# Kweri - COS 333 Design Document

Nick Jiang - nj3@princeton.edu
Susan Wang - xw7@princeton.edu
Erica Wu - eswu@princeton.edu
Jonathan Yu - jy11@princeton.edu
Jonathan Zhang - jz6@princeton.edu (Team Leader)

## Overview

Imagine this: You're in a lecture hall and the professor is going over a complicated new topic. You don't understand what's going on, but after looking around it seems everyone else is following along fine. What would you do?

A: Wait it out and see if you can ask your friends for help after class/consult the textbook and try to figure it out yourself.

B: Raise your hand and be "that guy" and ask the professor to repeat himself

We asked 767 students this question and found that 89% chose option A.

Kweri is a real time educational web app that provides a way for students to anonymously ask questions during lecture. Far too often we have found that students are not willing to ask questions during lectures, especially large lectures with upwards of 100 students because they do not want to be singled out as “that kid”.

Kweri allows students to anonymously ask questions that are then pushed to everyone in the lecture hall. Other students can then choose to either upvote or downvote the question. This way, the professor can see what the most pressing questions are at any time. Another feature included in Kweri is the Confusion Counter. Students can choose to set their status to “I’m confused” or “I’m not confused”. The professor will be able to see at any given point what percentage of the class is following along.

If we have the time, Kweri would also contain a third interface, the TA interface. Here, TA’s of the class can choose which questions the professor sees or answer some questions directly within the app so that the professor is not overwhelmed with numerous questions.

Our webapp will use the Meteor framework and will be hosted on Heroku.

## Requirements and Target Audiences

Since Kweri is a web app, our target audience would be lectures that allow students to have their laptops or phones out. Although in our examples we mention large lecture halls, Kweri is suitable for any sized classroom in which a professor wants to gauge some information from students.

Technology for anonymous real time question polling exists already, but the current softwares are locked behind pricey package subscriptions and custom software that needs to be installed on a computer. We plan on differentiating ourselves with ease of use. No payments (yet), no installation, no hassle. Just log onto our website and you are ready to go!

## Use Cases

### Scenario A:

Physics professor P is introducing a complicated new topic in his lecture. Since Professor P knows that these new concepts are extremely important and are the foundation for the material in the coming weeks, he wants to make sure the class is following along. He pauses and asks the class, “Any questions?” Nobody speaks up, so he thinks everyone understands and prepares to move on. However, when he glances at the Kweri page open on his laptop, he sees that more than half the class upvoted a question asking him to clarify a key equation. Thanks to Kweri, Professor P realized that most of his students did not, in fact, understand some of the concepts he had presented, and he was able to clarify his points and bring everyone back on the same page.

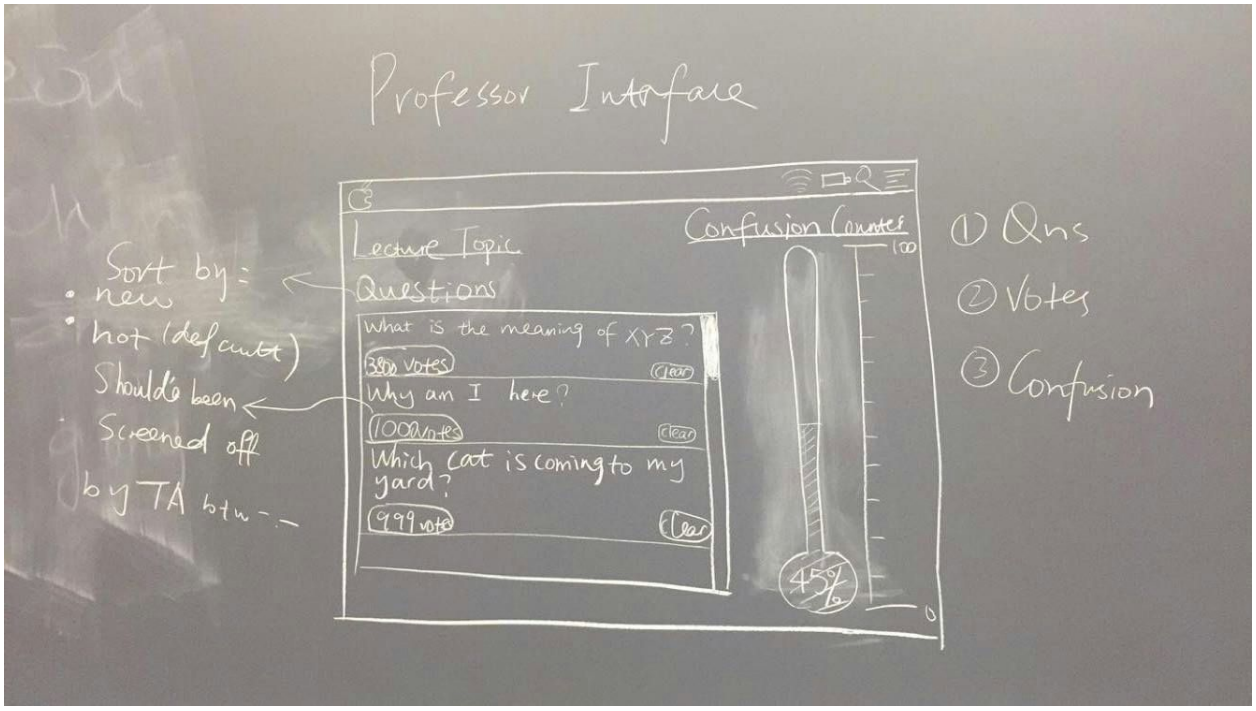
### Scenario B:

