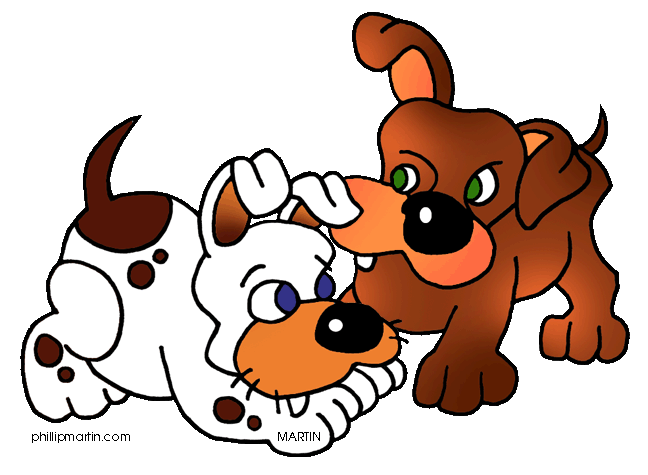
**Boston University**

**Electrical & Computer Engineering**

**EC464 Senior Design Project**

*Final Prototype Testing*

WOOF

*By: Team 02*

**Members**

Nafis Abeer [nafis@bu.edu](mailto:nafis@bu.edu)

Justin Lam [jlam8080@bu.edu](mailto:jlam8080@bu.edu)

Chase Maivald [chase1@bu.edu](mailto:chase1@bu.edu)

Rajiv Ramroop [raj19@bu.edu](mailto:raj19@bu.edu)

Daniel Shimon [dshimon@bu.edu](mailto:dshimon@bu.edu)

Testing Day: April 5th, 2022

**Content**

1. Required Material ………………………………………………….. 3
2. Demonstration Set Up
   1. Set Up ………………………………………………………… 4
   2. Pre-Testing Setup Procedure ………………………………… 4
3. Testing Procedure and Success Criteria
   1. Testing Procedure…………………………………………….... 6
   2. Measurable Criteria for Success ……………………………… 8
4. Data Flow Chart
   1. User Interface Screen Flow …………………………………….. 9
   2. End to End Data Flow ………………………………………….. 9

Required Materials

* Adafruit ESP32 Feather - microcontroller
* INMP441- microphone sensor
* jumper wires (x5)
* Python Flask - Backend ML API (Hosted on t2.xlarge ec2)
* Node JS - EC2 Server
* React Native - User Interface
* MySQL Amazon RDS - Database
* Wireless Router w/ ethernet cable - internet connection
* iphone

Demonstration Setup

2.1 Setup

Our setup consists of wiring up our microcontroller to collect audio and connecting it to the internet. Our Machine learning API is stored in a server ready to be bootstrapped. An intermediate node server is ready to receive wav files from the hardware. Our main node.js server is in a public facing EC2 server, but in a smaller instance, and it is ready to be deployed. Its security group is configured to accept connections to the private MySQL AWS RDS instance based on inbound rules allowing traffic over EC2’s elastic IP address with port 3300. Another node server is set-up on a separate EC2 instance to handle Map locations and requests. Our User Interface is running on react native on one of our member’s laptops, and it displays an iOS front end using an emulator. The UI connects to the backend through REST API calls.

2.2 Pre-testing setup procedure

We will set up our testing environment to be able to show end-to-end data flow. We will show data being collected with our microcontroller making an ML API call to receive a prediction. The ML API makes a post call with the username and prediction to our node server. The node server stores the prediction associated with the user’s identification number in the database. Our front end will perform a GET request to the back end server to display the new prediction.

* Set up ESP32 receiver
  + Wire up microphone
  + Connect to internet router
* Set up ESP32 transmitter
  + Wire up GPS module, connect to internet
  + Set up BLE to send signals
* Set up ML API
  + cd Senior/Design/model\_flask
  + ./bootstrap.sh
* Set up Node.js server
  + Connect to EC2 t2.micro instance
  + cd node
  + screen node ./server/index.js
* Set up Main Application User Interface
  + iPhone
    - build to iPhone through Xcode
  + Emulator
    - npx react-native start
    - npx react-native run-android
* Sending data
  + Set up just receiver
  + Record live audio and send directly to API via POST request from Arduino IDE

