Speak↑: Low-Fi Prototype

"Better lectures, powered by real-time student data"

Team

Name	Email	Role
Brad Reyes	breyes28@	Dr. Bradass: Professional Demon Exorcist
Karen Gomez	kgomez@	Mother of Dragons
Nick Akiona	nakiona@	Grandmaster Champion
Reid Watson	rawatson@	Supreme Mugwump

Introduction

This document describes a prototype mobile app designed by the SpeakUp team in the Fall 2014 offering of CS147 at Stanford. Team members constructed an interactive "medium-fi" prototype of a mobile app for enabling feedback and communication in large lectures. The web version of this project is hosted at https://web.stanford.edu/~rawatson/cs147/, where you can find links to the runnable prototype and README file.

Student Tasks

Task	Difficulty	Description
1	Simple	**Sign up for a class**: A student who has never used the SpeakUp app before must sign up as a student in the CS147 class at the prompting of the instructor.
2	Medium	**Respond to a question posed by the lecturer**: A lecturer has posed a "clicker question" for the class to answer. The student must respond with the correct answer ("low-fi")
3	Complex	**Provide feedback on a lecture**: A student sitting in lecture decides that the content is being presented in an unclear fashion, and the pace is moving too quickly. The student should also indicate that they don't understand the part about the human eye.

Based on the results of our user testing in the low-fi prototype, we decided to keep the student tasks unchanged, but designate the "sign up for a class" task as the easy task. This is based on the fact that most testers found that task to be very simple, and the previous simple task proved to be difficult for some participants.

The "pose a question to the lecturer" task is still a component of our app, and is working in the prototype. We simply decided to use it for the purposes of this report in order to ensure that we had an easy, medium, and complex task.

Lecturer Tasks

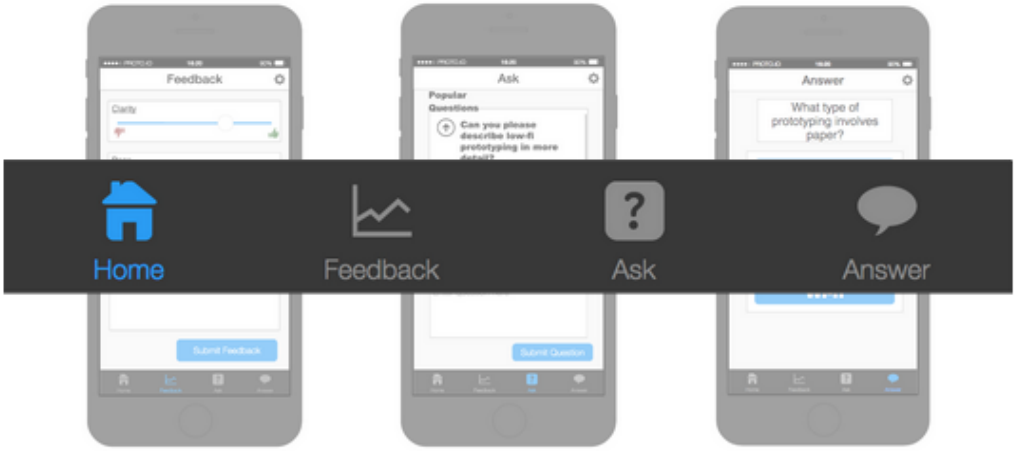
Task	Difficulty	Description
1	Simple	**View feedback on lecture**: An instructor should be able to glance at their feedback from lecture and understand what students are saying about their lecture.
2	Medium	**Read and answer questions from students**: An instructor should be able to view the top questions asked by students and answer the most popular questions, marking them as resolved once finished.
3	Complex	**Create and ask a "clicker question"**: An instructor should be able to create and ask a "clicker question", which is opened for responses from all students in the lecture.

The lecturer tasks we chose are unchanged from our previous iteration. These three tasks span a nice range of difficulty, and they cover the three most important features of our app (communication between student and lecturer).

