SCHOOL OF ENGINEERING

Diploma in Biomedical Engineering (EGDF09)
Diploma in Cybersecurity & Forensics (EGDF09)
Diploma in Aerospace Systems & Management (EGDF09)

Smart Luggage Tag

Prepared By

Sushmitha D/O Ravichandran (163796L)

Tan Pei Si (163240C)

Yeo Chee En, Luke (161788D)

Project Supervisors

Mr. Steven Ng Wei Hsien

Mr Kannapan Iynkaran

Abstract

Baggage tags are used by all who travel by air and are an essential part of the check-in system in the airport. However, tons of resources such as paper, ink and adhesives are used in the creation of one. As there is an exponentially increasing number of air traffic volume with each day, this is environmentally unsustainable in the long run. Hence, an environmentally viable long-term solution would be to digitalise the baggage tag.

An additional problem faced when travelling, is the weight of the luggage. Most travelers often discover that their luggage is overweight at the airport itself, right before check-in. Thus, our solution comprises of 2 components:

1. Digitalised Baggage Tag
2. Weighing Scale

Acknowledgements

We would like to sincerely thank all the people who have helped us throughout the period of this project. Without all these people who have supported and given us their input for the project, we would never have made it this far into our project.

We would like to thank our supervisor, Mr Steven Ng for his continued, unwavering support through these arduous 5 months. We are extremely grateful to him for all the times he steered us onto the right direction when we required help from him; he always gave the best advice possible and was our reliable pillar of support throughout the project. Without his support, this project would never have been able to reach the state that it is in currently.

Secondly, we are grateful to Dr Yeoh Wui Keat for helping us with our research and programming, thus enabling us to attain a higher quality of work in a shorter period of time.

Thirdly, we are very thankful to our friends residing in the same lab as us for giving us valuable advice when it came to programming and for enthusiastically participating in the User Acceptance Testing phase of the project.

Lastly, we would like to thank the IMP committee for giving us the opportunity to work on this project and for equipping us with the knowledge required to properly plan and execute our project. The IMP Bootcamp itself was fundamental in allowing us to grow in terms of project planning and management.

Table of Contents

Background

Baggage tags have traditionally been used by bus, train, and airline carriers to route checked luggage to its destination. The passenger stub is typically handed to the passenger to be pasted on the baggage. It has 2 main uses:

1. Aid the passenger in identifying their bag among similar bags at the destination baggage carousel
2. Allows the passenger and carrier to identify and trace a specific bag that has gone astray and was not delivered at the destination.

Current bag tags include a bar code using the Interleaved 2 of 5 symbology. These bag tags are printed using a thermal or barcode printer on an adhesive thermal paper stock. This printed strip is then attached to the luggage at check-in, allowing automated sorting of the bags by bar code readers.

There are 2 ways that bar code baggage tags are read: hand held scanners, and in-line arrays. In-line arrays are built into the baggage conveyor system and use a 360-degree array of lasers to read the bar code tags from multiple angles because baggage and the orientation of the bar code tag can shift as the bag travels through the conveyor belt system.

One of the limitations of this system is that in order to read bar codes from the bottom of the belt, laser arrays are placed below the gap between two sections of conveyor belt. Due to the frequent build-up of debris and dust on these lower arrays, the rate of successful reads can be low.

Frequently, the "read rate", the percentage of bar code tags successfully read by these arrays, can be as low as 85%, equating to more than 1 out of 10 bar code baggage tags are unsuccessfully read, and these bags are shunted off for manual reading, resulting in extra labor and delay.

Bar codes cannot be automatically scanned without direct sight and undamaged print. Because of reading problems with poorly printed, obscured, crumpled, scored or otherwise damaged bar codes, some airlines have started using RFID chips embedded in the tags.

Over the last years, there have been numerous of initiatives to develop electronic bag tags, by both independent technology companies as well as some airlines. The main benefits of electronic bag tags include self-control and ease-of-use by passengers, time-saving by skipping queues at the airport, improved read rates compared to printed bag tags and, as electronic bag tags are adopted, significant operational cost reduction for the airlines.

The first company to successfully launch has been Rimowa in a partnership with Lufthansa in March 2016. The concept of electronic bag tags has been gaining ground following that launch.

Bluetooth

Bluetooth is an open standard for short-range radio frequency (RF) communication. Bluetooth wireless technology is used primarily to establish wireless personal area networks (WPANs).

The Bluetooth RF transceiver (or physical layer) operates in the unlicensed 2.4000 gigahertz (GHz) to 2.4835 GHz Industrial, Scientific and Medical (ISM) frequency band. Numerous technologies operate in this band, including the IEEE802.11b/g/n wireless local area network (WLAN) standard. Bluetooth employs frequency hopping spread spectrum (FHSS) technology for transmissions. FHSS reduces interference and transmission errors but provides minimal transmission security. The core system employs a frequency-hopping transceiver to combat interference and fading.

Bluetooth devices are managed using an RF topology known as a "star topology." A group of devices synchronized in this fashion forms a piconet, which may contain one master and up to seven active slaves, with additional slaves that are not actively participating in the network. (A given device may also be part of one or more piconets, either as a master or as a slave.) In a piconet, the physical radio channel is shared by a group of devices that are synchronized to a common clock and frequency-hopping pattern, with the master device providing the synchronization references.

