Master Technical Documentation

CAMS OutSafe

Tyler Greenwood, Gregory Newman, Niranjan Varma

Link to GitHub Repository: https://github.com/gregorynewman12/OutSafe

Link to Word Document of this File:
https://docs.google.com/document/d/1FnRotjGMi1DqExgCMjEhqrHkZh1lRKj4/edit?usp=sharing&ouid=112522945967863058447&rtpof=true&sd=true

Setting Up the Production Environment

The Mosquitto MQTT Broker

Mosquitto MQTT is a service (daemon) that runs in the background on the central server. Its purpose is to accept incoming connections from MQTT devices and send them messages when a message request is received from the alert controllers.

Installing Mosquitto MQTT

Mosquitto MQTT does not need to be installed for this project, as it comes pre-bundled in the Docker container that runs the server software.

Configuring Mosquitto MQTT

Some basic configuration is needed to make the broker work correctly. By default, Mosquitto will refuse incoming anonymous connections. To ensure that the bracelets can successfully connect to the MQTT broker, edit the Mosquitto configuration file. **This has already been done in the mosquitto.conf file in the project code,** but this is included here for reference if it is ever necessary.

```
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

pid_file /run/mosquitto/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

include_dir /etc/mosquitto/conf.d

# This configures the broker to listen on port 1883 on all
network interfaces
listener 1883 0.0.0.0

# This allows connection without authentication credentials
allow_anonymous true
```

The Mosquitto broker is now running and ready to accept incoming connections and message requests. Refer to the Mosquitto documentation for configuration instructions for other operating systems.

Arduino IDE

Arduino IDE was the editor and integrated development environment used to build our project. It is capable of compiling and flashing the code to the bracelet, as well as displaying serial output for troubleshooting purposes.

Installing Arduino IDE

Install Arduino IDE with the instructions at the following link: https://www.arduino.cc/en/software .

Downloading and Opening the Source Code

Download our source code. Open Arduino Studio, click File -> Open to select a sketch to open, select the bracelet code folder and open the receiver_bracelet.ino sketch. The sketch will open in the editor. **Note: By default, all the other files in the folder will not be visible in the editor. To change this, select File -> Preferences and check the box for "Show files inside sketches."**

Installing Needed Libraries

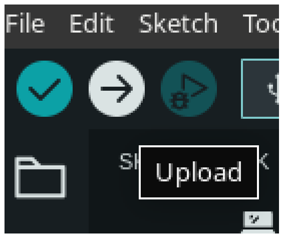
The Watchy bracelet relies on drivers for most of its built-in functions, and the correct libraries must be installed in Arduino for the source code to compile correctly.

1. Select Tools -> Board -> Boards Manager and install the latest version of the esp32 platform.
2. Select Sketch -> Include Library -> Manage Libraries, search for Watchy and install the latest version, **also installing all dependencies.** The dependencies are what we need, not the actual Watchy library itself. We want the project to compile using our modified version of the Watchy firmware, not the original source code.
3. Once the Watchy library and all needed dependencies are installed, open the library manager and uninstall the Watchy library (this will leave all the dependencies intact).
4. Finally, open File -> Preferences, and under Additional Board Manager URLs add: https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

All dependencies are now installed and the source code should be ready to flash to the bracelet.

Flashing the Code to the Bracelet

With Arduino Studio running, the bracelet will be detected when it is connected via USB. To flash the code onto the bracelet, simply click the “Upload” button in the top-left portion of the window:



Code compilation will commence and the code will be written onto the bracelet.

Note: If the watch is connected but the software says the port does not exist, this may be an issue with the port not having the correct permissions. This issue has only been encountered on Linux. On our development machine, the fix was: `chmod 777 /dev/ttyACM0`

Server

Here is how to run the server. However, we ran it using a docker file and how to run that is highlighted in the section below

1. Set up a working directory. In our case this is `/app`
2. Copy all the files from the node_server onto your work directory
3. Install all the dependencies for the server software by running: `npm install`
4. Next install, the Mosquitto library onto the working directory. Run the following commands, one line at a time:
   ```
   apt-get update && \
   apt-get install -y --no-install-recommends \
   mosquitto \
   mosquitto-clients && \
   rm -rf /var/lib/apt/lists/*
   ```
5. Copy the mosquito.conf file onto the directory
6. Open up the ports for both Node.js and mosquitto files:
   ```
   netstat -na | grep :8000
   netstat -na | grep :1883
   ```
7. Run the mosquito command to startup the server: `mosquitto -c /etc/mosquitto/mosquitto.conf & node server.js`

Docker

Our server software and the MQTT broker both run in a single Docker container, which is isolated, self-contained, and can be easily managed.

Installing Docker

Install Docker for your operating system of choice using instructions found on the Docker website: https://docs.docker.com/get-docker/

Note: Our development and testing was all done with Docker from the Linux terminal, and the instructions here will be based on that. This can be done on Windows using WSL or on MacOS with Docker, or by using Docker Desktop with a graphical environment.

