In-Class Student Feedback System

Simon Stepputtis
Tempe, USA
sstepput@asu.edu

Bijan Fakhri
Tempe, USA
bfakhri@asu.edu

Mao-Lin Li
Tempe, USA
maolinli@asu.edu

Chirav Dave
Tempe, USA
cdave1@asu.edu

ABSTRACT
Building effective student-teacher rapport in large classes is a challenge faced in many universities[1][3][6] and teaching centers. In response to this challenge, we have developed a Student Feedback communication system for enhanced real-time student-teacher communication. Our system allows teachers to field questions more effectively, gauge student understanding and interest, and conduct polls. By leveraging the ubiquity of mobile computers and their GPS capabilities, our system streamlines the classroom dialog while protecting against potential abuses.

Author Keywords
Human-computer interaction; Information visualization; Ubiquitous and mobile computing

INTRODUCTION
As classes become bigger, student engagement becomes an issue [2, 3]. One of the factors holding back students from engaging in class is the loss of anonymity when asking questions and the fear of being wrong in front of the whole class. To tackle this problem, we propose a Student Feedback System that allows for class engagement while giving students the ability to keep the notion of anonymity. After verifying a student is enrolled in the class, as well as his/her physically presence by using time and GPS data, our system allows the student to ask short questions to the lecturer as well as rate their current satisfaction of the class. All questions are shown to the rest of the class, giving them the ability to "up-vote" questions if they are also interested in the answer. Besides the question feature, students can also rate the current quality of the lecture from "too fast/don't understand" to "good speed/very clear". Both of these features are shown in a lecturer-only view (web interface) that visualizes the overall satisfaction over time, as well as the current questions and their votes. For this view, it is important to not interrupt the lecturer by requiring him/her to manually check the view. Thus, questions will be ordered by their number of up-votes to allow for a quick response. Also, the current satisfaction rate will be shown in a simple colored graph that conveys the information at a glance. Besides the student features, the lecturer also has the ability to do single-choice polls that are posted to the students app. The result of the polls will be shown in real time on the lecturer view, until the lecturer chooses to end the poll. With this app, we hope to provide a method to increase class engagement as well as enhancing the experience by introducing a modern approach to interactive teaching [4]. This application is also demonstrated in the following video, from a technical and visual standpoint: https://youtu.be/IPc3knbUP1E .

ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

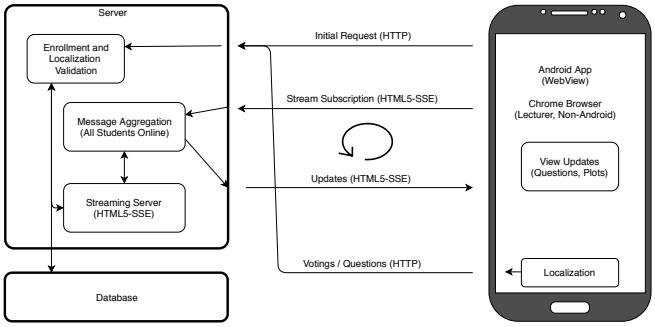

Figure 1. System overview and communication outline.

SYSTEM SETUP
Our system consists of two main components, (1) a back-end server that validates students and syncs requests; and (2) the actual clients, implemented as a collection of websites. A visualization of these components can be seen in figure 1.
On the client side, the user interface is implemented as a website, since this gives us the ability to be immediately usable by a wide variety of devices. Some features must also be accessible on a large screen for the lecturer to allow for a quick overview without having to pull out their personal phone. Since every classroom is equipped with a computer, this can be easily done by providing a website that enables such a use case. To reduce the workload of implementing and maintaining multiple front-ends, we created the Android app as simple webview that loads the requested website while making sure to do allow for GPS localization, hardware buttons and navigation.

The following two sub-sections discuss the actual implementation and functionality of the back-end server and the communication between the clients and the back-end.