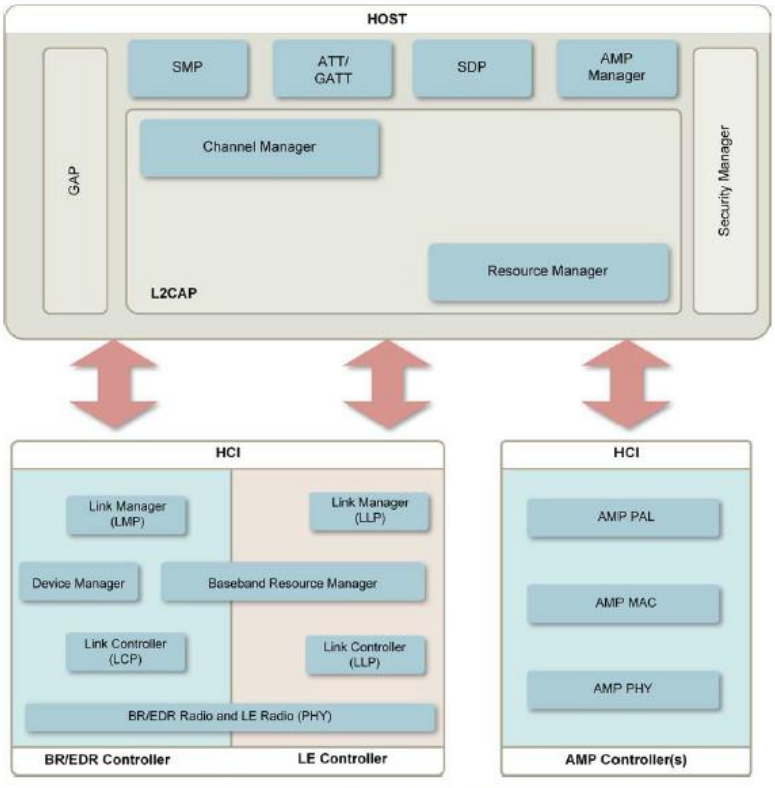


Figure 2-1. Bluetooth 4.x Device Architecture

Bluetooth low energy communication uses the same frequency range as BR/EDR devices but splits it instead into 40 channels of 2MHz width. 3 of these channels are used for advertising (broadcasting data and for connection setup) and the other 37 are data channels. These 40 channels, combined with a time division multiple access (TDMA) scheme, provide the two multiple access schemes for the low energy feature of Bluetooth.

A polling scheme is used in which the first device sends a packet at a predetermined time and a corresponding device responds after a predetermined interval. These exchanges of data are known as either Advertising or Connection Events.

Bluetooth also provides for radio link power control, which allows devices to negotiate and adjust their radio power according to signal strength measurements. Each device in a Bluetooth network can determine its received signal strength indication (RSSI) and request that the other network device adjust its relative radio power level (i.e., incrementally increase or decrease the transmission power). This is performed to conserve power and/or to keep the received signal characteristics within a preferred range.

The combination of a frequency hopping scheme and radio link power control provides Bluetooth with some additional, albeit limited, protection from eavesdropping and malicious access. The frequency-hopping scheme, primarily a technique to avoid interference, makes it slightly more difficult for an adversary to locate and capture Bluetooth transmissions than to capture transmissions from fixed-frequency technologies, like those used in IEEE 802.11b/g. Research has shown that the Bluetooth frequency hopping sequence for an active piconet can be determined using relatively inexpensive hardware and free open source software.

The range of Bluetooth BR/EDR devices is characterized by three classes that define power management. Table 2-1 summarizes the classes, including their power levels in milliwatts (mW) and decibels referenced to one milliwatt (dBm), and their operating ranges in meters (m). Most small, battery-powered devices are Class 2, while Class 1 devices are typically universal serial bus (USB) adapters for desktops and laptops, as well as access points and other mains powered devices. Many Bluetooth low energy devices are designed to run on very small batteries for a long period of time.

Table 2-1. Bluetooth Device Classes of Power Management

Type	Power	Max Power Level	Designed Operating Range	Sample Devices
Class 1	High	100 mW (20 dBm)	Up to 100 m (328 feet)	USB adapters, access points
Class 1.5 (low energy)[7]	Med-High	10 mW (10 dBm)	Up to 30 m (100 feet), but typically 5 m (16 feet)	Beacons, wearable sensors
Class 2	Medium	2.5 mW (4 dBm)	Up to 10 m (33 feet)	Mobile devices, Bluetooth adapters, smart card readers
Class 3	Low	1 mW (0 dBm)	Up to 1 m (3 feet)	Bluetooth adapters

Bluetooth wireless technology and associated devices are susceptible to general wireless networking threats, such as denial of service (DoS) attacks, eavesdropping, man-in-the-middle (MITM) attacks, message modification, and resource misappropriation. They are also threatened by more specific attacks related to Bluetooth wireless technology that target known vulnerabilities in Bluetooth implementations and specifications. Attacks against improperly secured Bluetooth implementations can provide attackers with unauthorized access to sensitive information and unauthorized use of Bluetooth devices and other systems or networks to which the devices are connected.

QR Code

A QR code is an information matrix. The difference between the 2 is that while a barcode only holds information nicely in the horizontal direction, a QR can do so vertically as well. Thus, QR codes are referred to as 2D, because they carry information both vertically and horizontally.

A QR code can carry much more information as compared to a barcode. When comparing the display of both, a conventional barcode can take up to 10 times the amount of printing space as a QR code carrying the same amount of information. A QR code is capable of being read in 360 degrees, from any direction, thus eliminating any interference and negative effects from backgrounds.

QR codes can hold up to 7100 characters of data, in comparison to the much lower number which barcodes hold. Additionally, the QR codes hold characters, numbers, symbols, text, and control codes. As codes are both horizontal and vertical, they store the same amount as the barcode, but in only 1/10 of the space the barcode requires. So, when choosing barcodes vs QR codes, in the arena of data storage, QR codes are far greater at holding and keeping storage and is able to store text messages or website addresses.

