Spring 2017

Forget-Me-Not

Daniel Barber-Cironi
The University of Akron, dab150@zips.uakron.edu

Shawn Nicholson
The University of Akron, spn11@zips.uakron.edu

Jake Kruse
The University of Akron, jek63@zips.uakron.edu

Nicole Dent
The University of Akron, cnd18@zips.uakron.edu

Forget-Me-Not

Project Design Report

Design Team Number: 7

Daniel Barber-Cironi

Jake Kruse

Nicole Dent

Shawn Nicholson

Dr. Carletta

5/1/2017

Table of Contents

List of Figures

Table of Tables

Abstract

The purpose of Forget-Me-Not is to provide another level of care and comfort to those suffering from mild dementia, as well as provide further assistance for a friend, family member, or caretaker who may look after them. Research shows that timely reminders and persistent information can greatly improve the quality of life for those afflicted with mild dementia (Mokhtari et al.). Forget-Me-Not's persistent display and wearable smart-bracelet offer a customizable and well connected system to provide these reminders. For the caretaker, a mobile application is provided in order to maintain the display and notify them of emergencies or critical events from virtually anywhere. The system is composed of four primary elements: a high contrast and easily understood display that remains in the patient's home, a mobile application wielded by the caretaker that can control aspects of the patient's display, a web server that holds persistent information for the system, and a wearable smart-bracelet equipped with attention grabbing elements to warn the patient of an ongoing alarm. The system is designed to be modular and extensible, thus leaving massive margins for expansion and future integration with trending technologies such as smart houses and appliances.

Key Features

- A persistent display in the patient's home.
- A wireless bracelet to supplement the display's alarms and notifications.
- A web server to hold key information for the display
- A mobile application for the care taker to control and monitor the display.
- A modular design, allowing for expansion in hardware and software to accommodate new technologies.

Problem Statement

Need [SN, JK, DBC]

Taking care of an aging family member can be a daunting task. This is especially true if this family member is one of the 5.3 million Americans afflicted with Alzheimer's or some other form of dementia (2016 Alzheimer's Disease Facts and Figures, 2016). Even the mildest cases require constant attention - from the seemingly mundane tasks like forgetting household chores, to more serious matters such as remembering to take medication or preparing meals. Performing a job alongside providing care is taxing, paying for personal nurses to do the job is expensive, and turning a family member over to the care of a nursing home is a last resort. Our loved ones want to spend their time in their home. They need some means to help organize and plan the day or set reminders for medication without paying excessive medical fees. Moreover, they need to do so in an easy, interactive way that they cannot easily forget or overlook.

Objective [ND, SN, JK, DBC]

A simple solution to the problem illustrated above is an interactive display positioned in a noticeable point in the person's home. The display would fetch important information and display it in a clear and concise manner for the patient each day. Information such as weather, calendar events, and even messages from the caretaker will be available right at the patient's fingertips. The display could even warn caretakers if the patient has failed to respond to an alarm or message so that they know to check on the patient immediately. The caretaker would also have the option of setting up lists and alerts and even customizing the display via an interactive application for Android or iOS. The display would help patients live a more independent life in their own homes and would quickly prove itself to be a convenient tool for the person providing care. To enhance the aid of the display even further, the patient will also have a wireless bracelet that gives off an alert light and vibration when an alarm needs to be acknowledged and it will be equipped with a "Help" pushbutton. This will be an additional reminder for the patient and allow them to be in other rooms of the house and still be notified. On top of this, the bracelet will also allow the display to track whether the patient has left the home or not. The functionality of both the display and the bracelet will be crucial for both the caretaker and patient alike.

Background [SN, DBC, JK]

Patent Search

Telecare or Telehealth Communication (PAT.NO. 9.286.442) is a system that uses a health monitoring control unit and predefined messages to ask questions or provide feedback to patients. This method of remotely providing care is similar to Forget-Me-Not in that the patient is notified of critical events in a clear and concise method that is done automatically. In this case this is done with predefined audio signals and voice recognition. The difference is that Forget-Me-Not will not use predefined responses and algorithms to attempt to administer care to the patient. Instead it will be more of a reminder and communication tool. Forget-Me-Not is going to be an internet enabled simplistic window showing only important information, not just a voice speaking to the patient.

A computer-based method and system for providing active and automatic personal assistance (PAT.NO. 9.223.837) is described. The system will use a sensor to sense data about a person or environment, then use that data along with a learning algorithm to assist the individual at home, in an automobile or on a mobile device. The patent does not specify how it will assist the patient, but it does note that a display or speaker can be used to convey information. Review of the "Other References" section alludes to the fact that the assistance will be similar to that offered by the "Siri" and "Google" personal assistants of modern smartphones. Though this system is similar to Forget-Me-Not in that it will attempt to assist the patient with daily activities, it does so in a much different manner and for a much different application. Forget-Me-Not may have some simple sensors, but these will not be used to learn behavioral patterns of the patient. The application of the display will be directed at those afflicted with dementia not general assistance while driving or on a mobile device.

Article Search

Karger Medical and Scientific Publishers, Vol. 58, No. 6, October 2012.

This article details the efficacy of timely prompts, alarms and reminders in helping elderly people suffering from dementia improve the quality of their lives. Through experimentation with television sets, tablets, sensors and speakers, researchers were able to show a major improvement in those patients suffering from mild dementia. The article focuses mainly on the type of interaction and how effective it is depending on the severity of their dementia. Researched showed an improvement in four areas of focus; information - managing the patient's agenda, communication - maintaining social links, action - controlling home devices and leisure - games, music and movies.

Computers and Human Behavior, Vol.25, Issue 3, May 2009

Researchers explore the benefits of an interactive device for creating music for patients suffering from dementia. The device takes advantage of a simple interface for creating single notes and complex chords. The purpose of this device to engage a patient to curb tendencies such as aggression, difficulties communicating and depression by eliciting creativity in the form of music. The device requires the patient to have no musical experience to create something that sounds pleasant. Participants in the study report the experience as enjoyable, even showing an increased interest in communication. Future experiments with the device plan to assess the patient's musical ability and potentially direct rehabilitation and treatment.

Marketing Requirements [ND, DBC, JK]

Marketing Requirement	Description
1	Affordable and reliable.
2	Able to display useful information in a sleek, concise, and organized manner without overwhelming the patient.
3	Accompanied by a mobile application that is intuitive and reliable so that a caretaker can easily send information to the patient using Forget-Me-Not.
4	Sized just right so that the placement of the device remains flexible, allowing the patient to place the device in a convenient, secure location of their choosing.
5	Easily portable between standard power outlets in order to support the needs of different users.
6	Paired with a wireless bracelet that adds considerable functionality to the system.

Table 1 - Marketing Requirements

Objective Tree [SN]

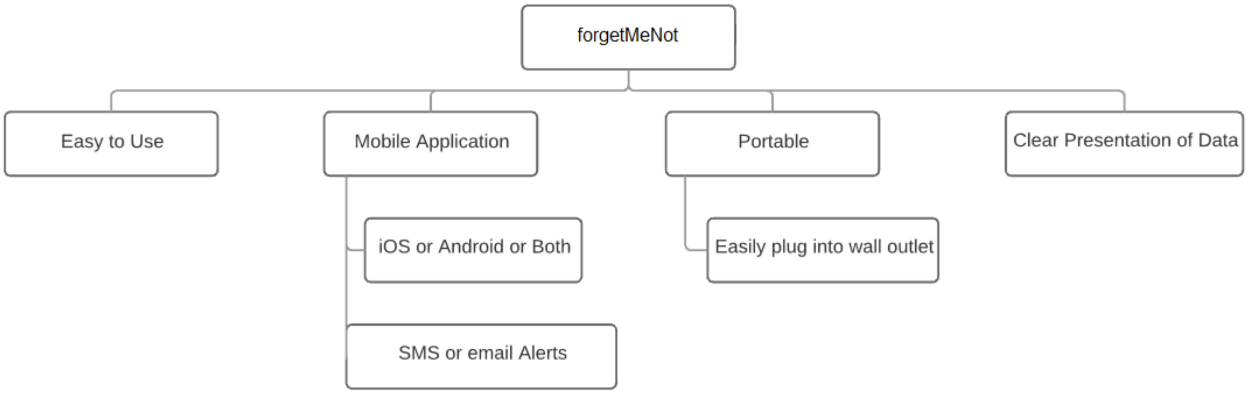


Figure 1 - Objective Tree

Design Requirements [DBC]

Marketing Requirement	Engineering Requirement	Justification
1	The display must be operational 24 hours a day / 7 days a week with maintenance downtime of no more than one hour each month.	This is an expected design trait of a system of this nature, achieved through redundancy and error detection.
2	The display system must display a clear, noticeable alarm to the patient at pre-programmed times.	Comparable to similar products targeted to this demographic.
2	The display must maintain basic alarm functionality even without an internet connection.	Based on common issue of inconsistent internet connections in most homes.
3	The screen must connect to the internet and be updated within 30 seconds of making a change through the mobile application.	This is a standard expected response time for similarly designed digital signage systems.
3	The mobile application must be compatible with Android 4.4+ as well as iOS 8+.	This covers over 90% of mobile device users on each platform.
4	The screen must be 18-22 inches in size with a bright, high contrast display.	Standard size of personal, semi-portable display screens.
5	The display and embedded computer must be powered through a single standard wall outlet.	Common standard for personal electronic devices.
6	The wearable peripheral device must operate for 20 hours between charges.	Matches the upper end of battery life for modern wearable devices.
6	The wireless bracelet must alert the patient through two separate methods when they have received a notification.	Redundancy is a key requirement in regards to the relay of critical information and alarms.
6	The wireless bracelet will have a range of 150 feet from the main display.	This is safely beyond the range that people can expect to travel throughout a personal home.

Table 2 - Design Requirements

Accepted Technical Design

Forget-Me-Not's main hardware design is broken into a small number of primary sections. These primary sections include the display itself, a small embedded computer with an operating system to drive the display, and finally the wearable bracelet. The main power supply, which will be an off-the-shelf solution, will provide power to two of the three primary sections, those being the display and the embedded computer.

Similar to the power supply, the display will be an off-the-shelf solution. After reviewing various options this seemed to meet the proper requirements pertaining to budget, size, features, and connectivity. The display will be around 20 inches in size with a high contrast, high brightness display with large viewing angles to provide the best display to the patient. It will be powered in parallel with the embedded computer from the same power source.

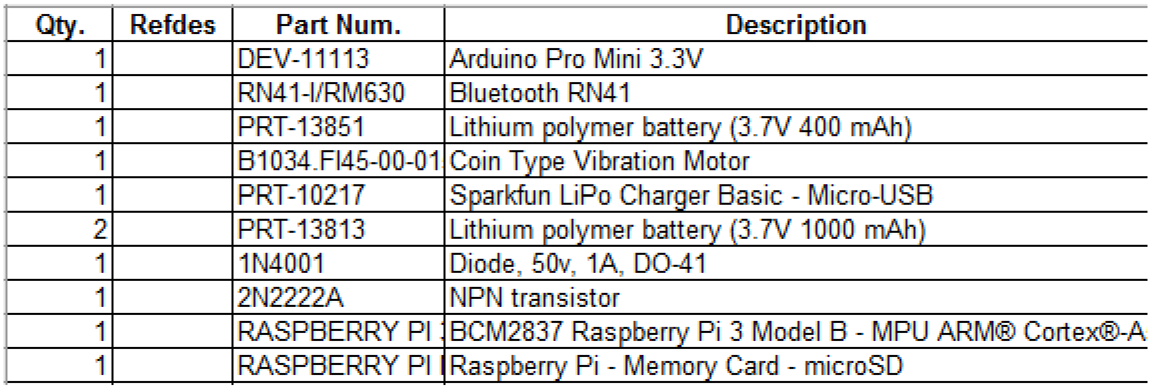
Along with the power input, inputs to the embedded computer will include: a hardwired Ethernet connection for quick communication to the cloud based server, a wireless connection to communicate with the bracelet, and a simple pushbutton in order for the patient to acknowledge and dismiss notifications as they are displayed on the screen. As for the outputs, from the embedded computer: there will again be a wireless output to communicate with the bracelet, a simple LED to relay to the patient that a notification is available, a speaker to add yet another method of alerting the patient of a notification, and finally the video output to connect to the display.

Hardware

Part List

Design

Below is a list of parts ordered at the time of the initial design before the implementation had begun. These parts covered the primary functionality of the system, as described in our design requirements.

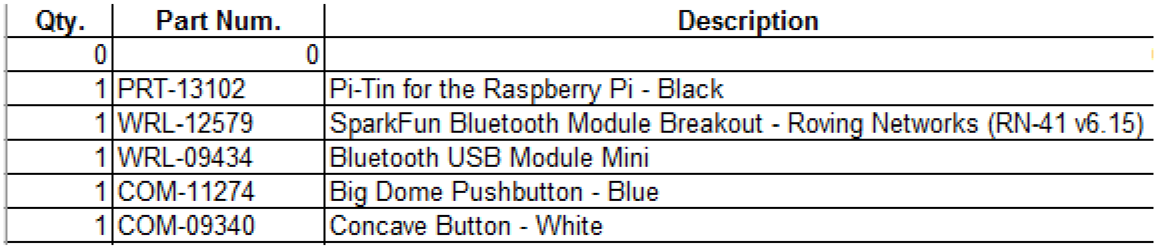


Qty.	Refdes	Part Num.	Description
1		DEV-11113	Arduino Pro Mini 3.3V
1		RN41-I/RM630	Bluetooth RN41
1		PRT-13851	Lithium polymer battery (3.7V 400 mAh)
1		B1034.FI45-00-01	Coin Type Vibration Motor
1		PRT-10217	Sparkfun LiPo Charger Basic - Micro-USB
2		PRT-13813	Lithium polymer battery (3.7V 1000 mAh)
1		1N4001	Diode, 50v, 1A, DO-41
1		2N2222A	NPN transistor
1		RASPBERRY PI	BCM2837 Raspberry Pi 3 Model B - MPU ARM® Cortex®-A
1		RASPBERRY PI	Raspberry Pi - Memory Card - microSD

Table 3 - Design Parts List

Implementation

Below is a list of parts that were ordered after the implementation of the design had begun. These parts were selected and added to the project in order to add further polish and reliability to the initial design.



Qty.	Part Num.	Description
0	0	
1	PRT-13102	Pi-Tin for the Raspberry Pi - Black
1	WRL-12579	SparkFun Bluetooth Module Breakout - Roving Networks (RN-41 v6.15)
1	WRL-09434	Bluetooth USB Module Mini
1	COM-11274	Big Dome Pushbutton - Blue
1	COM-09340	Concave Button - White

Table 4 - Implementation Parts List

System Block Diagram [DBC, ND]

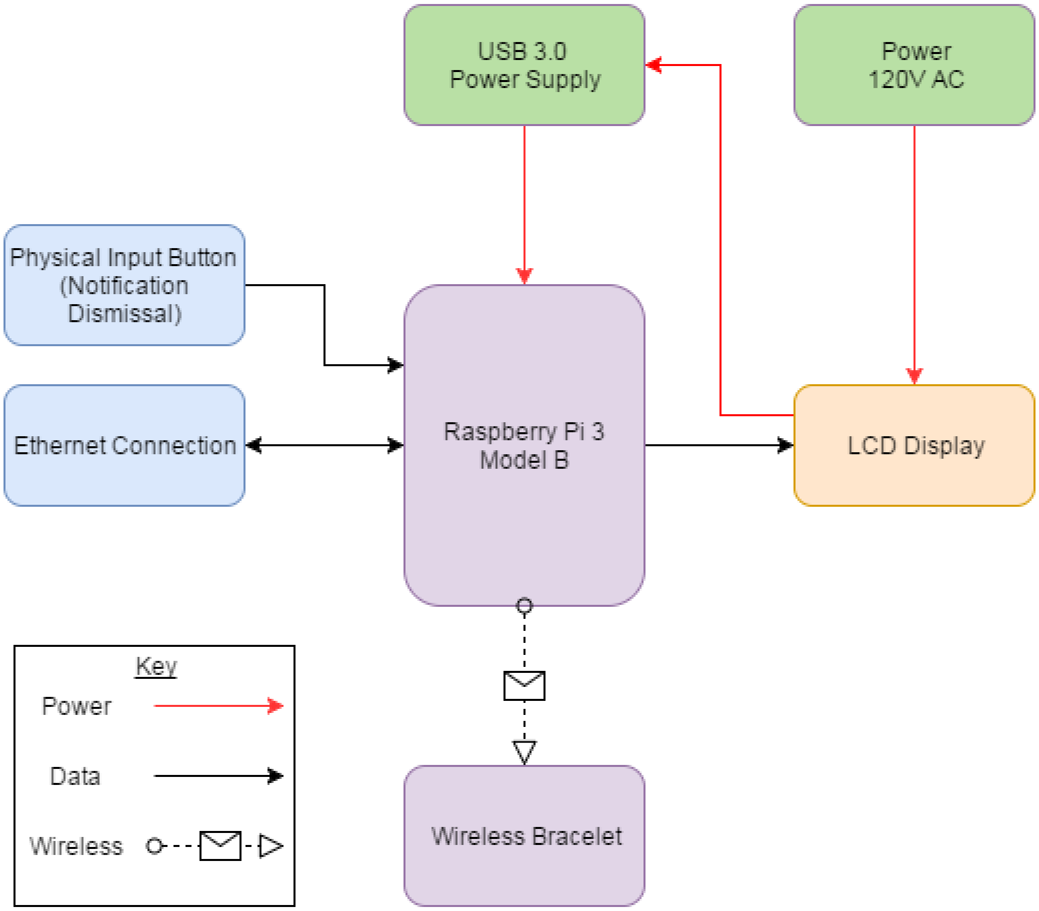


Figure 2 - Full System Layout

Bracelet Block Diagram [ND, DBC]

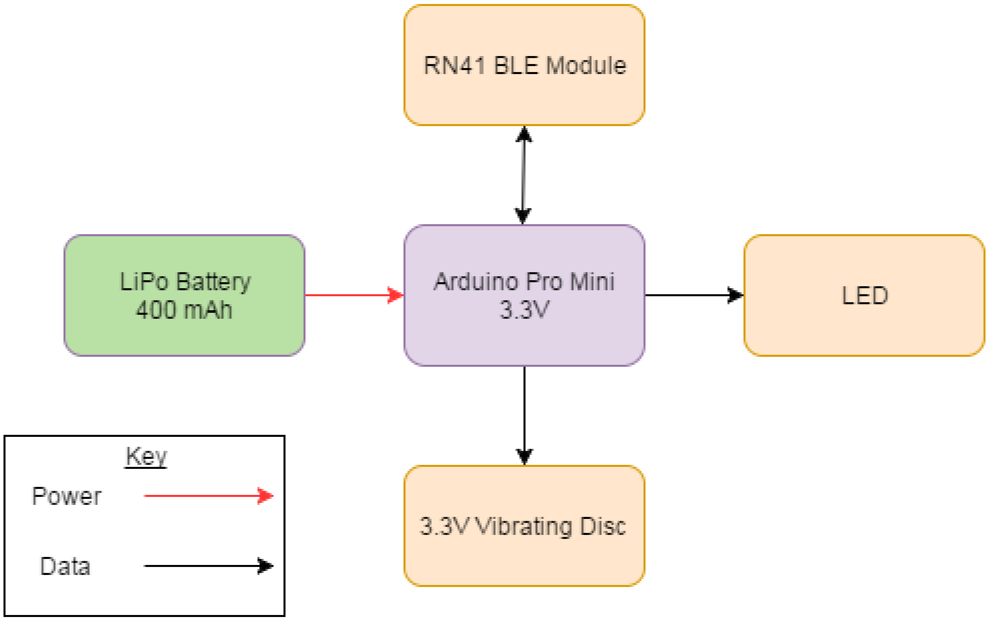


Figure 3 - Bracelet Layout

Display System Design [DBC]

The display system will consist of two primary components: an LCD display and a Raspberry Pi 3 Model B to drive said display. The LCD display will be an off-the-shelf solution in order to meet the requirements for affordability and upgradeability. It will connect to the Raspberry Pi over an HDMI cable connection as well as provide power to the Raspberry Pi through a built in USB 3.0 port. The USB 3.0 port will be sufficient to provide the necessary power requirements of 5V DC and up to 2A current draw, and it will allow us to power both the display and the Raspberry Pi through a single standard wall outlet, as designated in the design requirements.

There are several reasons that a Raspberry Pi 3 Model B was chosen as opposed to other similar systems such as a PIC microcontroller or an Arduino Uno. The first reason is the Raspberry Pi is an affordable and very widely supported system. Secondly, the Raspberry Pi has built in GPIO pins that will allow connectivity to devices such as buttons, LEDs, speakers, etc. that will accompany the primary display to make the use of the system simpler for the patient. The Raspberry Pi also has enough computing power to handle running a graphically based operating system, which is crucial for this design as it is built on creating a visually appealing and visually clear display. Another benefit of the Raspberry Pi is that it has several communication standards built directly into the main board such as Wi-Fi, Ethernet, and Bluetooth Low Energy. Wi-Fi and Ethernet will be of obvious importance as the device will need an internet connection for full functionality, and the Bluetooth Low Energy will be crucial for communication between the Raspberry Pi and the wireless bracelet, which will be described below.

The standard operating procedure of the Raspberry Pi can be seen in the flowchart below. It can be seen that the device will primarily be displaying persistent information such as weather information and calendar events. It will then be polling the API on the cloud server every 30 seconds to check for new alerts that need to be displayed. If there is a new alert available, such as a new message from the caretaker, the Raspberry Pi will blink the notification LED, sound a tone over the speaker, and display a message on the LCD display. The patient will then have to dismiss these notifications using the physical button connected to the GPIO of the Raspberry Pi. If the patient fails to dismiss the notification after fifteen minutes then the Raspberry Pi will alert the caretaker by pushing an alert to the cloud API, so that the caretaker is aware that the patient is not responding and there may be an emergency.