Testing Procedure and Success Criteria

Our demonstration will proceed as follows:

*Hardware/Server/Database:*

*ESP → AudioServer → ML API → Node.js Server → Database → Node.js server → UI*

1. We will first demonstrate our GPS capabilities by wiring up our module to ESP32 and displaying our Dog “Fluffy” on our Maps interface
2. We will then demonstrate an option to track our dog using the following API:
   1. <https://github.com/bachya/pytile>
3. We will then display our BLE capabilities to send integer UIDs
4. We will then show that we can record audio when integers are received
5. We will connect our ESP32 to the INMP441 via jumper wires, and connect it to a laptop containing code to record audio and upload it to our audio node server using a POST call
6. This server then invokes our FLASK API running the ML model and the model makes a prediction on the received file.
7. The ML API receives a call with a Form with “received” indicating a file has been received, and it runs our prediction algorithm on the file. It then creates a new json object with the prediction and username for another POST call
8. From here the ML API makes a POST call to the node.js server running on EC2. This call hits an endpoint which directly uploads the information passed through to the MySQL server.
9. As proof, we will show the node.js server’s console as well as the mySQL database as it is populated with the incoming information.
10. Lastly we will show the recommended user show up on the front end

*UI/Server/Database: Authentication → Follow-up registration page / Home page (if already registered) → Profile Page → Friends Page → Add Friends Page → Map Page → Chat Page → Logout of Application*

1. Show authentication functionality by registering a new account and show verification of codes to sign in (on emulator)
2. Show that the information sends to our SQL table after Registration for new account, and upon Registration, a SQL query check if the user exists
3. Show the Profile Page and how the data is pulled from our SQL server and matches what is shown on the User Interface
4. Sign into existing user’s account on iPhone (dshimon1: pre-existing state)
   1. Navigation to the home page is allowed because this user already exists in the database
5. Friends Page - shows all of a user’s friends
   1. Three SQL queries are called to prepare the friends screen, which include adding all friends who are mutually pending as friends to each other’s friends lists, deleting these new friends from each other’s pending lists to avoid bloating the table because they are no longer pending friends, and a return friends function which sends the authenticated user’s friends list to the front end
6. Show the Add Friend’s page
   1. Can search for and add friends on a button click
   2. See a certain user’s friends and add that user to your pending friends to the SQL database by clicking their name
   3. Show recommended friends to add based on previous hostile/friendly recommendations
   4. Emulator and iPhone search and add each other as friends
7. Chat Page - show chatting capabilities
8. Shows maps (both the users location and their friend’s locations)
9. Logout

Measurable Criteria for Success

1. The AWS Amplify signup sends verification code to email successfully.
2. The user interface supports signing up for an account. It supports navigation to the separate pages of maps, profile, friends, chat. It has end to end connectivity to the database, hardware, and back-end to support saving and displaying profile and sensor data.
3. The database should support storing the login information and data collected associated with each profile, showing how it has passed to and from the UI and the server, and furthermore from the hardware to the database, sent through the server to the front end.
4. The hardware should support recording audio files and uploading these files’ predictions to the back-end of the application.
5. The machine learning model should support returning real-time predictions from the hardware to the back-end, and through to the front-end. Its accuracy is such that dog barks, and related sounds can be classified appropriately.
6. The Maps component is able to render current friends locations, new friend’s as they are added, as well as supports the user’s current phone location. GPS location can be shown for the hardware if we bring the antenna outdoors.
7. The Chat Component is able to render new messages from the user and their friends as they are imputed, mimicking a real-time chat-room.

React Native App: Authentication Functionality

| **Task** | **Successful?** |
| --- | --- |
| User can sign up for a new account | **Y** |
| New accounts send a verification code to registered email | **Y** |
| User can log into an existing account | **Y** |
| User is denied if email and password are incorrect | **Y** |
| Users cannot register with a pre-existing email account | **Y** |

React Native App: From Main Menu

| **Task** | **Successful?** |
| --- | --- |
| Profile Page retrieves user’s info from SQL Database and displays correctly | **Y** |
| Friends Page retrieves a list of user’s friends and displays correctly | **Y** |
| Add Friends Page | **Y** |
| Chat Page sends messages between two accounts that have added each other as friends | **Y** |
| Maps Page shows location of user as well as locations of other friends | **Y** |

