Creating an Accessible Network of Life-Saving Medical Devices

A Thesis Presented to the Departments of

Computer Science

and

Electrical Engineering

at the

John A. Paulson

School of Engineering and Applied Sciences

in Partial Fulfillment for the

Joint Degree

of Bachelor of Arts

by

Gregory M. Hewett

on

March 31, 2017

Harvard University

Cambridge, Massachusetts

Abstract

This paper consists of the design and implementation of a software platform empowering users to quickly find autoinjectors and defibrillators in emergencies by leveraging both crowdsourcing and the Internet of Things (IoT). Both autoinjectors and defibrillators are needed within minutes of an emergency to counteract anaphylactic shock and cardiac arrest respectively. Besides receiving one from an ambulance or hospital, no platform currently exists for finding or receiving an autoinjector during emergencies, and automated emergency defibrillators (AEDs) can be difficult to locate quickly when they are needed. The goal of this project is to create a platform for users to quickly and efficiently find both autoinjectors and defibrillators using a mobile app. A mobile app is used for crowdsourcing autoinjectors and finding Internet-connected smart containers that house both autoinjectors and defibrillators.

Acknowledgements

I would like to thank my computer science faculty advisor, Professor Margo Seltzer, for being an incredible mentor on this project. Thank you for sharing your passion for building and empowering me to make something that could have a real impact on people.

I would also like to thank my electrical engineering faculty advisor, Professor Gu-Yeon Wei, for his fantastic mentorship and support. Thank you for believing in me, in this idea, and for encouraging me to see it all the way through.

I would like to thank the Harvard Active Learning Labs staff for their patience, availability, and willingness to share their expertise. This project would not have been possible without you.

I want to thank all of my friends who provided me with emotional support, laughter, and honest feedback. Thank you for helping me throughout the year.

Finally, I especially want to thank my wonderful fiancée and family for their continued support in all of my academic and personal pursuits. I can't imagine doing this without you, and without your love and guidance, I would not be where I am today. Thank you.

Contents

1 Introduction **8**

1.1 Motivation . . . 8

1.2 Scope . . . 9

2 Background **10**

2.1 Prior Art . . . 10

2.2 Population Considerations . . . 10

2.2.1 Autoinjectors Nearby . . . 11

2.2.2 Population Densities . . . 15

3 Data Privacy **18**

3.1 Location . . . 18

3.1.1 Location Privacy . . . 19

3.1.2 Data Privacy Contract . . . 20

3.1.3 Mercator Projection Cloaking . . . 20

4 Design Overview **23**

4.1 Mobile App . . . 23

4.2 Smart Containers . . . 24

4.3 Cloud . . . 24

4.4 Alternative User Experience . . . 25

5 Hardware Design **26**

5.1 Goals . . . 26

5.2 Form Factor . . . 26

5.3 Electronics . . . 28

5.3.1 Microcontroller . . . 28

5.3.2 Touch Screen . . . 29

5.3.3 Environmental Sensor . . . 29

5.3.4 Lights, Hall Effect, and Speaker . . . 30

5.3.5 Power . . . 31

5.3.6 Schematic . . . 32

5.4 Network Protocols . . . 33

5.4.1 HTTP . . . 33

5.4.2 AMQP . . . 33
5.4.3 MQTT . . . 33
5.5 Prototype 1 . . . 34
5.6 Prototype 2 . . . 34
5.7 Evaluation . . . 36

6 Software Design **37**
6.1 Goals . . . 37
6.2 Architecture and Implementation . . . 37
6.2.1 Device Hub . . . 38
6.2.2 Message Processor . . . 39
6.2.3 Message Queue . . . 39
6.2.4 Blob Storage . . . 39
6.2.5 NoSQL Storage . . . 40
6.2.6 Web App . . . 40
6.3 Hardware Dashboard . . . 41
6.4 REST API and Mobile App . . . 41
6.5 Evaluation . . . 43

7 Conclusions and Future Work **45**

A Container Administration Screenshots **46**

B Mobile App Screenshots **47**

C Bill of Materials **48**

D Engineering Drawings **49**

E Specification Sheets and Values **50**

F Repository **55**

References **58**

List of Figures

2.1 $P(N \geq k)$ for New York City and Houston. Each line represents a different value for $P(C)$, the probability that a person with a prescription is carrying an autoinjector, ranging from 10% to 90%. 14

2.2 Comparing $P(N \geq k)$ for different population densities. As the population density increases, the chances of having more autoinjectors nearby increases. The graph on the left assumes $P(C) = 0.3$ while the right assumes $P(C) = 0.5$. The unit of population density, ρ, is people/km^2. 15

2.3 Population of autoinjector carriers who live in areas of differing population densities. The y-axis shows the number of autoinjector carriers who live in an area of at least the density of the value on the x-axis. 16

5.1 An AED container currently in use in many communities [17]. 26

5.2 A 3D CAD model of the prototype that was manufactured and used for testing. 27

5.3 LM555 audio circuitry. Pin 4 of the left-most LM555 is connected to the microcontroller. 31

5.4 Complete hardware circuitry schematic. 32

5.5 (a) Circuit components and (b) front view of the first container prototype. . 34

5.6 (a) Closed front view and (b) perspective view of the first container prototype. 35

5.7 (a) Circuit components and (b) front view of the first container prototype. . 35

5.8 (a) Closed front view and (b) perspective view of the first container prototype. 36

6.1 Overview of software architecture. 38

6.2 Container administrator dashboard for viewing container telemetry and alarm data. 41

6.3 (a) Home screen and (b) after starting an emergency request, the user is prompted to first call 911. 42

6.4 (a) Finding nearby users and containers and (b) responding to a request from Jennifer. 43

A.1 Container administrator view of all container in the system. 46

A.2 Container administrator ability to add a real or simulated container. 47

B.1 (a) Pop-out menu and (b) user login. 48

D.1 CAD Drawing for prototype container. 50

E.1 Adafruit Arduino Feather M0 specification sheet with Atmel ATSAMD21318. 51

E.2 Adafruit 2.8” Color TFT Touchscreen specification sheet. 52

E.3 Adafruit Arduino Feather M0 specification sheet. 53
E.4 iPixel APA102 addressable LED strip specification sheet. 53
E.5 US5881LUA hall effect sensor. 54

List of Tables

2.1 Population of autoinjector carriers living in areas where the probability of having at least k autoinjectors nearby $P(N \geq k)$, is at least 1/2 for different values of k and $P(C)$. 16
C.1 Bill of materials for project implementation at scale. 49
E.1 Resistor and capacitor values used in circuitry. 51

1 Introduction

1.1 Motivation

Medical devices have shaped the practice and perception of medical care since the invention of the stethoscope and thermometer and into the age of the CAT scan and prosthetic limbs. The world's premier hospitals are now home to highly sophisticated technology capable of efficiently saving lives that could not have been saved in the past [26]. However, despite these advances, many life-saving medical devices often remain inaccessible in times of need. For instance, autoinjectors used to administer epinephrine during anaphylactic shock are needed immediately to counteract an allergic reaction for those who suffer from food and/or insect allergies [18]; yet, if someone in need does not have an autoinjector on their person, they waste crucial time waiting for an ambulance or driving to a hospital. Emergency defibrillators recognize heart arrhythmia and can save a person's life during cardiac arrest before an ambulance has time to arrive on the scene [14]. Many buildings contain public containers with access to defibrillators in the case of an emergency, but this requires someone near the emergency to find the container without knowing whether one is even nearby. In the cases of anaphylactic shock and cardiac arrest, the medical technology is available for saving a life, but the device may simply be physically inaccessible during the time of need.

Contrasted with the inaccessibility of medical devices such as these, crowdsourcing and the Internet have made many other goods and services dramatically more accessible. Ride-sharing mobile apps such as Uber and Lyft have dramatically increased the accessibility of transportation in recent years. These apps allow a user to find or request a ride nearby within seconds instead of looking for a taxi or walking to a bus stop. Websites such as Kickstarter and GoFundMe have removed the door-to-door hassle of fundraising for a startup or personal venture and brought investors and donors directly to users—faster than any door-to-door campaign ever could. A simple PA system allows an employee at an airline terminal to quickly find a volunteer to check a bag if the overhead bin space is sure to be full by asking everyone all at once instead of individually. Innovations such as these can provide quick and easy access to nearby services and information without the need to physically search for them.