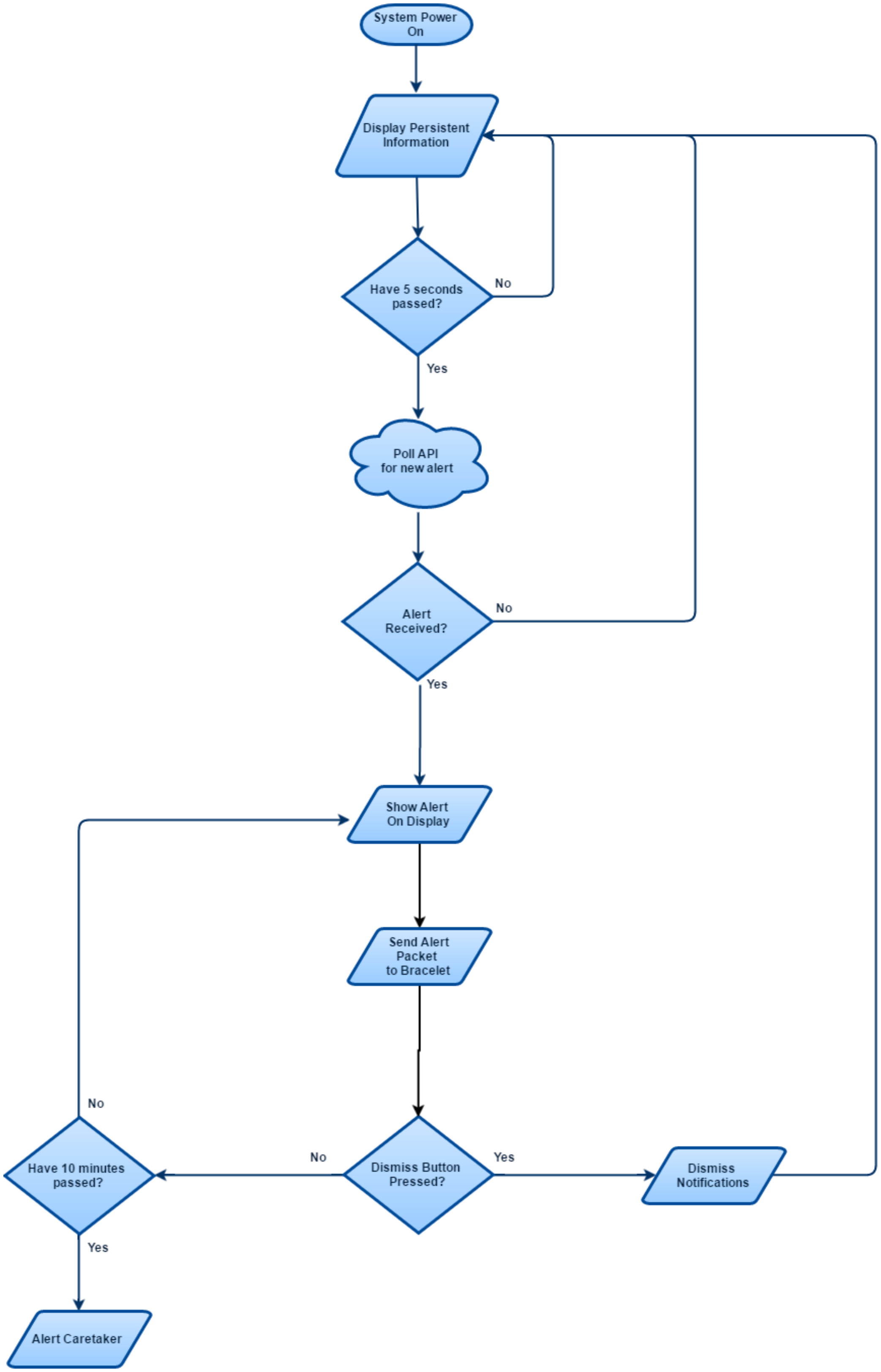
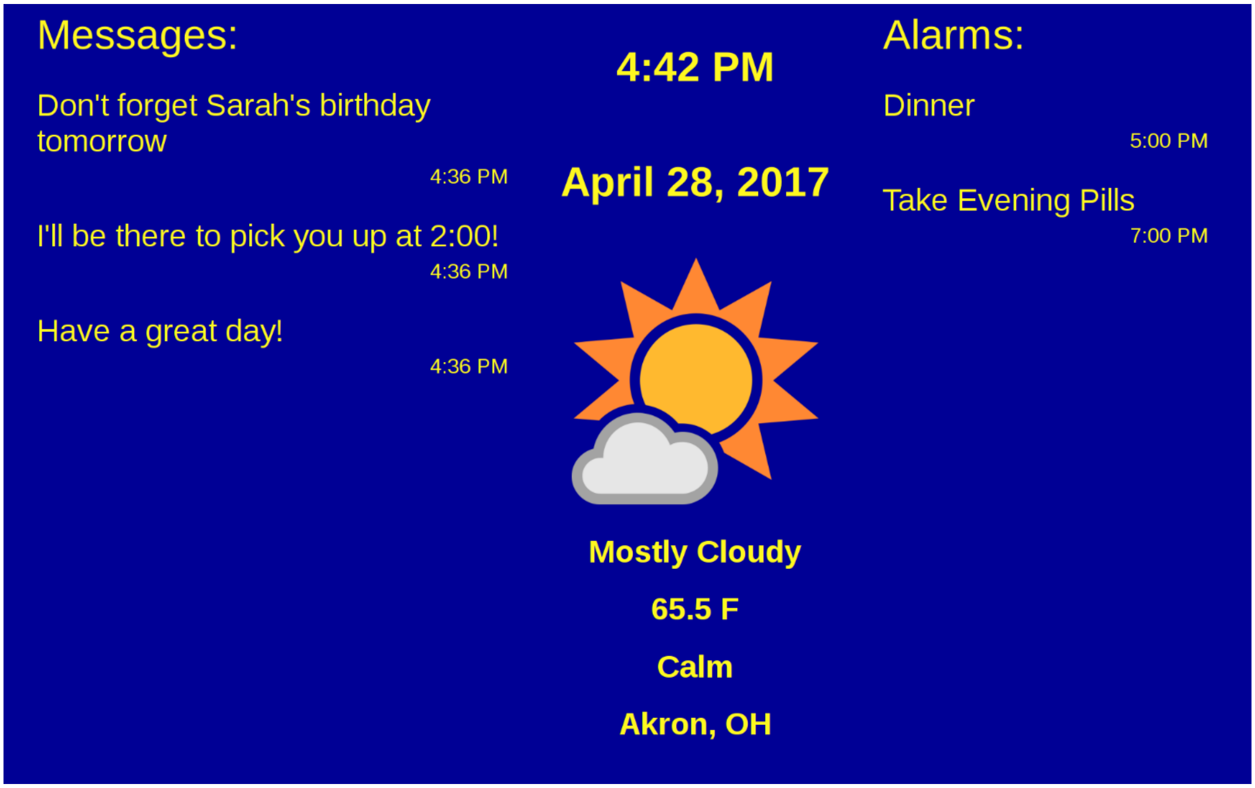


Figure 4 - Display System Flowchart



Messages:

Don't forget Sarah's birthday tomorrow
4:36 PM

I'll be there to pick you up at 2:00!
4:36 PM

Have a great day!
4:36 PM

4:42 PM

April 28, 2017

Mostly Cloudy

65.5 F

Calm

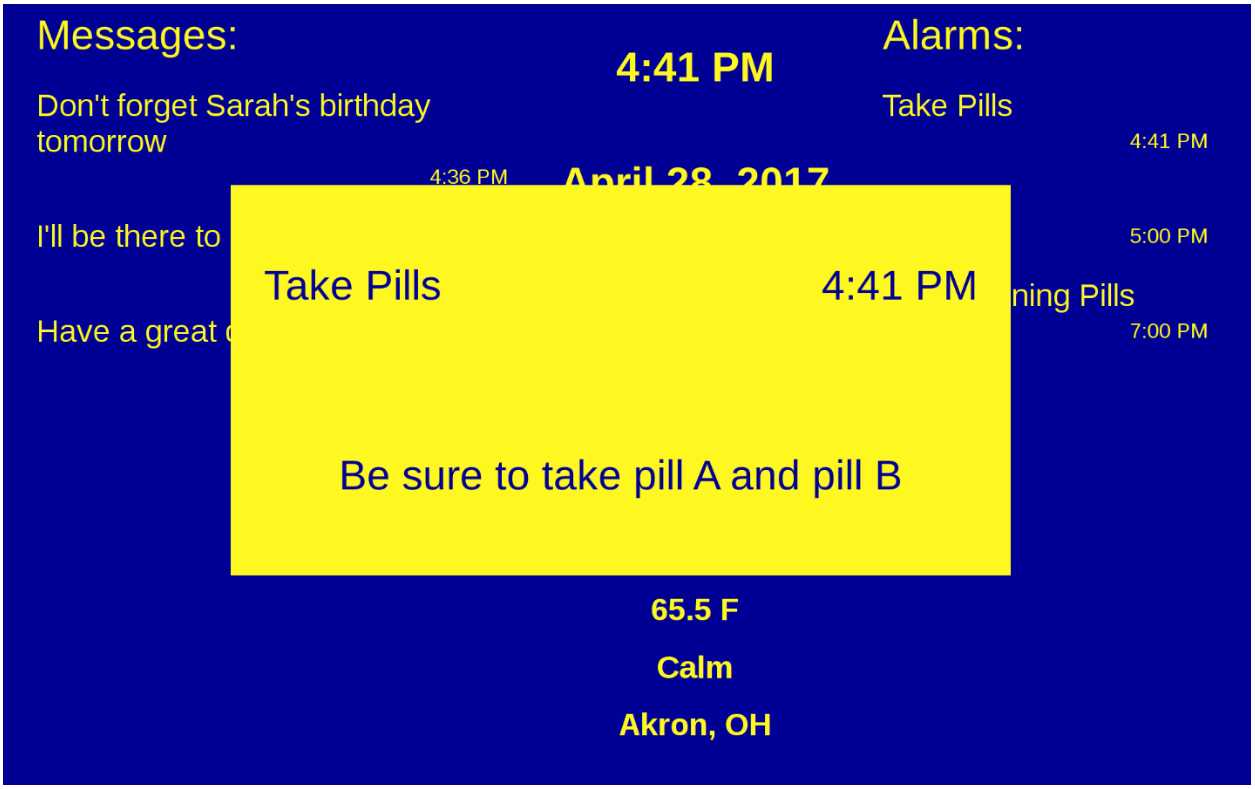
Akron, OH

Alarms:

Dinner
5:00 PM

Take Evening Pills
7:00 PM

Figure 5 - Primary Display



Messages:

Don't forget Sarah's birthday tomorrow
4:36 PM

I'll be there to

Have a great d

4:41 PM

April 28, 2017

Take Pills 4:41 PM

Be sure to take pill A and pill B

65.5 F

Calm

Akron, OH

Alarms:

Take Pills
4:41 PM

5:00 PM

ning Pills
7:00 PM

Figure 6 - Primary Display with Active Alarm

Bracelet Design [ND, DBC]

The final piece of primary hardware for this project is the wireless bracelet. The purpose of the bracelet is to supplement the display's notifications to the patient. It utilizes a vibrating disc motor and LED to alert the patient of alarms sounding at the display. It also features a push button that the patient can activate if they are in immediate need of help. To achieve this, the bracelet's embedded computer receives frequent updates from the display containing information about current alarms or notifications. The low power microcontroller lays dormant to save power and wakes up when the display dictates a notification. If the bracelet's push button is pressed for more than 3 seconds it will force a notification to the display. This notification causes the display to push another notification to the API, which the mobile application will receive. The mobile application will trigger an alert for the caretaker so they can check on the patient. The flowchart for the bracelet is pictured below.

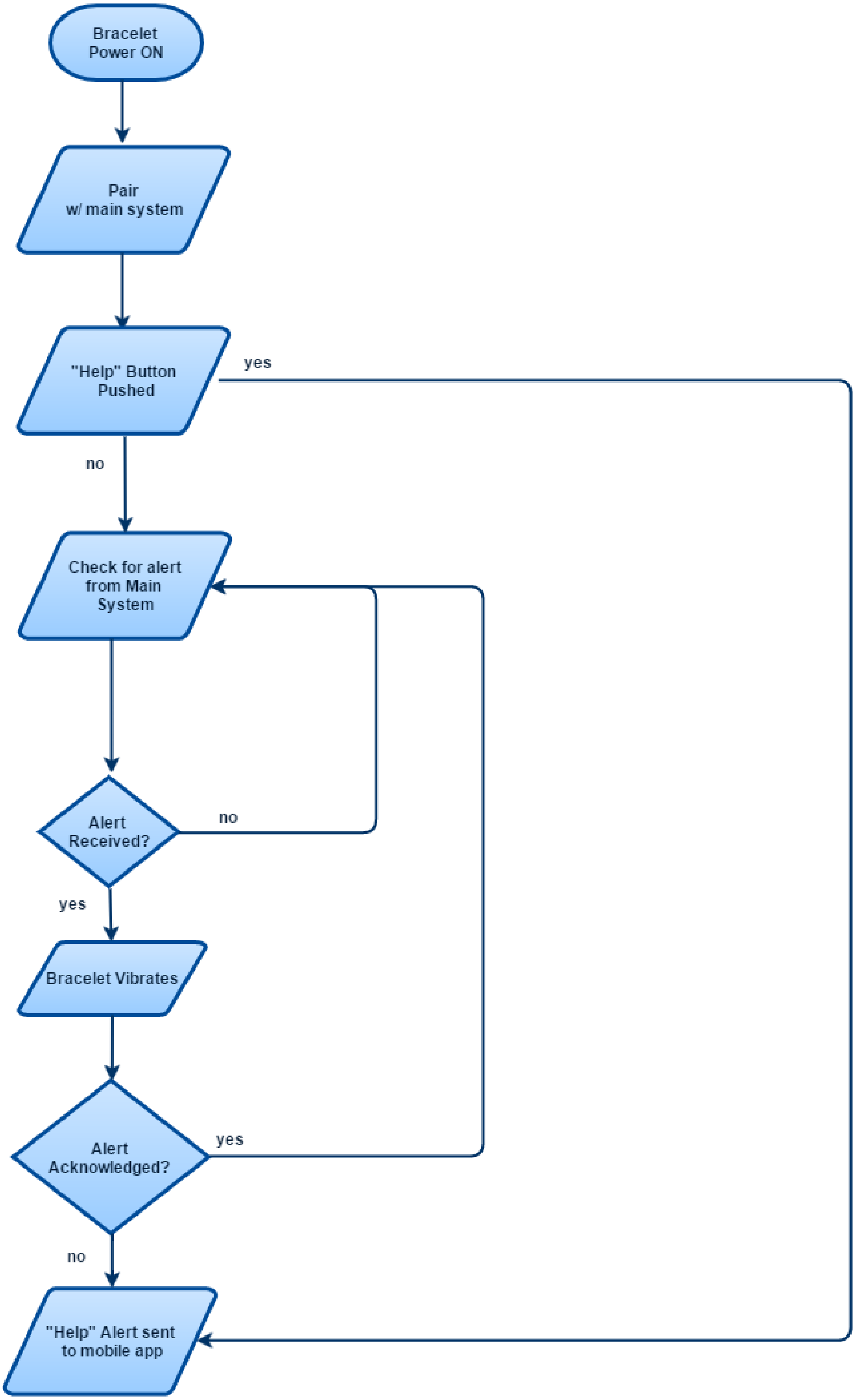


Figure 7 - Bracelet System Flowchart

The bracelet design and its components are heavily dependent upon power consumption and battery drain. The intended use of the bracelet requires it to be powered for 20 hour intervals between charges. Power consumption due to communication with the main embedded computer will determine the most efficient design. A Bluetooth Low Energy (BLE) connection is an excellent protocol to transmit the small amount of data over the 150 feet range required. Estimated power consumption for the Bluetooth LE is 0.01 to 0.5 watts. The power consumption calculations are done for the Arduino Pro Mini microcontroller Atmega328P. It can operate on a range of voltages and has optional low power clock mode (clock speed reduced by half and lowest working voltage). Below in Table 3 are the ratings the microcontroller will run at for the wireless bracelet.

Microcontroller	Voltage	Current
ATMEGA328P	3.3V	3.73mA (@3.3V and 8MHz clock)

Table 5 - Bracelet Power Calculations

When powering the remote circuit selecting a battery with the appropriate capacity is the first step. A 3.7V lithium ion polymer 2200mAh battery was chosen. Along with considering the weight and size of the battery, after calculations it has provided the desired design specifications. To determine the battery capacity needed the normal current the circuit will draw was calculated. To find the power consumption/current draw values Equation (1) was used. In addition to current draw for the ATmega328P calculated the current draw of each component added in the circuit in Table 4.

Equation 1 - Power = Voltage x Current

$$3.3V * 3.73mA = 12.3mW$$

Main Components	Voltage	Current Draw	Size/Weight

Microcontroller-ATMEGA328P	3.3V	3.73mA @3.3V and 8 MHz clock	33mm x 18mm
Bluetooth Low Energy	3.7	10.1uA	5mm x 5mm
LED	3.7V	10mA	3-5mm
Vibrating Mini Disc Motor	2-5V	60mA	10mm diameter, 2.7mm thick/.9g

Table 6 - Bracelet Power Calculations Continued

After calculating each current draw simplifying summing them all up would get a general total power consumption. With power consumption value known the battery can be selected based off battery life calculations.

Total Power Consumption = 75mA

Battery life is dependent upon the hardware, distance, and duty cycle. The Bluetooth LE can last up to 34 months on a single battery when only sending periodic impulse signals. The vibrating disc motor and LED will also be used periodically reducing the battery drainage. Battery calculations can be seen below using Equation 2. Using a 2200 mAh battery and for the 3.3v regulator:

Equation 2 - Battery Capacity (Ah)/Current Draw (A) = Battery Life (Hours)

$$\frac{2200mAh}{75mW} \approx \ 30hours$$

We would ideally get about 30 hours of battery life from the LiPo battery. Due to efficiency consideration of 85%, the battery life is about 25 hours. The bracelet power battery life exceeds the 24 hour design requirement and allows for a functional wireless bracelet for the patient that needs charged daily.

The wireless bracelet will be constructed of a flexible plastic band and a small rectangular compartment to hold all the components. In order to reduce the possibility of water damage the bracelet will be coated in a silicone mixture allowing it to be water resistant. The length of bracelet band will vary depending upon user. The wireless bracelet is intended to be comfortable, conveniently small and easy to use.

Bracelet Schematic [ND]

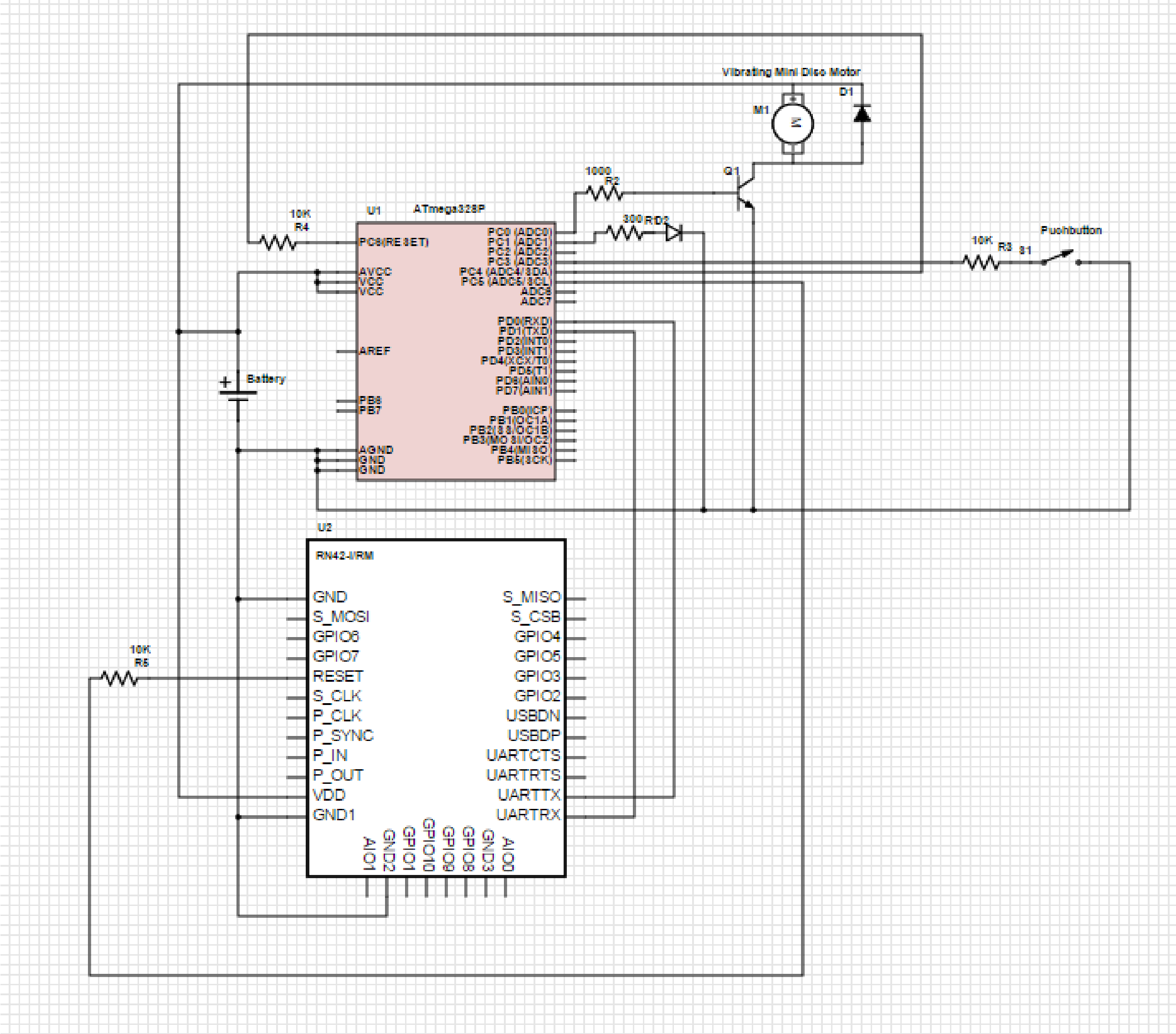


Figure 8 - Bracelet Schematic

The bracelet schematic in Figure 6 shows the detailed connections of each component in the circuit. The circuit layout consists of the following main components: lithium ion polymer battery, Arduino Pro Mini microcontroller, Bluetooth module, vibrating disc, LED and momentary pushbutton switch. The 3.7V LiPo battery powers the microcontroller, vibrating disc and the Bluetooth module, while the microcontroller controls the entire circuit. Due to the limited output current for each pin on the microcontroller a 2N2222A transistor is used to amplify the current and switch on/off to drive the vibrating mini motor disc. To ensure too much current does not flow from the output of the transistor a resistor is in series with the base. This attenuates the current to a reasonable amount so too much current isn't flowing through the motor.   A reversed biased diode is also put in place in parallel with the motor to act as a surge protector against negative voltage spikes. This will protect the other components in the circuit. Each GPIO pin has a resistor in place before all components to control current flow. Along with the vibrating motor to alert the patient wearing the bracelet in a different form a LED is connected to a pin. The LED will blink when an alert comes through. The final feature on the bracelet is the "Help" button. In the schematic a pushbutton is connected. When the pushbutton is un-pressed/open there is no connection between it so the pin is grounded and read as high. When the button is pressed/closed a connection is made and the pin will read low to signal a help alert to be sent to the main system. Both reset pins on the microcontroller and Bluetooth module will be controlled with a high from the respective pins.

Bracelet Communication [DBC, SN]

As discussed above the wireless bracelet and the Raspberry Pi that powers the primary display will communicate over Bluetooth Low Energy. This addresses the communication in terms of hardware, however there also exists a simple command system that allows the display to control various functions of the bracelet.

The structure of the command system that will facilitate control of the bracelet is as follows:

Action	Operation Triggered	Response to Display
Send bracelet a ‘0’	Cancel current alarm protocol	“Alarm canceled early.”
Send bracelet a ‘1’	Begin passive alarm protocol	“Passive alarm ran its course”
Send bracelet a ‘2’	Begin active (priority) alarm protocol	“Active alarm left unhandled”
Hold bracelet push-button for 5 seconds	Send panic signal to display	“Panic signal activated”

Table 7 - Bracelet Command Protocol

The bracelet operation is quite simple. It waits for the display to send a command and executes a protocol for some predefined amount of time depending on the type of operation being requested. Otherwise, the bracelet can trigger a push notification to any authenticated mobile device with the Forget-Me-Not application installed by sending the display a panic signal. The display uses this panic signal to trigger the push notification via “Firebase,” a google app engine push notification service When sent a character one (‘1’), the bracelet begins vibrating according to the passive alarm protocol for five minutes. Five minutes is the timeout for passive alarms on the display. That is, after five minutes the alarm will dismiss itself. Upon completing the passive alarm protocol the bracelet sends the display an acknowledgement. The passive alarm protocol generates a lighter less urgent vibration pattern. When sent a character two (‘2’) the bracelet begins vibrating according to the active (priority) alarm protocol for five minutes. It is important to note that for the active alarms the bracelet stops vibrating after its timeout and sends a timeout notification to the display. The display does not automatically dismiss active alarms due to the fact that they are of particular importance. The active alarm protocol generates a more urgent vibration pattern. At any point during an alarm protocol the character zero (‘0’) can be sent to cancel the current running protocol. If no protocol is active, then the zero is dismissed. Upon successfully canceling the bracelet sends an acknowledgment to the display, otherwise no response is sent.

The active and passive alarm protocols are defined as follows:

1.) **Active Alarm Protocol** - repeated until five total minutes have passed.

- Vibration period:
    - Vibrate continuously for three seconds then stop for two seconds (repeat three times for a total of three long buzzes)
- Sleep Period:
    - Bracelet stops vibration and waits for 30 seconds

2.) **Passive Alarm Protocol** - repeated until five total minutes have passed

- Vibration period:
    - Vibrate for one second the off for one second (repeat five time for a total of five short buzzes)
- Sleep Period:
    - Bracelet stops vibration and waits for a minute

Software

Introduction [SN]

Forget-Me-Not's software design is broken into three main pieces: a display, the web API and a mobile application. Each piece follows the accepted best practices in its design to allow for adequate testing and to ensure a robust final product. This section details the design patterns that are used as well as the final implementations of those patterns in the form of software diagrams for each of the three sections.

User Interface Design Pattern Overview: MVVM [SN, JK]

MVVM is a design pattern first showcased by Microsoft which allows programmers to eliminate tightly coupled classes from their user interfaces or user facing application. When developing a UI - or any other complex class structure - it is imperative to create a distinction between the various sub-parts of the system. Loosely coupled designs carry a multitude of benefits such as testability, extensibility and reliability. MVVM allows for this loose coupling by defining three main roles in which a project's class structure must fill. Forget-Me-Not's UI design will follow this pattern for the display and mobile application as it is considered a "best practice" in many programming communities for applications dealing largely with user interaction. Diagram 3 below shows how these roles are organized in relation to each other.

