Adam Stock

The University of the West of England Faculty of Environment & Technology Frenchay  

Discrete internet Communicator -An ultra-low power, low cost network connected haptic feedback wearable device.

Contents

[Abstract 3](#_Toc36654586)

[Acknowledgements 3](#_Toc36654587)

[Introduction 3](#_Toc36654588)

[Background 3](#_Toc36654589)

[Direction 3](#_Toc36654590)

[Intended audience 3](#_Toc36654591)

[Objectives 4](#_Toc36654592)

[Research 4](#_Toc36654593)

[Haptic feedback 4](#_Toc36654594)

[Does it exist? 5](#_Toc36654595)

[Hey Bracelet 5](#_Toc36654596)

[Sony Wena 7](#_Toc36654597)

[Generic low-cost smartwatch 7](#_Toc36654598)

[Comparison 8](#_Toc36654599)

[Energy harvesting 9](#_Toc36654600)

[Mechanical harvesting and automatic watches 9](#_Toc36654601)

[Piezoelectric 10](#_Toc36654602)

[Seebeck/Peltier 10](#_Toc36654603)

[Solar 10](#_Toc36654604)

[Conclusion 11](#_Toc36654605)

[Requirements 12](#_Toc36654606)

[Android app 12](#_Toc36654607)

[Functional 12](#_Toc36654608)

[Non-Functional 12](#_Toc36654609)

[Haptic Powerband 13](#_Toc36654610)

[Functional 13](#_Toc36654611)

[Non-Functional 13](#_Toc36654612)

[Database 14](#_Toc36654613)

[Functional 14](#_Toc36654614)

[Non-Functional 14](#_Toc36654615)

[Methodology 14](#_Toc36654616)

[Design 15](#_Toc36654617)

[High level design 15](#_Toc36654618)

[Architecture 16](#_Toc36654619)

[Implementation 18](#_Toc36654620)

[Prerequisite’s 18](#_Toc36654621)

[Platform selection 18](#_Toc36654622)

[CC2640R2F 18](#_Toc36654623)

[NXP-QN908X 19](#_Toc36654624)

[NXP-QN902X 19](#_Toc36654625)

[Comparison 20](#_Toc36654626)

[Hardware decision 20](#_Toc36654627)

[Software usage 21](#_Toc36654628)

[Virtual machine 21](#_Toc36654629)

[Software tools and development environment 22](#_Toc36654630)

[First steps, setup and hardware testing 22](#_Toc36654631)

[Database 28](#_Toc36654632)

[PHP Scripts 30](#_Toc36654633)

[Android App 30](#_Toc36654634)

[Wristband 31](#_Toc36654635)

[Evaluation 32](#_Toc36654636)

[Conclusion 32](#_Toc36654637)

[References and Bibliography 32](#_Toc36654638)

[Appendices 32](#_Toc36654639)

[Php scripts 32](#_Toc36654640)

# Abstract

Words go here

# Acknowledgements

Acknowledgements go here

# Introduction

## Background

This project was inspired by my housemate during my second year. She was experiencing some relationship issues with her partner at the time and after a long teary chat with her the primary cause of the issue was revealed – she felt that with her partner working long hours with his independent business and being out of contact with him for much of this time, that he would not think about her (it is worth noting that this is at time in history when most people have smart devices and are rarely unreachable).

This got me to thinking that a discreate device without the distraction abilities of a smartphone or computer could fill this void in the technological landscape while also being able to provide secondary useful functions to a user.

## Direction

Thinking on this subject further drew me to the concept of a simple low-cost device that can provide notification and potentially response to a user about an extremely limited number of matters without the user needing to actively seek out this information or be concerned with the devices upkeep or setup.

From this I noticed the ubiquitous appearance over the last decade or two of rubber wristbands worn by people to show support for various causes such as Cancer Research and Help for Heroes. These bands were inexpensive and did not require maintenance from the wearer as per other forms of jewelry.  
With this in mind, I wondered if my notification idea could be integrated into a band like this in such a way that the band maintains its key plus points – low cost, zero maintenance and robustness.   
I have an interest in mechanical watches and wondered if that kinetic (often described as ‘automatic’) mechanisms used to power many high-end watches such as those by Rolex, Omega and Patek Philippe could be adapted to provide power to the device allowing the user to never have to perform maintenance for the life of the device.

## Intended audience

Below we will outline the main intended audience for this product. However, this should not be considered an exhaustive list as use cases should be considered an interpretive element.

* Youth (>18)
  + ‘best friends’
  + Boyfriend/Girlfriend
  + Parents summoning children
* Adults
  + Parents of the above
  + Healthcare workers (if embedded in rubber as charity wristbands hygiene should be simple)
  + Hospital staff as a method of mass notification – as a replacement to a pager
  + Events staff as a method to notify groups of people
  + Lovers
  + Those in need of discreate notification

## Objectives

Below we will outline the main objectives of the project. Though not all of these objectives may be fulfilled they will be taken into consideration at each stage of the design and productions process.

* Low cost
* Low power requirements
* Short range communication with a more powerful device
* Haptic feedback

# Research

## Haptic feedback

In our day to day lives we are provided with huge amounts of information from our senses , our brains interpret these inputs from our senses and react accordingly (Filbey and Gazzaniga, 2013). Our brains prioritize these inputs and que them according to importance – for example if your hand was on fire that would take priority over an incoming social media message. Often events like the aforementioned social media message are not a priority and do not require immediate response, they are just a notification.

Studies have shown that notifications are often attended to immediately and that this can cause notable negative effects on task performance in a work context (Pielot and Rello, 2017). Many of these notifications do not require the immediate attention given to them when they create a visual or audible stimulus.

Another disadvantage of visual or audible notification systems in common use are that they are available to everyone within their area of effect – ‘area of effect’ or ‘aoe’ is a video/board game term to describe attacks or other effects that affect multiple targets within a specified area. Modern smart devices are capable of tailoring their notifications based on the source and settings. This customization may include different alert noises, different colored notifications LED’s and most obvious- display notifications. This clearly infringes or has potential to infringe on the user’s privacy if others are present in the aoe at the time of notification.

A way round this is to direct the stimulus at the user alone, this could be accomplished with a directional audio system (Sun and Okada, 2008) or a directional visual cue such as a laser. Seeing as the health effects of firing coherent high strength streams of photons at the human body has relatively well documented effects (Alam *et al.*, 2003; Tierney, Eisen and Hanke, 2011) we can rule this out for the time being. Regardless both aural or visual stimulus as expected above would require complex tracking of the user backed up by hardware physically capable of directing the output, both of which would likely require either physical tracking system such as a gimble or similar or an array of output devices arranged in such a way that one or more could be used to direct at a stand-alone target. Considering the notifications are to be provided from a portable device such as a mobile phone, both physical and computational prowess are limited and the usage of the aforementioned would provide a huge drain to the already limited resources, not to mention the line of sight required for both tracking and notification. This makes the whole idea quite impractical.

We are left with 3 (primary) senses that can potentially provide us with notification – smell, taste and touch. Given that both smell and taste are strongly related with both being chemical measurement systems based internally in the human body, both are realistically outside of the scope of this product.

This leaves us with our sense of touch, this is a part of the somatosensory system which includes other physical related senses such as the ability to sense heat (thermoception), chemicals (chemoreceptors as used in smell and taste), nociception (pain reception). The receptors involved in touch called mechanoreceptors translate mechanical movement against the skin into electrical signals that our brain can perceive (Tee *et al.*, 2015).

At this point touch is the best candidate for our purposes as our sense of touch can be stimulated mechanically without chemical intervention – making the design process significantly simpler as no components of the device are single use as may be the case with chemical stimulus. Due to its physical nature its scope and area of effect can be tightly controlled. In our case placing this stimulus on the wrist of the user has the benefits of both privacy – only the user will know when the stimulus is activated and the position of the wrist having strong association with emotional cues will help fulfil the design goals of the project.

Research has already been completed on the usage of haptic feedback embedded into clothing and during this it was noted that users struggled to understand messages from multiple sources placed on the arm at the same time (Ueda, Terada and Tsukamoto, 2019). In this project we will use a single actuator, this will stop the aforementioned from being an issue as well as reduce the complexity and therefore cost of the product.

## Does it exist?

Seeing as the technologies required for this project all exist already (energy harvesting, short range radio, haptic feedback) research was conducted by seeing if similar products already exist on the market and if so, do they meet the objectives that have been set out for the device? in short no, many devices exist that fulfill one or more of the objectives but they fail to meet all in a reasonable manor.

Some of the researched devices are outlined below –

### Hey Bracelet

(House of Haptics, 2018) – ‘The revolutionary bracelet that sends touch over distance’

A picture containing spectacles

Description automatically generated

Figure

The Hey Bracelet consists of a pair of bracelets and a smartphone companion app. These items align with my first idea closely with their tag line of ‘The revolutionary bracelet that sends touch over distance' they seem to be aimed squarely at lovers with distance separating them. The bracelet consists of a motor that allows the band to tighten around the wearer’s wrist and a touch sensor on the outside to instigate this action in another user. However, these devices diverge from my idea in two key areas. Firstly cost, at $114.99 at the time of writing (18/11/18) it is difficult to consider these a low-cost item, and secondly power, they require charging via USB.

### Sony Wena

(SONY EUROPE LIMITED, 2018)- ‘The watch. Reborn.’A picture containing sky

Description automatically generated

Figure

The Sony Wena is not a watch but in fact a strap that should fit many watch faces and is available in multiple sizes and designs, including several Sony made face designs. This device is positioned as a more all-round smart device with GPS (GPS, GLONASS, QZSS and SBAS compatible), Bluetooth 4.2 low energy, heartbeat and accelerations sensors, OLED display and NFC like (‘Wena pay’) hardware. Its features include receiving notifications from a smart phone, contactless payment and fitness tracking as well as being water resistant to enable to user to wear the device in the shower or humid conditions. The device has an expected battery life of approximately 1 week with a charging time of 1.5 hours. So, in many ways this device would cover the functions of my idea with two key differences, firstly it requires charging (albeit far less frequently than other described devices) and secondly its price tag. Starting at £349 the device is clearly aimed at the premium end of the market and can certainly not be considered disposable or suitable for a younger user.

### Generic low-cost smartwatch

There are many such devices on the market as of 2019 and it would be beyond the scope of this work to review all of them, so in this instance we will focus on a single model the ‘V8 Smartwatch’ this device is made by several companies and often sold under the label’s ‘GONOKER’ , ‘Leegoal’ and ‘DreaT’. 

