

Student interface for viewing past questions and performance: After logging into the EDUvote website, a student can view the past questions for courses he/she is registered for, as well as attendance records and percentage of questions correct. For each question, he/she will also be able to see the correct answer and his/her own answer.

Security concerns *(Katie)*

Security Requirements:

- Students cannot access the professor-side interface, and cannot create/edit/destroy problems that the professor has created.
- Students cannot view problems that the professor has not yet released to the class.
- Must securely store phone numbers so that only professors and the app can send notifications to students.

Potential Risks:

- Student submits a number that is not his/her own number and helps a friend cheat for the whole semester.
- Hacker gains another user's access token, and utilizes this to send fraudulent answers, thus sabotaging another's grade.
- User sends so many messages that the Twilio queue gets clogged.

Threat Model:

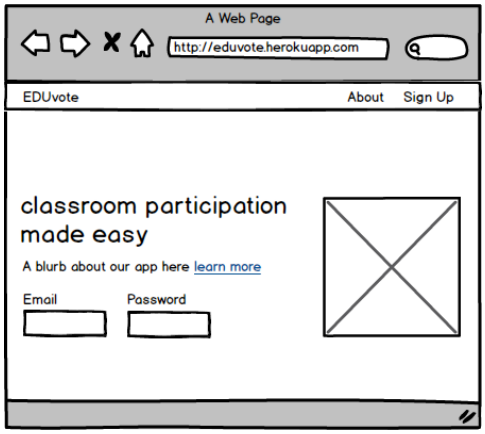
- Minimal information is stored for each user; thus, this app would not be a target for criminals, etc.
- Verifications are sent via text message to the user upon correct submission, so it would

be hard to spoof a sender without the student being notified.
- Only one answer may be registered per open question, thus preventing the Twilio queue from being backed up.

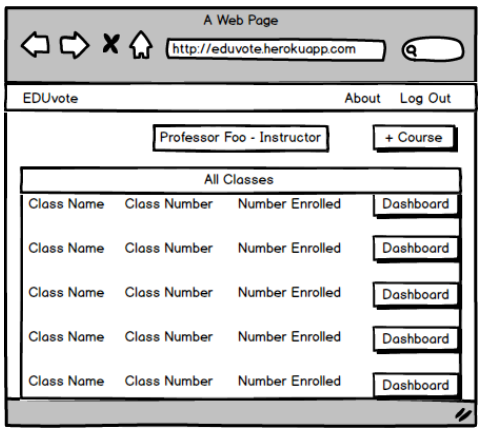
Mitigations:
- We will prevent SQL injection by using Rails' built-in escaping of SQL commands. So, we will not run SQL commands using Strings; instead, we will use commands in Ruby.
- Cross-site forgery (XSS, CSRF)
    - We will prevent cross-site scripting by only letting certain parameters take effect. In other words, we will only "listen" to whitelisted values, rather than not responding to blacklisted values.
    - We will prevent cross-site request forgery attacks by using Rails' built-in protect_from_forgery method. This attaches a security token to the appropriate requests and checks for that token.
- Only the users' names, emails, and phone numbers will be stored. This information is not particularly sensitive.
- User access control to prevent viewing of students' information by everyone except the professor of a class for which the student is registered.
- We will rely on Twilio to send text messages and to receive messages
- Registration requires verification via text message, double checking that the phone number registered is correct.
- Implement session timeouts to avoid insecurities involved with copying cookies or leaving a computer open.
- We will use SSL to encrypt cookies, so hackers would need to decrypt a cookie to get a user's access token.
- We will use rails' protect_from_forgery method to further encrypt information being sent by the app.
- The Twilio API requires an authentication token to be passed in each request, increasing the security of the app with regards to sending and receiving messages.
- We will send email verifications for sign-up (using the Devise gem), preventing users from submitting fraudulent email addresses or from signing up for their friends without their friends being aware that they are doing so.
- We will screen for .edu email addresses, preventing non-members of the class from signing up. Professors can also have the option of removing students from the class, allowing them to filter for illegitimate students.