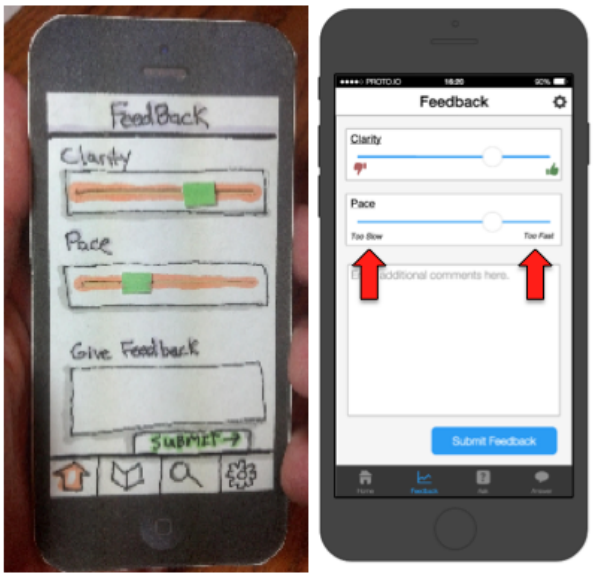
Revised interface design

Major UI Changes

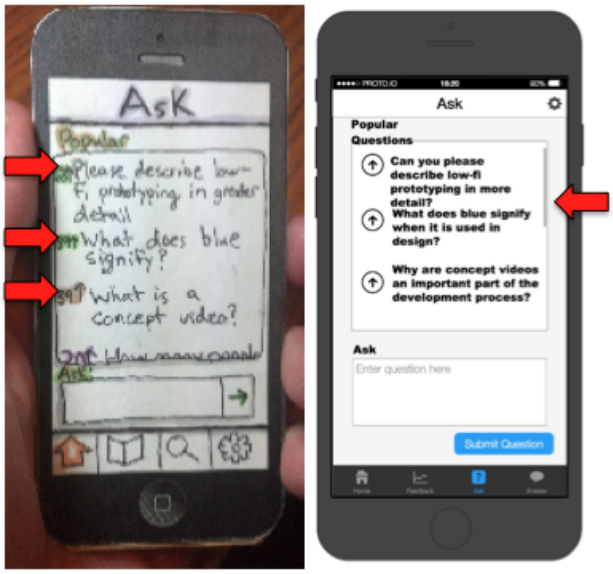
Menu Bar: To switch between the “Ask”, “Answer”, and “Feedback” screens, our low-fidelity prototype used a swipe feature where users could move between the three tasks. However, during testing, none of the participants knew that this feature existed, as there was no button to show the feature, and were unable to easily transition between the three tasks. By adding a menu bar at the bottom of the screen, with a button for each task, the student can easily and quickly navigate between the three tasks.



Sliders: While testing our low-fidelity prototype, we told users to submit feedback using the sliders to indicate that the professor was covering material too quickly. Some participants moved the slider to the left, while others slid it to the right. To eliminate this ambiguity, we added labels to the bottom of the slider named “Too Slow” and “Too Fast.”



Ask Screen: During testing, participants did not know they had the ability to look through already submitted questions before posing new questions. To make this function obvious, we added a scrollbar to the right-hand side of the box listing the questions. Additionally, we decided to remove the number of votes that each question has received to reduce clutter on the screen, and instead plan to make this a backend functionality, where we show the questions that have been upvoted more times at the top of the screen.



Greying Out Menu Bar: We decided to gray out the buttons on the menu bar until a particular class was selected, to prevent a student from navigating to an Ask or Answer screen without choosing a class.

Task 1 (student)

Task 1: A student signing up for a class in SpeakUp

Task 2 (student)

Task 2: Answering a "clicker question" posed by the lecturer

Task 3 (student)

Task 3: A student submitting feedback on a lecture

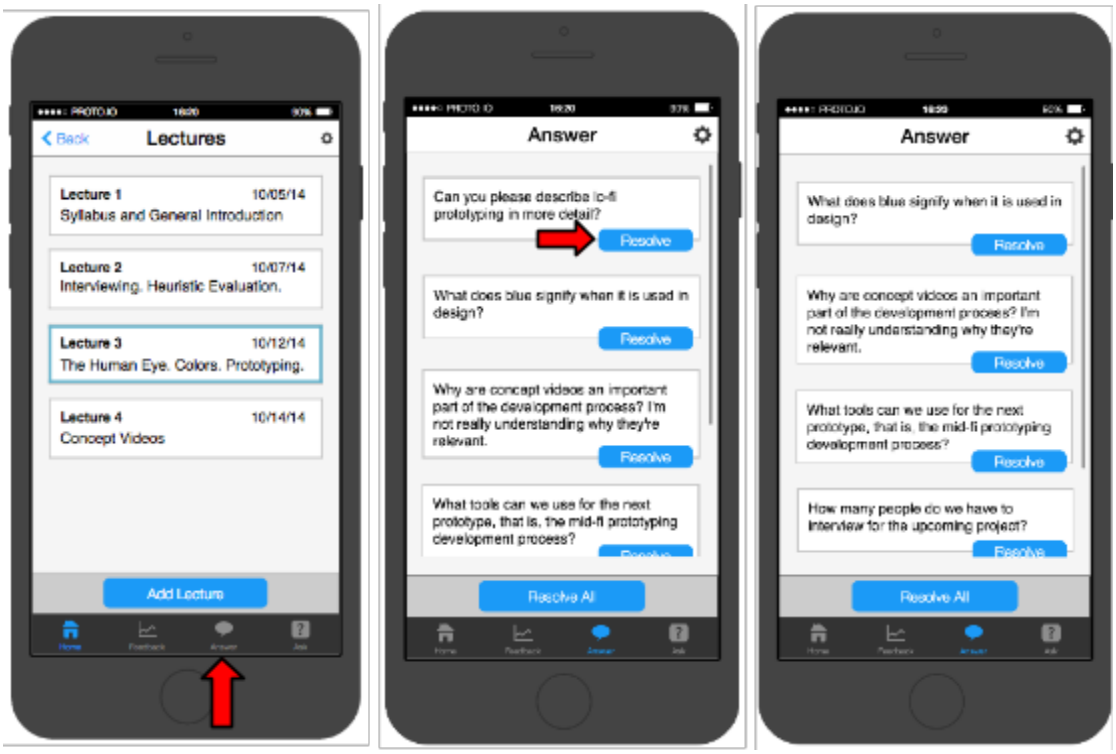
Task 0 (lecturer)