Figure

This device consists of the basic functionality of a feature phone in the form factor of a watch. It includes a MediaTek 6261s processor which is based around an ARM Cortex M4 or M7, this is assumed based on information from MediaTek’s website and a datasheet stating it uses the ARM7-E instruction set (Huang, 2012; MediaTek Inc, 2019) . This variant of the chip uses the J-S extensions to enable acceleration of Java based applications. It contains an inbuilt logic for Bluetooth 3.0, GSM/GPRS/EDGE, FM radio, AAC audio (with amplifier), I2C, SPI, UART, USB and SDIO and can be debugged using a JTAG interface. These are accompanied by 128MB or RAM, a 1.51” 240x240 display and a 0.3MP camera. This high amount of integration enables the device to run for a day or more on a single 280mAh battery.

The device can be set to take notifications from an accompanying Bluetooth enabled smart phone using its provided Android or IOS app and has a price of under £10 meaning that it is likely the closest to meeting the goals of our project.

### Comparison

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Product | Notification | Battery life | Additional features | Cost |
| Hey Bracelet | (Single notification based on proprietary app) | 1 week | Waterproof | £86.82 ($115 USD 7/03/2019) |
| Sony Wena | (Multiple based on a proprietary app) | 1 week | Waterproof, OLED display, Wena Pay | Starting at £349 |
| Generic Smartwatch | (Multiple based on a proprietary app) | 1-3 days | Phone features, Apps, Camera, SD card, Colour LCD display | > £10 |

Table

## Energy harvesting

A potential solution to power a device such as is being created in this project would be energy harvesting. This is the process of using some of the energy the human body naturally expends during its normal function and converting/storing this for use by a device, this is often transparent to the end user.

Methods for doing this have existing in many forms for hundreds of years but have only recently become viable to power wearable electronics. This is due to efficiency improvements in both the harvesting methods and in the devices they are intended to power. Below we will outline the details of several of these methods and their suitability for our purposes.

### Mechanical harvesting and automatic watches

This is likely the oldest form of energy harvesting done by humanity with origins known to be dating back at least as far as Daniel Schwenter (1585-1636) and his idea to use respiration to wind a mechanical watch (Richard Watkins, 2016) however this approach is not known to have been used practically.

Commonly this method involves a weight connected to a shaft in such a manner that when attached to the human body this weight will cause the shaft to partially rotate due to the common movement of the person. This shaft would then be used with a ratcheting mechanism to wind a spiral torsion spring storing the collected energy for slow and constant release via a lever escapement (WF&CO, 2019).

The amount of energy stored by mechanisms as described above can power a mechanical watch for upwards of 50 days from a full charge (Hublot, 2019) however at a cost of £262,000 a mechanical device like this is far beyond the funding and scope of this project. Research has been completed into the efficiency of these mechanisms and this is thought to be approximately 45%.(Longhan Xie, Carmen G. Menet, Ho Ching, Ruxu Du, 2009) This however does not give us power in a form that can be used to drive electronics.

This problem has to a certain extent been solved by companies starting with Seiko with their ‘Kinetic’ range beginning in 1988 (Seiko, 2018) followed by various other companies including Citizen and Swatch. These watches contain a mechanical harvesting system as described earlier but instead of transferring energy to the escapement of a watch, they power small generators and in turn a battery or capacitor used to run a quartz clock movement. As these technologies are proprietary the actual power values are not disclosed to the public, however upon research some have theorized the production to be within an order of magnitude of 1 joule every 2 days or 0.000277778 Watt hours (Wardell, 2016). At the 1.8v minimum we are likely to need for our microcontrollers this can be converted to 0.15432 mAh using the formulae

Or in our case

Or 154.32 Microamps. This is likely enough for our electronics but is unlikely to be enough to drive a haptic feedback device like a vibration motor.

### Piezoelectric

This method works on the concept that many materials (often crystalline) generate small quantities of electricity when put under mechanical stress, this may be from compression or deformation (flexing) this is often used in industry to detect vibration or sound. In order to use this energy circuitry is required to normalize the energy produced.

When put under stress the piezo element will output a voltage in a range based on the amount of mechanical stress inputted. This can typically range from 20/30 V + down to zero hence the requirement of a buck regulator or similar, to keep the output voltage at a level that is safe to use without burning out energy storage components or the microcontroller itself.

Piezoelectric energy is a reversable process so care must be taken that the energy harvested cannot be conducted back into the piezo element as this would result in deformation of the element and in turn loss of the potential energy.

The amount of energy created with this method is quite low with high voltage outputs and low current (ampere) output. It is not uncommon for elements to put out voltages in a range up to 100 volts or more if under extreme load (such as being hit by a hammer) but the ampere output is usually less than 100 microamps (Elhalwagy. et al, 2017; Hickman, 2017). With outputs like this, losses involved in rectifying and converting the energy output may result in very low efficiency.

### Seebeck/Peltier

The Seebeck/Peltier effect is the effect that when a temperature gradient is applied to a conductor the energy will excite negatively charged particles (electrons) at the hot end of the conductor and cause them to move towards the cold side producing a tiny voltage. Due to the tiny amount of energy produced connecting these in series as would be done with a battery results in the series connections losing the energy generated via resistance and the effects of the temperature gradient. To counter this a second material is used to provide the reverse connection, they are doped in such a way that positively charged particles move from the hot side to the cold thus continuing the flow, these are the same concepts as are used in semiconductors and are also called p-type and n-type conductors.

The output of a thermoelectric generator (TEG) such as this is affected by several factors. Starting with materials used, we are looking for materials with a high electrical conductivity and a low thermal conductivity at room temperature, in this case the likely choice would be bismuth telluride (Bi2Te3) as it provides these properties in a reasonable temperature range. Note other materials exist but either exhibit these properties in the wrong temperature ranges or require use of complex and expensive nanotechnologies bringing costs above the already expensive bismuth telluride.

Research has already been conducted into the viability of this form of energy production (Melissa Hyland, Haywood Hunter, Jie Liu, Elena Veety, Daryoosh Vashaee, 2016) . They conclude that in a best-case scenario we can expect production of around 20 μW/cm2 with a more realistic output of 2-8 μW/cm2. Given that our device electronics will likely only have a footprint of a few cm2 additional TEG’s would need to be located in the wristband giving additional cost and complexity to the design.

### Solar

Solar cells have been relatively common in portable electronics since pocket calculators popularized them in the late 1970s (Tout, 2019) and with recent pressure for renewable energy sources have become popular to power everything from cars to homes. These cells, also known as photovoltaic cells, receive energy in the form of light and use this to excite electrons in a layer of a semiconducting material to produce an electrical output. Being a relatively mature technology, these devices can be found in many form factors, the main one we will examine is the BPW34 silicone pin photodiode (Vishay.com, 2008). This device measures 5.4 x 4.3 x 3.2 mm and has a surface area of 0.23 cm2. It can produce around 50 µA under good conditions. For our purposes this means several of them may be required to meet our power needs even under good or near ideal conditions. Given that our device is going to be wrist mounted it is unlikely that these conditions will be met due to clothing covering cells, shallow light angles from the user having mobility of their hands, as well as factors like being indoors. This means in order to meet our power requirements multiple cells would be required as well as a method to store the energy for continued usage, this would add complexity and cost to the project.

### Conclusion

Below is a table summarizing the above methods –

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Technology | Cost | Complexity | Size/Practicality | Power potential |
| Mechanical | High (Commonly <£100) | High as no premade units are available | Acceptable as often already installed in wearables | Viable as already used to power similar wearables |
| Piezoelectric | Low (£1-5 for device and associated electronics) | Minimal as units already exist | Large to produce required power output | Low power density |
| Seebeck | Low (£1-5 for device and associated storage) | Minimal as units already exist | Large to produce required power output | Low power density |
| Solar | Low (£1.50 per cell as of 15/03/2019) | Minimal | Many cells required to negate physical limitations | Viable as multiple cells can be used |

Table

A characteristic of the aforementioned energy harvesting methods is that they provide extremely low amounts of current. Given proper storage using a capacitor or even a small rechargeable battery the amount of energy collected even while being intermittent in delivery should be enough for the relatively low power but frequent draw of a microcontroller. However, the use of a vibration motor will cause a high current draw for a relatively long time and is likely to overwhelm most of the above methods if used frequently – this should be taken as a consideration during the design process.

Given that a wearable device will often have a lifespan of less than 5 years, the complexity and cost of the above methods and the low cost and ubiquity of high capacity low volume batteries such as the CR-2477 by Panasonic (Mouser, 2019). The work involved in producing a product with a harvesting method like this is likely not worthwhile.

The aforementioned battery has an output of 3v meaning it can power the device directly, a capacity of 1000mAh meaning it can potentially power a low power device for several years. As well as having a large operating range in heat, solid state that is unlikely to be affected by movement and no reliance on light meaning it will likely provide a more reliable power source for the lifetime of the device.

# Requirements

This section of the document outlines the overall requirements for the project and uses MoSCoW (Must have, Should have, Could have, Won’t have) prioritization. They are split into two sections functional and non-functional for each element of the product.