User Interface *(Annie, Katie)*

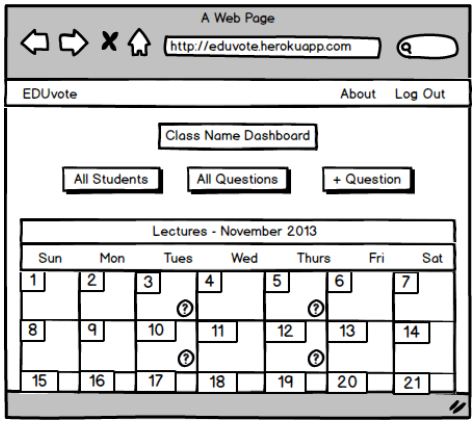
Wireframes



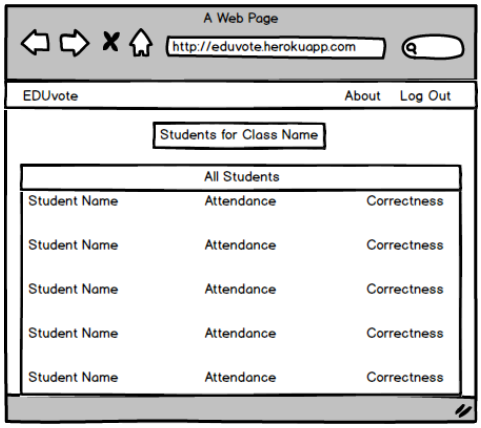
Login/Signup Page (both)



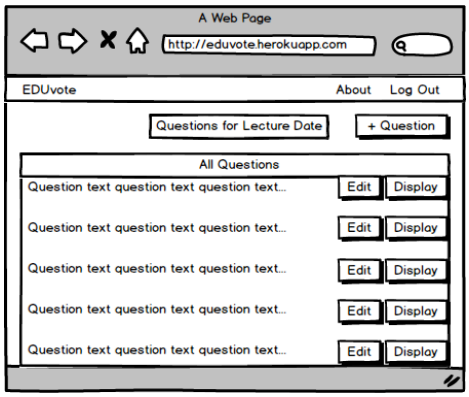
Classes Index Page (both)



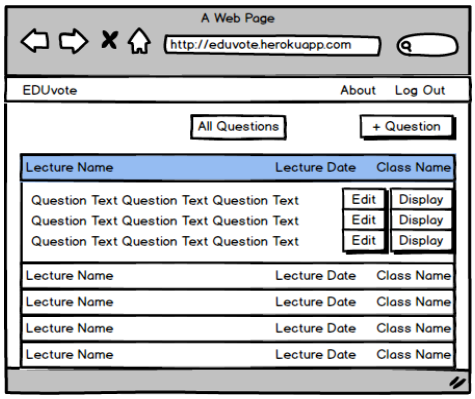
Course Dashboard (both - student view w/o '+Question' button)



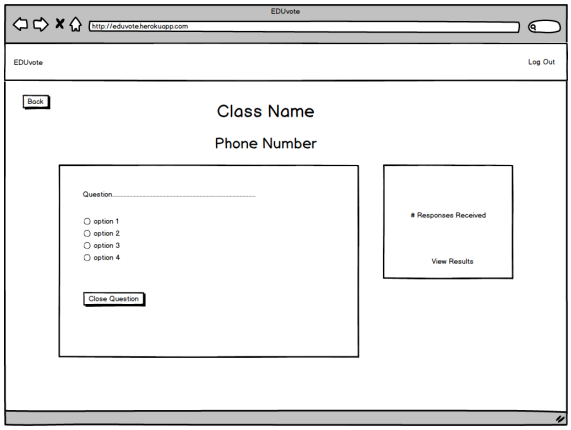
Students Index Page (Instructors)



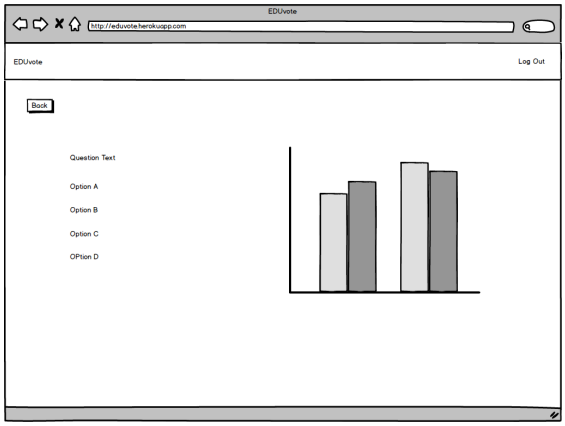
Questions for Lecture (both - student view w/o '+Question' button)



