Boston University
Electrical & Computer Engineering
EC463 Senior Design Project

First Semester Report

WOOF

Submitted to

Gabriella McNevin
Associate Director of Marketing and Communications
&
Proud Dog-owner
Boston University College of Engineering
617-353-9747
gmcnevin@bu.edu

by
Team 02
Team Woof

Team Members

Nafis Abeer nafis@bu.edu
Justin Lam jlam8080@bu.edu
Chase Maivald chase1@bu.edu
Rajiv Ramroop raj19@bu.edu
Daniel Shimon dshimon@bu.edu

Submitted: December 12th, 2021

Woof: Social Media Application for Dogs and Their Owners

Abstract—Woof is an ideal cross-platform social media application for dog owners. Rather than relying on individual owners to actively maintain dog friends, we propose a React Native maps interface to display locations of “friends” added by the user. To cater the application for dogs’ preferences, we require owners to purchase a dog-tag fitted with a GPS tracker, accelerometer, and a microphone. We are implementing a recommender system based on the owner’s dog’s interactions with other dogs wearing the tag. This recommender system will leverage the dog-tag’s proximity to other tags to start collecting audio. The collected data would be processed in the cloud before being sent back to the user as a friend suggestion. Woof is expected to deliver accurate suggestions for “friends” whenever the dog has a positive interaction with another dog. Users will be provided a profile page, instant messaging, “friends” page, interaction-history page along with a Dogmaps screen.

1 Introduction

1.1 Problem Statement

There is a need to bridge the disconnect in communication between humans and their pet dogs. Dogs are unable to tell their human when they would like to go for walks, and it is up to the human to incentivize themselves to take their pets out. Additionally, dogs cannot tell their humans that they like hanging out with another dog. Unless the owner is attentive to every interaction their dog has, they miss most of the positive interactions their dog experiences. These positive interactions are beneficial for dogs, and repeated interactions with the same dog are great for reinforcing behavior learned during previous exchanges.

Studies have shown that dogs can greatly benefit from long term friendships and playtime outdoors. In fact, “if deprived of the opportunity to play with other dogs, they have no other choice but to direct play behavior towards humans” [1], meaning repeated dog-to-dog interaction could reduce responsibilities for owners. Dogs tend to show more brain activity when shown other dogs [2], indicating that they enjoy the presence of their companions. Furthermore, playtime for dogs could serve the following functions: 1) developing motor skills; 2) training for the unexpected; 3) social cohesion; and 4) play as a by-product of biological processes [3]. These benefits are enough to show why its important for owners to let their dogs make friends, or at least take them out for walks.

But what incentivizes owners to take their dogs out? What if humans were too lazy to play with their dog for a day, but their dog still needs daily social interaction? We humans love coordinating activities, for the most parts, but what mediums do dog owners have for coordinating park visits? Upon discussion with our client, we came to realize that there are no specific large-scale applications geared towards the dog community. As she had put it, “We like using instagram to show off our dogs, but we don’t really collect Instagram tags of dog owners we meet in the park. We like knowing when our friends are out, but we don’t really have strangers on our Snapmaps.” That is a fair assessment seeing how it could be awkward to ask people you just met for Instagram and Snapchat handles. It would be much better if those people popped up on your “app” as a suggestion. Even better would be if the suggestion was mutual and both parties agree to add eachother as friends without verbally communicating the exchange. This takes away the social awkwardness for owners to add friends. Verbal communication to arrange dog-park meetings is another extra step that our clients must deal with, and there should be a medium to simplify this process. (Nafis)

1.2 Woof and its purpose

Woof's purpose is to be the medium of communication for the dog community. The application will not only be the mode of messaging for the dog community, but also a method for owners to understand their dogs' behaviors. Dogs are constantly interacting with other dogs while on their daily walks, and these interactions could either be positive, negative or neutral. Woof's purpose is to keep a record of these "positive" interactions. The application will leverage hardware enabled dog-tags to first understand an interaction as positive or negative. The dog-tag would need several sensors to record the dogs' motions, as well as their barks and utilize some form of machine learning algorithm to decide if this sensor information are indicators of positive or negative interactions. Woof then serves the purpose of connecting the two parties that pertook in these "positive" interactions. The indicators of positive interactions would translate to "friend" suggestions. Our solution, Woof, is going to help foster dogs' relationships with each other, allowing them to connect more frequently. We are proposing a social media platform to allow owners to connect with other friendly dogs, as well as recognize other dogs which aren't too friendly with your own dog. The next time one's dog is playing around with another dog in the park for hours, there will be an easy way to connect and keep in touch with the other owner. Next time either of the owners want a companion to visit the park with, they will have an easy method to analyze their previous good interactions and decide if they would want to visit the park together. If either of the owners are already at the park, and there was no time communicated for a visit, the lack of messaging could be made up for with Woof's map features. Finally, Woof's hardware could be used to also track one's dog in real time, so if the dog were to ever run away, we will have the owner covered. (Nafis)

2 Concept Development

2.1 Engineering understanding of customers problem

There is a certain process that goes into taking issues that our potential customers face, and further converting those into working components that would solve these issues. One of the main issues is being able to provide a social media application to give dog owners a place to interact with and discover other dog owners. There would have to be a profile page for users to be able to provide information on their dog, very similar to how current social media applications and even dating applications would function. Pet information, pictures, and preferences, and input would all have to be stored in a proper back end to then be able to be pulled into the application. Our two proposed back ends consisting of Firebase and SQL are helpful for doing so. Moving forward from profile pages, there must be a method for users to interact with one another and create these meaningful relations. This moves into another main part of this social media application, the maps and messaging component. Once friends are made, in similar fashion to 'Find my Friends' on iPhone and SnapMaps on Snapchat, friends want to be able to see where their friends are currently located and be able to interact with them from there. This is broken up into several different parts. From a UI perspective, there must be an easy-to-use interface that will display a user's friends' locations. We must also include a messaging interface to allow communication between friends. While this is not developing anything revolutionary, this is a staple to any social media application. Diving deeper into simply the UI aspect of these two components, there are several things that must be taking place in the background to ensure user safety, as well as functionality. This is one of the places where hardware must be introduced into our project. We need to be able to track user data, using GPS location, and relay this to the application. GPS location can be tracked using a microcontroller, and using a modem, this information is able to be sent to our SQL server. These two parts combined with integrating with the UI will be key to providing a functioning 'Find my Friends' like feature on our social media application. Now that the issue regarding friends, finding one's friends on the application, as well as communicating with those friends has been solved, the main issue remaining is arguably the most important problem we aim to solve. That is how to find those friends for your dog. It is much more than simply finding friends for one's dog. This component is the solution for understanding users' dog's behavior and interactions when their dog is out in the world. If users were able to understand when their dogs have positive interactions with other dogs, then of course these users would like to ensure their dog becomes friends and continues developing these positive connections and interactions. Furthermore, it is just as important for dog owners to be able to see when their dog has had a negative interaction, to be able to then recognize and avoid those types of interactions in the future. This is where the use case for our Machine Learning Model integrates with our hardware component, the dog collar, and furthermore is brought in together with our back-end servers to display for the user on our UI. This Machine Learning Model will be able to classify these interactions, which comes in from microphone data. Using this audio consisting of dog barks, the model will be able to identify positive interactions. By first training a model based on a large set of dog barks, the model which has a planned accuracy percentage of 94%, is aimed to tackle this issue of classifying these interactions. This connects to the hardware component as our microcontroller will leverage a microphone. All in all, by being able to understand one's dog's interactions with other dogs and allowing methods to continue engaging with those dogs that have positive interactions, we provide the Dog community with a robust, multi-purpose application. (Daniel)