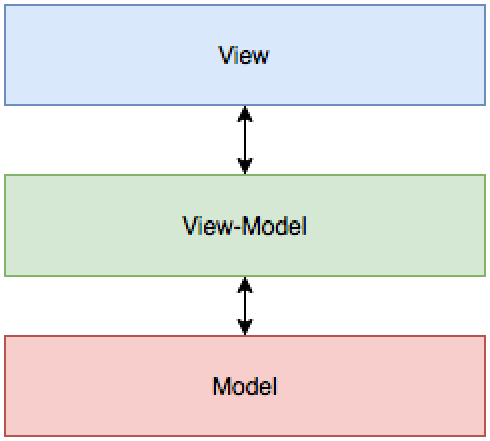


Figure 9 - MVVM

The view encapsulates everything that a user will interact with directly. It has no knowledge or ties to the data it needs to represent; it only knows how to represent data it is given. This makes the view unit testable and easily to debug, as the programmer only needs to worry about data representation rather than storage and manipulation.

The view-model is like a controller between the view and the model it represents. Because loose coupling is desired, programmers don't want the model to be aware of the view's functionality. Rather, the model gives raw data to the view-model (the controller) when requested and then the view-model informs the view that it must update or refresh what it is showing the user. In Forget-Me-Not's design this section will also be in charge of initiating HTTP requests from the web server discussed later and storing the response data in the model object.

Finally, the model is a simple container for the data the programmer is trying to represent. It holds information and should do very little, if not, nothing else. Note how this pattern is used in the following diagrams.

Other Useful Design Patterns for this Section

Favoring Composition to Inheritance

Some figures may abstract away the complexity of the underlying classes. Each block in the software diagram may represent several classes. This pattern keeps object oriented programs from abusing their inheritance capabilities when constructing potentially dependent classes. This method couples classes loosely by coding to abstract classes rather than concrete implementations.

The Communicator Design Pattern

The communicator design pattern allows various functions and objects to send messages to each other. This is advantageous when two classes that have little in common need to share some bit of information. Rather than coupling them concretely and spreading bad dependencies throughout the code base, communication is performed via lightweight signals.

Display Software Design [SN, JK]

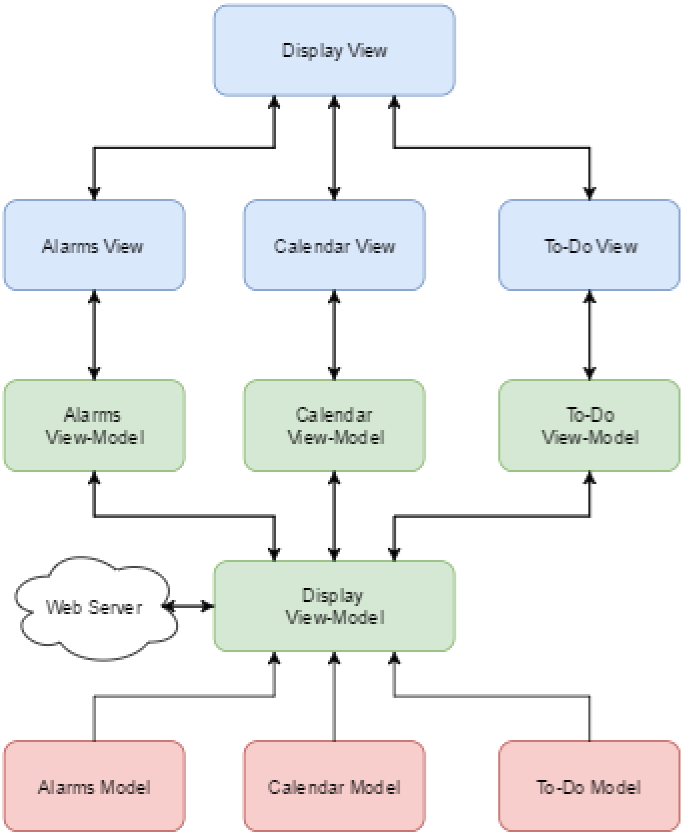


Figure 10 - Display Software Design

The display's software design adheres to MVVM's decoupled format allowing for a very modular arrangement. Only three modules were added for the final iteration of the display, however, the modular nature enables us to add or remove modules while having to alter only a small part of the code base. In fact, the only existing files that would need to be altered are those the module directly touches (e.g. the *Display View* and *the Display View-Model*). The *Alarm, Messages* and *Weather* modules are an example of Forget-Me-Not's minimum viable product, but with little effort the system can be expanded to include much more functionality. Designing a product for extensibility is necessary to reduce overhead and keep a lean codebase for future iterations of the product.

The view section of Forget-Me-Not's software design consists of the main display view and the modules that it manages. The main *Display View* is responsible for laying out the various modules cleanly on the screen. Ideally, each module would be granted a section of the main view on which to display its information. Should there be too many modules to fit on the screen the main display view will be responsible for re-formatting or paginating the modules' view in a clean and attractive manner. The latter case is currently unnecessary as the minimal feature set created for the final iteration of the display fits cleanly without pages.

The view-model section of the design is similar to the view section in that it has a main *Display View-Model* class and various other module classes that it manages. Again, in the name of extensibility, these sub-modules are in charge of knowing all the behavior for their own view and model in the system. This ensures that if this module is removed, no other dependencies will be broken. This set of classes will contain most of the control logic for the display. One sub-module will contain the behavior for communicating with the web server and other will handle passing information between model and view.

The model section is the simplest section of the display's software design. Its purpose is to hold the raw data that is to be represented. However, the model still must have structure. Data contained in the model is stored in variables such that its structure is logical to the programmer. For example, a file model may contain a name field and a data field among many others – it is not ideal to have to parse some contiguous array of information to get a name or path versus the data segment of interest.

The primary code that drives the application on the display was written in C++ using the QT Creator development environment, and can be seen below.

Display UML Diagram: [JK, SN]

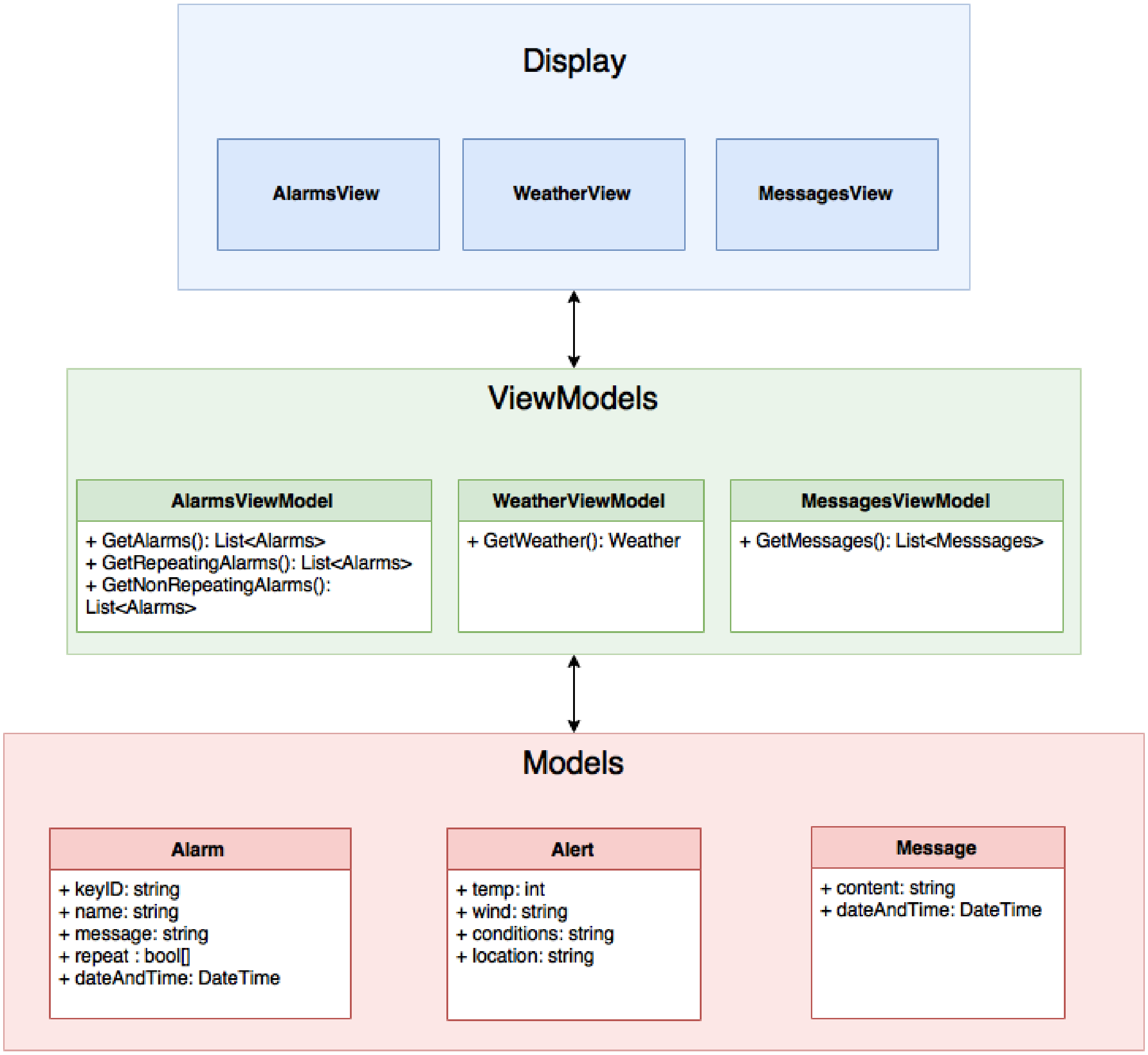


Figure 11 - Display UML Diagram

Mobile Application [JK]

The mobile application will give the caretaker the ability to interact with the display without being on site. This helps the caretaker feel better about their patient's day, as well as assist in the patient feeling more grounded. The remote interaction with the display includes setting alarms for the patient, sending messages to the display, and getting important updates.

There are 4 different pages in the application: the dashboard, alarm editor, communications, and settings pages. The dashboard page (shown in Figure 13) is a landing page for any urgent information that the caretaker should know. This includes information such as the patient failing to respond to a configured alarm, or the patient pressing the panic button on the bracelet for 5 seconds. In both of these cases there will be a generated push notification sent to the mobile device that will alert the caretaker via a dynamic label at the top of the dashboard page. This page also doubles as a visual list of any configured alarms. There exists two lists, one for alarms with repeating preferences and one for alarms without repeating preferences. If the caretaker taps on one of the alarms they will be brought to the alarm editor page that will be pre-populated with that specific alarms information. Finally, the dashboard page will have a button where the caretaker can add a brand new alarm.

On the alarm editor page (shown in Figure 14) the patient can create new alarms/ edit old alarms. This includes changing the title, message, repeating preferences, priority, and date/time associated with that alarm. When the save button is tapped an http request is sent to the API to save that alarm in the API's database. If the alarm is an existing alarm, there is be an option to delete that alarm which will also send an http request to delete that specified alarm from the API's database.

The communications page (shown in Figure 15) provides the caretaker a way to send messages to the display for the patient to read. This is presenting in the same fashion as many messaging applications. Having this ability as the caretaker makes it much easier to reach the patient for small reminders such as someone's birthday, an event coming up in their lives, etc.

The settings page (shown in Figure 16) is screen used to configure of the general setting for the mobile application of the display. As of the current design the only settings that exist there are the patients name and their phone number. This settings screen can be expanded upon to add in numerous other settings such as the color scheme of the display, the default weather unit (Fahrenheit or Celsius), etc.

The design of the application follows the aforementioned MVVM design pattern, so there is a Model, View-Model, and View for each of the pages on the mobile application. The only exception is that a Model for the communications page is not needed. This is because instead of saving the messages to the mobile device's onboard memory, the messages will be stored on the cloud server and will be accessed as needed through HTML requests.

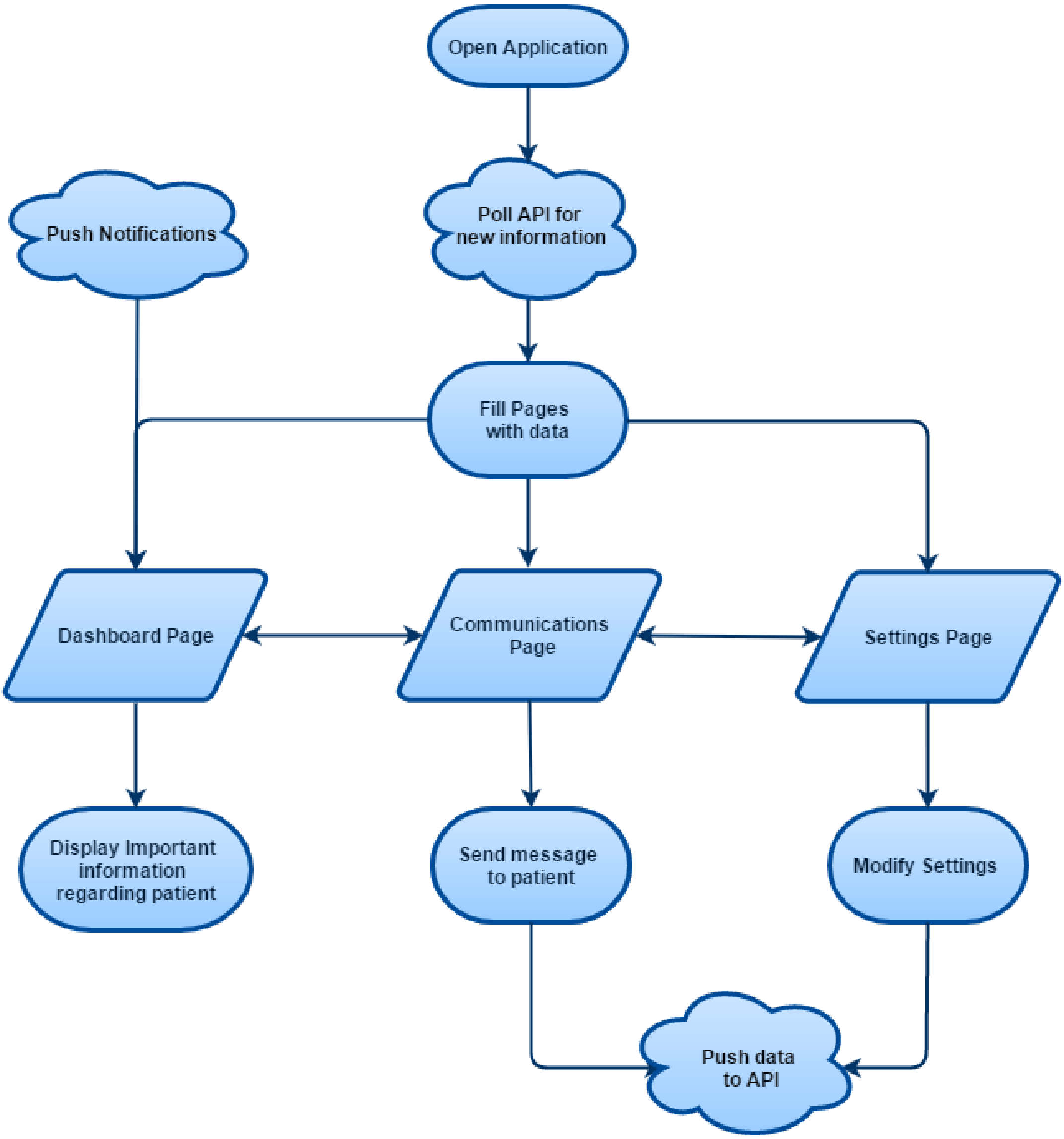


Figure 12 - Mobile Application Flow Chart

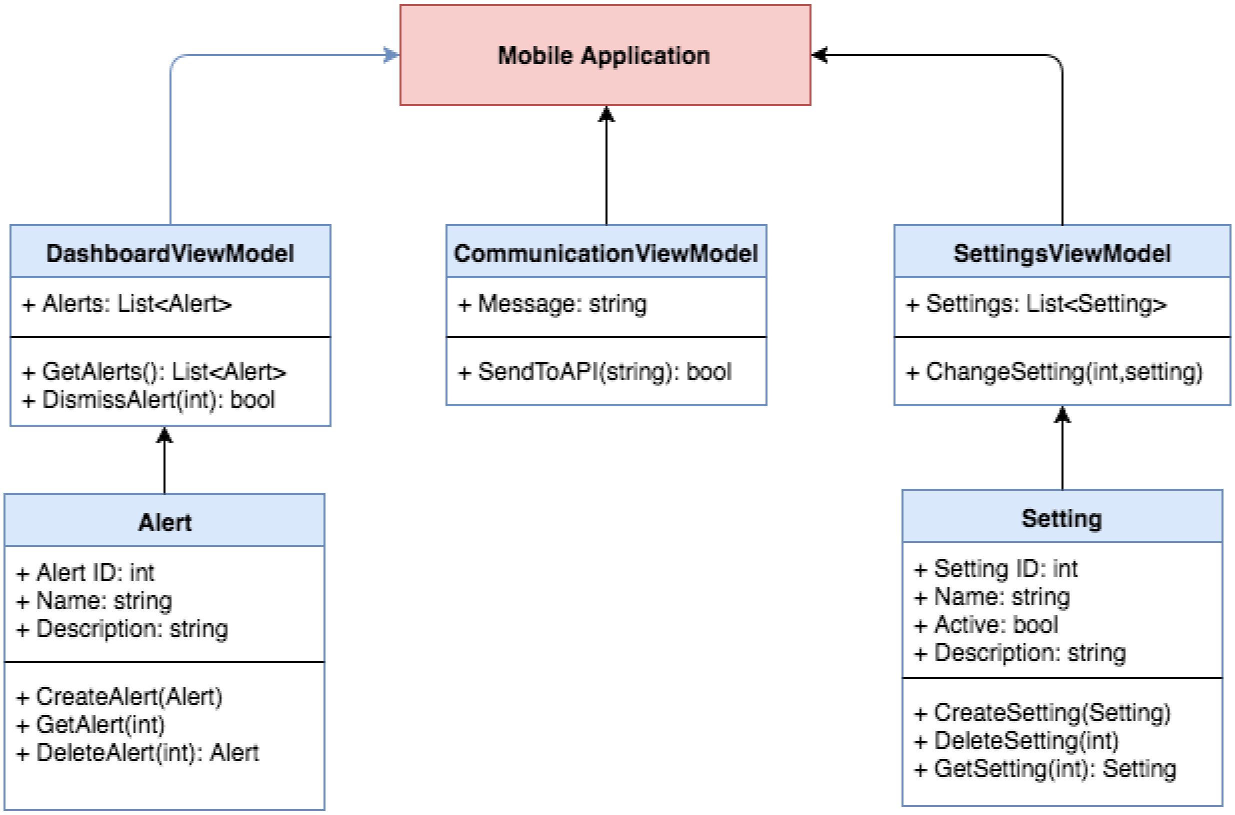
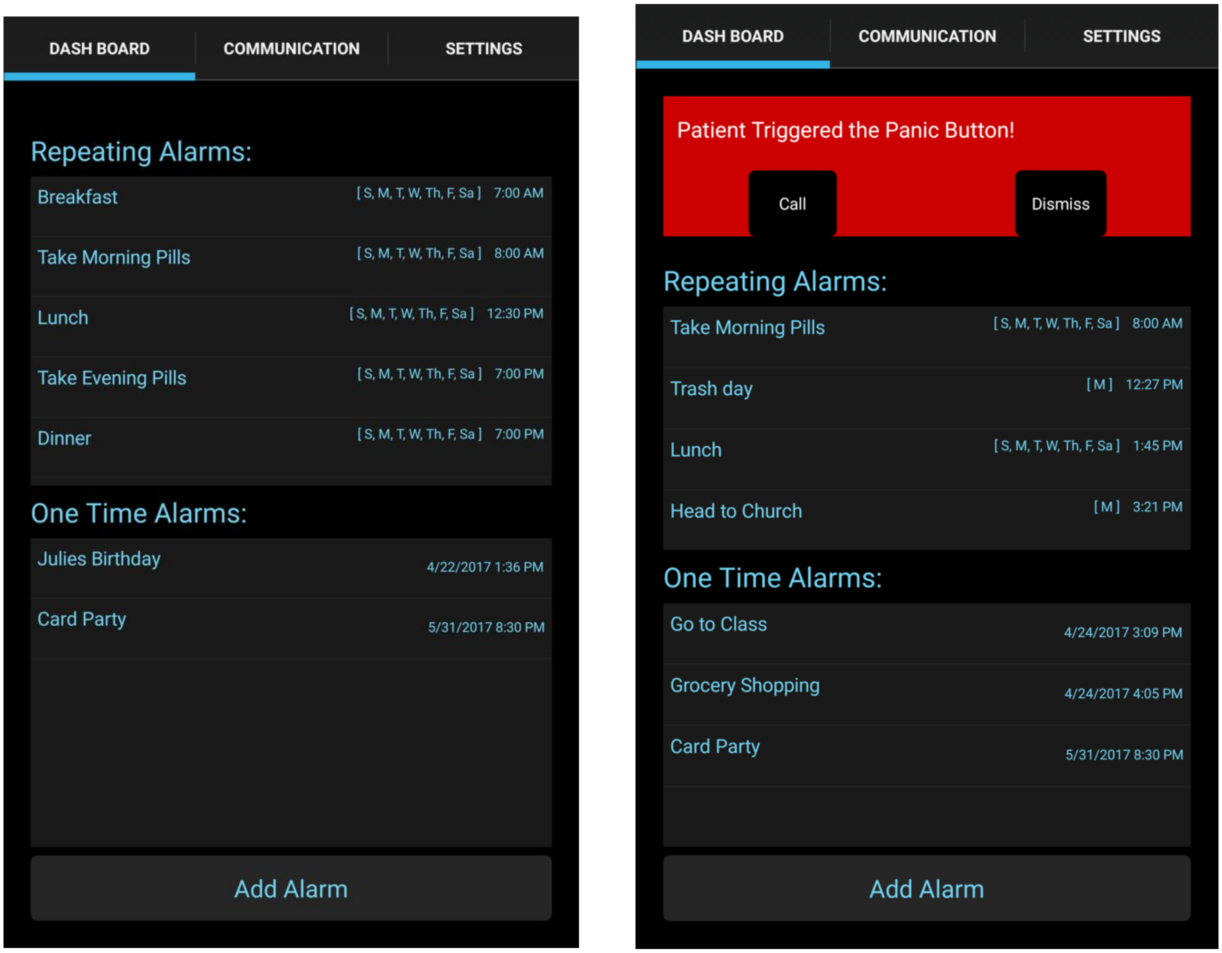


Figure 13 - Mobile Application UML Diagram



Repeating Alarms:

Breakfast [S, M, T, W, Th, F, Sa] 7:00 AM

Take Morning Pills [S, M, T, W, Th, F, Sa] 8:00 AM

Lunch [S, M, T, W, Th, F, Sa] 12:30 PM

Take Evening Pills [S, M, T, W, Th, F, Sa] 7:00 PM

Dinner [S, M, T, W, Th, F, Sa] 7:00 PM

One Time Alarms:

Julies Birthday 4/22/2017 1:36 PM

Card Party 5/31/2017 8:30 PM

Patient Triggered the Panic Button!