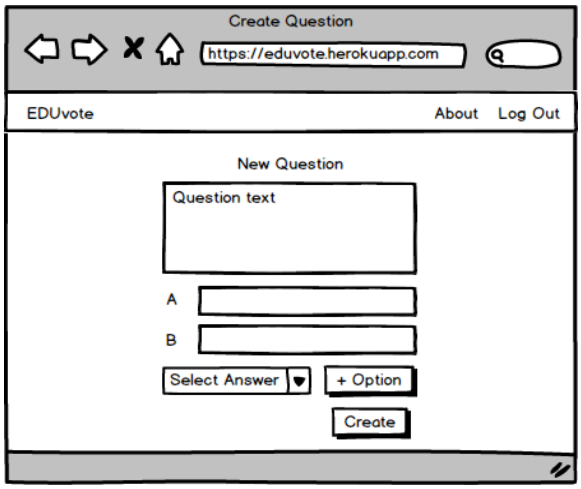
All Questions Page (both - student view w/o 'Edit' and '+Question' buttons)



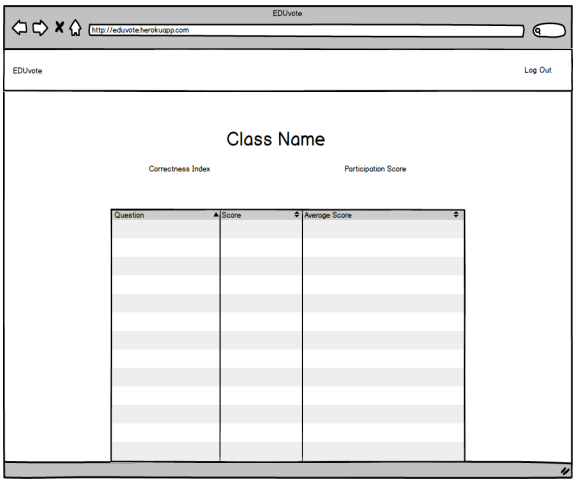
View Question (Instructor)



View Metrics (Instructor)