2.3 Other potential concepts

Initially, we planned on having a simple product in which there are posts on the social media platform. Users would simply interact based on posts from their friends, like other mainstream social media applications such as Facebook, Instagram, and Twitter. However, we decided to modify the project to not include posts, but to instead include a SnapMaps like functionality to display users' friends' whereabouts. With such an interface, it would be ideal for users to open their application and physically see where their friends are, so that if they are at a nearby park or dog gathering, they can join in. We could have simply output the address of the other users, but this would add less to our interface. This also adds a layer of uniqueness that sets our app apart from other traditional social media platforms that dog owners currently use. Additionally, our original implementation solely included a tracking feature, which will still be included, but we now currently have a way to recommend friends for users on the app based on microphone data that gets sent. Previously, we considered a method of making recommendations based on how much time dogs spent playing with eachother. We would have analyzed how long the GPS trackers spent near eachother, and if the distance limit and time thresholds were met, we would make recommendations. Unfortunately, training such a model would have required us to use YOLOv3 and video processing along with object tracking in videos, and because of several constraints, we decided to not pursue such a model. We realized that it would be much more realistic to train a base model that solely utilizes audio data. We were also going to make a model based on the dog's motions, and our recommender system would have used motion associated with positive interactions as additional features. But training this base model would also face similar constraints as the proximity model. Training on videos requires normalized environments and labled videos for what the interaction in the videos represent (friendly vs. hostile). We did scrape this motion-based model, but we can still provide accelerometer information and make inferences based on the accelerometer motion to inform owners what their dogs are up to outside of the owner's eye sights. While we could have simply created a tracking device for dogs, we now have other data that dog owners would love to have to ensure their pet is content. (Rajiv)

3 SYSTEM DESCRIPTION

Our project architecture is divided into three different components: Hardware, Back-end, and Frond-end. The Hardware will be a dog tag that records acceleration, audio, and GPS location. All this data will be sent to our Back-end SQL server with the help of an IoT modem. From there, the audio data will be cycled through Amazon's S3 storage service and SageMaker, Amazon's Machine Learning Tool. As of now, the audio helps train our model to determine if noises are dog barks vs not dog barks. We will develop a second model to determine types of barks and predict what the dog is feeling in that instance. This data will help recommend friends to dogs who have positive interactions, and these recommendations will end up back in our SQL server.

All the sensor data and recommender data stored in the SQL server will be sent to our front-end, a React Native mobile application. The Location and Instant messaging back-end will be handled by Scaledrone, while our user profiles back-end will be handled by Firestore. Any messaging instances will be hosted in Scaledrone, and GPS information will be transferred from SQL server to our RN app and then to Scaledrone. Scaledrone will be able to display live locations for our React-Native Maps interface. Any time the user adds a new friend on the front-end, their profile will be updated in Firestore to reflect the new information. (Justin)

3.1 Hardware Dog-Tag

The hardware required for this implementation is a sensor to collect accelerometer data (accelerometer), a sensor to detect audio (microphone), a GPS module, a BLE, an IoT Modem to send data and a microcontroller capable of utilizing all the sensors. The data will get sent via the modem to a server, where that data will be processed and fed into the recommender system. The microcontroller that is optimal for doing so is the STM32- a microcontroller capable of capturing all required sensor data. The modem that will be used is the industrial IoT wireless mega modem, which can transmit its own Bluetooth signal. As such, there will be no need for a separate BLE module, and data can be wirelessly transmitted to our web server by the same system that detects nearby dog-tags. Initially, we were considering using a filter for the microphone, so that we would be able to filter out noises that are not barks to be processed. However, taking into consideration the different types of dogs that exist, it is difficult to filter out sounds without filtering out some of the noises made by the dogs themselves. We initially thought that dogs would emit frequencies within a certain range but seeing as there are many types of dogs in the world, such a range would not exist. Dog barks can assume any frequency that humans can perceive. (Rajiv)

3.2 WebServer

Uploading sensor data to a SQL server instance from the microcontroller is performed by connecting to the SQL server's respective endpoint and port number. This connection is not performed directly within the live application, as that imposes inherent security risks with a MySQL user account remotely accessing the database. Instead, our microcontroller will make an HTTP request to a web server, which responds by running the script described in section 4.5 to get data, process it, interact with the database, then return the result to the RN app as an HTTP response [5]. The SQL instance will be instantiated in the cloud, so for either the small-scale or large-scale option in section 6, the server has the ability for 1,000-80,000 input/output operations per second [6]. Access times for read/write speeds are in the order of a few milliseconds. It will be read on the mobile application by connecting to the same endpoint and port number as is accessed when writing to the server from the microcontroller. To differentiate sensor data between users, each table of sensor data is grouped by each BLE sensor's ID number. When two BLEs come in close enough proximity to pair, their ID numbers are exchanged such that paired UIDs can be stored in each UID's list of paired UIDs. From each UID's list of paired UIDs, accelerometer and microphone data can be compared between users by looking up each table of sensor data by the ID numbers listed in the paired UIDs list. For every addition to the list of paired UIDs, the newly paired UID's sensor data is searched, such that active comparisons between each SQL table of a UID's sensor data may provide a better understanding of each UID's behavior. There will also be a smaller list containing all paired UIDs which are added as friends. (Chase)

3.3 ML Model in the cloud

There would have to be a machine learning model making recommendations in real time. In order to accomplish such feat, we would need to have pre-trained a base model to classify dog-barks as friendly vs. hostile. First, we will need to identify which part of an audio clip is considered a dog-bark. This feature has been completed already. Next, we will utilize AudioSet's Dog library to classify each bark into one of 7 different classes. These classes include "Bark", "Yip": two neutral kinds of noises, "Howl", "Bow-wow", which are more communicative noises and indicators of more positive interactions, "Growling" and "Whimper" which are indicators of more negative interactions and finally, "Bay" which is the sound made by hounds on the scent and could also be considered neutral. Once we have trained a model to classify noises into these labels, we will have to export the model to the Sagemaker cloud instance. Our training will happen on BU's Shared Computing Cluster to save money with computing power usage. Sagemaker solely exists to receive audio clips from S3 and export recommendations to our SQL server. The job of associating each audio clip with a specific user is completed by our SQL server. Once recommendations are received on the SQL server, they are sent out to our React-Native application to be displayed as a suggestion. Any negative interaction will also be exported by our model to the SQL server, and these negative interactions will also be displayed on the history page of our front-end. (Nafis)

3.4 Application Interface

The Application Interface we plan on providing will be centered around a React Native mobile app. The app will feature email authentication with sign up/login capabilities interacting with Firebase for profile storage. We will have a UI page dedicated for user's profiles. They can add a profile image for their dog, and 'about me' fill out a section, and other miscellaneous details such as weight and breed. All of this will be sent back to Firebase. We will create a Settings Page for editing app settings as well as the Profile Page.

