Do-It-Yourself IoT for Homes: How Simple Can It Be?

Bijan Naimi
Saratoga High School
Saratoga, CA, USA
bijanrnaimi@gmail.com

Tommi Mikkonen
University of Helsinki
Helsinki, Finland
tommi.mikkonen@helsinki.fi

ABSTRACT

With the Internet of Things (IoT), the world is rapidly becoming more and more programmable. This opens up new opportunities for creating devices in a fashion where individual contributors come up with a concept, and then create a corresponding design – assuming that the technologies, tools, and development kits are simple and straightforward enough for non-professional use. In this paper, we study the do-it-yourself approach to IoT for home security to learn how practical it is to enter IoT development in this domain without formal programming education, other than training and guidance that can easily be found on the internet. More precisely, the first author of this paper – a high school student – designed and implemented a home security IoT application that monitors premises using off-the-shelf hardware and Node.js. In order to generalize said student to the average DIY developer, we will then break down the skill sets used in our implementation and analyze the accessibility of such knowledge. In this paper, we introduce the system and reflect on what was learned in the development process as well as the implementation.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**;

KEYWORDS

Internet of Things, IoT, do-it-yourself

ACM Reference Format:
Bijan Naimi and Tommi Mikkonen. 2020. Do-It-Yourself IoT for Homes: How Simple Can It Be?. In *Proceedings of MindTrek Conference (MindTrek Conference'18).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

With the Internet of Things (IoT), the world is rapidly becoming more and more programmable [7, 8]. This opens up new opportunities for creating devices in a fashion where individual contributors come up with a concept, and then create a corresponding design – assuming that the technologies, tools, and development kits are simple and straightforward enough for non-professional use.

For the purposes of this paper, we define DIY (do-it-yourself) as the practice of developing a project from scratch that would normally be found built and completed out of the box, although there are also wider definitions [1]. Obviously, it would be illogical for the average consumer to build their own sensors, network protocol, and central computers to draw data from the sensors, and therefore we accept these components of the system can be purchased off-the-shelf.

In this paper, we study the DIY approach to IoT for home security to learn how practical it is to enter IoT development without formal programming education other than training and guidance that can easily be found on the Internet. More precisely, the first author of this paper – a high school student – designed and implemented a home monitoring IoT application that keeps an eye on given premises using readily available hardware and platforms such as Raspberry Pi and Node.js. Towards the end of the paper, we reflect learnings from the development process as well as the implementation. Specifically, we will generalize the high school student by breaking down the skill sets used in our implementation and the online accessibility of such knowledge, such as simple mobile development, embedded devices, and server-side development.

To be clear, our implementation is merely a sample implementation. There are a multitude of methods and materials a DIY user can use to implement the design requirements we will lay out. Our implementation is the least complex it can be in order to argue how simple DIY home monitoring can be.

To enable wider study and evaluation of the results presented in the paper, the complete implementation mentioned in this paper is available at https://github.com/branai/rpi-zwave.

The rest of this paper is structured as follows. In Section 2, we present the background of the work. In Section 3, we give the requirements for the design and materials that could be used to achieve these design requirements. In Section 4, we introduce the implementation created as a part of the study. In Section 5, we present a discussion regarding the lessons we have learned during the implementation process. Finally, in Section 6, we draw some final conclusions.

2 BACKGROUND AND MOTIVATION

As the domain of the experiment, we have selected home monitoring, a common target for IoT applications [2, 5]. From a commercial point of view, it seems as though now more than ever we are surrounded by a growing market for home security technology, surveillance, and other services. This growing market and the companies and products striving to occupy it creates ruthless competition. Subsequently, these companies constantly try to exceed one another in their technology and the benefits in their packages. And while this competition may produce innovation and advancements in the field of connected devices, the competition often also pushes these home security companies to provide unnecessary features in their packages that inevitably forces these companies to increase their markup as a means to break even.