Back-End Server

The back-end server is implemented in Python (using Flask) and uses a general database to store and access student and building information. It has two main purposes: First it validates if students are actually enrolled and physically present in the class they are trying to participate in and second, it synchronizes user responses and requests.
For the purpose of demonstrating the capabilities of our application, we created a database of the graduate level CSE 500 courses offered in spring 2018, and enrolled over 1700 randomly generated students in these classes, who each have a chance of 60% to take more than just one class (up to two). Besides checking student enrollment, the system is charged with validating correct location based on GPS coordinates, and also validate correct time and day information based on the server clock. For that purpose, we collected the GPS positions of all the buildings that are listed as classrooms for the CSE 500 level classes. When a user attempts to log in, his or her GPS location is taken and validated, to be within a radius of 200 meters of the location of the classroom. In the app, the location estimates are taken from the GPS sensor and showed to be pretty accurate, where as the Localization performed by a browser seems to have a high error margin. Both of the systems use the same code for localization, namely the JavaScript function *navigator.geolocation.getCurrentPosition(...)*. Another thing to notice is that the localization in the browser takes significantly longer than the when using the mobile app. However, in the crucial sections, a loading wheel is used to indicate that the view is currently gathering localization data. As of now, fresh data is used in each view.
Another important functionality of the server is to synchronize questions, satisfaction and polls. The communication is discussed in the next section.

Communication

Communication is an important requirement in the this project. Not only is it required when the client requests things from the server, which is the classic scheme, but also when other students contribute in any way (satisfaction, questions, polls), the server needs the ability to push information to the client.
For this project we decided to use HTML5-Server-Side-Events (HTML5-SSE) [5]. This technique allows the web-server to notify the clients when new data is available. This decision was made to prevent permanent custom TCP connections and Google push-notifications. The latter would have been the obvious choice, but requires a registration as Google developer to be usable, which we didn't want to to. Anyways, HTML5-SSE serves the desired purpose by keeping the connection that was initially opened by the client open, such that the server can later on use it to transmit additional data. In the code, this is realized with producer functions, that never return and *yield* information as they become available. The related code can be seen in the streaming functions in the server file.

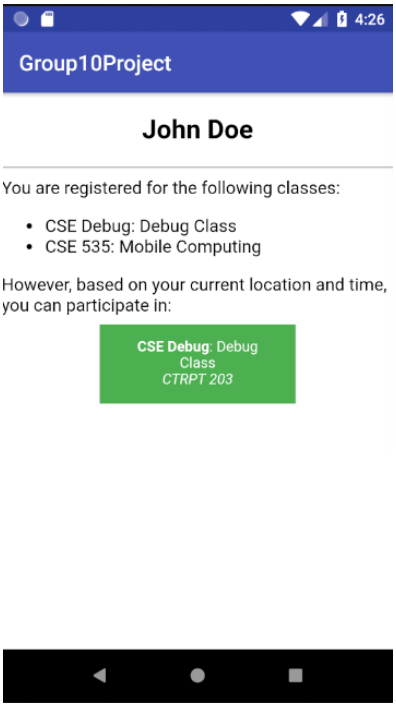

Figure 2. This page lists the courses the student can currently participate in.

FEATURES

In this section, we discuss the actual features of the applications, which are directly visible to the user.

Student Login

Upon starting the app, a student is greeted with the welcome screen (figure 7. The welcome screen prompts the student to sing-in using their username and password (shown in figure 8)). After a successful sign-in, the student is met with a page that displays all of the courses they are enrolled in and which are considered *active*. An *active* class is one that is currently in-session (according to the scheduled day and time of the class) and the student is currently in the vicinity of the classroom, as corroborated by GPS data. Active courses are displayed as shown in figure 2.

Student Satisfaction

After the student selects an active course from the list, they are taken to the student satisfaction tab (shown in figure 3). This tab displays, in real-time, the satisfaction of the students in the class. This allows for real-time feedback from the class to the professor. In large classrooms where gauging the satisfaction of the class as a whole is difficult, it is important for the professor to have the ability to quickly ascertain to what extent the class is following along. Techniques that work in small class settings do not translate well to larger classrooms. For example, in a small classroom, the professor often pauses their lecture to ask the class if they are following along, or if they need further explanation. Likewise, students in small class sizes are less likely to be intimidated when asking the professor for clarification or a change in lecture speed. To mitigate these pressures and facilitate useful feedback to the professor,

