

# iClicker Web Application Project

Kelly Yeh, Ryan Siu, Carolyn Hanson Lee  
CS176B Winter 2020

## Introduction

Our project is a live streaming question/answer application that is designed to replace the need for iClicker in lecture halls, providing an alternative way for professors to interact with students in class and take attendance. In our application, we utilized Django channels and Redis to add websocket support along with Daphne which is a websocket protocol server for ASGI. Django is a web framework that provides rapid and maintainable development. Django channel is built upon Python specification ASGI. Implementing it into our project allows us effectively create a persistent multi-client connection between professor and students. We specified the IP address and port number Daphne binds to, and it auto-negotiated between HTTP and Websocket. Whilst building this application and debugging, we added the Django debugger toolbar, which is a configurable set of panels that display various debug information about the current request/response.

## Background

### Networking Components:

Thanks to Python, Django and Django Channels, we were able to implement a web application with very ambitious features. We decided to use these tools because it allowed us to complete our goal without as much heavy lifting. Once we had configured the fundamental asynchronous connection of Django Channels for the server and multiple clients, we spent time working on the user interface, user experience, login functionality, database/model creation, and CSV uploading. In terms of synchronous connections, we were able to establish the primary HTTP connection between a user and the web application's server. An HTTP request example is when a user makes an HTTP request, we create a new http scope with the user's path, method, and headers. Then, our web app can send an `http.request`

event with HTTP content. On the other end, a `http.response` event is generated back to the browser. Afterwards, the connection is permanently closed.

Django channels is our project's integration layer which allows us to implement WebSockets built over ASGI. The channels application has capability to handle asynchronous connections and sockets, and our main website uses HTTP protocols with synchronous connections. When a server creates a new asynchronous connection, we want to disallow all clients from IP addresses that do not match UCSB's private network IP address. When a professor creates a new lecture iClicker session, on the backend, a channel layer "group" is created on the live server. From there, every time a student/client logs in with the correct lecture id, they are added into the same group, called 'lecture'. For added security, we only allow asynchronous client connections to the server when the lecture id parameter is correct and the IP address is permitted. This allows the professor to securely broadcast messages to everyone in the lecture (the students), which creates our fundamental iClicker functionality. The professor can choose to broadcast multiple questions while the asynchronous connection is kept alive, or, he/she can close the "lecture" session and kick out everyone else. We use Docker and Daphne together to locally host the server and the client simultaneously. Daphne handles the transitions between synchronous and asynchronous connections for us.

We use sessions to save important information pertaining to a currently logged in user, such as a session key, session user, session lecture id, and so forth. Sessions comes from Django's SessionMiddleware feature. Each time an HTTP Request object is generated, the session dictionary is passed along with the request. This upholds the integrity of a single user's experience past login and lets us pass data to new website views. For more permanent information, we connected our application to the sqlite3 database.

### How to Run:

Our server files for establishing the networking components are as follows: 'questions/consumers.py', 'iclicker/asgi.py', 'iclicker/routing.py', 'iclicker/wsgi.py', and 'iclicker/settings.py'. Our primary client files for connecting to the server are 'questions/urls.py', 'question/views.py'. Assuming compilation is required, we absolutely stress the importance of Docker and Daphne. Docker is a container platform and Daphne is Django Channel's HTTP/WebSocket protocol server for ASGI and ASGI-HTTP. The software requirements for compiling our project include Python, various Python modules, SQLite3, Django, Docker, Daphne, Redis, Django Channels. Hardware requirements are any PC - we did not use CSIL Machines for any part of our project.

### Performance Goals and Testing:

The first performance metric that we focused on while implementing our web application was regarding server connection scalability with multiple clients in an asynchronous connection. Unfortunately, we were unable to evaluate performance results. The documentation for testing Channels consumers says that it is much more difficult due its underlying asynchronous nature and we did not have enough time

in our quarter. However, Django Channels provides a testing framework that would be considered in the future. Django's Async tests allow you to test the HTTP and WebSocket connections between servers and clients. We would be able to send and receive fake input, create fake client connections, force the server to delay responses, and force disconnections. We would like to try testing our web application with a single server against 30 - 50 clients using the same asynchronous connection. For security testing, we would like to know if our web application actually has the ability to block asynchronous connections that are not verified as UCSB IP addresses. Our primary issue with testing for this was that our local application is not running under UCSB's network currently. However, in the future we would attempt to connect under UCSB's private network in order to run our testing.