Repeating Alarms:

Take Morning Pills [S, M, T, W, Th, F, Sa] 8:00 AM

Trash day [M] 12:27 PM

Lunch [S, M, T, W, Th, F, Sa] 1:45 PM

Head to Church [M] 3:21 PM

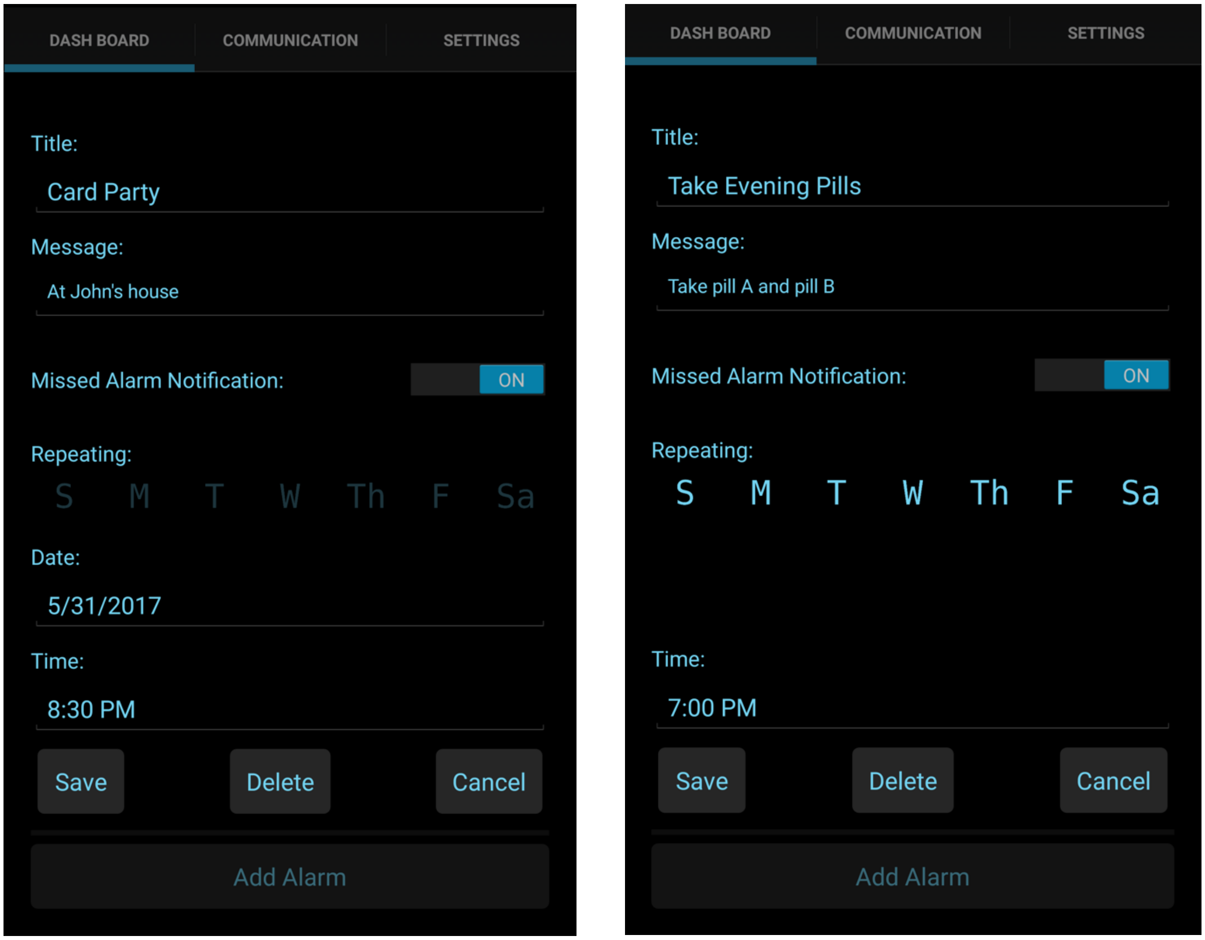
One Time Alarms:

Go to Class 4/24/2017 3:09 PM

Grocery Shopping 4/24/2017 4:05 PM

Card Party 5/31/2017 8:30 PM

Figure 14 - Dashboard Page



Title: Card Party

Message: At John's house

Missed Alarm Notification: ON

Repeating: S M T W Th F Sa

Date: 5/31/2017

Time: 8:30 PM

Title: Take Evening Pills

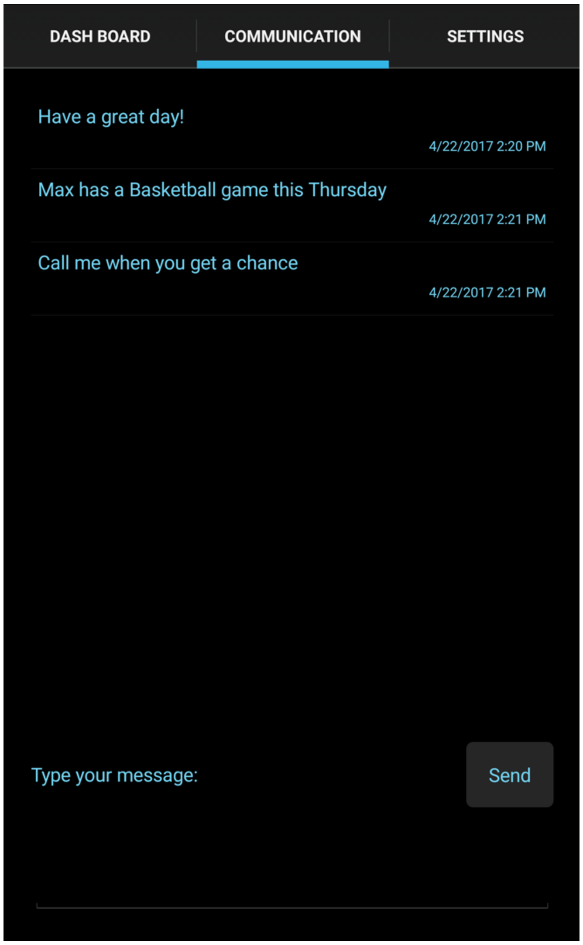
Message: Take pill A and pill B

Missed Alarm Notification: ON

Repeating: S M T W Th F Sa

Time: 7:00 PM

Figure 15 - Alarm Editor Page

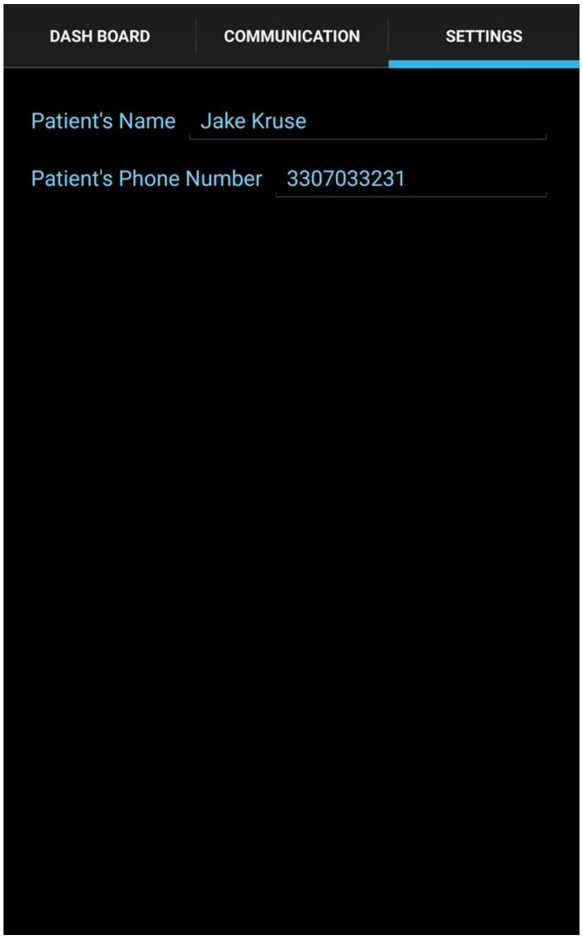


Have a great day!
4/22/2017 2:20 PM

Max has a Basketball game this Thursday
4/22/2017 2:21 PM

Call me when you get a chance
4/22/2017 2:21 PM

Figure 16 - Communications Page



Patient's Name Jake Kruse

Patient's Phone Number 3307033231

Figure 17 - Settings Page

RESTful Web Server Design Pattern Overview [SN]

REST is a web API design pattern that details how HTTP requests are handled. It utilizes the commands GET, PUT, POST and DELETE to transfer information to and from a server. REST is not a clearly defined pattern as there are numerous ways to adhere to its stipulations. The only concretely defined components are the aforementioned commands as well as the idea of “statelessness”. The atomicity of requests and responses form a very simply structured communication protocol. The commands are detailed in table 4 below.

- **REST** stands for **Representational State Transfer**. It is considered a “best practice” to construct a web API that follows a RESTful pattern.
- Requires a **stateless** client server. This means that each request to the web server contains all the necessary information for that request.

- Almost always executed using HTTP protocol.
- Relies on four main commands to manipulate “endpoints” and their data

	Command	**Description**
1	GET	Get data or object at specified endpoint.
2	POST	Create data or object at specified endpoint.
3	PUT	Update existing data or object at specified endpoint.
4	DELETE	Delete data or object at specified endpoint.

Table 8 - API Endpoints

Decorator Design Pattern

The decorator design pattern describes how using function decorator capabilities of a programming language can simplify code. In the case of Forget-Me-Not, decorators will be used to define HTTP endpoints. See *Example Endpoints* at the end of this section for examples.

RESTful API Design [SN]

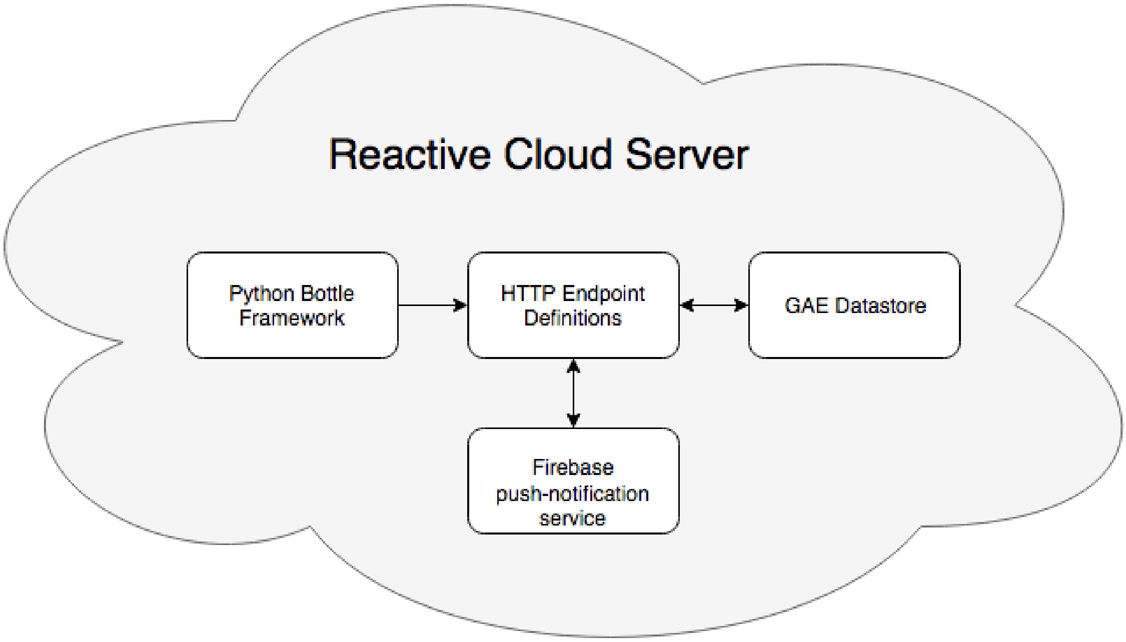


Figure 18 - API Cloud Server

Forget-Me-Not's API is responsible for all the persistent storage of the system details. It contains all the data that is to be represented on the phone and display. Each of the devices still have a local copy of the information, but regularly pull information from the API so that all local information is up to date. If a change is made by a user of the mobile device, that device will send a request to the API to update information. Currently, best practices recommend using a Python *Bottle* server for lightweight web API architecture. It allows for a clean logical definition of HTTP endpoints. *Bottle* is the library responsible for all of the HTTP functionality of the API. It uses function decorators to define an endpoint. If that endpoint is reached, it executes the function it is wrapping to perform some task. Google's built in "datastore" feature is used to store all information. The datastore is based on Google's GQL framework. GQL is a reference to SQL, as google based its framework largely on the functionality of SQL databases. All items stored in the database are referred to as "entities". A *Notification* class supplements this RESTful API by allowing push notifications to be sent to any authenticated mobile device. Because pushing information to a client is not a RESTful practice, the *Notification* class takes care of all pushing tasks. It watches the api and if the *warnings* endpoint is updated the *Notification* class uses Google's cloud messaging service, *Firebase*, to push the desired notification. Each of these components is described in detail in the following subsections.

API Endpoints and Examples [SN]

The following table lists each endpoint and summarizes its function. Complete API documentation containing all necessary details to use the API can be found in the appendix. **NOTE:** each endpoint URL is preceded by the host URL of the Forget-Me-Not API web application: "https://forgetmenotapi-145619.appspot.com/"

Endpoint URL	Request Operation	Summary of Endpoint's responsibilities
/api/v1/alarms	GET	Returns a JSON object containing all alarm entities in the datastore.
/api/v1/alarms	POST	Creates a new alarm entity and places it in the datastore.
/api/v1/alarms/{alarmId}	PUT	Pulls a specific alarm from the datastore, updates its value and returns it to the datastore.
/api/v1/alarms/{alarmId}	DELETE	Deletes a specific alarm from the datastore.
/api/v1/messages	GET	Returns a JSON object containing all message objects
/api/v1/messages	POST	Creates a new message entity and places it in the datastore.
/api/v1/messages/{messageId}	DELETE	Deletes a specific message from the datastore.
/api/v1/tokens	POST	Adds an authentication token to the datastore if it does not already exist.
/api/v1/warnings	POST	Adds a warning message for the *Notification* class to push to any authenticated device.

Example Endpoints

- https://forgetmenotapi-145619.appspot.com//api/v1/alarms

- The alarms endpoint
- GET to this endpoint returns JSON data structure containing all saved alarms.
- No DELETE, POST, or PUT defined for this endpoint –ideally, operations will not be performed on all alarms at once, other than returning them to the devices.

- https://forgetmenotapi-145619.appspot.com//api/v1/alarms/{alarmID}
  - The endpoint for a specific alarm (differentiated from all others by its unique ID).
  - GET to this endpoint returns a specific alarm object in JSON data structure form.
  - Similarly the other operations PUT, POST and DELETE are defined for this endpoint.

Code for an Endpoint

```
@app.get("/api/v1/alarms")
@app.get("/api/v1/alarms/")
def get_alarms():
    allAlarms = Alarm.get_all()
    if allAlarms == None:
        response.status = 204
        return None
    else:
        #Wraps array in JSON object for
convenience with QT functions
        jsonReturn = {"alarms":allAlarms}
        response.status = 200
        return json.dumps(jsonReturn)
```

Note the “@” - this is the portion known as the decorator. The decorator wraps the function to be executed when the endpoint is reached.

API Backend [SN]

A well-defined and constructed API can crumble if it lacks the proper platform on which to run. The Forget-Me-Not design team has produced a functional product, which means all subsystems are designed with future expansion in mind. Creating a product with no room to breathe or grow is synonymous with setting it up for failure. Currently, the API is so tiny that nearly any functioning computer could serve its functions to users adequately. However, if demand increases or fluctuates a more powerful system would be necessary to react without crashing or slowing down. In a system that is designed to monitor and care for loved ones, room for error or malfunction is sparse.

API Documentation

Any API must be well documented, even if it will only be used internally. Public APIs must be designed so that any developer may learn to use the interface. Likewise, internal APIs such as the one under development for Forget-Me-Not, must still be easily accessible to the engineers using it. YAML is a recursively defined acronym for Yaml Ain't Markup Language. It is often used for storing object trees, but also lends itself very well to documentation. It is often considered easier to read than JSON or XML. The complete API documentation can be found in the appendix.

Google App Engine: A Reactive Cloud Server

Google has a massive server infrastructure that they allow people to use for any sort of project they can think of. This system is known as Google App Engine (GAE). These servers react to the current load and allocate more resources if necessary, never consuming more than they need. Moreover, if a large spike in users attempting to access the server occurs there will be no crash or slowdown. Zero hardware maintenance and an easy to use interface make GAE the perfect choice for a small project with lots of budget constraints.

- Cost
    - GAE is free until a certain threshold is reached. This threshold is so high that the design team will have to pay absolutely nothing to use this server infrastructure.
    - Future iterations of Forget-Me-Not will be able to expand within a nearly infinite server space, paying only for the amount used.
    - Zero hardware maintenance means easy and cheap upkeep for a small team.

- Reliability
    - Forget-Me-Not has very little room for failure, so the robustness provided by a professional server system is a must.
    - Any crashes will quickly be corrected. GAE will sense that the app has crashed, spin up another virtual machine in its place and continue to serve the API. This minimizes downtime and allows developers to serve the API while performing

maintenance.

Datastore

Forget-Me-Not's API provides all the data permanence for the system. To do this GAE's datastore, a database application based largely on SQL, integrates with the python API via a class called NDB. The datastore is the recommended platform for database applications within Google's App Engine such as Forget-Me-Not. The datastore stores objects as entities. These entities can define any number of fields they wish to store as well of the type of that field. The entity is its own class, so it can define functions to abstract the details of storage and retrieval from the API. The datastore has filtering and sorting features so that objects can be added and retrieved from the datastore with ease. The API defines three entities for storage: Alarm, Message and Notification. The Alarm and Message entities simply store the JSON key-value pairs associated with that object. The Notification class uses the datastore, but has other functionality and should be discussed on its own

Notification Class

The Notification class handles all push notification related actions. The class is notified whenever the warnings or tokens endpoint is reached in the REST API. Upon receiving the signal, the Notification object checks its list of authorized tokens and contacts the Firebase API to push the notification to each device. Firebase is a large utility with much functionality. Most of its details are abstracted away from the Forget-Me-Not project and do not belong in this report.

Testing and Development Strategies [SN]

Forget-Me-Not's design consists of many small modules, each with a concise set of responsibilities. This style of design solves some very large and overarching problems for software systems, but also introduces other challenges. Common low level coding mistakes, complex object interactions, various submodules communicating with each other all lead to inevitable problems. A well-structured testing strategy can alleviate these challenges and provide a robust debugging platform for current development and future iterations.

Three Methods to Test Code

Note that each section (Display, Mobile App and API) will provide details for specific tools and strategies that help execute these test procedures.

Unit Testing

Unit testing is the verification of the smallest possible unit of testable code, such as a specific function within a class. This helps surface low level bugs that often plague complex systems by helping pinpoint precisely where a bug has occurred. These tests are designed to be run in a vacuum. That is, all external factors are removed from the system under test (SUT). Actual objects or classes that the function depends on should be mocked with stand-in objects. A mock object is designed to be a perfect clone from the perspective of the SUT, but it behaves exactly as the developer expects. Moreover, future changes to the dependent classes may result in tests failing with very inconclusive results. Mocks ensure that external changes will never break a functioning unit test. Various tools and mocking libraries exist to solve this problem, though mocks must be manually constructed in some instances. Frequently run unit tests will comprise that majority of this project's testing platform.

Integration Testing

Integration testing focuses on larger modules or classes as they are brought together. Integration tests highlight higher level bugs such as failures between the View, View-Model and the Model. This will be performed by writing larger scale test cases where actual objects are used rather than mocks. However, care must still be taken to construct atomic test cases. Testing more than one function or feature will lead to inconclusive results. Integration tests will be run each time a new module is added or an existing one altered.

End To End Testing

End to end testing requires developers to use the device verifying that the end to end experience meets design requirements. For a complex user facing system, testing will have to be done manually for the most part. End to end testing is the most expensive and time consuming method. Therefore, the need for end to end testing will be minimized by maximizing unit and integration testing.

Display and Mobile App Testing Tools

Google's Testing Framework

- Allows unit tests to be constructed as features are developed
- Provides a robust and simple means to mock objects and organize tests with scripts or within most IDEs
- Libraries implemented in most modern programming languages allows the same framework to be used for all other sections

Microsoft Visual Studio and Xamarin Studio

- IDEs with support for testing frameworks
- Can easily run all tests from a single location, helping to lighten the burden of testing.

API Testing Tools

Postman

Postman is a tool designed to help with the development and testing of APIs that use the HTTP protocol. It provides a set of tools that allows users to easily send requests without having to type out every part of the request such as headers and body statements. Postman also has a testing suite which allows testers to construct string of various request types and confirm valid results. These tests cases can be easily developed within Postman's GUI interface then exported as JSON. Exporting the tests as JSON allows us to incorporate the tests with a script or automation service so that they can be run without the need for a human to trigger them manually.

Jenkins

Testing can be a very taxing aspect of a product’s development. Developers must constantly remember to run tests when it is appropriate, which is not always as simple as clicking a run button. This leads to lazy testing practices and more bugs making their way into production code. Jenkins is a test automation tool that can integrate with git, a software version control system. Each time a new piece of code is committed to a feature branch on git, Jenkins will be notified. This allows Jenkins to trigger any test suites it has been given. Once tests pass or fail the developers receive notifications from Jenkins with the results. Passing code is ready to be merged from the feature branch to the main development trunk. Jenkins does not care what language the code is in so it will be able to run any test that is written for any part of the project. The Forget-Me-Not team was focused on developing features and unfortunately were not able to incorporate a development operations platform such as jenkins into the project. However, if development continues, the groundwork is in place to add Jenkins functionality.

Operation and Maintenance Instructions

Operation of System

As described in more detail previously in this report, the system is composed of three primary components: the primary display, the wearable bracelet, and the mobile application. Standard operation of the system would involve the primary display being placed in a convenient location in the patient's home and the wearable bracelet being worn on the patient's wrist. As for patient interaction with the screen, it would be minimal. There are no necessary updates, no input beyond basic alarm dismissals, and no need for the patient to ever worry about the display portion of the system. As for the wearable bracelet, it is also very maintenance free, however it will require the patient to charge it every 24 hours. The charging is very simple and requires the patient to turn off the bracelet with the power switch and plug it into a standard micro USB charger. When the device finishes charging, the patient can turn the switch back on and the bracelet will auto connect to the display within twenty seconds.

The mobile application is a bit more advanced than the display due to the fact that it will be used by the caretaker and it is responsible for the majority of the operation of the display. The mobile application can be run like any other application and maintains an intuitive user interface that allows the caretaker to set alarms and send messages to the display, as well as receive push notifications of critical events from the display. The application does not need to be running in the foreground in order for the functionality to remain working, so there is very little involvement from the caretaker once the alarms are setup properly.

Finally, as discussed previously, the system as a whole relies heavily on Google's Application Engine (GAE) to receive, send, and store data such as alarms and messages. At the time of demonstration an account was setup in order to have the functionality of the display as we needed for demonstration purposes. This account will no longer be active in a short time. Therefore, in order for the device to continue to be used, a GAE account must be setup with credentials and credit card information in order to use the services associated with this system.

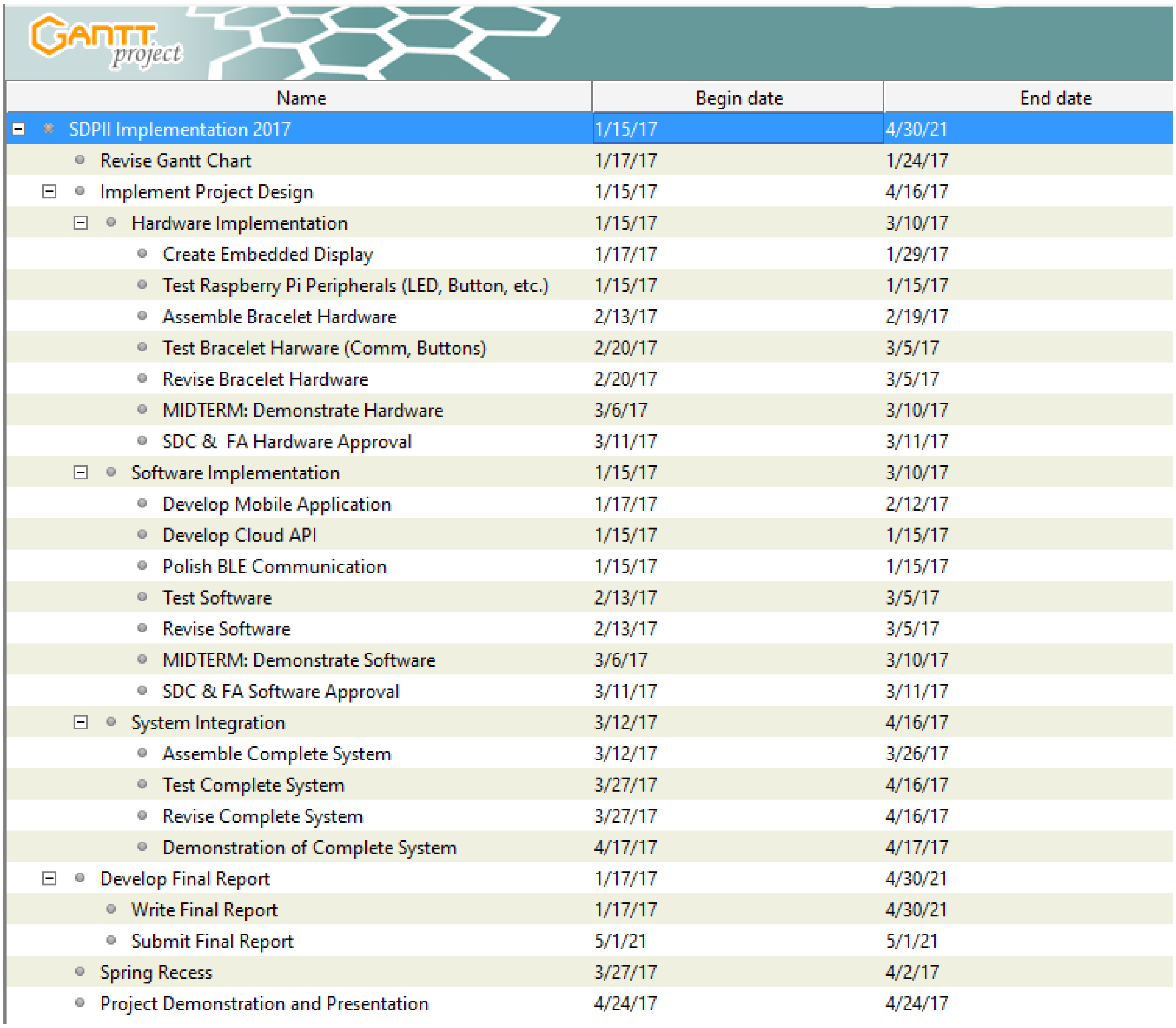
Maintenance of System

The system is very reliable when left to run as intended, however there are certain cases where minor maintenance may be required by the caretaker. In case of a power cycle for instance, the system will shut down and upon restarting, the program may need to be restarted as well if it fails to auto start at boot. This can be done by simply running the “Forget-Me-Not” executable file located on the desktop of the display. Once the program has started it will handle connecting to the bracelet and updating the information on the display.