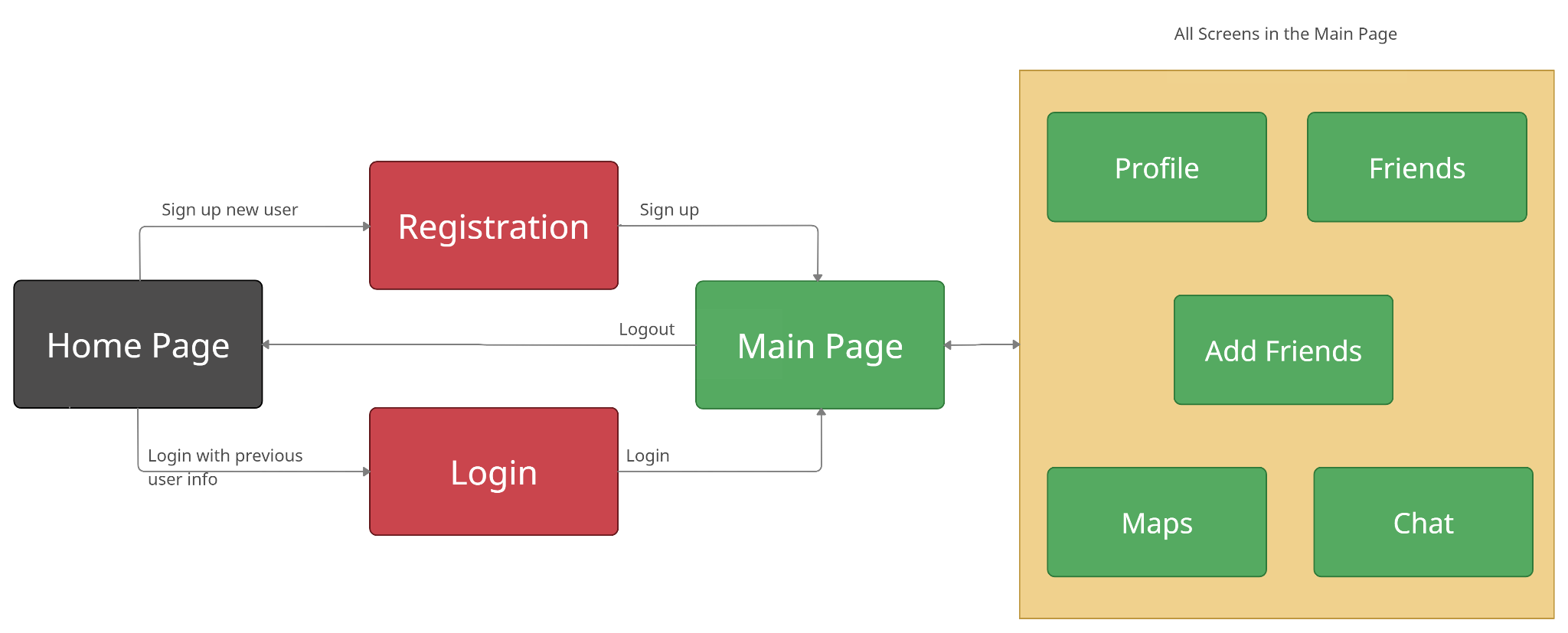
Database and Back End Servers

| **Task** | **Successful?** |
| --- | --- |
| SQL login credentials masked through an indirect combination of public and private security groups for greater security | **Y** |
| Cloud server deployed to an elastic IP to access the database from anywhere, anytime | **Y** |
| SQL endpoint which has read/write speeds to support several thousand users reading and writing from the tables simultaneously | **Y** |
| SQL functions in a script hosted on the cloud server which support reading and writing profile information, friends, and data collected from the hardware | **Y** |
| Access the machine learning API through a configuration of a cloud server’s security group and association of the server’s endpoint to an elastic IP | **Y** |

Hardware/REST calls to ML API

| **Task** | **Successful?** |
| --- | --- |
| GPS locations being sent and API receives them | **Y** |
| Transceiver BLE module sends integer and receiver records | **Y** |
| Recorded audio and sent it as POST request | **Y** |
| ML API runs prediction on received data | **Y** |
| Recommended user sent to front end | **Y** |

4.1 User Interface Screen Flow



4.2 End to End Data Flow