MindTrek Conference'18, October 2018, Tampere, Finland

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

For example, monitoring solutions from companies like ADT, Bay Alarm etc. come with monthly fee averaging between $30 to $50[1] which is prohibitive to consumers on a budget. As for activation fees, the most popular providers, average at around $100. Even the most popular self monitoring systems lure customers in with the independence of their systems only to force large initial costs (e.g. Adobe: $429[2], Canary: $349[3], iSmartAlarm: $149[4]). All in all, we consumers are left with our only options being to pay large fees for services that are comparable to what one could build themselves.

The essentials of home security are unique to the individual. In a perfect world, users can add features to their systems based off of necessity, and more importantly, affordability for the end user. This is the very definition of do-it-yourself and the goal of this paper – to bring home monitoring to the DIY domain. Hence, the deliverable of this paper is a minimalistic home security system and the analysis will be on the amount work and knowledge needed to create said system. We will not only assess the quantitative code one would need to write from scratch to build the system, but the knowledge needed in creating the system and understanding how it operates to prove whether or not IoT home security is an applicable project for the average DIY user.

To be clear, we understand that simply presenting a system implemented by an individual without professional training says little; a single individual can not speak for all. However, by understanding the elementary nature of the qualitative requirements needed to implement the system, it can be made clear that the development of a reliable, "starter" home security system can be achieved without difficulty by the average DIY user.

3 DESIGN REQUIREMENTS AND MATERIALS

We defined DIY as the practice of developing a project from scratch that would normally be found built and completed out of the box. We must remember, however, that it would be illogical for the average consumer to build their own sensors, network protocol, and central computers to draw data from the sensors. Therefore, these are all components of the system that should be purchased off shelf.

Again, all of these components are aspects of our sample implementation. Of course, there are a multitude of methods and materials a DIY user can use to implement the design requirements we will lay out, but these materials allow for the most simplicity in all aspects of the implementation. See Lessons Learned for more details.

3.1 Central Computer

This component is the computer that will create the gateway for the sensors on the home security system and handle the readings and informations of these sensors. While this makes the central computer arguably the "brain" of the system (sensor information and its translation into a readable format is the backbone of a home security system), the concrete requirements we must consider when choosing which brand to purchase off shelf are fairly straightforward.

The essentials are simply an operating system running a variant of Linux, available at an affordable retail price. However, we must remember that an average DIY user would most likely already own a computer with a Linux-based operating system, allowing the user to save money by avoiding purchasing a whole new computer. Nevertheless, if one were forced to purchase the central computer from off the shelf for whatever reason, the Raspberry Pi[5] seems the most fitting.

3.2 Sensor Protocol

A sensor protocol is needed to enable wireless communication with the various sensors in our home monitoring system. It is vital the most secure protocol is the one used for our system; a home monitoring system is only as secure as its most insecure component. The three leading sensor protocols on the market are ZigBee[6], Bluetooth Mesh[7], and Z-Wave[8].

ZigBee advertises itself as a "complete and interoperable IoT solution

...

that allows many smart objects to work together". And while ZigBee as a product works very similarly to its competitors (as we will find with many of these protocols), there has been research [4] that raises questions about the security of touchlink commissioning, one of the two commissioning modes ZigBee devices operate on. Again, the a home monitoring system is only as secure as its most insecure component. Therefore, because of these questions raised, we will not use ZigBee as our protocol. Bluetooth Mesh may be the most recognizable product, but the protocol operates on a noisy frequency. While this gives no direct reason to not use Bluetooth Mesh, it does create a sense of ambiguity in whether or not the protocol would be able to operate successfully as often as the other protocols on the market. Finally, Z-Wave operates on a quiet frequency and seems to be widely accepted by both users and the developer community. And while this also may raise concern about whether this creates an opportunity for a malicious third party to tamper with your system, Z-Wave is more established and accepted than Bluetooth Mesh (2001 vs 2015). Therefore, we find that Z-Wave is the most reasonable networking protocol to use.