Another small issue that often arises after an irregular restart is with the Bluetooth adapter on the Raspberry Pi that is driving the display. After a reboot it is often necessary to unplug and plug the USB Bluetooth adapter back into the Pi in order to reset it. Again, once this is done it should auto connect to the bracelet and everything will run smoothly.

Project Schedules

Gantt Chart - Design



Name	Begin date	End date
SDPII Implementation 2017	1/15/17	4/30/21
Revise Gantt Chart	1/17/17	1/24/17
Implement Project Design	1/15/17	4/16/17
Hardware Implementation	1/15/17	3/10/17
Create Embedded Display	1/17/17	1/29/17
Test Raspberry Pi Peripherals (LED, Button, etc.)	1/15/17	1/15/17
Assemble Bracelet Hardware	2/13/17	2/19/17
Test Bracelet Harware (Comm, Buttons)	2/20/17	3/5/17
Revise Bracelet Hardware	2/20/17	3/5/17
MIDTERM: Demonstrate Hardware	3/6/17	3/10/17
SDC & FA Hardware Approval	3/11/17	3/11/17
Software Implementation	1/15/17	3/10/17
Develop Mobile Application	1/17/17	2/12/17
Develop Cloud API	1/15/17	1/15/17
Polish BLE Communication	1/15/17	1/15/17
Test Software	2/13/17	3/5/17
Revise Software	2/13/17	3/5/17
MIDTERM: Demonstrate Software	3/6/17	3/10/17
SDC & FA Software Approval	3/11/17	3/11/17
System Integration	3/12/17	4/16/17
Assemble Complete System	3/12/17	3/26/17
Test Complete System	3/27/17	4/16/17
Revise Complete System	3/27/17	4/16/17
Demonstration of Complete System	4/17/17	4/17/17
Develop Final Report	1/17/17	4/30/21
Write Final Report	1/17/17	4/30/21
Submit Final Report	5/1/21	5/1/21
Spring Recess	3/27/17	4/2/17
Project Demonstration and Presentation	4/24/17	4/24/17

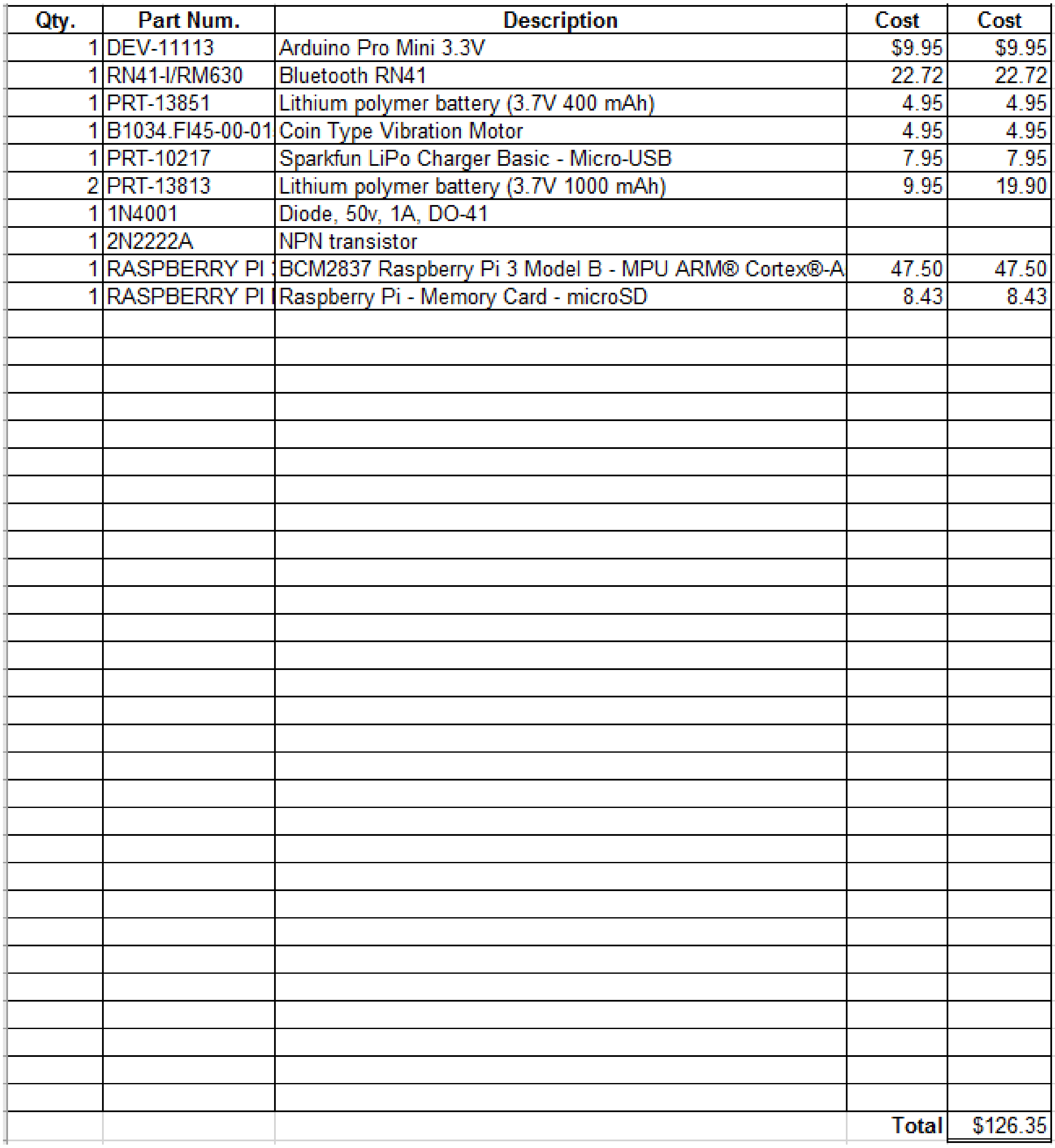
Figure 19 - Gantt Chart - Design

Gantt Chart – Implementation

Gantt Chart - Actual

Financial Budget

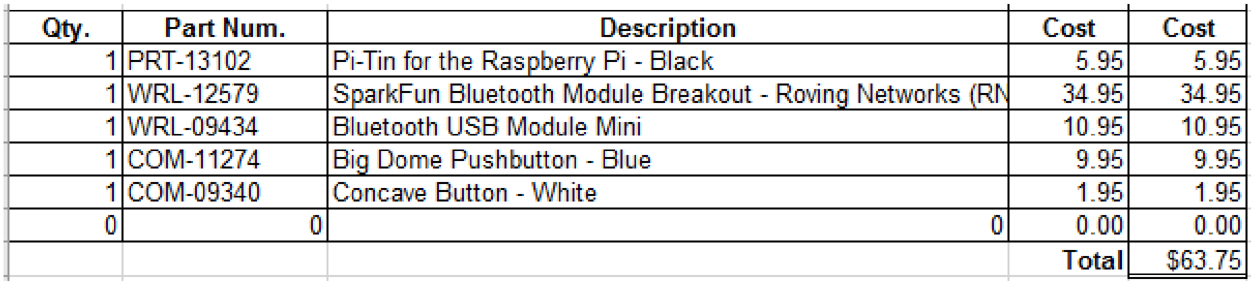
The maximum budget for this design was $400.00. After the initial design phase the estimated budget for this project was $126.35, as can be seen below in the initial part order form:



Qty.	Part Num.	Description	Cost	Cost
1	DEV-11113	Arduino Pro Mini 3.3V	$9.95	$9.95
1	RN41-I/RM630	Bluetooth RN41	22.72	22.72
1	PRT-13851	Lithium polymer battery (3.7V 400 mAh)	4.95	4.95
1	B1034.FI45-00-01	Coin Type Vibration Motor	4.95	4.95
1	PRT-10217	Sparkfun LiPo Charger Basic - Micro-USB	7.95	7.95
2	PRT-13813	Lithium polymer battery (3.7V 1000 mAh)	9.95	19.90
1	1N4001	Diode, 50v, 1A, DO-41		
1	2N2222A	NPN transistor		
1	RASPBERRY PI	BCM2837 Raspberry Pi 3 Model B - MPU ARM® Cortex®-A	47.50	47.50
1	RASPBERRY PI	Raspberry Pi - Memory Card - microSD	8.43	8.43
			Total	$126.35

Table 9 - Initial Financial Budget

However, after the implementation of the design as underway, there were a few more parts that were necessary to complete the project. These were small things that allowed the design to be better polished, such as a case for the Raspberry Pi, and added more reliability, such as a better Bluetooth adapter, to the design for a small cost. This secondary part order was for the amount of $63.75, as seen below:



Qty.	Part Num.	Description	Cost	Cost
1	PRT-13102	Pi-Tin for the Raspberry Pi - Black	5.95	5.95
1	WRL-12579	SparkFun Bluetooth Module Breakout - Roving Networks (RN	34.95	34.95
1	WRL-09434	Bluetooth USB Module Mini	10.95	10.95
1	COM-11274	Big Dome Pushbutton - Blue	9.95	9.95
1	COM-09340	Concave Button - White	1.95	1.95
0	0	0	0.00	0.00
			Total	$63.75

Table 10 - Additional Financial Budget

Therefore, while the estimated budget was $126.35, the actual budget ended up being $190.10 – a difference of $63.75. This number was still well below the maximum budget allotted of $400.00.

Design Team Information

Shawn Nicholson - Team Lead
Daniel Barber - Cironi - Archivist
Jake Kruse - Software Lead
Nicole Dent - Hardware Lead

Dr. Joan Carletta - Faculty Advisor

References

1. Mokhtari M, Aloulou H, Tiberghien T, Biswas J, Racoceanu D, Yap P, *New Trends to Support Independence in Persons with Mild Dementia – A Mini-Review*. Gerontology 2012;58:554-563

2. Philippa Riley, Norman Alm, Alan Newell, *An Interactive Tool to Promote Musical Creativity in People With Dementia*, *Computers in Human Behavior*, Volume 25, Issue 3, May 2009, Pages 599-608, ISSN 0747-5632, http://dx.doi.org/10.1016/j.chb.2008.08.014. (http://www.sciencedirect.com/science/article/pii/S0747563208001672)

3. *Atmega328 Run for a Year on Batteries*. (2012, November 8). Retrieved October 23, 2016, from http://electronics.stackexchange.com/questions/49182/how-can-i-get-my-atmega328-to-run-for-a-year-on-batteries. (http://electronics.stackexchange.com/questions/49182/how-can-i-get-my-atmega328-to-run-for-a-year-on-batteries)

4. *The Hitchhikers Guide to iBeacon Hardware*. (2015, May 4). Retrieved October 24, 2016, from (http://www.aislelabs.com/reports/beacon-guide/)

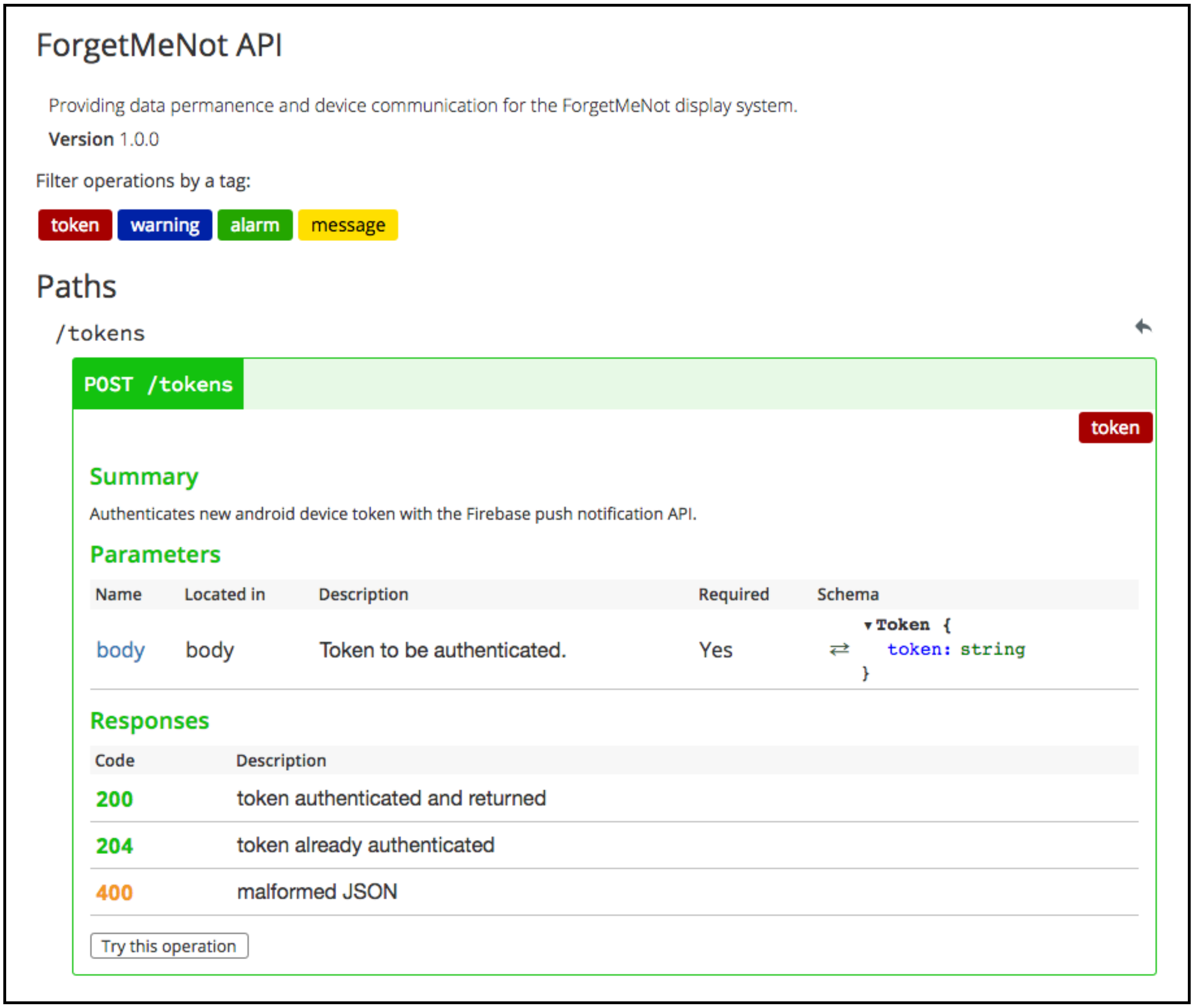
5. *2016 Alzheimer's Disease Facts and Figures*. (2016, December 6). Retrieved from Alzheimer's Association: (http://www.alz.org/facts/)

Appendices

Datasheet Reference

Component	Link to Datasheet/Schematics
Raspberry Pi 3 Model B	https://cdn.sparkfun.com/datasheets/Dev/RaspberryPi/2020826.pdf
RN42 Bluetooth Module	https://www.sparkfun.com/datasheets/Wireless/Bluetooth/rn-42-ds.pdf
Arduino Pro Mini 3.3V	https://www.arduino.cc/en/uploads/Main/Arduino-Pro-Mini-schematic.pdf

API Documentation



ForgetMeNot API

Providing data permanence and device communication for the ForgetMeNot display system.

Version 1.0.0

Filter operations by a tag:

Paths

/tokens

POST /tokens

Summary

Authenticates new android device token with the Firebase push notification API.

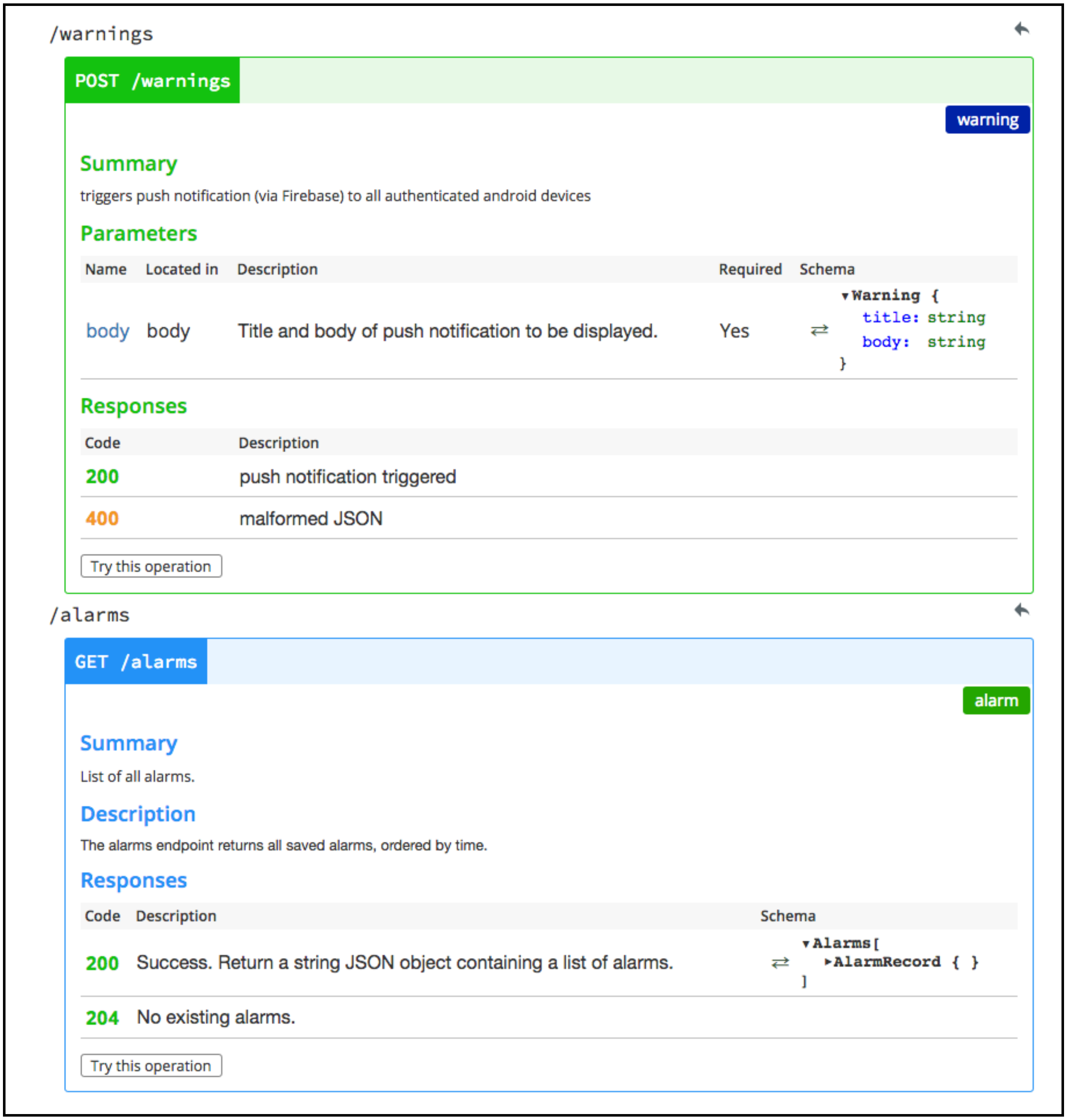
Parameters

Name	Located in	Description	Required	Schema

body	body	Token to be authenticated.	Yes	⇄ ▾Token {
				token: string
				}

Responses

Code	Description
200	token authenticated and returned
204	token already authenticated
400	malformed JSON



/warnings

POST /warnings

warning

Summary

triggers push notification (via Firebase) to all authenticated android devices

Parameters

Name	Located in	Description	Required	Schema

body	body	Title and body of push notification to be displayed.	Yes	⇄ ▾Warning {
				title: string
				body: string
				}

Responses

Code	Description
200	push notification triggered
400	malformed JSON

Try this operation

/alarms

GET /alarms

alarm

Summary

List of all alarms.

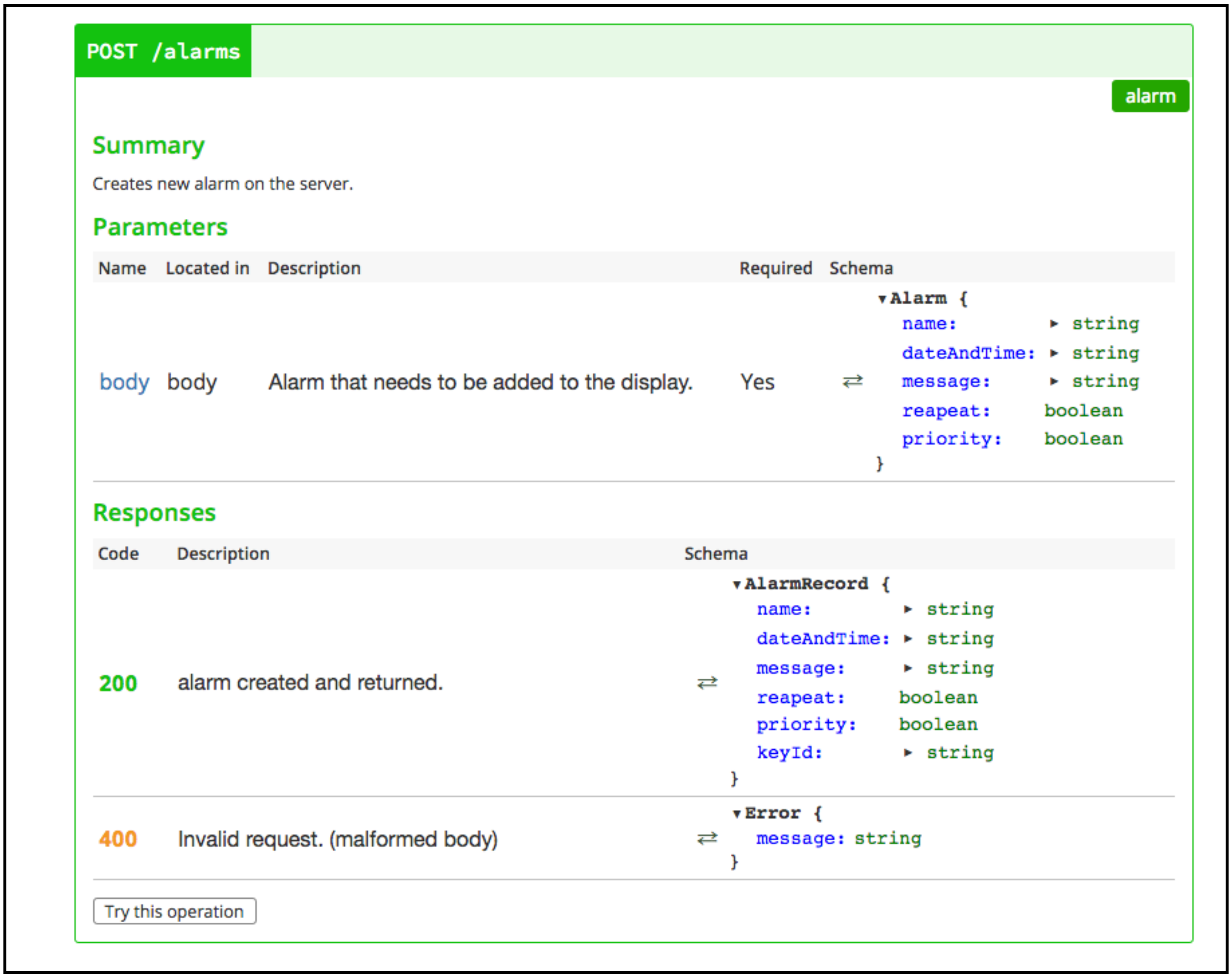
Description

The alarms endpoint returns all saved alarms, ordered by time.

