Multi Client Android Chatroom Using Socket.IO

Wesley Ma, Nadia Osipova, Harshit Kumar

Department of Computer Science, College of Engineering, University of Illinois at Urbana-Champaign

Introduction

Writing a chat application with popular web applications stacks like LAMP (PHP) has traditionally been very hard.

It involves polling the server for changes, keeping track of timestamps, and it's a lot slower than it should be. Sockets have traditionally been the solution around which most real-time chat systems are architected, providing a bi-directional communication channel between a client and a server.

This means that the server can *push* messages to clients. Whenever one writes a chat message, the idea is that the server will get it and push it to all other connected clients.

However we felt that native socket programming in Java for android wasn't as smooth an experience that we envisioned. Thus we planned on using Socket.IO. It is a JavaScript library for real-time web applications. It enables real-time, bi-directional communication between web clients and servers. It has two parts: a client-side library that runs in android or a browser, and a server-side library for node.js.

Aim

Our aim was two fold – First was to set up a public server using node.js, Express and Socket.io. This would handle all client connection and communications.

We then planned on writing an android client application capable of connecting to the server and sending/receiving messages.

We also wanted to create a nice looking UI for all this.

ILLINOIS

Method

Connecting to a server through Socket.io works much the same as in traditional socket programming. Below is a code snippet to get a socket for a server and establish a connection to it.

```
private Socket some_socket;
{
    try {
          some_socket = IO.socket("address");
    }     catch (URISyntaxException e) {}
}

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    some_socket.connect();
}
```

Sending data via the socket looks like below.

```
private EditText text_box;
private void attemptSend() {
    String message = text_box.getText().toString().trim();
    if (TextUtils.isEmpty(message)) { return; } text_box.setText("");
    some_socket.emit("new message", message);
}
```

We can make the socket listen for new messages from the server and asynchronously call a function.

```
mSocket.on("new message", onNewMessage);
```

And this function handles displaying the message on the client side.

```
private Emitter.Listener onNewMessage = new Emitter.Listener() {
        @Override
        public void call(Object... args) {
            JSONObject data = (JSONObject) args[0];
            String username;
            String message;
            try {
                username = data.getString("username");
                message = data.getString("message");
            } catch (JSONException e) {
                return;
            }

            // add the message to view
            user_message(username, Color.GRAY, message);
        }
    };
```

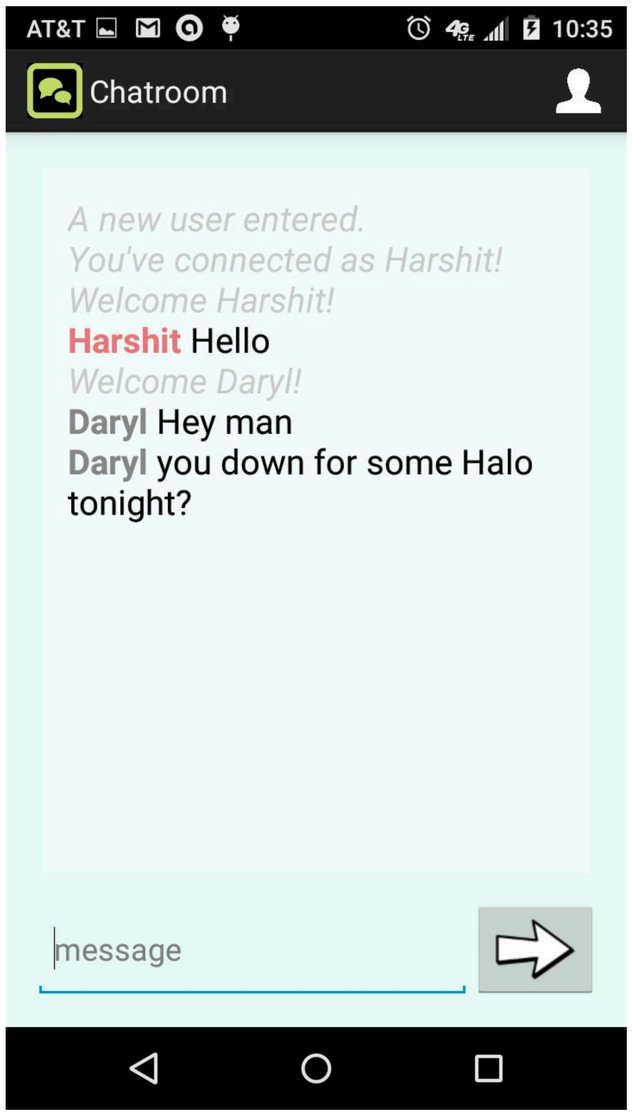
All sockets need to be closed and all listeners removed at the end of the lifetime of our Android activity.

```
@Override
    public void onDestroy() {
        super.onDestroy();

        mSocket.disconnect();
        mSocket.off("new message", onNewMessage);
        mSocket.off("user joined", onUserJoined);
        mSocket.off("user left", onUserLeft);
    }
```

Result

We succeeded in creating a chat client for android that could allow the user to chat in real-time with multiple other clients on android/web.



Acknowledgments

We would like to thank our mentor Utsav Shah for all his help and support as well as the CS 296-41 course staff for providing us with the opportunity to work on this project.