[redacted]

Desktop Texting App for Android Phones Progress Report One

Project Description

Our project consists of three major pieces: the Android application, the desktop client, and the server in the middle. The Android application is used to create a way to communicate with the server and client. This Android application in no way replaces a user's current texting app. It used to send messages from the phone's number, and also notifies the client of any incoming messages.

The next piece is the client. The desktop client is an Electron app created using React. This is the user interface which the user can interact with in order to send and receive messages from the comfort of a desktop, much in the way that imessage allows this for iOS users.

The server is a .NET Core application which handles communication between the client and Android application. This is effectively a web API which has endpoints facing both the Android application and desktop client, to be requested as needed. The server also integrates google Firebase to allow for sending messages directly to the app or client without having been requested first. This is important so that the server is able to tell the app to send a new message, without the app having to ping the server every X seconds.

This is an open source project, so it is important to us that this code be completely runnable without reliance on some server being hosted. As a result, the use case (for now) is for a user to run the desktop client and the server from the same computer. Additionally, the phone and computer are required to be on the same internet (though this is mostly a restriction as a result of CWRU's internet security settings). This way, with all the code open on GitHub, anyone could clone the repo, follow installation instructions, and use this application without messages being routed through some centralized server which could easily read and store user messages.

Background / Related Work

Below is a list of alternative solutions to the problem which already exist, along with some of their shortcomings (namely, none of them are open source in the same way ours is).

- **Android Messages**

    - Requires Android Messages app to be the primary texting application used. Currently only available for web app usage. There are also a limited number of browsers that the application works with; however, the web app is free.
- **MightyText**
    - Texting application that allows users to send and receive SMS, send photos, files, and look through their contacts. The application is app agnostic meaning that regardless of what application the user primarily uses to send SMS, MightyText can be used. The application is launched in a web app page, or as a web browser extension. There is also a fee to use the application past a certain number of SMS sent.
    - Because the user is required to log in using an email account, synchronization issues can occur between the phone and the web app.
- **Android Mirroring Applications (e.g. AirDroid)**
    - There are desktop applications that can mirror what currently is open on the user's Android screen. Unfortunately, some of these applications can have a high level of latency which is inconvenient for users. Additionally, if the user solely wants to send and receive text messages, mirroring apps may automatically forward all push notifications to the user's desktop which can be a nuisance.
- Sources:
    - https://www.theverge.com/2018/6/19/17479554/Android-messages-how-to-text-from-web-feature
    - https://www.greenbot.com/article/2102552/4-Android-apps-that-put-sms-on-your-desktop.html