3.3 Network Protocol

A requirement of any home security system is the characteristic of wireless communication between the components of the system. An important "off the shelf" aspect of the system we must then consider is the network protocol. The two obvious contenders for our network protocol are file transfer protocol FTP [6] and hypertext transfer protocol HTTP [3]. Both have variations that encrypt all communications (SFTP and HTTPS), but as discussed in the design flow of the Implementation section, an authentication system can be implemented in the home security system that deems the benefits of using encrypted connections negligible.

[1] http://www.asecurelife.com/home-security-systems-comparison/
[2] https://goabode.com/sale/
[3] https://canary.is/multipacks/#starter-pack
[4] https://www.ismartalarm.com/packages/preferred-package/ISA00001.html
[5] https://www.raspberrypi.org/
[6] http://www.zigbee.org/
[7] https://www.bluetooth.com/bluetooth-technology/topology-options/le-mesh
[8] http://www.z-wave.com/

The major benefit of FTP over HTTP is that FTP is faster when transferring files with larger file sizes. However, the information needed to be communicated for a minimalistic home security system is minimal, as discussed in the design flow of the Implementation section. As for the JavaScript that the protocol clients and server run, the amount of code written for both are fairly similar. However, the amount of steps FTP takes to establish connections and transfer files far exceeds the request-response two-step process of HTTP. This is important because we want to preserve the simplicity of the system in all ways possible.

4 IMPLEMENTATION

The two vital parts of any home security system are the central computer and the sensors connected to said computer. The client device is simply the device the user uses to receive information about the sensors from the central computer. This will most likely be a mobile device of some sort, such as a smart phone. The server-side components are simply all other aspects of the home security system: the sensors on the ZWave network and the central computer to these sensors. We used a Raspberry Pi, ZWave PIR sensors, and an iPhone 6S in the deliverable to act as the central computer, the sensors connected to it, and the client device respectively. Other materials are added in the design flow.

4.1 Standards

While we have recognized the basic functionality of each of the aspects of the minimalistic home monitoring system, it is important to also identify the standards and more specific attributes our system must have to not be just a home monitoring system, but a successful and reliable one as well.

The central computer will collect all sensor states and changes to these states and make this available for the client to download in the form of a string we will call the "state" string, as the string holds all information about the state of the sensors, including, but not limited to, the date of activation, battery power, and last trigger date. Naturally, the last trigger date and battery power are updated in the state string whenever their respective values change.

The client then must contain the logic to react to these changes to this state string correctly based off of whether or not the user has set an alarm. The steps taken to "send" this string to the client are discussed in the Design Flow, but each time the state string is downloaded by the client, the client will compare the last trigger date of each sensor with the dates on the state string from the previous download by JSON parsing the strings and traversing through them. If there is a change, a sensor has been triggered and the client can proceed with respect to the alarm the user has set.

4.2 Design Flow

We can begin creating our minimalistic home monitoring system by first imagining an unrealistically simplistic system. Consider if all sensor information were to be stored on the Raspberry Pi in a JSON object called state. Then, assuming the Raspberry Pi is also taking HTTP requests, the client can access the sensors' state JSON object with a simple HTTP protocol, as the stringified state JSON object can be made the response message of the Raspberry Pi to the client. This is demonstrated in **Figure 1**.

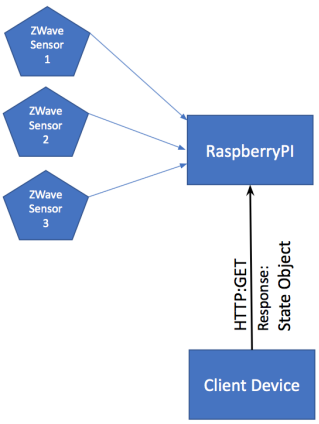

Figure 1: Overview of the design.