Note requirement numbers are labeled in the following way –

X1.X2.X3

1. Functional (1) or Non-Functional (2)

2. Component, Android application (1), Hardware device (2) or Database (3)

3. Requirement number

## Android app

Requirements for the android application portion of the product are as follows -

### Functional

|  |  |  |
| --- | --- | --- |
| Requirement # | Requirement description | Category |
| 1.1.1 | App boots in android | Must have |
| 1.1.2 | App has network functionality | Must have |
| 1.1.3 | App has Bluetooth functionality | Must have |
| 1.1.4 | App retries dropped network attempts | Should have |
| 1.1.5 | App retries unsuccessful Bluetooth communication | Should have |
| 1.1.6 | App should set notification in database | Must have |
| 1.1.7 | App should receive notification from database | Must have |
| 1.1.8 | App should create new user | Should have |
| 1.1.9 | App should have function to silence device | Should have |

Table

### Non-Functional

|  |  |  |
| --- | --- | --- |
| Requirement # | Requirement description | Category |
| 2.1.1 | App should communicate over wide area network | Must have |
| 2.1.2 | App should communicate over a personal area network | Must have |
| 2.1.3 | App should perform notification in a timely manner (>10s) | Should have |
| 2.1.4 | App development should be low cost | Should have |
| 2.1.5 | App should perform its own housekeeping | Should have |

Table

## Haptic Powerband

Requirements for the powerband hardware are as follows -

### Functional

|  |  |  |
| --- | --- | --- |
| Requirement # | Requirement description | Category |
| 1.2.1 | Device should power to a ready state automatically | Must have |
| 1.2.2 | Device should get to a pairable state from first boot automatically | Must have |
| 1.2.3 | Device should retry connections if dropped | Should have |
| 1.2.4 | Device should pair to android app | Must have |
| 1.2.5 | Device should receive notification from app | Must have |
| 1.2.6 | Device should vibrate upon notification receipt | Must have |
| 1.2.7 | Device should send confirmation of vibration | Should have |
| 1.2.8 | Device should have a way to silence notifications | Could have |
| 1.2.9 | Device should provide differentiation between reasons for notification via haptic feedback (intensity and/or duration of pulse) | Could have |
| 1.2.10 | Device should have a visual representation for reason of notification (RGB light or display) | Could have |

Table

### Non-Functional

|  |  |  |
| --- | --- | --- |
| Requirement # | Requirement description | Category |
| 2.2.1 | Device must provide primary notification to the use alone (discretion) | Must have |
| 2.2.2 | Device must provide long service life without user input | Should have |
| 2.2.3 | Device must have physical attributes to enable comfortable wearing/use | Must have |
| 2.2.4 | Device should be of near disposable cost (>£10) | Must have |
| 2.2.5 | Device should provide a widespread method to connect to an internet enabled device (Bluetooth) | Must have |
| 2.2.6 | Device must work over a reasonable range between it and internet enabled device | Must have |
| 2.2.7 | Device must serve notifications in a timely manner (>10s) | Must have |

Table

## Database

Requirements for the database are as follows –

### Functional

|  |  |  |
| --- | --- | --- |
| Requirement # | Requirement description | Category |
| 1.3.1 | Database must store users powerband contact details | Must have |
| 1.3.2 | Database must be accessible remotely | Must have |
| 1.3.3 | Database must have security against unauthorized access | Must have |
| 1.3.4 | Database must store a unique value for each user | Must have |

Table

### Non-Functional

|  |  |  |
| --- | --- | --- |
| Requirement # | Requirement description | Category |
| 2.3.1 | Database must run on common hardware | Must have |
| 2.3.2 | Database must run without huge computational/power draw | Must have |
| 2.3.3 | Database must use a compatible interface language | Must have |
| 2.3.4 | Database must update in a timely manner (>10s) | Must have |

Table

# Methodology

The methodology being used for this project is the Agile method. This will allow us to change features and design considerations throughout the process.

Two methodologies were considered for the project, Waterfall and Agile. Though for a single person project with limited scope the waterfall method may work, it does not allow for additions or revisions to the end product without significant time delays. It also introduces the potential for scope creep turning the project into an impossible task as features are proposed during any resets in the design process.

Using an Agile methodology allows us to expand on features continuously throughout the project while not suffering from the scope creep issues of Waterfall – this is accomplished using the previously mentioned MoSCoW method. This allows new features to be proposed and logged without requiring them to be in the final product while also giving a list for any new iterations or revisions to follow – saving on future development time.

# Design

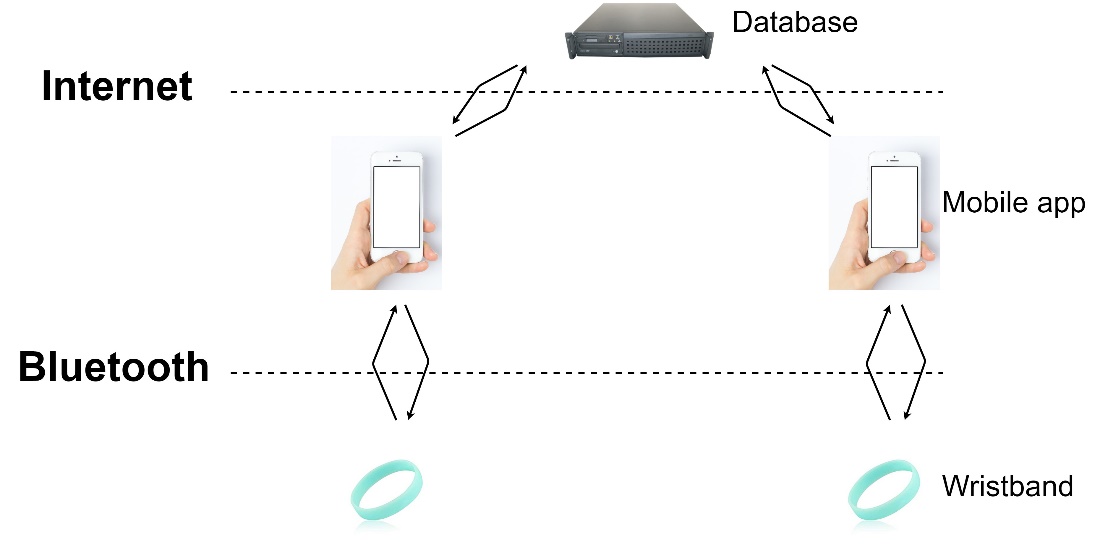
In this section we will discuss design considerations as well as outline how the system will work as a whole.

## High level design

At the highest level our system is made from 3 components –

* A mobile phone app (in our case android)
* A physical wristband
* A database

These components will use the most ubiquitous communications methods between each other,  
so in our case communication between the wristband and the mobile phone device will be via Bluetooth LE as this is an exceptionally common standard and should keep compatibility at its maximum (and thus give the highest potential customer base).   
The communication between the mobile devices and the database will be handled by the internet so in this case we do not have to be concerned as to the exact method as this will be dealt with by the Android operating system and the internet itself provided a TCP/UDP connection is established.



Figure

#### Outline

The device being created is a small low-cost wearable for personal notifications.

Its purpose is to provide discreate notifications to a user with minimal setup and configuration.

It is intended to be used by children and adults with access to a smart phone but without any prior experience with such devices.

#### Design goals

* Be easy to use
* Require no prior knowledge to use
* Provide discreate notification for local or remote matters using a vibrate motor
* Be energy efficient to run long term from an embedded battery

### Architecture

For this project research has been performed as above into potential competitor’s devices with the aim to show any pitfalls in similar designs and to allow the final product to outperform other similar devices on the market.

#### Data

This product will need to store some customer data and as such some considerations must be made as to what should be stored, where it should be stored and how that information is secured.

##### What is stored

Data to be stored should be kept to a minimum, the minimum that can be stored for this system to work would consist of the following –

* Users name

This will be taken when the user signs up and stored entirely on the database making it harder for an attacker to acquire.

(in future iterations this would be secured with a one-way hash function so that the data is never stored on either device or database in an easily readable way)

* Users unique identification

This should be a unique number generated by the database and given to the user, this will allow multiple people of the same name to use the system and cut down on data being sent and received.

* The wristband status

This is a Boolean value that shows if the user’s device is/should be triggered.

In further and expanded iterations of this project more details could be stored in further fields of the database for the purpose of creating an interface from other systems – such as a web interface or a method to mass notify people at an event or similar.

##### How it is stored

The physical device will store no user information.   
The only unique information on the device will be the BD\_ADDR, this is a 48bit address commonly displayed as a 12-digit hexadecimal number. The first 24bits of this address is a organization unique identifier (OUI), which identifies the manufacturer and the last 24bits are a unique number(JIMBLOM, 2020). This information should not be modifiable.

The android app stores only the user’s unique identification number as this is what it would use to look up its respective band’s status. As this number alone gives no sensitive information its protection is not essential. However, the number could be used to intercept another user’s notification and so some form of encryption may be used. The app also contains the password for connection to the database and this should be protected so that an attacker cannot read the contents of the database.

The database stores user’s names, unique identifiers and the statuses of their respective bands and as such care must be made to make this inaccessible outside of the confines of the app. This may be achieved using password security built into MySQL but may be further strengthened by other methods.

#### Communication

This project will use two instances of communication to function, the first being between the device and the smart phone, and the second between the smartphone and the database.

##### Device – smartphone

This will be handled by Bluetooth low energy and should rely on the GATT service feature (Generic ATTributes). These are a collection of ‘services’ and ‘characteristics’ that allow communication between devices (Bluetooth SIG, 2020).

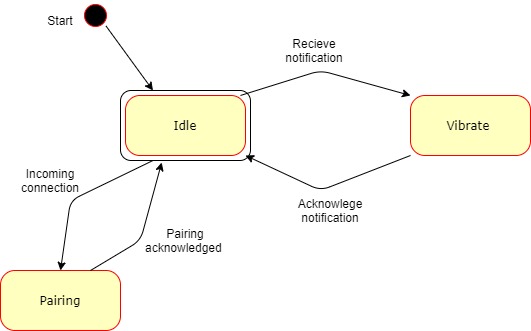
##### Smartphone – database

This will be handled by android using either Wi-Fi or mobile internet to connect to the internet. Messages will be sent via HTTP invoking PHP scripts on the database server. These scripts will be what directly communicates with the database.

#### Finite state machine

This FSM diagram provides a high-level visual representation of the states of a functioning system and the transitions between those states.

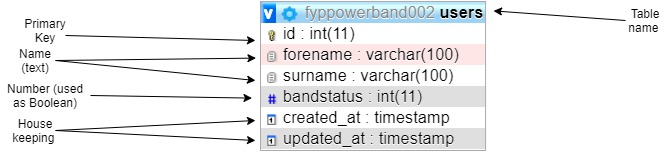
##### The band –



Figure

##### The app –

#### Database diagram



Figure

#### Modules

The app is comprised of various system modules with specific functions, this choice has been made to make updates and upgrades easier in the future as a component can be replaced without needing to rewrite the entire application.

* + - Main application
      * Shows menus and functions of the app to the end user
      * Handles user input via android
    - JSON parser
      * Handles compilation of data for database requests
      * Compiles HTTP requests
      * Sends commands via HTTP
    - Notification handler
      * Handles Bluetooth requests
      * Deals with GATT services

# Implementation

## Prerequisite’s

Before attempting this project, multiple systems were required to be in place, many of these systems relied on each other to function. In order to accomplish this, I did the following –

## Platform selection

Below are several chips and developments boards considered for the project plus a short description of their pro’s and con’s and suitability for the task at hand.