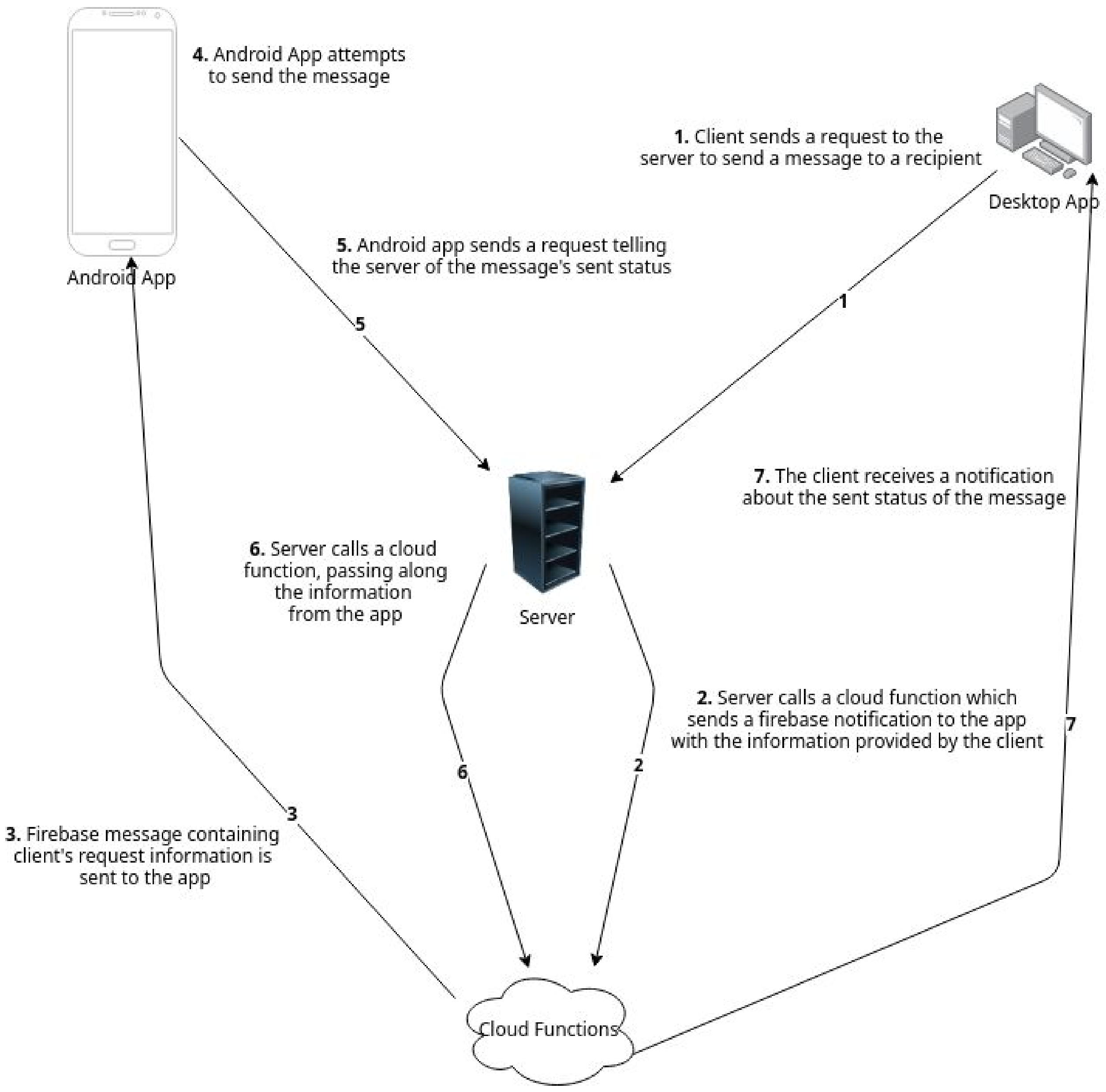
Completed Work

The major goals for the first progress report were to prove our concept,have client to Android app communication successfully tackled, and have our UI designed, but not necessarily implemented. Below, progress will be split up into the three major components to make things clearer.

Server

A lot of the largest challenges for the beginning of the project were making sure that all of the pieces of the project would actually be able to fit together, which wound up being the server's responsibility. Currently, the server has a set of endpoints facing the client and Android app, which allow for basic message sending functionality (i.e. the desktop client can give a recipient and message, which the server then will pass onto the phone, which will then be sent to the recipient's phone).

Additionally, the server has another component to deal with, which is Firebase. Firebase is our way of communicating between the server and client/app outside of generic HTTP endpoints. The normal use case for Firebase is having some sort of centralized server which has a private key tied to a Firebase project, allowing it to send messages to any registered device it chooses. Because of our open source model, we cannot create a server using this private key without giving our project's private key to everyone on the internet. As a result, we have to use Google Cloud functions, which are cloud-hosted functions which are allowed to send Firebase messages to registered devices. An example information flow diagram is shown below to explain how information gets passed around in the application.

