



# ClassHub

Project URL:

<https://sites.google.com/scarletmail.rutgers.edu/classhub>

## Group Number 6: Report 3

<b>Omar Atieh</b>	<b>oa183@scalretail.rutgers.edu</b>
<b>Abderahman Sherif</b>	<b>ajs557@scarletmail.rutgers.edu</b>
<b>Nada Ali</b>	<b>n.ali@rutgers.edu</b>
<b>Wahhaj Zahedi</b>	<b>wmz3@scarletmail.rutgers.edu</b>
<b>Mohammad Alnadi</b>	<b>ma1322@scarletmail.rutgers.edu</b>
<b>Shazim Chaudhary</b>	<b>shc77@scarletmail.rutgers.edu</b>
<b>Salman Hashmi</b>	<b>sah285@scarletmail.rutgers.edu</b>

# Table Of Contents

---

<b>Contributions Breakdown</b>	<b>3</b>
<b>Summary of Changes</b>	<b>4</b>
Enumerated Functional Requirements: Added many use cases and updated diagrams	5
<b>Customer Problem Statement:</b>	<b>6</b>
A. Student Perspective	6
B. Instructor Perspective:	7
C. Glossary of Terms:	9
<b>3. System Requirements:</b>	<b>10</b>
A. Enumerated Functional Requirements:	10
B. Enumerated Non-functional Requirements	11
C. On Screen Appearance Requirements	11
User Interface Requirements	13
<b>4. Functional Requirements Specification</b>	<b>13</b>
A. Stakeholders	13
B. Actors and Goals	14
C. Use Cases	14
D. Use Case Diagram	17
E. Traceability Matrix	17
E. Fully-Dressed Descriptions	19
<b>5. Effort Size Estimation</b>	<b>25</b>
Calculating UCP:	25
Calculating UUCP:	26
Calculating TCF:	28
Calculating UCP:	29
Calculating Duration:	29
<b>6. Domain Analysis</b>	<b>29</b>
A. Domain Model	29
1. Concept Definitions (D-doing; K-knowing; N-neither)	30
2. Association Definitions	31
3. Attribute Definitions	33
4. Traceability Matrix	35
B. System Operation Contracts	36

<b>7. Interaction Diagrams</b>	<b>38</b>
Login/Register	38
Class Setup	39
Questions	40
Feedback	41
<b>8. Class Diagram and Interface Specifications</b>	<b>42</b>
Class Diagrams	42
<b>9. System Architecture and System Design</b>	<b>53</b>
Architectural Styles:	53
Identifying Subsystems:	54
Mapping Subsystems to Hardware:	56
Persistent Data Storage:	57
Network Protocol:	57
Global Control Flow:	58
Hardware Requirements:	58
<b>10. Algorithms and Data Structures</b>	<b>59</b>
<b>11. UI Design and Implementation</b>	<b>59</b>
<b>12. Design of Tests:</b>	<b>75</b>
Unit Testing	75
<b>13. History of Work, Current Status, and Future Work</b>	<b>79</b>
History of Work	79
Key Accomplishments	82
<b>14. References</b>	<b>82</b>

## Contributions Breakdown

	Wahhaj Zahedi	Salman Hashmi	Shazim Chaudhary	Omar Atieh	Nada Ali	Abderahman Sherif	Mohammad Alnadi
Class Diagram	14.28	14.28	14.28	14.28	14.28	14.28	14.28
Data Types and Operations	14.28	14.28	14.28	14.28	14.28	14.28	14.28
Traceability Matrix	14.28	14.28	14.28	14.28	14.28	14.28	14.28
Identifying Subsystems	14.28	14.28	14.28	14.28	14.28	14.28	14.28
Mapping Subsystems to Hardware	14.28	14.28	14.28	14.28	14.28	14.28	14.28
Persistent Data Storage	14.28	14.28	14.28	14.28	14.28	14.28	14.28
Network Protocol	14.28	14.28	14.28	14.28	14.28	14.28	14.28
Global Control Flow Hardware Requirements	14.28	14.28	14.28	14.28	14.28	14.28	14.28
Algorithms and Data Structures	14.28	14.28	14.28	14.28	14.28	14.28	14.28
User Interface Design and Implementation	14.28	14.28	14.28	14.28	14.28	14.28	14.28
Design of Tests	14.28	14.28	14.28	14.28	14.28	14.28	14.28
Project Management and Plan of Work	14.28	14.28	14.28	14.28	14.28	14.28	14.28
General UI	14.28	14.28	14.28	14.28	14.28	14.28	14.28
Instructor View UI Interface	14.28	14.28	14.28	14.28	14.28	14.28	14.28
Database/server implementation	14.28	14.28	14.28	14.28	14.28	14.28	14.28
Web Socket communication	14.28	14.28	14.28	14.28	14.28	14.28	14.28

Student view UI interface	14.28	14.28	14.28	14.28	14.28	14.28	14.28
User Portal UI	14.28	14.28	14.28	14.28	14.28	14.28	14.28
Custom UI designs	14.28	14.28	14.28	14.28	14.28	14.28	14.28
Quiz UI and Functionality	14.28	14.28	14.28	14.28	14.28	14.28	14.28
Poll UI and Functionality	14.28	14.28	14.28	14.28	14.28	14.28	14.28
Google Nearby API Functionality	14.28	14.28	14.28	14.28	14.28	14.28	14.28
Audio Recording of Answers	14.28	14.28	14.28	14.28	14.28	14.28	14.28
Upvoting Comments and Questions in Feed	14.28	14.28	14.28	14.28	14.28	14.28	14.28
Direct Messaging Instructors	14.28	14.28	14.28	14.28	14.28	14.28	14.28
Resource Uploading	14.28	14.28	14.28	14.28	14.28	14.28	14.28

**\*\*All Team Members Contributed Equally\*\***

Each member contributed equally to the project in terms of module break down. Wahhaj Zahedi worked on database/server and implementation across the full stack. Mohammad Alnadi and Shazim Chaudhary worked on the Instructor View module and UI. Salman Hashimi and Omar Atieh worked on the User Portal and web socket communication. Nada Ali and Abdurrahman Sherif worked on the Student View module and UI. The team collaborated very closely such that each team member was able to contribute equally..

## Summary of Changes

### 1. Attendance Mechanism:

- Main Change, using Google Nearby API to make attendance absolute and uncheatable. Google Nearby uses a combination of Bluetooth, Bluetooth Low Energy, Wi-Fi and near-ultrasonic audio to communicate a unique-in-time pairing code between devices that are connected to the internet, but not necessarily on the same network.
- **Details on the API:** <https://developers.google.com/nearby/messages/overview>

## 2. System Requirements:

- **Enumerated Functional Requirements:** Added many use cases and updated diagrams
- Added many more requirements and change priorities
- Change login and register priorities much less
- Added new features
  - Direct Messages to Instructors
  - Upvoting system for live session feed
  - Resource uploading
  - Live session polls
  - Audio answer pinning to student questions
  - Revamped attendance system using a mix of different connective mechanisms (using Google's Nearby Messages API)

## 3. Functional Requirements Specification

- Updated All use cases and added many use cases to account for added features
- Updated matrix
- Added fully dressed description
- Added system sequence diagram

## 4. Effort Estimation

- Accounted for new features

## 5. Domain Analysis

- Updated matrix per additional use cases.
- Added new responsibilities per new features added to the mobile application.

## 6. Interaction Diagrams

- Improved interaction diagrams

## 7. Class Diagram

- Added new classes and function to class diagram to account for new features
- Added OCL
- Updated traceability matrix

## 8. System Architecture

- Explained database schema better

## 9. History Of Work

- Updated history of work
- Added key accomplishments

# Customer Problem Statement:

## A. Student Perspective

As a student, attending lecture is often troublesome as it may seem like a waste of time. The current lecture system has several flaws that make attending lecture more adverse than beneficial. These problems include: a lack of efficient and accurate attendance-taking, an absence of immediate test-taking to solidify understanding of the lecture material, a need for efficient communication between the professor and students, and a lack of the involvement in major decisions of the course such as major due dates.

A lack of efficient attendance-taking can prove to be quite a massive waste of time for a large class. For example, if 300 students have to sign a sign in sheet each lecture just to provide proof of their attendance than valuable time is wasted. This valuable time can be used by students in order to understand the material, take notes, and ask questions before lecture time is over. The main issue that arises as a student is that the sign in process takes up a large portion of the total time allotted for the lecture. At times, up to 15 minutes of class-time is wasted on attendance. The lecture commences with only a short time remaining - forcing the lecturer to rush through material. This puts tremendous pressure on students to take notes, understand what's being said, and comprehend what's being presented in too short of a time period. Additionally, this puts more pressure on Teaching and Learning Assistants, as they are forced to go over material in even more detail in order to compensate for the lost time due to the extensive sign-in process. This is not a good experience for the students as it encourages students to leave lecture immediately after signing in to go study the material at their own speed and time. This is not the only problem in the current lecture system which encourages the student to skip lecture, the lack of immediate test-taking during the lecture has also a significant effect on students' performance.

A lack of immediate test taking does not imply a graded 30-60 minute test on the material that was presented in the lecture as a whole. Statistics have shown that testing yourself immediately on a specific concept one has just learned improves one's understanding and

retention drastically. Take Sololearn for example, Sololearn is a web and mobile application that enables users to learn programming languages and concepts utilizing immediate test taking strategies. After each new concept, one would face a very short quiz that would ask the student to apply the concept they just learned in a scenario. If the often low exam scores and class averages are any indication, immediate test-taking is an absolute must in today's learning environment to boost the understanding and application of important concepts being presented during the lecture by the professor. Not only does this provide us with a better way to solidify the material, it also can be used as a metric to gauge whether students are actually understanding the material. Watching a problem being solved is significantly easier than attempting a problem on one's own. When students are tested under the supervision of the lecturer, they are able to figure out where they are getting stuck on. They then have the opportunity to ask the lecturer for clarification regarding the material. This provides an overall better experience than the one that students unfortunately go through today.

The last problem that needs to be addressed is the lack of student involvement in the course's major decisions such as deadlines. Often times, professors don't take into account the collective schedules of students when they decide on project deadlines. This can produce unnecessary hardship for both sides. Students often find themselves taking on an extreme amount of stress at times throughout the year because of impending deadlines that fall during the same time period. A polling system is a solution for this problem. The professor would use an application that is connected with the students to post a question along the lines of "When should this be due?" and the students would pick a due date from a selection of viable options. This alleviates a magnormous amount of stress off student's shoulders. The professor could then determine how he or she wants to proceed with the poll results and whether he or she wants to follow a majority-rules system.

A mobile application centered around the classroom communication between a student and professor is best option since almost every student possesses a smartphone. This mobile application can be the communication hub between the students and professors addressing all the solutions mentioned above from the students' perspective.

## **B. Instructor Perspective:**

As an instructor, holding lecture can often feel like a waste of time. If attendance isn't mandatory then most students would choose not to attend. Even when attendance is mandatory towards the class grade, the methodology of taking attendance through sign in



sheets, or iClicker allows students to simply walk into class, sign in, and exit right after. There can be various reasons for a student to leave, but a massive drop in lecture attendance indicates that the lecture is not beneficial. This brings in the issue of easy communication between the instructor and students regarding the lecture and course. Comfortable communication between the instructor and students is very important if the instructor aims to improve through student feedback. A comfortable communication platform also provides solutions to several immediate questions an instructor has during lecture such as: Are most of the students understanding the material being presented? Are students coming to class? If so, how many per lecture? What do the students think about the lecture and instructor's teaching methods?

To start with attendance, an efficient, full-proof attendance taking tool is required such that students are required to attend lecture and stay in lecture until the end all while participating actively with questions and quizzes. The old sign in sheets are often very time consuming if the lecture contains more than a hundred students. If the sign in sheet is left at the front, the rush/line of students around it often render the lecturer unable to start the lecture until everyone has signed in and is seated. The lecture can be started on time where the sign in sheet is passed around the classroom, however this provides problems of its own. This method allows students to dishonestly sign in for other students who are not present in the lecture to fulfill their attendance requirement. Not only that, but this method often leaves the instructor asking the class 'if everyone signed in' to which several students respond 'no' and come up to sign in at the end of the class. Overall, the sign in sheet is an unreliable and time-consuming attendance taking tool. Another option that instructors have tried is the iClicker attendance taking where the students bring iClickers to class and press a button to sign in. Although this method is less time consuming, it poses other problems such as: student has to buy an iClicker which is expensive on a college student's budget, and students can sign in and leave. A software platform that involves the student to sign in and stay until the end of the lecture with participation is the most viable solution.

A massive drop in attendance often indicates that the lecture is not proving to be beneficial towards student understanding. One solution for this is a software that provides a feedback platform for the students to express their constructive critique. Which then guides the professor on where to improve. This platform should provide a rating of the attended lecture along with a description explaining why the student gave the rating they did. This saves valuable time before and after lecture where students express concerns such as instructor's volume during lecture, the size of their font, their teaching style, and any concepts the instructor did not explain properly. On the note of beneficial feedback, the instructor requires immediate feedback on whether the students in the lecture are

understanding the theory behind the concept as well as its application. This calls for immediate 1-2 question quizzes during lecture after the coverage of each new concept. This component is essential for the instructor to understand how well the students are understanding the new material.

A strong communication system is still lacking in today's lecture system. Usually when an instructor asks a question, at most, one or two students raise their hands to resolve their misconceptions. However this is highly unrealistic. Only one or two students cannot be the only students among hundreds in to have questions at a college level course. Students often find it uncomfortable to shout in the middle of a lecture hall. This calls for a question-answer component such that students can post questions onto the application and have others upvote these questions if they have it as well. These questions are then addressed by the instructor either in the same application or during lecture.

This is a software platform built for mobile that allows the students and instructors to have attendance-taking, frequent test-taking, constructive feedback, and strong communication. This will enhance the classrooms of today's education system immensely.

## C. Glossary of Terms:

**Session:** A live virtual lecture for the class where instructors and students can interact with each other in real time through attendance, quizzes/polls, questions & answers, and feedback.

**Class:** A set of sessions, that holds the class roster information, class title, start and end dates for the class.

**Live session:** a video live stream for the active session. It can only be viewed by students enrolled in the class. The instructor is the only one who has the option to host a live session.

**Session history:** Live session record and statistics where instructor views class grades, attendees, and session information.

**Feedback:** At the end of the session, students are asked to rate it and give their comments on how a session can be improved.

## 3. System Requirements:

### A. Enumerated Functional Requirements:

IDENTIFIER	PRIORITY	REQUIREMENT
REQ-1	3	The system will allow users to register as either a student or instructor and login after authentication with their information.
REQ-2	4	The system will enable students to ask question real-time during the session, for all students and the instructor to view the students' questions.
REQ-3	5	This Application will use Google's nearby API, which uses a combination of Bluetooth, Bluetooth Low Energy, Wi-Fi and near-ultrasonic audio to communicate a unique-in-time pairing code between devices. Which can be utilized as an uncheatable attendance mechanism.
REQ-4	2	The system should allow both instructors and students to view the session history including questions, quizzes, and specifically attendance and reviews for the instructor.
REQ-5	4	The system should allow instructors to create quizzes within the session for students to answer in real-time.
REQ-6	2	The system shall provide statistics of attendance, quizzes and feedback per session for the instructor, as well as aggregated statistics per class.
REQ-7	1	The system shall enable instructors to send notifications and reminders to all students enrolled within a class.
REQ-8	3	The System should enable students to directly message the instructors
REQ-9	4	Upvoting system for live session feed to allow certain comments/question to be put on priority
REQ-10	2	The system should allow instructors to upload resources(pdf or slides) for students in the class.
REQ-11	5	Live session polls to take polls during live sessions

REQ-12	5	Audio Answering to allow instructors to record their verbal in class answer to a student's question and post it in resources
REQ-13	5	Functional database hosted on server to store all user data and classroom data for push and fetch commands.

## B. Enumerated Non-functional Requirements

IDENTIFIER	PRIORITY	REQUIREMENT
REQ-14	2	The system's view should be consistent for both students and instructors, and follow similar design patterns.
REQ-15	2	The system should be secure and each account should be private and only accessible by its owner.
REQ-16	2	The system should be accessible by different devices, and be responsive to all of them.
REQ-17	2	A forgot password mechanism must be implemented
REQ-18	4	Robust user navigation and interface

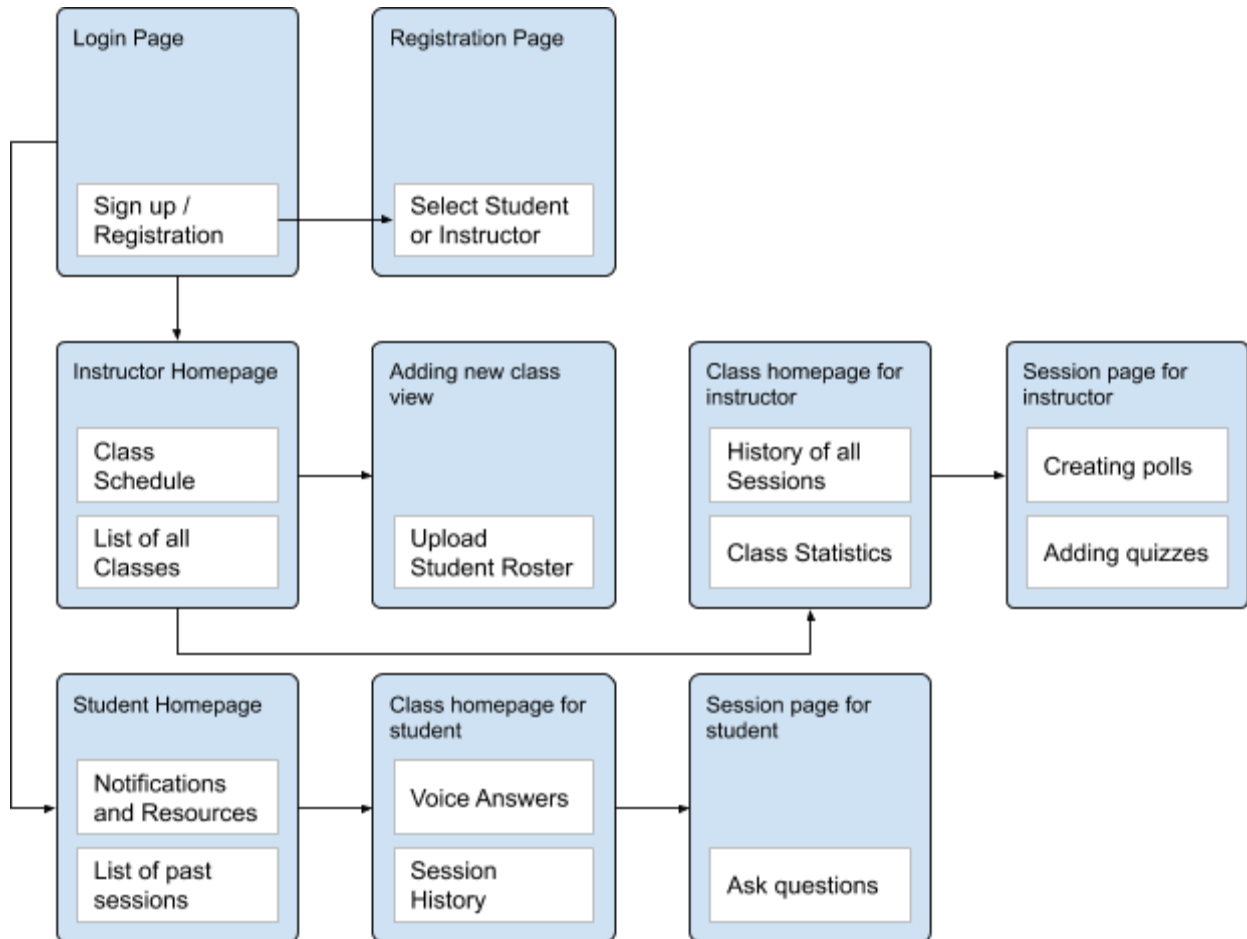
## C. On Screen Appearance Requirements

IDENTIFIER	PRIORITY	REQUIREMENT
REQ-19	3	Registration page for new users to sign up with their information in the system either as an instructor or a student.
REQ-20	3	Log-in page for students and instructors to access their accounts.
REQ-21	2	A forgot password interface to reset the password
REQ-22	3	Home page for students viewing their schedule of classes, and to access the classes already enrolled in.
REQ-23	3	Side menu to view additional classes.
REQ-24	3	Class home page for instructors to view previous sessions' history and statistics.