## **Methods**

In order to run the program make sure to have all the software requirements satisfied, first, to be able to access the application, add the server's IP address to the ALLOWED\_HOST in settings.py which enables connection to Django. Then, start docker and migrate all database components to apply any changes. Then, we use the command to run docker with redis. This command starts the server while allowing you to asynchronously run a server. Next, run daphne main server command to allow automatic handling between synchronous and asynchronous connections to have daphne listen for HTTP and WebSocket requests on 0.0.0.0:8001. Running on local host will take you to the Iclicker home login page. Depending on which user's lecture you are in: student or professor, which can be assigned by the admin on admin authorization and authentication page, you will be redirect to either professor home page or student home page.

On the professor home page, you will have the option of uploading csv file of poll questions. Once you upload the csv file, a live poll associated with a unique lecture ID will open with the first question of the csv file being displayed. Students will be able to submit answers to the poll by entering the lecture ID. Once all answers are submitted for the respective question, the professor can click the "Next" button to go to a live poll for the next question. The "Next" button will create a live poll for each question until there are no more questions. At this point, clicking "Next" again will close the poll.

On the student home page, you will be prompted to enter the lecture ID, which is created by the professors' poll. The question and answer choices to the question will be displayed after entering the lecture ID. Everytime the professor clicks "Next", the next question and its answer choices will display. Once the professor closes the poll, the server will disconnect all clients connected to the lecture ID within the asynchronous connection.

## Admin authorization and authentication:

**Django administration**  
WELCOME, KELLYYEH. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Home > Authentication and Authorization > Groups

Select group to change ADD GROUP +

Q  Search

Action:  Go 0 of 2 selected

<input type="checkbox"/>	GROUP
<input type="checkbox"/>	professor
<input type="checkbox"/>	student

2 groups

**Django administration**  
WELCOME, KELLYYEH. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Home > Authentication and Authorization > Groups > professor

Change group HISTORY

Name:

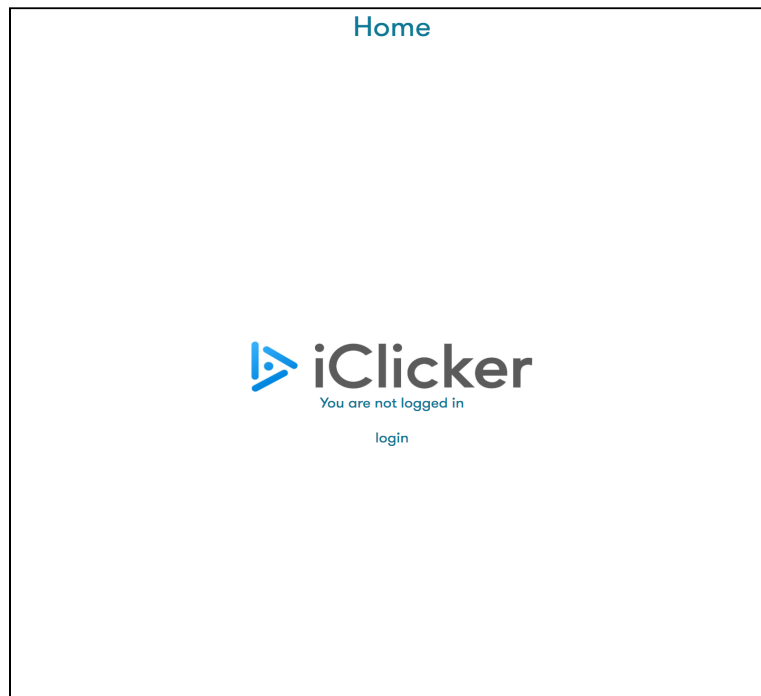
Permissions:

Available permissions	Chosen permissions
<p>Q <input type="text" value="Filter"/></p> <p>admin   log entry   Can add log entry admin   log entry   Can change log ent admin   log entry   Can delete log entry admin   log entry   Can view log entry auth   group   Can add group</p> <p>Choose all</p>	<p>contenttypes   content type   Can add contenttypes   content type   Can char contenttypes   content type   Can dele contenttypes   content type   Can view polls   mc vote   Can add mc vote polls   mc vote   Can change mc vote polls   mc vote   Can delete mc vote polls   mc vote   Can view mc vote</p> <p>Remove all</p>

Hold down "Control", or "Command" on a Mac, to select more than one.