Note prices are taken from <https://uk.farnell.com> on the 23/2/2019 unless otherwise specified.

### CC2640R2F

This is an ultra-low power microcontroller (cortex m3) and RF system on a single chip, it is designed specifically to be capable of running on a coin cell battery or via an energy harvesting device (Texas Instruments, 2019). The wireless capabilities include Bluetooth 4.3 and Bluetooth 5le connected to an ultra-low power sensor controller (based on a cortex m0) this is designed to run independently of the main cortex m3 processor allowing for extremely low power consumption while the main processor is in sleep states.  
The cost of this chip is approximately £4.07 per chip and the development kit (Launchpad) is £23.55 and is available immediately.

* Wide Supply Voltage Range
  + Normal Operation: 1.8 to 3.8 V
  + External Regulator Mode: 1.7 to 1.95 V
* Active-Mode RX: 5.9 mA
* Active-Mode TX at 0 dBm: 6.1 mA
* Active-Mode TX at +5 dBm: 9.1 mA
* Active-Mode MCU: 61 µA/MHz
* Active-Mode MCU: 48.5 CoreMark/mA
* Active-Mode Sensor Controller: 0.4mA + 8.2 µA/MHz
* Standby: 1.1 µA (RTC Running and RAM/CPU Retention)
* Shutdown: 100 nA (Wake Up on External Events)

### NXP-QN908X

QN908x is an ultra-low-power, high-performance and highly integrated Bluetooth Low Energy solution for Bluetooth® Smart applications such as sports and fitness, human interface devices, and app-enabled smart accessories. It is specially designed for wearable electronics with a small capacity battery.(NXP products, 2019). This chip is based around the Cortex®-M4F microcontroller and is capable of providing floating point operations as denoted by the ‘F’ postfix and provides Bluetooth Low Energy support.   
This chip is based on a single CPU design and as such requires its main CPU to be active in order to use its Bluetooth functionality.  
The cost of this chip is approximately £4.52 and the development platform (QN9080-DK) is £88.93 but is currently out of stock.

* Single 1.8 V ~ 3.6 V power supply
* 1 µA deep sleep mode
* 2 µA sleep mode (32-kHz OSC/RTC on)
* 3.5 mA RX current with DC-DC at 3 V supply
* 3.5 mA TX current @0dBm Tx power with DC-DC at 3 V supply

### NXP-QN902X

QN902x is an ultra-low power, high-performance and highly integrated Bluetooth LE solution. It is used in Bluetooth Smart applications such as sports and fitness, human interface devices, and app-enabled smart accessories. It is specially designed for wearable electronics and can run on a small capacity battery such as a coin cell battery(NXP products, 2019).   
This MCU is designed around Cortex® M0 design and also provides Bluetooth Low Energy support.   
The cost of this chip is approximately £3.95 and the development platform (QN9020DKUL) is £37.40 but is again unavailable at the time of research.

* Single power supply of 2.4 V to 3.6 V for QN9020/1
* Single power supply of 1.8 V to 3.6 V for QN9022
* Integrated DC-to-DC converter and LDO
* 2 μA deep sleep mode
* 3 μA sleep mode (32 kHz RC oscillator on)
* 9.25 mA RX current with DC-to-DC converter
* 8.8 mA TX current @0 dBm TX power with DC-to-DC converter

### Comparison

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Chip | Architecture | Power Consumption | RF support | Cost for chip | Cost for development |
| CC2640R2F | Cortex® M3 + Cortex® M0 components |  | Bluetooth 4.3, Bluetooth 5 low energy | £4.07 | £23.55 |
| NXP-QN908X | Cortex®-M4F |  | Bluetooth 5 low energy | £4.52 | £88.93 |
| NXP-QN902X | Cortex®-M0 |  | Bluetooth 5 low energy | £3.95 | £37.40 |

Table

### Hardware decision

The decision for what MCU to use was almost made for us due to time/availability constraints, with the only device both available and at a reasonable cost being the TI CC2640R2F. In actuality the NXP-QN902X may have been the better choice as its simpler design based on a Cortex M0 is likely to yield minor power efficiency advantages however the larger costs of the development kit and at the time of research being unavailable made the CC2640R2F the better choice. The NXP-QN908X was largely ruled out due to its very high development cost, also being based on a Cortex M4 means its computational power is fay beyond what is required – its inclusion of a floating-point math coprocessor as denoted by the ‘F’ part of Cortex M4F is also an addition that would be unused in this project.  
An advantage of using the TI CC2640R2F allows for usage of Texas Instruments own Eclipse based ‘Code Composer Studio`.

#### CC2640R2FRSMT MCU

The hardware will be based around the CC2640R2FRSMT MCU and developed on the relevant ‘Launchpad’ development board for this chip and using technical data from the ‘CC13x0, CC26x0 SimpleLink™ Wireless MCU Technical Reference Manual’ (Texas Instruments, 2017). Once development is completed using the Launchpad development device a cost reduction will be performed. This will involve installing the same MCU onto a bespoke designed PCB that will be considerably smaller than the launchpad and contain no extraneous devices for programming and debugging as are found on development boards.

The final hardware should consist of –

* The MCU
* PCB
* Ancillary components including –
  + Resistors
  + Capacitors
  + 24Mhz Crystal

Note these components are found on page 43 of the Texas Instruments datasheet ‘swrs204b.pdf’ as found on their website.

Further attention will need to be paid as some items in this document are described as components when they are in fact resistance/capacitance of tracks on the PCB, specifically those related to the Bluetooth antenna.

## Software usage

In this section we will outline the software environments used the reasons for their selection and give an outline of how each is setup -

### Virtual machine

The CC2640R2F launchpad development board by Texas Instruments is recommended to be programmed via the ‘Code Composer Studio’ IDE. This was already being used with another project so as to stop cross contamination between projects and to enable work from multiple places we created a Windows 10 virtual machine on an external SSD. The SSD running over USB 3.0 enabled the Virtual machine to run at near native performance compared to a USB 2.0 Hard drive or Memory stick.

The virtual environment used is a blank version of Windows 10 Home (the 128GB ram limit on home versions is not an issue with the hardware available) running on VMware Workstation Pro. This environment was then used on 3 machines and forwarded the following resources –

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Machine | CPU available | CPU allocated | Ram available | Ram allocated |
| CLEVO N850HK1 | Intel I7 7700HQ  2.8-3.8 GHz  4 cores  8 threads | 4 virtual cores | 8GB | 4GB |
| Custom Desktop | AMD 3900x  3.8-4.6 GHz  12 cores  24 threads | 16 virtual cores | 16GB | 12GB |
| UWE HP Desktop | Intel I5 7500  3.4-3.8 GHz  4 cores  4 threads | 4 virtual cores | 16GB | 8GB |

Table

The usage of more Ram provided a better experience while programming – faster switching between applications, however more cores allowed for considerably faster compilation of larger projects.

### Software tools and development environment

Inside the virtual machine various tools were installed for development, the main tools used are listed as follows –

* Notepad++
* CodeComposerStudio 9.0.1 (CCS)
* Android studio 3.6.1
* SimpleLink™ CC2640R2 SDK - Bluetooth® low energy v3.30

Additionally, outside of the virtual machine, either on another machine on the same network (for ease of security) or occasionally in the host machine for the VM the following tool was used to provide a remote MySQL database –

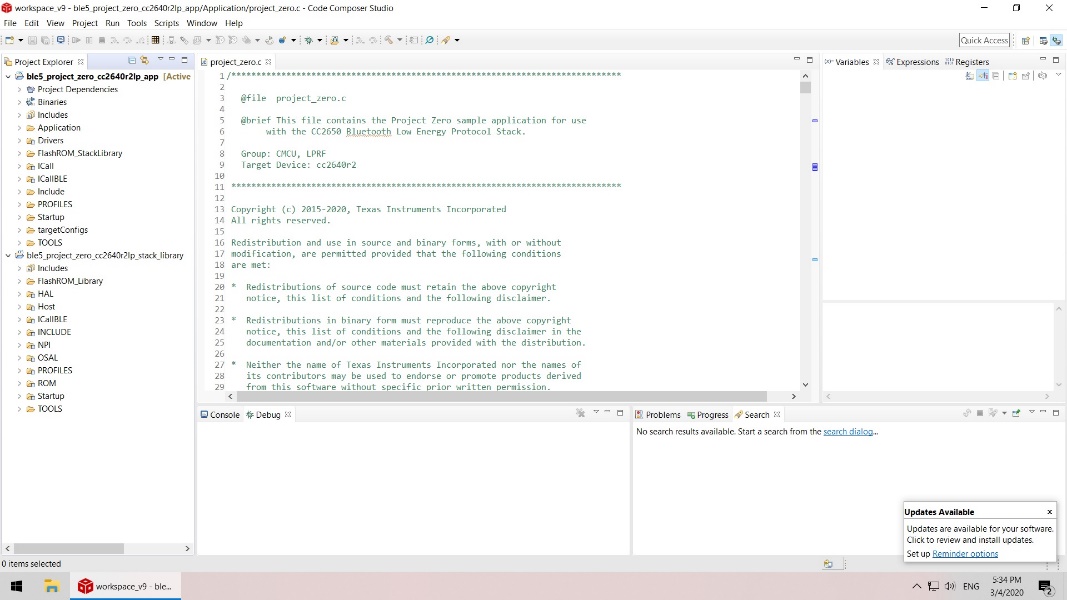
* WampServer 3.2.0

During the development cycle several of these tools were updated as will be discussed further on.

### First steps, setup and hardware testing

Before starting development in earnest some testing was done to check that the hardware was working correctly so that testing of my own code and general troubleshooting would not be futile. This involved downloading the demo application from the CC2640R2 SDK on to the development board and connecting too it with the TI Simplelink starter application (available [here](https://play.google.com/store/apps/details?id=com.ti.ble.simplelinkstarter&hl=en_GB)).