REQ-25	4	Home page for instructors to view their schedule, and all classes they are instructing.
REQ-26	5	Session view for instructors to post quizzes and host a live session.
REQ-27	3	Add a new session button for instructors to be able to upload a CSV file of a class roster
REQ-28	3	Session view for students to be able to take quizzes in class
REQ-29	4	Polls interface in session view to allow the instructor to ask polls and get results from students.
REQ-30	3	Feedback interface to show after a session for students to rate the class.
REQ-31	5	In session feed to allow communication for everyone in the class, via comments, questions, polls and quizzes.
REQ-32	3	Resources page for instructors to post lecture slides and other class documentation.
REQ- 33	2	Notification system for students to view

## User Interface Requirements

---



## 4. Functional Requirements Specification

---

### A. Stakeholders

*This system is created for implementation in classroom settings to help adapt classroom lectures and improve the overall education system.*

*Below are examples of Groups who would be interested:*

- *Students*
- *Professors*
- *Universities and schools*

## B. Actors and Goals

ACTORS	GOALS
<b>Students (participating)</b>	<ul style="list-style-type: none"> <li>To familiarize and test oneself on the material presented in class.</li> <li>To empower students to ask questions and comment on lecturer's sessions.</li> <li>To enable an elevated means of interaction with instructors.</li> </ul>
<b>Instructors (initiating)</b>	<ul style="list-style-type: none"> <li>To improve and make lectures more effective.</li> <li>To test student's knowledge and receive feedback on lectures.</li> <li>To ensure students attend lecture and are participating.</li> <li>To ease the process of facilitating quizzes and polls.</li> <li>To be able to communicate with students conveniently and in real-time.</li> </ul>
<b>Universities &amp; Schools</b>	<ul style="list-style-type: none"> <li>Ensure Professors are productive and utilizing their classroom times accordingly</li> <li>To ensure students are participating and are productive in lectures</li> </ul>

## C. Use Cases

### Casual Description

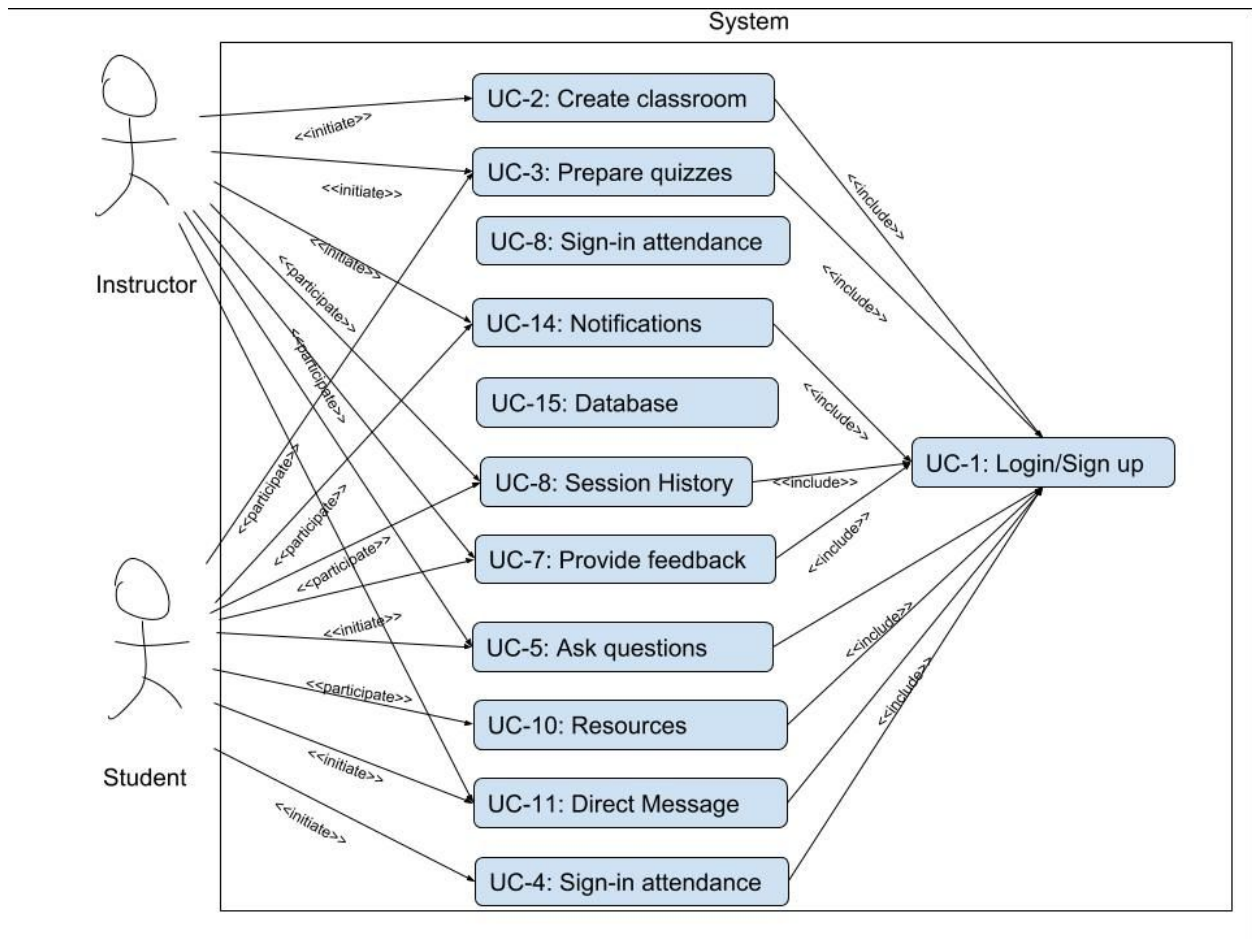
<i>Use Cases</i>	<i>Description</i>	<i>REQS</i>
<i>UC-1: Register</i>	<i>To create an account that will allow a student/instructor to join/create a classroom.</i>	REQ 1, 20, 15, 17, 19, 21

<i>UC-2: Class Setup</i>	<i>Allows an instructor to create a classroom by uploading a CSV file. Set up and organize classroom</i>	REQ 15, 16, 24, 25, 27
<i>UC-3: Session Preparation</i>	<i>Instructor prepares participation quizzes for each lecture session.</i>	REQ 23, 22
<i>UC-4 Attendance</i>	<i>Students get marked as present when they attend lecture through Nearby Messages API*</i>	REQ 3
<i>UC-5 Questions</i>	<i>Students asks questions in lecture through messaging feed</i>	REQ 12, 2
<i>UC-6 Answers</i>	<i>Instructor receives questions and answers them in class. Instructor can use Audio feature to record his answer and upload it to resources.</i>	REQ 12
<i>UC-7 Quizzes</i>	<i>Instructor creates an in class multiple choice quiz and sends it to all students.</i>	REQ 28, 26, 5
<i>UC-8 Polls</i>	<i>Instructor creates a poll inside a session and allows the students to vote</i>	REQ 29
<i>UC-9 Messaging feed</i>	<i>Allow students to upvote messages, questions, or comments during class to prioritize.</i>	REQ 9, 31
<i>UC-10 Resources</i>	<i>Archive of all resource uploaded by instructors for students</i>	REQ 10
<i>UC-11 Direct Message</i>	<i>Allow students to send messages directly to the instructor privately.</i>	REQ 8
<i>UC-12 Feedback</i>	<i>Students provide feedback for each lecture to help ensure the instructor if the students have grasped the</i>	REQ-30, 4, 6



	<i>material or whether there is something the instructor can improve on.</i>	
<i>UC-13 Session History</i>	<i>Student/Instructors can look through each session's history to see results of quizzes, attendance, feedback, and resources uploaded for that session.</i>	REQ 24, 6, 4, 11
<i>UC-14 Notifications</i>	<i>Instructor can send notification about upcoming events via app</i>	REQ 7, 33
<i>UC-15 Database</i>	<i>Functional Database to record all data of classes and users</i>	REQ 7, 13
<i>UC-16 Student/Instr uctor interface</i>	<i>View data that is fetched from database to display user data.</i>	REQ 14, 15, 18, ,21, 24, 25, 23

## D. Use Case Diagram



## E. Traceability Matrix

Req't	P W	U C1	U C2	U C3	U C4	U C5	U C6	U C7	U C8	U C9	UC 10	UC 11	UC 12	UC 13	UC 14	UC 15	UC 16
REQ1	3	x															
REQ2	4					x											
REQ3	5				x												
REQ4	2												2	x			
REQ5	4							x									
REQ6	2												2	x			

REQ7	1															x	x	
REQ8	3											x						
REQ9	4									x								
REQ10	2										x							
REQ11	5													x				
REQ12	5					x	x											
REQ13	5																x	
REQ14	2																	x
REQ15	2	x	x															x
REQ16	2		x															
REQ17	2	x																
REQ18	4																	x
REQ19	3	x																
REQ20	3	x																
REQ21	2	x																x
REQ22	3			x														
REQ23	3			x														x
REQ24	4		x											x				x
REQ25	4		x															x
REQ26	5							x										
REQ27	3		x															
REQ28	3							x										
REQ29	4								x									
REQ30	3													x				
REQ31	5									x								
REQ32	3																	
REQ33	2																x	

Max PW		4	4	3	5	5	5	5	4	5	2	3	3	5	3	5	4
Total PW		15	15	6	5	9	5	12	4	9	2	3	7	13	3	6	21

## E. Fully-Dressed Descriptions

<b>Use Case UC-1</b>	<b>Login/Sign up</b>
Related Req'ts	REQ 1, REQ 15, REQ 17, REQ 19, REQ 20, REQ 21,
Initiating Actor	Instructor, Student
Actor's Goal	To create an account or login to access the system. All user's data must be pulled from server
Participating Actors	Instructor, Student
Preconditions	<ul style="list-style-type: none"> <li>● The system database is running live</li> <li>● The system displays login/sign up interface with email and password input</li> <li>● Database will verify all credential of login to launch the user's data</li> </ul>
Postconditions	<ul style="list-style-type: none"> <li>● System allows user to reset password if they have forgotten it.</li> </ul>

Flow of Events for Main Success Scenario:

1. Instructor/Student opens the app and inputs user information
2. System a) verifies users, b) displays home page
3. System a) detects error, b) signals the user to enter valid input

<b>Use Case UC-2</b>	<b>Class Setup</b>
Related Req'ts	REQ 15, REQ 16, REQ 24, REQ 25, REQ 27
Initiating Actor	Instructor
Actor's Goal	Instructor is able to make a class by uploading a roster of emails in the form of CSV file. Instructor is able to plan class by uploading

	resources, planning quizzes and notify all students.
Participating Actors	Student
Preconditions	<ul style="list-style-type: none"> <li>• The system database is running live</li> <li>• User is logged in with instructor account</li> <li>• Instructor can now upload their CSV file to populate their roster</li> <li>• All students on roster automatically get enrolled into class through registered email</li> </ul>
Postconditions	<ul style="list-style-type: none"> <li>• Instructor sets up class by uploading resources, and sets up quizzes</li> </ul>

Flow of Events for Main Success Scenario:

1. Instructor logs into ClassHub
2. System a) verifies users, b) displays home page
3. User clicks on plus button, and adds class with appropriate class information
4. Instructor uploads the roster by csv file
5. Database parses the csv file and registers emails listed on the roster
6. Instructor can set up resources and quizzes for this class.

Use Case UC-6	Answers
Related Req'ts	REQ 12
Initiating Actor	Student
Actor's Goal	Instructor views the questions asked by student in live session. If the chooses to answer them, then he can answer them via audio recording and uploading to messaging feed.
Participating Actors	Instructor
Preconditions	<ul style="list-style-type: none"> <li>• The system database is running live</li> <li>• User is signed in with student account</li> <li>• User is enrolled in class</li> <li>• Class session is active and students are able to ask questions</li> <li>• Questions submitted through server and can be seen on feed by instructor</li> </ul>

Postconditions	<ul style="list-style-type: none"> <li>• Students can upvote certain question to introduce priority questions.</li> <li>• Instructor uses audio to save verbal answers for students</li> </ul>
----------------	--

<p>Flow of Events for Main Success Scenario:</p> <ol style="list-style-type: none"> <li>1. User signs into ClassHub with student account</li> <li>2. User enters into active class session</li> <li>3. User submits question for instructor to view and answer</li> <li>4. Students can upvote the more important questions</li> <li>5. Instructor can answer the questions via audio and upload that audio to the database</li> </ol>
--

<b>Use Case UC-12</b>	<b>Feedback</b>
Related Req'ts	REQ30 REQ 4 REQ 6
Initiating Actor	Student
Actor's Goal	To provide lecture-specific constructive feedback and rating for instructor to use to improve lecture quality
Participating Actors	Instructor
Preconditions	<ul style="list-style-type: none"> <li>• The system database is running live</li> <li>• Class session had been active and concluded</li> <li>• Student is enrolled in class</li> <li>• Feedback is submitted through server and instructor has access to feedback to improve lecture quality</li> </ul>
Postconditions	<ul style="list-style-type: none"> <li>• Overall feedback will be aggregated, calculated in to an overall review</li> </ul>

<p>Flow of Events for Main Success Scenario:</p> <ol style="list-style-type: none"> <li>1. User logs into ClassHub with student account</li> <li>2. User accesses concluded class session and submits feedback for instructor to view</li> <li>3. Feedback is submitted to database and can be viewed in session history</li> <li>4. Feedback is aggregated overall review of the professor will be shown</li> </ol>
--

<b>Use Case UC-4</b>	<b>Attendance</b>
----------------------	-------------------

Related Req'ts	REQ3
Initiating Actor	Instructor
Actor's Goal	Once instructor starts a session, <a href="#">Google's Nearby API</a> will use internets, bluetooth, bluetooth low Energy, and near ultra sonic audio to communicate a unique in time pairing code between devices.
Participating Actors	Instructor, Student
Preconditions	<ul style="list-style-type: none"> <li>• Live session are active using real time sockets. Attendance entrance and exit events are able to be marked in the database.</li> <li>• Nearby API has been tested and works undoubtedly</li> </ul>
Postconditions	<ul style="list-style-type: none"> <li>• Need to run more test with greater amount of users</li> </ul>

**Flow of Events for Main Success Scenario:**

Instructor launches live session in class.

Students who are logged in to the internet are able to be detected by instructor

Students are able to gain access to the session and participate in class.

Once session has ended they all data is sent to database

<b>Use Case UC-8</b>	<b>Polls</b>
Related Req'ts	REQ29
Initiating Actor	Instructor
Actor's Goal	Instructor is able to ask a poll question in the classroom. All student will get this popup and answer the poll. The poll results will be displayed in the feed.
Participating Actors	Instructor, Student
Preconditions	<ul style="list-style-type: none"> <li>• Live session is established and connect through sockets to all students in class.</li> <li>• Feed is active and connected to all the students</li> <li>• Poll is able to be launched and made by the instructor</li> <li>• Students are able to view and answer the poll.</li> </ul>

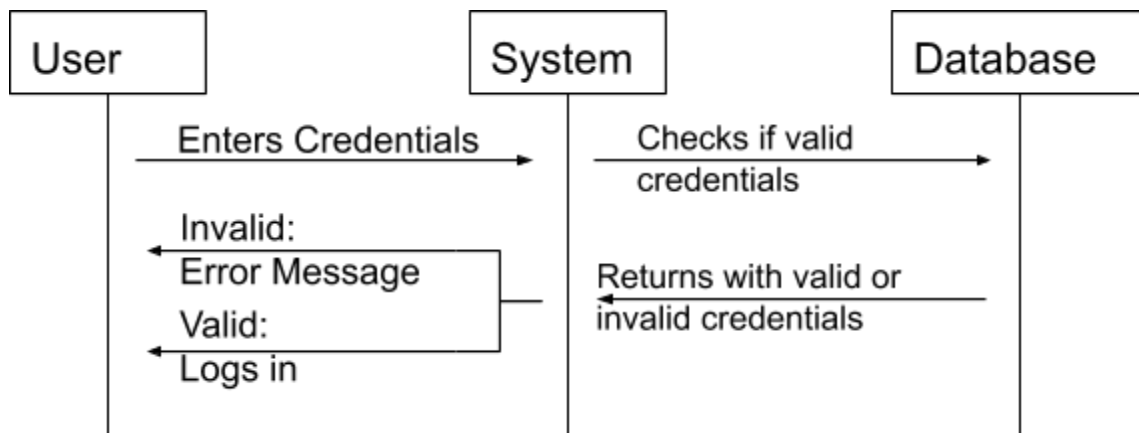
	<ul style="list-style-type: none"> <li>• Poll results are shown</li> </ul>
Postconditions	<ul style="list-style-type: none"> <li>• Poll results must be shown in session history</li> </ul>

Flow of Events for Main Success Scenario:

1. Instructor launches live session in class.
2. Instructor presses on poll button and asks a question as well as the options students are able to choose.
3. Poll is sent to server where its relayed to students
4. Student's results are stored in database and shown in feed.