Responses

Code	Description	Schema
200	Success. Return a string JSON object containing a list of alarms.	⇄ ▾Alarms[▸AlarmRecord { }]
204	No existing alarms.	

Try this operation



POST /alarms

Summary

Creates new alarm on the server.

Parameters

Name	Located in	Description	Required	Schema

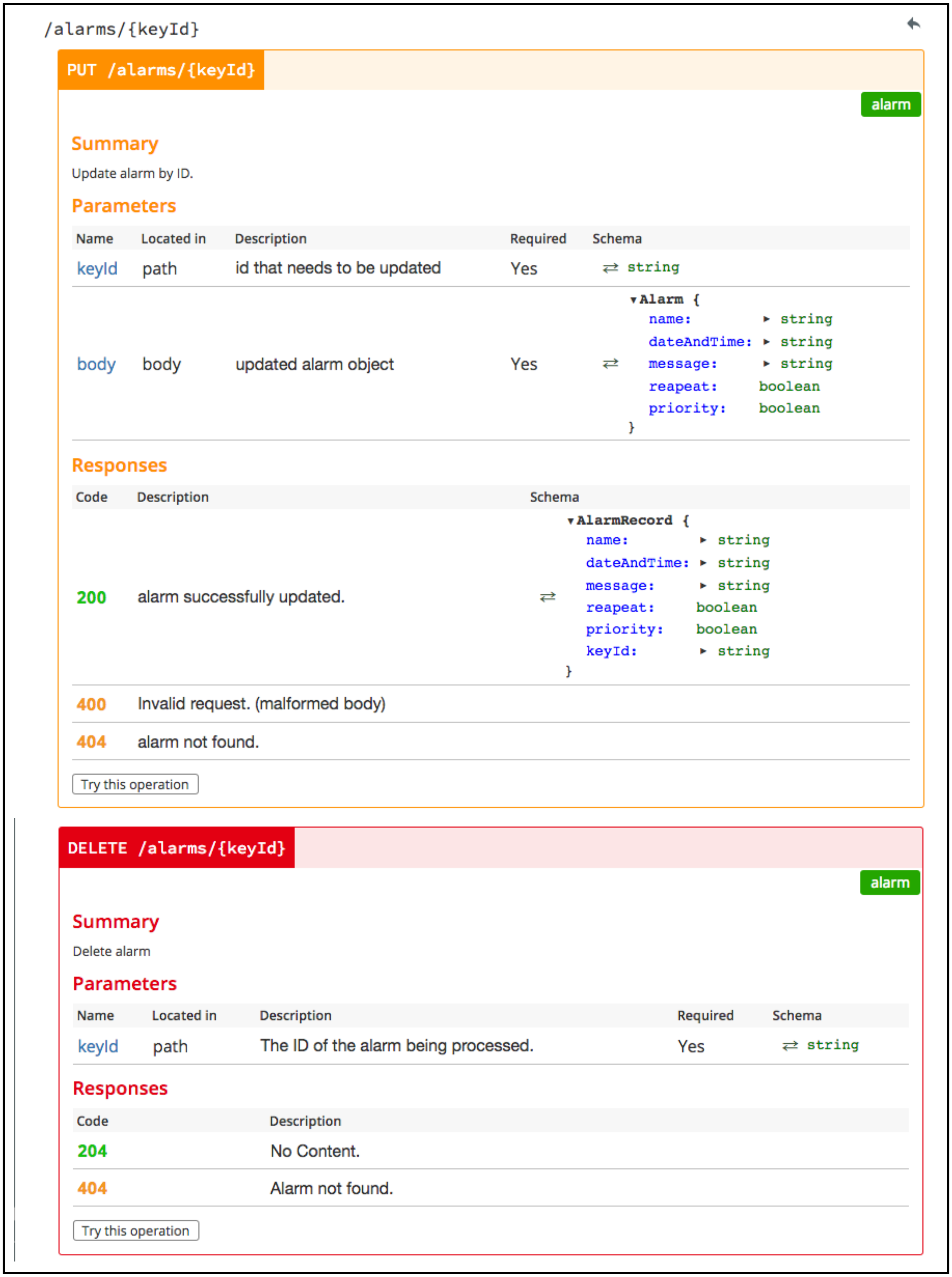
body	body	Alarm that needs to be added to the display.	Yes	⇄ ▾Alarm {
				name: ▸ string
				dateAndTime: ▸ string
				message: ▸ string
				reapeat: boolean
				priority: boolean
				}

Responses

Code	Description	Schema

200	alarm created and returned.	⇄ ▾AlarmRecord {
		name: ▸ string
		dateAndTime: ▸ string
		message: ▸ string
		reapeat: boolean
		priority: boolean
		keyId: ▸ string
		}

400	Invalid request. (malformed body)	⇄ ▾Error {
		message: string
		}



Summary

Update alarm by ID.

Parameters

Name	Located in	Description	Required	Schema
keyId	path	id that needs to be updated	Yes	⇄ string

body	body	updated alarm object	Yes	⇄ ▾Alarm {
				name: ▸ string
				dateAndTime: ▸ string
				message: ▸ string
				reapeat: boolean
				priority: boolean
				}

Responses

Code	Description	Schema

200	alarm successfully updated.	⇄ ▾AlarmRecord {
		name: ▸ string
		dateAndTime: ▸ string
		message: ▸ string
		reapeat: boolean
		priority: boolean
		keyId: ▸ string
		}

400	Invalid request. (malformed body)	
404	alarm not found.	

Summary

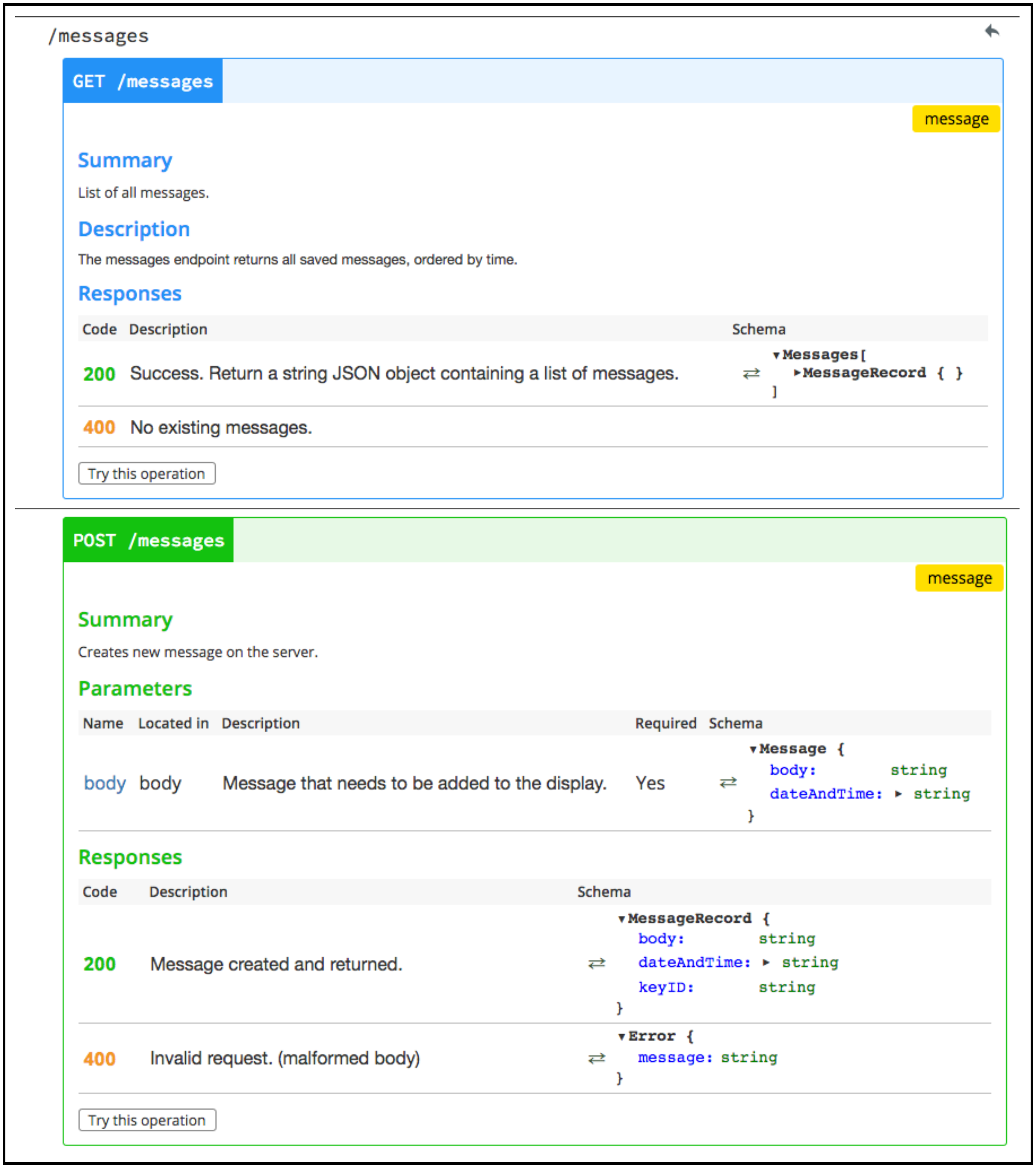
Delete alarm

Parameters

Name	Located in	Description	Required	Schema
keyId	path	The ID of the alarm being processed.	Yes	⇄ string

Responses

Code	Description
204	No Content.
404	Alarm not found.



/messages

GET /messages

message

Summary

List of all messages.

Description

The messages endpoint returns all saved messages, ordered by time.

Responses

Code	Description	Schema
200	Success. Return a string JSON object containing a list of messages.	⇄ ▾Messages[▸MessageRecord { }]
400	No existing messages.	

Try this operation

POST /messages

message

Summary

Creates new message on the server.

Parameters

Name	Located in	Description	Required	Schema

body	body	Message that needs to be added to the display.	Yes	⇄ ▾Message {
				body: string
				dateAndTime: ▸ string
				}

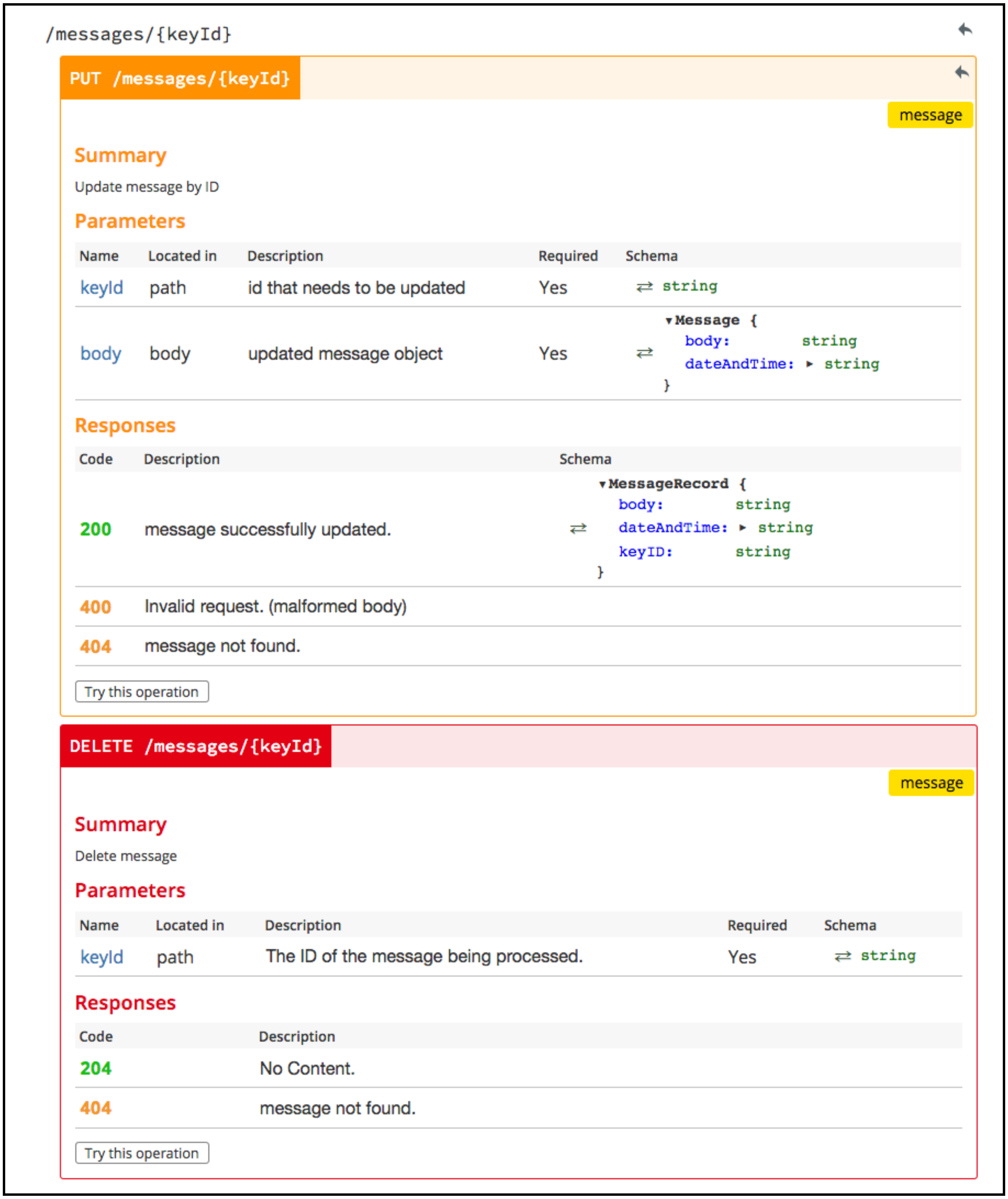
Responses

Code	Description	Schema

200	Message created and returned.	⇄ ▾MessageRecord {
		body: string
		dateAndTime: ▸ string
		keyID: string
		}

400	Invalid request. (malformed body)	⇄ ▾Error {
		message: string
		}

Try this operation



/messages/{keyId}

PUT /messages/{keyId}

Summary

Update message by ID

Parameters

Name	Located in	Description	Required	Schema
keyId	path	id that needs to be updated	Yes	⇄ string

body	body	updated message object	Yes	⇄ ▾Message {
				body: string
				dateAndTime: ▸ string
				}

Responses

Code	Description	Schema

200	message successfully updated.	⇄ ▾MessageRecord {
		body: string
		dateAndTime: ▸ string
		keyID: string
		}

400	Invalid request. (malformed body)	
404	message not found.	

DELETE /messages/{keyId}

Summary

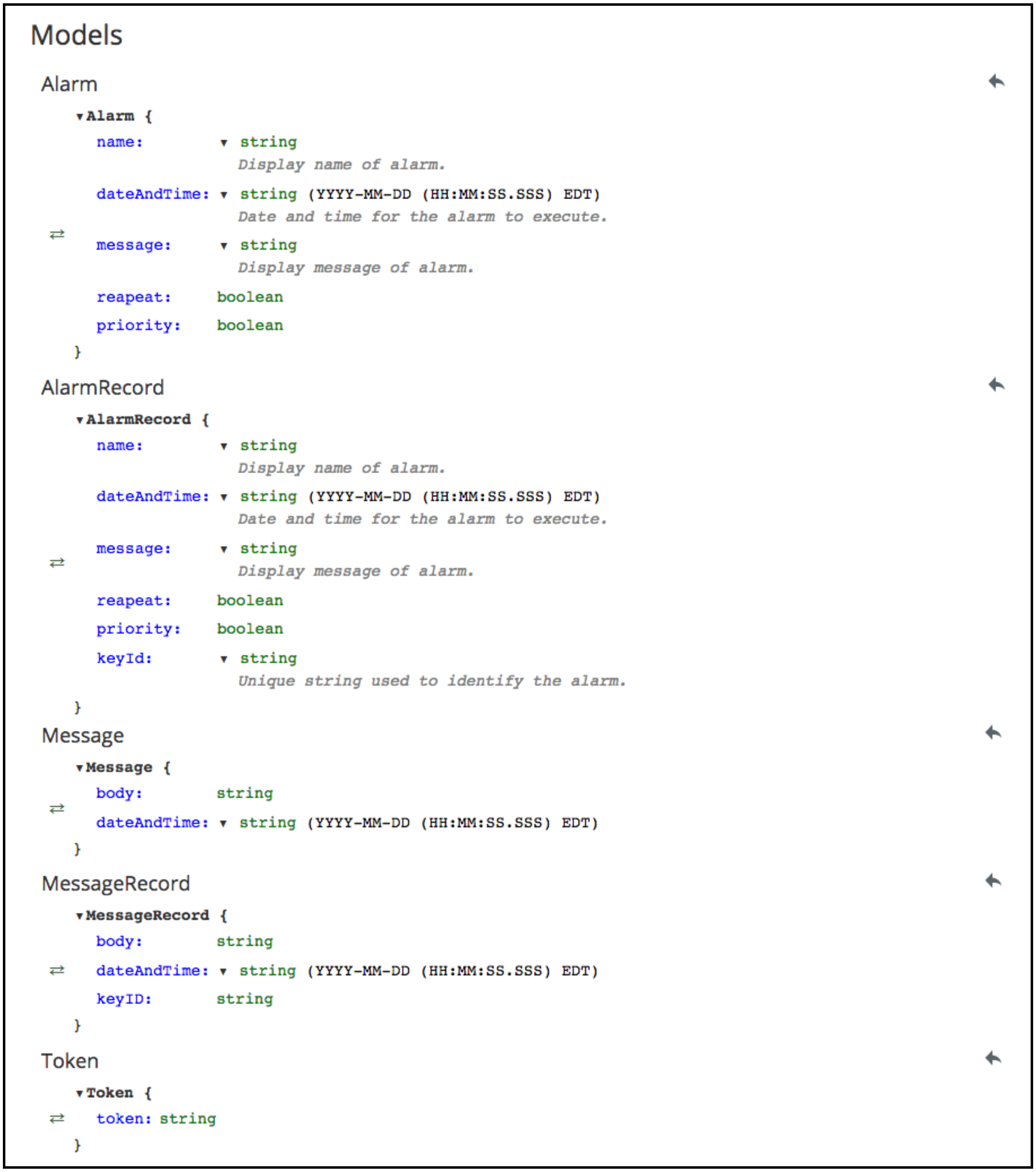
Delete message

Parameters

Name	Located in	Description	Required	Schema
keyId	path	The ID of the message being processed.	Yes	⇄ string

Responses

Code	Description
204	No Content.
404	message not found.



Models

Alarm

```
▾Alarm {
  name:        ▾ string
                 Display name of alarm.
  dateAndTime: ▾ string (YYYY-MM-DD (HH:MM:SS.SSS) EDT)
                 Date and time for the alarm to execute.
  message:     ▾ string
                 Display message of alarm.
  reapeat:     boolean
  priority:    boolean
}
```

AlarmRecord

```
▾AlarmRecord {
  name:        ▾ string
                 Display name of alarm.
  dateAndTime: ▾ string (YYYY-MM-DD (HH:MM:SS.SSS) EDT)
                 Date and time for the alarm to execute.
  message:     ▾ string
                 Display message of alarm.
  reapeat:     boolean
  priority:    boolean
  keyId:       ▾ string
                 Unique string used to identify the alarm.
}
```

Message

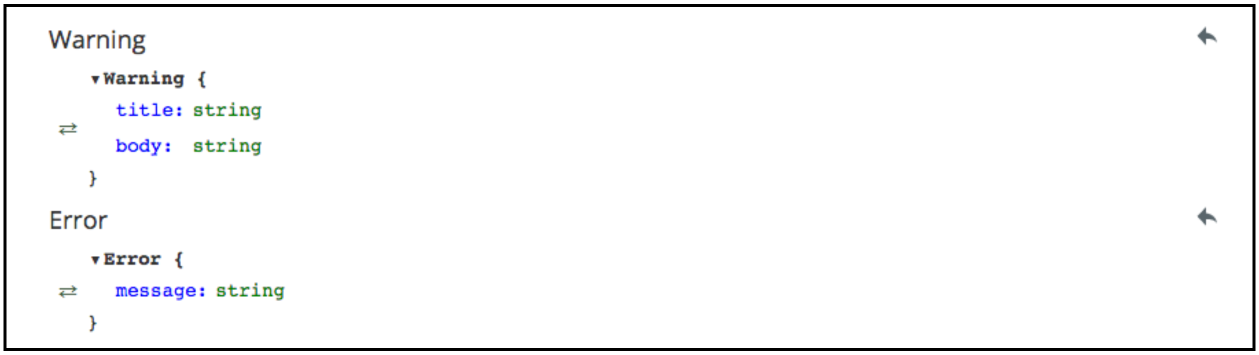
```
▾Message {
  body:        string
  dateAndTime: ▾ string (YYYY-MM-DD (HH:MM:SS.SSS) EDT)
}
```

MessageRecord

```
▾MessageRecord {
  body:        string
  dateAndTime: ▾ string (YYYY-MM-DD (HH:MM:SS.SSS) EDT)
  keyID:       string
}
```

Token

```
▾Token {
  token: string
}
```



```
Warning
    ▾Warning {
      title: string
⇄
      body:  string
    }
Error
    ▾Error {
⇄     message: string
    }
```

YAML Code to Produce Interactive Documentation

```
swagger: '2.0'
info:
  title: Forget-Me-Not API
  description: Providing data permanence and device communication for the Forget-Me-Not display system.
  version: "1.0.0"
host: cloud.google.com
schemes:
  - https
basePath: /v1
produces:
  - application/json
paths:
  /alarms:
    get:
      summary: List of all alarms.
      description: |
        The alarms endpoint returns information about the current existing alarms. The responses include the alarm ID and a alarm name. Currently, the API only supports a single alarm, so this endpoint will always return an array with a single alarm element containing {"alarmId" = "1","alarmName":"defaultAlarm"}. This will be expanded upon in future versions.
      tags:
        - alarm
      responses:
        200:
          description: Success. Return an array of alarms.
          schema:
            title: Alarms
            type: array
            items:
              $ref: '#/definitions/AlarmSummary'
        204:
          description: No existing alarms.
    post:
      summary: Creates new alarm (or replaces current existing one).
      description: Adds a new alarm to the display.
      tags:
        - alarm
      parameters:
        - in: body
          name: body
          description: Alarm that needs to be added to the display.
          required: false
          schema:
            $ref: "#/definitions/Alarm"
      responses:
        201:
          description: alarm created and returned.
          schema:
            $ref: '#/definitions/AlarmRecord'
        400:
          description: Invalid request. (malformed body)
          schema:
            $ref: '#/definitions/Error'
  /alarms/{alarmId}:
    get:
      tags:
        - alarm
      summary: Find alarm by ID
      produces:
        - application/json
      parameters:
        - in: path
          name: alarmId
          description: ID of alarm that needs to be fetched.
          required: true
          type: integer
          format: int64
      responses:
        200:
          description: Alarm found. Return alarm.
          schema:
            $ref: "#/definitions/Alarm"
        404:
          description: Alarm not found.
    put:
      tags:
        - alarm
      summary: Update alarm by ID
      parameters:
        - in: path
          name: alarmId
          description: id that needs to be updated
          required: true
          type: string
        - in: body
          name: body
          description: updated alarm object
          required: false
          schema:
            $ref: "#/definitions/Alarm"
      responses:
        201:
          description: alarm successfully updated.
          schema:
            $ref: '#/definitions/Alarm'
    400:
     description: Invalid request. (malformed body)
    404:
     description: alarm not found.
  delete:
   tags:
    - alarm
   summary: Delete alarm
   parameters:
    - in: path
     name: alarmId
     description: The ID of the alarm being processed.
     required: true
     type: string
   responses:
    204:
     description: No Content.
    404:
     description: Alarm not found.
    500:
     description: Internal server error. (request was good, but something went wrong)

definitions:
 AlarmSummary:
  properties: &ALARM
   alarmId:
    type: string
    description: Unique identifier representing a specific alarm.
   alarmName:
    type: string
    description: Display name of alarm.
 Alarm:
  properties:
   alarmName:
    type: string
    description: Any id/number/string used to identify the alarm
   alarmTime:
    type: string
    description: The time at which the alarm is meant to sound.
   alarmMessage:
    type: string
    description: Unique identifier representing a specific Tech.
 AlarmRecord:
  properties:
   <<: *ALARM
   alarmId:
    type: string
    description: Any id/number/string used to identify the alarm.
   alarmTime:
    type: string
    description: The time at which the alarm is meant to sound.
   alarmMessage:
    type: string
    description: Unique identifier representing a specific Tech.
 Error:
  type: object
  properties:
    message:
      type: string
```