Here, the client is downloading the state of the sensors straight from the Raspberry Pi. While this framework may be adequate for client to Raspberry Pi communication that takes place purely within the same home WiFi network, the obvious truth is that users will want to monitor their homes from outside their homes; i.e., outside WiFi networks or cellular data. If the user wishes to do this using only a Raspberry Pi that is connected to their home WiFi network, they would need to forward a port on the router of the user's home network and route this port to the Raspberry Pi. In other words, the user must relinquish the security of their home network - the very network our home security is connected to - in order to allow client server communication from outside the home's WiFi network.

Therefore, in order to preserve the security of our home, we must introduce an intermediary server as shown in **Figure 2**.

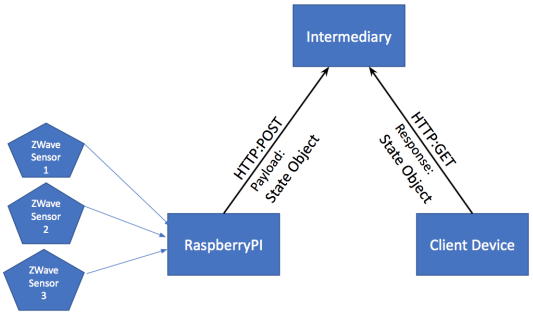

Figure 2: Introducing an intermediary server.

As seen in the diagram, we have turned the Raspberry Pi into an HTTP client of sorts; the Raspberry Pi will send the state JSON string to the intermediary server and the intermediary server will make the state JSON string accessible to the client accordingly. In other words, the only function of the intermediary server is to act as a middle man that just holds files on a public port; the existence of an intermediary server allows the user to access the state JSON

string. This intermediary server can be any web hosting service, but in the deliverable AWS[9] is used.

This structure allows user communication from outside their home WiFi network. However, allowing an unencrypted state JSON string to be accessed by the general public creates yet another issue, as users would feel insecure in having the information about their sensor network accessible to literally anyone who knows the IP address of the intermediary. Therefore, we must introduce an entity that can make the state JSON string secret. And while user password authentication can be done with HTTP, this option will create additional lines of code and memory. Instead, we can introduce AES encryption to encode the state JSON string. We can then write send this encrypted string to the HTTP server using a POST request, and the HTTP server can make this encrypted string the response to any HTTP request that is not a POST method. Therefore, anyone can download the encrypted state string, but only those with the encryption key can decrypt, read, and parse the string into a JSON object. Then the client can save a downloaded state string and decrypt its contents using the client's local key; the state string is public to all, but it is only readable to those who have a key i.e. the client. This concept is shown in **Figure 3** on the following page.

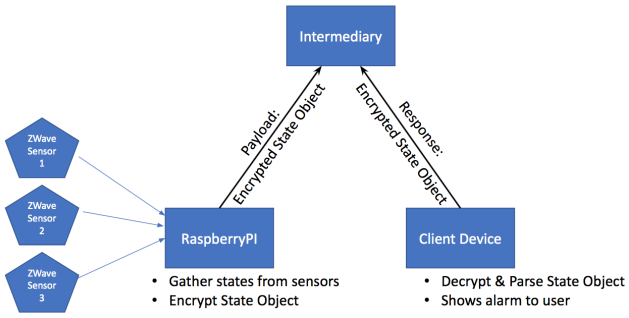

Figure 3: Adding encryption.

This not only solves the problem of making private information inaccessible (or in this case, unreadable) to a malicious third party, but it also facilitates protecting against malicious third parties sending "fake" state strings to the intermediary in hopes that the fake state string will corrupt the whole system. However, now that we have introduced encryption, the client knows what strings they download are safe versus fake, as they can check if a parse of the decrypted contents of the string return an error or not. The client can then ignore strings that return an error when parsed and save strings that parse successfully.

As a matter of fact, this "parse checking" method of verifying string legitimacy takes only a couple of lines of code. Therefore, it makes sense to implement this into the intermediary server as well, because that way all components of the user's system are verifying the strings they receive and make public. Now the user can rest assured that at any given time, the state information stored on any component is legitimate.