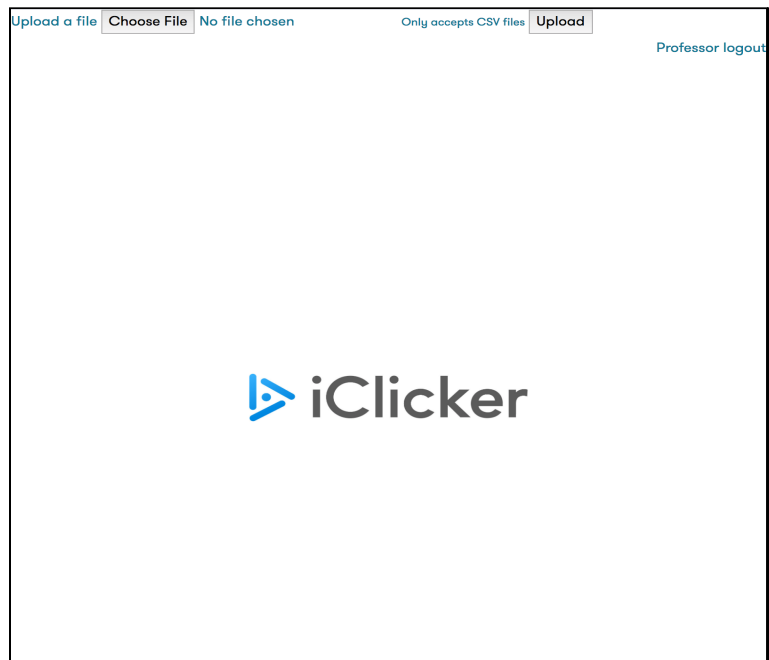
Delete Save and add another Save and continue editing SAVE

Login and after logout:

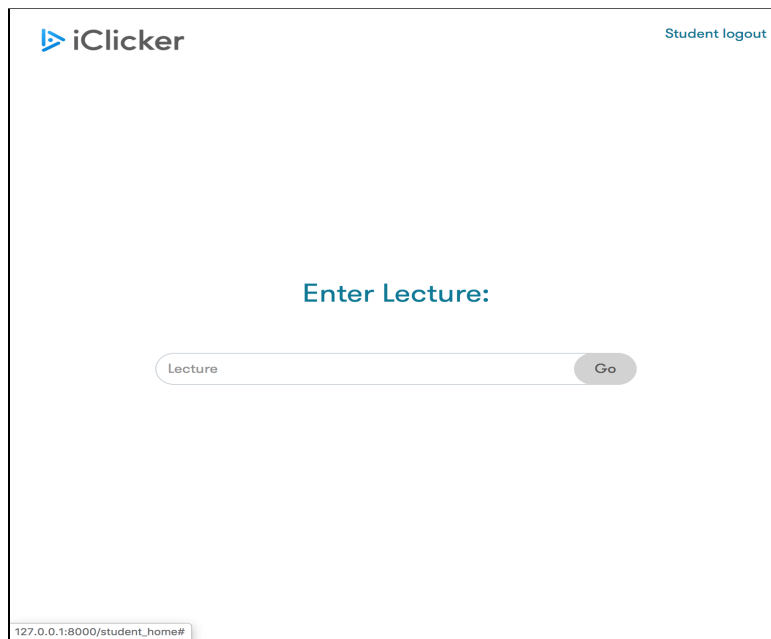


A screenshot of the iClicker application's login page. At the top center, the word "Login" is displayed in a bold, black font. Below it, there are two input fields: "Username:" followed by a text box with a blue border, and "Password:" followed by a text box with a grey border. At the bottom center, there is a small, rounded button labeled "Login".