Building the Docker Container from the Dockerfile

Navigate to the location of the project source code in a terminal. Then, with Docker installed and running, enter the following command:
`“docker build -t outsafe .”` (Don’t forget the period at the end).
This will build a Docker image and name it “outsafe.”

Running the Docker Container

To run the Docker container after building it, use the command:

```
docker run -d --name OutSafe -p 8000:8000 -p 1883:1883 outsafe
```

Confirm it is running by using the command:

```
docker ps
```

It should appear in the list. If so, it is now listening and ready for MQTT connections on port 1883 and connections to the server on port 8000.

Controller

The controller is the UI for the devices that are used to send alerts to the watches. The following are instructions to set up an emulator for the current version of the controller software.

Installing Tools

1. Run any Unix system, either Windows Subsystem for Linux (Ubuntu), MacOS, or a Linux distribution.
2. Install homebrew.
3. Run brew install node.
4. Run brew install watchman.
5. Run brew tap homebrew/cask-versions.

6. Run brew install --cask zulu11.
7. Install Android Studio. Check the following boxes:
    a. Android SDK
    b. Android SDK Platform
    c. Android Virtual Device
8. Install the Android SDK.
    a. Open Android Studio, click on "More Actions", and select "SDK Manager"
    b. Select the "SDK Platforms" tab and check the "Show Package Details" box.
    c. Expand the Android 13 (Tiramisu) entry, and make sure the following are checked:
        i. Android SDK Platform 33
        ii. One of the following:
            1. Intel x86 Atom_64 System Image, or
            2. Google APIs Intel x86 Atom System Image, or
            3. Google APIs ARM 64 v8a System Image
    d. Select the "SDK Tools" tab, and check the "Show Package Details" box here as well.
    e. Expand the "Android SDK Build-Tools" entry, and make sure 33.0.0 is selected.
    f. Click "Apply" to download and install.
9. Add the following lines to your ~/.zprofile or ~/.zshrc (~/.bash_profile or ~/.bashrc for bash):
    a. export ANDROID_HOME=$HOME/Library/Android/sdk
    b. export PATH=$PATH:$ANDROID_HOME/emulator
    c. export PATH=$PATH:$ANDROID_HOME/platform-tools
10. Either restart your shell or run source ~/.[your_shell_file_from_previous_step].
11. Run echo $ANDROID_HOME and echo $PATH to verify these have been added.

Running on a Physical Device

If you have a physical Android device, you can use it for development in place of an AVD by plugging it into your computer using a USB cable and following the instructions here.

Running on a Virtual Device

If you use Android Studio to open this project, you can see the list of available Android Virtual Devices (AVDs) by opening the "AVD Manager" from within Android Studio.

If you have recently installed Android Studio, you will likely need to create a new AVD. Select "Create Virtual Device...", then pick any Phone from the list and click "Next", then select the Tiramisu API Level 33 image.

Click "Next" then "Finish" to create your AVD. At this point you should be able to click on the green triangle button next to your AVD to launch it, then proceed to the next step.

Running the React Native application

Start Metro

Run “npm start” to start the bundling application. Completing the next step should run this automatically, but this is how to do it if this server does not start automatically.

Start the application

Run “npm run android”. This should launch the application in Android Studio's Emulator.

Running the bracelet software

With the correct Wi-Fi credentials *and a unique MQTT ID* (see comments in Watchy.cpp for details) flashed onto the bracelet, press the lower-left button on the watch from the main screen. Several options will appear. Press it again to select “OutSafe.” A Wi-Fi connection will be initiated targeting the credentials specified in the source code, and the watch will attempt to connect to the MQTT broker once a successful Wi-Fi connection is established. The bracelet will indicate “listening” upon success, and will begin listening for messages. If the function times out, simply restart it by following these steps again.

If changes must be made or a new test run needs to be completed, usually the best option is to reflash the bracelet code, which is the most effective way to reset the device.

End-To-End Demonstration Video:

https://drive.google.com/file/d/1WIPIVk9nbgLOByH3D0Xxc5a5GIIPtRLW/view?usp=sharing

Known Issues and Incomplete Items

1. **Watch Screen not fully refreshing:** Occasionally, when pressing the top-left back button from the options menu of the watch, the watch screen will not fully refresh back to the home screen.
2. **MQTT connection ID is not randomized:** If two watches both call client.connect(string s) with the same string parameter, only one will be allowed to successfully connect with that given ID. Each watch in the network must call client.connect() with a distinct string parameter.
3. **Server permanently sends last alert status:** Upon multiple calls to send out messages, the server remembers whatever the last alert status was, whether drill, safe, or shelter-in-place, and automatically assigns it to the new alert, even if a new kind of alert was specified. This should not be extremely difficult to fix.

4. **Watch breaks words up over multiple lines:** We attempted to write a function to prevent the watch from splitting a single word over multiple lines, but this did not end up being fully implemented.
5. **Cannot exit the bracelet software once it is running:** We unsuccessfully attempted to implement interrupts for the bracelet software, which would have allowed the user to return to the home screen at will. In this version of the bracelet, there is no way to exit the running program once it is listening other than re-flashing the code onto the bracelet using Arduino IDE.