[9]https://aws.amazon.com

What we are left with is a secure pathway of communication in which the Raspberry Pi can relay information about its sensors to the client. Now that our final design is established, we can take the time to analyze the more vital aspects of the design on a more in depth level.

4.3 State JSON

Throughout the Design Flow we repeatedly referred the "state JSON," the object or string containing information about the sensors. When the Raspberry Pi receives sensors and live data from these sensors, the Raspberry Pi will create and add these changes to a JSON object called state. In the deliverable, the state object holds each sensor's id number, name, date added, battery life, ready boolean, and last trigger date (as shown in **Figure 4**). While some of this information is not vital to a minimalistic home monitoring system, we added it to the JSON object to illustrate how easily accessible the information is for a more complex system.

```
State object has an array of
these sensor objects:
• id: integer
• name: string
• battery: string
• date: string
• ready: boolean
• lastTriggerDate: string
```

Figure 4: JSON object describing the state.

While most modern home securities have extra features that require more data types (i.e., video surveillance or carbon dioxide sensors), a minimalistic home security system needs only to record a true or false reading (movement or no movement, danger or no danger). Ideally, we would store the boolean in the state object. However, in preserving the simplicity of our system, we set up the Raspberry Pi component to send the stringified state JSON whenever any value is changed is setup, whether it be the motion boolean or a battery level change. Therefore, a better option than storing the boolean in the object, which changes and sends the state object twice per sensor trip (false to true to false) is to store the last date a sensor read true, a strategy that only changes and sends the object once per sensor trip. That way, the simplicity of the process of communication is maintained.

4.4 Alarms

In the Standards, we touched on how the client will set alarms for their home security network, so here we will delve deeper into this aspect of our system. The client device is used by the user to access the state JSON string and use the local private key to decrypt the the string and parse to be left with an object. This information about the sensors is always updating because the the sensors are always on and reading; the client is downloading the state JSON string frequently enough so that the information on the object on the client can be considered a "live view" of the sensors.

The user can use this "live view" with methods programmed on the client device to handle abnormal readings with alarms. The user can then set an alarm to on, which will notify the user when the client reads a last trigger date that is different from the last trigger

date of the previous download. On the other hand, the user can set the alarm to off, which will not download anything from the intermediary until the alarm is set to on again. When the alarm is switched on again, the client application will alert the user if any motion was sensed while the alarm was off: a status report on the home security system while the alarm was switched off. This process is shown in **Figure 5**.

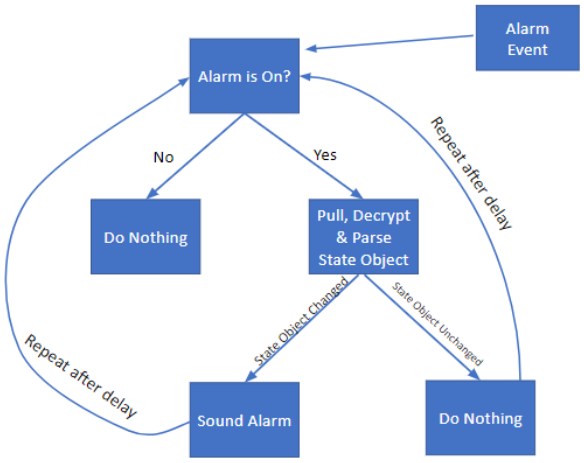

Figure 5: Resulting design.

Put simply, the Raspberry Pi and intermediary server create and make the availability of data to the user. In other words, the Raspberry Pi and intermediary server act no differently if the object they receive would imply a sensor trip or not. All of the alarm logic detailed above is only on the client device.

4.5 System Security

The security of our system extends further than the "parse checking" we implemented at the end of the design flow. We originally implemented "parse checking" into the intermediary and client components of our system to identify fake state strings malicious third parties try to upload to the intermediary server or the client. However, while adding this feature may allow the system to detect bad clients, stopping here still leaves the system prone to attackers whose objectives are to break the system rather than surpassing its security. Therefore, we also implemented a kick-out feature that destroy the HTTP requests of IP addresses with a history of bad state JSON string parses.