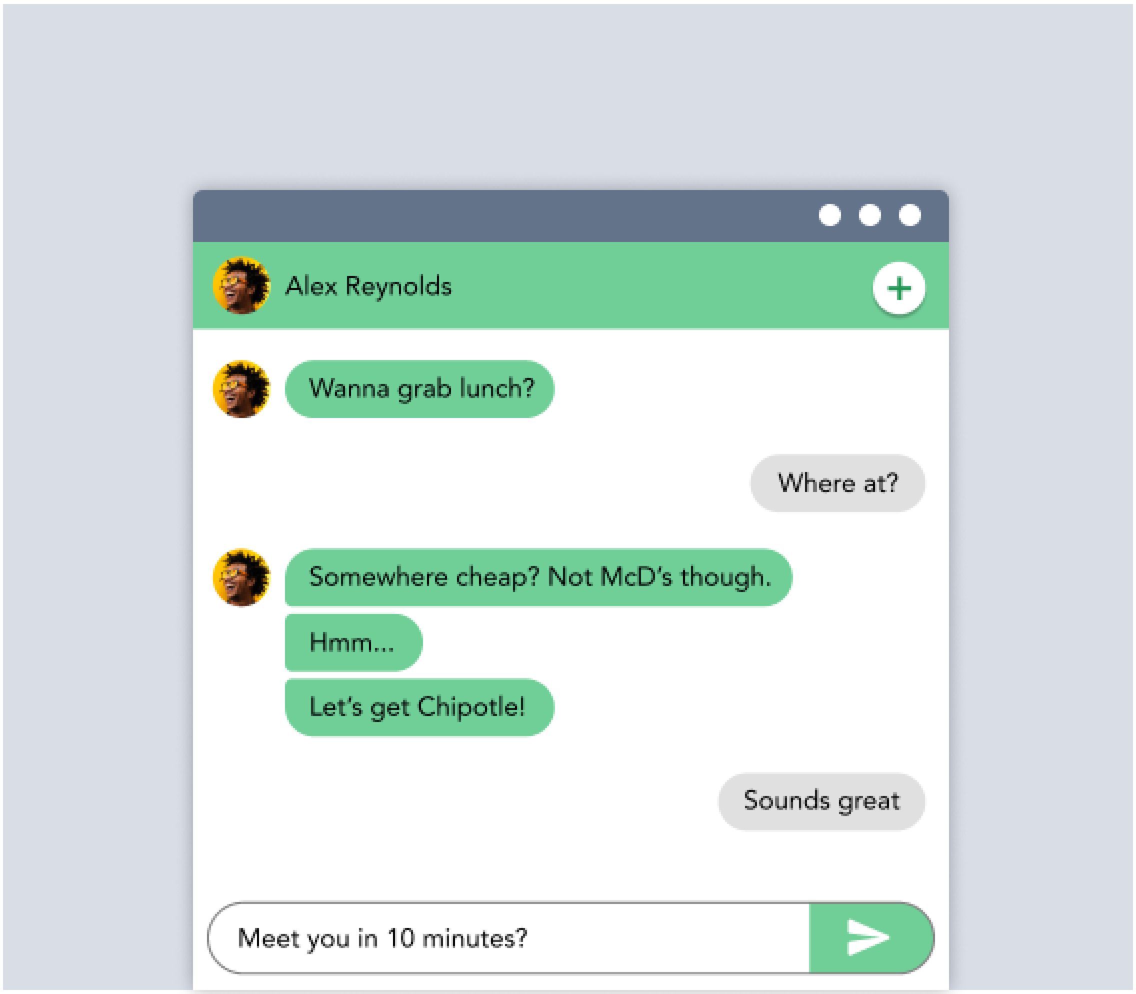

4. Android App attempts to send the message
1. Client sends a request to the server to send a message to a recipient
Desktop App
Android App
5. Android app sends a request telling the server of the message's sent status
1
5
7. The client receives a notification about the sent status of the message
6. Server calls a cloud function, passing along the information from the app
Server
2. Server calls a cloud function which sends a firebase notification to the app with the information provided by the client
7
6
2
3
3. Firebase message containing client's request information is sent to the app
Cloud Functions

Android Application

The Android application presently has the ability to retrieve a list of conversations and corresponding previous SMS messages. Additionally, it can send SMS messages from the phone to a given recipient. The user interface for the application is not a strong focus for this project, though given time at the end we may attempt to touch it up. Presently, the interface includes an area to type in the local IP address of the server and a button to "connect" to that server. Android has a particularly poorly laid out system for handling MMS messages which has taken up a great deal of development time and is the major remaining hurdle for the app.