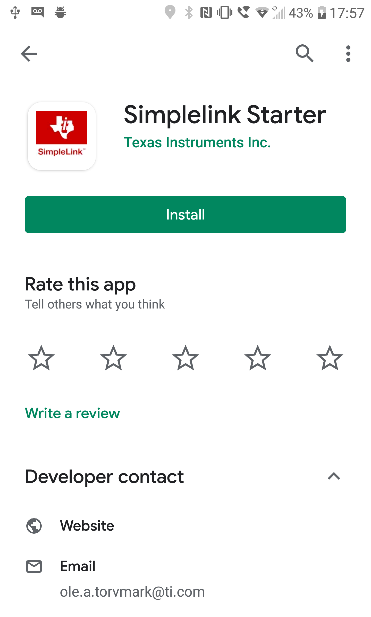
Upon attempting to download this as per the ‘Getting started’ section of the TI website we came upon the first hurdle, the links in the ‘Resource Explorer’ section of CCS (this is essentially a web browser linked directly to TI’s support pages) no longer linked correctly to its web servers and gave various connection/404 errors. To circumvent this a regular browser could be used as well as a combination of Google searches and using the search form on the TI website, however this would often serve archived results that were no longer relevant to this version of the SDK or CCS or serve results relevant to other development boards – frequently the MSP430. This meant great care had to be used when vetting the usefulness of these results. Originally the demo software was available as a separate download, but this has since been integrated into the SDK. After acquiring the correct information several dependencies needed to be loaded into CCS in order to make Project Zero compile, these were mostly related to the BluetoothLowEnergyStackHandler and TIRTOS, the latter of which had updates giving issues with its integration, luckily an update was released for the SDK containing all the required components and dependencies giving a project that looked like the image below.



Figure

After resolving this issue, the project could be loaded up in CCS easily complete with its single dependency ble5\_project\_zero\_cc2640r2lp\_stack\_library that is included in the SDK files.

Next on an Android device with access to the play store the Simplelink software should be downloaded and installed giving the results as per the following images –



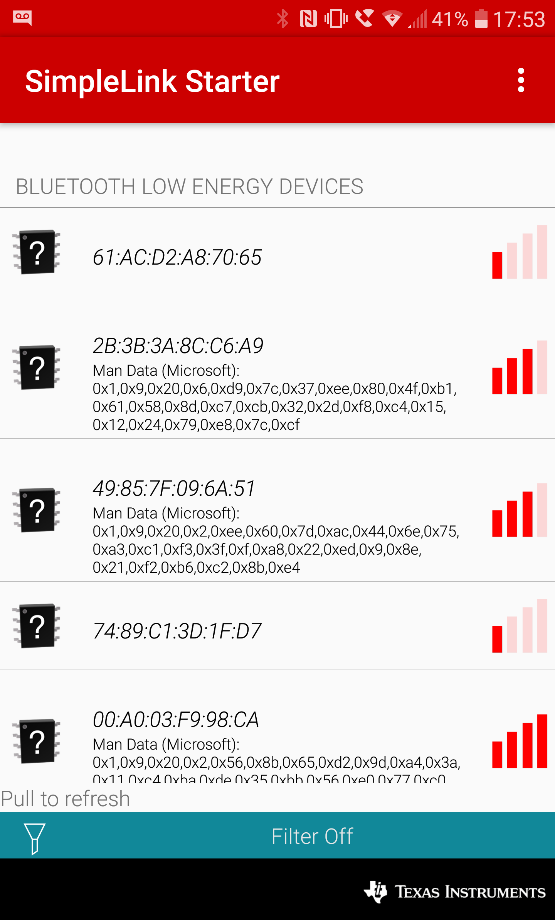
Figure

Installation of this will provide an entry into the main app drawer looking like the following –



Figure

Opening this application will give you the following provided that Bluetooth is enabled on the device–



Figure

Upon deploying projectzero to the board using the deployment buttons at the top of the IDE the board will start broadcasting its name, this can be scanned for in the simplelink app.

INSERT SCREENSHOT

Once connected the advertised GATT services can be seen for the LED’s and Button’s.

INSERT SCREENSHOT

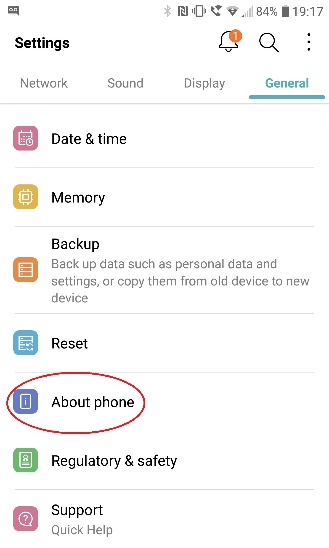
The status of these items can either be see physically on the board as with the LED’s or read in the app with the buttons.

INSERT DOUBLE SCREENSHOT HERE

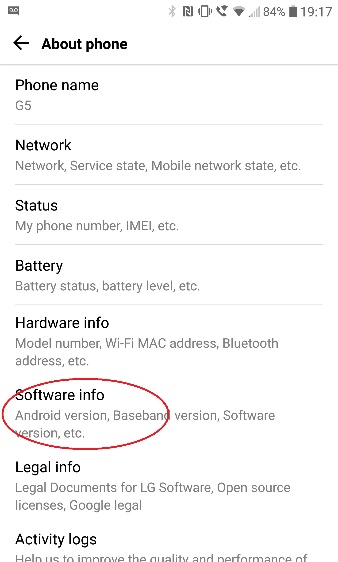
Next a basic application was written in Android studio to test its ability to be deployed to a device and that its debugging tools are available for use. In this instance a tutorial on database usage was followed in order to achieve multiple targets in one application(Tamada, 2017).

In order to make the debugging tools work a development enabled version of android must be used. These features can be enabled in most android devices.

This is accomplished by going into settings – about device – and to software info

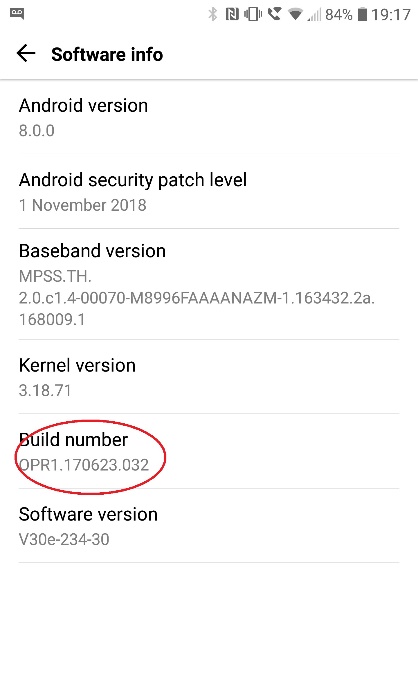


Figure



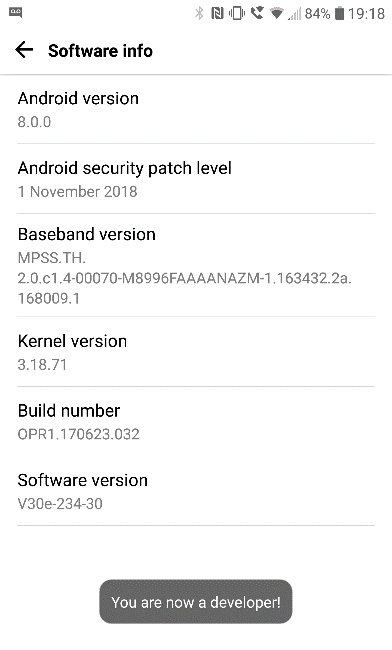
Figure

And pressing repeatedly on the Build number

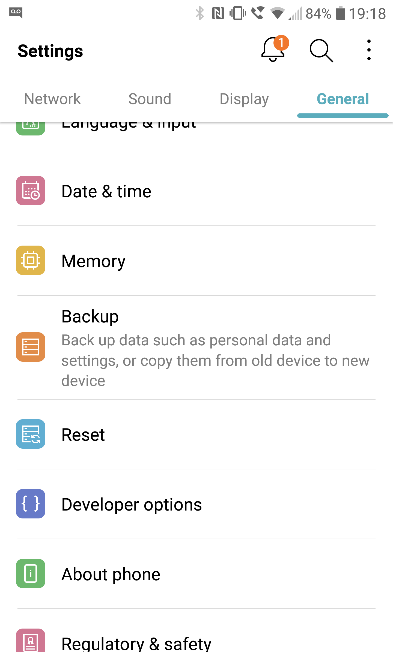


Figure

If performed correctly a countdown should begin with each press resulting in a message confirming you to be a developer and giving a new subcategory in settings for development tools.

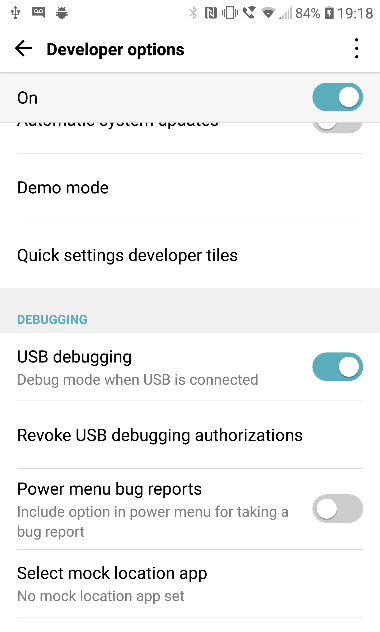


Figure



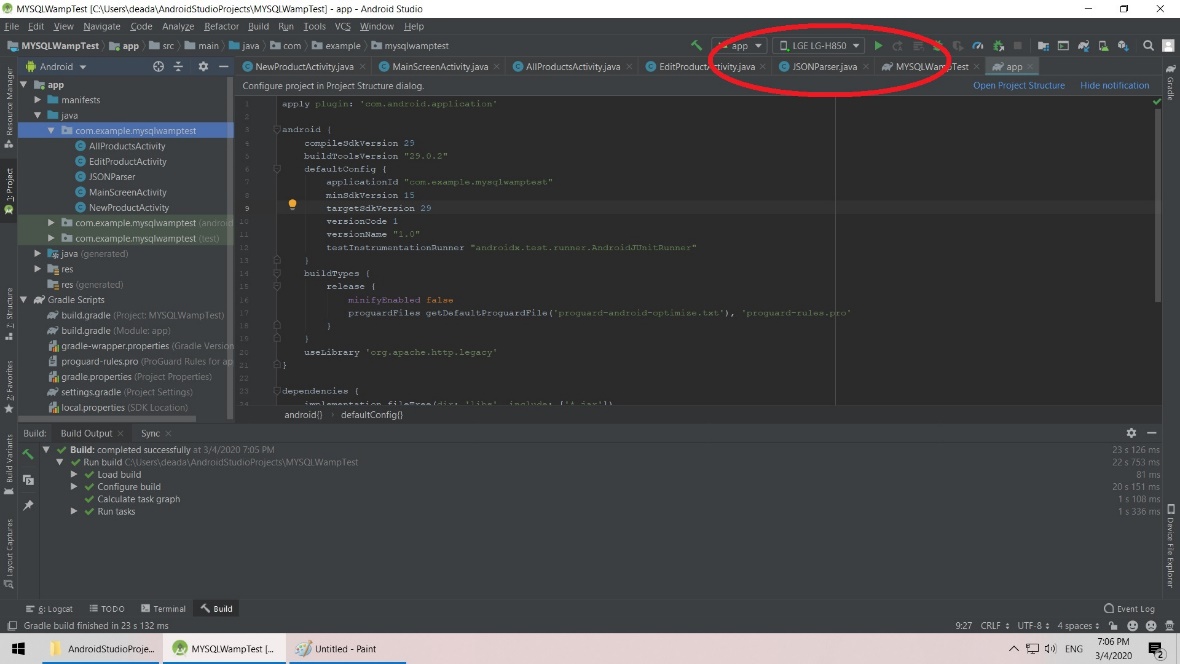
Figure

In the Developer options menu, the function ‘USB debugging’ needs to be enabled



Figure

Once this is done the device can be connected to a machine running Android Studio and should appear as a deployable test device.