Additionally, when it comes to data restoration, barcodes are incapable of reading data when damaged and are unable to be scanned. On the other hand, when QR codes are

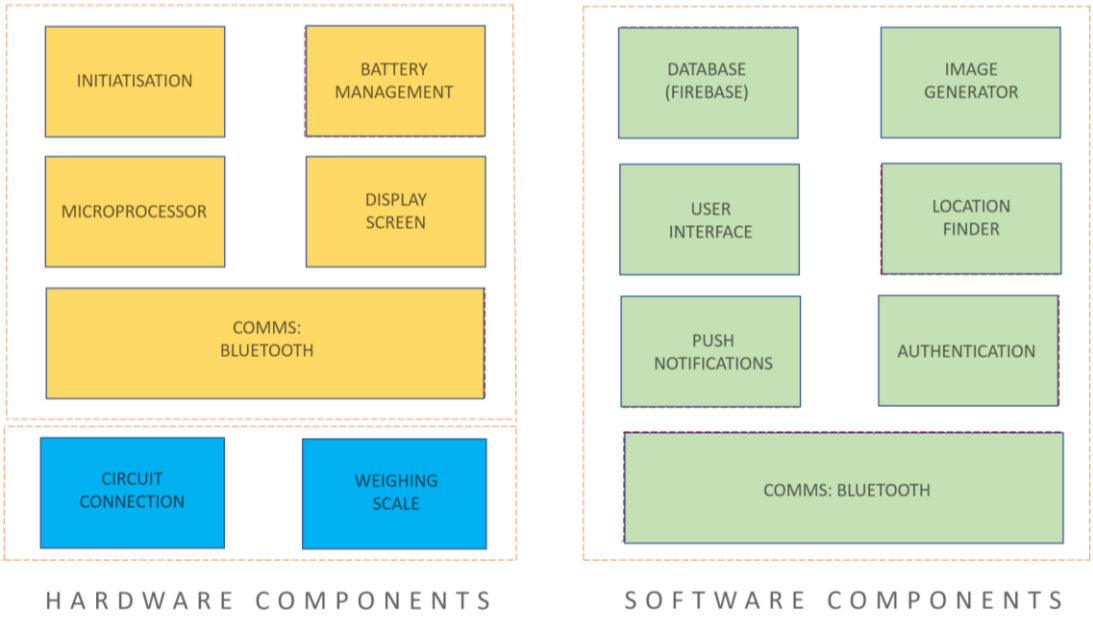
damaged, they are still capable of being scanned. Additionally, when damaged, the QR code can still recover from 30 to 35% of the damaged data, words, or symbols, making the QR code far superior in the capabilities or restoring data, or recovering information which has been lost or damaged for any reason.

QR Code Type	**Maximum Data Capacity (Characters)**
Numeric	7089
Alphanumeric	4296
Binary (8 bits)	2953
Kanji/Kana	1817

QR Level	**Error Correction Capacity (%)**
L	7
M	15
Q	25
H	30

Thus, we chose to use QR codes.

System Diagram



SOFTWARE COMPONENT

Software Component Overview

The 2 diagrams below illustrate the modules contained within the software components and the software stack behind the mobile application.

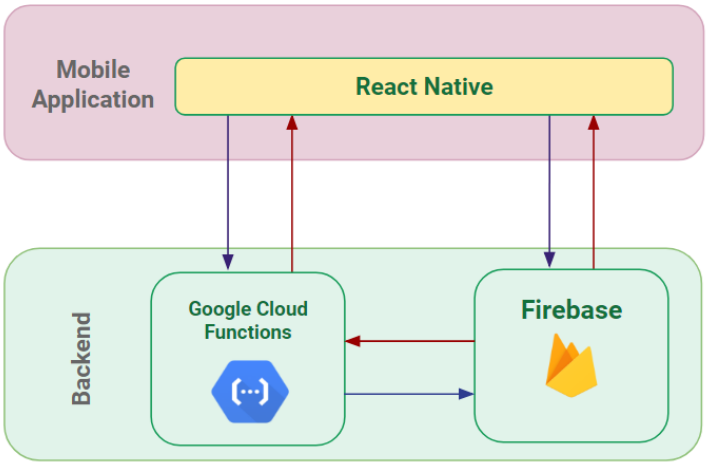


Diagram 1.1 Software Stack

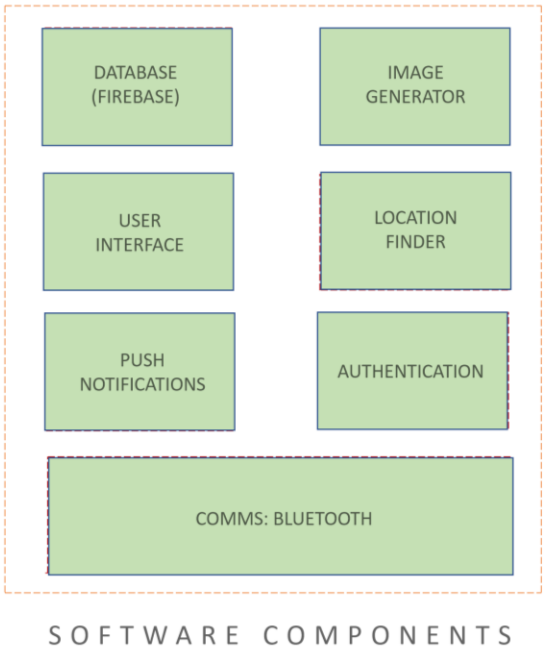


Diagram 1.2 Software Component

Frontend: React Native

React is Facebook's Javascript library for building user interfaces that are targeting mobile platforms. React Native is a Javascript framework for rendering mobile applications in iOS and Android, making cross-platform mobile application development in Javascript possible.