A lecturer selecting the class and lecture they would like to view

Task 1 (lecturer)

A lecturer selecting the class and lecture they would like to view

Task 2 (lecturer)



A lecturer viewing the questions asked by students during lecture.

Task 3 (lecturer)

A lecture creating a question and asking it to students

Prototype overview

Tools Used

We used proto.io as the primary prototyping tool for our app. We used their built-in assets and screen editor to design our UI and include basic interactivity. A link to the live prototype built is available at http://stanford.edu/~rawatson/cs147.
Proto.io proved to be a great tool for this:

- The UI for proto.io was very straightforward, and after a couple of hours it became much easier to add in components to our prototypes
- Easy to implement a clean, professional, and minimalist design.
- Lots of high-quality icons and templates to work with.

There were a few drawbacks to proto.io though:

- Documentation was very poor, and it was difficult to discover how to complete tasks that we were unfamiliar with. For example, it took a great deal of time to learn how to scroll an area of content.
- It was extremely difficult to work with UI elements which are generated from dynamic data. For example, when we wanted to set up a UI for searching for a class, it was difficult to create a meaningful UI for searching with autocomplete without an obscene amount of hackery.

We also used the color choosing tool Palleton to evaluate possible color schemes. Unfortunately, we weren't too happy with the color combinations it suggested. The suggestions seemed out of place for the setting and goals of our project, so we decided not to use it's results.

Limitations

While we were mostly happy with proto.io, the issues we had with it forced us to make some tradeoffs in the current prototype. An exhaustive list of limitations is available at http://stanford.edu/~rawatson/cs147. Instead of repeating the itemized limitations, we'll instead focus on the types of limitations and reasons we made the choices we did:

- **Proto.io doesn't support dynamically updating UI components**: If you've seen the autocomplete feature on Google web search, you've seen an example of a UI which makes realtime updates based on user input. For example, we wanted to make our search functionality update possible results in real-time, but we couldn't due to proto.io limitations. We believe that this is an acceptable limitation, since faking the feature would have been prohibitively difficult.
- **It's hard to keep transitions between states consistent**: In our "answer a question" screen, the "Go back to answer" should send you back to the answer screen with the selected answer highlighted. We couldn't figure out how to do this with proto.io, and there were a few other similar areas where we had trouble keeping consistency between transitions. This is an OK limitation, since it makes the UI only marginally less pleasant, and the fix would have been very difficult.

- **Impossible to dynamically update data**: When a user submits a question, we can't add the question to the list of "currently asked questions". This is an acceptable limitation because it 's seems literally impossible to do this using proto.io.

Wizard of Oz Techniques

An exhaustive list of Wizard of Oz techniques is available at http://stanford.edu/~rawatson/cs147. Instead of repeating the itemized limitations, we'll instead focus on the reasons we made the choices we did:

- **The settings button does nothing**: There are probably a few more things we'll need to configure (login, etc). However, they aren't very interesting or relevant to the primary tasks in the UI. As such, we decided to leave out the actual content for the settings button.
- **All data used in the UI is hardcoded**: Take our class list for example. In this class list, we only allow users to click on CS147. We could have made the other classes clickable, but it would have taken up a lot of time. This is an acceptable tradeoff, since the real product would be powered by real data instead of hardcoded UIs.

Prototype screenshots

In order to reduce filesize, we've linked all of our screenshots in a separate directory here: https://drive.google.com/folderview?id=0B6ylrwKHp7pYUUIzVEVQZ2FaSk0&usp=sharing