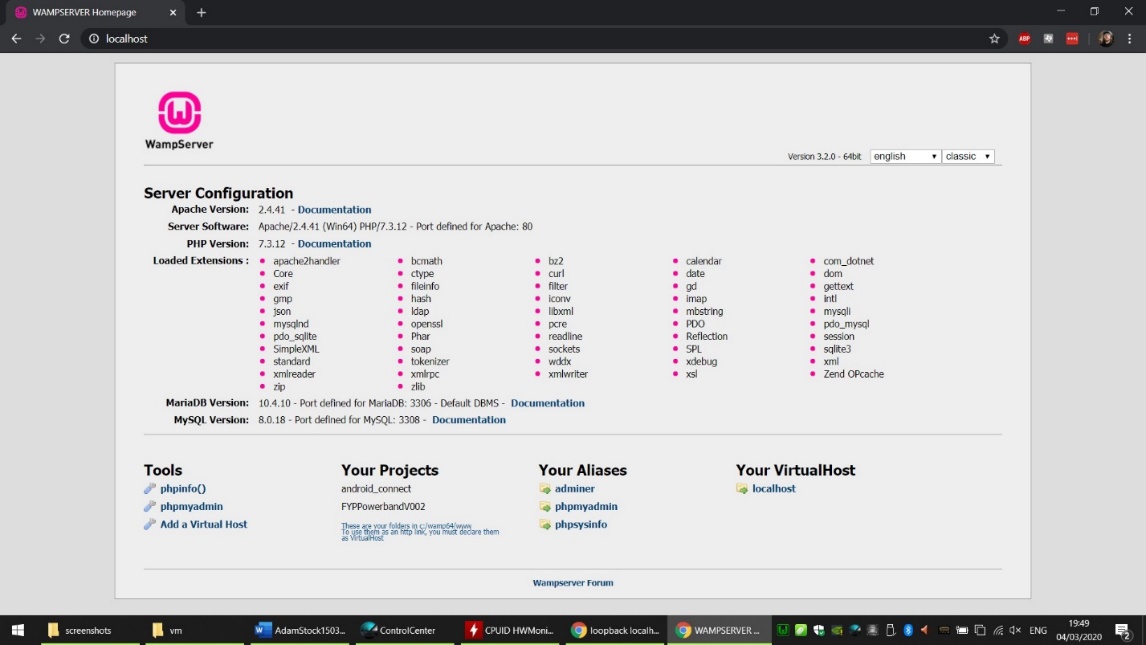


Figure

For purposes of testing the IDE and its debugging tools this worked with the application being deployed to a test device (LG G5 mobile phone).

## Database

Next a MySQL server was created using the WampServer software stack available here. In order to achieve this the software should be downloaded onto a PC on the same network as the target. Once installed opening a browser and directing it to the Localhost via the loopback function – commonly found by using the IP address 127.0.0.1 will bring you to the main controls for Wamp.



Figure

From this the phpMyAdmin link was pressed, this asked us to log in and select what type of database is to be used, in our case we will use MySQL rather than MariaDB and the default username and password combination. At this point it is worth checking that both the computer and mobile device being used are on the same subnet and to edit the httpd.conf file in Wamp to enable connections from other IP addresses. This is accomplished by editing the listen lines (in this case line 70-71) to the following –

Listen 0.0.0.0:80

Listen [::0]:80

This will allow connections from port 80 from other IP addresses. Doing this at this stage will significantly cut down troubleshooting later.

After doing this the Wampserver should be restarted before proceeding.

From phpMyAdmin the ‘databases’ link should be clicked, and a new database created. For this project a database called fyppowerbandXXX will be used (the XXX to be replaced by a version number). In this database a table called users is created with the fields –

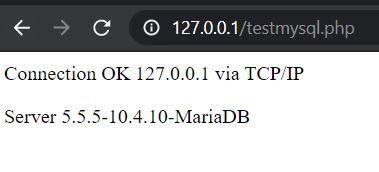
|  |  |
| --- | --- |
| ID | This is the primary key and as such is also the unique number for each user |
| Forename | User forename |
| Surname | User surname |
| Bandstatus | This is a number to be used as a Boolean to enable/disable the band |
| Created\_at | For testing |
| Updated\_at | For testing |

Table

Once this is created the ability to connect to it should be tested, to do this a php file included with Wamp is used. This is accomplished by typing the following into a web browser–

IP ADDRESS OF WAMP/testmysql.php

And should provide this output –



Figure

Note the database type is not relevant here as this just shows the network is allowing communication.

At this point we should have a working Database…… that we have no way of interacting with.

### PHP Scripts

In order to be able to interact with the database we will invoke scripts written in PHP. PHP is a recursive acronym for - **P**hp: **H**ypertext **P**reprocessor. This language is used as it is relatively easy to embed into HTML and pass arguments too.

A separate script will be written for each function we want to achieve, the primary functions we need are as follows –

* Create a user
* Set a user’s band
* Unset a user’s band

Additional functions that may be useful include –

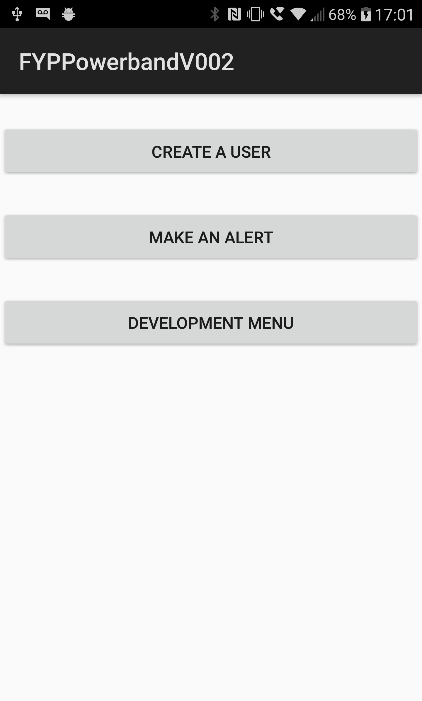
* Delete a user
* Search for a user’s details

All of these scripts are included in the appendices.

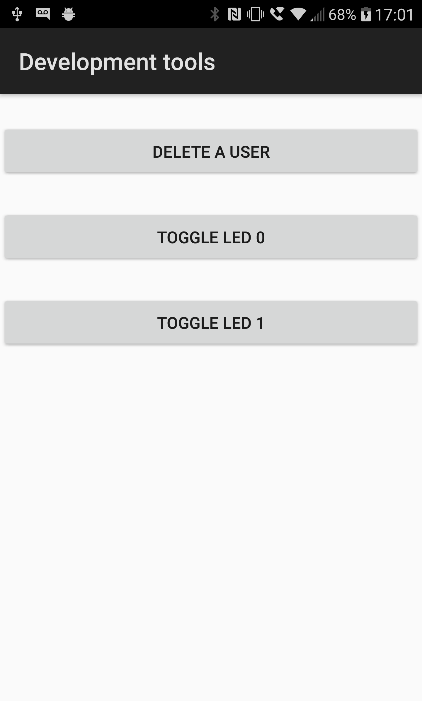
## Android App

Now that the background components are tested and working the initial iteration of the Android application can be created. From following the aforementioned tutorial, we found several of its functions to be deprecated, luckily this was found when completing the tutorial and did not require troubleshooting once the main applications was being created.

To begin, as with most android apps we started with a main page, the code for this is in the class ‘MainScreenActivity.java’ and its respective layout ‘main\_screen.xml’. This page contains the buttons for the main features of the app – primarily ‘Create a User account‘ and ‘Make an alert’ , as this is a prototype application we also included a button for a development menu giving additional options to deleted a user account and to trigger the band manually (note there are two buttons available as the development board has two easily triggered services connected to LED’s). Screenshots of these are provided below in figures 20 and 21(note that style and placement of buttons maybe subject to changes as development continues).



Figure

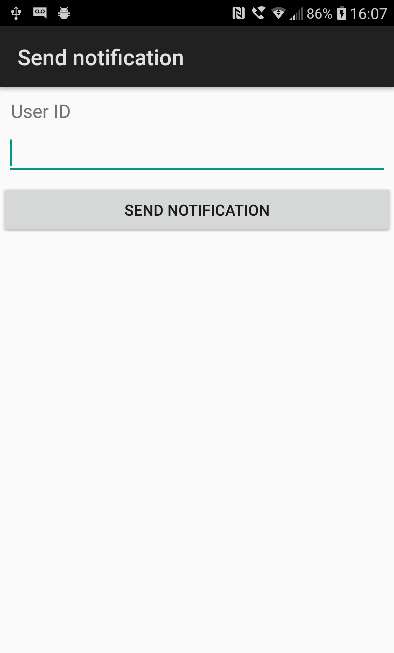


Figure

When selecting to create a user another class will be invoked called ‘create\_user.java’ and its respective xml file. This page contains text entry for the users details and a button to start the process.