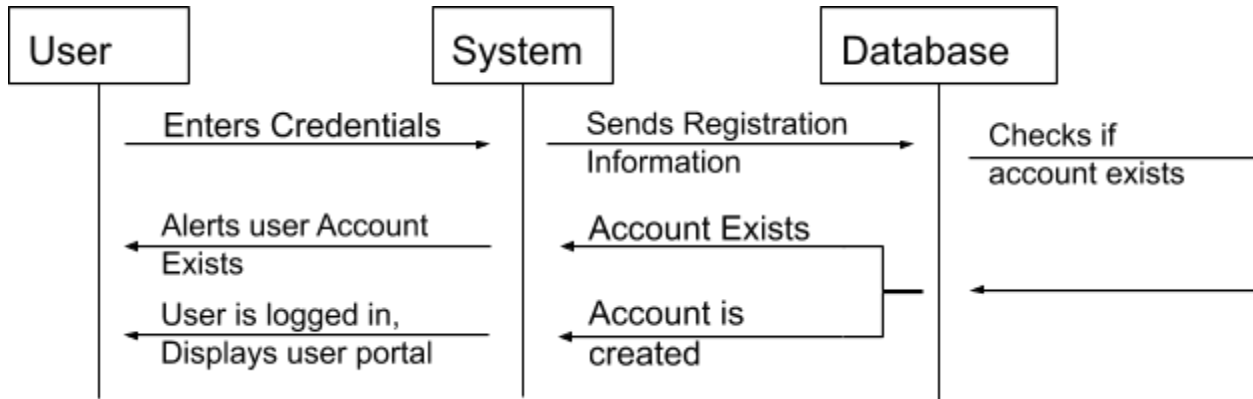
## 4.D Sequence Diagrams

**Login:**

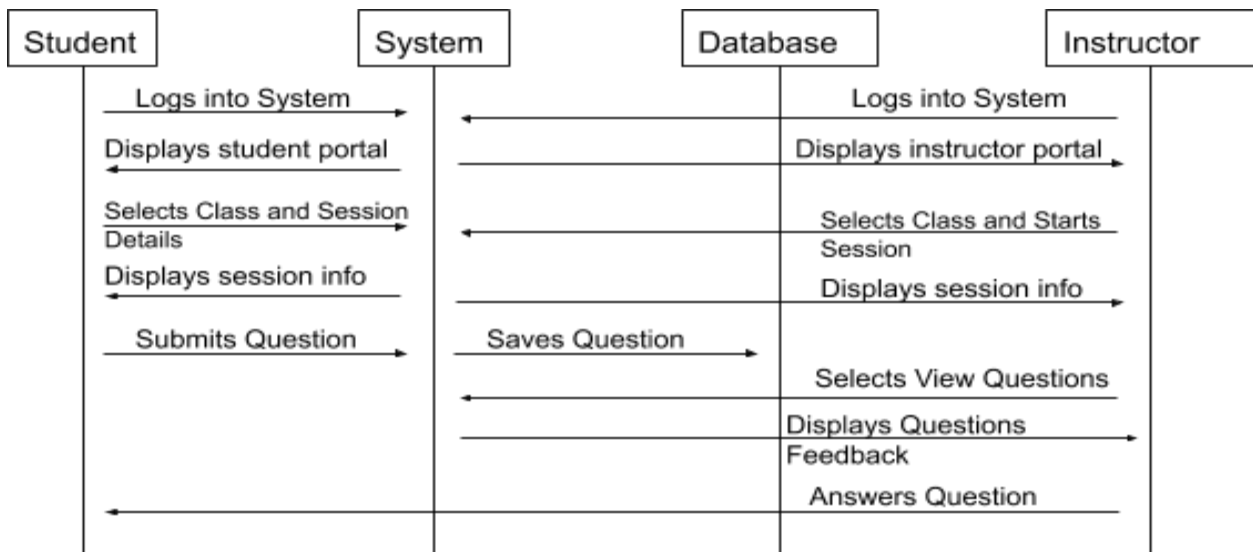


**Register:**

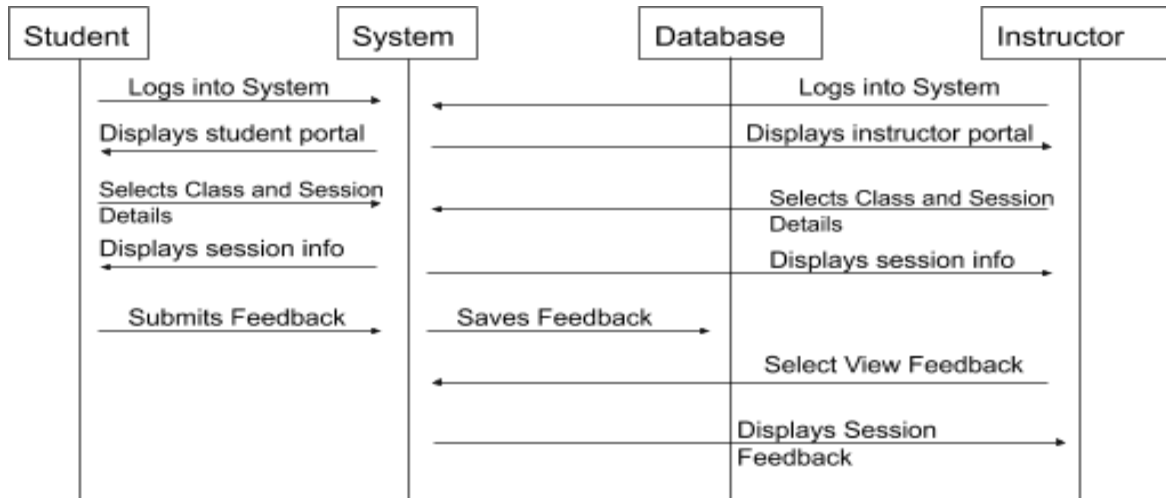




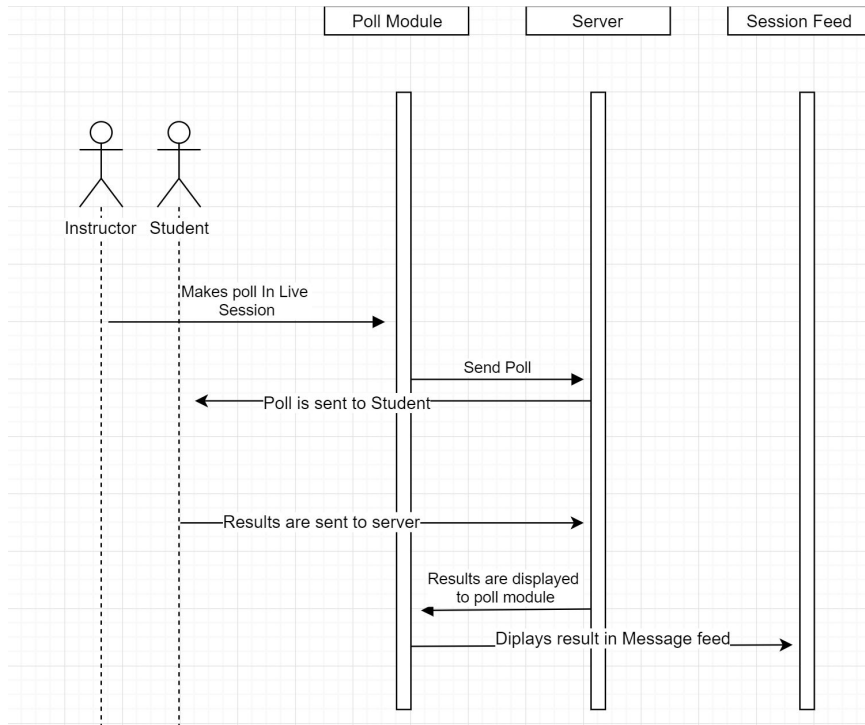
### Answers:



### Feedback:



**Polls:**



## 5. Effort Size Estimation

### Calculating UCP:

ACTORS	Description	Complexity	Weight
Students (participating)	Students interact with the application (and thus, the professors) using a text based system	Average	2
Instructors (initiating)	Instructors interact with the application using a GUI Based system	Complex	3
Universities & Schools	Interact with the application using a GUI based system to ensure that professors and students are in the correct classes	Complex	3
Database	The database is constantly in sync with a database which stores all of the saved data	Simple	1
Sockets	Sockets allow live communication between instructors and students during a session	Complex	3

$$UAW = 2 + 3 + 3 + 1 + 3 = 12$$

### Calculating UUCP:

Use Cases	Description	Category	Weight
-----------	-------------	----------	--------

UC-1: Register	Simple user interface. 4 steps to achieve main success scenario. Participating actors: Student, Instructor, and Database	Average	10
UC-2: Class Setup	Average User Interface. 3 Steps to achieve main success scenario. Participating actors: Instructor, Database, and University	Average	10
UC-3: Session Preparation	Average User Interface. 3 Steps to achieve main success scenario. Participating actors: Instructor, and Database	Average	10
UC-4 Attendance	Simple User Interface. 1 Step to achieve main success scenario. Participating actors: Student and Database	Simple	5
UC-5 Questions	Complex User Interface. 7 Steps to achieve main success scenario. Participating actors: Student, Database, Sockets, and Instructor	Complex	15
UC-6 Answers	Simple User Interface. 1 Step to achieve main success scenario	Simple	5
UC-7 Quizzes	Average User Interface. 4 steps to achieve main success scenario. Participating actors: Student, Instructors, Sockets, and Database	Complex	15
UC-8 Polls	Simple User Interface. 3 steps to achieve main success scenario. Participating: Instructor, Student, and database	Simple	5
UC-9 Messaging feed	Average User Interface. 3 steps to achieve main success scenario.	Complex	15

	Participating: Instructor, Student, Sockets, and database		
UC-10 Resources	Average User Interface. 3 steps to achieve main success scenario. Participating actors: Instructor, Students, and Database	Average	10
UC-11 Direct Message	Average User Interface. 3 Steps to achieve main success scenario. Participating actors: Student, Database, and Instructor	Average	10
UC- 12 Feedback	Complex User Interface. 5 Steps to achieve main success scenario. Participating actors: Student, Database, and Instructor	Average	10
UC-13 Session History	Complex User Interface. 7 Steps to achieve main success scenario. Participating actors: Student, Database, and Instructor	Complex	15
UC-14 Notifications	Simple User Interface. 3 Steps to achieve main success scenario. Participating actors: Student, Database, and Instructor	Average	10
UC-15 Database	Simple User Interface. 1 Step to achieve main success scenario. Participating actors: Student, Database, and Instructor	Simple	5
UC-16 Student/Instr uctor View	Simple User Interface. 2 steps to achieve main success scenario. Participating actors: Student, Database, and Instructor	Simple	5

$$UUCW = 10(7) + 5(5) + 15(4) = 155$$

## Calculating TCF:

Use Cases	Description	Weight	Score
T1	Distributed System Required	2.0	3
T2	Response Time Is Important	1.0	5
T3	End User Efficiency	1.0	5
T4	Complex Internal Processing Required	1.0	2
T5	Reusable Code Must Be a Focus	1.0	2
T6	Installation Ease	0.5	3
T7	Usability	0.5	4
T8	Cross-Platform Support	2.0	4
T9	Easy To Change	1.0	1
T10	Highly Concurrent	1.0	3
T11	Custom Security	1.0	1
T12	Dependence on Third Party Code	1.0	1
T13	User Training	1.0	1

$$TCF = 0.6 + (6+5+5+2+2+3+4+4+1+3+1+1+1)/100 = 0.98$$

## Calculating UCP:

$$UCP = (UUCW + UAW) \times TCF$$

$$UCP = (155+12) \times 0.98 = 163.66$$

## Calculating Duration:

Duration =  $163.66 * 28 = 4582.48$  Hours

## 6. Domain Analysis

---

### A. Domain Model

#### 1. Concept Definitions (D-doing; K-knowing; N-neither)

Responsibility	Type	Concept
R1: Allow new user to create an account and to choose between instructor and student	D	Account Controller
R2: Fetch Data from server based on User Profile	D	Communication Controller
R3: Display User's(Instructor or Student) previous classes, sessions, and upcoming sessions	D	(Student/Instructor) Interface
R4: Once new session has been initiated by instructor, start the peer to peer Bluetooth range to check the students' attendance	D	Peer to Peer module
R5: Server will distribute the messages from instructor to all clients(registered students) in the database	N	Real-time Interactor
R6: When instructor starts a session, all students within peer to peer range (Bluetooth) of instructor's device will be	K	Peer to Peer Attendance module

marked as attended and a timestamp of attendance will be marked		
R7: Data is sent to database to record timestamp of attendance in real time.	K	Communication Controller
R8: Students are able to type out questions to professor	D	Question Module
R9: Questions asked to instructor by students are sent to server and transmitted to instructor instantly	D	Real-time Interactor
R10: Instructor is able to give quiz questions	D	Quiz module
R11: Quiz questions asked by instructor are sent to server and transmitted to students instantly using sockets API	D	Real-time Interactor
R12: Students are able to answer, and the answers stored directly to database	D	Communication Controller
R13: When student leaves class and out of instructor Bluetooth range, the app will record timestamp of departure.	D	Peer to Peer module
R14: Data is sent to database to record timestamp of attendance in real time.	K	Communication Controller
R15: Once instructor ends session all data from session is uploaded to database along with student feedback.	K	Communication Controller
R16: Student's display of session history, upcoming sessions, and classes will be updated	K	Communication Controller
R17: Allow students to direct message instructors.	D	Communication Controller
R18: Instructors are able to start a poll for students to give their preferences	D	Real-time Interactor
R19: Instructors are able to send notifications for students to see	D	Communication Controller
R20: Instructors are able to audio record answers for student questions live.	D	Real-time Interactor



R21: Students are able to up vote any comments, questions, or answers in the feed.	D	Real-time Interactor
R22: Instructors are able to upload any resources relevant to class material.	D	Communication Controller

## 2. Association Definitions

Concept pair	Association Description	Association Name
Account Controller<> Communication controller	Login request made by student or Instructor sent and received through server	Login
User<>Communication controller	Fetch User's data from database	Student/Class Data
Communication Controller<>Interface	Display interface of classes, session history, and upcoming sessions	Class Display
Instructor Interface<>Communication Controller	Allow an instructor to create session	Session initiation
Communication Controller <>Student Interface	Display active sessions to all students	Student Session begin
Peer to Peer Module<>Attendance module	Timestamp of the students' start of attendance is recorded.	Attendance
Attendance module<>Communication Controller	Time stamp of entrance is sent to database	Attendance correspondence data
Instructor Interface<> Quiz Module	Instructor will click on "New Quiz" and ask questions to class	Quizzes

Quiz Module<>Communicator Controller	Quiz questions will be sent all clients in server using Sockets API	Quiz distribution
Student Interface <> Quiz Module	Students will answer all questions	Recording Answers
Quiz Module <> Communication Controller	All answers will be sent to database	Saving Results
Student Interface<>Question Module	Students are able to type out questions to professor	Student Questions
Question Module<>Communication Controller	Question is sent to server using Sockets API	Saving Student questions
Communication Controller <> Instructor Interface	From server question is sent to instructor	Instructor receives question
Instructor Interface<>Communication Controller	Instructor ends session and send message to server	Session end
Communication Controller <>Student Interface	Student will have session appear as ended in session history	Session history

### 3. Attribute Definitions

Concept	Attribute	Description
Account Controller	<ol style="list-style-type: none"> <li>1. <u>UserType</u></li> <li>2. <u>Password</u></li> <li>3. <u>Name</u></li> <li>4. <u>Email</u></li> </ol>	<ol style="list-style-type: none"> <li>1. Associated account with either student or instructor</li> <li>2. Password of user account</li> <li>3. Full name of account owner</li> <li>4. Email of the account holder</li> </ol>
Communication Controller	<ol style="list-style-type: none"> <li>1. <u>createData</u></li> <li>2. <u>readData</u></li> <li>3. <u>updateData</u></li> <li>4. <u>deleteData</u></li> </ol>	<ol style="list-style-type: none"> <li>1. Post new data to server for database to store.</li> <li>2. Get data queries from database through server API endpoints</li> <li>3. Post and replace old data with new data</li> <li>4. Find and delete data from database</li> </ol>
Student Interface	<ol style="list-style-type: none"> <li>1. <u>sessionHistory</u></li> <li>2. <u>Classes</u></li> <li>3. <u>Sessions</u></li> </ol>	<ol style="list-style-type: none"> <li>1. Recorded history from database of all previous sessions</li> <li>2. Fetched data of all the student's current classes</li> <li>3. Fetched data of all the student's past sessions or any current live sessions</li> </ol>

Peer to Peer module	<ol style="list-style-type: none"> <li>1. <u>Instructor Location</u></li> <li>2. <u>Student Location</u></li> </ol>	<ol style="list-style-type: none"> <li>1. Location of the instructor's phone</li> <li>2. Distance allotted for the student's phone to be registered as present by the instructor. This depends on the Bluetooth range for the instructor's phone.</li> <li>3. Current Location of the student</li> </ol>
Real-time Interactor	<ol style="list-style-type: none"> <li>1. <u>Questions</u></li> <li>2. <u>Quizzes</u></li> <li>3. <u>Server Call</u></li> </ol>	<ol style="list-style-type: none"> <li>1. Direct questions asked by students to instructor during session</li> <li>2. Quizzes made by teacher for students in sessions</li> <li>3. Socket communication between both instructor and student devices</li> </ol>
Attendance Module	<ol style="list-style-type: none"> <li>1. <u>Present</u></li> <li>2. <u>Absent</u></li> </ol>	<ol style="list-style-type: none"> <li>1. Status feedback if the student is present in class</li> <li>2. Status feedback for if the student is absent from class</li> </ol>
Instructor Interface	<ol style="list-style-type: none"> <li>1. <u>Classes</u></li> <li>2. <u>Create session</u></li> <li>3. <u>Session History</u></li> <li>4. <u>Add Class</u></li> </ol>	<ol style="list-style-type: none"> <li>1. Classes that instructor is teaching</li> <li>2. Ability to create new lecture session</li> <li>3. Log of all previous sessions retrieved from database</li> <li>4. Ability to create new class and add a roster of students</li> </ol>
Quiz Module	<ol style="list-style-type: none"> <li>1. <u>Questions</u></li> <li>2. <u>Answer choices</u></li> <li>3. <u>Correct answer</u></li> </ol>	<ol style="list-style-type: none"> <li>1. Real-time question presented to class from instructor</li> <li>2. Multiple choice options for students to choose from</li> <li>3. The indicated correct answer by the instructor</li> </ol>

## 4. Traceability Matrix

R/UCs	UC 1	UC 2	UC 3	UC 4	UC 5	UC 6	UC 7	UC 8	UC 9	Uc 10	UC 11	UC 12	UC 13	UC 14	UC 15	UC 16
R1	X															X
R2		X													X	X
R3		X											X			X
R4			X	X												
R5						X			X					X	X	
R6				X											X	
R7				X											X	
R8					X											
R9					X											
R10			X				X									
R11			X			X	X									
R12						X									X	
R13				X												
R14				X												
R15												X	X	X	X	
R16														X	X	X
R17											X					
R18								X								
R19														X	X	
R20																

R21									X						
R22										X			X		X

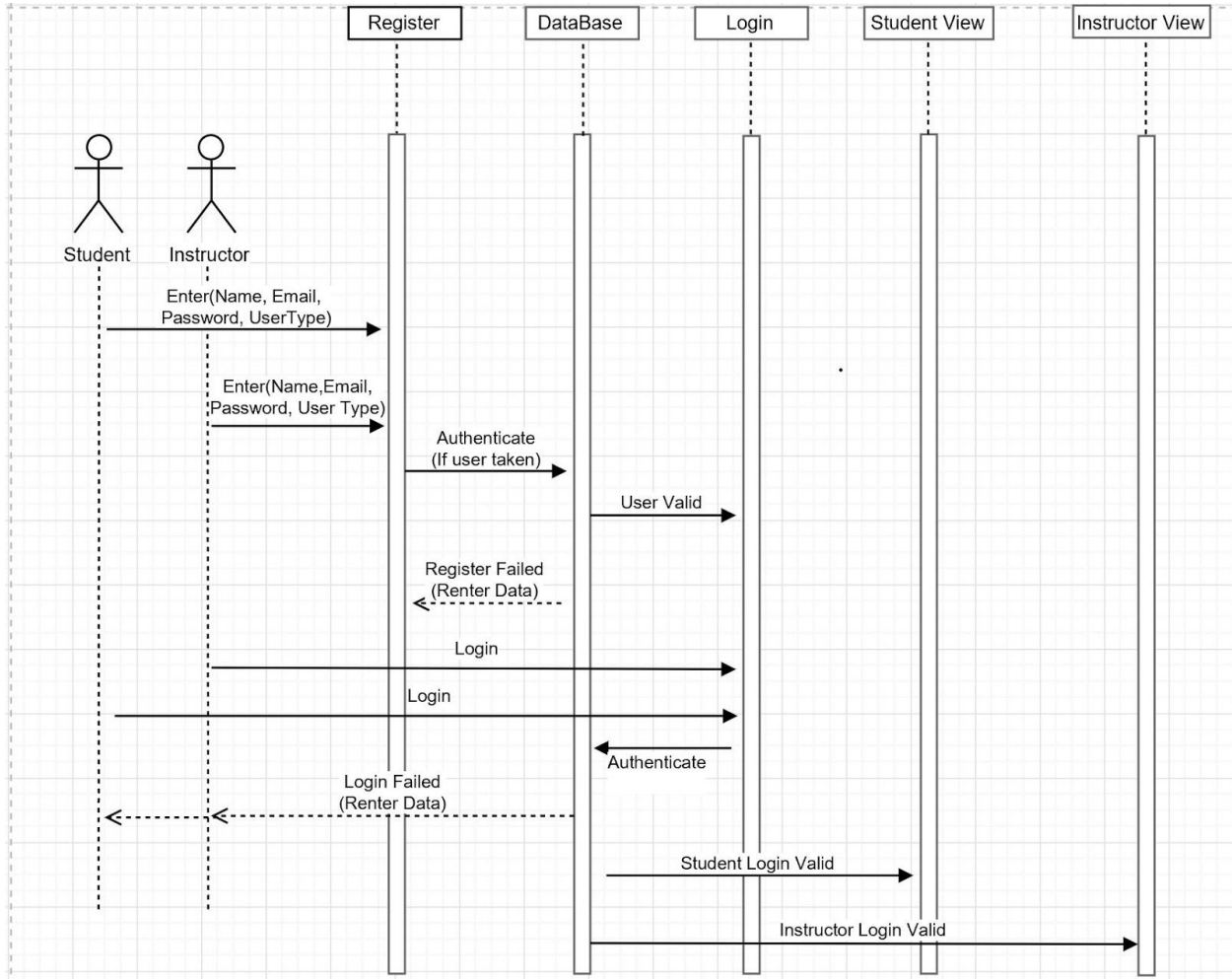
## B. System Operation Contracts

- **Login/Signup**
- **Pre-Conditions**
  - The system database is running live
  - The system displays login/sign up interface with email and password input
- Post-Conditions
  - The system verifies the user's email and password/ stores the text input of a new user
- **Class Setup**
  - Pre-Conditions
    - The system database is running live
    - User is logged in with instructor account
  - Post-Conditions
    - The system creates class with appropriate information and roster
    - All students on roster automatically get enrolled into class through registered email
- **Attendance**
  - Pre-Conditions
    - Session is running live on instructor phone
  - Post-Conditions
    - Student enters into Bluetooth radius and is marked as attended
    - If a student never enters into Bluetooth radius for the entire duration of the session, then they are marked as absent.
- **Answers**
  - Pre-Conditions
    - The system database is running live
    - User is signed in with student account
    - User is enrolled in class
    - Class session is active
    - Student asks a question live
  - Post-Conditions
    - Text or Audio Answer is submitted by the instructor through server and can be seen on the feed by instructor
- **Feedback**

- Pre-Conditions
  - The system database is running live
  - Class session had been active and concluded
  - Student is enrolled in class
- Post-Conditions
  - Feedback is submitted through server and instructor has access to feedback to improve session quality

# 7. Interaction Diagrams

## Login/Register



### UI Description:

User is allowed to login to an existing account or create a new account using their Name, Email, a Password, and their UserType (instructor or student).