Primary Display Source Code

Note that the entire source code can be found in an online repository at https://bitbucket.org/jaguar_224/forgetmenot. Below are the header files that illustrate the functionality of the code.

mainWindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QObject>
#include <qtimer.h>
#include <QAbstractItemDelegate>
#include <QHash>
#include "ui_mainwindow.h"
#include "weather.h"
#include "message.h"
#include "alarm.h"
#include "bluetoothmanager.h"
#include "alarmdismissal.h"
#include "apiinterface.h"
#include <wiringPi.h>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
    Ui::MainWindow *ui;
    void connectToBT();
    int _buttonState = 0;
    const QColor c = QColor(0, 0, 149);

private:
    weather _weather;
    QList<message> _messages;
    QList<alarm> _alarms;
    QList<alarm> _activeAlarms;
    QList<alarm> _passiveAlarms;
    //_isAlarmSounded is a QHash that pairs the keyid with a bool that determines whether or not the alarm
has sounded
    QHash<QString, bool> _isAlarmSounded;
    BluetoothManager * _bluetooth;
    AlarmDismissal *_alarmDismissalWindow;
    int _isAlarmActive;
    int _isPassiveAlarmActive;

    QTimer * clockTimer;
    QTimer * weatherTimer;
    QTimer * getAlarmsTimer;
    QTimer * getMessagesTimer;
    QTimer * alarmCheckTimer;
    QTimer * buttonPressTimer;
    QTimer * _passiveAlarmTimer;
    QTimer * _activeAlarmTimer;

public slots:
    void getMessages();
    void getAlarms();
    void getWeather();
    void updateDateTime();
    void dismissActiveAlarm();
    void dismissPassiveAlarm();
    void checkForAlert();
    void cycleGpio();
    void sendAlert();
    void dismissAlert();
    void checkForButtonPress();
    void sendEmergencyNotification();
};

#endif // MAINWINDOW_H
```

alarm.h

```
#ifndef ALARM_H
#define ALARM_H

#include <QWidget>
#include <QListWidget>
#include <QObject>
#include <QString>
#include <QDateTime>
#include <vector>
#include "ui_mainwindow.h"

class alarm : public QListWidgetItem
{
private:
    QString _keyId;
    QString _name;
    QString _message;
    std::vector<bool> _repeat;
    bool _priority;
    QDateTime _dateAndTime;

public:
    alarm();
    alarm(QString name, QString message, std::vector<bool> repeat,  bool priority, QDateTime
dateAndTime);
    alarm(QString keyId, QString name, QString message, std::vector<bool> repeat,  bool priority,
QDateTime dateAndTime);
    ~alarm();
    void setName(QString name);
    void setMessage(QString message);
    void setRepeat(std::vector<bool> repeat);
    void setDateAndTime(QDateTime dateAndTime);
    void setPriority(bool priority);
    QString getKeyId() const;
    QString getName() const;
    QString getMessage() const;
    std::vector<bool> getRepeat() const;
    bool getPriority() const;
    QDateTime getDateAndTime() const;
    QString getDateAndTimeString();
    bool repeatingPrefSetForToday();
};

#endif // ALARM_H

#ifndef ALARMDELAGATE_H
#define ALARMDELAGATE_H
#include <QPainter>
#include <QStyledItemDelegate>
#include <QAbstractItemDelegate>

//This is the code for the custom Alarm Delagate
//The custom Alarm Delagte defines how the message is rendered in the QListView
class alarmDelagate: public QStyledItemDelegate
{
public:
    alarmDelagate(); //: QAbstractItemDelegate(parent);
    void paint(QPainter* painter, const QStyleOptionViewItem& option, const QModelIndex& index)
const;
    QSize sizeHint(const QStyleOptionViewItem &option, const QModelIndex &index) const;

    virtual ~alarmDelagate();

};

#endif // ALARMDELAGATE_H
```

alarmDismissal.h

```
#ifndef ALARMDISMISSAL_H
#define ALARMDISMISSAL_H

#include <QDialog>
#include <QTimer>
#include <wiringPi.h>
#include "alarm.h"

namespace Ui {
class AlarmDismissal;
}

class AlarmDismissal : public QDialog
{
    Q_OBJECT

public:
    AlarmDismissal(alarm a);
    ~AlarmDismissal();

private:
    Ui::AlarmDismissal *ui;
    alarm _alarm;
    int _buttonStateAlarmScreen;
    QTimer * buttonPressTimer;
    QTimer * closeTimer;

private slots:
    void checkForButtonPress();
    void closeScreen();
};

#endif // ALARMDISMISSAL_H
```

apiInterface.h

```
#ifndef APIINTERFACE_H
#define APIINTERFACE_H

#include <QObject>
#include <QJsonObject>

class ApiInterface : public QObject
{
    Q_OBJECT
public:
    explicit ApiInterface(QObject *parent = 0);
    static QJsonObject sendRequest(QString url);
    static QJsonObject postAlert(QString messageString);
    static QPixmap iconRequest(QString imageUrl);
    static QJsonDocument sendRequestForMessages(QString url);
};

#endif // APIINTERFACE_H
```

bluetoothManager.h

```
#ifndef BLUETOOTHMANAGER_H
#define BLUETOOTHMANAGER_H

#include <QObject>
#include <QTimer>
#include <QBluetoothDeviceDiscoveryAgent>
#include <QBluetoothAddress>
#include <QBluetoothSocket>
#include "apiinterface.h"

class BluetoothManager : public QObject
{
    Q_OBJECT
public:
    explicit BluetoothManager(QObject *parent = 0);
    ~BluetoothManager();
    void startDeviceDiscovery();
    void sendMessage(const QString &message);

public slots:

private:
    QBluetoothDeviceDiscoveryAgent *discoveryAgent;
    const QBluetoothAddress * dongleAddress;
    const QBluetoothAddress * braceletAddress;
    QBluetoothSocket * socket;
    void setupSocket();
    QTimer * bluetoothReconnect;

private slots:
    void deviceDiscovered(const QBluetoothDeviceInfo &device);
    void readSocket();
    void connected();
    void disconnected();
    void reconnect();
};

#endif // BLUETOOTHMANAGER_H
```

message.h

```
#ifndef MESSAGES_H
```

```
#define MESSAGES_H

#include <QWidget>
#include <QListWidget>
#include <QObject>
#include <QDateTime>
#include <string>
#include "ui_mainwindow.h"
#include "apiinterface.h"

class message : public QListWidgetItem
{
public:
    message();
    message(QString content, QDateTime dateTime);
    void setContent(QString content);
    void setDateTime(QDateTime datetime);
    QString getContent();
    QDateTime getDateTime();
    QString getDateTimeString();
    ~message();

private:
    QString content;
    QDateTime dateAndTime;
};

#endif // MESSAGES_H

#ifndef MESSAGEDELAGATE_H
#define MESSAGEDELAGATE_H
#include <QPainter>
#include <QStyledItemDelegate>
#include <QAbstractItemDelegate>

//This is the code for the custom Message Delagate
//The custom Message Delagte defines how the message is rendered in the QListView
class messageDelagate: public QStyledItemDelegate
{
public:
    messageDelagate(); //: QAbstractItemDelegate(parent);
    void paint(QPainter* painter, const QStyleOptionViewItem& option, const QModelIndex& index)
const;
    QSize sizeHint(const QStyleOptionViewItem &option, const QModelIndex &index) const;

    virtual ~messageDelagate();

};

#endif // MESSAGEDELAGATE_H
```

weather.h

```
#ifndef WEATHER_H
#define WEATHER_H

#include <QWidget>
#include <QObject>
#include <string>
#include "ui_mainwindow.h"
#include "apiinterface.h"

class weather
{
public:
    weather();
    void setTemp(double temp);
    void setStation(QString station);
    double getTemp();
    QString getStation();
    ~weather();

private:
    double temp;
    QString station;
};

#endif // WEATHER_H
```

Mobile Application Source Code

Alarm Editor Page

```
class AlarmEditorViewModel : INotifyPropertyChanged, IDisposable
  {
    public const int DAYSINAWEEK = 7;
    private Alarm _alarm;
    private bool _newAlarm;

    public delegate void UnhandledExceptionEventHandler(object sender, EventArgs e);
    public event UnhandledExceptionEventHandler UnhandledException;

    public delegate void PopupPageCloseEventHandler(object sender, EventArgs e);
    public event PopupPageCloseEventHandler ClosePopupPage;

    public delegate void ShowMessageEvent(string message, EventArgs e);
    public event ShowMessageEvent ShowMessage;

    public event PropertyChangedEventHandler PropertyChanged;
    private void OnPropertyChanged(string nameOfProperty)
    {
      if (PropertyChanged != null)
        PropertyChanged(this, new PropertyChangedEventArgs(nameOfProperty));
    }
    public AlarmEditorViewModel()
    {
      _newAlarm = true;
      _alarm = new Alarm();
      SaveTappedCommand = new Command<object>((key) => OnSaveTapped());
      DeleteTappedCommand = new Command<object>((key) => OnDeleteTapped());
      CancelTappedCommand = new Command<object>((key) => OnCancelTapped());
      RepeatDayTappedCommand = new Command<object>((key) => OnRepeatDayTapped(key));
    }

    public AlarmEditorViewModel(Alarm alarm)
    {
      _newAlarm = false;
      SaveTappedCommand = new Command<object>((key) => OnSaveTapped());
      DeleteTappedCommand = new Command<object>((key) => OnDeleteTapped());
      CancelTappedCommand = new Command<object>((key) => OnCancelTapped());
      RepeatDayTappedCommand = new Command<object>((key) => OnRepeatDayTapped(key));
      _alarm = new Alarm();
      _alarm = alarm;
    }

    ~AlarmEditorViewModel()
                  {
      this.Dispose();
                  }
```

```
public Alarm Alarm
{
    get
    {
        return _alarm;
    }
    set
    {
        _alarm = value;
        OnPropertyChanged("Alarm");
    }
}

public DateTime CurrentDate
{
    get
    {
        return DateTime.Now;
    }
    set
    {

    }
}

public TimeSpan AlarmTime
{
    get
    {
        return _alarm.dateAndTime.TimeOfDay;
    }
    set
    {
        //adds the selected date with the timespan selected by the user
        _alarm.dateAndTime = _alarm.dateAndTime.Date + value;
    }
}

public DateTime AlarmDate
{
    get
    {
        return _alarm.dateAndTime.Date;
    }
    set
    {
        DateTime tmp = value;
        _alarm.dateAndTime = tmp + _alarm.dateAndTime.TimeOfDay;
    }
}

public bool DateIsVisible
{
    get
    {
        if (_alarm.repeat.All(day => day == false))
            return true;
        else
            return false;
    }
    set { }
}

public bool DeleteButtonVisible
{
    get
    {
        //If the alarm is a brand new alarm there should not be a delete functionallity
        //as it does not exist in the api yet
        return !_newAlarm;
```

```
            }
            set { }
        }

        public string SundayImage
        {
            get
            {
                if (_alarm.repeat[(int)DaysOfTheWeek.Sunday])
                    return "SundayChecked.png";
                else
                    return "SundayUnChecked.png";
            }
            set { }
        }

        public string MondayImage
        {
            get
            {
                if (_alarm.repeat[(int)DaysOfTheWeek.Monday])
                    return "MondayChecked.png";
                else
                    return "MondayUnChecked.png";
            }
            set { }
        }

        public string TuesdayImage
        {
            get
            {
                if (_alarm.repeat[(int)DaysOfTheWeek.Tuesday])
                    return "TuesdayChecked.png";
                else
                    return "TuesdayUnChecked.png";
            }
            set { }
        }

        public string WednesdayImage
        {
            get
            {
                if (_alarm.repeat[(int)DaysOfTheWeek.Wednesday])
                    return "WednesdayChecked.png";
                else
                    return "WednesdayUnChecked.png";
            }
            set { }
        }

        public string ThursdayImage
        {
            get
            {
                if (_alarm.repeat[(int)DaysOfTheWeek.Thursday])
                    return "ThursdayChecked.png";
                else
                    return "ThursdayUnChecked.png";
            }
            set { }
        }

        public string FridayImage
        {
            get
            {
                if (_alarm.repeat[(int)DaysOfTheWeek.Friday])
                    return "FridayChecked.png";
                else
                    return "FridayUnChecked.png";
            }
            set { }
        }

        public string SaturdayImage
        {
            get
            {
                if (_alarm.repeat[(int)DaysOfTheWeek.Saturday])
                    return "SaturdayChecked.png";
                else
                    return "SaturdayUnChecked.png";
            }
            set { }
        }

        public ICommand SaveTappedCommand { get; set; }

        public async void OnSaveTapped()
        {
            if (_alarm != null)
            {
                if (string.IsNullOrEmpty(_alarm.name))
                {
                    ShowMessage("The alarm must have a title.", null);
                    return;
                }

                try
                {
                    if (_newAlarm)
                    {
                        JsonValue json = await ApiViewModel.PostAsync(_alarm, ApiClasses.Alarms);
                    }
                    else
                    {
                        JsonValue json = await ApiViewModel.PutAsync(_alarm, ApiClasses.Alarms);
                    }
                }
                catch (Exception ex)
                {
                    UnhandledException?.Invoke(ex, null);
                }
            }

            if (ClosePopupPage != null)
                ClosePopupPage(this, EventArgs.Empty);
        }

        public ICommand DeleteTappedCommand { get; set; }

        public async void OnDeleteTapped()
        {
            try
            {
                JsonValue response = await ApiViewModel.DeleteAsync(_alarm.keyId, ApiClasses.Alarms);
                if (ClosePopupPage != null)
                    ClosePopupPage(this, EventArgs.Empty);
            }
            catch (Exception ex)
            {
                UnhandledException?.Invoke(ex, null);
            }
        }

        public ICommand CancelTappedCommand { get; set; }

        public void OnCancelTapped()
```

```
{
    try
    {
        if (ClosePopupPage != null)
            ClosePopupPage(this, EventArgs.Empty);
    }
    catch (Exception ex)
    {
        UnhandledException?.Invoke(ex, null);
    }
}

public ICommand RepeatDayTappedCommand { get; set; }

public void OnRepeatDayTapped(object day)
{
    try
    {
        switch (int.Parse(day.ToString()))
        {
            case 0:
                _alarm.repeat[(int)DayOfWeek.Sunday] = !_alarm.repeat[(int)DayOfWeek.Sunday];
                OnPropertyChanged("SundayImage");
                OnPropertyChanged("DateIsVisible");
                break;

            case 1:
                _alarm.repeat[(int)DayOfWeek.Monday] = !_alarm.repeat[(int)DayOfWeek.Monday];
                OnPropertyChanged("MondayImage");
                OnPropertyChanged("DateIsVisible");
                break;

            case 2:
                _alarm.repeat[(int)DayOfWeek.Tuesday] = !_alarm.repeat[(int)DayOfWeek.Tuesday];
                OnPropertyChanged("TuesdayImage");
                OnPropertyChanged("DateIsVisible");
                break;

            case 3:
                _alarm.repeat[(int)DayOfWeek.Wednesday] = !_alarm.repeat[(int)DayOfWeek.Wednesday];
                OnPropertyChanged("WednesdayImage");
                OnPropertyChanged("DateIsVisible");
                break;

            case 4:
                _alarm.repeat[(int)DayOfWeek.Thursday] = !_alarm.repeat[(int)DayOfWeek.Thursday];
                OnPropertyChanged("ThursdayImage");
                OnPropertyChanged("DateIsVisible");
                break;

            case 5:
                _alarm.repeat[(int)DayOfWeek.Friday] = !_alarm.repeat[(int)DayOfWeek.Friday];
                OnPropertyChanged("FridayImage");
                OnPropertyChanged("DateIsVisible");
                break;

            case 6:
                _alarm.repeat[(int)DayOfWeek.Saturday] = !_alarm.repeat[(int)DayOfWeek.Saturday];
                OnPropertyChanged("SaturdayImage");
                OnPropertyChanged("DateIsVisible");
                break;

            default:
                break;
        }
    }
    catch (Exception ex)
    {
        UnhandledException?.Invoke(ex, null);
    }
```

```
        }
        public void Dispose()
        {
            PropertyChanged = null;
            Alarm = null;
            SundayImage = MondayImage = TuesdayImage = WednesdayImage = ThursdayImage = FridayImage = SaturdayImage = null;
            _alarm = null;
            SaveTappedCommand = null;
            DeleteTappedCommand = null;
            CancelTappedCommand = null;
            RepeatDayTappedCommand = null;
            ShowMessage = null;
            ClosePopupPage = null;
            UnhandledException = null;
        }

    }
```

DashBoard Page

```
class DashBoardViewModel : INotifyPropertyChanged, IDisposable
    {
        private ObservableCollection<Alarm> _alarmList;
        private ObservableCollection<Alarm> _alarmListRepeating;
        private ObservableCollection<Alarm> _alarmListNonRepeating;
        private SettingsDatabase _settingsDatabase;

        public delegate void UnhandledExceptionEventHandler(object sender, EventArgs e);
        public event UnhandledExceptionEventHandler UnhandledException;

        public event PropertyChangedEventHandler PropertyChanged;
        private void OnPropertyChanged(string nameOfProperty)
        {
            if (PropertyChanged != null)
                PropertyChanged(this, new PropertyChangedEventArgs(nameOfProperty));
        }

        public DashBoardViewModel(SettingsDatabase settingsDatabase)
        {
            _settingsDatabase = settingsDatabase;
            _alarmList = new ObservableCollection<Alarm>();
            _alarmListRepeating = new ObservableCollection<Alarm>();
            _alarmListNonRepeating = new ObservableCollection<Alarm>();

            this.GetAlarmsCommand = new Command(OnGetAlarms);
        }

        ~DashBoardViewModel() { this.Dispose(); }

        public ObservableCollection<Alarm> RepeatingAlarmsList
        {
            get
            {
                return _alarmListRepeating;
            }
            set { _alarmListRepeating = value; }
        }

        public ObservableCollection<Alarm> NonRepeatingAlarmsList
        {
            get
            {
                return _alarmListNonRepeating;
            }
            set { _alarmListNonRepeating = value; }
        }

        public ICommand GetAlarmsCommand { protected set; get; }

        private async void OnGetAlarms()
        {
            try
            {
                JsonValue jsonAlarms = await ApiViewModel.GetAsync(ApiClasses.Alarms);

                if (jsonAlarms != null)
                {
                    JArray jArray = JArray.Parse(jsonAlarms["alarms"].ToString());
                    _alarmList.Clear();
                    foreach (JObject o in jArray.Children())
                    {
                        //Parse out the messages and add the top like 25 of them
                        DateTime tempDate = new DateTime();
                        DateTime.TryParse((string)o["dateAndTime"], out tempDate);
                        _alarmList.Add(new Alarm((string)o["keyId"], (string)o["name"], (string)o["message"], o["repeat"].ToObject<bool[]>(),
(bool)o["priority"], tempDate));
                    }
                }
                _alarmListRepeating = new ObservableCollection<Alarm>(_alarmList.Where(a => a.HasRepeatingPrefs()));
                _alarmListNonRepeating = new ObservableCollection<Alarm>(_alarmList.Where(a => !a.HasRepeatingPrefs()));

            }
            catch (Exception ex)
            {
                UnhandledException?.Invoke(ex, null);
            }
            OnPropertyChanged("AlarmsList");
            OnPropertyChanged("RepeatingAlarmsList");
            OnPropertyChanged("NonRepeatingAlarmsList");
        }

        public void Dispose()
        {
            PropertyChanged = null;
            UnhandledException = null;
            _alarmList = null;
            _alarmListRepeating = null;
            _alarmListNonRepeating = null;
            _settingsDatabase = null;
            RepeatingAlarmsList = null;
            NonRepeatingAlarmsList = null;
            GetAlarmsCommand = null;
        }

    }

    public class AlarmPageEventArgs : EventArgs
    {
        public AlarmPageEventArgs()
        {
            this.Alarm = new Alarm();
        }
        public AlarmPageEventArgs(Alarm alarm)
        {
            this.Alarm = alarm;
        }
        public Alarm Alarm { get; set; }
}
```

Communication Page

```
class CommunicationPageViewModel : INotifyPropertyChanged, IDisposable
    {
        private ObservableCollection<Message> _messageList;
        private string _currentMessage;
        private SettingsDatabase _settingsDatabase;

       public CommunicationPageViewModel(SettingsDatabase settingsDatabase)
        {
            _settingsDatabase = settingsDatabase;
            MainText = "Type your message: ";
            _currentMessage = "";
            _messageList = new ObservableCollection<Message>();
            this.SendMessageCommand = new Command(SendMessage);
            this.GetMessagesCommand = new Command(GetMessages);
        }

        ~CommunicationPageViewModel() { this.Dispose(); }

        public delegate void SendButtonPressedEventHandler(object sender, EventArgs e);
        public event SendButtonPressedEventHandler SendButtonPressed;

        public delegate void UnhandledExceptionEventHandler(object sender, EventArgs e);
        public event UnhandledExceptionEventHandler UnhandledException;

        public event PropertyChangedEventHandler PropertyChanged;
        protected virtual void OnPropertyChanged(string nameOfProperty)
        {
            if (PropertyChanged != null)
                PropertyChanged(this, new PropertyChangedEventArgs(nameOfProperty));
        }

        public string MainText { get; set; }

        /// <summary>
        /// This list will hold a previously sent messages to display to the caretaker
        /// </summary>
        public ObservableCollection<Message> MessageList
        {
            get
            {
                return _messageList;
            }
            set
            {
                _messageList = value;
                OnPropertyChanged("MessageList");
            }
        }

        /// <summary>
        /// The Message that will be added to the List of messages and sent to the forgetMeNot display.
        /// </summary>
        public string CurrentMessage
        {
            get
            {
                return _currentMessage;
            }
            set
            {
                _currentMessage = value;
                OnPropertyChanged("CurrentMessage");
            }
        }

        /// <summary>
        /// Command used to send the Message String to the forgetMeNot display.
        /// </summary>
        public ICommand SendMessageCommand { protected set; get; }

        /// <summary>
        /// Sends the message to the API
        /// </summary>
        private async void SendMessage()
        {
            if (!string.IsNullOrEmpty(_currentMessage))
            {

                Message myMessage = new Message(_currentMessage, DateTime.Now);

                try
                {
                    JsonValue json = await ApiViewModel.PostAsync(myMessage, ApiClasses.Messages);
                }
                catch (Exception ex)
                {
                    UnhandledException?.Invoke(ex, null);
                }

                _currentMessage = "";
                OnPropertyChanged("CurrentMessage");
                GetMessages();
            }
        }

        public ICommand GetMessagesCommand { protected set; get; }

        private async void GetMessages()
        {
            try
            {
                JsonValue jsonMessages = await ApiViewModel.GetAsync(ApiClasses.Messages);

                if (jsonMessages != null)
                {
                    JArray jArray = JArray.Parse(jsonMessages["messages"].ToString());
                    _messageList.Clear();
                    foreach (JObject o in jArray.Children())
                    {
                        //Parse out the messages and add the top like 25 of them
                        _messageList.Add(new Message((string)o["body"], DateTime.Parse(((string)o["dateAndTime"]))));
                    }
                }
                SendButtonPressed?.Invoke(this, null);
            }
            catch (Exception ex)
            {
                UnhandledException?.Invoke(ex, null);
            }
            OnPropertyChanged("MessageList");
        }

        public void Dispose()
        {
            PropertyChanged = null;
            _messageList = null;
            _currentMessage = null;
            _settingsDatabase = null;
            MainText = null;
            MessageList = null;
            CurrentMessage = null;
            UnhandledException = null;
            SendMessageCommand = null;
            GetMessagesCommand = null;
        }
```

Settings Page