There are multiple advantages of developing in React Native, mainly:

- Cross-Platform Deployable
  React Native allows for cross platform deployability- the same code for iOS can be deployed to Android with minimal changes to the code. This means that development time is reduced by threefold. 90% of the code can be reused across Android and iOS.

- Effective
  Just like in ReactJS, changes to the code can be observed immediately just by saving the code. This saves a lot of time as opposed to the usual workflow of recompiling the entire application to observe the changes made.

- High Performance
  In comparison to Swift, it is 35% more efficient in terms of memory usage. As shown in Diagram 2.1 below taken from an article on Medium, React Native used significantly less memory, especially when it came to utilizing Map features.

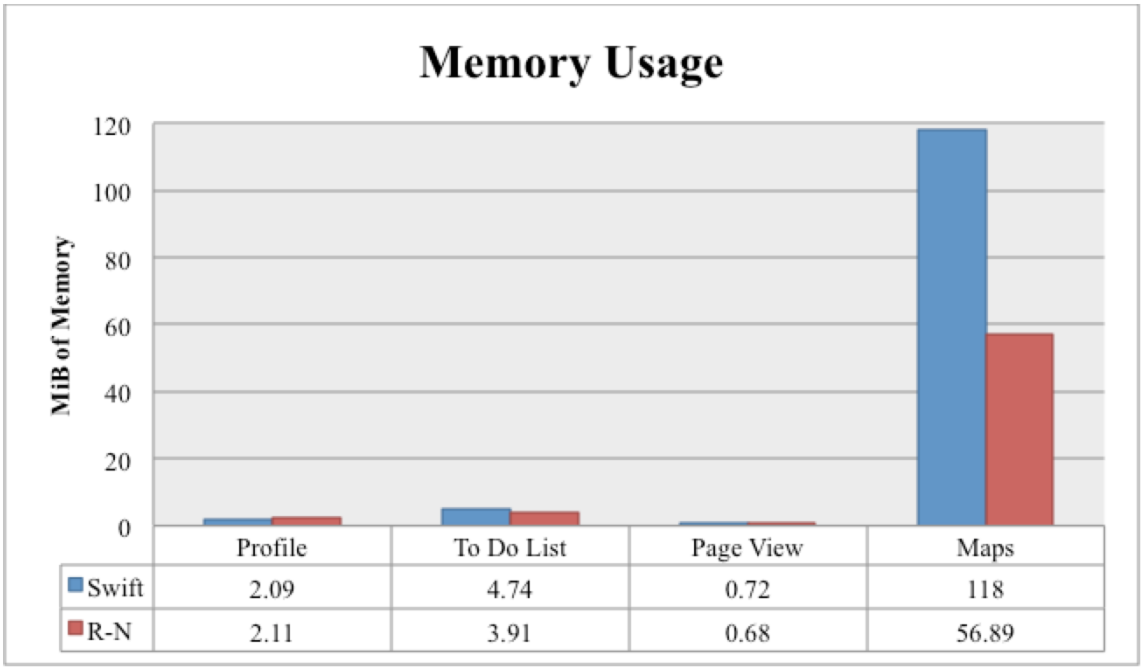


Diagram 2.1: Graph showing memory consumption of React Native & Swift

- Easy Application Testing (Live Reload)
  With React Native, it removes the need of recompilation of the application each time a change has been made. If you have two windows opened – one containing the code and the other showing a mobile screen as a result of the code – you can immediately see the effect of what you have changed in one screen, on the other screen.

There are 2 types of application development routes when you develop in React Native:

1. Create-React-Native-App (CRNA)
2. React-Native init (Native Codes)

CRNA Projects

Initializing your application using CRNA means developing using Expo, an open source toolchain built around React Native to build React Native applications. Expo provides a set of libraries to allow for easier development with React Native and it allows you to get started quicker.

To use Expo, you'll have to download their client Expo app in either Apple's App Store or Google's Play Store and download the corresponding Expo XDE from their website (expo.io) to run your codes with the Expo packager. This allows you to develop an iOS application even without owning a Macintosh computer, with access to a plethora of libraries available for use. However, using Expo means that there is a different file structure- there aren't any iOS or Android specific folders, there's only an App.js file, which is the cross-platform code for React. This means that there aren't any native codes associated with projects creating using CRNA as all the native codes have been masked by Expo SDK.

Expo applications run inside of a host app (client app) and you can publish the application within Expo itself to share it with other users. As much as this is neat and it masks a lot of the work needed to be done as compared to Native Codes, you're unable to add new native codes into the app. Only the main app-logic javascript code that interacts with the Expo components gets updated. Whilst Expo's components are vast and are usually sufficient for most applications, in my use case, it was not enough. There was only 1 React Native Bluetooth library that was stable and I had to use it if I did not want to write my own Objective-C API component.

Running a pre-existing CRNA application requires the user to use Terminal or Command Prompt.

1. cd to the project folder
2. Ensure that the directory you are in contains a package.json file
    a. If you use Yarn, run 'yarn install'
    b. If you use npm: run 'npm install --save'
3. To run the application after installation:
    a. If you use Yarn, run 'yarn start'
    b. If you use npm: run 'npm start'
4. Expo should now be up and running. To view your application using Exponent, press 's' in the terminal/cmd and enter your handphone number to receive the link to access your Expo application via an SMS.

Note: You might experience this error: *Switched to a LAN URL because the tunnel appears to be down. Only devices in the same network can access the app.*

To resolve this issue, follow these steps:

1. Open Control Panel
2. Click on Network and Internet
3. Open Network and Sharing Center
4. Go to Connections: Wifi (Open Wifi Status)
5. Open Properties
6. Click on Internet Protocol Version 4 (IPv4)
7. Open Properties
8. Click on Advanced
9. Uncheck Automatic Metric
10. Enter Interface Metric: 10
11. Close all dialog boxes and restart packager by pressing 'r' in the terminal/cmd

The downside to CRNA projects is that you cannot use custom libraries without first ejecting the whole project, and after ejecting the project, this process is irreversible, and you cannot use any of the Expo libraries. This renders all the components that you've utilized Expo's library for building the application useless. However, I had to eject the project at one point in time during development as there was no way that I could go on without using open source components that were not included in Expo's library.