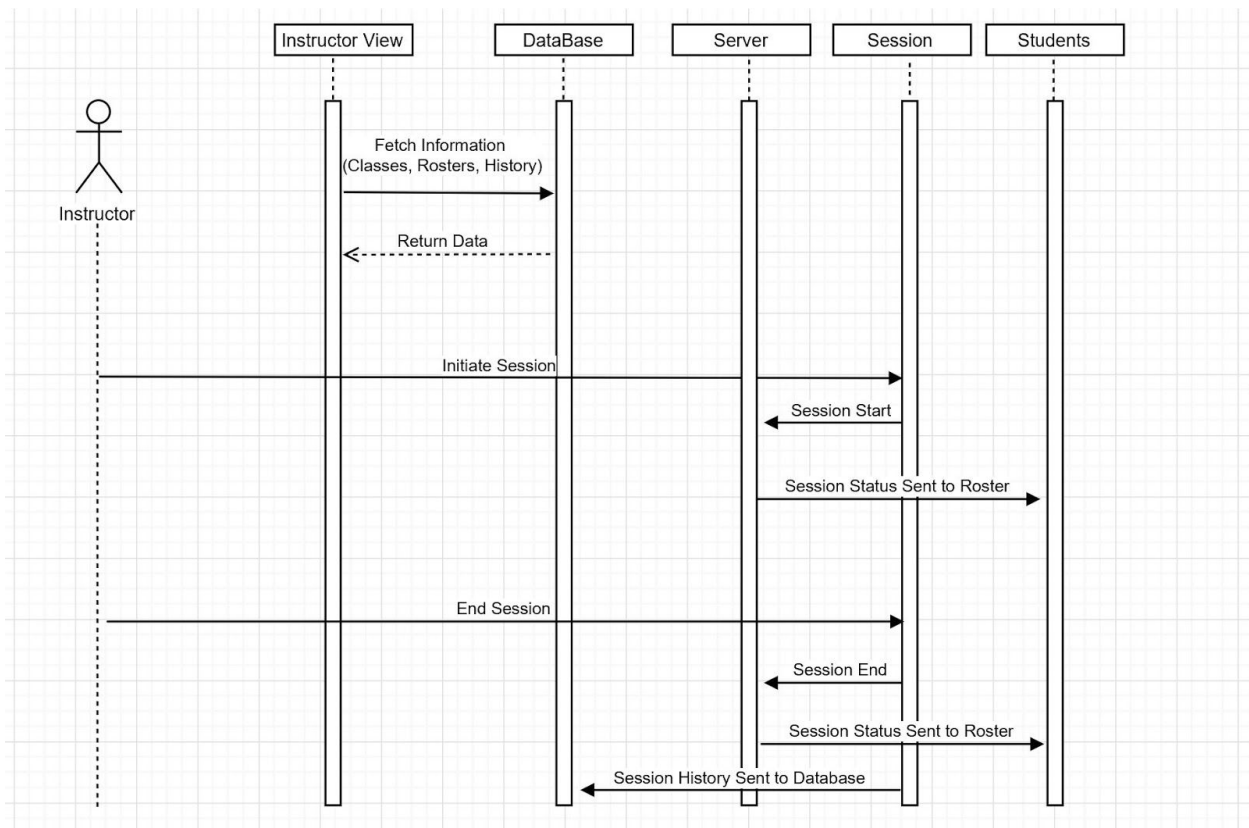
### UML Description:

In order to use the application, an account must first be created, with a selection of either a student account or instructor account. The information used to register is then sent to the server for authentication. Should there be an error with the user's entry, either while logging in or registering,



the user will be prompted to correct the mistake and continue. Otherwise, the user's account is created if registering and logged in to ClassHub. The model used for this use case is the Publisher-Subscriber design pattern. The user is the subscriber in the registration use-case, whereas the publisher is the application itself, either alerting the user of issues with their entry or logging them in. The login, however, follows the Proxy design pattern. If the user is an instructor, the user has access to functionality such as creating a class or starting sessions, whereas if the user is a student, the user does not have access to that functionality.

## Class Setup



### UI description:

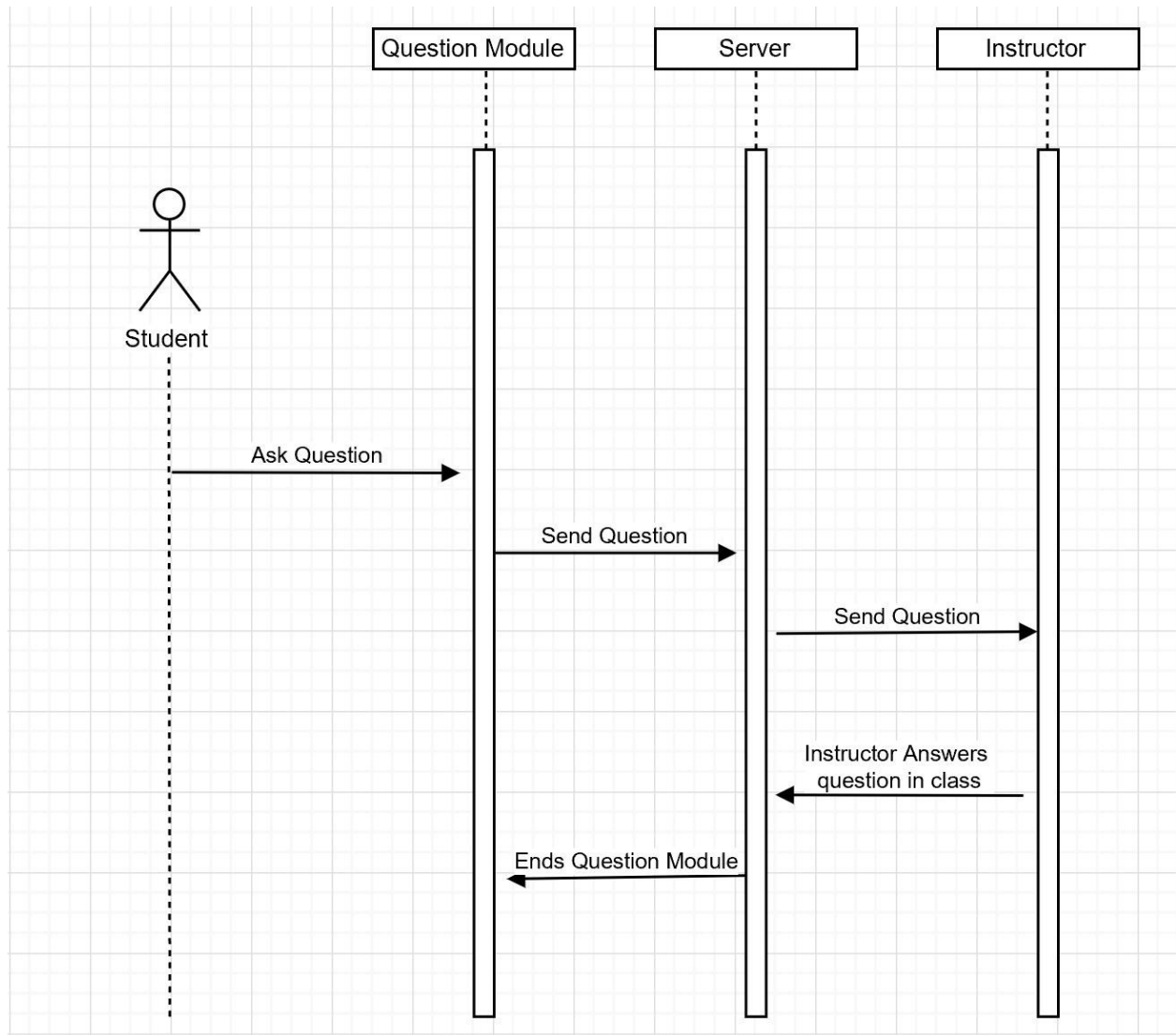
The instructor will be able to view classes, rosters, and history of sessions. Instructor will have the option to start and end the session and students will be notified of the session status.

### UML description:

The data (classes, rosters, and history of sessions) for the instructor will be fetched from the database. Once the instructor starts a session, the session status will be sent to server and then will be sent to all students enrolled in the class. When the instructor end the session, the session status will be updated and the history of the session will be sent to the server. The session history can be

viewed by both students and instructors. Class setup follows the Publisher-Subscriber design pattern. The Publisher is the server, which creates the new class after an instructor requests it, and adds the class to the list of classes of all users, in which the users are the Subscribers.

## Questions



### UI Description:

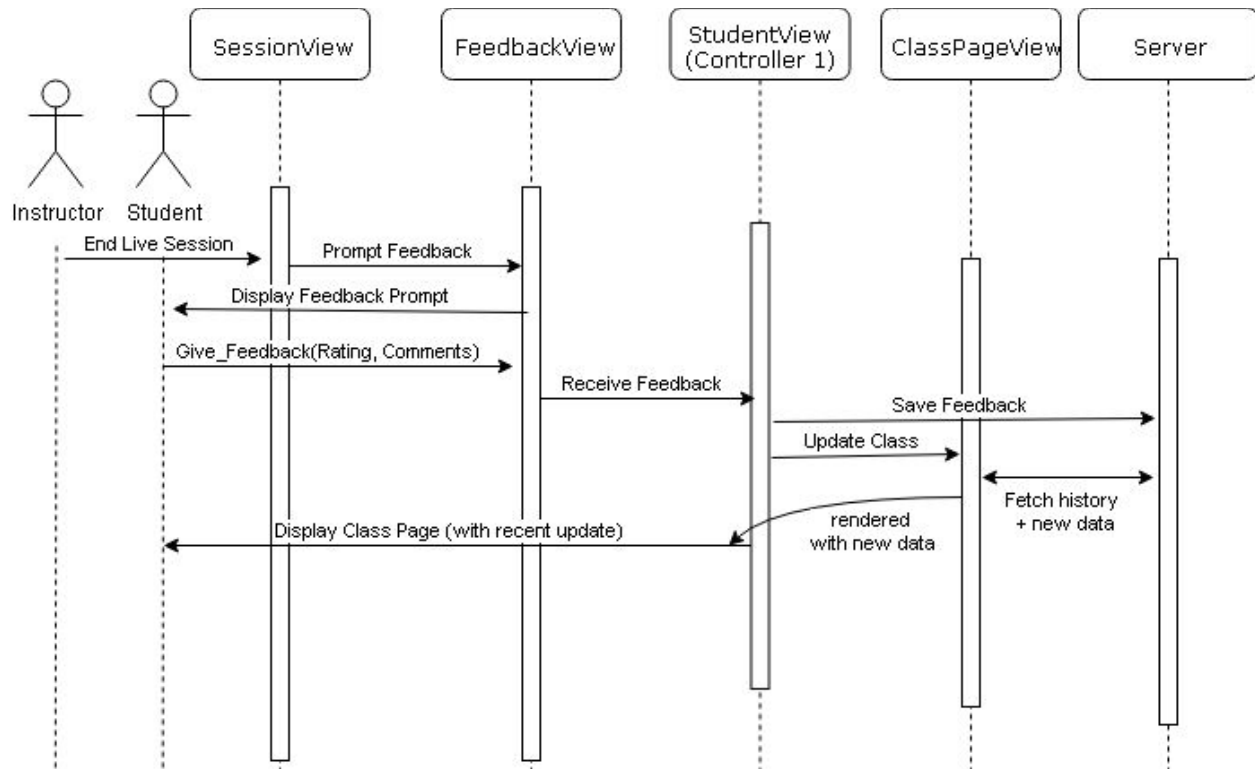
A chat window which allow students to ask questions and send them and view the other students questions. In the instructor's side, all questions asked by students during the session will appear.

### UML Description:

During class, ClassHub makes it easy for students to ask the instructor questions, even if the user is shy or too far away. An active session will pull up as a chat feed, which all students in the class as well as the instructor have access to. Students can submit any questions they have about the

current lecture material on the chat feed, upon which the instructor can answer the question in class. This Questions feature follows the Publisher-Subscriber design pattern as well, where the server once again serves as the Publisher, and the students and instructor are the Subscribers.

## Feedback



### UI Description:

Instructor ends the live session. Student receives feedback prompt asking for rating out of 5 stars and a brief comment description about the lecture. The student gives their input and press confirm which leads them back to the class page to which the session belongs to. They can view the latest session history there.

### UML Description:

Instructor ends live session. SessionView calls Feedback View in Student's view. Feedback View asks for feedback parameters: rating and comments. Student gives appropriate information and presses confirm which informs Controller (StudentView) to save feedback to server with given parameters. Controller then performs an update on the current class which fetches new information from the server. Controller then displays the current class's new updated page. This use case, similar to some of the previous use cases, also follows the Publisher-Subscriber design pattern. Once the user ends the live session, the students are prompted to submit a rating and feedback for the lecture that just ended. Once the students submit the ratings, the feedback is aggregated into the instructor's session history. The Publisher is, once more, the server, which takes the information and performs all necessary functions, before sending it back to be displayed to the Subscriber, the instructor.

# 8. Class Diagram and Interface Specifications

## Class Diagrams

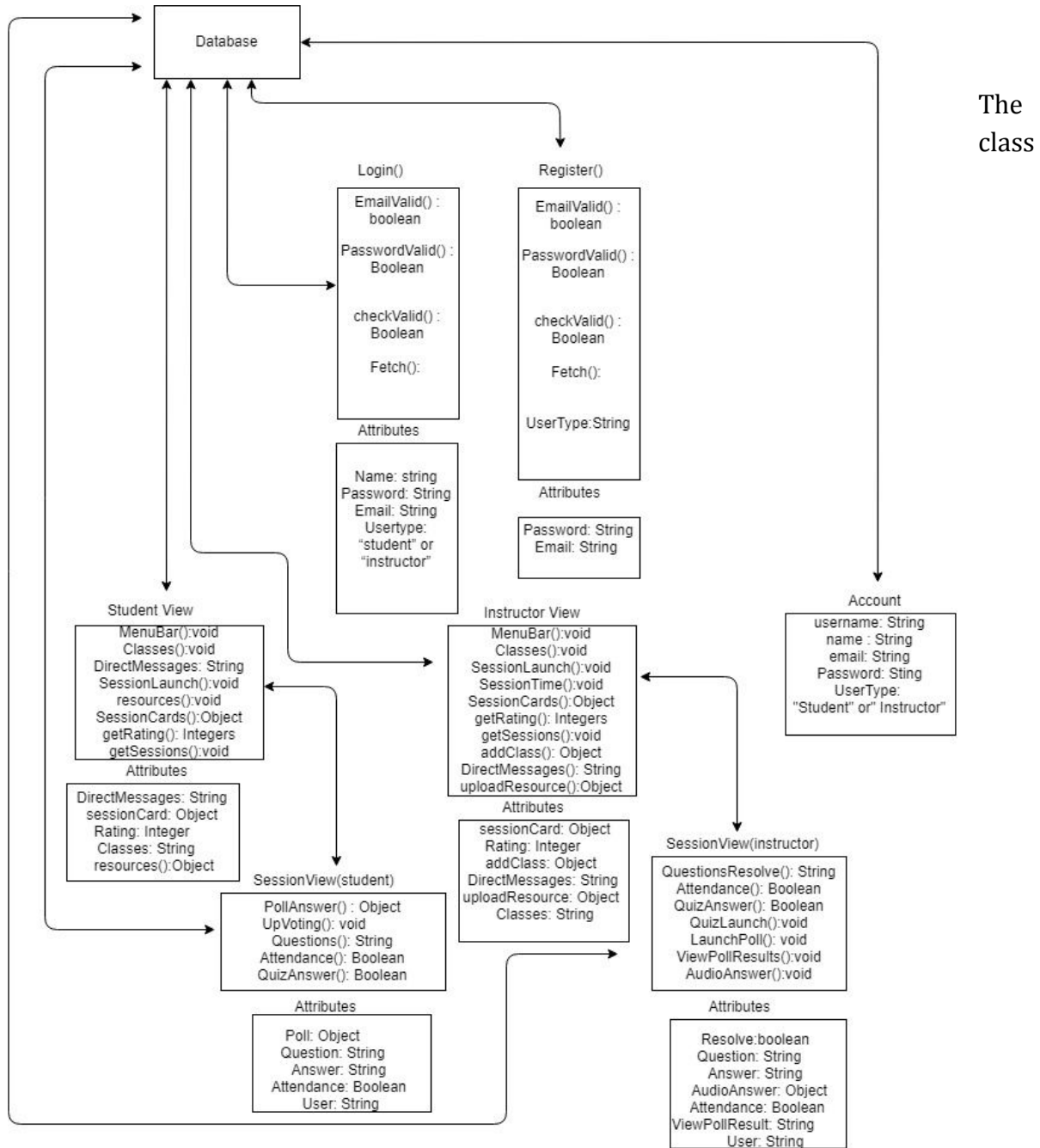


diagram displays exactly how the entire system is connected. It begins through a login or register. Once logged in then you will be taken to either student or instructor view. In this view the student or instructor is able to view all their classes, session history, and upcoming sessions. Instructors are also able to add classes, resources, and launch sessions. Everything is connected to the database because it is responsible for fetching and displaying all the data.

## Data Types and Operations:

---

These are the list of the main classes we utilize.