Jay Zee is diligently taking notes in COS 340 and listening to the professor explaining a complicated proof. Halfway through the proof, the professor says, “And obviously, equation x can be derived from equation y through the multiple theorems we learned about 3 weeks ago.” Poor Jay had no idea how equation y was derived, nor did he know which theorems the professor was referring to. In addition, the professor had made the process sound trivial and he was afraid of sounding stupid in class. Thankfully, Jay knew he could ask anonymously on Kweri, and within a few minutes of posting a question asking the professor to explain his process, the question had garnered more than 30 upvotes. Seeing this, the professor realized that what he had assumed to be trivial was confusing for his students, and so he went back and took a minute to fully explain the steps required.

**User Experience**

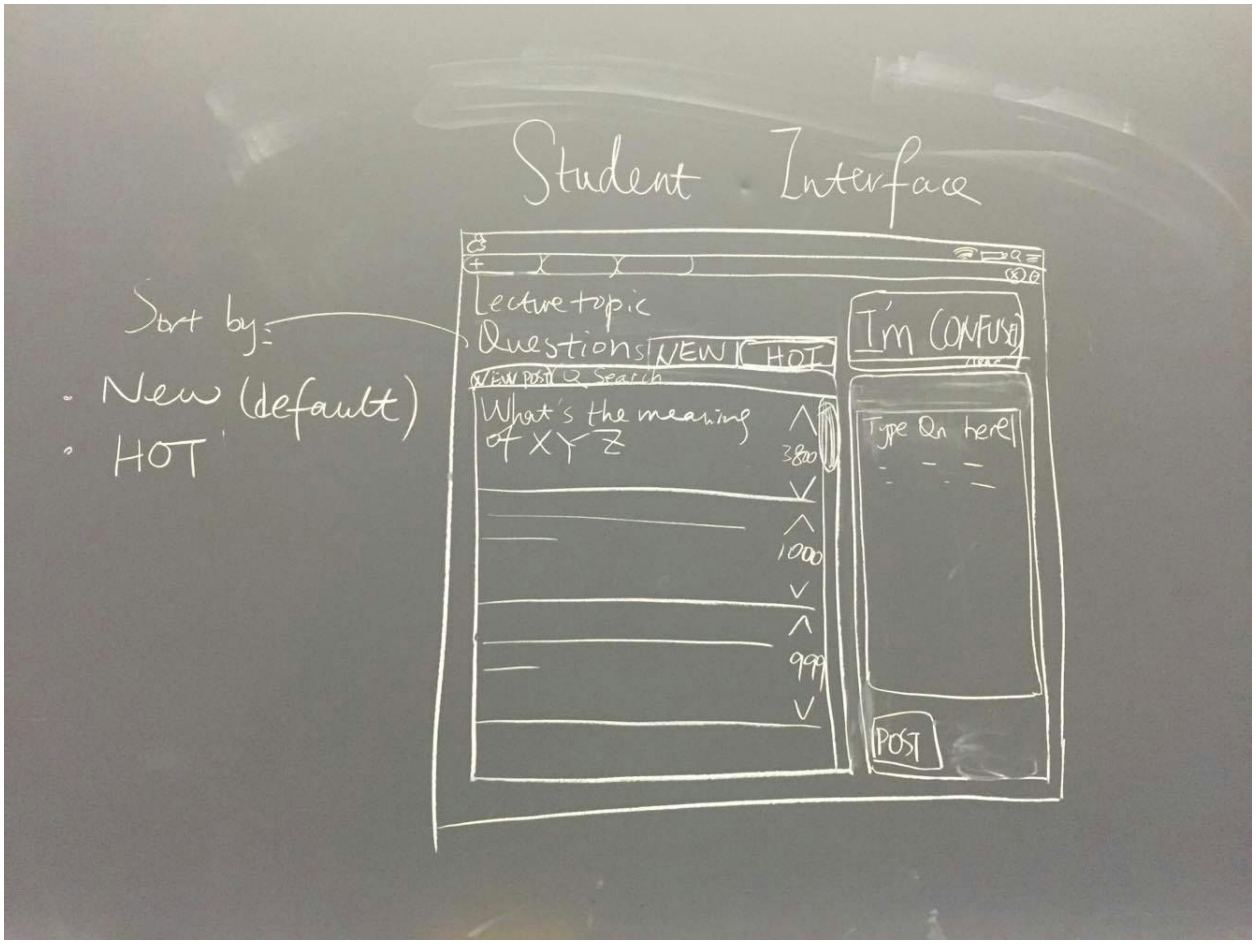
For professors:

On their screen, professors can see the questions students have posted and the votes on each particular question. Questions can be sorted by “new” or “hot” (similar to yik yak), with the default being set to “hot”. They can click on the “clear” button after explaining each question to clear the entries. On the side, there is a thermometer indicating the percentage of students that are confused at the moment. The number is updated in real time.




For students:

Students can post questions anonymously on Kweri during lecture for the lecturer to see. They can upvote or downvote the questions that other students have posted. The questions are sorted by “new”, by default. They can also click on the “I’m confused” button on the top right-hand corner to indicate that they are confused about the topic the lecturer is going through at the moment.




Maybe:
For TAs:

> The TAs can view the list of questions that students have posted. They can serve as a filter to prioritize more important questions/bigger concepts to reduce the distractions that the lecturer has.

**Design**

To create our Kweri web application we will be using Meteor, a full-stack JavaScript framework built on top of Node.js that supports all three components of our app: the user interface (frontend), the app logic (middleware), and the database (backend). Meteor provides a number of helpful classes and downloadable packages that can be utilized to achieve our design goals.

The user interface will be built using Meteor along with Blaze, an engine used to render templates, allowing for a UI that updates in realtime without page refreshing. More concretely, the web page will be created using HTML with Spacebars, a

Handlebars-like templating language that enables a dynamic UI as the Spacebars templates react to changes in the data they are representing and update automatically. Additionally we will utilize CSS to customize the visual style of our webpage.

The database used in our application will be the MongoDB database automatically provided by Meteor. Meteor wraps access to the database within a JavaScript API in the form of the Mongo.Collection class. The API allows for database access from both the client and server side through the use of the same syntax though the functionality differs slightly with each client only having as much access to the database as the server allows. A collection is the MongoDB equivalent of a table in SQL and the objects stored in the collection are JSON objects.

**Database format:**

- Class (Professor)
  - Lecture
    - confusion counter
    - question
      - upvotes
      - time
      - poster
      - boolean value (highlighted by TA)
- We will first be constructing a prototype that begins at the lecture sublevel. In other words we will only be storing one lecture at a time. However the finished product will be able to keep track of all past lectures so professors will be able to refer to past questions if necessary.

The app logic will be written in JavaScript utilizing several classes/libraries provided by Meteor either inherently or through additional packages. As stated earlier, Meteor provides the Mongo.Collection JavaScript class to handle database access. We will also be using the Tracker class provided by Meteor which integrates with Spacebars and Blaze to enable reactive updating of the UI instead of having to manually monitor database data and adjust the client UI through the use of jQuery. To handle user accounts we will likely start by using the downloadable Accounts package for Meteor which streamlines the creation of and logging in to user accounts. Similar packages exist for Google account login which we may also consider. We ideally would like to integrate the Princeton CAS login with our web application but this is an area we will explore more near the end of our project as there does not seem to be a direct JavaScript API available for the CAS system.

## Technical Summary

Frontend
-Meteor with Blaze (HTML + Spacebars)
-CSS for appearance

Middleware
-JavaScript for app logic
-Meteor's Tracker library for reactive UI updating

Backend
-Store data in "collections", the MongoDB equivalent of SQL tables.
-Meteor's Mongo.Collection class for database access from both the client and server
-Client database is a "pseudo-database," a subset of the server database that the server has published to the client, allowing the server to hide data from individual clients
-Login/User account using Princeton CAS (Potentially)
-Heroku to host our web server

## Timeline

- **Week 1**: Minimal interface, but connects to backend
- **Week 2:** Minimal interface, working features
- **Week 3**: Student/Professor interface divide, working features
- **Week 4**: Nice interface, working features
- **Week 5**: Prepare demos
- **Week Extra time**: TA Interface
- **Week Extra Time**: Mobile apps

## Risks and Outcomes

- Our group has minimal web development experience, but we are eager, excited, and enthusiastic to learn!
- To launch this website, we need good marketing strategies so that professors actually use it. We are not guaranteed how popular it is going to be.
- Students will need their laptops in class. They could be easily distracted online and professors may not be a big fan of this idea (although let's be real most students are already on Facebook/Reddit if they have their laptops out)

- Lecturers may also need to juggle many tasks at once: lecturing, changing slides, keeping an eye on the questions and the confusion counter, answering questions and so forth. They may be reluctant to use this tool.
    - But if we hit our stretch goal, the TA can handle this! (Of course they may not be as thrilled about this as we are, since it's extra work on their part.)