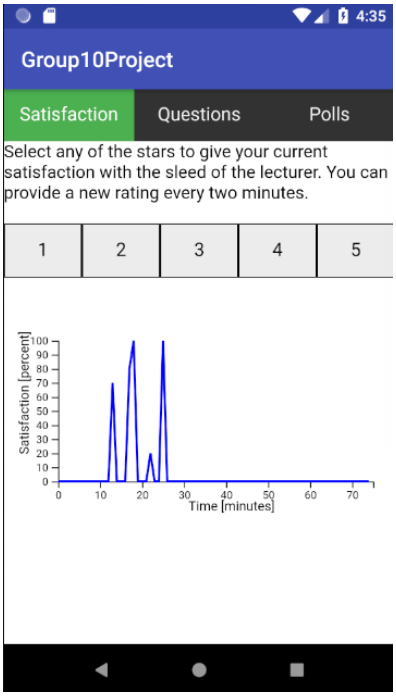

Figure 3. An image of the student satisfaction tab

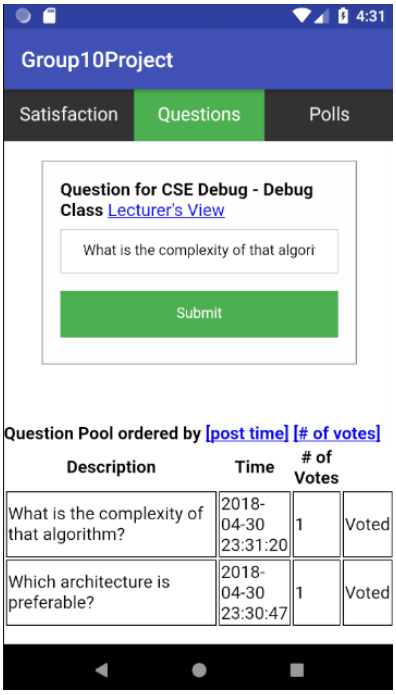

Figure 4. Shows questions logged to the server. The professor and all students can see these questions, allowing a democratic question-answering process.

we have implemented the student satisfaction feature of the Student Feedback System.

The student satisfaction page (shown in figure 3) displays a graph, illustrating the cumulative class satisfaction versus time. Above this graph, there are five buttons students can press in order to submit their satisfaction. If a student feels compelled to share their current satisfaction, they can press one of those buttons (1 - least satisfied, to 5 - most satisfied). The application will take that satisfaction score and send it to the server. The server compiles all of the students score submissions and makes it available to all of the student's applications, making graphing the student satisfaction in real-time possible.
This feature is safeguarded from abuse via timestamping methods. After submitting a score, students are barred from submitting another satisfaction score within a predefined time window by the server. This prevents students from spamming the server; and wielding exaggerated influence on the satisfaction graph. The server also re-validates the student's GPS position after every satisfaction score submission. This ensures that students who have left the classroom can no longer submit student satisfaction scores, preventing students from signing in while in the classroom, leaving, then wreaking havoc on the student satisfaction graph from outside of the classroom. Such an abuse would obfuscate feedback to the professor.

Student Questions

One of the essential elements of the student-teacher relationship is the ability to ask questions in real-time, concerning the immediate topics at hand. The expertise of the professor can be fully utilized, as students can request clarification, elaboration, and discussion of related topics outside the scope of the scheduled topics (for example, topics related to a student's research or trending subjects). This element though is often at-risk in larger classroom settings. Asking questions in-class is inherently a difficult task for shyer students. This is made worse in larger class settings where asking a question can be perceived to invite the judgment of more people, and carries the risk of being perceived as wasting the time of many others. For the reasons above, we propose a democratic and anonymous real-time question submission process by which questions submitted can be viewed by anyone. These questions can be voted on by other students based on their usefulness, and are visible to the professor, allowing the professor to answer questions in order of importance. Figure 9 shows the questions tab, which implements these features.
Once students have submitted their questions, the question tab shows the available questions (as shown in figure 4) in order of post time or number of votes (question's importance).

Tables for Student Questions in Database

To make sure lecturers can reply to student questions not only during lecture("on-line") by real-time interaction, but also after class ("off-line") by web, e.g. blackboard, questions are stored in the server. Here we describe the back-end question storage implementation. First we introduce the database table design for student questions and the utilities we support for data access.
In our implementation, we use two tables *questions* and *votes* to store student questions and related voting information. Figure 5 shows the schema and the relation between these two tables.