The contribution of this thesis is leveraging crowdsourcing and the Internet to design and implement a platform for quickly finding autoinjectors and defibrillators in emergencies. I

show that, by using the web, mobile apps, and Internet-connected smart containers, it is possible to create a network that significantly improves the accessibility of these devices. Specifically, a mobile app is used for both crowdsourcing autoinjectors and locating nearby smart containers that house autoinjectors and defibrillators.

1.2 Scope

In Section 2, I provide background on what others have already done to address this inaccessibility problem, and Section 2.2 gives a detailed statistical analysis revealing why a crowdsourcing network is possible for finding and obtaining an autoinjector quickly via a mobile app.

Data privacy considerations are important to consider for a service providing access to medical devices, and I discuss these in section 3. Section 4 provides a high-level overview of the system design and describes the user experience, while sections 5 and 6 reveal the detailed hardware and software design goals and implementations. In these sections, I discuss what the service needs to allow users to quickly find autoinjectors and defibrillators in emergencies and, specifically, what features the app and smart containers must have to be functional and secure.

In Sections 5.7 and 6.5, I evaluate how well the hardware and software implementations met the design specifications and addressed the overall issue of autoinjector and defibrillator inaccessibility. I found that my approach was successful and could be deployed to production after addressing additional legal and manufacturing matters. Section 7 provides conclusions and plans for future work.

The service described in this paper provides users with a mobile app for finding, sharing, or receiving medical devices. As such, the deployment of this service would be subject to regulatory and compliance scrutiny. Legal considerations are beyond the scope of this paper and not needed for internal testing and development, but these considerations would be necessary and prudent for actual product deployment.

2 Background

2.1 Prior Art

Finding and receiving emergency medical attention during emergencies with calls to 911 is common in most communities. Receiving emergency medical attention via a mobile app, however, is not. Only one app, PulsePoint, was found that currently provides a similar service.

PulsePoint [21] is a suite of mobile apps released in October of 2016 that can be used by 911 operators to direct CPR certified users to heart attacks incidents and defibrillators during emergencies. When someone reports a heart attack or heart failure to a 911-center integrated with PulsePoint's technology, the emergency operator notifies all qualified users nearby of the location of the person in need. A registered responder can then navigate to the emergency and perform CPR or administer a defibrillator. To date, the app has helped save numerous lives and has been adopted in various locations across the country. A version of PulsePoint's model of incorporating emergency call centers into the network is discussed in Section 4.4. Other apps and websites such as AED Locations [15] and HeartSafe [12] allow users to manually search and find AEDs in their communities, but do not integrate with 911 services as PulsePoint does.

No product or service was found that provides a mobile app for finding or receiving autoinjectors in emergencies.

2.2 Population Considerations

In this application, autoinjectors are the only devices crowdsourced. They are designed to be carried at all times by those with prescriptions, unlike defibrillators which are bulky and typically stored in cabinets. A crowdsourcing network is useful only if certain population and active-user requirements hold. Specifically, for the network to be used within a particular geographic area, the population density of autoinjector carriers must be high enough to make sharing autoinjectors logistically possible within a reasonable amount of time. Based on this necessity, the following probabilistic measure is considered:

In the event that Person X is looking for an autoinjector in Area A, what is the prob-

ability that there exists some Autoinjector Carrier Y such that the distance between X and Y makes it possible for Y to walk to X faster than an ambulance could travel to X?

This asks whether this network is useful based on the population densities of people who carry autoinjectors. This probability will be used to show that the network effect of sharing autoinjectors is, in fact, possible. I address this question in the following two sections.

2.2.1 Autoinjectors Nearby

Let R be the event that Person X is having an allergic reaction and needs an autoinjector in area A. Let the random variable N be the number of autoinjector carriers nearby Person X in Area A, where nearby is defined to be walking distance within 8 minutes and will be further derived and defined below. To determine the usefulness of a peer-to-peer network of autoinjectors, I find the probability that there are at least k autoinjector carriers nearby Person X, given that Person X has a reaction in Area A. Thus, I am looking for $P(N \geq k|R)$. For purposes of simplicity, I assume that Person X having a reaction does not increase the probability that someone with an autoinjector is nearby, and thus assume N is independent of R. Thus, I determine the probability that there are at least k people carrying autoinjectors in area A at any given time, or

$$P(N \geq k)$$

By taking the complement,

$$\begin{aligned} P(N \geq k) &= 1 - P(N < k) \\ &= 1 - \sum_{j=0}^{k-1} P(N = j). \end{aligned}$$

Therefore, in order to evaluate $P(N \geq k)$, the probability mass function (PMF) of N, $P(N = k)$, needs to be determined. According to the U.S. Census Bureau, population varies drastically during normal working hours, so I account for this in the model by using different population statistics between 8:00 AM and 5:00 PM and 5:00 PM and 8:00 AM [8]. Additionally, Person X might be more likely to be eating and, thus, more likely to have a reaction during working hours. However, eating behaviors are difficult to model and vary

among different populations; therefore, I ignore eating behaviors in this model and assume independence. Let W be the event that the current time is between 8:00 AM and 5:00 PM. Then $P(W) = 9/24$ and $P(W^c) = 15/24$. By the Law of Total Probability [5],

$$P(N = k) = P(N = k|W)P(W) + P(N = k|W^c)P(W^c).$$

Next, I need to clarify a few definitions. First, I define an autoinjector carrier near Person X to be a person with an autoinjector who could, on average, walk to Person X faster than an ambulance could drive to Person X. While there is no federally-mandated ambulance response time standards, many cities and municipalities choose to set an 8-minute response time as the gold standard [20]. The National Fire Protection Agency also recommends that Advanced Life Support teams should aim to respond to all calls within 8 minutes [3]. In actuality, average response times vary across regions, with some cities averaging below 8 minutes and some above [10][23]. Therefore, for the purposes of analysis, I assume that the average response time of an ambulance to Person X is 8 minutes. Now, the effective nearby Area A around Person X is anywhere within an 8-minute walk from Person X. The average walking speed of adults is 5 km/h [7]. I thus calculate A as

$$\begin{aligned} A &= \pi r^2 \\ &= \pi \left((5 \text{ km/h}) \, (8 \text{ min}) \, (1/60 \text{ h/min}) \right)^2 = 1.40 \text{ km}^2. \end{aligned}$$

Now, I determine the probability that a person within Area A has an autoinjector. According to Food Allergy Research and Education, 12% of people within the U.S. have food allergies, with children and adults accounting for 8% and 4% respectively [22]; however, the subject of interest is people who are carrying autoinjectors, not necessarily those who have allergies. According to John Lee, M.D., Director of the Food Allergy Program at Boston Children's Hospital, more than 4.5 million autoinjector twin packs were prescribed in 2015 for children and adults combined [16]. Because food allergies are on the rise [22], I assume that the number of autoinjectors prescribed next year will at least remain the same despite recent rate increases [24]. Let m be the number of unexpired autoinjector twin packs prescribed to consumers. Most autoinjectors expire 1.5 years after the date of manufacture, so I assume that

$$m = 1.5(4.5 \times 10^6) = 6.75 \times 10^6.$$

The population of the U.S. is $u = 321{,}418{,}820$ [8]. Let T be the event that a given person in the U.S. has a prescription for an autoinjector twin pack.

$$\begin{aligned} P(T) &= m/u \\ &= \frac{6.75 \times 10^6}{321418820} \\ &= 0.021 \end{aligned}$$

While this is a simplistic model, it serves as a decent approximation.[1] In reality, infants should not be included in the total population. I could try and treat population of children and adults differently who might have different probabilities of having prescriptions, but parents of those children might be just as likely to carry an autoinjector as their children. Therefore, I assume this probability for purposes of analysis. I do account for the fact that a person with a prescription may not have the autoinjector on their person at all times. Let $P(C)$ be the probability that any given owner or user of an autoinjector is actually carrying it on their person. Then, assuming independence, the probability p that any given person has an autoinjector on their person is

$$p = P(T)P(C)$$

Let ρ be the population density of Area A around Person X. The number of people, n, in Area A is modeled as

$$n = \lfloor \rho A \rfloor .$$

Each person in Area A has an independent probability p of having an autoinjector on their person. This is the story of the binomial distribution [5], as each person's having an autoinjector can be modeled as an independent Bernoulli trial with probability of success p. Thus, the number of pens during working and non-working hours both have the binomial distribution, so

$$N|W, N|W^c \sim \text{Bin}(n, p).$$

[1] More granular prescription data for autoninjectors prescribed was not found. Ideally, the analysis should use the number of users in a given city and number of prescriptions in that city. Instead, I perform the analysis with the naive definition.