Upon pressing the button, the text inside the entry fields is converted into strings and passed to the create\_new\_user class to continue in the background. This displays a message and then invokes the JSONparser class in the background with the relevant arguments to post. The JSONparser creates an http request containing the details of the php script and the arguments it requires.

When pressing the ‘Make an alert’ button a similar process is invoked where a text entry box allows the entry of a user’s ID, pressing the ‘Send notification’ button again invokes the JSONparser class creating the correct http request. In this instance only the ID is required as the Boolean state of the band is coded into the php script.



Figure

The last button on the main screen is that of the ‘Development menu’ this is a submenu containing features unlikely to be used directly in the final product but provide useful functions during development.   
  
INSERT PIC HERE

The first button is programmatically similar to ‘Make an alert’ in that it contains a text box, the contents of which are sent to the relevant php URL deleting the selected user.

The final two buttons are to directly control two output lines on the wristband, these are connected to LED’s on the development board.

First attempts for this function were created by looking at the .APK file created by Texas Instruments as a test for the development board, this was accomplished by decompilation using an online tool.   
Android applications use the Java language and so are not strictly compiled or interpreted. When the program is ‘built’ in android studio its class files are compiled into bytecode which is neither machine nor native code. When the code is run a Java Virtual Machine interprets the bytecode into a usable program with components of it compiled into native code as required using just in time compilation.

The result of this is that the code is possible to deconstruct to allow insight into its inner workings.

While this code was only used as an inspiration for initial test code, it provided some useful insight into the workings of Bluetooth LE, namely the GATT system.

Bluetooth Low Energy (BLE for short) is the low power variant of the Bluetooth PAN radio system that uses Generic ATTribute profile’s (GATT) to communicate. The key concepts used in GATT are as follows-

Client – this begins communications, sends requests and receives the responses. In our case this role will be played by the Android device.

Server- this receives requests and returns responses. In our case played by the wristband.

Characteristic- this is a value stored on the server that can potentially be read or written too.

Service- this is a collection of characteristics that are all accessed via the same method, this may also contain descriptors to enable easier comprehension of the stored characteristics.

Grouped together services and characteristics are referred to as attributes and are identified via Universally Unique IDentifiers (UUID). Deconstructing the APK as mentioned earlier allows us to see what these identifiers are, though this could also be accomplished using an android app that scans for BLE services.

The UUID’s we are concerned about are the following –

f0001110-0451-4000-b000-000000000000 – the LED service

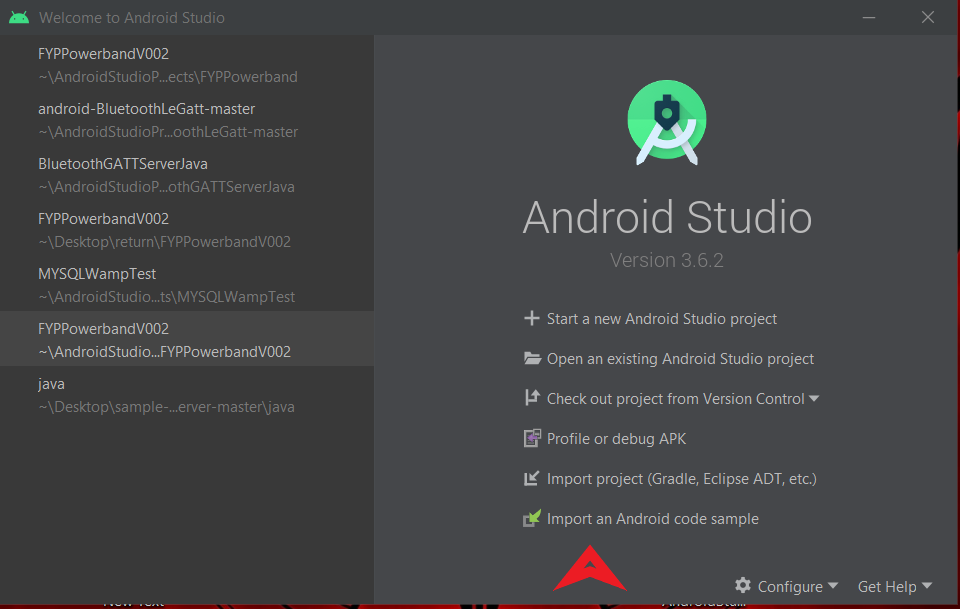
f0001111-0451-4000-b000-000000000000 – LED 1

f0001112-0451-4000-b000-000000000000 – LED 2

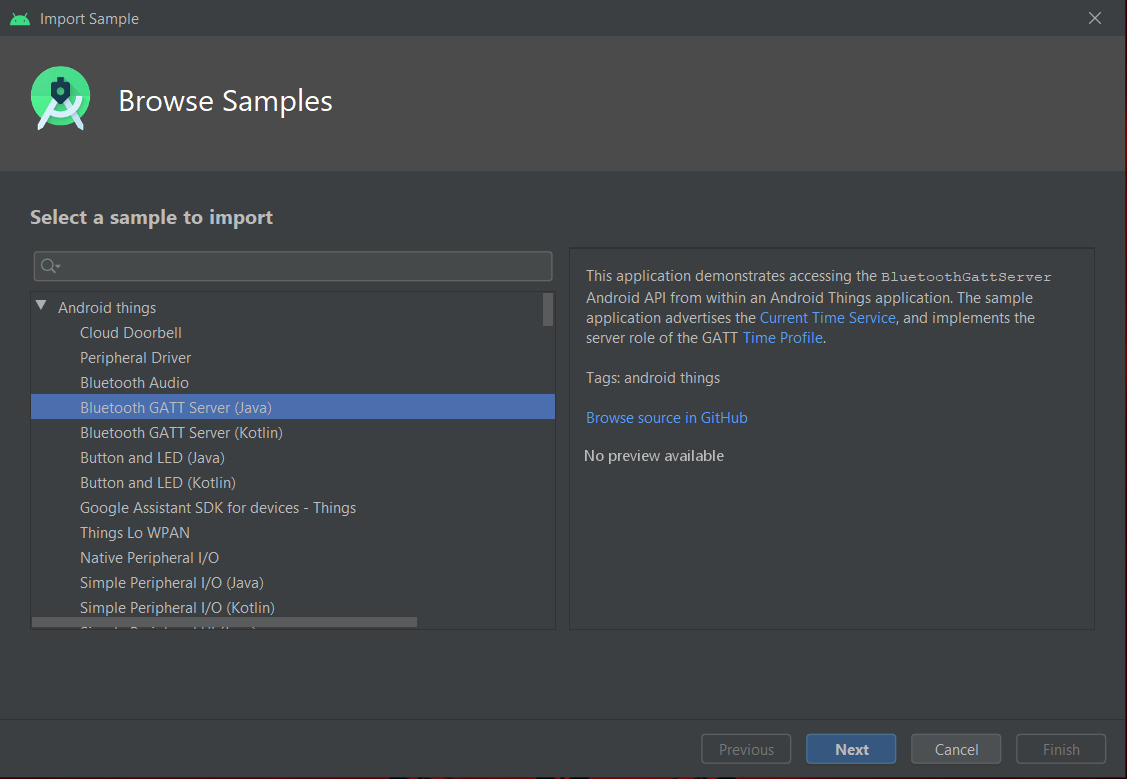
Ultimately attempting to make an entirely custom interface to the BLE services proved fruitless. This was due to the program being analyzed having many features, and therefore code and dependencies that are not required for our purpose and it would likely have taken longer to cull and reverse engineer this than starting over.

At this stage we ran into one of a number of issues with our working environment. The main issue witnessed involved an update to Android Studio to version #AI-192.7142.36.36.6308749 and its respective Gradle update. This caused the debug system to break as the bytecode provided in the SDK’s for android no longer matched that created when building the program with its dependencies. Upon research this seems to be a common issue in the IntelliJ platform by JetBrains. This issue was fixed by reverting back to a previous version and was likely fixed entirely in a further update as these errors no longer seem to arise.   
From this issue we would recommend not updating the development environment during development unless the update is a required security update or provides needed functionality.

Our next attempt we based our design on that provided by the Android developers with examples provided through the android studio itself and is downloaded from GitHub. Note the images show the Gatt server but clients can be found in the Googlearchives and androidthings sections of GitHub.