Native Code

Native Code, or Native Module is used when an application requires the usage of a platform API and Expo doesn't provide it yet. I had to utilize this component of React Native as Expo doesn't have any Bluetooth modules as of the time of writing. Thus, I had to eject the whole project from Expo.

Ejecting a project from Expo results in having to build iOS applications using XCode and Swift, as well as developing Android applications in Android Studios. Additionally, every time you want to utilize a Native Module, you are required to manually link the module to your project. However, you are able to use a wide variety of native modules. As for the Bluetooth module that I was required to use, installation is complicated and much longer than usual.

Installation of Polidea's Bluetooth Module:

1. Install Carthage
2. npm install
3. Open iOS folder (cd ./ios)
4. Move BleClient.xcodeproj located in .node_modules/react-native-ble-plx/ios and drop to Libraries folder in the project
5. In the general Move BleClient.xcodeproj located in .node_modules/react-native-ble-plx/ios using drag & drop to Libraries folder in your project.
6. In general settings of a target add libBleClient.a to Linked Frameworks and Libraries.
7. In Build Settings/Search Paths/Framework search paths add path: $(SRCROOT)/../node_modules/react-native-ble-plx/ios/BleClientManager/Carthage/Build/iOS.
8. In Build Settings/Build Options/Always Embed Swift Standard Libraries set to Yes.
9. In Build Phases click on top left button and add New Run Script Phase.
10. Shell command: /usr/local/bin/carthage copy-frameworks
11. Input Files:
    $(SRCROOT)/../node_modules/react-native-ble-plx/ios/BleClientManager/Carthage/Build/iOS/BleClientManager.framework
    $(SRCROOT)/../node_modules/react-native-ble-plx/ios/BleClientManager/Carthage/Build/iOS/RxSwift.framework
    $(SRCROOT)/../node_modules/react-native-ble-plx/ios/BleClientManager/Carthage/Build/iOS/RxBluetoothKit.framework
12. Pass restoreStateIdentifier and restoreStateFunction to BleManager constructor.

Firebase

Firebase is Google's cloud database and it is a NoSQL database. Data is synced across all clients in realtime and remains available even when your app is offline, as all cloud features do. As this uses Serverless Computing, processes are short-lived (20ms to 3000ms) and codes are organized into functions. I compared this against hosting a SQL database on AWS's EC2 Micro instance, and Firebase is comparatively much easier to get started with. If EC2 is used, the process is much more tedious- initialize an AWS EC2 instance, choose your server side environment and language, database setup and selection, domain names and DNS configuration, load balancing and scaling, VPC's and Security Groups; a lot of work just to get an application up.

I utilized 2 features from Firebase- Google Cloud Function and Realtime Database.

Firebase's Realtime Database is my database of choice for 2 main reasons:

1. Hierarchical Database
2. Compatibility with React Native

Hierarchical database matches the requirements of our project much better in multiple ways. As it follows the JSON tree structure, it is much easier to manage the user data and sort it. It has the concept of using 1 main key per node. Thus, there aren't foreign keys or tables and data is sorted according to nodes. This makes it much easier to call the data as a whole as you can refer to the nodes.

Additionally, as React is known for handling multiple changes in data state quickly, Firebase is perfect for this as it allows you to listen for changes in database data.

Google Cloud Functions

As Firebase is serverless, to execute typical server-side functions like generating a JSON web token for authenticating a user, you use Google Cloud Functions. Google Cloud Functions enable you to write simple, single-purpose functions that are attached to events emitted from your cloud infrastructure (i.e. Firebase Realtime Database), being triggered when an event being monitored is called. Furthermore, as the environment is fully managed by Google, there is no need to provision any infrastructure or manage servers.

In my application, I wrote 3 Google Cloud Functions:

1. Create New User (create_user.js)
2. Request OTP (request_one_time_password.js)
3. Verify OTP (verify_one_time_password.js)
4. Send OTP via Twilio's messaging API (twilio.js)

JSON Web Tokens (JWT)

JSON Web Tokens are critical in ensuring trust and security in an application, thus I chose to implement it as a way to authenticate a user in the application. JWT is a JSON object that is defined in RFC7519 as a safe way to represent a set of information between 2 parties. The token is composed of a header, a payload and a signature. JWT authentication setup in our application uses a symmetric key algorithm (HS256).