For any random variable $X \sim \text{Bin}(n, p)$, the PMF of X is

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}.$$

Therefore, $P(N = k)$ is

$$\begin{aligned} P(N = k) &= P(N = k|W)P(W) + P(N = k|W^c)P(W^c) \\ &= \left(\binom{n_W}{k} p^k (1-p)^{n_W - k} \right) (9/24) + \left(\binom{n_{W^c}}{k} p^k (1-p)^{n_{W^c} - k} \right) (15/24), \end{aligned}$$

where n_W and n_{W^c} are the populations of Area A during working hours and non-working hours respectively. Now, I can use this to determine $P(N \geq k)$—the probability that there are at least k autoinjectors near Person X in Area A—if the population density of Area A is known, since $n = \lfloor \rho A \rfloor$. Using U.S. Census Bureau data for population densities [8], Figure 2.1 shows $P(N \geq k)$ for New York City and Houston, and each line represents a different value for $P(C)$. The population density of New York City is 10908 people/km^2 while Houston is 1480 people/km^2, and as expected the number of autoinjetors likely to be in any given area is much higher in New York City.

Figure 2.1: $P(N \geq k)$ for New York City and Houston. Each line represents a different value for $P(C)$, the probability that a person with a prescription is carrying an autoinjector, ranging from 10% to 90%.

As can be seen, a higher population density corresponds to an increase in the expected number of autoinjectors nearby. As shown in Figure 2.1, even if 50% of people with autoinjector prescriptions carry it on their person, there is only a 50% chance there will be at least 25

autoinjectors nearby in Houston compared to New York's 160. New York and Houston are the first and fourth largest populated cities in the U.S. respectively [8], yet there is a significant difference in likelihood due to their differing population densities. I next consider this effect more formally and determine the number of people living in different population densities in the U.S. and the resulting effect on $P(N \geq k)$.

2.2.2 Population Densities

As shown in Section 2.2.1 through the cities of New York and Houston, a higher population density corresponds to an increase in the expected number of autoinjectors nearby. Figure 2.2 shows this effect of population density on $P(N \geq k)$ for two different values of $P(C)$. As expected, the probability of having an autoinjector nearby is proportional to population density.

Figure 2.2: Comparing $P(N \geq k)$ for different population densities. As the population density increases, the chances of having more autoinjectors nearby increases. The graph on the left assumes $P(C) = 0.3$ while the right assumes $P(C) = 0.5$. The unit of population density, ρ, is people/km^2.

An important consideration that arrises from this analysis is determining how many people in the U.S. live in an area with a population density that allows a reasonable probability of having an autoinjector nearby. If the number is too low, then an implementation of this network might not be practical. Figure 2.3 shows $uP(T)P(C)$—the number of people in the U.S. population who have prescriptions for autoinjectors and carry them on their person—living in different population densities. The y-axis shows the number who live in an area of

at least the density of the value on the x-axis.

Figure 2.3: Population of autoinjector carriers who live in areas of differing population densities. The y-axis shows the number of autoinjector carriers who live in an area of at least the density of the value on the x-axis.

The relationship between Figures 2.2 and 2.3 combine to show how many autoinjector prescribers live in areas where $P(N \geq k) > 0.5$ for different values of k. These results are shown in Table 2.1. Recall that $P(C) = 1.0$ means that everyone with an autoinjector is carrying it on their person, and thus $uP(T)P(C)$ represents the total number of people with autoinjector prescriptions. Thus, for any value of k in Table 2.1, $uP(T) : P(N \geq k) > 0.5$ indicates of the total number of people *who might need* an autoinjector living in areas where $P(N \geq k) > 0.5$. As can be seen, over 6 million people who might need an autoinjector live an area where it is more likely than not that there are at least 10 autoinjectors within walking distance, even if only 30% of prescribers are carrying their autoinjector.

	$uP(T) : P(N \geq k) > 0.5$	
k	$P(C) = 0.3$	$P(C) = 0.5$
10	6.06×10^6	6.31×10^6
20	4.89×10^6	5.09×10^6
50	3.11×10^6	3.34×10^6
90	9.89×10^5	1.50×10^6

Table 2.1: Population of autoinjector carriers living in areas where the probability of having at least k autoinjectors nearby $P(N \geq k)$, is at least 1/2 for different values of k and $P(C)$.

These results indicate that millions of people who might need an autoinjector in an emergency live in areas where a crowdsourcing app could, on average, supply one faster than an

ambulance. Based on these results, I conclude that creating such a crowdsourcing network is indeed possible.

3 Data Privacy

3.1 Location

Leveraging the crowdsourcing network in emergencies does not necessarily require the system to process or store user locations. For instance, if Person X in Area A is looking for an autoinjector and sends a request to the service without any location information, the system can simply send contact info for Person X to all other users, or contact info for all other users to Person X, and allow the location sharing to happen via non-system communication methods. However, even with a very small number of users, this is highly impractical and provides very little benefit.

On the other hand, if the system collects user locations, then the service can become more streamlined and useful. Suppose the service does not persistently store any user location information. If Person X in Area A sends a request to find an autoinjector with their current location information, then all other mobile devices on the network can be notified of that location and alert the user if they are within a certain proximity of that location. Then, user can opt to respond if they are able. This method is much better than sharing no location information with the service at all; yet, if there are n users on the network, $n - 1$ users will always be notified for every emergency, regardless of their location or proximity. This protocol creates many useless notifications and still requires $n-1$ client apps to check if they are nearby the emergency.

Persistent system knowledge of all users' locations, however, can improve the service drastically: when Person X requests an autoinjector, the system can send notifications only to users within a reasonable proximity to Person X. Users who receive a notification will immediately know they are within a reasonable distance of someone in need of an autoinjector and can opt to respond. This protocol will likely be taken seriously by users and does not require any unreasonable communication of users' locations. However, storing users' locations comes at a price: privacy. Users might be reluctant or unwilling to have their locations stored in or processed by the system. If the system did store user locations and became compromised, location information would be exposed. In the next section, I discuss different methods to store user locations while providing some degree of anonymity in the event of a security breach and still retaining the usefulness of the location data.

3.1.1 Location Privacy

In the context of protecting privacy for the public or private release of any data set for purposes of research, Sweeney [25] presents a model known as k-anonymity, where any data set released "provides k-anonymity protection if the information for each person contained in the release cannot be distinguished from at least $k-1$ other individuals in the data set" [25]. This model, while considered weak in some aspects, can be adopted to the storage and processing of user information in network databases. If a database is k-anonymized and later breached, then the users whose information was compromised still retain some form of privacy: namely, anonymity amongst k other users.

Gedik and Liu [11] describe a k-anonymity model for protecting user privacy in mobile systems that allows for each user to indicate the level of k-anonymity desired for each message sent to a particular service. The k-anonymity is achieved by either decreasing location accuracy until $k-1$ other users share the same spatial location or delaying message processing until $k-1$ other messages have been received from the same spatial region.

Cheng et al. [9] present a different scheme that uses a more probabilistic approach. They argue that k-anonymity may not be used if there are fewer than k users in the system, and that even if there are more than k users, "they may span a large area over an extended time period, in which case the cloaked location can be large and cause a severe degradation of service quality." They instead provide a framework that stores cloaked geometric areas for each user, where the users' true location is somewhere in the area stored. The framework lets each user decide on the size and boundaries of the cloaked location stored, and the larger the area, the more anonymity. k-anonymity may be achieved as a side effect in this framework if cloaked locations overlap, but it does not dictate how locations are anonymized. Location-based search queries find cloaked areas that overlap with a given query range, and queries have a runtime in the worst case of $O(e^2 \log e)$ and best case of $O(m+e)$, where e is the number of sides of the geometric shapes stored and m is the number of sides of any polygon used for the query computation.