In our UI, we will also have a 'Friends/Suggestions Page.' As the name suggests, this page has two components. The friend's page is fairly straightforward, Each profile will have a list of friends that the user has already connected with and added as friends. The list will be displayed on this page using ScrollView for people with several connections. The Suggestions component will be driven by the data from the ML Model through our databases. The ML Model does all the determination on good/bad interactions, and our Suggested Friends component will simply present users' options to befriend users with dogs that had good interactions with their own dog.

There will be a 'Requests/Search Page', where users can search up different users and their dogs, even if they have never interacted before. This page will simply check our database to see if such a profile exists. An additional 'History of Interactions Page' will be added so users can keep track of their dog's interactions with other dogs. The Application Interface will handle displaying this information.

We will also be providing a simple 'instant messaging feature' which will allow users to message other users. (Justin)

3.5 Maps and Database

The map will store its coordinates in a list within each table of the SQL server. Each table is grouped by the user's BLE ID, but instead of how accelerometer and microphone data is compared between all users who have paired BLEs, location is only compared between users who are added as friends. Thus, to update the map's list of friends' locations, each friend appears on the map by the lookup of location associated with each friend's BLE ID. This association will be updated in Scaledrone, as each user that subscribes to a channel must consist of their own respective ID for Scaledrone as well. As each respective user receives its authentication JSON Web Token when trying to connect to the channel, and from there its location will be updated from the SQL server, furthermore updated in Scaledrone, and ultimately updated in the map component. Calculations for the cost of the back end of our application in section 6 assume that accelerometer, microphone, and location can update every ¼ second. (Daniel)

4 FIRST SEMESTER PROGRESS

4.1 Ability to send data out of ESP32

This semester, we proved that we could work with microcontrollers to obtain BLE data as well as accelerometer data. The sensors working allowed us to send RSSI and accelerometer data through the serial monitor in real time, in a way that is understandable. As a part of the dog tag, we can use this data to make predictions and recommendations about where the dog has been and based on its behavior in particular locations. We were also able to connect our ESP32 to the internet. Doing so would allow us to send any sensor data that is processed by the ESP32 to a server in which we can further process the data. (Rajiv)

4.2 ML model trained + uploaded to SageMaker

I tr We trained a Machine Learning model to classify noises into dog barks. To train this model we first downloaded the UrbanSound8K dataset that contained 8732 naturally occurring sound excerpts and 10 classes of labels to choose from for each clip. We extracted the audio file properties for each of the samples and analyzed the difference in sample rates, bit depth and number of audio channels. We standardized these audio file properties using the python's Librosa library. We used Librosa's load functionality to convert the sample rate to 22.05 kHz and normalized the bit depths to range between -1 and 1. The load() function also flattened the audio channel to mono. To extract features, we generated Mel-Frequency Cepstral Coefficients (MFCC) to visualize the spectrum of frequency for each sound. This image representation of audio files helps to create an image classification model. We used Librosa's mfcc() function to do this conversion. We used sklearn.preprocessing.Labelencoder to encode the categorical text into model-understandable numerical data. We used sklearn's .model_selection.train_test_split to split the data into 80% for training and 20% for testing. We constructed a sequential model with 2 dense layers, each followed by relu activation and 0.5 dropout. The model had a training accuracy of 92% and a testing accuracy of 86%. We wrote a print_prediction function to then display the result of running the model on individually downloaded files, and personally tested if the model worked.

The next portion was to make this model able to run on a cloud to be used during runtime. For this functionality we first saved the model using keras.model.save into a model file. This pretrained model could be trained on BU's Shared Computing Cluster and loaded on any system. This is useful for when we train the dog-bark classification model on the Cluster. We utilized Amazon Sazemaker to be the cloud interface. We created a sagemaker notebook instance after getting an AWS account. We used the free t2.medium instance for the notebook. Once loaded, we used keras.models.load() in the cloud to load the model and display that it still worked in the cloud.

4.3 Mapbox and RN Maps

There were several steps that went into the development of a Maps component for this project. For a majority of the first semester development of this Maps component was done with MapBox. This was supposed to be a way to add customizability into our project and truly create a friendly UI environment for users to interact with. By the time of the prototype, this MapBox component included iOS development that was able to capture user location in real-time. This was ready to be replaced by hardware GPS location once that was ready. Furthermore, the MapBox component was able to track coordinate location, and users were able to pin their favorite locations on the Map using text input. These functionalities were displayed for the prototype and the uses were all met. There was a critical issue found with MapBox that only came to light as the semester moved on and development on the MapBox component slowed down. This is where the shift to React Native Maps paired up with Scaledrone was introduced. There was a clear whole in design for a secure and authenticated way to store and display our user's location and messaging information. This is where Scaledrone was introduced. Scaledrone is a real-time safe and secure message service. This service takes care of everything on the backend and simply allows for development of what developers care for, the actual application. This combined with the fact that MapBox documentation for React Native was not up to proper standards, created the need to shift as quickly as possible. This leads into the most updated progress by the end of the first semester. I chose to proceed development using bare React Native Maps, as there is sufficient documentation and the functionality needed would still be able to be captured and implemented. This map component using Scaledrone and React Native shows a basic development of what would be 'Find my Friends', which is the main functionality being worked towards the final project. The application can track the iPhone simulator's location, as well as data sent through a node server. This data includes generated friends' locations that move on a fixed time interval. This movement is clearly depicted in the application as the friends' pins are clearly seen moving. This shows Scaledrone's capability to act as a secure real-time service to store location data and furthermore display these locations on a map. The flow of data between the users, the authentication server, as well as Scaledrone can be explained as follows. The React Native Application sends the authentication server the user's name. This generates a JSON Web Token. The application can then connect the user and subscribe them to the respective channel. This channel consists of any friends that the user has. Of course, users do not want to share their information with random people. This is where Scaledrone allows us to keep their information secure and updated to only those specified friends. The aforementioned JSON Web Token then connects the user to the application. Friends' locations are furthermore sent to Scaledrone over the server and displayed on the main Map Component. From there, Scaledrone can recognize all users in the room. Having this 'Find my Friends' capability working with a local node server is a great step in the process to what the final product should consist of.