And while parse checking occurs on both the server and client components of our system, we must remember that the client device acts as an HTTP client, so we would be unable to implement a kick-out on the client device. With a kick-out feature in place, the system is now protected from high frequency string uploads meant to preoccupy the system from the Raspberry Pi's uploads, unrealistically long strings meant to break the system, and any other pragmatic security threats meant to distract the server from legitimate HTTP clients.

5 LESSONS LEARNED

All things considered, the ability to communicate relevant information securely on a system that can defend against realistic attacks makes the minimalistic home security system the deliverable demonstrates a commonsensical utility in the real world. Now that we have laid out a sample implementation, we can study the amount of work and requirements – both quantitative and qualitative – needed to apply this implementation; we can then determine whether or not home security is reasonable in the DIY domain. Any assumption made that someone without a professional education is able to learn or do something about the project build is, again, supported by the fact that the developer of the sample implementation is a high school student. As we will show, however, these aspects of the implementation are all basic and elementary to begin with. This sample implementation is the project deliverable.

5.1 Quantitative Requirements

By analyzing the quantitative requirements of implementing our design into the deliverable, we can estimate the amount of concrete work and costs that come with DIY home security.

The total file size of the deliverable, including these two packages and the NativeScript framework, is 255.4MB. As for amount of user written code, the server application has a total of 46 lines of written code, the RaspberryPi application has a total of 82 lines of written code, and the client has a total of 55 lines of written code (not including the front-end). This brings the total number of lines of user-written code to about 183 lines. As a result, besides the actual learning of the programming language used by the user to build the home monitoring system, there is a surprisingly small amount of quantifiable work in the form of written code needed to be done to build the system.

Consider the visual comparison of the amount of code written in each of the components in **Figure 6**.

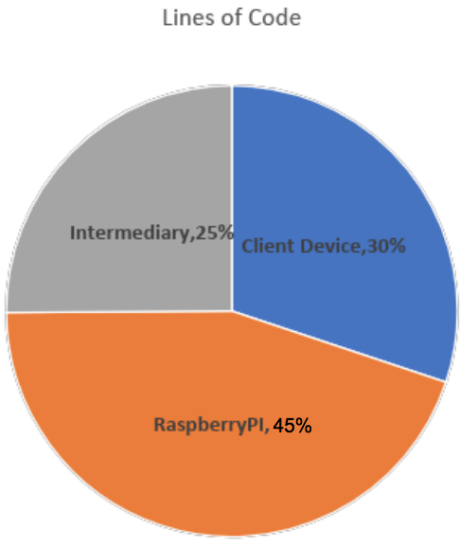

Figure 6: Distribution of code in lines.

The amount of code on the server application and the client application are somewhat similar due to their similar functionalities. Notice how the amount of written code on the client and server applications are half of the the amount of written code on the Raspberry Pi application: an application which has the main functionality of creating and updating the information of a simple object. This further illustrates how quantitatively low-demanding this project is, as the application that requires the most code has a functionality that is very simple in its development. Obviously, it should be noted this is a low-level concept in programming and it can be learned without major background knowledge in computer science, but we discuss this and other qualitative needs (i.e., encryption and networking) in the Qualitative Requirements.

Another major quantitative aspect is the cost to set up this home security system. As stated earlier, the mobile device used in the deliverable to act as the client device was the iPhone 6S. However, we will not include this in the final cost evaluation, given the user is most likely to have a mobile device of their own. Also stated earlier, we used the Raspberry Pi in the deliverable. However, we will not include the cost of the Raspberry Pi in the final cost evaluation because of the level of ambiguity in how much money an average user would have to spend to achieve this requirement. This is because any potential user could achieve just as good of a final product with any computer on the market, even with a computer with minimal specs they may or may not already own. It is this level of ambiguity that prompts us to not include the cost of a the central computer in the total cost.