### UserPortal Class

#### **Login():void->**

Allows the user to effectively call on the server to login with inputted information

### **Subclasses (Operations)**

#### EmailValid() : boolean

*Checks if the email is valid*

*Checks if email contains illegal special characters or not in proper format*

#### PasswordValid() : boolean

*Checks if the password is valid*

#### checkValid() : boolean

*Checks if name is <=0*

*Checks if password is longer than 8*

*Checks if passwords match*

#### Fetch():

Checks if database console accepts the email and password and proceeds to interface

### **Attributes(for the above methods):**

Password: String

Email: String

### **Register():void->**

Allows new users to create a login and send their login information to database

### **Subclasses(Operations)**

EmailValid(): boolean

*Checks if email is valid*

*Checks if email contains illegal special characters or not in proper format*

PasswordValid() : boolean

*Checks if the password is valid*

checkValid() : boolean

*Checks if name is <=0*

*Checks if password is longer than 8*

*Checks if passwords match*

UserType() : String

*Checks if user is going to be "instructor" or "student"*

Fetch():void

*Will send data to database, If console accepts data then user will be created and stored*

### **Attributes(for the above methods):**

Name: string

Password: String

Email: String

Usertype: "student" or "instructor": String

## **StudentView Class**

**MenuBar()->**

**Classes()->**

*Will display all classes the student is taking*

**SessionLaunch()->** *Will take user to sessionView*

### **Subclasses (operations)**

SessionCards():object

*Will display all upcoming and previous sessions in card looking format*

getRatings():integer

*Will pull allow users to view the ratings of every session of the professor*

getSessions():void

*Will fetch data from server to pull history of sessions and upcoming sessions*

resources():void

*Will allow students to view and download resources uploaded by the professor*

DirectMessages():String

*Will allow the student to send private messages to professor*

### **Attributes(for the above methods):**

sessionCard: Object

Rating: Integer

Classes: String

resources: Object

DirectMessages: String

## SessionView class (FOR STUDENT)

SessionLaunch()->

### Subclasses (operations)

Questions(): String

*Will allowing user to input question*

Attendance(): Boolean

*Will return true if user is in Instructor range when session is active*

QuizAnswer(): Boolean

*If student chooses correct answer then returns true*

PollAnswer(): Object

*Popup of poll will appear and the answer choice indicated by student will be recorded*

Upvoting(): void

*Each time a student presses to up vote the count of upvotes will appear on the display for both student and instructor*

### Attributes(for the above methods):

Question: String

Answer: String

Attendance: Boolean

User: String

PollAnswer: Object



## **InstructorView Class**

**MenuBar()->**

**Classes()->**

*Will display all classes the instructor is teaching*

**SessionLaunch()->**

*Will allow instructor to create a session*

**SessionTime()->**

*Indicate the start and end time of the session. Will send data to the server to students*

### **Subclasses (Operations)**

SessionCards()

*Will display all upcoming and previous sessions in card looking format*

getRatings()

*Will display the ratings of previous session in the session cards*

getSessions()

*Will fetch data from server to pull history of sessions and upcoming sessions*

DirectMessages():String

*Will allow user to answer and send direct messages for students*

uploadResource(): Object

*Instructor will be able to upload resources from his or hers local storage*

### **Attributes(for the above methods):**

sessionCard: Object

Rating: Integer

Classes: String

uploadResources: Object

DirectMessages: String

## SessionView class (FOR INSTRUCTOR)

### Subclasses (Operations)

QuestionResolve(): boolean

*Will return true if question a student asked is resolved*

Attendance():

*Will run in background recording all attending users*

QuizLaunch()

*Will allow instructor to create a quiz question and ask students the answer*

QuizAnswer()

*Instructor will indicate the correct choice for the quiz*

LaunchPoll(): void

*Instructor will be able to launch a poll and write the questions and choices*

ViewPollResults():void

*Poll results will be displayed on the feed*

AudioAnswer():void

*Instructor is able to record his answer and post his audio recording on the feed*

### Attributes(for the above methods):

Question: String

Answer: String

Attendance: Boolean

User: String

Resolve: Boolean

AudioAnswer: Object

ViewPollResult: String

## Traceability Matrix:

Domain Concept	Derived Class	Explanation
Account Controller	User Portal Class	The concept required that a new user be able to create a new account either a student or an instructor and that existing users be able to log in to their existing accounts. The User Portal class provided an interface and a backend to act as such account controller.
Communication Controller (Student)	StudentView Class	<p>The concept required several functions:</p> <ul style="list-style-type: none"> <li>- Fetch Data from the server based on the user profile.</li> <li>- Data is sent to database to record timestamp of attendance in real time.</li> <li>- Students are able to answer, and the answers stored directly to database.</li> <li>- Student's display of session history, upcoming sessions, and classes will be updated</li> </ul> <p>Since all of these functions were integral to the communication of the student with the app and its data, it felt best to organize it into a separate controller specific to the communication between the student and the app.</p>
Communication Controller (Instructor)	InstructorView Class	<p>The concept required several functions:</p> <ul style="list-style-type: none"> <li>- Fetch Data from server based on User Profile</li> <li>- Data is sent to database to record timestamp of attendance in real time.</li> <li>- Once instructor ends session all data from session is uploaded to database along with student feedback.</li> </ul> <p>Similar to the studentview class, this class acts as the communication controller between the instructor and the app. It differs from the studentview where in the last point, the student only acts as the data input (feedback) which is then organized and rendered only for the instructor, thus its placement in the instructorview only.</p>
Student /Instructor Interface	StudentView Instructor View	<p>Since we are using react technology which is built upon front-end languages of HTML, CSS, and JS. It was automatic that the classes themselves can act as the user interfaces as well as communicate with backend as needed.</p> <p>The direct messages client is a new feature added to add more communication methods between student and instructor</p> <ul style="list-style-type: none"> <li>- DirectMessages()- Student view/ instructor view</li> </ul> <p>Allows for the student and instructor to communicate with each other. Once a student sends a message to the instructor, the message gets sent to the database and shows to the respective client.</p>
Google's Nearby API	SessionView (Instructor &	Google Nearby uses a combination of Bluetooth, Bluetooth Low Energy, Wi-Fi and near-ultrasonic audio to communicate a

(Attendance) module	Student)	<p>unique-in-time pairing code between devices that are connected to the internet, but not necessarily on the same network.</p> <ul style="list-style-type: none"> <li>- Live session are active using real time sockets. Attendance entrance and exit events are able to be marked in the database.</li> <li>- Nearby API has been tested and works undoubtedly</li> <li>- Instructor launches live session in class.</li> <li>- Students who are logged in to the internet are able to be detected by instructor</li> <li>- Students are able to gain access to the session and participate in class.</li> <li>- Once session has ended they all data is sent to database</li> </ul>
Question Module	SessionView (Instructor & Student)	<p>It was discussed best that the question module split into two appropriate components such that the student and the instructor have their own piece and one cannot edit the other. React-Native allows for this very easily where each big class can have any number of components and share data flexibly. Therefore, the question module was split into:</p> <ul style="list-style-type: none"> <li>- Questions() - students' session view</li> <li>- QuestionsResolve() - instructors' session view</li> </ul>
Quiz Module	SessionView (Instructor & Student)	<p>Similar to the questions module, the quiz module has its respective components in each of the sessionView classes of the student and instructor. The quiz module is split into:</p> <ul style="list-style-type: none"> <li>- QuizAnswer() - students' session view</li> <li>- QuizLaunch() - instructors' session view</li> <li>- QuizAnswer() - instructors' sessionview</li> </ul> <p>The QuizAnswer from the student side will be their answer to the quiz. The QuizLaunch from the instructor side will render the instructor's quiz to the students through the server using sockets. The QuizAnswer from the instructor's side will be used to check each student's answer with the correct answer when the instructor finishes the quiz.</p>
Poll Module	Session view (Instructor & Student)	<p>Similar to the quiz module the poll module is split in to</p> <ul style="list-style-type: none"> <li>- Launch poll()- Instructor view</li> <li>- ViewPollResult()-Both student and instructor view</li> <li>- PollAnswer() - students' view</li> </ul> <p>This module allows for instructors and students to communicate polls with each other via the instructor asking the poll and the student answering them. Communication between student and instructor is established with sockets.</p>

**\*\*Note about module splitting:**

The main reason these modules were chosen to be split as such was to avoid overlap of actions that could cause the component to bug out and/or render in an ugly fashion as everyone would be directing their actions on the same main module class. If the component is split such that the students and instructors have something of their own to edit then the

communication between the two can remain smooth and lead way to easier debugging in the future. Rather than looking at the same class, the debugging process is made easier automatically where the compiler will indicate whether the quiz/question is bugging on the student's side or the instructor's.

## Design Patterns

---

Overall the design patterns for certain components were very similar to each other. For some of the functions such as quizzes and polls, the components were nearly identical but the handling of the results differed. The student view and Instructor mirrored each other, but the functions for each differed respectfully.

To improve the design patterns you can separate the view for students and instructors and make each one look different. One can add more individual aesthetics for each view to make each more appealing for its respective viewer.

## Object Constraint Language

---

### 1. Student/Instructor Interface

Student/Instructor Interface	
Operations	
<u>SessionCards()</u> - Make session cards to store session history	
<u>getRatings()</u> - Retrieve ratings	
<u>getSessions()</u> - Retrieve previous session history and logistics	
<u>resources()</u> - Retrieve resource	
<u>DirectMessages()</u> - Retrieve previous messages	
Invariants	Retrieving user Data from the database
Preconditions	Student/ Instructor has logged into their accounts
Postconditions	All classes, session history, upcoming sessions, resources and direct messages are retrieved from database

## 2. Session view(Student)

Session view(Student) <u>Questions()</u> <u>Attendance()</u> <u>QuizAnswer()</u> <u>PollAnswer()</u> <u>Upvoting()</u>	
Invariants	Internet Connection between student and instructor through sockets.
Preconditions	Students are in range of professor and professor has launched session. Students have joined the session
Postconditions	Students are able send questions, answer quizzes, and upvote questions.

## 2. Session view(Instructor)

Session view(Student) <u>QuestionResolve()</u> <u>Attendace()</u> <u>QuizLaunch()</u> <u>LaunchPoll()</u> <u>QuizAnswer()</u> <u>ViewPollResults()</u> <u>AudioAnswer()</u>	
Invariants	Internet Connection between student and instructor through sockets.
Preconditions	Professor launch session and decides how long the session will be. All attending students will be able to join session
Postconditions	Students are able send questions, answer quizzes, and upvote questions.

### 3. Account Controller

Account Controller <u>EmailValid()</u> <u>PasswordValid()</u> <u>checkValid()</u> <u>fetch()</u> <u>UserType()</u>	
Invariants	Logging in and authenticating user
Preconditions	User or Instructor has download and launched the app and is beginning to make a new account or logging in to a previous account
Postconditions	User's data is sent to database to be verified and authenticated.

## 9. System Architecture and System Design

---

### Architectural Styles:

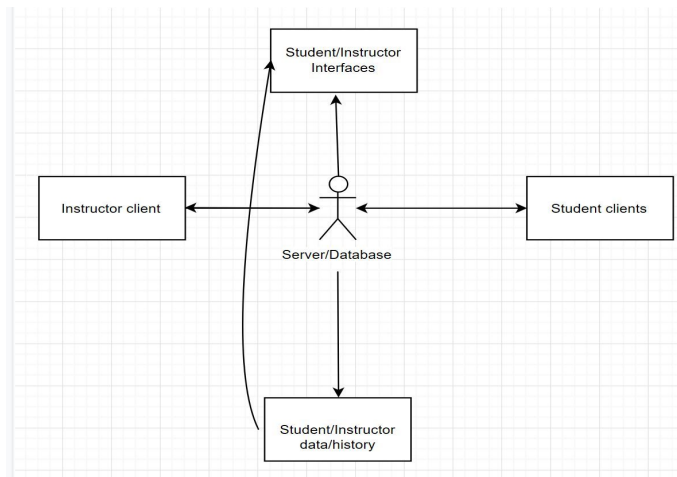
The overall style of our system is component/state based methodology. Our overall system will be comprised of a cluster of students' mobile phones connected to an instructor's mobile phone via an app. We are working on the implementation of sockets through our server. In which through networks we are able to communicate a message from an instructor's mobile to our running server and then that message is relayed to all the clients of that server.

For our app we need a functional database to hold all the data collected of users and their individual history. The architecture we implemented is NoSQL in which we send and pull information such as login credentials, registered user information, and history to and from the database. Currently, our database is running and hosted on Amazon Web services.

- The mobile app is programmed through React Native. This is a cross platform architecture comprised of AJAX (asynchronous JavaScript and XML) and CSS. The advantage of this

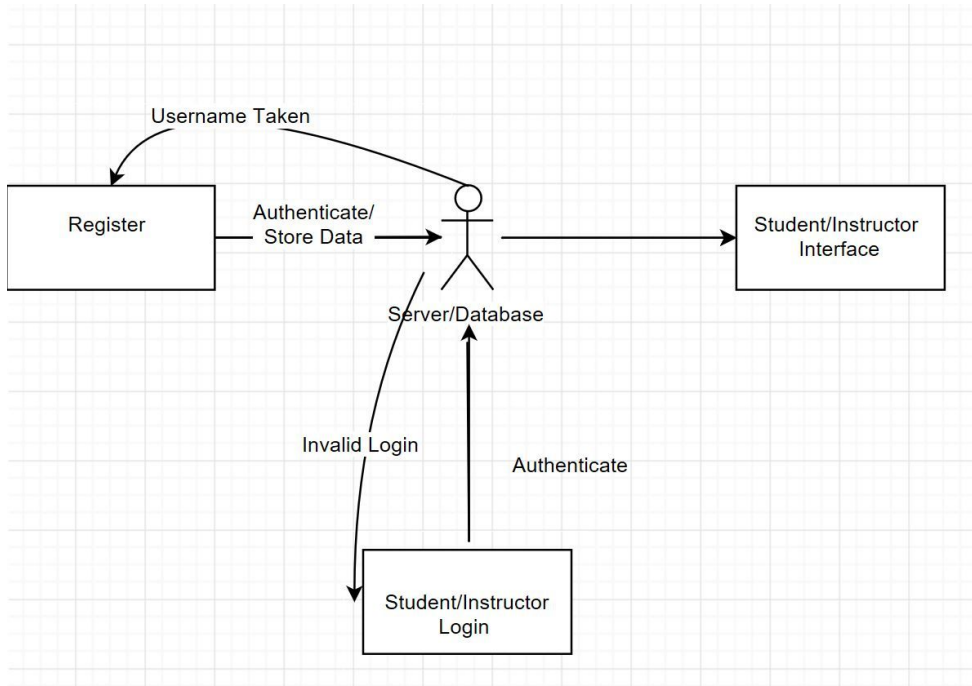
architecture allows for students who have android or IOS to download and use the app. React Native allows the programmer to easily create UI designs and integrate them by adding functionality to components and their states. An application called expo is used to continuously test the app on our own devices as we code.

## Identifying Subsystems:

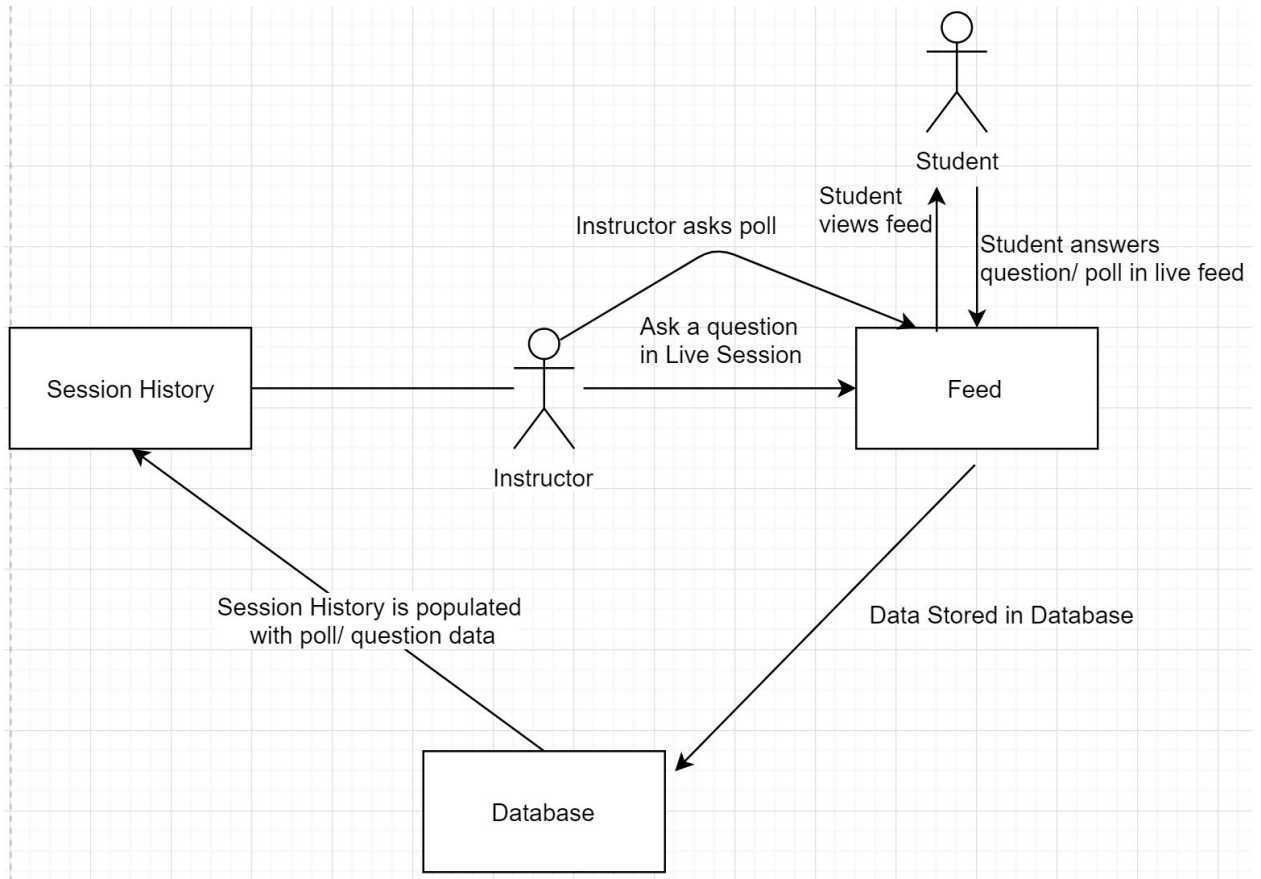


This subsystem is used to depict the essence of the server. It allows us to create sockets and send messages to and from the instructor and student in real time. It is the first thing established when the student or instructor logs in. The database will search for the user in its itineraries and sends the user's data to the interface. The history or previous sessions and personal data will also be pulled from the database to the UI interface to accustom each user who has logged in.





This subsystem specifies our login and register protocol. Each login whether student or instructor has to be authenticated by the database. If valid then the user's next view will be their specified interface along with the rest of the data that was stored in their itinerary. For the register client, the database is first checked if the information sent to the database is valid or previously taken. If it has been previously registered, then the database will ask user to register again. However, if the email is valid then the user will be made an itinerary and taken to the interface.



This subsystem displays a rough picture of the path for the quiz question and poll system. In the diagram it is known before that all data sent from instructor to student and vice versa is first sent to the server and is programmed through sockets to deliver the data accordingly. For instance the instructor asks a question or poll question, the server is the one that sends the data to all students. Once the students answer the questions/polls, the feed will display the results and send to the database. The database then send this data to be seen in the session history.

## Mapping Subsystems to Hardware:

The overall system does indeed need hardware. However, we reduced the need of heavy hardware through web technology. For instance, we host our server through Amazon Web Services rather than locally. So as data is constantly called/sent the cloud is constantly running and working on handling the data. The other mandatory hardware we utilize is smart phones. To get the most out of the app the system needs two smartphones. One actor to be the instructor and one to be the student. Each actor will be able to set up their classroom, communicate through sockets, and take attendance. In essence the system needs

a minimum of three pieces of hardware to communicate, two smartphones and a server web base in the middle.