These frameworks (and others) have their advantages, but neither is the best fit for this application. k-anonymity is not a good option for reasons described by Cheng et al. [9]: if there are too few users within a given region, then the size of the cloaked location stored would be too large to provide useful data to the service. Similarly, letting each user specify the size of the cloaked region might cause the service to run into problems as discussed

previously if they specify a large area. Instead, I present the following location privacy contract to the users of the application and an implementation differing from those above to achieve it.

3.1.2 Data Privacy Contract

1. A user's exact location will never be stored in any system database.
2. For any user $u_i \in U$, let the stored location of that user in the system be $L(u_i)$. The granularity of $L(u_i)$ will be no better than 1.4 km^2, where $\text{Area}(L(u_i)) \geq 1.4 \text{ km}^2 \ \ \forall\, i$, and the true location of user u_i is uniformly distributed across $\text{Area}(L(u_i))$.
3. Only the most recently received location will be stored for each user, and location history will never be stored.
4. User locations will be stored separately from other user information, including user IDs. The IDs for each user location entry will be hashed using a secure hash function. An adversary would need the user ID, hash function, and entire location table to find a specific user's entry.

This contract ensures that in the event of a data breach or compromise, even if an adversary had full knowledge of the source code, any user locations would only be known within 1.4 km^2 of the true locations. If an adversary only had access to the locations table, then the hashed user IDs would prevent them from knowing which cloaked location belonged to whom without knowledge of user IDs and the secure hash function. Additionally, this granularity of locations stored allows the service to retain the functionality it needs. Next, I present a method for implementing this contract.

3.1.3 Mercator Projection Cloaking

To store the location of the user while maintaining no more than 1.4 km^2 granularity, we opt to take a grid-based approach. We conceptually divide the Earth into a grid, where each square in the grid has area 1.4 km^2. When the system receives a latitude and longitude point from a user, it maps this latitude and longitude on to the grid and stores which square the user reported. As distances between adjacent meridians and parallels on the globe are not constant, the mapping is non-trivial but can easily be solved with a Mercator projection. As

Weisstein [29] shows, for any (latitude, longitude) pair (λ, ϕ), where λ and ϕ are measured in radians, (λ, ϕ) can be mapped to a Cartesian (x, y) point such that $x \in (-1, 1)$ and $y \in (-1, 1)$, where

$$x = \lambda - \lambda_0$$
$$y = \ln \tan \left(\frac{\pi}{4} + \frac{\phi}{2} \right)$$

and λ_0 represents the reference meridian and is equal to 0 when used with standard latitude and longitude systems. To scale these components into units of kilometers, we multiply each component by the radius of the Earth, R_E, and to decrease the granularity in the x and y directions, we can take the floor of each component divided by α and β respectively, where α and β are the granularities desired in each direction. Based on our privacy contract, α and β will each be 1.4 km. Now, our cloaked components take the form

$$x_{\text{cloaked}} = \left\lfloor \frac{R_E \lambda}{\alpha} \right\rfloor$$
$$y_{\text{cloaked}} = \left\lfloor \frac{R_E}{\beta} \ln \tan \left(\frac{\pi}{4} + \frac{\phi}{2} \right) \right\rfloor$$

Note that any (λ, ϕ) pair such that $\phi = \pm\pi$, the y component value is undefined. To store any user location in or query for users that are within proximity to a particular location, I present Algorithms 3.1, 3.2, and 3.3. In Algorithm 3.3, U is the set of all users and τ is used as a constant to eliminate any entries in the database that have expired.

The runtime for MPCloak is $O(1)$, assuming the multiplications and division run in constant time. Under most systems and (λ, ϕ) values, this a reasonable approximation. The runtime of StoreLocation and QueryLocation depend on the storage implementation. Assuming a binary heap and that all of our data fits in memory, we achieve $O(n \log n)$ for both, where n is the total number of users.

Using α and β values of 1.4 km with these algorithms fulfills point 2 of the location privacy contract. Points 1, 3, and 4 depend strictly on implementation, and care to ensure they are met is needed.

One way to ensure that the granularity requirement is met by an application is to move the location cloaking logic to a trusted third party or perform it on the mobile client. The third party can handle cloaking and storage, and the application can simply make queries

Algorithm 3.1 Mercator Projection Cloaking

```
procedure MPCloak((λ, φ))
    if φ is ±π then
        reject
    x ← ⌊R_E λ/α⌋
    y ← ⌊R_E ln tan(π/4 + φ/2) /β⌋
    return (x, y)
```

Algorithm 3.2 Mercator Projection Storage

```
procedure StoreLocation(u_id, (λ, φ), t)
    (x, y) ← MPCloak((λ, φ))
    store(h(u_id), (x, y), t)
```

Algorithm 3.3 Mercator Projection Query

```
procedure QueryLocations((λ, φ))
    (x, y) ← MPCloak((λ, φ))
    R ← u_i ∈ U : h(u_i).location = (x, y)
    S ← u_i ∈ U : h(u_i).t − t_cur < τ
    retrieve(R ∩ S)
```

and handle requests to and from the third party. This may or may not always be practical for end user experience depending on the implementation, and in this implementation, the cloaking and storage is performed server-side while respecting abstraction barriers.

4 Design Overview

The system design consists of three main components:

1. A user-facing mobile app for users to find autoinjectors and defibrillators.
2. Strategically-placed smart containers that house autoinjectors and defibrillators.
3. A cloud presence for handling data and communication between users and smart containers.

These parts work together to ensure that users can find autoinjectors and defibrillators near their location quickly and efficiently. In the following sections, I give an overview of the user experience for each of these components and how they interact during usage. Sections 5 and 5 provide detailed, technical descriptions of these components.

4.1 Mobile App

The mobile app is the primary interface for user interaction with this service. When a user downloads the app on their mobile device, they are required to create an account with system using a unique email address and password. During the onboarding process, the user is required to give the app permission to deliver remote notifications and monitor the device's location. Remote notifications allow the service to inform the user if they are nearby an emergency and can possibly provide assistance. The location of the user is monitored and uploaded to the system's cloud presence using the methods described in Section 3. Monitoring the locations of all users allows the service to send targeted notifications to those near emergencies.

Once a user has been onboarded and their location is being monitored, the user spends little time interacting with the app. If a user is experiencing or witnessing an emergency that requires an autoinjector or defibrillator, the user initiates the "Find an Autoinjector" or "Find a Defibrillator" functions of the app, prompting the user to always call 911-services before proceeding. If the emergency request is for an autoinjector, the cloud service notifies all users nearby the emergency in hopes that they can help. Regardless of the emergency type, the mobile app displays any smart containers housing requested devices nearby and any users who have opted to provide assistance. If a user wants to navigate to a smart

container, the app provides directions. If a user has responded to help, the app shows who is on the way and where they are.

If a user receives a notification that someone nearby needs an autoinjector, the app displays the location of the emergency to the user. The user then has the option to confirm that they have an autoinjector and are willing to provide it to the user in need. The app then provides the user directions to the emergency.

4.2 Smart Containers

As discussed in detail in Section 5.2, the smart containers are meant to resemble, or retrofit to, existing AED defibrillator cabinets (see Figure 5.1). These containers connect to the internet and house the autoinjectors and defibrillators in easy-to-see and easy-to-access locations. During an emergency, a user can navigate to a container via the mobile app. When a user sends their intention to find a container to the cloud, a signal is sent to the appropriate smart container indicating that someone is looking for it. The smart container then activates its LEDs and speaker to flash red lights and sound an alarm to help the user find the container.

When a container is not in alarm mode, it periodically sends telemetry data to the cloud to make sure it is working properly. Because autoinjectors are sensitive to different temperatures, the device monitors and sends temperature and humidity information to the cloud to ensure that no thresholds are breached. In addition, any time someone opens the door of the container, the container lights up and a message is sent to the cloud.