4.4 User Interface with Authentication

Over the course of the first semester, our user interface was developed in React Native using Expo. Our preliminary GUI features a sign-up page, as well as a login screen. Both display a nice dog image and a simple header title for showing which slide. Each screen features two textboxes. The top one is to enter a user's email, and the bottom one is for a password. If a user signs up with a new email, our code utilizes a simple Firebase authentication that will remember these login credentials. In order to successfully login using the Login page, your email and password must be previously registered to Firebase through our sign-up page. Similarly, if you have already signed up with an email, our sign-up page will prevent you from signing up with the same email.

After logging in, the preliminary GUI presents a main page, with a simple welcome text and some navigation buttons. More importantly, part of the welcome text includes the email you used to sign up or login with. This shows that we were able to pull the email out of Firebase and back into our UI. All these features were successfully tested during our First Prototype Demo.

The mobile application also features a functional navigation, where we could toggle between different screens with integrated buttons. We did this by importing StackNavigator from React Navigation's stack library. In the First Prototype Demo, we showed successful navigation with the integrated buttons throughout the application.

4.5 Mapbox and RN Maps

We demonstrated uploading to an Amazon S3 bucket with both the AWS CLI and the AWS SDK. The AWS SDK was shown in the video because it will be how we chose to implement uploads being automated for folders or for files. Uploads to S3 will be performed on the webserver, which runs the node script shown in the video in a folder containing all the latest files of sensor data for upload. It will synchronously access reading each of these files, and upload in the matter of milliseconds such that sensor data can be fed into the machine learning model in real time. The node script for S3 uploads will be executed by the web server running on the microcontroller, with the hardcoded path for upload containing all the latest files of sensor data. It goes off the documentation for logging into the AWS developer account but is custom for the process of uploading for each file in the given directory as the 4th command line argument. The team already requires that there is a script ready to upload files or directories in the form of sensor data from the web server to the SQL server for saving, and ready to upload to S3 for live usage in the machine learning model. So, it is important we have already completed the step of uploading sensor data in this script which the HTTP web server on the microcontroller will run to add a layer of security to accessing S3 or the SQL database. This is because accessing the database directly within the application produces more memory overhead and allows the security risk of exposing the ability for SQL user accounts to remotely access the database. Instead, we planned for the group's intent for the user's device to establish a secure connection to the back-end of our database. (Chase)

5 TECHNICAL PLAN

5.1 Hardware: (Rajiv)

Over Winter Break, we plan on ordering the ncd IoT mega modem, as well as acquiring the IIS2DHTR accelerometer sensor so that once the semester resumes, we can send the required data through the modem to the server. Specifically, the mega modem described before should be able to collect all necessary data.

Task 1: Set up modem and connect all necessary sensors. Chase and Rajiv have obtained specifics of how data is sent to SQL server.

Task 2: Obtain STM32 microcontroller. Collect sensor data from accelerometer, microphone. Send this data to the modem.

Task 3: Obtain power supply for microcontroller. Obtain coin sized lithium 3-volt battery that can charge the controller for a consumer who will utilize this product over a long period of time. This battery should meet specification for weight, battery life, and heat dissipation. Since the STM32 requires 5 volts to be powered, a relay will be used to give the coin battery an additional 2 volts.

Overall Milestone- The milestone to accomplish in terms of hardware is successfully sending data to the SQL server. We must consider different types of compatible hardware. Though we understand the data flow as well as the necessary modules, the biggest milestone would be to connect these modules properly. We also ideally want to ensure every sensor can connect to a wireless microcontroller (STM32), and that data can continuously be sent once the BLE of the modem is activated. The BLE of the modem will serve as an indicator to when the modem will begin to send data to the web server. If the BLE is powered off, then data is not sent, but is still being collected. Data should cease to be collected once the battery dies. The BLE itself must be able to detect other BLEs, and as such, each BLE must be capable of obtaining a unique user identification from the other, so that a friend recommendation can be sent. This will be accomplished by obtaining the address of the other user's BLE (each BLE contains a specific public global fixed address), and then obtaining the user data associated with that address. This will be a 48-bit identifier that is unique to each BLE device, and as such, each user will have their own.

5.2 WebServer: (Chase)

Task 1 is to indirectly authenticate to the AWS SDK so that our login information is not exposed in using the application. Then, Task 2 is to upload files using the AWS SDK. This is relatively simple, there is plenty of documentation online to use a filename as a command line argument to upload to an S3 bucket. It took more time to go through uploading folders, which is Task 4. It ended up being performed with a for each loop selecting each file within the directory given as a command line argument and then proceeding to upload each file. Task 5 is to authenticate to MySQL in this same node script to upload files or directories to S3. This should be relatively easy, we already have bookmarked code to connect to the SQL server's endpoint and port number, other than that, it is just a matter of creating a master password on the server. Next, Task 6 is to perform our first SQL queries, which requires a connection to the HTTP web server installed on the microcontroller. This is because for a select statement to select all lines from each file containing sensor data from the accelerometer, microphone, and GPS, the web server must be able to execute the node script with hard coded filename or folder name parameters. If this works to upload to S3, it must work for SQL as well, we will be using Cloud SQL as noted in section 3.2 because it removes the memory overhead of accessing the database on-premises of the device running the application. Once this is in working order, Task 7 is to add a NOSQL HandlerSocket to increase the throughput of the application. This is because although HandlerSocket is on-premises, it's a NOSQL solution, meaning no memory overhead, and very high IOPS (750,000). Thus, adding it along with the baseline

AWS RDS MySQL used in Tasks 5-6 gives a total average throughput of around 300,000 IOPS. Task 8 is necessary if we scale from 1,500 users to 10,000 users. Task 8 involves replacing the AWS RDS MySQL server in Tasks 5-7 with PostgreSQL for Amazon EC2. This gives an average throughput of closer to 400,000 IOPS, considering Amazon EC2 PostgreSQL for complicated queries and HandlerSocket for simpler queries.

5.3 ML model: (Nafis)

Since we have already developed a model to classify noises into dog_barks along with other kinds of labels, we will need to convert this model to become a binary classifier. Any classification that is not a dog_bark does not need to be separated into its own class. This classification model will need to extract the section of the audio that consists of the barks and this isolated clip will be fed into a second model that we are developing.

We will need to first extract our dataset from AudioSet's website. There are 13000 annotations in the dataset we would like to use, but the data is in the form of videos. The first task for this section is to extract only the audio from all 13000 of these videos. Once we have a new dataset consisting of just audios and labels, we will proceed to training our second model. The training process is like before, where we load the clips with Librosa's Load () function to normalize the audio clips. The standardized format will take care of clips if they are too loud or too quiet. The next section is to use various dense layers followed by activation functions and use a soft-max feature at the last layer of the model. We will have to experiment with a few different kinds of layers during training to reach our desired accuracy.