Instead, we can analyze the more concrete purchases a user would be required to make. In the deliverable, the sensor protocol ZWave was used, and this requires the purchase of a ZStick[10] ($44.95)to create a wireless gateway for these sensors to operate on. These sensors average around 50 dollars each, with the cost of the sensor[11] in the deliverable being $44.36. Finally, the intermediary server was made using the free tier[12] of Amazon Web Services, so this contributes nothing to the final evaluation.

For the average user, the price of a DIY minimalistic home monitoring system is $44.95 plus about $50 per sensor, which even with several sensors is less than the cost of many of the commercial home security systems mentioned in the Background section. Therefore, we can definitively say that given the cost requirements to build a minimalistic DIY home security system versus these commercial prices, the DIY option can be the more appealing choice and should not be an issue for the average DIY user.

In terms of quantitative requirements, the DIY domain proves to be applicable territory for home security, as it is both cost-efficient and work-efficient.

5.2 Qualitative Requirements

In addition to the quantitative requirements discussed above, a potential user would need to meet several qualitative requirements, like a certain level of competence and prerequisite knowledge, to successfully undergo such a project. It should be noted again that the first author of the paper is a high school student: the deliverable that implements our presented design proves it is possible to undergo such a project without a professional background.

We understand that simply presenting a system implemented by an individual without professional training says little; a single individual can not speak for all. However, by understanding the elementary nature of the qualitative requirements needed to build the system, it can be made clear that the development of a reliable, "starter" home security system can be achieved without difficulty.

Throughout this paper, we have referenced how the DIY user must have "basic programming knowledge" in Node.js, the programming language used in the deliverable. We define "basic programming" knowledge as the understanding of simple concepts such as control flow, loops, and variables. We provide a link to the deliverable in the acknowledgements. However, this is not all the programming knowledge needed to undergo a DIY home security project; we analyze the additional requirements and obstacles below to determine if the DIY domain is right for home security.

Firstly, in order to use the ZStick to communicate with ZWave sensors, one must use the only working OpenZWave wrapper for NodeJS, openzwave-shared[13]. This package provides the developer with the software needed to receive information from ZWave sensors. However, this information is massive on its own; the information presented to the developer by openzwave-shared details every minor piece of data about the ZWave sensors on the network. Therefore, the only major qualitative requirement on the DIY end is the ability to locate the necessary information in this data structure, but we argue this falls under basic programming that goes no further than an understanding of objects.

The client device was programmed using the NativeScript[14] framework in order to keep the use of NodeJS consistent across the deliverable. The front-end needed in the client application is almost non-existent, but we wanted to at least display a switch that acted as the alarm to the system. To understand how to do so, one must look no further than example front-ends that come with NativeScript.

The only other third party package needed is a cryptography toolbox that is supported by NativeScript (seeing that the built in module crypto is not supported by NativeScript). In the deliverable, we used crypto-js[15]. To be clear, just because cryptography is needed in the design does not mean the DIY developer must be fluent in the crypto library. As a matter of fact, AES string encryption, which is used in the design, is the first concept documented in the API of the npm page[15] of crypto-js.

In addition, as we have stated in the design flow, in the implementation presented, an EC2 instance running Linux with Amazon Web Services is used to act as our intermediary server. However, the actual knowledge the user would need to set up such a server is negligible, as step by step tutorials[16] detailing how to create such virtual machine are provided by Amazon themselves.

[10] https://www.amazon.com/Aeotec-Z-Stick-Z-Wave-create-gateway/dp/B00X0AWA6E/ref=sr_1_1?s=hi&ie=UTF8&qid=1521075947&sr=1-1&keywords=zstick

[11] https://www.amazon.com/Z-wave-Detector-install-Immunity-PIRZWAVE2-5-ECO/dp/B01MQXXG0I/ref=sr_1_11?s=hi&ie=UTF8&qid=1521074567&sr=1-11&keywords=zwave+sensor

[12] https://aws.amazon.com/free/

[13] https://www.npmjs.com/package/openzwave-shared

[14] https://www.nativescript.org

[15] https://www.npmjs.com/package/crypto-js

[16] https://aws.amazon.com/getting-started/tutorials/launch-a-virtual-machine/

Finally, a DIY user would need to have a general understanding of HTTP, specifically the methods needed to allow client to server and server to client communications like POST and GET. However, these two methods are as far as the DIY user must need to go to implement the system. As a matter of fact, the user would only need to scratch the surface of HTTP to find an understanding of POST and GET in online tutorials[17].