## Persistent Data Storage:

A noSQL database was used to store the data acquired. The data that is stored includes user information. Which entails Email, encrypted password, and userType. Under the email we store the classes one is in as well as the history of previous sessions. This data is pulled each time a user logs in. The user history will keep updating every time a lecture session ends. Each session information includes the attendance, questions and quizzes.

## Network Protocol:

Our network protocol is pretty much standard sockets using React Native. For instance when it comes to creating a new session, An instructor will send the request to the server. The backend database will send this message to all the clients that are registered in the class schema. If a student inside a session decides to ask a question in class, then the message will be sent from the student to the server. The server will then send out the message to all clients including the instructor.

We chose this approach because the app needs to be in constant communication between student and instructor. Also, most of the time when one message is sent from one client it needs to be sent to all users within that schema.

An example of how each student's view is customized to provide their previous sessions is this fetch call.

```
async getSessions(){
  const courseObj = navigation.getParam('courseObj', 'null');
  this.setState({course: courseObj})
  let response = await fetch(SERVERURL/getsessions', {
    method: 'post',
    headers: {
      'Accept': 'application/json',
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      classID: courseObj._id,
    })
  })
}
```

```
let data = await response.json()
if ('status' in data) {
    console.log('No Sessions in this Class')
} else {
    this.setState({ sessions: data })
}
}
```

We customize the state and view based upon the information gathered from the database.

## Global Control Flow:

The flow control is driven through both execution and time dependency. For instance, the events that every student has to go through include logging in, view their classes, and click on the running session. A running session will only appear when the instructor of particular class has created it. Once created, the server will send the session's information to all clients of that class. The system later becomes time dependent. The session will have a timer in which during this time frame, students may ask question, answer quizzes, and take attendance. Concurrent activity between student and Instructor in real time live sessions is realized through web sockets. When the session ends all the data will be collected in subsequent session name. Then this data will appear in the database where the users account will pull from it. The attendance flow is reliant on location of the instructor and student location. The location of the instructor when activated is given a range, if student is inside range then he or she is marked present.

In reality the system does rely on certain events to occur before others. However when a session is created the session will partly rely on time.

## Hardware Requirements:

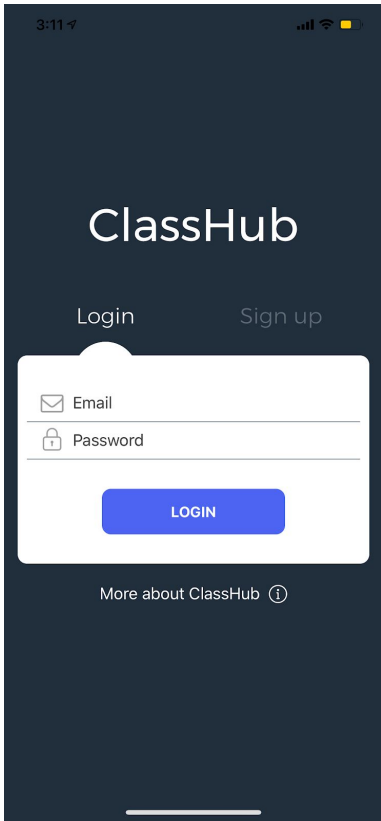
The system resources we depend on are mobile devices and the internet. The communication network between users' mobile device to server to mobile device is indicted heavily on constant internet connection. The database being hosted on Amazon Web Services creates less dependency on having the database stored on a device. However, in order to effectively use the app, a constant internet connection is needed. Without a network then not only can a user be logged on but a previously logged in user cannot view

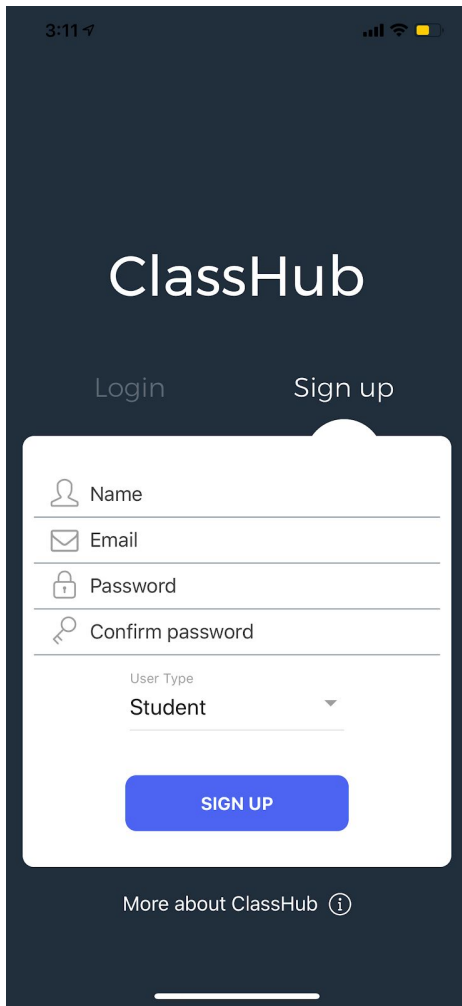
any previous session history or classes. The local storage on each mobile device does not have to be large at all. All larger pieces of data accumulated will be sent and saved to the database.

## 10. Algorithms and Data Structures

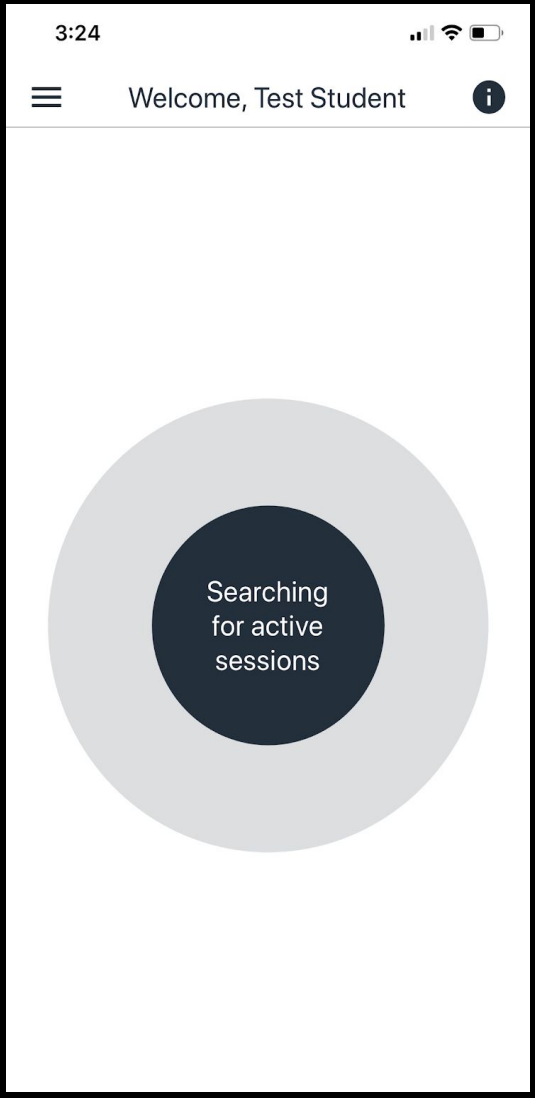
We use arrays to store the data sent from the backend API. React native allows us to dynamically map the arrays to render UI elements using `map()` and `filter()` functions. We chose this method as it is the most optimal and generally considered best practices for JavaScript. We also use the average to calculate the session ratings displayed in the instructor view. We aggregate the ratings given by every student for a particular session and take the average.

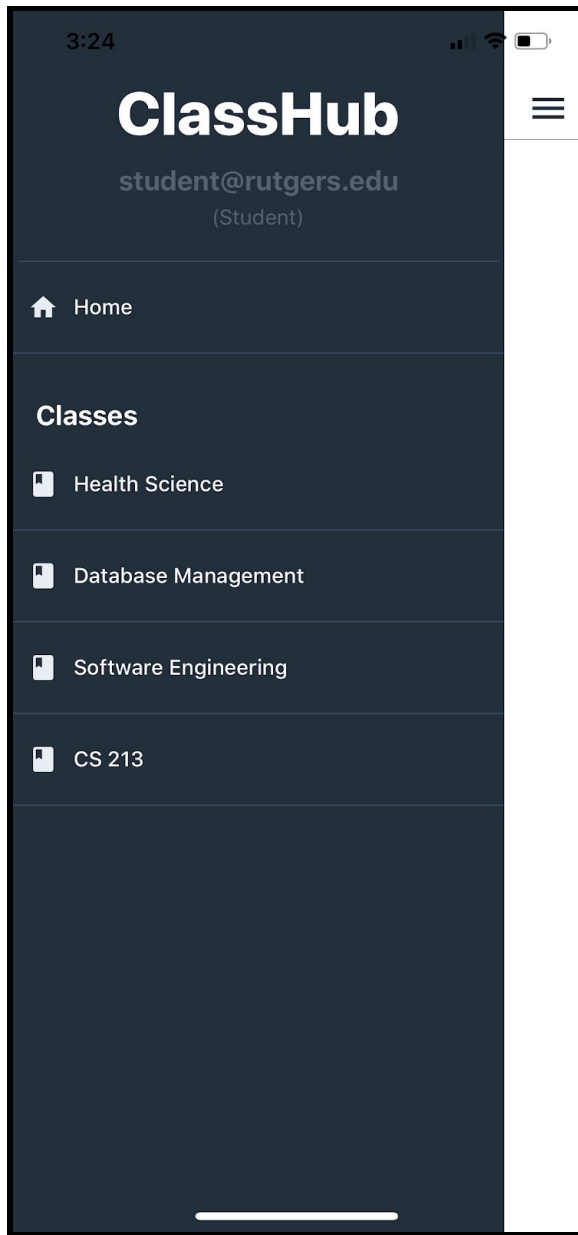
## 11. UI Design and Implementation

UI Design	Implementation details
 The image shows a mobile application login screen for 'ClassHub'. The background is dark blue. At the top, the time is 3:11. The app name 'ClassHub' is centered in white. Below it are 'Login' and 'Sign up' options. A white login form contains 'Email' and 'Password' fields, a blue 'LOGIN' button, and a 'More about ClassHub' link with an information icon.	<p>This is the landing view of the user who has opened the application for the first time. This login portal is where each user will be able to login and authenticate their username with the database. If they do not already have an account they will be required to sign using the sign up tab next to the “Login” title and above the white box.</p>



If a student or instructor does not have a username then they will be able to sign up through the signup portal. ClassHub requires a user choose their respective User Type. We were able to make the design optimal for any user to register their information in to the database. In addition to that, email formatting and password encryption have been added to the final product.

 A screenshot of a mobile application interface. At the top, the status bar shows the time 3:24, signal strength, Wi-Fi, and battery icons. Below the status bar is a navigation bar with a hamburger menu icon on the left, the text "Welcome, Test Student" in the center, and an information icon on the right. The main content area features a large, light gray circular graphic with a dark gray center. Inside the dark center, the text "Searching for active sessions" is displayed in white.	<p>This is the homepage for the student once they have successfully logged in with their credentials. The purpose of the student's home page is to search for any nearby active sessions as the main reason they should be logged in is to participate in the lecturer's active session. All other features are a priority below that.</p> <p>Once a lecturer has set up his/her active session, the nearby API implementation of ClassHub will locate the single lecturer from the student's view in a moderate radius around the student's phone and the implementation of sockets will confirm their connection and join them into the lecturer's session automatically.</p>
---	---



The side menu is implemented in almost the same way for both the instructor and the student. Both the instructor and students can access this menu by swiping the left side of their screen to the right or simply press the triple bar button on the top left corner of their screen.

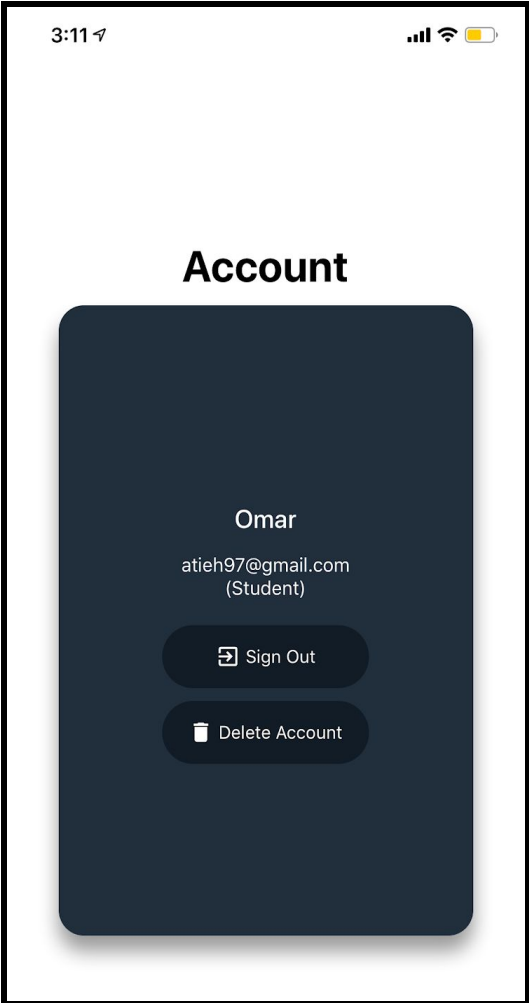
For the student, the home button will lead them to their homepage where the app will begin “searching for sessions”.

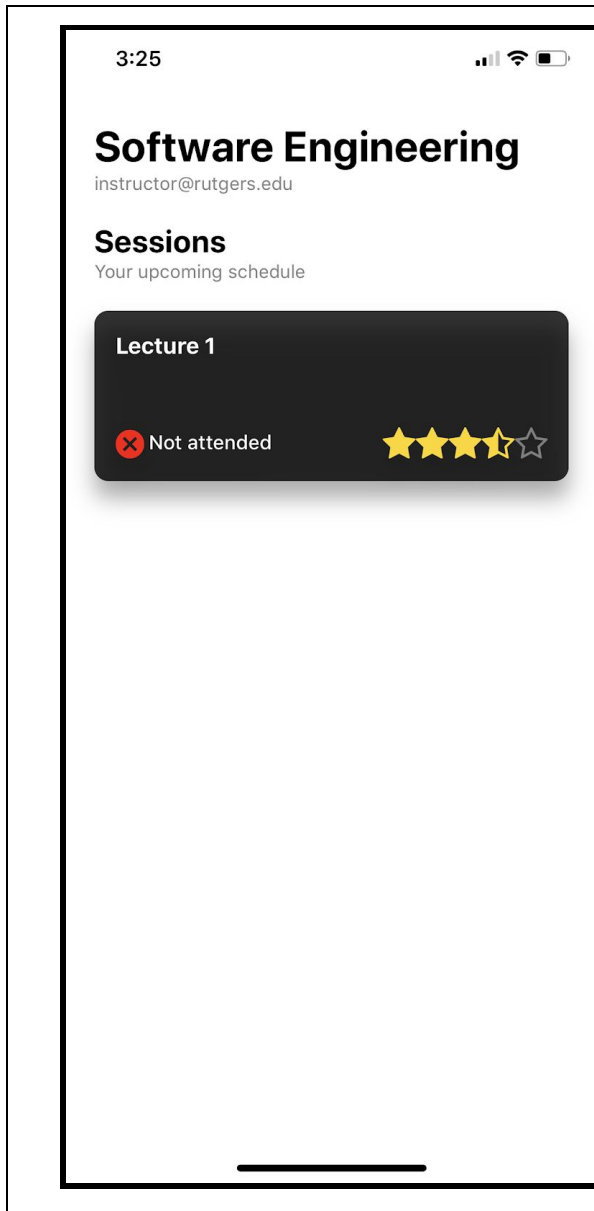
All classes that are listed are the classes that the student has been registered into by their respective professor. The student can press on any of them and will be lead to that respective “Class Page” as will be shown soon.

These buttons lead to different pages for the instructor. For example the classes listed will be the classes the instructor teaches, has created in the app and is in charge of managing. There is one more bonus to the instructor that will be shown soon.

The email address directly below the “ClassHub” logo leads to the user’s “User Profile” since this app is not a social media platform, the UI and functionality of the user’s profile was kept minimal as will be shown next.

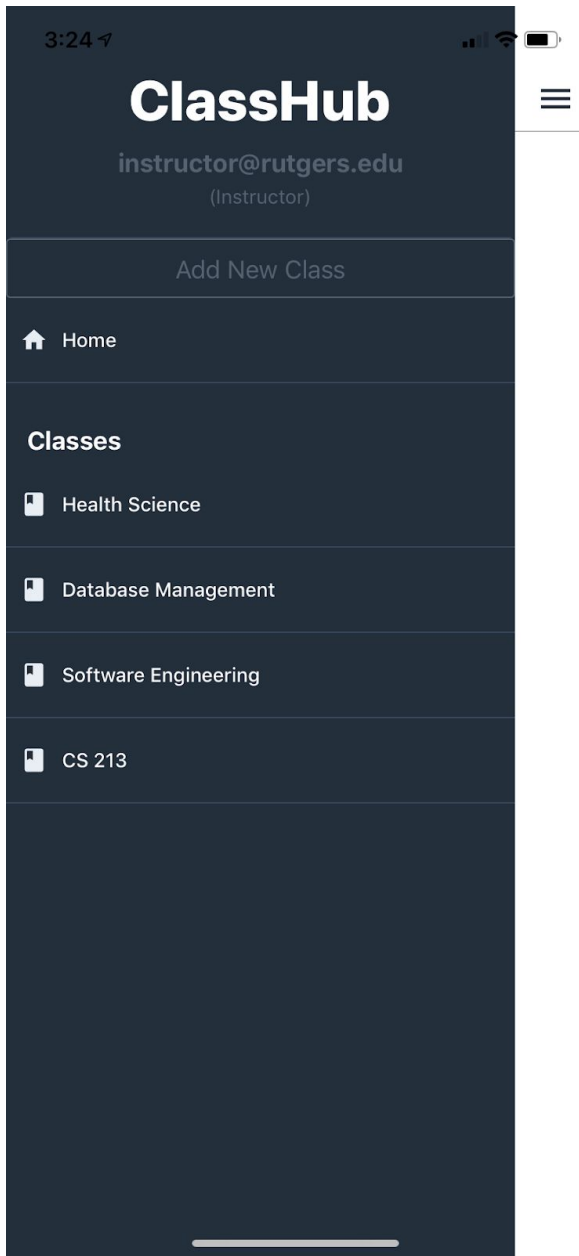


 <p>The screenshot shows a mobile application interface for an account page. At the top, the time is 3:11 and there are icons for signal strength, Wi-Fi, and battery. The word 'Account' is centered at the top in a bold, black font. Below it is a dark blue rounded rectangle containing the user's profile information: the name 'Omar', the email address 'atieh97@gmail.com', and the role '(Student)'. At the bottom of this rectangle are two buttons: 'Sign Out' with a door icon and 'Delete Account' with a trash can icon.</p>	<p>This is each user's profile that they can access by pressing their email address located directly below the app's logo on the menu bar. Since the app's purpose lies in acting as a major tool in bridging the communication between students and instructor, there was no perceived need to waste time and energy on sprucing a simple account page.</p> <p>Therefore, the functionality was kept to signing out of the app or deleting the account.</p>
---	--



This is the student's class page. They can access each respective class page by pressing the class' name in the menu bar. Each class page acts as a record of the student's experience of the class' lectures so far. The sessions are rendered as cards with their title, student's attendance of that lecture, and the student's rating of that lecture visible. As the class progresses through the semester/school year, the class page will serve as a data book of the lectures of that class as well.