The training will happen on the BU SCC. Once trained, we will export the model out to Sagemaker, and set up Sagemaker to consistently be listening for audio inputs from S3. The Sagemaker model will send its outputs directly to SQL, and the inputs and outputs will both come with identifiers, so we know the source of the audio clips. It is worth mentioning that Sagemaker will contain two models and an audio processing unit in between the model that isolates sections of audio that contain dog_barks. Sagemaker will be free so long as we do not store any of the clips on the cloud, but once utilization reaches above the 10 Gb limit, we would need to consider paid options. For development purposes, this will not be a worry.

5.4 RN Maps and Friends: (Daniel)

The plan moving forward for some of the main aspects of the Social Media component, is to continue development on the Maps component, develop a Messaging component, and furthermore integrate all of these with the UI. This timeline for this plan would be an overachievement for the semester, however this was done on purpose to ensure there was slack time in the development process, as well as integration process. Therefore, there would be more than enough time to fix issues as they arise. The first part of the plan involves integrating the current React Native Maps functionality alongside the current working UI for login and signup. Ideally the application would have been developed for iOS, however many issues during the semester have led our team to now pursue Expo as the main host of our social media application. Once the current maps component is integrated with the current stage of the UI component, development of the messaging component will be able to start. This is going to be very similar to development of the map component. To recap Scaledrone again, Scaledrone contains functionality to create different channels, and different channels will be hosted by different friends. Once friends subscribe to a channel, notifications can be sent out to everyone in the channel. The concept of these notifications is how maps locations are currently being updated, and it is also how messages will be sent out to users in the chatroom. These channels are otherwise called rooms. This development will start from first creating a basic separate working component, where different users will be able to connect to a room and from there, send messages to each other over the channel. Scaledrone ensures safety due to the use of generated JSON Web Tokens to ensure functionality. From here this messaging component will be taken a step further as it is integrated into the application. This will include real users joining the channel and sending messages to one another. Now that there is the chat functionality in place, the final step is updating the Maps component to take in live user data through the SQL server and update the location in Scaledrone to be displayed on the application. The reason for the delay in this development is to allow time for the hardware component to first capture this GPS data, and furthermore allow capability for this to be sent and stored using SQL. Once this is ready, the plan is to remove these generated friends' locations and update the friends' channel currently in Scaledrone with real-time data. From here, the functionality for maps and messaging should be complete, an integral part in creating this social media application and connecting these dogs and their owners with each other.

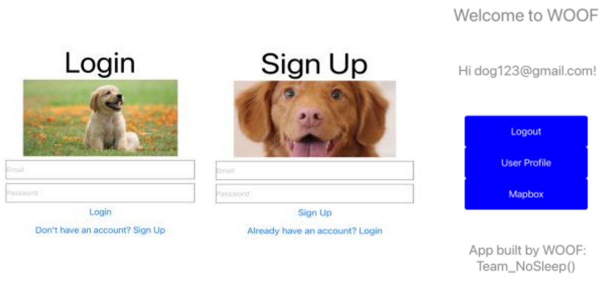
5.5 Plan for User interface: (Justin)

There are a couple of unfinished GUI deliverables that need to be completed by the end of March. The plan is to complete verifying authentication in the first half of winter break. We have a simple email/password Firebase Authentication, but it would be more professional if we achieved google authentication alongside our preliminary login GUI.

The next step is to finalize the navigation page. Our preliminary StackNavigation works fine, but it uses some expo dependencies. The goal is to try to have 2 versions, one with and one without the expo dependencies. With these 2 options, testing different integration with Maps and Instant messaging will be much easier to add to our UI. The goal is to achieve this by winter break

After this, Integrating the UI portion of Messaging into our app will be the next goal. This may take a considerable amount of time, since it requires the backend of Messaging to be already completed. This task will most likely be juggled with other tasks while the backend database gets completed.

After winter break, the GUI of our Pages relating to the user's friends will be developed. Once the backend is proven to be able to store and append to a user's friend's list, the first step will be to simply be able to retrieve that list and display it onto the mobile application. There will probably be an intermediate step where we try to display this to a computer console. At the same time, the UI will need to also be able to search for specific profiles in our search page. Again, we will first test functionality of front-end/back-end communication through a terminal before completing the GUI. With this functionality, the History of Interactions UI page should not be too difficult to implement. All of this contains the UI workload of February and March.



6 BUDGET PLAN

So far we have spent $75.50 on the MPU6050 accelerometer sensor, the Adafruit Bleu-art BLE module, jumper wires, the Arduino Uno, and the MAX4466 microphone. We plan on utilizing our full $1000 budget. We will spend $199.95 to for each mega IoT modem by ncd products. We will spend $5.00 per IIS2DHTR accelerometer sensor, and $7.00 per CGIP.25.4.A.02 GPS module. We plan on buying 2 of each of the items above, due to the necessity of having two modules to interact with each other. We will also need to order 2 STM32 microcontrollers, each costing $58.80, and an additional Adafruit Bleu-art BLE module, costing around $17.50. We will also spend $10 on a relay and lithium coin cell batteries. This leaves us with $285.40 of our budget left. This money will be used to pay for our software development costs, including
training our machine learning model using 4 cores and 2 cpus on the scc ondemand cluster, which costs $200 per month with ~60 hours of usage, but is free through our student account access. For S3 storage of sensor data, S3 standard costs $.023 per GB for the first 50 TB per month. Requests and data retrievals from this bucket are $.005 per 1,000 requests. With an initial maximum of 250 users logged in to the application at once, given three sensors running concurrently every .25 seconds, for 12 IOPS per user, 3,000 total IOPS per 250 users logged in at once costs $.015 per second comparing all 250 users sensor data at once. MySQL on AWS RDS managed to have the right size for phase 1 of launching our application to a smaller set of 1,500 users, it supports 750 instance hours per month, 20 GB of storage per month, and 20 GB of backup storage per month. So, this supports 250 users spending three hours on the application per month. Scaling our application to phase 2 of launching it to a large set of 10,000 users would first involve using a NOSQL HandlerSocket for simpler queries and keeping AWS RDS MySQL for complicated queries. This scales our IOPS to a combined 100,000 per second. If this is not sufficient, we will scale to PostgreSQL for Amazon EC2 to consistently allow a baseline 30,000 IOPS, which is $.09 per GB for the first 10 TB per month, and about $.075 per hour per 10 GB instance and add a NOSQL HandlerSocket as needed to ensure there is enough maximum throughput for reads or writes. In full, the software for the phase 1 small scale 1,500 users will cost $.023 for 10 GB on S3, $5.18 per 24 hours straight of