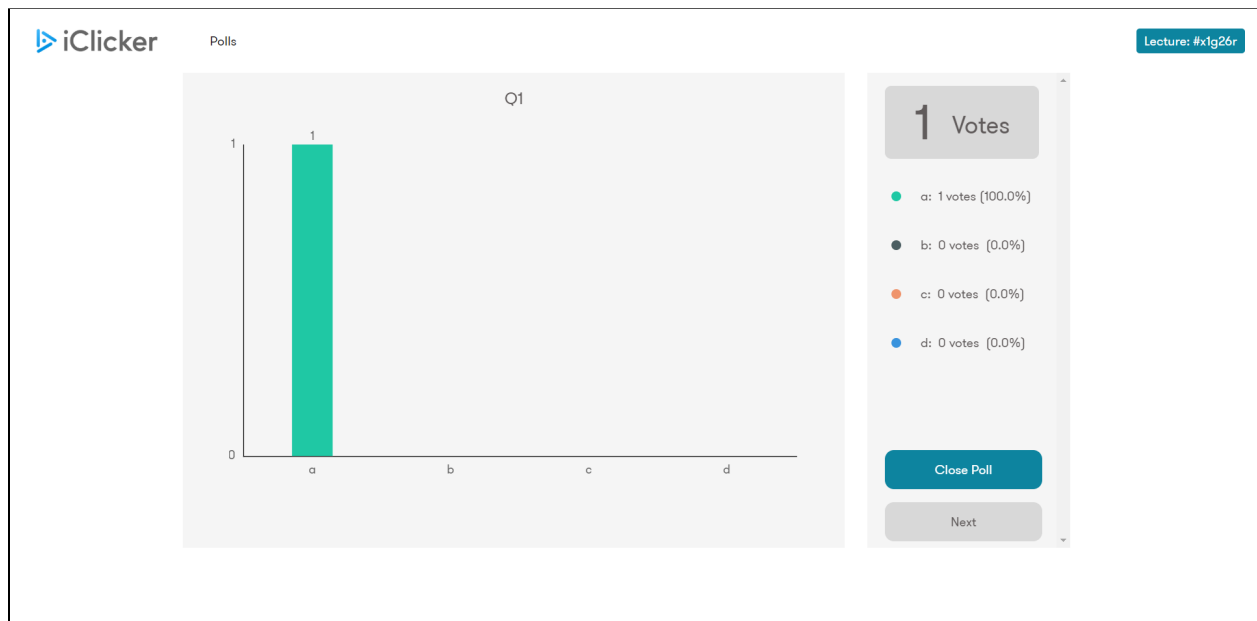
## Professor Home Page:




## Student Home Page:



## Professor Live Poll UI:



## Student Poll UI:

 LIVE POLL IN PROGRESS... Student Log Out

Q1

A. a

B. b

C. c

D. d

 LIVE POLL IN PROGRESS... Student Log Out

Q1

A. a

B. b

C. c

D. d



## **Conclusion**

For the future, if we ever improved this project, we would want to add scalability for handling high volumes of clients connected to the same server in a single asynchronous connection. This project was exhaustive because we had to combine our efforts in full stack development with networking functionality. This included HTML and CSS on the front end and Javascript, Python, and JQuery on the backend. The networking functionality is based on Python, Javascript, and Django's Channel app.

When starting the project we had to familiarize ourselves with the Django framework which was much more complicated than we had anticipated it to be. Trying to run the project locally on its own took a lot of work because many errors were raised with the different software installed and trying to figure out these errors so that we could run was very time consuming. Django utilizes databases for its models and we had to learn how to correctly configure and utilize them, while also providing the functionality that we were expecting them to have. Since our project is a web application, we also had to learn a lot of techniques for using JavaScript and HTML in order to implement the front-end UI we had in mind. Trying to implement the UI was extremely hard to debug since JavaScript is an object-based language.

To create this project, we worked on this project extensively and utilized Visual Studio Live Sharing so that we could use a single repository to modify existing code, save changes, and even run the app on localhost using a live shared terminal. This was very helpful because we've all experienced the unfortunate side effects of merging and conflict resolution in Github standard practices. Ultimately, this iClicker web application was a very educational experience for learning more about asynchronous connections for web sockets. We came into the quarter with very high hopes and a lot of ambitious goals, but we quickly realized that setting up the asynchronous connection was actually a very challenging and time intensive task. We believed that implementing a solid foundation for asynchronous connection was the most significant part of our project, and we are happy to conclude that we achieved this goal.

## **References**

*Youtube Video Demo Link:*

[https://www.youtube.com/channel/UCJbkiJKHJNdnvLLCH1\\_9r1Q/](https://www.youtube.com/channel/UCJbkiJKHJNdnvLLCH1_9r1Q/)

*Github Repository:*

<https://github.com/kellyyeh/IclickerProject>