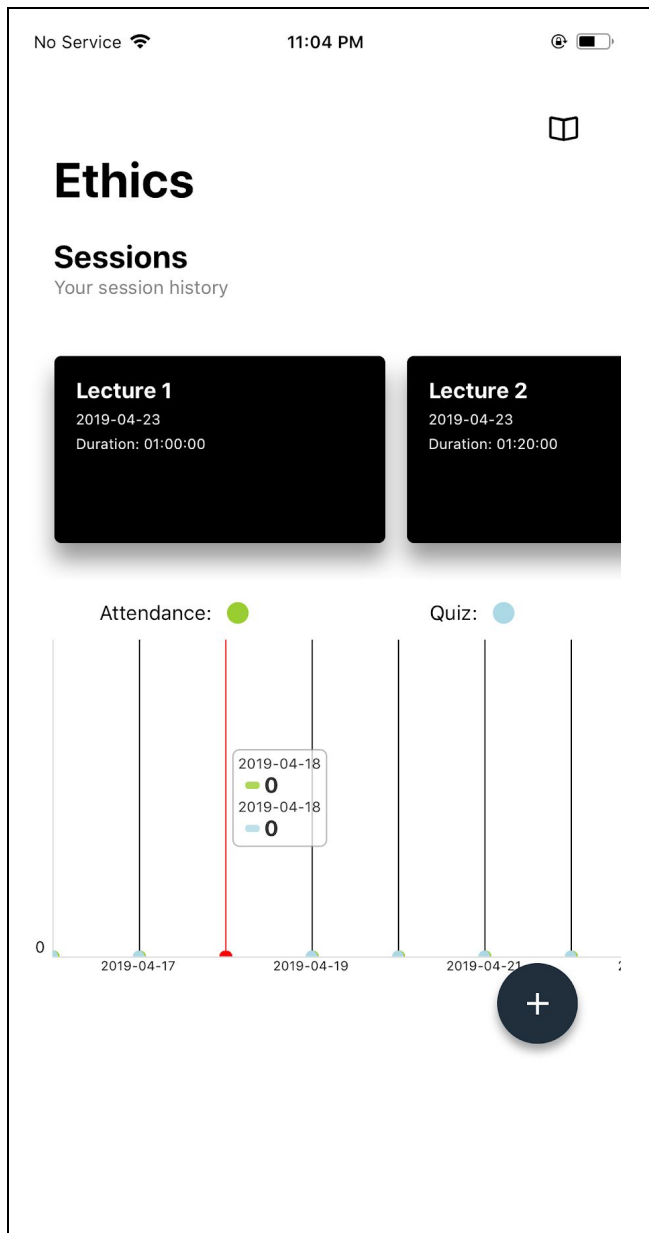
To elaborate, each session card can be pressed to view further details about that specific lecture of that specific date, on that specific topic (that will usually be the title of the session). These details include questions asked during the lecture, the instructor's remarks, and quizzes taken during that lecture. More on this later.



Now we move onto the instructor's perspective.

This is the menu bar for the instructor. As can be seen there is an addition to this bar as opposed to the student's bar and that is the "Add New Class" button. The instructor can press this button to create a new class that they are in charge of and fill the roster for that class in by uploading a comma separated file (.csv) of a column of emails that they would like to add to their class as students.

Other than that, the class buttons lead to a different class page than the students' as discussed before that will be shown next.

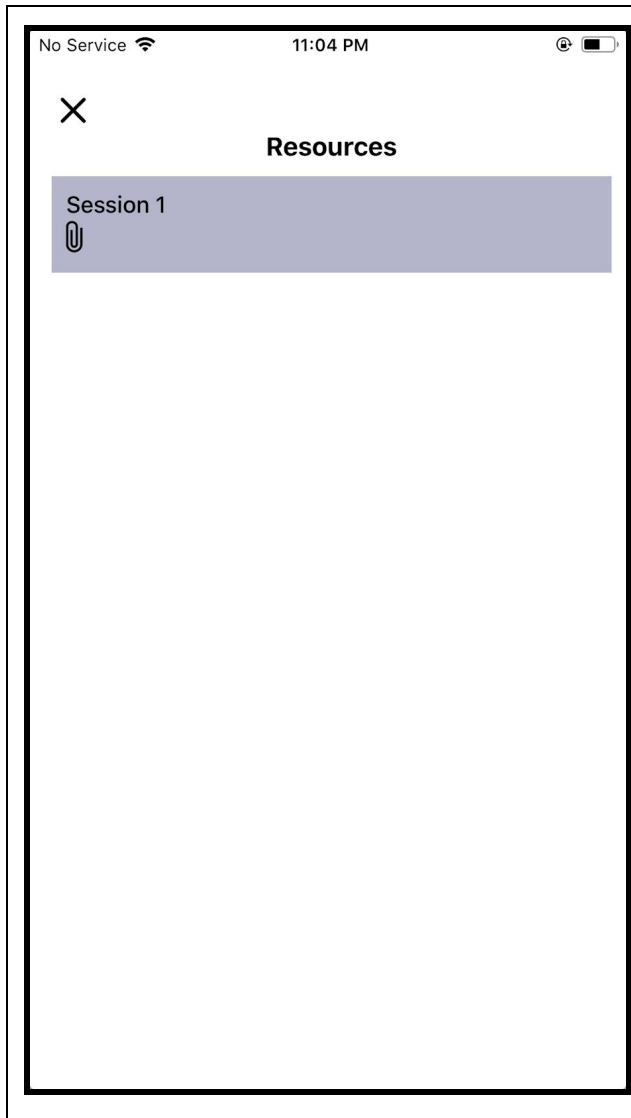


The class page for the instructor consists of two main parts: the sessions and the class statistics.

The sessions are rendered in a horizontal, scrollable row, and similar to the student's view, they display the title, date, and duration of the session and can be pressed on to view more details about that specific session. There are more details that the instructor can view about the sessions than what the student can view, as will be shown soon.

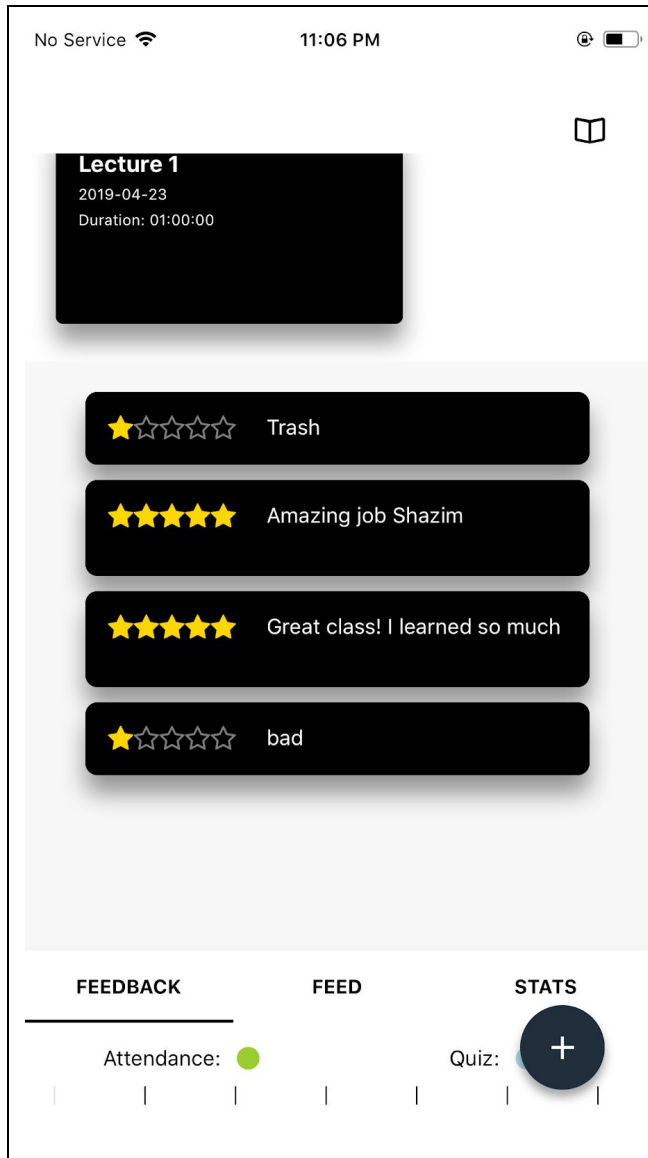
The class statistics section is a graph that the instructor can view to judge how the sessions are going in the attendance and quiz departments over the course of the semester/school year. The x-axis are dates, and the y axis is a percentage of how many students attended out of the total roster, and a percentage of the total average grade of quizzes for the session of that day.

To elaborate, if the students collectively scored 10 correct quizzes out of the 20 issued for that session. The class average score for that quiz was a 50% and so the graph would show a point on 50% on the date of that session.



The instructor can upload resources related to the class such as slides, documents, images etc. This feature is not yet complete as it is recently conjured and the students cannot view the uploaded resources. Given enough time, this feature can also be given proper life using sockets and react-native magic.

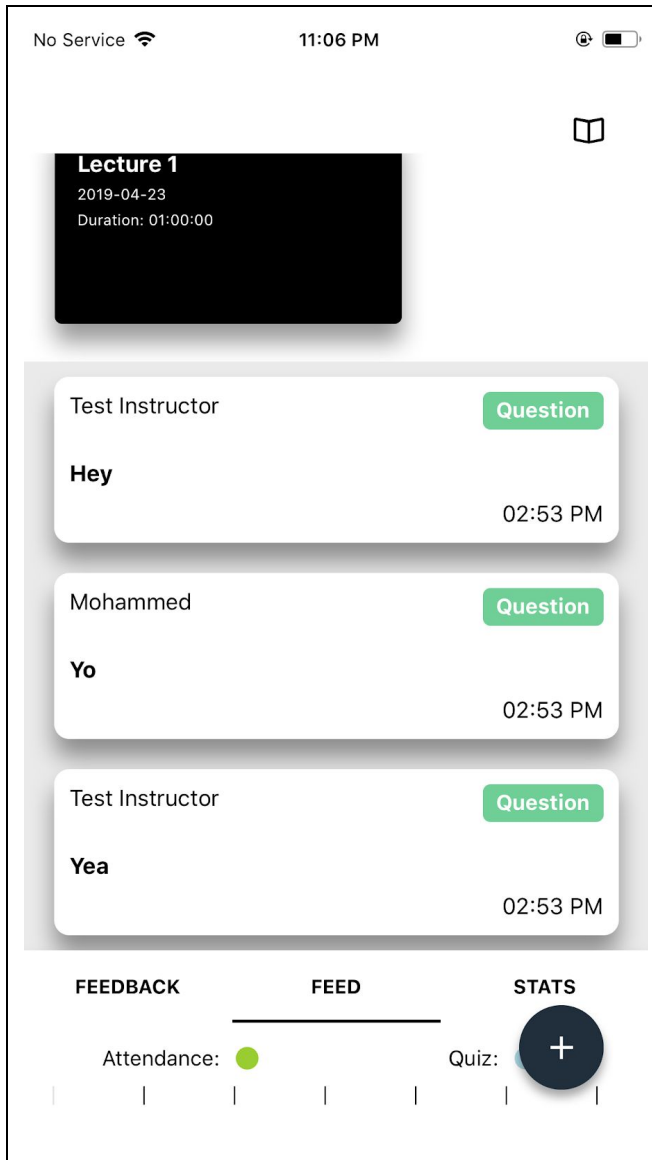
This UI component can be accessed by pressing on the book button on the top right corner of the class page screen as shown in the previous image.



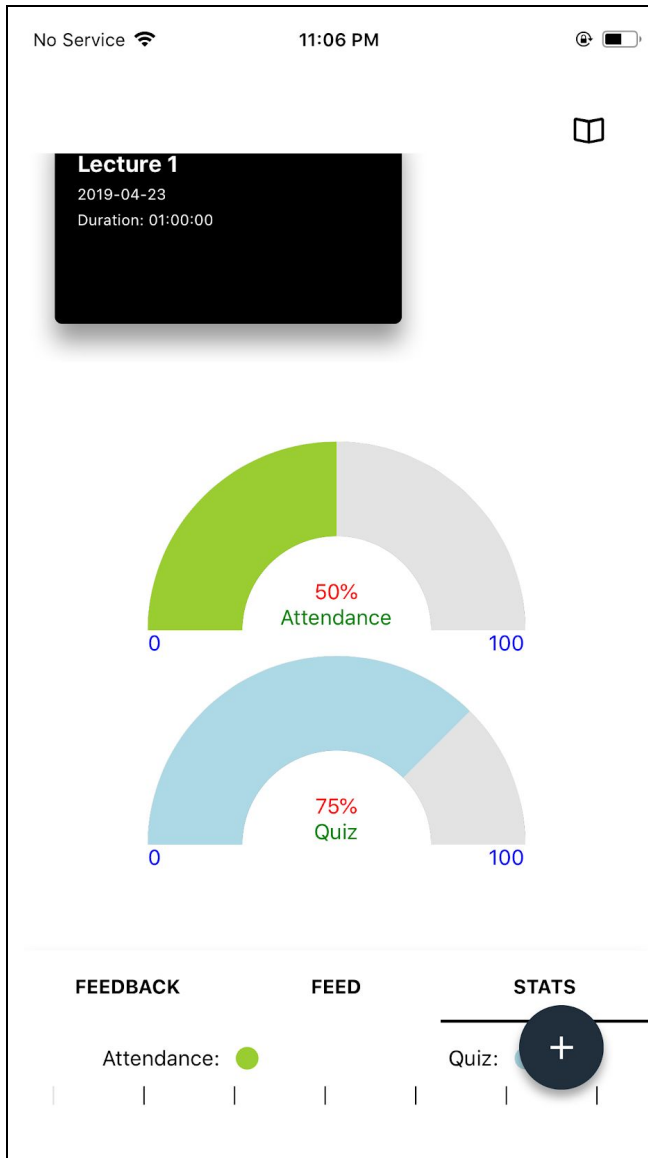
Upon pressing a session card, a sub-view below the card row will unveil itself by sliding down. In this sub view there are three tabs that describe the relevant details about the session and a tab bar that controls the navigation of this sub view. These three tabs are: feedback, feed, and stats. The instructor can choose to navigate using the tab bar buttons or by swiping left and right on the view itself.

In the feedback tab, the instructor can view the feedback left by the students about the session. This feedback is prompted to the student when the instructor ends the session (more on this later). This feedback consists of a 5 star rating of that session along with some brief comments. These are rendered vertically and will expand the sub view according to their quantity.

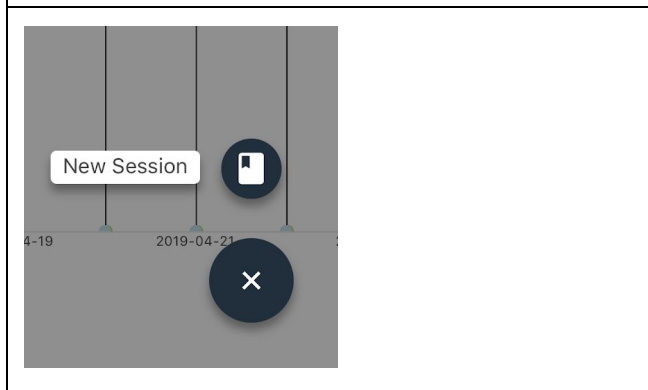
The sub view's maximum height is determined by which tab's content covers the most vertical distance. This is not the most pleasing to the eyes sometimes but if more time was available, a solution was inevitable.



The feed view of the sub view allows the instructor to revisit the session's discussions, questions, quizzes and comments. This data is taken straight from the live session's chat feed present in the app's database and formatted in a more fitting manner.

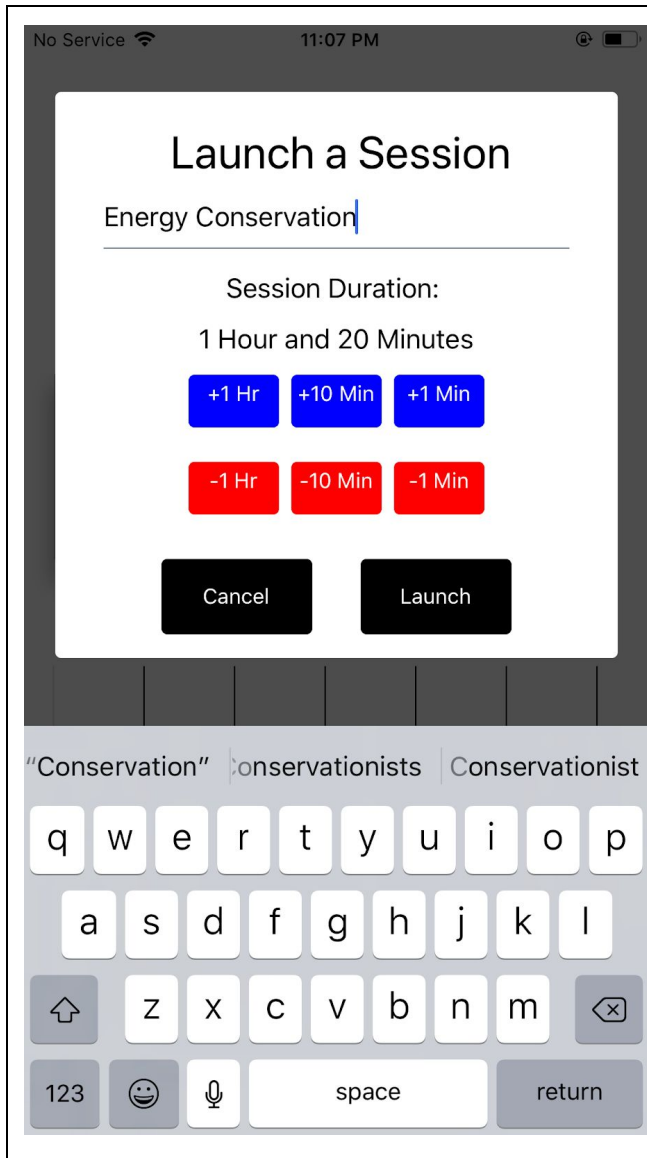


The stats tab contains the attendance and quiz quantities and average percentages for the session. The data from the database is intended to be rendered using the gauges shown, however since this was a new feature attempted to be implemented at the last moment, data population issues could not be resolved in time and are left to the next generation of capable developers. Good Luck!



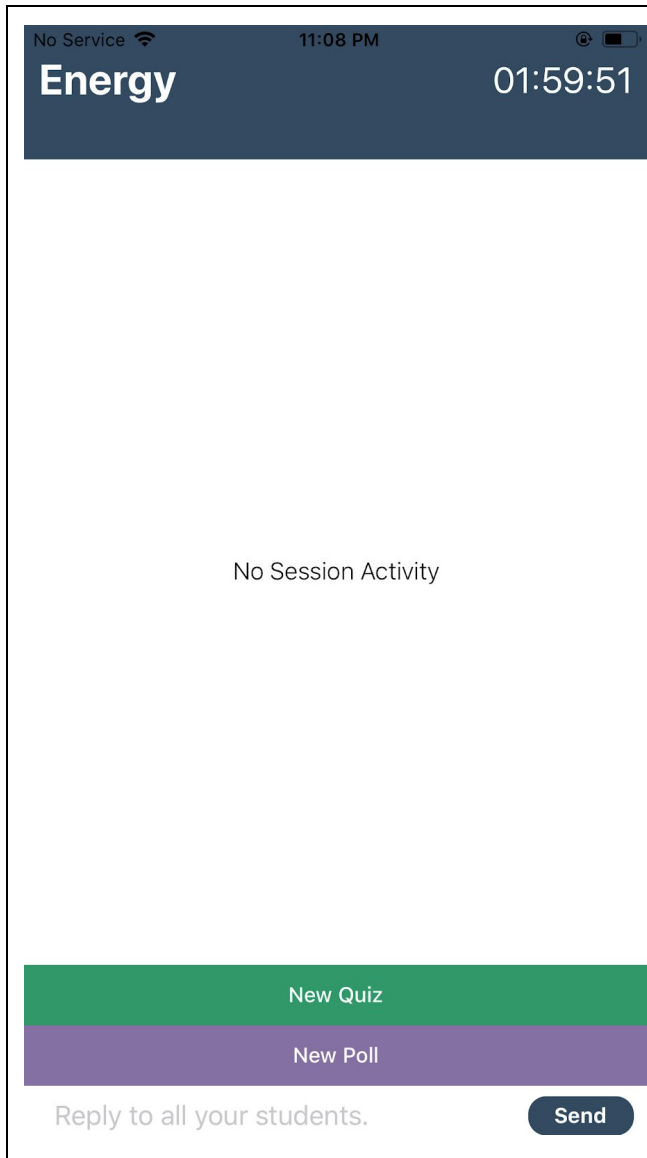
Now to move on to the actual live sessions of the lectures. They are born when the instructor decides to press the + button available to them on their respective class page. This button leads to a magical command the instructor unleashes onto the app, the command known as "New Session".