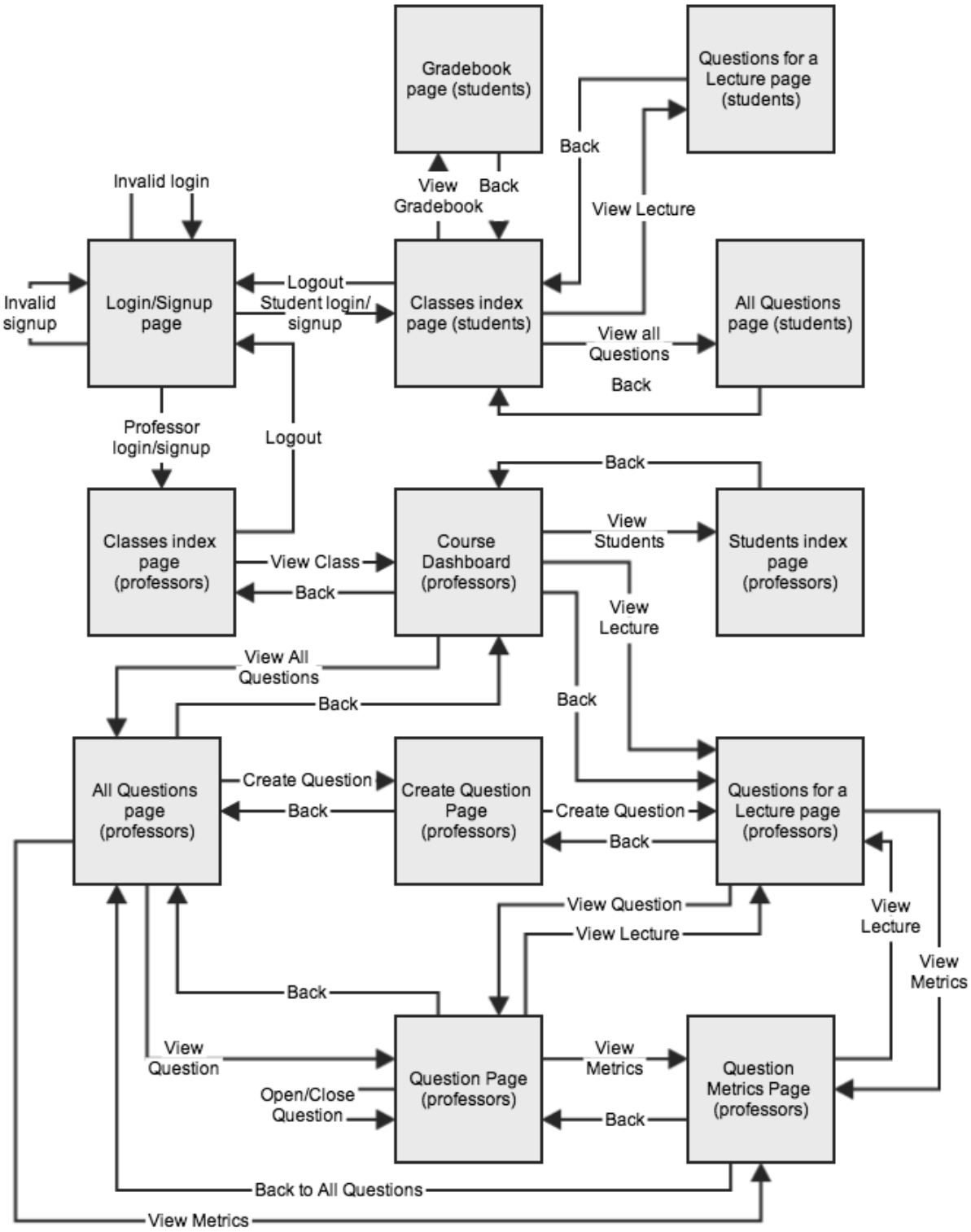


Create Question (Instructor)



Gradebook Page (Student)

State Diagram



Design Challenges *(Katie, Annie)*

1. Problem: Possible lack of cell service in the classroom
    - Considerations:
        - Students could use Wifi
        - Online portal for students would communicate with Twilio via Wifi
    - Solution:
        - We will create an online voting platform that students can use if there is no service in the classroom

2. Problem: Some students don't have phones or have a limited data plan.
    - Option 1: Have students submit answers through an online system
        - Pros: all students have access to computers, especially in TEAL classrooms. Also, this allows for the interface to be more complex and give more feedback.
        - Cons: laptops are more distracting and the interface is more complicated to use. Also, repeatedly taking out and putting back one's laptop in the middle of class disrupts class unnecessarily.
    - Option 2: Have students submit answers through their phones
        - Pros: most students have phones, and this is a simple system that has no embellishments (simply text messaging). This solution can be easily implemented via the Twilio API.
        - Cons: Phones also present distractions, and professors may have issues getting students to put away their phones in between questions. Also, not all students have the ability to text or possess phones, so choosing this option would limit the scope of the app to just people with smartphones and data plans.
    - Decision: We will implement both options, but favor developing the phone-centric version of the app for the MVP. We reasoned that most people possess phones, which should allow us to adequately demonstrate our app. In the final product, we will create an online interface for voting so that students without phones or texting plans will not be excluded.

3. Problem: Calculating trends over time (i.e. how how a class is improving on answering the correct answers) from data.
    - Considerations:
        - Performance might be an issue if we are looking up an entire database table each time the view is called
        - Could make use of middle-tier database caching to achieve high scalability and performance
            - Since we are looking at simple database tables that only store a few values, caching might be too much effort than its worth
        - We could create sophisticated methods in our models that can easily be called upon in our controllers to display different kinds of analytics
    - Solution:
        - We will first implement without database caching. Because our tables are

relatively simple, performance should not be an issue. However, we will decide after the MVP is implemented whether or not we should do some sort of database caching.

4. Problem: Twilio can only handle 1 text per second for each phone number
    - Considerations:
        - Issue of processing speed for collecting submissions
        - Possible lagging between getting the answers and sending feedback
        - Could have multiple numbers for a single classroom, and divide up who texts to which number
    - Solution:
        - Depending on how many students are registered for a class, we will generate a number of different phone numbers that all belong to a single class. Groups of students will text into different numbers so that the speed that Twilio processes texts will not be too much of an issue

5. How to mark attendance for a course
    - Option 1: Attendance is computed by lecture (students are 'present' if they answer all questions for a given lecture)
        - Pros: This will do a better job preventing students from cheating, as a student who is not in class is unlikely to be able to answer all the questions within their time frames
        - Cons: Students who only attend part of a lecture for some reason will recieve no credit for the lecture.
    - Option 2: Attendance is computed by question (a student's overall attendance rate is the percentage of total questions a student has answered)
        - Pros: Allows students to receive partial credit for a class
        - Cons: May make it easier for students to cheat, as it is likely that some students will be able to answer some questions from a given lecture even if they are not present. If a student misses a lecture which has an usually high number of questions, their grade would suffer unproportionally.
    - Decision:
        - Attendance will be calculated by lecture. This is a better measure of the overall attendance of the class and promotes lecture attendance best.