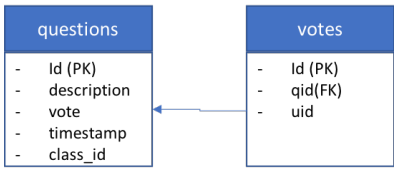

Figure 5. The relation between *questions* and *votes* tables and their schema.

Components of Question Table Schema

- id: The primary key to indicate a question
- description: The description for a question
- vote: The number of votes for a question
- timestamp: The timestamp for posting a question
- classId: The id to indicate the class that a question is posted.

Votes Table

The schema of *votes* table includes:

- id: The primary key to indicate a vote
- qid: The question id as a foreign key to questions table
- uid: The user id for this vote.

Data access functions

Here we list the functions and the SQL statements we use for the student question table.

1. listQuestion - Lists all questions in specific class, which includes question description number of vote and post timestamp. The SQL statement:
   *SELECT * FROM questions WHERE classID = "class id" ORDER BY "list order" DESC;*, we use the post timestamp as a default order, user can also use the number of votes as a list order.

2. insertQuestion - Inserts a new question into database, which includes question description, number of votes (default value: 1), and a timestamp. The SQL statements:
   INSERT INTO questions (description, classId) VALUES ("question description", "class id");
   INSERT INTO votes (qid, uid) VALUES ("question id", "user id");

3. voteQuestion - Votes for a question, which includes questionID and the user id who voted for the question. The SQL statements:
   UPDATE questions SET vote = vote +1 WHERE id="question id";
   INSERT INTO votes (qid, uid) VALUES ("question id", "user id");

Note that we assume when a user posts a new question that they intend to vote for their own question, so we set the default value to 1 for the initial number of votes. To avoid duplicate voting, when listing questions for a specific user, we will filter questions that user has already voted for by checking the *votes* table.
Furthermore, as we mentioned in the beginning, lecturers also can utilize the above functions to access all students' question in specific classes within the lecturers' view.

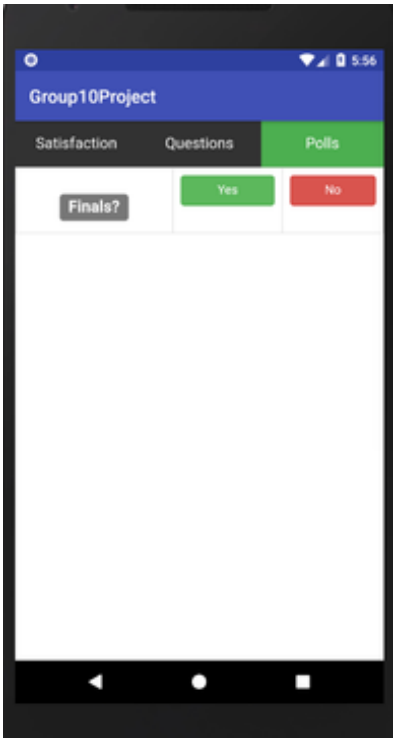

Figure 6. Student view for answering the question.

Polling System

Motivation

In addition to features beneficial for the students, we implemented features intended to benefit the lecturer as well. One such feature is the polling system, allowing the lecturer to perform single-choice polls which are then posted to the student's app. The result of the polls will be shown in real time on the lecturer view, until the lecturer chooses to end the poll. This feature allows accurate and efficient polling of the class, and has the inherent advantages of anonymity (encouraging more honest answers).

Implementation

The system first checks whether the user who is signed-in is a student or a lecturer. If that user is a lecturer, he/she will be allowed to see the results of the old poll if that poll has not yet been seen, otherwise he/she can create a new question. If the user logged in is a student, the interface will allow him/her to answer the current question. Once answered, the student will not be able to see the question until a new question is asked by the lecturer.

Work-flow

- *Lecturer View*

1. The lecturer signs-in to the system.

2. The lecturer can view the results of the previous poll if not yet seen (as shown in figure 9).
3. The lecturer can then ask a new question (as shown in figure 10).

- *Student View*

1. The student signs-in to the system.
2. The student will get to answer the current question only if he/she has not answered yet (as shown in figure 11).
3. Once the student answers the question, he/she will not be able to see the question until a new question is asked by the lecturer (as shown in figure 12).