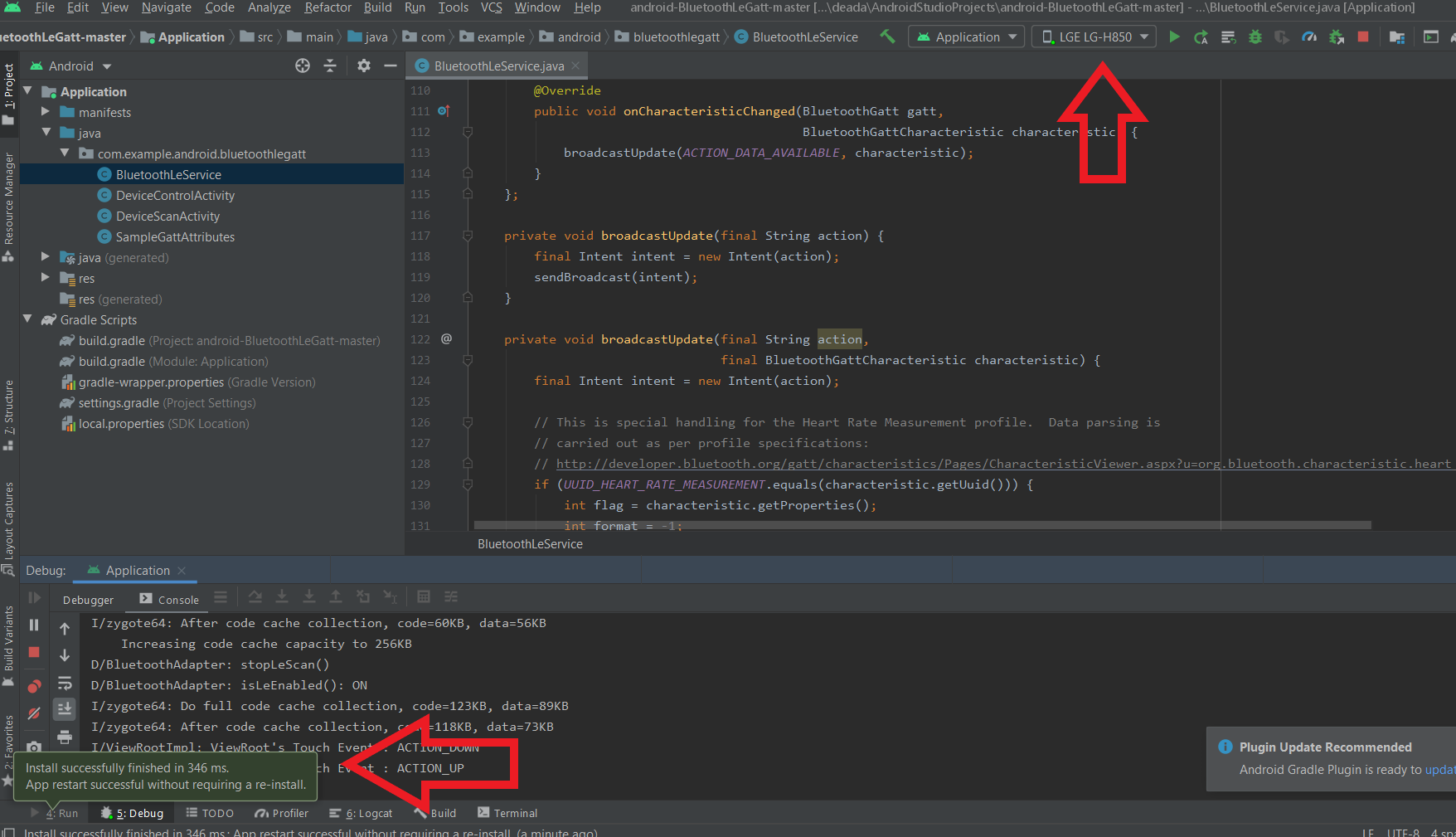
Figure



Figure

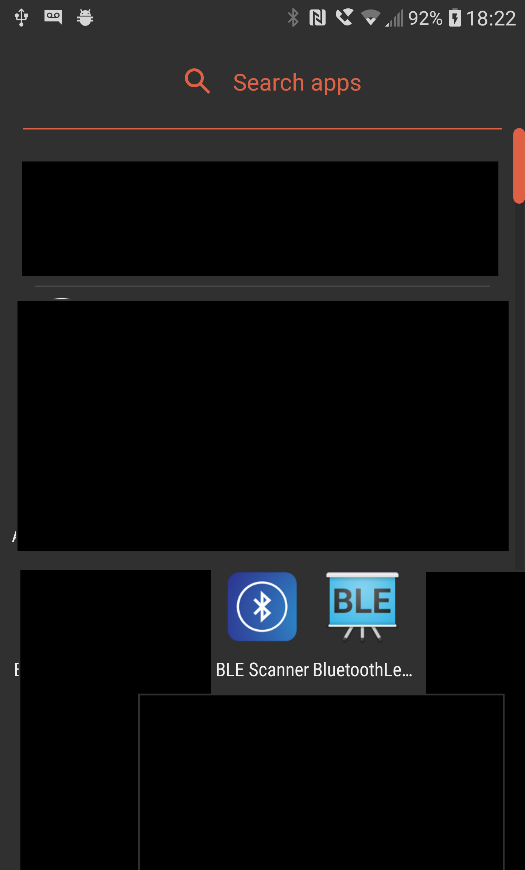
However, this is where problems arose. The code provided in studio will not build and deploy to an android version of 8.1(API 27) or lower, this causes an issue in that relatively few android devices use this version as it is quite new, if nothing else using this code could potentially limit our customer base greatly as many would simply not be able to install the software. This issue is compounded further by the fact the release date of the code to GitHub considerably predates the release of version 8.1 and that the code mentioned on the readme.md file of GitHub refers mostly to android version Android 4.3 (API 18).

With some tweaking of the manifest file this code can be made to work on our test device running version 8.0 however this is not where the issues end.



Figure

This app builds and deploys to a device with no issues as well as debug hooks working allowing us to step through code as it runs. Visually everything seems to work fine on the system, and it scans for devices.



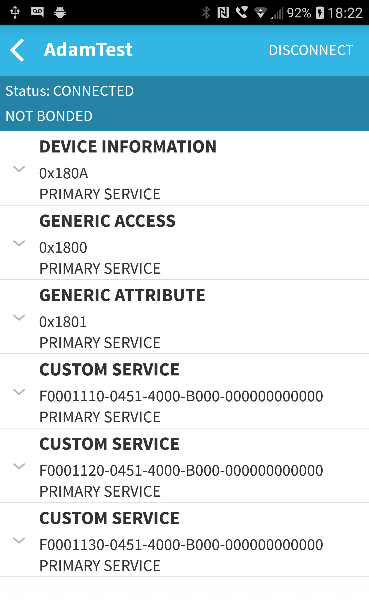
Figure



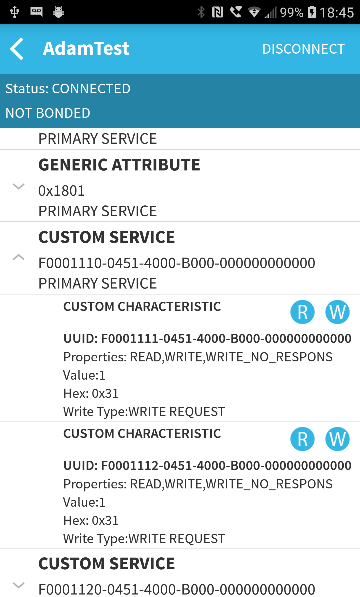
Figure

The scanning process however yields no results. This is unexpected as no errors are generated in the debug logs outside of saying the BLE is unavailable if one forgets to turn Bluetooth on on the device beforehand (on the device a prompt is displayed requesting this to be switched on).

To test other factors a 3rd party application was downloaded from the android store called BLE Scanner this scans for Bluetooth devices and gives information such as signal strength, Bluetooth standard and proximity as well as having a Gatt client showing us available services and allowing us to read and write to them as appropriate. This application has no issues seeing the device (in this instance called ‘AdamTest’), seeing the available services and letting us read or write to the characteristics at will.



Figure



Figure

This shows that our devices and setup are correct and highlights a potential issue in the development documentation provided by Google.

The remaining component required by this application to be feature complete once the above challenge has been overcome is a service that does the following in order –

1. Checks the database for the band status related to the user’s ID periodically
2. If this is 0 do nothing, or if 1 continue
3. Check and enable Bluetooth
4. Read the Gatt characteristic – this should be 0
5. Write 1 to the characteristic
6. Read the Gatt characteristic – this should be 1
7. Write 0 to the user’s band status
8. Wait a predetermined amount of time (several seconds so the user has time to notice the band)
9. Write 0 to the characteristic
10. Read the Gatt characteristic – this should be 0

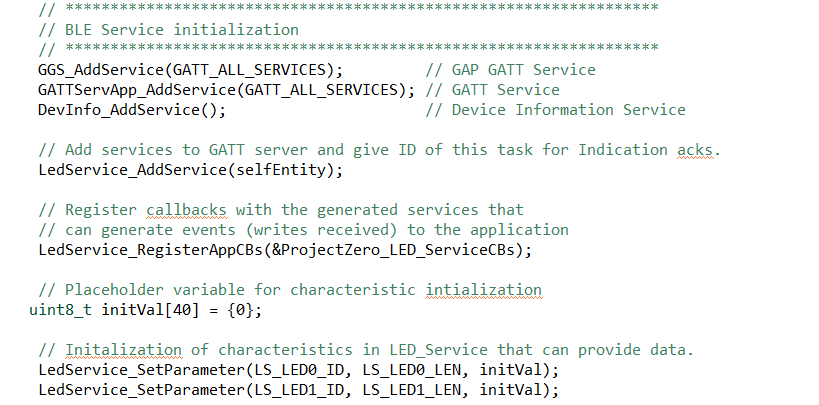
## Wristband

Next we move to the code for the embedded system based on the TI CC2640R2 microcontroller.

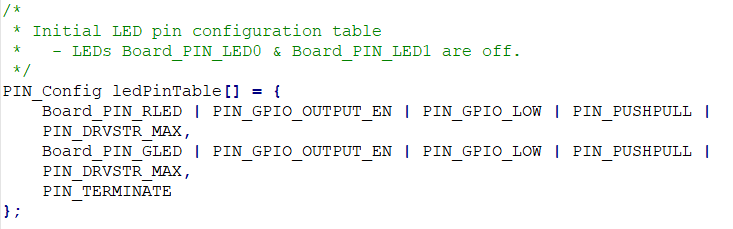
This component is based on a modified version of the example code provided by Texas Instruments called project zero mentioned previously. This code provides an operating system and interfaces to the hardware components of the board such as LED’s and buttons. The code as provided contains many features outside of the scope of this project and these would take up memory and potentially energy if included.

After studying the provided code and understanding its contents, functions that were outside of the scope of the project were removed including some background functions such as the ability to debounce switches.

This can be seen in the BLE Service initialization section where services are added, and ID’s given for services that will be visible on the GATT server, in our case we only need the single service called LedService (please note that it is titled this way as on the development board it is connected to an LED, in the final product this would be replaced with a transistor to enable control of the higher current of the vibrate motor).



The pins used by the program could be changed easily changed if a specific pin is better located by referring to the LED pin configuration table.



The two main features removed were as followed -

Button inputs including its relevant pin table, clock handlers for counting time during its debouncing, and its defined messages that would be sent to the main application task and its callback functions.

The DataService, this would provide a service in the GATT that when read would return a string of text. This service contains no pin table or ancillary functions however it does require callbacks which have been removed and can be used to store text on our device which will take up large amounts of memory without providing any useful features to the end user.

One feature that was not required that was left in intentionally was Uartlog (#include <uartlog/UartLog.h>) and its ancillaries as this allows us to pass information to the UART output by means of the log\_info code. This feature enables us to debug code via two methods, the debug features inside CCS, and by reading the serial data returned by the XDS110 UART included in the development board with a program such as Teraterm. The former can cause issues when dealing with things such as Bluetooth as stopping at break points can prevent time sensitive code from completing as intended and thus preventing connections. The latter does not interfere with the code running on the device and can be considered analogous to printing debug statements to a screen. A subsection to this feature that should also be removed in further iterations is that of the ansicolour header include (#include <ti/display/AnsiColor.h>) as this adds a table to add colour to the characters outputted and if used frequently could add significant bulk to the application.

# Evaluation

# Conclusion

# References and Bibliography

# Appendices

## Php scripts