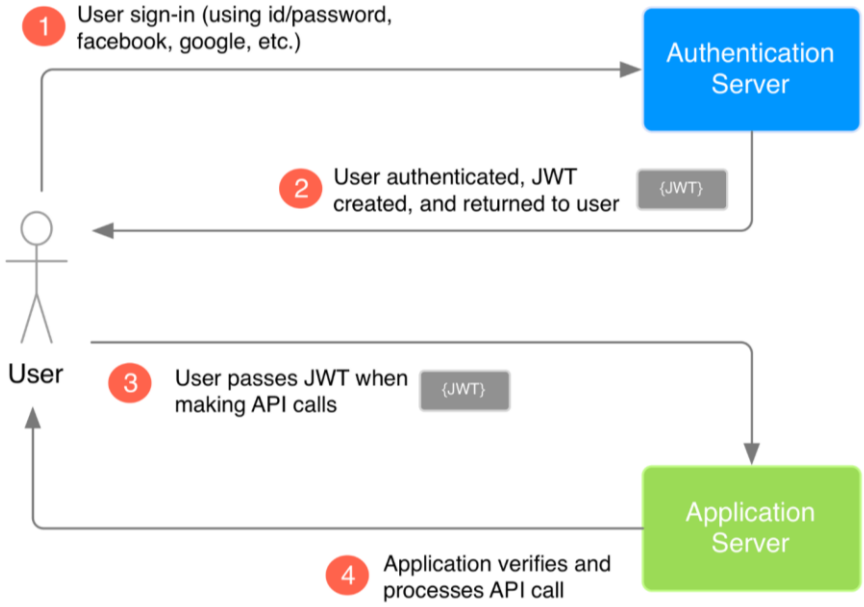


Diagram 2.3 How an application uses JWT to verify the authenticity of an user

The purpose of using JWT is not to hide or obscure data in any way. JWT is used to prove that the sent data was created by an authentic source. The data inside a JWT is encoded and signed, not encrypted. The purpose of encoding data is to transform the data's structure. Signing data allows the data receiver to verify the authenticity of the source of the data. Thus, encoding and signing data does not secure the data. On the other hand, the main purpose of encryption is to secure the data and to prevent unauthorized access.

JWT Header

The header component of the JWT contains information about how the JWT signature should be computed. The header is a JSON object in the following format:

In this JSON, the value of the "typ" key specifies that the object is a JWT, and the value of the "alg" key specifies which hashing algorithm is being used to create the JWT signature component. In this example, I'm using the HMAC-SHA256 algorithm, a hashing algorithm that uses a secret key, to compute the signature.

Example:

```
{
    "typ": "JWT",
    "alg": "HS256"
 }
```

JWT Payload

The payload component of the JWT is the data that's stored inside the JWT (this data is also referred to as the "claims" of the JWT). In this example, the authentication server creates a JWT with the user information stored inside of it, specifically the user ID.

Example:

```
{
    "userId": "b08f86af-35da-48f2-8fab-cef3904660bd"
 }
```

In our example, only 1 claim is placed into the payload. You can put as many claims as you like. There are several different standard claims for the JWT payload, such as "iss" the issuer, "sub" the subject, and "exp" the expiration time.

The size of the data will affect the overall size of the JWT, this generally isn't an issue but having excessively large JWT may negatively affect performance and cause latency.

JWT Signature

The signature is computed using the following pseudocode:

```
data = base64urlEncode (header) + "." + base64urlEncode (payload)
signature = Hash (data, secret);
```

What this algorithm does is base64 encodes the header and the payload created in steps 1 and 2. The algorithm then joins the resulting encoded strings together with a period (.) in between them. In the pseudocode, this joined string is assigned to data. To get the JWT signature, the data string is hashed with the secret key using the hashing algorithm specified in the JWT header. The base64 encoded result:

Header:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9

Payload:
eyJ1c2VySWQiOiJiMDhmODZhZi0zNWRhLTQ4ZjItOGZhYi1jZWYzOTA0NjYwYmQifQ

Then, using the joined encoded header and payload, and applying the specified signature algorithm(HS256) on the *data* string with the secret key set as the string “secret”, you obtain the following JWT Signature:

-xN_h82PHVTCMA9vdoHrcZxH-x5mb11y1537t3rGzcM

JWT Signature

In the example, a JWT is signed by the HS256 algorithm where only the authentication server and the application server know the secret key. The application server receives the secret key from the authentication server when the application sets up its authentication process.

Since the application knows the secret key, when the user makes a JWT-attached API call to the application, the application can perform the same signature algorithm as the creation of the payload on the JWT. The application can then verify that the signature obtained from it’s own hashing operation matches the signature on the JWT itself (i.e. it matches the JWT signature created by the authentication server). If the signatures match, that means the JWT is valid which indicates that the API call is coming from an authentic source. Otherwise, it means that the received JWT is invalid, which may be an indicator of a potential attack on the application. Thus, by verifying the JWT, the application adds a layer of trust between itself and the user.

Twilio

Twilio allows for programmed SMS delivery services via their Programmable SMS feature. It is able to send and receive text messages over the carrier network to any phone anywhere in the world. It has a developer panel to allow you to identify details about the format and connnection type of the phone numbers to reduce undevliered messages and protect from spam and fraud. As the free tier only allows you to send messages to 1 registered handphone number, I was only able to send SMS to my own handphone number.