Item	**Year**	**Description**	**Cost $**
Sensors and microcontroller	F’21	MPU6050 accelerometer, Adafruit Bleu-art BLE, jumper wires, Arduino Uno, MAX4466 microphone	75.50
Modem	S’22	Mega IoT ncd products	199.95
Accelerometer	S’22	IIS2DHTR	5.00
GPS module	S’22	CGIP.25.4 A 02	7.00
Microcontroller	S’22	STM32	58.80
BLE module	S’22	Adafruit Bleu-art	17.50
Power management	S’22	Relay, lithium coin cell batteries	10.00
Storage Bucket	S’21	Amazon S3, 10 GB	5.20
Messaging and location sharing	S’22	Scaledrone small or medium	9/$29
Storage Bucket	S’22	Amazon S3, 66 GB	36.06
Total: hardware	S’21 &22	All hardware	644.50
Total: software	S’21 &22	All software	50.26/ 70.26
Total: total	S’21 &22	All items	694.76/ 714.76

requests to S3 per month, AWS RDS MySQL and a NOSQL HandlerSocket are free, Firestore React Native and React Native maps are free, Scaledrone is $9/month for 100 concurrent connections and 200,000 daily events, and finally SageMaker for 250 hours is free, for a combined total of $14.41 per month for 1,500 users. In full, the software for the phase 2 large scale 10,000 users will cost $1.53 for 66 GB on S3, $34.53 per 24 hours straight of requests to S3 per month, AWS RDS MySQL and a NOSQL HandlerSocket are free, Firestore React Native and React Native maps are free, Scaledrone is $29/month for 500 concurrent connections and 1,000,000 daily events, and finally SageMaker for 250 hours is free, for a combined total of $65.06 per month for 10,000 users. (Chase and Rajiv)

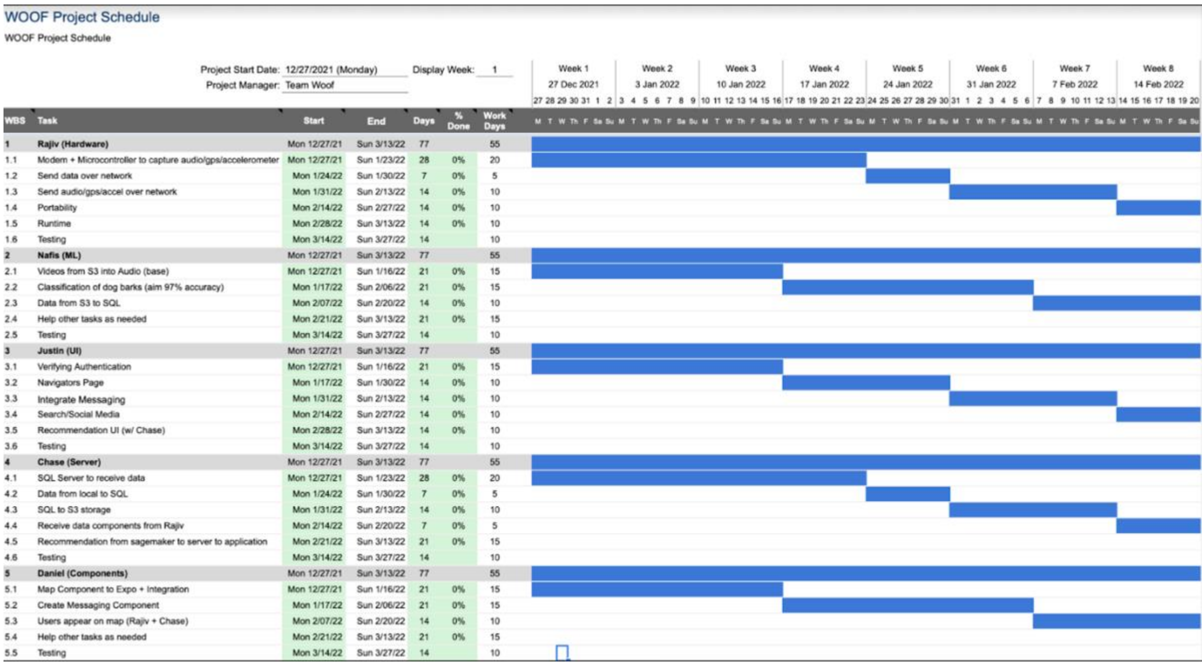
7 APPENDIX

7.1

Engineering Requirements (Chase)

Technology	Requirements
BLE	Initiates the exchange of data between dogs within 5 meters of eachother
Microphone /accelerometer	Records dogs' x,y,z acceleration in m/s^2 and audio in dB (decibals) when near other dogs
Microcontroller	Powered by battery and need to be rechargable
Mobile application	Components include sign up/login, friends/suggestions, friend requests/search, maps, history of interactions, and settings
Recommender system	Accurately suggests friends to add using runtime model uploaded to Sagemaker cloud instance
Dog Maps	Displays your friends based on their privacy settings
Database	Save/read sensor data or user profile per user, supporting real time reads/writes