4.3 Cloud

Processing and handling data and communications between users and smart containers requires many cloud-based components. A web server exposes REST API endpoints that handle all communications between mobile devices and the cloud. Message queues handle data sent to the cloud from smart containers to ensure delivery even when the number of containers get large. Multiple databases store all system data. This includes device information telemetry messages, and statuses for each smart container, as well as account information, location, and emergency status information for each user. The cloud processes requests and

messages from users and smart containers to manipulate the data and provide the backend of the service.

In addition to these functions that drive user experience, the cloud also provides an administrator web dashboard for managing and adding smart containers on and to the network. In addition, a simulator provides the ability to simulate smart containers for testing with the mobile app before they have been deployed.

4.4 Alternative User Experience

A mobile app that aims to change behavior during emergencies asks a lot of its users. The mobile app implemented stresses that calling emergency services should always be the first step in an emergency, but often behaviors aren't rational in these situations. A different user experience model involves incorporating emergency operators in the system.

Instead of using the app to find users or smart containers nearby in an emergency, a user in need calls 911 as usual, and the emergency service operator uses the network infrastructure to direct nearby users to the emergency or send the location of nearby smart container to the calling-user's mobile device. The emergency services operator has full knowledge of all ambulances en route and can provide the best action directions to the user based on all knowledge available. This model does not change user behavior during emergencies themselves and offers a more reliable and familiar experience. Letting emergency operators dispatch nearby users was adopted by the PulsePoint app discussed in Section 2.1.

While promising, this model is difficult to build and demonstrate in the amount of time allotted for paper due to the coordination and infrastructure implementations needed to interact with a 911 provider. Instead, the platform utilizes the mobile app described above to let users find nearby users and smart containers themselves. The mobile app, smart containers, and cloud-based services can, however, be easily be adopted to incorporate the emergency operator model described here.

5 Hardware Design

5.1 Goals

The smart containers that house the emergency medical devices need to safely store medical devices and be easy to find. To meet these needs, the hardware design must meet the following specifications for the container:

- Easily recognizable.
- Ability to locate within 1.4 km.
- Easy to access medical devices.

The specifications for the smart container to meet these goals are presented below.

5.2 Form Factor

In order to make the containers easily recognizable, this model builds off of existing Automated Emergency Defibrillator (AED) units that are currently in use. An example of such a unit is show in Figure 5.1. This existing type of container is well-known in most communities and serves as a familiar interface for users when using the smart containers.

Figure 5.1: An AED container currently in use in many communities [17].

Current AED defibrillator containers come in a variety of sizes and models depending on how the container is mounted to the wall, but the interior of most boxes measures approximately $12'' \times 12'' \times 5''$, with the defibrillators themselves measuring approximately $8'' \times 6'' \times 3''$. I assume that the autoinjectors stored in the containers are the same size as the widely-used EpiPens® purchased with a prescription (approximately $5'' \times 1'' \times 1''$), but in practice it might be prudent to create a larger autoinjector form factor that would be harder to steal from the container. Current AED defibrillator containers store one defibrillator per container, and I will do the same. Two autoinjectors are stored per container, as most autoinjector manufacturers recommend always carrying two in case one should fail.

For purposes of prototyping, a CAD model was created and used for manufacturing and testing. Figure 5.2 shows a rendered version of the model. Exterior and interior dimensions of the prototype are smaller than current AED cabinets, but this was simply a manufacturing limitation due to limited access to material and laser cutter printing area and would not be done in practice. Appendix D contains a dimensioned drawing of this model.

Figure 5.2: A 3D CAD model of the prototype that was manufactured and used for testing.

This form factor modeled after existing AED cabinets is easily recognizable in most communities. Retro-fitting existing AED cabinets with the technology described below presents a way to integrate this system into existing locations and communities, perhaps at a lower cost. The design of a retro-fit is not considered in this paper, but because the circuitry components are housed in a small level in the upper-portion of the box, a retro-fitting form

factor could be achieved with minimal invasion of existing cabinets.

5.3 Electronics

Besides simply storing the medical devices, the main value-add of the smart containers is assisting users who are looking for the containers and making them easy to locate. Instead of only providing the location of a container through the mobile app, the container uses physical light and sound signals to help direct users to the correct location. The container features LED lights that flash and a speaker that makes noise when a user is looking for a container. In addition to these signals, a touch screen provides a user interface that allows users or emergency responders to control the lights and sounds of the box in the event they are no longer needed.[2] Furthermore, a screen can be used to show the location of a user who is looking for the container, allowing people nearby to assist in getting the necessary medical devices to the use in need. Lastly, the server is notified if anyone opens the door to the containers that is not registered on the network.

5.3.1 Microcontroller

To facilitate the communication between the container and the server, I selected a microcontroller (μC) that connects to the internet via a WiFi module. These are discretely located within the container and used to control the signal peripherals based on messages received from the server and commands received from the user-facing touch screen.

The microcontroller selected was the Atmel ATSAMD21G18. This chip has 256 kB of flash, 32 kB of SRAM, 6 serial communication modules, and 20 output pins. This is enough memory and I/O capability to allow for multiple software libraries and peripherals, including the touch screen, temperature sensor, LEDs, hall effect sensor, and speaker while remaining comparatively inexpensive.

The WiFi module selected was the Atmel ATWINC1500, an IEEE 802.11 b/g/n WiFi chip with a single-band 2.4 GHz channel that supports WPA/WPA2 Personal and SSL security

[2]This functionality would depend on production circumstances. For instance, if emergency responders are incorporated into the service, perhaps they should control when to turn the container's alarm off. If emergency responders are not involved, restricting passersby from turning the container off for a certain period would be appropriate.

protocols. The Adafruit Arduino Feather M0 was selected as the breakout board for prototyping and contains both the ATSAMD21G18 and ATWINC1500 chips. The Arduino IDE and bootloader were used for programming the microcontroller. Importantly, HTTP, AMQP, and MQTT protocols are supported by the Atmel ATWINC1500 allowing for a choice of protocol configuration, discussed in Section 5.4.

5.3.2 Touch Screen

When a user is looking for an autoinjector or defibrillator, the container lights up and makes noise. Eventually, the user will want to turn the lights and sounds off, and while this can be done remotely from the server, users in the vicinity should be given the ability to do so if enough time has passed since the user in need initiated the alarm. To accomplish this, a display is needed that can render text and accept user input.

The touch screen selected was the Adafruit 2.8″ TFT LCD Touchscreen Breakout Board. This board is capable of simple SPI communication and allows for minimal microcontroller pin usage. The board also has a self-contained graphics controller, so minimal graphic computations are required on the main microcontroller. This screen is small compared to most mobile phone touch screens but is large enough to provide a simple user interface for controlling the container. The screen also has resistive touch capabilities for each pixel of the display, allowing for an easy to manipulate user interface.

5.3.3 Environmental Sensor

Because the epinephrine used in autoinjectors is sensitive to temperature changes and must be kept within a specified temperature range, I use a discrete temperature sensor inside the container to monitor the containers internal temperature an ensure that these temperature requirements are not broken.

The temperature sensor selected was the Bosch BME280, an environmental sensor that measures temperature, humidity, and air pressure with $\pm$ 1.0°C, $\pm$ 3%, and $\pm$ 1 hPa accuracies respectively. According to the datasheet [6], when measuring only temperature and humidity once per second, it draws only 1.8 μA of current. Therefore, when operating at the 3.3 V

logic of the microcontroller, the power dissipated is

$$\begin{aligned} P &= VI \\ &= (3.3\ \text{V})(1.8\ \mu\text{A}) \\ &= 5.94\ \mu\text{W}, \end{aligned}$$

a relatively low power consumption.

One protocol for determining when a temperature or humidity threshold has been breached requires programming each microcontroller to send a message to the cloud service reporting the threshold breach. This requires hard-coding the value into the controller's firmware and offers little flexibility if the devices inside the container and/or their thresholds change.

Instead, the device sends temperature and humidity data to the cloud periodically, around once every 15 minutes. This allows threshold definitions to be created on a per-device basis on the cloud and allows the ability to persistently store temperature and humidity data.