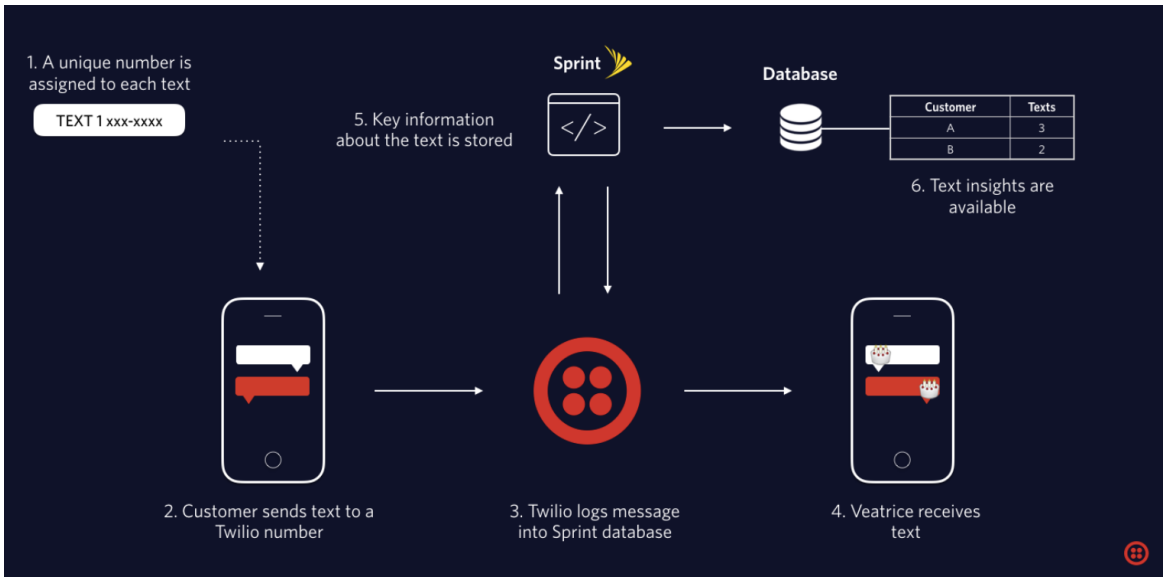


Diagram 2.4 How Twilio's Messaging system works

Passwordless Login

As security is only as strong as the weakest link in the chain and when users' behaviour determines the strength of the security of the application, passwordless login is the best fit for this use case.

Unfortunately, when you allow your users to set a password, they tend to set easier, less complex passwords. Increasing the minimum password complexity and setting a minimum/maximum password age is incredibly frustrating to users and it usually results in users not wanting to use the application entirely. Furthermore, everyone has passwords to remember across multiple platforms. In a survey conducted by Telesign, 73% of users reused their passwords across multiple platforms. More than half of consumers (54%) use five or fewer passwords across their entire online life, while 22% use just 3 or fewer and almost half (47%) of consumers rely on a password that hasn't been changed for 5 years.

Many users reuse their passwords across multiple platforms, making all their accounts vulnerable to attacks once one account is compromised in a data breach. Our application is meant to target everyone who travels by plane, thus the application has to cater all

people of all ages. The older generation usually finds passwords a hassle and would usually not want to use an application entirely if there is a minimum password complexity involved.

The solution to all of these problems is a passwordless login. The user's password always change and the user doesn't have to remember any passwords, thus striking a balance between security and complexity.

Authentication Flow	**Description**
Email/Password	Send email and password. If the email and the hash of the password matches the ones in the database, the user is authenticated.
OAuth	3rd party authentication by providing an identifying token to the backend. (E.g. Facebook Authentication)
2FA (2 Factor Authentication)	In addition to OAuth or email/password, the user is sent an additional identifying token via SMS, phone call or email.
OTP	The user provides a phone number as an identifying token. The code is then texted to the user, which they can enter into the app.

Diagram 2.5 Illustration of a variety of Authentication Flow

On the server side, there are many internal features that’s going on in the application, as illustrated on the diagram below.

Step	**Description**
1	User enters email and phone
2	Verify phone is not in use
3	Create a new user record in Firebase
4	Respond to request, stating user was created
5	User requests to login with phone number
6	Generate a code
7	Save the code to the user’s record
8	Text the code to the user
9	User enters code
10	Compare codes
11	Mark code as invalid
12	Return a JWT to user

Diagram 2.6 Illustration of my Login flow

Location Mapping

To indicate where the airport is located at, I utilized Expo's Maps API. Using their reverse geolocating feature and auto-parametrization of geolocated buildings, the user is able to see where the airport is. It also enables the user to locate where the F&B outlets are within the airport, helping the user to purchase any necessities.

QR Code Generation

A QR Code will be generated according to the data pulled from Firebase via Expo's QR Code library.

Bluetooth Services

Polidea's React Native Bluetooth component was utilized in the creation of the Bluetooth feature. The application is able to scan, connect, discover services and characteristics via location of their UUIDs. It utilizes the writeCharacteristicWithoutResponseForService method to write to Arduino. Initially my codes used the writeCharacteristicWithResponseForService method to write to Arduino. However, we soon encountered an issue when I wrote to Arduino with response (asynchronous request).

BleManager

BleManager is an entry point for the react-native-ble-plx library. It provides all means to discover and work with Device instances. It should be initialized only once with new keyword and method destroy() should be called on its instance when user wants to deallocate all resources.

Example: const manager = new BleManager();

Connect to device

This method enables you to connect to the specified device with partially filled arguments and it returns a promise detailing on whether or not it has successfully connected.

Example: connect(options: ConnectionOptions?): Promise<Device>

Device

Device instance which can be retrieved by calling bleManager.startDeviceScan(). The function takes in parameters such as the device's name, device's UUID, service UUIDs, requested MTU

Example: new Device(nativeDevice: NativeDevice, manager: BleManager)

Write Characteristics to Bluetooth

There are 2 methods to write the characteristics to Bluetooth:

1. bleManager.writeCharacteristicWithResponseForDevice()
2. bleManager.writeCharacteristicWithoutResponseForDevice()

Both methods require the data passed in to be converted to Base64 format, as the library itself will then convert the Base64 encoded information into binary data before sending it to the Bluetooth module.

Writing of characteristic with response is the default way of writing characteristics (also known as a Write Request). It consists of a Write Request and a following Write Response indicating whether the write has been accepted by the peripheral (promise). It takes in 4 parameters:

1. deviceIdentifier: Obtained from a previously declared device.id
2. serviceUUID: UUID of service consisting of the characteristic to write to
3. characteristicUUID: UUID of characteristic to write to
4. base64Value: Value of data to be sent in base64 value

I initially tested the characteristic writing with response. However, it was only able to send data to the Bluno board's Bluetooth module and not the external Bluetooth module (HM-11). When I tried sending data via the method that did not return a response for device, it worked.

Writing of characteristic without response does not induce a Write Response from the peripheral. Not all peripherals/characteristics support this type of write. It is generally faster to send more data to the peripheral but in expense of losing the ability to know if the peripheral is able to process the bulk of data (it is peripheral's responsibility to control the flow). It takes in the same 4 parameters as writing of characteristic with response.