7.2 Gantt Chart



WOOF Project Schedule

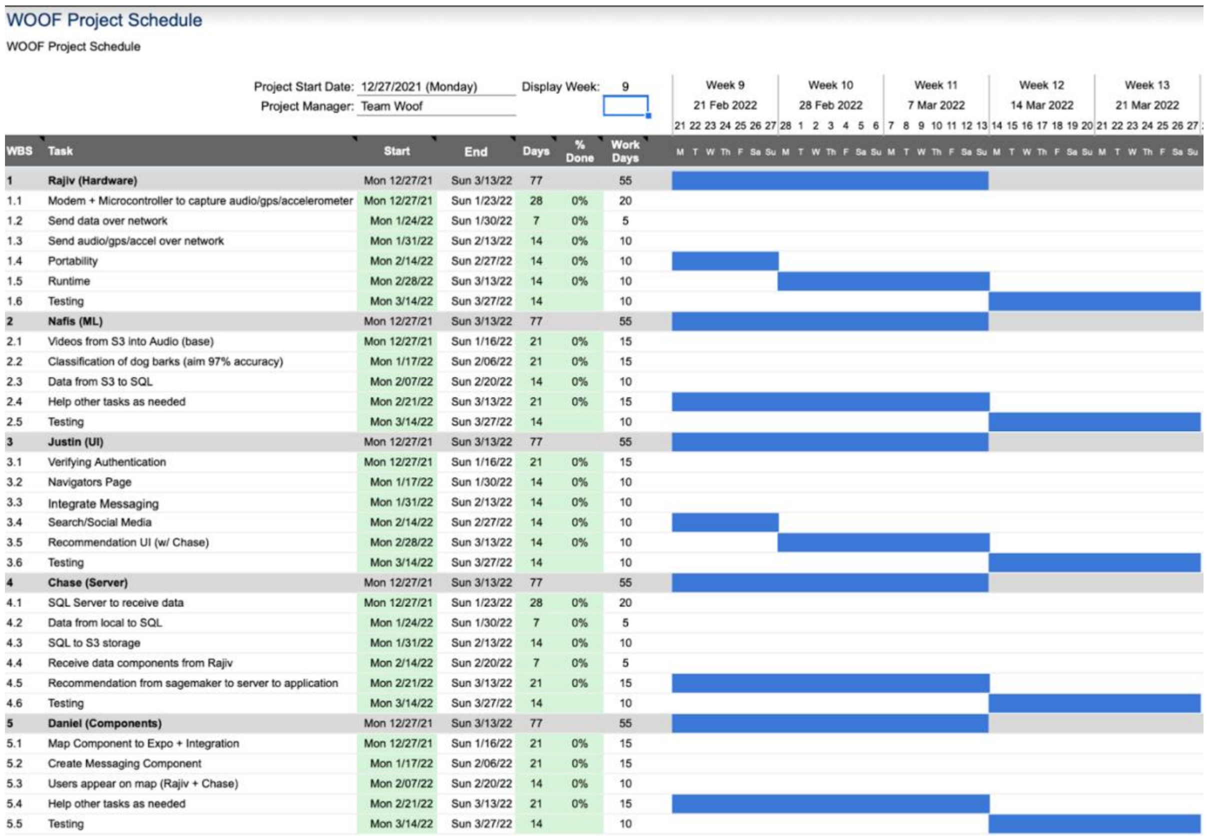
WOOF Project Schedule

Project Start Date: 12/27/2021 (Monday)　Display Week: 1

Project Manager: Team Woof

WBS	Task	Start	End	Days	% Done	Work Days
1	**Rajiv (Hardware)**	Mon 12/27/21	Sun 3/13/22	77		55
1.1	Modem + Microcontroller to capture audio/gps/accelerometer	Mon 12/27/21	Sun 1/23/22	28	0%	20
1.2	Send data over network	Mon 1/24/22	Sun 1/30/22	7	0%	5
1.3	Send audio/gps/accel over network	Mon 1/31/22	Sun 2/13/22	14	0%	10
1.4	Portability	Mon 2/14/22	Sun 2/27/22	14	0%	10
1.5	Runtime	Mon 2/28/22	Sun 3/13/22	14	0%	10
1.6	Testing	Mon 3/14/22	Sun 3/27/22	14		10
2	**Nafis (ML)**	Mon 12/27/21	Sun 3/13/22	77		55
2.1	Videos from S3 into Audio (base)	Mon 12/27/21	Sun 1/16/22	21	0%	15
2.2	Classification of dog barks (aim 97% accuracy)	Mon 1/17/22	Sun 2/06/22	21	0%	15
2.3	Data from S3 to SQL	Mon 2/07/22	Sun 2/20/22	14	0%	10
2.4	Help other tasks as needed	Mon 2/21/22	Sun 3/13/22	21	0%	15
2.5	Testing	Mon 3/14/22	Sun 3/27/22	14		10
3	**Justin (UI)**	Mon 12/27/21	Sun 3/13/22	77		55
3.1	Verifying Authentication	Mon 12/27/21	Sun 1/16/22	21	0%	15
3.2	Navigators Page	Mon 1/17/22	Sun 1/30/22	14	0%	10
3.3	Integrate Messaging	Mon 1/31/22	Sun 2/13/22	14	0%	10
3.4	Search/Social Media	Mon 2/14/22	Sun 2/27/22	14	0%	10
3.5	Recommendation UI (w/ Chase)	Mon 2/28/22	Sun 3/13/22	14	0%	10
3.6	Testing	Mon 3/14/22	Sun 3/27/22	14		10
4	**Chase (Server)**	Mon 12/27/21	Sun 3/13/22	77		55
4.1	SQL Server to receive data	Mon 12/27/21	Sun 1/23/22	28	0%	20
4.2	Data from local to SQL	Mon 1/24/22	Sun 1/30/22	7	0%	5
4.3	SQL to S3 storage	Mon 1/31/22	Sun 2/13/22	14	0%	10
4.4	Receive data components from Rajiv	Mon 2/14/22	Sun 2/20/22	7	0%	5
4.5	Recommendation from sagemaker to server to application	Mon 2/21/22	Sun 3/13/22	21	0%	15
4.6	Testing	Mon 3/14/22	Sun 3/27/22	14		10
5	**Daniel (Components)**	Mon 12/27/21	Sun 3/13/22	77		55
5.1	Map Component to Expo + Integration	Mon 12/27/21	Sun 1/16/22	21	0%	15
5.2	Create Messaging Component	Mon 1/17/22	Sun 2/06/22	21	0%	15
5.3	Users appear on map (Rajiv + Chase)	Mon 2/07/22	Sun 2/20/22	14	0%	10
5.4	Help other tasks as needed	Mon 2/21/22	Sun 3/13/22	21	0%	15
5.5	Testing	Mon 3/14/22	Sun 3/27/22	14		10

The following is our Gantt Chart (Shimon)



WOOF Project Schedule

WOOF Project Schedule

Project Start Date: 12/27/2021 (Monday)
Project Manager: Team Woof
Display Week: 9

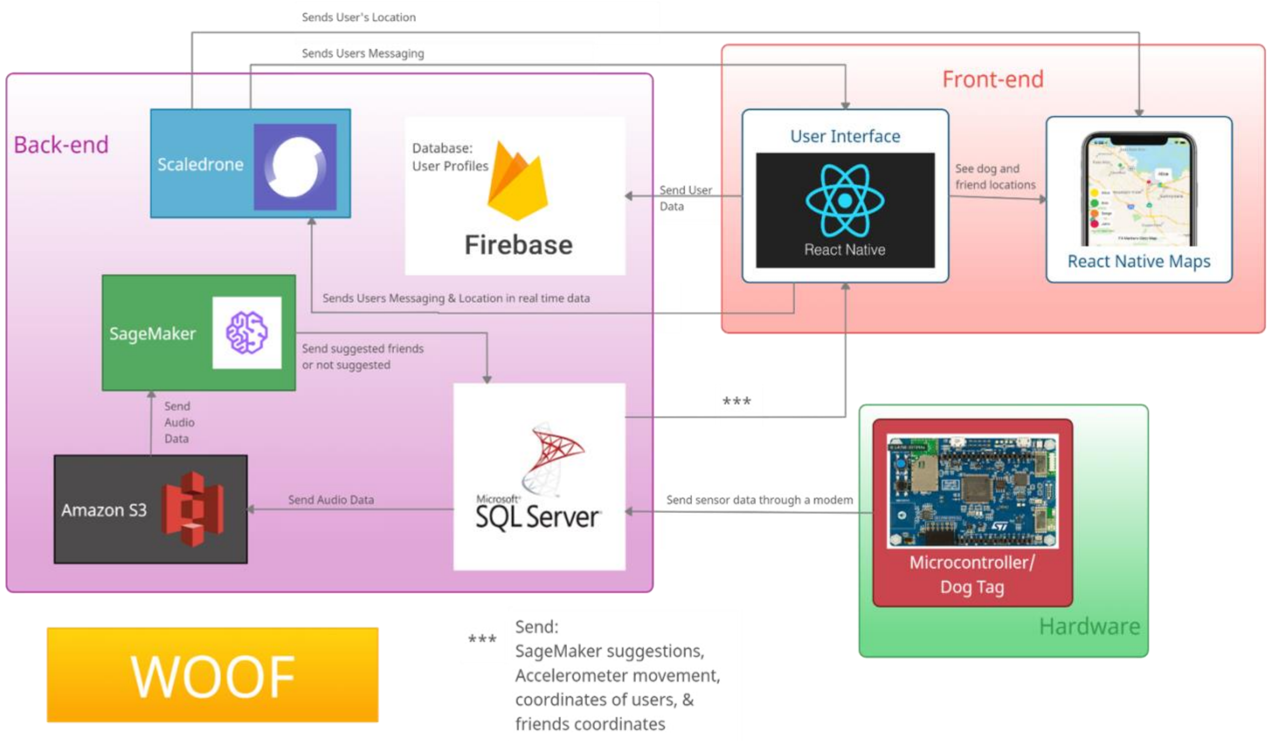
Week 9 21 Feb 2022 | Week 10 28 Feb 2022 | Week 11 7 Mar 2022 | Week 12 14 Mar 2022 | Week 13 21 Mar 2022