Once the instructor presses that the app will need some information from the instructor like “What do you wanna call this session? How long do you want it to be? Are you really sure about this? You can still cancel now and we can pretend this never happened.”

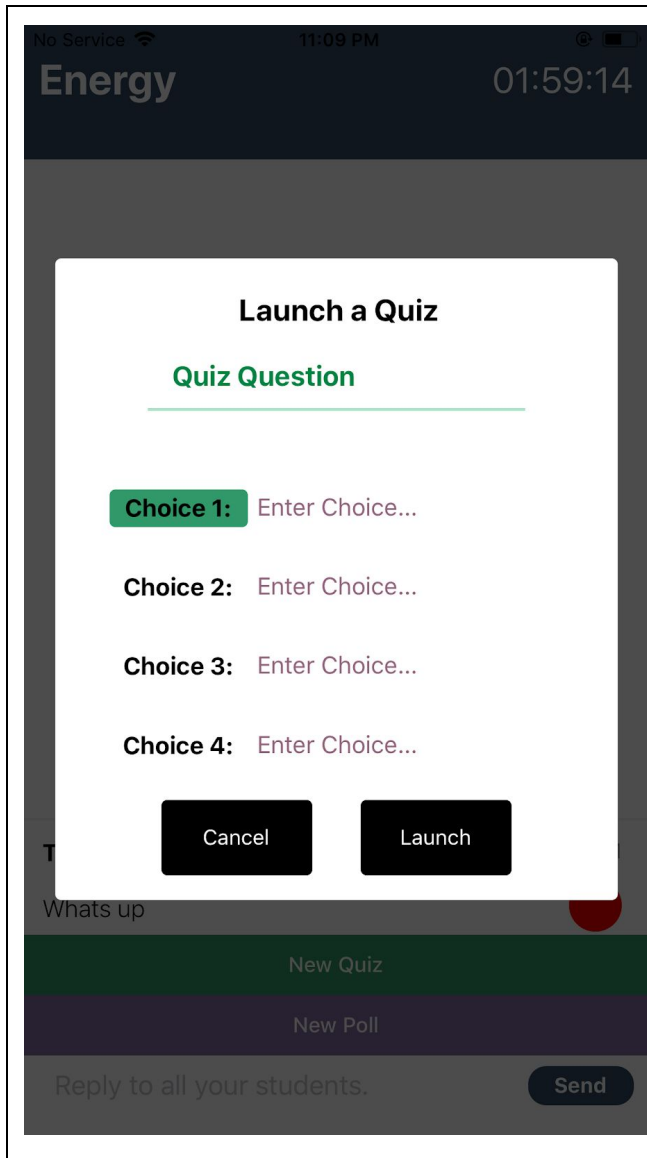
But alas, if the instructor is adamant in their decision, they can fill the title and add time as they see fit using the buttons provided and then launch the session.



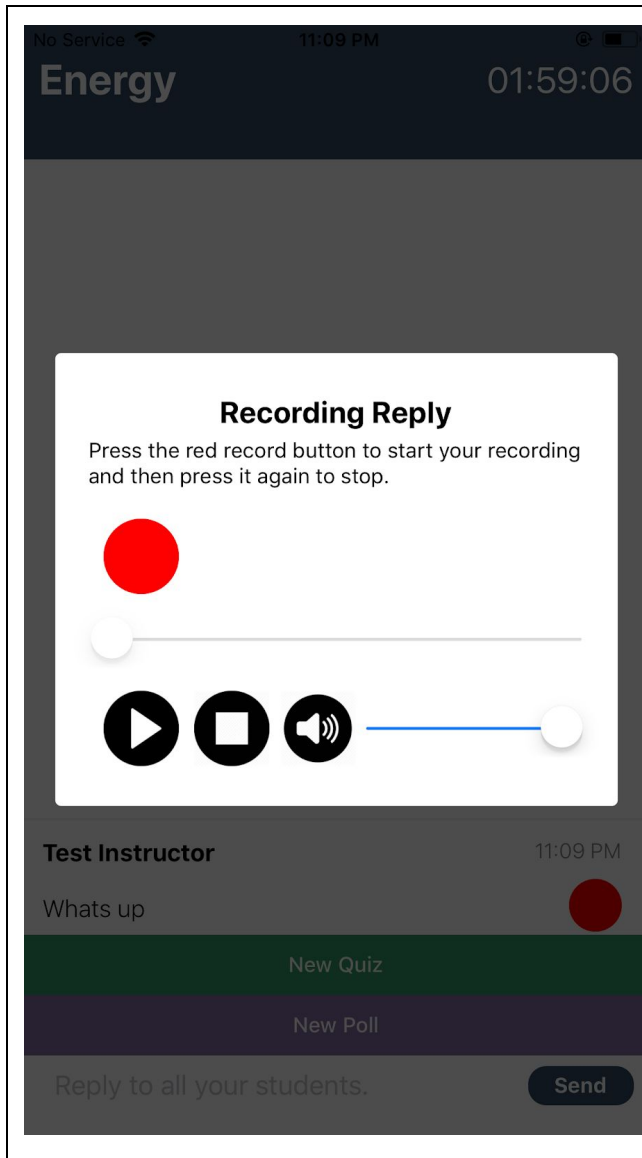
Pressing the launch button will launch the session for the instructor immediately. Some permissions will be asked for location, and audio. The instructor can post text, quizzes and polls as shown.

The students do not have the options to post anything except text (in the form of questions mostly). They do however have the option to upvote other's posts.

After launching the session, any students that are in a certain range of the student will be able to find it using their app automatically. And will be rendered the same view immediately.



Upon pressing new quiz or poll, this format will arise, asking for further information. After filling it out and pressing launch, all students present in the live session will be given the quiz to select their answer. After the instructor stops the quiz the students will be taken back to the live session's chat feed.



This a brand new feature for the ClassHub app that makes it unique for what it is.

Since this is a lecture app. The instructor can choose to answer his students by pressing the red circular recording button on any posted text in the chat feed. He/She can follow the instructions, record the answer, play it back if needed, and then the intention is to have it available in the live chat and chat history for students to listen to and use to study.

Although the recording itself is fully functional, this idea was thought of at some of the last moments of the course, it could not be fully implemented as the instructor's recording is not fully connected to the text they respond to and is not available for the students to view in their history. The creators of ClassHub leave this to the next generation of passionate coding warriors to come.

Within all the design implementations, "Ease-of-use" is among the biggest priority. We had to change the gestures for many popups as well as the state because they did not provide the user with ease of use. The menu had to be rearranged for it to open and close by sliding gesture. When pop ups appear we initially had "x" buttons to close them when we realized it's easier for the user to simply slide down on the popup to close. The UI has been heavily focused on for this phase of the project. The UI has been made functional using static data, and now it will be implemented using sockets and real-time updated data from the database.

For the various props and components we need to fetch data from the database so they could activate. For instance, with the session cards we need to fetch from the database the data of the session history in order for them to display and provide the statistics of the session. We also enabled the slide gesture to scroll through them. The aim of design is for the instructor and student to not feel like uses this app is going to be a burden or consume time. The objective of the app is

both make the student and instructor experience ideal. This demand that the application's UI has to be simple and easily accessible.

Additionally, designing a modal to create new sessions that was quick and easy to use was pretty challenging. We had to redesign and implement the modal three different times after the first two times were considered difficult to use. In our last implementation, we brainstormed together to have a design that will be easy to read and use for instructors to quickly launch a session for their students.

## 12. Design of Tests:

### Unit Testing

- TC-1 Login/ Sign up functionality as well as Authentication**
- TC-2 Instructor/Student ability to view menu screen populated with classes**
- TC-3 Instructor's ability to add a class**
- TC-4 Tests Instructor's ability to launch session**
- TC-5 Tests Student's ability to receive a session**
- TC-6 Tests Student's ability to perform attendance**
- TC-7 Tests Instructor's ability to launch a quiz**
- TC-8 Tests Student's ability to answer quiz**
- TC-9 Tests Student's ability to ask questions within session**
- TC-10 Tests Students ability to comment and rate session afterwards**
- TC-11 Tests Instructor's ability to see session history**
- TC-12 Tests Instructor's ability to launch a poll**
- TC-13 Tests Student's ability to answer the poll**
- TC-14 Tests class ability to see poll**
- TC-15 Tests Student's ability to upvote questions and comments in feed**
- TC-16 Tests Instructor ability to record answers for student questions**
- TC-17 Tests Student ability to send direct messages to class instructor**
- TC-18 Tests Instructor ability to upload resources for a class**
- TC-19 Tests Google Nearby API ability to test attendance**

TC	Tests	Test Coverage
1	Sign up several email types (gmail, msn, rutgers) and lengths as both student and instructor and login.	<ul style="list-style-type: none"> <li>• If any type, length, or amount of email produces an issue for signing up or logging in to the database.</li> <li>• If any type, length, or amount of email produces as issue from either being designated as a student or instructor in the</li> </ul>

		database..
2	<p>Use instructor accounts to create classes and upload rosters and logout/in to test them being rendered on the menu.</p> <p>Use student accounts uploaded as rosters to check if the class is available on the menu screen.</p>	<ul style="list-style-type: none"> <li>● If instructors are able to create classes.</li> <li>● If instructors are able to upload rosters.</li> <li>● If instructors are able to login/out with classes being rendered correctly.</li> <li>● If menu is rendered correctly.</li> <li>● If all student emails in the roster supplied are applied to the appropriate class.</li> <li>● If students are connected to the instructor after roster upload.</li> <li>● If students can view the class they are connected to.</li> </ul>
3	<p>Use instructor accounts to enter the menu bar and use the add class option to add class and upload roster.</p> <p>Login/out to check if classes are rendered correctly.</p>	<ul style="list-style-type: none"> <li>● If all instructor accounts are able to create classes.</li> <li>● If all instructor accounts are able to upload rosters.</li> <li>● If classes are rendered correctly upon class creation and roster upload.</li> <li>● If classes are rendered correctly upon login/out.</li> </ul>
4	<p>Use menu bar in instructor account to access a class. Inside the class use the action button on the bottom right to launch sessions of multiple lengths. Do this for several classes.</p>	<ul style="list-style-type: none"> <li>● If the instructor can launch a session in any class successfully.</li> <li>● If the session can be of any length.</li> <li>● If the session's launch is successful for any class and any instructor account.</li> </ul>
5	<p>Login to several students in the roster of the class for which a session has been launched and check if the student can see the session as live and can access it.</p>	<ul style="list-style-type: none"> <li>● If the session notifies all students in the class roster that it is live without exception.</li> <li>● If any student on the roster can view the session as live.</li> <li>● If any student can access the live session.</li> </ul>
6	<p>Use of the add quiz button in instructor account to be able to write questions and answers and launch the quiz.</p>	<ul style="list-style-type: none"> <li>● If the instructor can write question and all answers for the quiz.</li> <li>● If the instructor can launch the quiz during the active session.</li> <li>● If all students in the roster receive the same quiz the instructor's version.</li> </ul>

7	The ability of students to answer quiz during the active session and record the students answers.	<ul style="list-style-type: none"> <li>• If students in the roaster can view the quiz launched by the instructor.</li> <li>• If students are able to answer the quiz and there answer get recorded in the database.</li> <li>• If the quiz disappears after the time is up and students can go back to the live session.</li> </ul>
8	Use the student's session history so students can view their answers to quizzes and know the correct answer.	<ul style="list-style-type: none"> <li>• If students can view the quizzes asked during the live session after session is done.</li> <li>• If students can know the correct answer for the quiz and can view their own answers.</li> <li>• If students can view all the questions answered in the session.</li> </ul>
9	Use of geolocation so instructor can take attendance.	<ul style="list-style-type: none"> <li>• If instructor can set specific diameter so student in this diameter can be marked as attended.</li> <li>• If only students in the roaster in the given diameter can be marked as attended.</li> <li>• If student who leaves the session before the session is attended, they will be marked as not attended.</li> </ul>
10	Use the active session window for student's to ask their questions	<ul style="list-style-type: none"> <li>• If students in the roaster can access the live session for a given class.</li> <li>• If students can type their questions and send it.</li> <li>• If all students in the active session can view the questions asked by other students.</li> <li>• If the instructor can view all students questions.</li> </ul>
11	Use the feedback window after the session is ended to rate the session.	<ul style="list-style-type: none"> <li>• If the feedback window appear right after the instructor ends the session</li> <li>• If the students can type their opinion about the session in the feedback section.</li> <li>• If the students can rate the session in a scale out of five.</li> <li>• If the students feedback get recorded in the database.</li> </ul>

12	Use the session history for the instructor's side to view the students' feedback about the session and statistics for quizzes.	<ul style="list-style-type: none"> <li>• If the instructor can view the session history after the session is ended.</li> <li>• If the instructor can view the students' comments and their rating of the session.</li> <li>• If the instructors can view the statistics for the quizzes asked in the session.</li> </ul>
13	Use the poll modal for students to answer polls launched by instructors in session feed.	<ul style="list-style-type: none"> <li>• If student can view poll question and choices</li> <li>• If student can answer poll and the answer is recorded in the database</li> </ul>
14	The ability for the entire class (both students and instructor) to view poll results in session feed	<ul style="list-style-type: none"> <li>• If students and instructor can see poll statistics in session feed.</li> <li>• If poll results are stored in the database</li> </ul>
15	The ability for students to upvote questions and comments in session feed	<ul style="list-style-type: none"> <li>• If students can upvote questions in session feed</li> <li>• If upvotes are stored in database and rendered on the session view for both students and instructor</li> </ul>
16	The ability for instructors to record audio as answers for student questions posed in the feed	<ul style="list-style-type: none"> <li>• If instructors are able to record audio to answer student questions</li> <li>• If the database stores the audio and links it to the particular question</li> <li>• If students are able to playback the instructor recording whether in the live session or session history.</li> </ul>
17	The ability for students to send direct messages to instructors	<ul style="list-style-type: none"> <li>• If students can send message to instructor</li> <li>• If instructor is able to view and respond to message</li> </ul>
18	The ability for instructors to upload class resources for students	<ul style="list-style-type: none"> <li>• If instructor is able to upload resources for class</li> <li>• If resources are stored in database</li> <li>• If students are able to view resources and download them on to their device</li> </ul>



19	The ability for Google nearby API works for attendance for students	<ul style="list-style-type: none"> <li>• If students within the classroom are able to be signed in as “attended”</li> <li>• If students outside of the class room are not signed in</li> <li>• If students who walk in late to the class are signed in as “attended”</li> </ul>
----	---	---

## 13. History of Work, Current Status, and Future Work

History of Work				
Tasks	Description	Start	End	Status
<b>1)User Portal</b>				
	Create a register window using react components	2/10/19	2/18/19	Complete
	Make a login portal for users	2/10/19	2/18/19	Complete
	Backend Support (data management)	2/10/19	2/18/19	Complete
	Login Authentication with server	2/10/19	2/18/19	Complete
	Design initial user interface	2/10/19	2/18/19	Complete
	Forgot password module	4/10/19		Complete
<b>2)Student portal</b>				
	Navigation, side menu, home page and intuitive design	2/10/19	2/16/19	Complete

	Aggregate view to show the user's classes and previous sessions	2/27/19	3/15/19	Complete
	Make session cards to correlate to each previous session	2/18/19	2/27/19	Complete
	Make a profile page to allow user to log out or delete account	2/10/19	2/16/19	Complete
	Show all classes in navigation pane	2/10/19	2/26/19	Complete
	Allow students to access resource files	4/10/19	4/15	Complete
<b>3) Session portal</b>				
	Enable socket communication from student to server to everyone in class	2/27/19	3/5/19	Complete
	Create message feed every client is able to communicate with each other	3/5/19	3/20/19	Complete
	Create question popup for instructor to ask a question in live session and create multiple choice answers	3/5/19	3/18/19	Complete
	Communicate with sockets to send quizzes to students	3/5/19	3/18/19	Complete
	Create a poll window for instructors to ask students	4/1/19	4/18	Complete
	Communicate through sockets to send and receive poll questions and answers	4/1/19	4/18	Complete
	Audio answers recorded by professor and sent to session history	4/5/19	4/18	Complete

	Upvoting system for students to be able to upvote certain questions in session message feed	4/5/19	4/15	Complete
	Utilize Google Nearby API to Track attendance of nearby devices	4/5/19	4/14/19	Complete
<b>4. Instructor Portal</b>	Navigation Bar	2/10/19	3/14/19	Complete
	Session history tabs fetching from database and displaying	2/18/19	2/27/19	Complete
	Create and Send Notifications to all students	3/1/19	3/14/19	Complete
	Ability to add a new class via CSV file	3/10/19	3/18/19	Complete
	Ability to add resource file for all students in class to view	4/5/19	4/10	Complete
	Ability to view all results from previous sessions including quizzes, polls, and attendance.	3/1/19	3/25/19	Complete
<b>5. Database</b>				
	Express server on Node js hosted on AWS	2/10/19	2/12/19	Complete
	Rest API Endpoints on AWS EC2 instance	2/3/19	3/6/19	Complete
	Database schema using MongoDB	2/10/19	3/28/19	Complete
	Socket.IO and real-time communication between clients and server	2/23/19	3/25/19	Complete
	Receive all data from sessions and store and send to session history	2/23/19	3/15/19	Complete

Above we have our plan of work and we accomplished a lot. However, after demo 1 we realized that we needed to add some more functional features to improve product. We also had to leave the geolocation idea as it did have some pitfalls. We are working diligently to add all the features before demo 2. We have already implemented Google's Nearby API to improve attendance functionality.

## Key Accomplishments

- Created entire app and all its features from scratch
- Created the versatile and reliable attendance taking mechanism
- Created a live virtual classroom
- Allowed for in class virtual quizzes, polls, and messages to be taken in real time through socket communication
- Created session history to keep track of all data
- Created very pleasant UI design

Some possible directions for this project is to incorporate it with sakai like features to make it the ultimate University resource

## 14. References

- [1] Software Engineering Book. Ivan Marsic.  
[http://eceweb1.rutgers.edu/~marsic/books/SE/book-SE\\_marsic.pdf](http://eceweb1.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf)
- [2] React-Native Documentation. Facebook.  
<https://facebook.github.io/react-native/>
- [3] Git & Github Crash Course. *Traversy Media*, Youtube.  
[https://www.youtube.com/watch?v=SWYqp7iY\\_Tc](https://www.youtube.com/watch?v=SWYqp7iY_Tc)
- [4] Expo Mobile App quick-start Documentation. Expo.  
<https://docs.expo.io/versions/latest/>
- [5] Node.js Documentation. Node.js.  
<https://nodejs.org/en/about/>
- [6] Yarn Dependency Management Documentation. Yarn.  
<https://yarnpkg.com/en/docs/getting-started>
- [7] React Tutorial. *The Net Ninja*, Youtube.  
<https://www.youtube.com/playlist?list=PL4cUxeGkcC9ij8CfkAY2RAGb-tmkNwQHG>
- [8] Google nearby API  
<https://developers.google.com/nearby/connections/overview>.