5.3.4 Lights, Hall Effect, and Speaker

For lighting, an iPixel APA102 addressable 1 m LED light strip was selected. This allows for unique light patterns to be generated by the microcontroller that can be used to direct users to the container. A US5881 hall effect sensor is used to detect when the door is opened. A magnet is attached to the inside of the door, and when the magnet moves away from the hall effect sensor, the output pin of the hall effect sensor triggers a hardware interrupt on the microcontroller. When the door is open, the lights come on and a message is sent to the cloud.

One 1″ 8 Ω 0.2 W speakers was used for sound signals. Instead of pulse-width modulating sound from the microcontroller, two LM555 timers are used with a 9V battery to generate a 440 Hz square sound wave. One LM555 generates a square wave that is high for approximately 1 second and low for approximately 0.5 seconds, and the other generates the 440 Hz square wave. The output of the first time is connected to the rest pin of the second, creating a periodic alarm noise. An output pin on the microcontroller controls the reset pin on the first LM555 to turn the sound on and off completely. Figure 5.3 shows the LM55 circuit schematic. Using the LM555 timers instead of pulse width modulation of the microcontroller frees up needed computational resources on the controller for more important tasks such as

network communication.

Figure 5.3: LM555 audio circuitry. Pin 4 of the left-most LM555 is connected to the microcontroller.

According to the LM555 datasheet [13], the following equations govern the time-high, τ_H, and time-low, τ_L, in seconds of the square wave produced by the LM555.

$$\tau_H = 0.693(R_2 + R_3)C_5$$
$$\tau_L = 0.693R_3C_5$$

The actual resistor and capacitor values used to achieve the 1-second on and 0.5-second off and 440 Hz waves are detailed in Table E.1.

5.3.5 Power

When the Arduino Feather M0 breakout board is connected to a computer via a USB cable, the microcontroller is fully-powered and performs all required functions. The microcontroller contains a 3.3 V voltage regulator that supplies power to all peripherals, except for the LM555 timers. These are powered by a 9 V batter due to the need to produce a loud alarm noise via a sound wave with large amplitude.

A Lithium Ion battery is also connected to the Arduino Feather M0 breakout board and charges whenever the device is connected to the computer via the USB cable. When the microcontroller is disconnected from the USB, it reverts to battery power. Both the Lithium Ion battery and 9 V battery were used for development and testing.

In production, however, a persistent, stable power source would be needed. A 110 V wall power source could be used with a voltage regulator to power the circuitry, although far less

voltage is required. Existing AED cabinets and their proximity to a power source would need to be considered before production.

5.3.6 Schematic

Figure 5.4 shows the complete schematic for the hardware circuitry, and Table E.1 in the Appendix lists the resistor and capacitor values used.

Figure 5.4: Complete hardware circuitry schematic.

5.4 Network Protocols

A major consideration in the hardware design is the choice of communication protocol with the server. There are several choices supported by the network controller, and each has its own strengths and weaknesses for this application. The details of these are presented in the following sections below.

5.4.1 HTTP

Hyper-text transport protocol. This is the defacto network protocol used by most web sites and mobile apps today.

Pros. Widely used and supported. Easy to integrate with existing systems [2].

Cons. This is a complex protocol. It has no built-in Quality of Service mechanism, so any failed request must be manually retried by the server or client [27]. Requests sent over HTTP are much larger than the same ones sent over AMQP or MQTT, and thus require more overhead on the client [27].

5.4.2 AMQP

Advance Message Queuing Protocol. This is a protocol built for use in message-oriented environments and is often used for similar IoT applications.

Pros. The protocol is lightweight compared to HTTP. It is known for its reliable message handling via queues and wire-level protocol [1]. This protocol is often used to communicate between systems in enterprise applications [19].

Cons. The protocol is very complex and low-level, and currently not all systems support it [2].

5.4.3 MQTT

Message Queue Telemetry Transport. Developed for instances where client memory and/or bandwidth is limited.

Pros. This protocol is lightweight compared to AMQP [28]. It does well with devices that do not have reliable connections to the network and has three built-in quality of service layers. Additionally, using a C MQTT library only uses approximately 30 kB of data, less than that of AMQP [27].

Cons. It is not as widely adopted and supported as HTTP.

For this application, the MQTT protocol is utilized. It provides a lightweight framework that uses very little memory and power for requests. Additionally, the reliability of sending/receiving messages is much greater than that of HTTP and comparable to AMQP.

5.5 Prototype 1

The first prototype was built using the components listed above. As a simple proof of simple, a cardboard box was used to house the components and simulate the functionality of an actual container. Figures 5.5 and 5.6 present the implementation.

Figure 5.5: (a) Circuit components and (b) front view of the first container prototype.

This prototype succeeded in communicating with the cloud platform described in Section 6 and performed all required design specifications.

5.6 Prototype 2

To improve upon the design, a new container housing was built with a more durable and appropriate structure, and the circuit was re-implemented to fit more streamlined with the

Figure 5.6: (a) Closed front view and (b) perspective view of the first container prototype.

desired form factor from Figure 5.2. Figures 5.7 and 5.8 present this second iteration.

Figure 5.7: (a) Circuit components and (b) front view of the first container prototype.

When a user is locating a container using the mobile app described in Section 6.4, the LED lights start flashing and the speaker begins to sound an alarm. The screen displays a message detailing that someone is looking for the container. The screen also has a button to turn off the lights and sound. The lights and sounds turn off automatically after 15 minutes regardless.

The container also regularly communicates with the device administration service described in Section 6. Every 15 minutes, the container sends a telemetry message to the service reporting the temperature, humidity, and whether the door is closed. In addition, if the door is ever opened, a hardware interrupt causes the container to alert the service immediately.

Figure 5.8: (a) Closed front view and (b) perspective view of the first container prototype.

This constant communication ensures that the service knows which devices are working and if any have gone offline.

5.7 Evaluation

The smart container built satisfies all of the goals specified in Section 5.1. Because of the form factor, the container resembles current AED containers, and, with the proper materials and manufacturing, a model could be achieved with even better resemblance. The flashing lights and sounds make the container extremely easy to locate when the user is nearby, and the internet connectivity coupled with the mobile app described in Section 6.4 allow a user to find the device from any location—meeting the 1.4 km requirement proximity requirement. The large front-facing door makes it easy and intuitive for users to obtain the medical devices inside.

6 Software Design

6.1 Goals

This application requires three main software components: a website for the administration of the smart containers, a REST API for manipulating system data, and a mobile app to let users interact with the service.

For the device administration website, we present the following design requirements:

- Add, update, retrieve and delete smart containers and their properties from the service.
- Receive, store, and process device data and messages from the smart containers.
- Simulate smart containers for testing.

For the mobile app and REST API, we require the following:

- Creation of individual user accounts for use with the service.
- Ability for users to find smart containers nearby.
- Crowdsourcing autoinjectors by pinging nearby users.
- Directing a user providing an autoinjector to a user in need.

These design requirements, in conjunction with the smart container hardware in Section 5, provide a service for users to quickly find defibrillators and autoinjectors in emergencies.

6.2 Architecture and Implementation

There are three types of "devices" accessing this service: smart containers that hold medical devices, browsers that allow device administration, and mobile apps that allow users to use the core service features. These devices all interact with cloud components to communicate with each other, manipulate data, and cause physical actions via the smart containers.

As described in Section 5, the smart containers are constantly sending messages to the service reporting their current statuses. Therefore, the devices need a server-side MQTT compatible endpoint for delivering and receiving messages to and from the service. When the number of devices gets large, processing many messages becomes difficult and resource intensive; to

compensate, we designed the service with a message queuing system for processing these telemetry messages reliably and quickly.

In addition to the smart containers' communication with the service, an administrator also needs to be able to manipulate devices via a device administration platform. To achieve this, we designed a website for accessing, modifying, and controlling devices and their data. This requires a web server to handle traffic from browsers and a database for persisting data. The mobile app also needs access to these resources to let the client interact with the service. To achieve this, we built a REST API into web server to handle these requests.