RUNNING THE CODE

A demonstration of the application can be seen in the following video: https://youtu.be/IPc3knbUP1E
In the case you want to run the code on your own, navigate a terminal in the "backend" folder and run *python main.py*. Then, open a browser and go to "https://127.0.0.1:5000/". After accepting the security alert (ssl is required to be able to use GPS in the browser), you should see the login screen.
In case you want to run this on your phone (or simulator), a few code changes need to be made to allow communication. First off, the IP of the current machine needs to be found (127.0.0.1 does not work). This IP needs to be entered in "server.py" line 104 and MainActivity.java line 71. However, a real phone and the computer need to be on the same network and able to communicate with each other. The latter does not seem to be possible on the "asu" network.
To sign in to the system, a student ID between 1 and 1723 can be chosen. For demonstration purposes, the password is the same as the student ID. After login, it is unlikely you will see an available class (because your location, time, and student enrollment must match the database). For demonstration, this can be simulated in the database by creating a class that happens to occur in your office at the current time/day. For that, edit the database file in the "data" folder named "database.sqlite3". To simulate a class in your office, edit the "classes" table and change the day and time of the class with id 34 to the current day and time. Next, change the building with Id 21 to your current GPS location. This should allow you to select the "CSE Debug" class. However, there is another way to test the app. In database.py line 14, "disable_localization" can be set to "True". This just accepts all days, times and locations; and allows for immediate testing. One drawback is that the "Satisfaction" plot will not work in this case, since actual class times can not be calculated. For that purpose, you would need to fake an actual class in your office.

TASK DISTRIBUTION AND COMPLETION

The distribution of the different tasks can be seen in table 1. All tasks have been completed.

CONCLUSION

We developed application to help drive discussion, decrease confusion, democratically give voice to questions, and did so in a way that is robust and difficult to abuse. Our application

Id	Task	Assignee
1	Create a dummy database with student enrollment data	Simon
2	Gather the GPS position of the Tempe lecture halls place them in a database to verify student locations.	Simon
3	Create a centralized server that synchronizes questions, polls and satisfaction by using JSON or Android Push-Notifications.	Simon
4	Create a security feature, that makes sure students are actually enrolled and present in class	Simon
5	reate a simple website skeleton for the lecturer view, containing a menu and empty frames for each feature	Simon
6	Create and design a tab in the Android app called "Satisfaction" and provide a description	Bijan
7	Make sure that each student can only submit their satisfaction once every 2 minutes	Bijan
8	Create an html frame that shows a graph in the lecturer view, indicating the current student satisfaction over time and make sure it updates on a regular basis.	Bijan
9	Create a general communication system with the server.	Bijan, Simon
10	Take care of the general Android app design.	Bijan, Simon
11	Create and design a tab in the Android app called "Questions" and provide a description	MaoLin
12	Allow students to post short questions and send them to the back-end server	MaoLin
13	Allow students to up-vote questions other students asked.	MaoLin
14	Make sure the view queries the server for updates on a regular basis as well as manual updates.	MaoLin
15	Create an html frame in the lecturer view that shows all questions and make sure the frame updates on a regular basis.	MaoLin
16	Create and design a tab in the Android app called "Polls" and provide a description.	Chirav
17	Make the tab update on a regular basis (and manually)	Chirav
18	Provide a method to answer questions in a single choice manner from within the student app	Chirav
19	Make sure students can't answer the same question multiple times	Chirav
20	Create an html frame in the lecturer view that allows for creating a question and see the live-answers. Make sure the frame updates on a regular basis.	Chirav

Table 1. Task distribution

is a response to the problems that arise in large classrooms but can also facilitate discussion in smaller classrooms. We also developed and employed anti-abuse features such as the GPS localization, and day and time corroboration to make this application more robust and less susceptible to abuse.

ACKNOWLEDGMENTS

We would like to thank Dr. Banerjee and Junghyo Lee for giving us the opportunity and guidance to develop this application.