WBS	Task	Start	End	Days	% Done	Work Days
1	**Rajiv (Hardware)**	Mon 12/27/21	Sun 3/13/22	77		55
1.1	Modem + Microcontroller to capture audio/gps/accelerometer	Mon 12/27/21	Sun 1/23/22	28	0%	20
1.2	Send data over network	Mon 1/24/22	Sun 1/30/22	7	0%	5
1.3	Send audio/gps/accel over network	Mon 1/31/22	Sun 2/13/22	14	0%	10
1.4	Portability	Mon 2/14/22	Sun 2/27/22	14	0%	10
1.5	Runtime	Mon 2/28/22	Sun 3/13/22	14	0%	10
1.6	Testing	Mon 3/14/22	Sun 3/27/22	14		10
2	**Nafis (ML)**	Mon 12/27/21	Sun 3/13/22	77		55
2.1	Videos from S3 into Audio (base)	Mon 12/27/21	Sun 1/16/22	21	0%	15
2.2	Classification of dog barks (aim 97% accuracy)	Mon 1/17/22	Sun 2/06/22	21	0%	15
2.3	Data from S3 to SQL	Mon 2/07/22	Sun 2/20/22	14	0%	10
2.4	Help other tasks as needed	Mon 2/21/22	Sun 3/13/22	21	0%	15
2.5	Testing	Mon 3/14/22	Sun 3/27/22	14		10
3	**Justin (UI)**	Mon 12/27/21	Sun 3/13/22	77		55
3.1	Verifying Authentication	Mon 12/27/21	Sun 1/16/22	21	0%	15
3.2	Navigators Page	Mon 1/17/22	Sun 1/30/22	14	0%	10
3.3	Integrate Messaging	Mon 1/31/22	Sun 2/13/22	14	0%	10
3.4	Search/Social Media	Mon 2/14/22	Sun 2/27/22	14	0%	10
3.5	Recommendation UI (w/ Chase)	Mon 2/28/22	Sun 3/13/22	14	0%	10
3.6	Testing	Mon 3/14/22	Sun 3/27/22	14		10
4	**Chase (Server)**	Mon 12/27/21	Sun 3/13/22	77		55
4.1	SQL Server to receive data	Mon 12/27/21	Sun 1/23/22	28	0%	20
4.2	Data from local to SQL	Mon 1/24/22	Sun 1/30/22	7	0%	5
4.3	SQL to S3 storage	Mon 1/31/22	Sun 2/13/22	14	0%	10
4.4	Receive data components from Rajiv	Mon 2/14/22	Sun 2/20/22	7	0%	5
4.5	Recommendation from sagemaker to server to application	Mon 2/21/22	Sun 3/13/22	21	0%	15
4.6	Testing	Mon 3/14/22	Sun 3/27/22	14		10
5	**Daniel (Components)**	Mon 12/27/21	Sun 3/13/22	77		55
5.1	Map Component to Expo + Integration	Mon 12/27/21	Sun 1/16/22	21	0%	15
5.2	Create Messaging Component	Mon 1/17/22	Sun 2/06/22	21	0%	15
5.3	Users appear on map (Rajiv + Chase)	Mon 2/07/22	Sun 2/20/22	14	0%	10
5.4	Help other tasks as needed	Mon 2/21/22	Sun 3/13/22	21	0%	15
5.5	Testing	Mon 3/14/22	Sun 3/27/22	14		10

7.3

Flow Diagram

This is how information will flow for our project (Justin)



ACKNOWLEDGMENT

THE AUTHORS WISH TO THANK PROFESSOR OSAMA ALSHAYKH FOR HIS SUPPORT IN FORMULATING A PLAN FOR THE RECOMMENDER SYSTEM, AS WELL FOR HIS SUPPORT THROUGHOUT ALL OTHER PARTS OF THE PROJECT.

REFERENCES

[1] Rooney, Nicola J, John W.S Bradshaw, and Ian H Robinson. “A Comparison of Dog–Dog and Dog–Human Play Behaviour.” Applied Animal Behaviour Science. Elsevier, January 17, 2000. https://www.sciencedirect.com/science/article/pii/S0168159199000787.

[2] Carroll, Linda. “Your Dog May Love You, but Doesn't Love the Sight of Your Face, Study Finds.” NBCNews.com. NBCUniversal News Group, October 5, 2020. https://www.nbcnews.com/health/health-news/your-dog-may-love-you-doesn-t-love-sight-your-n1242079.

[3] Sommerville, Rebecca, Emily A. O’Connor, and Lucy Asher. “Why Do Dogs Play? Function and Welfare Implications of Play in the Domestic Dog.” Applied Animal Behaviour Science. Elsevier, September 20, 2017. https://www.sciencedirect.com/science/article/pii/S0168159117302575.

[4] Chambers, Robert D., Nathanael C. Yoder, Aletha B. Carson, Christian Junge, David E. Allen, Laura M. Prescott, Sophie Bradley, Garrett Wymore, Kevin Lloyd, andScott Lyle. “Deep Learning Classification of Canine Behavior Using a Single Collar-Mounted Accelerometer: Real-World Validation.” bioRxiv. Cold Spring Harbor Laboratory, January 1, 2020. https://www.biorxiv.org/ content/10.1101/2020.12.14.422660v1.full

[5] “Arduino - Mysql: Arduino Tutorial.” Arduino Getting Started. Accessed December 10, 2021. https://arduinogetstarted.com/tutorials/arduino-mysql.

[6] “Amazon RDS DB Instance Storage - Docs.aws.amazon.com.” Accessed December 10, 2021.

https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Storage.html