Figure 6.1 provides a high level overview of the software architecture. The sections following detail each component and the actual platform used for implementation. We utilized Microsoft Azure's IoT products and adopted components from one of their solutions [4] to streamline the design process.

Figure 6.1: Overview of software architecture.

6.2.1 Device Hub

The device hub establishes the network entry point for the smart containers. The device hub selected for implementation is the Microsoft Azure IoT Hub. It supports MQTT communication between devices and provides a per-container security interface. When a container entity is created in the backend, a specific device with unique access keys are generated in the

hub. These access keys are then copied on to the physical container and used to authenticate all communication with the hub. Messages and telemetry data received from the containers are forwarded to the message processor via the device hub, and the main web app uses the device hub to send messages and commands to the smart containers.

6.2.2 Message Processor

The message processor is used to handle the large ingestion and processing of messages passed from the smart containers via the device hub. To free up resources for the rest of the service, the message processor determines whether each message from the container should be stored immediately or forwarded along to the main web service for further action. For instance, if the container is simply reporting the humidity and temperature and both fall within a pre-determined threshold, the message processor simply stores the telemetry data in blob storage (see Section 6.2.4). However, if the telemetry data raises an alarm, or if the message is meant to be processed further, it forwards the message to the message queue for further processing. The blob storage container contains per-container thresholds for temperature and is used by the message process to make the decisions. The message processor used for implementation is the Microsoft Azure Stream Analytics engine.

6.2.3 Message Queue

The message queue receives all messages received from smart containers that require action beyond long-term storage. The message queue holds all messages and allows the web app (see Section 6.2.6) to process them individually as resources allow. The message queue used for implementation is the Microsoft Azure Event Hub.

6.2.4 Blob Storage

Blob storage is a type of unstructured storage that is useful for storing non-relational data. Our blob storage database stores three things:

1. Telemetry history (temperature, humidity, door status) from smart containers.
2. Per-container threshold telemetry rules for each device.

3. Information for devices being simulated by the device administration simulator.

This storage is cheap and allows for large amounts of data to be stored without much overhead. The blob storage used for implementation is Microsoft Azure Blob Storage.

6.2.5 NoSQL Storage

NoSQL storage is desired because it allows for low-latency data retrievals. The NoSQL database stores two things:

1. Properties such as latitude, longitude, command history, and other metadata for each smart container that is accessed by web app.
2. All user data related to the app, such as account information, instances of emergencies, user locations, etc.
   (a) It should be noted, though, that as per Section 3.1.2, user location and user metadata are not stored in the same specific database—only databases of the same type.

This database is what the mobile app and device administration platform interact with the access data. The NoSQL storage used for implementation is Microsoft Azure DocumentDB.

6.2.6 Web App

The web app is effectively the center of the architecture. It handles all the web requests from the device administration website, API requests from the mobile app, and messages from the smart containers received from the message queue. In addition, it also hosts a simulator that simulates smart containers on the network by interacting with the device hub. The web app used for implementation is the ASP.NET framework built on Microsoft Azure Web Services.

6.3 Hardware Dashboard

The hardware dashboard interface is adopted from a Microsoft Azure IoT example [4]. Figure 6.2 shows the main administrative dashboard. An administrator can view telemetry from real or simulated containers and see alerts triggered for each container. In Appendix A, Figure A.1 shows the view listing all real and simulated devices on the network and their properties. From here, an administrator can change container properties, send commands to a container, or disable one from interacting with the service. In Appendix A, Figure A.2 shows where an admin can add a real or simulated device to the network.

Figure 6.2: Container administrator dashboard for viewing container telemetry and alarm data.

6.4 REST API and Mobile App

Besides device administration, the web app also contains the REST API used by the mobile app. The REST API exposes endpoints that allow a user from the mobile app to create

an account, report their current location, and notify the service they need a defibrillator or autoinjector.

The mobile app provides the client-side interface for all non-administrative users. Once the user opts-in to sharing their location, the mobile app will periodically send the user's location to the service and store it using Algorithm 3.2. Then, when a user is looking for other users with autoinjectors nearby, the user sends a request via the app, and the service notifies users nearby who can opt-in to respond.

The screenshots in Figures B.1 and 6.4 show a few of the essential screens from the app. Figure B.1(a) is the home screen. Here, a user can respond to an emergency or find an autoinjector or defibrillator. After requesting one of these devices, in Figure 6.4(b) the user is prompted to always call 911 first before proceeding to finding users and/or containers near their location.

Figure 6.3: (a) Home screen and (b) after starting an emergency request, the user is prompted to first call 911.

Figure 6.4(a) shows a user looking for autoinjectors and container nearby, and Figure 6.4(b) shows the user responding to an emergency request from a user named Jennifer. More

screenshots are provided in Appendix B.

Figure 6.4: (a) Finding nearby users and containers and (b) responding to a request from Jennifer.

6.5 Evaluation

The software implemented satisfies the goals presented in Section 6.1. The container administration platform allows an administrator to add, create, update, and delete smart containers from the service. The web app and message queuing system handles all messages from smart containers efficiently, and the web app provides a simulator for simulating containers for testing.

In addition, all the mobile app goals were accomplished. Users have their own accounts and can use the app to find smart containers and autoinjectors nearby. When pinged by another user for an autoinjector, a user can navigate to someone in need.

To make this software infrastructure scalable, relevant coding and software engineering practices were used. For instance, the inversion of control paradigm was used for both the web

server and mobile app design. In this model, each class or object created to perform tasks does not initialize its own dependencies. Instead, pre-initialized dependencies are taken as constructor arguments for all classes. Each class uses the objects it is passed for execution. This paradigm provides an abstraction barrier to the services used by each class, as each class is coded around its dependency interface. It also allows for easy and thorough unit testing, as mocked objects can be passed in as decencies when testing a particular class. This paradigm allows for more abstraction and a scalable infrastructure.

In addition to inversion of control, cloud-based, dynamic configuration settings are used for all database connection strings and application settings for the web app. This allows for private connection strings to be stored outside of the code base, and application settings can be changed without scheduling a new deployment. This dynamic settings are useful when an application is being used in different regions and both production and development servers are running.

For deployment, a continuous integration protocol was used. Git was used as the version control system, and any time a new commit is pushed to the mater branch, a new deployment is triggered and all web services are updated and relaunched.

7 Conclusions and Future Work

This system allows users to quickly and efficiently find autoinjectors and defibrillators near them by leveraging crowdsourcing and strategically-placed smart containers. To create the system, the service utilizes a user-facing mobile app, Internet-connected smart containers, and a cloud service for processing and handling data and communications. Based on population and prescription data, crowdsourcing autoinjectors is feasible given reasonably high population densities, and the presence of smart containers makes the network useful even when no registered users are nearby. Under the hood, the service processes user locations while preserving a reasonable level of privacy and an added level of comfort to the user.

Initial prototypes of the mobile app and smart containers are useful and promising. To prepare forreal deployment, a user-friendly and intuitive design is needed for the app. Hardware components would need to be implemented on a printed circuit board (PCB) and optimized for mass production. In addition, creating a "retro-fit" system for integrating the smart container circuitry into existing AED cabinets would allow for optimal placement and integration into communities.

Working with 911 call centers also has promising indications. By integrating with existing systems, this service can more easily integrate with current user behavior during emergencies while moving some liability issues to the call centers.

Other future steps include conducting user interviews. Putting this app in the hands of users under controlled circumstances and receiving feedback would be invaluable to the development. Additionally, partnerships with local hospitals and allergy clinics would provide insight into the perspectives of experts on cardiac arrest and anaphylactic shock and a line of communication with potential customers.

A Container Administration Screenshots

Figure A.1: Container administrator view of all container in the system.

Figure A.2: Container administrator ability to add a real or simulated container.

B Mobile App Screenshots

Figure B.1: (a) Pop-out menu and (b) user login.

C Bill of Materials

Table C.1 lists all project items and their costs. Resistors, capacitors, wires, and solder are omitted. Microsoft Azure prices listed are for production level implementation. During development, free tiers were used and the price was significantly lower.