```
public class SettingsPageViewModel : INotifyPropertyChanged
    {
        private SettingsDatabase _settingsDatabase;
        private Settings _settings;

        public SettingsPageViewModel(SettingsDatabase settingsDatabase)
        {
            _settingsDatabase = settingsDatabase;
            _settings = _settingsDatabase.GetSettings();

            this.UpdateSettingsCommand = new Command(OnUpdateSettings);
        }

        public event PropertyChangedEventHandler PropertyChanged;
        protected virtual void OnPropertyChanged(string nameOfProperty)
        {
            if (PropertyChanged != null)
                PropertyChanged(this, new PropertyChangedEventArgs(nameOfProperty));
        }

        public string PatientPhoneNumber
        {
            get
            {
                return _settings.PatientPhoneNumber;
            }
            set
            {
                _settings.PatientPhoneNumber = value;
                OnPropertyChanged("PatientPhoneNumber");
            }
        }

        public string PatientName
        {
            get
            {
                return _settings.PatientName;
            }
            set
            {
                _settings.PatientName = value;
                OnPropertyChanged("PatientName");
            }
        }

       public ICommand UpdateSettingsCommand { protected set; get; }

        private void OnUpdateSettings()
        {
            _settingsDatabase.UpdateSettings(_settings);
        }
```

API View Model

```
class ApiViewModel
    {

        private const string _forgetMeNotURL = "https://forgetmenotapi-145619.appspot.com/api/v1/";
```

```
// Gets weather data from the passed URL.
public static async Task<JsonValue> FetchWeatherAsync(string url)
{
    // Create an HTTP web request using the URL:
    HttpWebRequest request = (HttpWebRequest)HttpWebRequest.Create(new Uri(url));
    request.ContentType = "application/json";
    request.Method = "GET";

    // Send the request to the server and wait for the response:
    using (WebResponse response = await request.GetResponseAsync())
    {
        // Get a stream representation of the HTTP web response:
        using (Stream stream = response.GetResponseStream())
        {
            // Use this stream to build a JSON document object:
            JsonValue jsonDoc = await Task.Run(() => JsonObject.Load(stream));

            // Return the JSON document:
            return jsonDoc;
        }
    }
}

/// <summary>
/// returns a json string from the forget me not API
/// </summary>
/// <param name="apiClass"></param>
/// <returns></returns>
public static async Task<JsonValue> GetAsync(ApiClasses apiClass)
{
    try
    {
        using (HttpClient httpClient = new HttpClient())
        {
            string url = _forgetMeNotURL;
            switch (apiClass)
            {
                case ApiClasses.Alarms:
                    url += "alarms/";
                    break;
                case ApiClasses.Messages:
                    url += "messages/";
                    break;
                case ApiClasses.Settings:
                    url += "settings/";
                    break;
                default:
                    throw new NotImplementedException();
                    break;
            }

            var response = await httpClient.GetAsync(url);

            if (response.StatusCode != HttpStatusCode.NoContent)
            {
                response.EnsureSuccessStatusCode();

                string content = await response.Content.ReadAsStringAsync();
                JsonValue jsonDoc = await Task.Run(() => JsonObject.Parse(content));
                return jsonDoc;
            }
        }
        return null;
    }
    catch (Exception ex)
    {

        throw ex;
    }
```

```
        }

        public static async Task<JsonValue> PostAsync(object data, ApiClasses apiClass)
        {
            try
            {

                using (HttpClient httpClient = new HttpClient())
                {
                    string url = _forgetMeNotURL;
                    switch (apiClass)
                    {
                        case ApiClasses.Alarms:
                            url += "alarms/";
                            break;
                        case ApiClasses.Messages:
                            url += "messages/";
                            break;
                        case ApiClasses.Settings:
                            url += "settings/";
                            break;
                        default:
                            throw new NotImplementedException();
                            break;
                    }

                    string jsonObject = JsonConvert.SerializeObject(data, Formatting.None, new IsoDateTimeConverter() { DateTimeFormat =
"MM/dd/yyyy hh:mm:ss tt" });
                    var response = await httpClient.PostAsync(url, new StringContent(jsonObject));

                    response.EnsureSuccessStatusCode();

                    string content = await response.Content.ReadAsStringAsync();
                    JsonValue jsonDoc = await Task.Run(() => JsonObject.Parse(content));
                    return jsonDoc;

                }
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        public static async Task<JsonValue> PutAsync(object data, ApiClasses apiClass)
        {
            try
            {
                using (HttpClient httpClient = new HttpClient())
                {
                    string url = _forgetMeNotURL;
                    switch (apiClass)
                    {
                        case ApiClasses.Alarms:
                            Alarm tempAlarm = (Alarm)data;
                            url += "alarms/" + tempAlarm.keyId;
                            break;
                        case ApiClasses.Messages:
                            url += "messages/";
                            break;
                        case ApiClasses.Settings:
                            url += "settings/";
                            break;
                        default:
                            throw new NotImplementedException();
                            break;
                    }

                    string jsonObject = JsonConvert.SerializeObject(data, Formatting.None, new IsoDateTimeConverter() { DateTimeFormat =
"MM/dd/yyyy hh:mm:ss tt" });
```

```
                    var response = await httpClient.PutAsync(url, new StringContent(jsonObject));

                    response.EnsureSuccessStatusCode();

                    string content = await response.Content.ReadAsStringAsync();
                    JsonValue jsonDoc = await Task.Run(() => JsonObject.Parse(content));
                    return jsonDoc;

                }
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        public static async Task<JsonValue> DeleteAsync(object keyId, ApiClasses apiClass)
        {
            try
            {

                using (HttpClient httpClient = new HttpClient())
                {
                    string url = _forgetMeNotURL;
                    switch (apiClass)
                    {
                        case ApiClasses.Alarms:
                            url += "alarms/";
                            break;
                        case ApiClasses.Messages:
                            url += "messages/";
                            break;
                        case ApiClasses.Settings:
                            url += "settings/";
                            break;
                        default:
                            throw new NotImplementedException();
                            break;
                    }

                    var response = await httpClient.DeleteAsync(url + keyId);

                    response.EnsureSuccessStatusCode();
                    return null;
                }
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }
    }

    public enum ApiClasses { Alarms = 0, Messages, Settings
```

Alarm Model

```
public class Alarm : IDisposable
    {

        private string _keyId;
        private string _name;
        private string _message;
        private bool[] _repeat;
        private DateTime _dateAndTime;
        private bool _priority;

        public Alarm()
```

```
{
    _keyId = "";
    _name = "";
    _message = "";
    _repeat = new bool[7] { false, false, false, false, false, false, false };
    _priority = false;
    _dateAndTime = DateTime.Now;
}

public Alarm(string keyId, string name, string message, bool[] repeat, bool priority, DateTime dateAndTime)
{
    _keyId = keyId;
    _name = name;
    _message = message;
    _repeat = repeat;
    _priority = priority;
    _dateAndTime = dateAndTime;
}

~Alarm()
{
    this.Dispose();
}

public string keyId
{
    get
    {
        return _keyId;
    }
    set
    {
        _keyId = value;
    }
}

public string name
{
    get
    {
        return _name;
    }
    set
    {
        _name = value;
    }
}

public string message
{
    get
    {
        return _message;
    }
    set
    {
        _message = value;
    }
}

public bool[] repeat
{
    get
    {
        return _repeat;
    }
    set
    {
        _repeat = value;
    }
```

```
        }

        public DateTime dateAndTime
        {
            get
            {
                return _dateAndTime;
            }
            set
            {
                _dateAndTime = value;
            }
        }

        public bool priority
        {
            get
            {
                return _priority;
            }
            set
            {
                _priority = value;
            }
        }

        public string RepeatingLabelText
        {
            get
            {
                string repeatingDays = string.Empty;
                for (int i = 0; i < 7; i++)
                    if (_repeat[i])
                    {
                        DaysOfTheWeekABV day = (DaysOfTheWeekABV)i;
                        repeatingDays = repeatingDays + day.ToString() + ", ";
                    }

                return "[ " + repeatingDays.TrimEnd(',', ' ') + " ]    " + dateAndTime.ToString("h:mm tt");
            }
            set { }
        }

        public void Dispose()
        {
            _keyId = null;
            keyId = null;
            _name = null;
            name = null;
            _message = null;
            message = null;
            _repeat = null;
            repeat = null;
        }

        public bool HasRepeatingPrefs()
        {
            foreach (bool b in _repeat)
            {
                //if there are any true bools in the list of bools then the alarm has repeating prefs
                if (b)
                    return true;
            }
            return false;
        }
    }

    public enum DaysOfTheWeek
    {
        Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday
    }
    public enum DaysOfTheWeekABV
    {
        S, M, T, W, Th, F, Sa
```

Settings Model

```
public interface ISQLite
    {
        SQLiteConnection GetConnection();
    }

    public class Settings : IDisposable
    {
        [PrimaryKey, AutoIncrement]
        public int Id { get; set; }
        public string PatientName { get; set; }
        public string PatientPhoneNumber { get; set; }

        public Settings()
        {
        }

        public Settings(bool tempWeatherUnit, string patientName, string patientPhoneNumber)
        {
            PatientName = patientName;
            PatientPhoneNumber = patientPhoneNumber;
        }

        ~Settings() { this.Dispose(); }

        public void Dispose()
        {
            PatientName = null;
            PatientPhoneNumber = null;
        }
    }

    public class SettingsDatabase
    {
        private SQLiteConnection _connection;

        public SettingsDatabase()
        {
            _connection = DependencyService.Get<ISQLite>().GetConnection();
            _connection.CreateTable<Settings>();
        }

        public Settings GetSettings()
        {
            return _connection.Query<Settings>("Select * From [Settings]").FirstOrDefault();
        }

        public void UpdateSettings(Settings settings)
        {
            _connection.Update(settings);
        }

        /// <summary>
        /// Only use for Testing
        /// </summary>
        public void DropAndCreateSettingsTableDatabase()
        {
            _connection.DropTable<Settings>();
            _connection.CreateTable<Settings>();
        }

        public int GetSettingsDatabaseRowCount()
        {
            return _connection.Table<Settings>().Count<Settings>();
        }

        public void DeleteSetting(int id)
        {
            _connection.Delete<Settings>(id);
        }

        public void AddDefaultSetting()
        {
            _connection.Insert(new Settings(false, "", ""));
        }
```

API Source Code

Notifications class (API push-notification service implementation):

```
class Notification(ndb.Model):
    tokens = ndb.StringProperty(repeated=True)
    body = ndb.StringProperty()
    title = ndb.StringProperty()


    @classmethod
    def authenticate(cls, newToken):
        notifications = cls.query()
        notification = notifications.get()
        if len(notification.tokens) >= 10:
            count = 0
            while count <= 5:
                notification.tokens.pop(0)
                count = count + 1
        for token in notification.tokens:
            if token == newToken:
                return False
        notification.tokens.append(str(newToken))
        notification.put()
        return True


    @classmethod
    def push_notification(cls):
        notification = cls.query().get()
        if len(notification.tokens) >= 1:
            push_service = FCMNotification(api_key=APP_KEY)
            data_message = {"title":notification.title, "body":notification.body}
            result = push_service.notify_multiple_devices(registration_ids=notification.tokens, data_message=data_message)
            return result
```

Alarm and Message Entity Classes

```
class Alarm(ndb.Model):
    name = ndb.StringProperty()
    message = ndb.StringProperty()
    dateAndTime = ndb.DateTimeProperty()
    time = ndb.TimeProperty()
    repeat = ndb.BooleanProperty(repeated=True)
    priority = ndb.BooleanProperty()

    @classmethod
    def get_all(cls):
        alarmEntities = cls.query()
        if alarmEntities.count() > 0:
            alarmEntities = alarmEntities.order(cls.time)
            allAlarms = []
            for alarm in alarmEntities:
                alarmDictionary = {"name":str(alarm.name),
                                   "message":str(alarm.message),
                                   "dateAndTime":datetime.strftime(alarm.dateAndTime, '%m/%d/%Y %I:%M:%S %p'),
                                   "repeat":alarm.repeat,
                                   "priority":alarm.priority,
                                   "keyId":str(alarm.key.id())}
                allAlarms.append(alarmDictionary)
            return allAlarms
        else:
            return None


class Message(ndb.Model):
    dateAndTime = ndb.DateTimeProperty()
    body = ndb.StringProperty()


    @classmethod
    def get_recent(cls):
        messageEntities = cls.query()
        numMessages = messageEntities.count()
        if numMessages > 0:
            messageEntities = messageEntities.order(-cls.dateAndTime)
            recentMessages = []
            for message in messageEntities:
                messageDictionary = {"dateAndTime":datetime.strftime(message.dateAndTime, '%m/%d/%Y %I:%M:%S %p'),
                                     "body":str(message.body),
                                     "key":str(message.key.id())}
                recentMessages.append(messageDictionary)
                #only allow 50 most recent messages
                if len(recentMessages) == 2:
                    return list(reversed(recentMessages))
            return list(reversed(recentMessages))
        else:
            return None
```

RESTful API Endpoints

```
from bottle import Bottle, request, run, response, get, put, post, delete
from google.appengine.ext import ndb
from datetime import datetime
from pyfcm import FCMNotification
from requests_toolbelt.adapters import appengine
import os.path
import json

appengine.monkeypatch(validate_certificate=False)
APP_KEY =
"AAAAvzuOGxs:APA91bE4v4JdiH_v0WqHeS50ZIFD1vBXdugTz9wmFAQfWQN2Bo9vfOVIaN1Xx0krIF3uM87dlVUcCT7Sr9EWMLb
wRCVqAfxrN3_LzHNzj3bgom-0Ls3PDVX-90M6eGmp_NEzvg-n1XpD"
app = Bottle()


@app.post("/api/v1/tokens")
@app.post("/api/v1/tokens/")
def post_token():
    receivedToken = request.body.read()
    if Notification.query().count() == 0:
        notification = Notification()
        notification.tokens.append(json.loads(receivedToken)["token"])
        notification.put()
        response.status = 200
        return
    try:
        receivedTokenJson = json.loads(receivedToken)
        if Notification.authenticate(receivedTokenJson["token"]):
            response.status = 200
            return receivedToken
        else:
            response.status = 204
    except (ValueError, KeyError) as e:
        response.status = 400
        return 'Malformed JSON'


#@app.get("/api/v1/warnings")
#@app.get("/api/v1/warnings/")
#def get_warnings():


@app.post("/api/v1/warnings")
@app.post("/api/v1/warnings/")
def post_warnings():
    receivedWarning = request.body.read()
    notification = Notification.query().get()
    if notification:
        receivedWarningJson = json.loads(receivedWarning)
        notification.body = receivedWarningJson["body"]
        notification.title = receivedWarningJson["title"]
        notification.put()
        Notification.push_notification()
        response.status = 200
        receivedWarningJson["token"] = notification.tokens
        return json.dumps(receivedWarningJson)
    else:
        response.status = 204
        return "No device authenticated"

@app.get("/api/v1/alarms")
@app.get("/api/v1/alarms/")
def get_alarms():
    allAlarms = Alarm.get_all()
    if allAlarms == None:
        response.status = 204
        return None
    else:
        #Wraps array in JSON object for convenience with QT functions
        jsonReturn = {"alarms":allAlarms}
        response.status = 200
        return json.dumps(jsonReturn)


@app.post("/api/v1/alarms")
@app.post("/api/v1/alarms/")
def post_alarms():
    receivedAlarm = request.body.read()
    try:
        receivedAlarmJson = json.loads(receivedAlarm)
        newAlarm = Alarm()
        newAlarm.name = receivedAlarmJson['name']
        newAlarm.message = receivedAlarmJson['message']
        newAlarm.dateAndTime = datetime.strptime(receivedAlarmJson['dateAndTime'], '%m/%d/%Y %I:%M:%S %p')
        newAlarm.time = newAlarm.dateAndTime.time()
        newAlarm.repeat = receivedAlarmJson['repeat']
        newAlarm.priority = receivedAlarmJson['priority']
        repeatLength = len(newAlarm.repeat)
        if not (repeatLength == 7):
            raise ValueError('boolen repeat list must be exactly 7 characters')
        newAlarmKey = newAlarm.put()
        response.status = 200;
        receivedAlarmJson['keyId'] = str(newAlarmKey.id())
        return json.dumps(receivedAlarmJson)
    except (ValueError, KeyError) as e:
        response.status = 400
        return 'Malformed JSON'

@app.put("/api/v1/alarms/<alarmId>")
@app.put("/api/v1/alarms/<alarmId>/")
def put_alarm(alarmId):
    alarmIdInt = int(alarmId)
    receivedAlarm = request.body.read()
    if alarmId != "":
        toBeUpdated = Alarm.get_by_id(alarmIdInt)
        if toBeUpdated is None:
            response.status = 204
            return
        else:
            try:
                receivedAlarmJson = json.loads(receivedAlarm)
                toBeUpdated.name = receivedAlarmJson['name']
                toBeUpdated.message = receivedAlarmJson['message']
                toBeUpdated.dateAndTime = datetime.strptime(receivedAlarmJson['dateAndTime'], '%m/%d/%Y %I:%M:%S %p')
                toBeUpdated.time = toBeUpdated.dateAndTime.time()
                toBeUpdated.repeat = receivedAlarmJson['repeat']
                toBeUpdated.priority = receivedAlarmJson['priority']
                repeatLength = len(toBeUpdated.repeat)
                if not (repeatLength == 7):
                    raise ValueError('boolen repeat list must be exactly 7 characters')
                newAlarmKey = toBeUpdated.put()
                response.status = 200;
                receivedAlarmJson['keyId'] = str(newAlarmKey.id())
                return json.dumps(receivedAlarmJson)
            except (ValueError, KeyError) as e:
                response.status = 400
                return 'Malformed JSON'
    else:
        response.status = 400
        return


@app.delete("/api/v1/alarms/<alarmId>")
@app.delete("/api/v1/alarms/<alarmId>/")
def delete_alarm(alarmId):
    alarmIdInt = int(alarmId)
    if alarmId != "":
        toBeDeleted = Alarm.get_by_id(alarmIdInt)
        if toBeDeleted is None:
            response.status = 204
            return
        else:
            toBeDeleted.key.delete()
            response.status = 200
            return
    else:
        response.status = 400
        return

@app.get("/api/v1/messages")
@app.get("/api/v1/messages/")
def get_messages():
    recentMessages = Message.get_recent()
    if recentMessages == None:
        response.status = 204
        return None
    else:
        #Wraps array in JSON object for convenience with QT functions
        jsonReturn = {"messages":recentMessages}
        return json.dumps(jsonReturn)


@app.post("/api/v1/messages")
@app.post("/api/v1/messages/")
def post_messages():
    receivedMessage = request.body.read()

    try:
        receivedMessageJson = json.loads(receivedMessage)
        newMessage = Message()
        newMessage.dateAndTime = datetime.strptime(receivedMessageJson['dateAndTime'], '%m/%d/%Y %I:%M:%S %p')
        newMessage.body = receivedMessageJson['body']
        newMessageKey = newMessage.put()
        response.status = 200;
        receivedMessageJson['keyId'] = str(newMessageKey.id())
        return json.dumps(receivedMessageJson)
    except (ValueError, KeyError) as e:
        response.status = 400
        return 'Malformed JSON'

@app.put("/api/v1/messages/<messageId>")
@app.put("/api/v1/messages/<messageId>")
def put_message(messageId):
    messageIdInt = int(messageId)
    receivedMessage = request.body.read()
    if messageId != "":
        toBeUpdated = Message.get_by_id(messageIdInt)
        if toBeUpdated is None:
            response.status = 204
            return
        else:
            try:
                receivedMessageJson = json.loads(receivedMessage)
                toBeUpdated.dateAndTime = datetime.strptime(receivedMessageJson['dateAndTime'], '%m/%d/%Y %I:%M:%S %p')
                toBeUpdated.body = receivedMessageJson['body']
                newMessageKey = toBeUpdated.put()
                response.status = 200;
                receivedMessageJson['keyId'] = str(newMessageKey.id())
                return json.dumps(receivedMessageJson)
            except (ValueError, KeyError) as e:
                response.status = 400
                return 'Malformed JSON'
    else:
        response.status = 400
        return


@app.delete("/api/v1/messages/<messageId>")
@app.delete("/api/v1/messages/<messageId>/")
def delete_message(messageId):
    messageIdInt = int(messageId)
    if messageId != "":
        toBeDeleted = Message.get_by_id(messageIdInt)
        if toBeDeleted is None:
            response.status = 204
            return
        else:
            toBeDeleted.key.delete()
            response.status = 200
            return
    else:
        response.status = 400
        return


if __name__ == "__main__":
    app.run(server='gae', debug=True)
```

Bracelet Code

```
int MOTORPIN = 12;
int ALARMLED = 8; //mirrors motor, shines when motor on
int BUTTONPIN = 9;
int VCCOUT = 7;
char BTDATA;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  //Serial.begin(9600);
  //Serial.println("Press 1 to trigger a passive alarm or 2 to trigger a priority alarm. Any alarm can be cancelled with 0.");
  pinMode(MOTORPIN,OUTPUT);
  pinMode(ALARMLED, OUTPUT);
  pinMode(BUTTONPIN, INPUT);
  pinMode(VCCOUT, OUTPUT);
  digitalWrite(VCCOUT, 1);
}

void loop() {
    String success = check_alarm();
    if (success == "passive_complete") {
      Serial.println("Passive alarm ran its course");
    }
    else if(success == "priority_complete") {
      Serial.println("Priority Alarm unhandled!");
    }
    else if(success == "canceled") {
      Serial.println("Alarm canceled early.");
    }

    check_panic_button();
}

void check_panic_button() {
    int holdtime = 0;
    int sensorValue = digitalRead(BUTTONPIN);
    while(sensorValue) {
        sensorValue = digitalRead(BUTTONPIN);

        delay(1000);
        holdtime++;
        if(holdtime >= 5) {
          send_panic_signal();
          return;
        }
    }
    if(holdtime >= 5) {
        send_panic_signal();
        return;
    }
}

void send_panic_signal() {
  Serial.println("panic signal, yo");
}

String check_alarm() {
    if (Serial.available()){
        BTDATA = Serial.read();
        if(BTDATA == '1') {
            return passive_alarm(5, 3, 22.5);
        }
        else if(BTDATA == '2') {
            return priority_alarm(2, 3, 22.5);
        }
  }
  return "none";
}

String passive_alarm(int numBuzzes, int numSnooze, double snoozeTime) {
  for(int j = 0; j < numSnooze; j++) {
    for(int i = 0; i < numBuzzes; i++) {

      digitalWrite(MOTORPIN, 1);
      digitalWrite(ALARMLED, 1);
      if(wait(0.5) == 0) {
        digitalWrite(MOTORPIN, 0);
        digitalWrite(ALARMLED, 0);
        return "canceled";
      }

      digitalWrite(MOTORPIN, 0);
      digitalWrite(ALARMLED, 0);
      if(wait(0.5) == 0) {
        return "canceled";
      }
    }
    if( wait(snoozeTime) == 0) {
      return "canceled";
    }
  }
  return "passive_complete";
}


String priority_alarm(int numBuzzes, int numSnooze, double snoozeTime) {
  for(int j = 0; j < numSnooze; j++) {
    for(int i = 0; i < numBuzzes; i++) {

      digitalWrite(MOTORPIN, 1);
      digitalWrite(ALARMLED, 1);
      if(wait(3.0) == 0) {
        digitalWrite(MOTORPIN, 0);
        digitalWrite(ALARMLED, 0);
        return "canceled";
      }

      digitalWrite(MOTORPIN, 0);
      digitalWrite(ALARMLED, 0);
      if(wait(2.0) == 0) {
        return "canceled";
      }
    }
    if( wait(snoozeTime) == 0) {
      return "canceled";
    }
  }
  return "priority_complete";
}

/*
 * Waits for some period checking for incoming BT data every half second
 * so we can stop an alarm early.
 *
 * returns 0 if BT recieves a zero during the wait period (wait cancelled)
 * returns 1 if wait runs its course successfully
 */
int wait(double seconds) {
  while(seconds > 0) {
   delay(500);
   check_panic_button();
   if(Serial.available()) {
    BTDATA = Serial.read();
    if(BTDATA == '0') {
     return 0;
```

```
        }
      }
      seconds -= .5;
    }
    return 1;
}
```