Notice the trend with the programming requirements we have found. While there are concrete domains of knowledge referenced throughout the development of the system, the actual knowledge needed is basic and elementary, just scratching the surface in each of their respective domains. An average DIY user with basic programming knowledge should have no problem implementing a home monitoring system.

While progressing through the design flow, we encountered several minor issues in the development process. For example, the multitude of releases of NodeJS[18] created trial and error in finding versions supported by all of the third party packages used. However, we must remember that online communities like Stack Overflow[19] provide assistance in minor problems such as these and other common developmental problems. While this could be tedious at times, we thought this to be a small road bump in the scope of the whole development process.

It is safe to say the most tedious part of building the deliverable issues like the one described above. However, this should illustrate how uncomplicated all other aspects of the development process were: facilitation exists on the Internet in extremely accessible ways. As a result, we can definitively say that due to the simplicity of knowledge needed in the domains of skill referred to in the deliverable, the DIY domain proves to be an applicable territory for DIY home security – even if certain minor aspects of the development process may seem tedious to the average DIY user.

6 CONCLUSIONS

To summarize our findings, it is becoming increasingly feasible to implement sophisticated IoT systems without professional education. As a matter of fact, the resolving of any issues that arose during our implementation were just a click away. Time and time again we found that the all documentation, issue support, and general help can be found on the Internet. We expect that this trend will only strengthen in the future, as new technologies will become available in open source contributions.

At the same time, the professional dimension of software industry – extended product life, upgrades that protect from security flaws, and end-user support – will become important factors. Indeed, while it is feasible to produce individual designs without much technical professional background and domain knowledge, their long-term survival needs commercial interests.

REFERENCES

[1] Paul Atkinson. 2006. Do it yourself: democracy and design. *Journal of design history* 19, 1 (2006), 1–10.

[2] Sean Dieter Tebje Kelly, Nagender Kumar Suryadevara, and Subhas Chandra Mukhopadhyay. 2013. Towards the implementation of IoT for environmental condition monitoring in homes. *IEEE Sensors Journal* 13, 10 (2013), 3846–3853.

[3] Jeffrey C Mogul, Jim Gettys, Tim Berners-Lee, and Henrik Frystyk. 1997. IETF RFC 2616 Hypertext Transfer Protocol–HTTP/1.1. (1997).

[4] Philipp Morgner, Stephan Mattejat, Zinaida Benenson, Christian Müller, and Frederik Armknecht. 2017. Insecure to the touch: attacking ZigBee 3.0 via touchlink commissioning. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 230–240.

[5] D Pavithra and Ranjith Balakrishnan. 2015. IoT based monitoring and control system for home automation. In *Communication Technologies (GCCT), 2015 Global Conference on*. IEEE, 169–173.

[6] Jon Postel and Joyce Reynolds. 1985. IETF RFC 959 File Transfer Protocol: FTP. (1985).

[7] Antero Taivalsaari and Tommi Mikkonen. 2017. A roadmap to the programmable world: Software challenges in the IoT era. *IEEE Software* 34, 1 (2017), 72–80.

[8] Bill Wasik. 2013. In the programmable world, all our objects will act as one. *Wired. Available online: http://www. wired. com/2013/05/internet-of-things-2/(accessed on 22 January 2016)* (2013).

[17] https://www.w3schools.com/tags/ref_httpmethods.asp

[18] https://nodejs.org/en/download/releases/

[19] https://stackoverflow.com