REFERENCES

1. Nathan Barrett and Eugenia F. Toma. 2013. Reward or punishment? Class size and teacher quality. *Economics of Education Review* 35 (2013), 41 – 52. DOI: http://dx.doi.org/https://doi.org/10.1016/j.econedurev.2013.03.001
2. Peter Blatchford, Paul Bassett, and Penelope Brown. 2011. Examining the effect of class size on classroom engagement and teacher–pupil interaction: Differences in relation to pupil prior attainment and primary vs. secondary schools. *Learning and Instruction* 21, 6 (2011), 715–730.
3. Steven G. Dieterle. 2015. Class-size reduction policies and the quality of entering teachers. *Labour Economics* 36 (2015), 35 – 47. DOI:http://dx.doi.org/https://doi.org/10.1016/j.labeco.2015.07.005
4. TU Freiberg. 2014. myTU App. (2014). http://mytu.tu-freiberg.de/mytu-evaluierung-erfolgreich-abgeschlossen/
5. F. Reynolds. 2009. Web 2.0 150;In Your Hand. *IEEE Pervasive Computing* 8, 1 (Jan 2009), 86–88. DOI: http://dx.doi.org/10.1109/MPRV.2009.22
6. Claudio Sapelli and GastÃČÂşn Illanes. 2016. Class size and teacher effects in higher education. *Economics of Education Review* 52 (2016), 19 – 28. DOI: http://dx.doi.org/https://doi.org/10.1016/j.econedurev.2016.01.001

APPENDIX

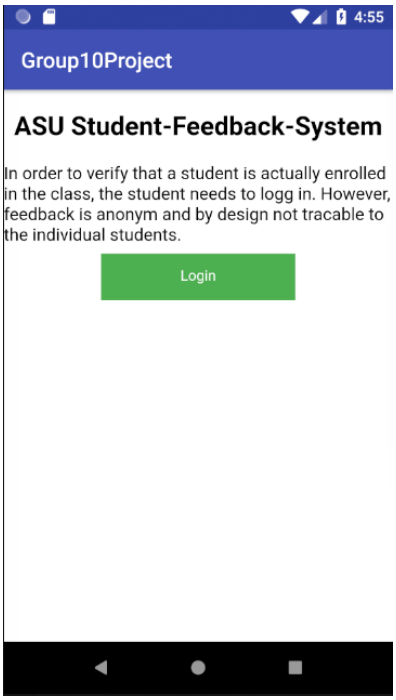

Figure 7. Testing

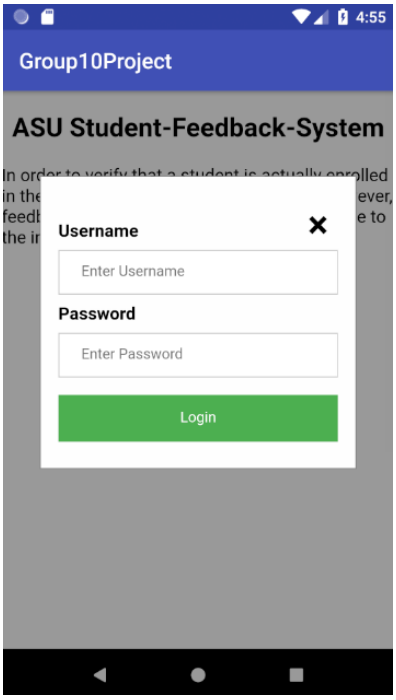

Figure 8. The user is prompted for their username and password on this page.

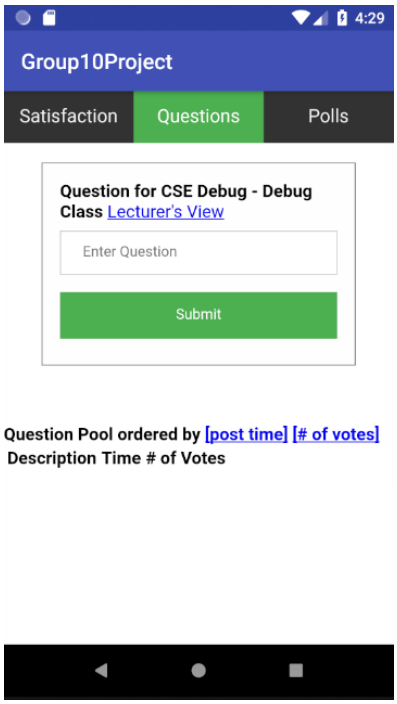

Figure 9. The questions tab. Students can type a question and submit it to the rest of the class by pressing the "submit" button.

Figure 10. Result view of the poll.

Figure 11. New poll created by the lecturer.

Figure 12. Once a student answers a question, he/she cannot see the question anymore.