Item	Vendor	Part No.	Qty	Unit Cost	Monthly Cost
Adafruit Feather M0	Adafruit	3010	1	34.95	0.00
1″ 8 Ω Speaker	Digikey	102-2501-ND	1	2.99	0.00
2.8″ TFT LCD Touchscreen	Adafruit	1710	1	29.95	0.00
US5881LUA Hall Effect Sensor	Adafruit	158	1	2.00	0.00
High-Strength Magnet	Adafruit	9	1	2.50	0.00
Full-Size Breadboard	Adafruit	239	1	5.95	0.00
Half-Size Breadboard	Adafruit	64	1	5.00	0.00
Addressable LED Strip	Sparkfun	14015	1	15.95	0.00
Lithium Ion Battery	Adafruit	15778	1	7.95	0.00
Web App Services	Msft. Azure	-	1	0.00	55.88
IoT Hub	Msft. Azure	-	1	0.00	50.00
DocumentDB Storage	Msft. Azure	-	1	0.00	23.81
Blob Storage	Msft. Azure	-	1	0.00	0.03
Event Hub	Msft. Azure	-	1	0.00	11.16
Stream Analytics	Msft. Azure	-	1	0.00	79.20
				$107.24	**$220.08**

Table C.1: Bill of materials for project implementation at scale.

D Engineering Drawings

Figure D.1 shows the major dimensions of the prototype container created.

Figure D.1: CAD Drawing for prototype container.

E Specification Sheets and Values

Component	Value	Units
R_1	10	kΩ
R_2	5	kΩ
R_3	91	kΩ
R_4	5	kΩ
R_5	160	kΩ
C_1	4.7	nF
C_2	4.7	nF
C_3	4.7	nF
C_4	4.7	nF
C_5	15	μF
C_6	0.1	μF
C_7	10	nF
C_8	0.1	nF

Table E.1: Resistor and capacitor values used in circuitry.

Figure E.1: Adafruit Arduino Feather M0 specification sheet with Atmel ATSAMD21318.

Figure E.2: Adafruit 2.8” Color TFT Touchscreen specification sheet.

Figure E.3: Adafruit Arduino Feather M0 specification sheet.

Figure E.4: iPixel APA102 addressable LED strip specification sheet.

Figure E.5: US5881LUA hall effect sensor.

F Repository

The source files for this project are far too numerous to include in this paper. Instead, the following GitHub repository will be maintained for as long as circumstances allow and can be found at

`https://github.com/gmhewett/senior_thesis`

References

[1] Guizani M. Mohammadi M. Aledhari M. Al-Fuqaha, A. and M. Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 14(4):2347–2376, 2015. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7123563.

[2] Chad Arimura. Cloud messaging protocol: Amqp vs http, February 2013. URL: https://www.iron.io/cloud-messaging-protocol-amqp-vs-http/.

[3] National Fire Protection Association. NFPA 1710, standard for the organization and deployment of fire suppression operations, emergency medical operations, and special operations to the public by career fire departments, 2016.

[4] Microsoft Azure. Remote monitoring preconfigured solution walkthrough. URL: https://docs.microsoft.com/en-us/azure/iot-suite/iot-suite-remote-monitoring-sample-walkthrough.

[5] Joseph K. Blitzstein and Jessica Hwang. *Introduction to Probability*. CRC Press, Boca Raton, Florida, 2014.

[6] Bosch. Bme280. URL: https://cdn-shop.adafruit.com/datasheets/BST-BME280_DS001-10.pdf.

[7] Raymond C. Browning, Emily A. Baker, Jessica A. Herron, and Rodger Kram. Effects of obesity and sex on the energetic cost and preferred speed of walking. *Journal of Applied Physiology*, 100(2):390–398, 2006. URL: http://dx.doi.org/10.1152/japplphysiol.00767.2005.

[8] US Census Bureau. American community survey 1-year data (2011-2015) api. Generated by Gregory Hewett. Accessed 1 November 2016. URL: https://www.census.gov/data/developers/data-sets/acs-1year.html.

[9] Reynold Cheng, Yu Zhang, Elisa Bertino, and Sunil Prabhakar. *Preserving User Location Privacy in Mobile Data Management Infrastructures*, pages 393–412. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. URL: http://dx.doi.org/10.1007/11957454_23, doi:10.1007/11957454_23.

[10] Erin Durkin. Longer 911 response times stir city council. *New York Daily News*, March 2014. URL: http://www.nydailynews.com/news/politics/longer-911-response-times-stir-city-council-article-1.1737399.

[11] Bugra Gedik and Ling Liu. Location privacy in mobile systems: A personalized anonymization model. *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference*, pages 620–629, 2005.

[12] HeartSafe. Saving lives in the community. URL: http://www.heartsafe.org.uk/.

[13] Texas Instruments. Lm555 timer. URL: http://www.ti.com/lit/ds/symlink/lm555.pdf.

[14] Richard E. Kerber, Lance B. Becker, Joseph D. Bourland, Richard O. Cummins, Alfred P. Hallstrom, Mary B. Michos, Graham Nichol, Joseph P. Ornato, William H. Thies, Roger D. White, and Bram D. Zuckerman. Automatic external defibrillators for public access defibrillation: Recommendations for specifying and reporting arrhythmia analysis algorithm performance, incorporating new waveforms, and enhancing safety. *Circulation*, 95(6):1677–1682, 1997. URL: http://circ.ahajournals.org/content/95/6/1677, arXiv:http://circ.ahajournals.org/content/95/6/1677, doi:10.1161/01.CIR.95.6.1677.

[15] Able Technology Limited. Aed locations. URL: https://itunes.apple.com/us/app/aed-locations/id424094430?mt=8.

[16] John Lee MD. Boston Children's Hospital. Phone Interview. 27 October 2016.

[17] Tiger Medical. Image. URL: http://images.tigermedical.com/Products/LargeImages/ZOL8000-0855.jpg.

[18] Mylan. Accessed: 2017-01-03. URL: https://www.epipen.com/en/what-is-anaphylaxis.

[19] Oasis. Amqp is the internet protocol for business messaging, March 2017. URL: http://www.amqp.org/about/what.

[20] Peter T Pons and Vincent J Markovchick. Eight minutes or less: does the ambulance response time guideline impact trauma patient outcome? *Journal of Emergency Medince*, 23(1), 2002. URL: http://dx.doi.org/10.1016/S0736-4679(02)00460-2.

[21] PulsePoint. Prince george's county fire/ems department brings lifesaving technology to region via 9-1-1 integrated smartphone app. October 2015. URL: http://www.pulsepoint.org/.

[22] Food Allergy Research and Education. Food allergy facts and statistics for the u.s. *Journal of Applied Physiology*, 100(2):390–398, 2016. URL: http://dx.doi.org/10.1152/japplphysiol.00767.2005.

[23] Matt Rocheleau. Boston ems response times slow amid 911 call surge. *Boston Globe*, November 2015.

[24] Jonathan D. Rockoff. Mylan faces scrutiny over epipen price increases. *Wall Street Journal*, August 2016. Online. URL: http://www.wsj.com/articles/mylan-faces-scrutiny-over-epipen-price-increases-1472074823.

[25] L Sweeney. k-anonymity: A model for protecting privacy. *International Journal Of Uncertainty, Fuzziness & Knowledge-Based Systems*, 10(5):557, October 2001.

[26] TMHC. URL: http://www.topmastersinhealthcare.com/30-most-technologically-advanced-hospitals-in-the-world/.

[27] Leonardo Olivera Sweta Rawat Nagesh Subrahmanyam Rong Xiang Valerie Lampkin, Weng Tat Leong. Building smarter planet: Solutions with mqtt and ibm websphere mq telemetry. URL: http://www.redbooks.ibm.com/redbooks/pdfs/sg248054.pdf.

[28] VMWare. Choosing your messaging protocol: Amqp, mqtt, or stomp, February 2013. URL: https://blogs.vmware.com/vfabric/2013/02/choosing-your-messaging-protocol-amqp-mqtt-or-stomp.html.

[29] Eric W. Weisstein. Mercator projection. *MathWorld*–A Wolfram Web Resource. URL: http://mathworld.wolfram.com/MercatorProjection.html.