<u>Notes on Code Design</u>

1. Subclassing Instructors and Students from User
    - Option 1: Maintain a single User class, use a boolean field to indicate Instructor/Student
        - Pros: Keeps the code simple, keeps us from using duplicate code since Instructor and Student are very similar.
        - Cons: Makes it difficult to maintain a relationship between Students and Courses as well as Instructors and Courses, forces us to maintain the same fields for both

Instructors and Students (for example, we would like phone numbers to be required for Students but not for Instructors).

- Option 2: Maintain separate models and tables for Student and Instructor (do not subclass User)
    - Pros: Allows us to only validate the fields that we really need (so we won't need to valide phone numbers on Instructors), makes it simple to maintain the relationship between Students/Instructors and Courses.
    - Cons: Results in two tables which are very similar, will end up requiring a lot of duplicate code.
- Option 3: Create Student and Instructor models which inherit from a User model, maintain both in a single table
    - Pros: Allows us to have different code for Students/Instructors where needed, but to use the same code where they would be duplicated.
    - Cons: The model is slightly more complicated than other options.
- Decision: Create Student and Instructor models which inherit from a User model, maintain both in a single table. This is the most space-efficient method which still allows us to create all the necessary relationships.

2. How to store the options for a question
    - Option 1: store options in columns in the questions table
        - Pros: Simple, and easy to specify where the correct answer is stored.
        - Cons: number of possible options is limited by the number of option columns, but having too many option columns wastes space in the database.
    - Option 2: store options in a separate table, with an association between questions and options.
        - Pros: each question only uses space for as many options as it requires
        - Cons: more complex implementation. In addition, now each option must have a letter identifying it (with options in columns, the column number could be used), so that the correct option can still be identified and checked from the question.
    - Decision:
        - Options will be stored in a separate table. This is much more space-efficient, especially since we are allowing free response questions without preset options.

3. How to allocate phone numbers for classes
    - Option 1: Use the Twilio API to purchase a new phone number every time a new class is created.
        - Pros: Each class is guaranteed to have a unique phone number. There also could potentially be an infinite number of classes.
        - Cons: This is extremely expensive of the developer, who must pay every time a new class is created. Additionally, if a class is deleted, this phone number will not be used and the money spent by the developer in purchasing the number via the Twilio API will be wasted.
    - Option 2: Have a static set of phone numbers and pull the first unused phone number

every time a new class is created.
    - Pros: This is inexpensive for the developer, since only a set number of phone numbers need to be purchased. Also, this allows for conservation of phone numbers, since when classes are deleted, other classes can then use these phone numbers
    - Cons: There is the potential that more phone numbers will be needed than will exist in the static pool of phone numbers (however, the developer can easily check how many numbers are in use and increase the pool of these phone numbers if necessary).
- Decision: we chose option 2, because for the purposes of this project, we would like to keep monetary costs to a minimum. If we were to commercialize this product, we would likely combine options 1 and 2 so that numbers are re-used, but they are also dynamically generated via the Twilio API.

4. How much information to collect during signup? (Katie)
    - Options considered: We debated which information would be necessary to uniquely identify different individuals during signup, and also provide instructors with enough information to be able to contact individuals.
    - Password: This is clearly necessary, as it allows users to log in.
    - Email: We use this to determine whether a user is a student; student emails are unique and can be used to verify the legitimacy of a student's identity.
    - Name: This would be a quick and easy way to identify a certain student given a list of students, as instructors would ordinarily identify students by name rather than by email address.
    - Phone number
        - Choice 1: Collect the phone number of all individuals associated with the class, regardless of whether they are a student or an instructor.
        - Choice 2: Collect only the phone numbers of the students in a class, since the instructor's phone number is not needed.
        - Decision: choice 2. We would not like to store unnecessary information in our databases.

5. How to display a question in the list of questions for a course
    - Option 1: Display full question text
        - Pros: This allows the instructor to easily see the content of a question, and also gives them the ability to Find questions within the page easily.
        - Cons: If there are a lot of questions, this could lead to a very cluttered page, making it more difficult for the instructors to find what they are looking for.
    - Option 2: Display truncated question text
        - Pros: Lessens the clutter on the page, still allows the instructor to get an idea of what the questions are about.
        - Cons: Could cause problems with LaTeX questions (if only part of a LaTeX formula were included in the truncation), would no longer allow the instructor to

easily search through the questions using Find.

- Option 3: Display question numbers
    - Pros: Keeps very little clutter on the page, and doesn't have any problems with rendering LaTeX
    - Cons: Doesn't allow the instructor to have any insight into what the question is about without clicking on it first
- Decision: Display full question text. We will break the list of questions down by lecture in order to make the page look less cluttered.