Client Application

The client application's user interface has been designed and effectively finalized. Below is a mockup of the interface. The client app is going to use React.js inside an Electron container. In addition to having just designed the interface, many of the React components have been created and are in the process of being styled.


Alex Reynolds

Wanna grab lunch?

Where at?

Somewhere cheap? Not McD's though.

Hmm...

Let's get Chipotle!

Sounds great

Meet you in 10 minutes?

In terms of client functionality, the client has a field for a recipient, a field for the actual text of a message, and a send button. When a user types information into these fields and hits send, the client will pass the information onto the server and eventually the phone as shown in the diagram above.

Overall Functionality

The three pieces of the application work together, though there are some specific circumstances required. Namely, the server has to be up and running before the client is opened, because the client sends information to the server on startup. Additionally, the Android application has to have linked up with the server before trying to send a message from the client, otherwise it will not know where the message should go. These are caveats that we hope to remove later in the engineering process, but our first goal is functionality of the application.

Project Completion Plan

Following the submission of this progress report, there are approximately three weeks until the next progress report is due. Our goal is to finish almost, if not all, of the functionality by the second progress report. Our remaining time will be finishing anything we couldn't get completed and then final touch ups and quality of life changes. As a result, our completion goals are below:

Date to Complete By	**Things to Complete**
10/17 at 11:15 AM, no later than 10/18	● Android ○ Sending MMS messages (mostly group messages) ○ Send a notification when a new message is received by the phone ● Server ○ Additional Endpoints to support communication between Android and client ○ Additional cloud functions ● Client - FrontEnd ○ Create some sort of function to handle adding a new message to the view, such that it updates nicely ● Client - Functionality ○ Should be able to request for a phone's conversations/messages and retrieve them from the server ○ Should be able to populate a view with a conversation's information
10/24 at 11:15 AM	● Android ○ Conversation retrieval working with server ○ Fully functional message retrieval for conversation, including MMS ● Server ○ Additional endpoints to support communication between Android and client ○ Additional cloud functions ● Client - FrontEnd ○ Create side panel for conversation information and a way to populate this space with information ○ Handle dynamic sizing changes of layout ● Client - Functionality ○ Handle when to request the server for conversation and messageList information and when to populate it onto the screen
10/31 at 11:15	● Client - Functionality ○ Handle new message notifications

Management Plan:

The roles of group members have not changed since the project proposal and are as follows:

[redacted] Team Lead, Server, Server API integration, Architecture Design
[redacted] Android App, Server API Integrations, Architecture Design
[redacted] React client functionality
[redacted]: React client design and app design

The group meets each week for approximately an hour on Thursdays at 11:15 AM. This time is used to go over what has been accomplished, what still needs to get done, areas where members are stuck, and any additional topics as needed. Additional meetings are sometimes held to tackle larger problems which were unforeseen, or to help troubleshoot particularly difficult issues.

Member Contributions:

[redacted]

- Lead and organized team meetings
- Co-designed system architecture
- Coded the .Net core server
- Coded the Firebase cloud functions
- Handled setting up Firebase integration in all layers of the application
- Tested Server functionality
- Code Review for Android App
- Wrote Progress Report

[redacted]

[redacted] Co-designed system architecture
- Inspired the open source system design principles
- Coded the Android app functionality and current UI
- Firebase integration with Android
- Android Testing
- Code Review for Server

[redacted]

- App UI/UX design mockup
- React App setup
- Created React elements for the UI

- Created development server workflow to host and test React app

[redacted]
- Setup Electron app
- Handled functionality of UI by communicating with server and Firebase
- Tested client functionality