HARDWARE COMPONENT

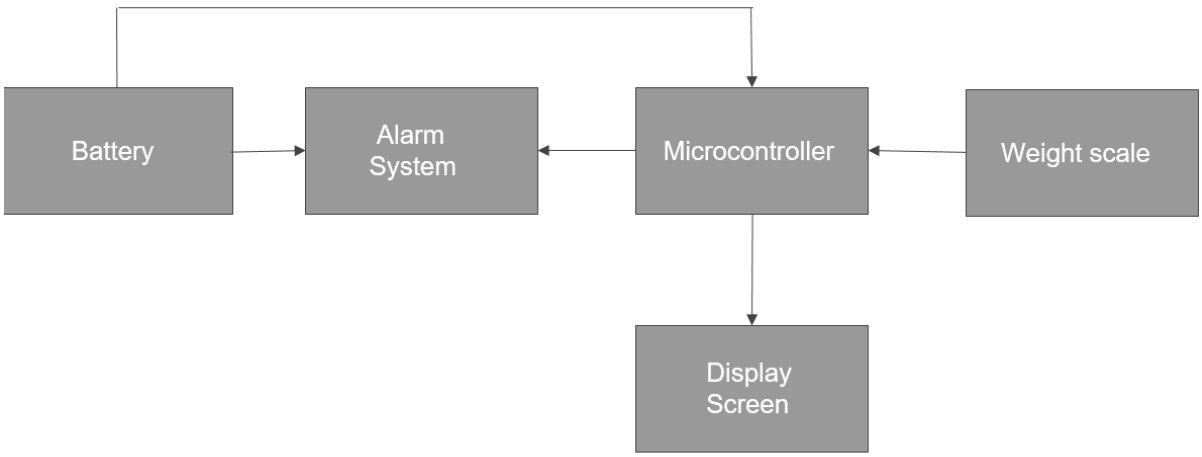


Figure 2.7: Block diagram of Hardware

LOAD CELLS

A load cell is a device that converts force to an electrical signal. There are a types of load cell from hydraulic, pneumatic, strain gauge, piezoelectric. Depending on the industry and purpose of an equipment, the number and type of load cell varies as there are load cell of different size and capacities. In this project, we have decided to use 4 strain gauge load cell. We decide to use the strain gauge load cell as compared to the other forms of load cell, it is relatively smaller and extremely sensitive.

Strain Gauge

The strain gauge load cell works by the change in resistance when a load is act upon the strain gauge load cell. The change in resistance provides an electrical signal value change that is calibrated to the load placed on the load cell. Strain gauge also use the wheatstone circuit connection. The wheatstone bridge connection is needed as it helps the strain gauge load cell to balance the reading in the resistance between the load cells. Wheatstone bridge circuit is also used to measure both static and dynamic changes resistance. However, the changes in the values are extremely small, typically in the order of a few millivolts, thus an amplifier is needed.

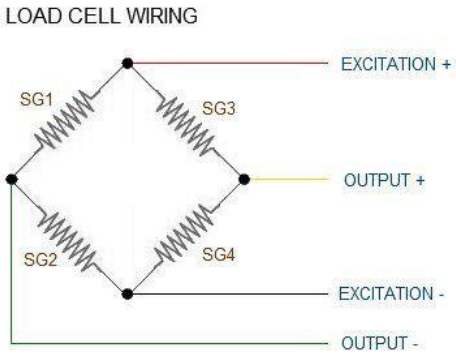


Figure 2.8: Picture of a wheatstone circuit

Piezoelectric

The piezoelectric load cell works in the same principle as the strain gauge load cell except that the voltage output is generated by the piezoelectric material. The piezoelectric effect is dynamic whereby the electrical output is sudden, impulsive function, not static.

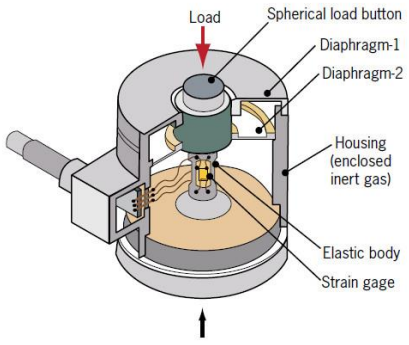


Figure 2.9: Picture of a piezoelectric load cell

Hydraulic

The hydraulic load cell uses a piston and cylinder arrangement where the piston is in a diaphragm and works through the principle of hydraulic. The load cell is then filled with fluid, an example being oil. When a load is applied on the piston, the movement of the piston increases the fluid pressure. The pressure is then transmitted to a hydraulic gauge which reads the pressure then displaying it on the dial.

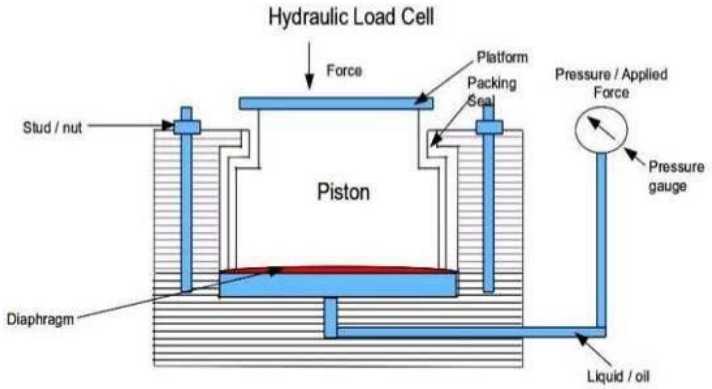


Figure 3.0: Picture of a hydraulic load cell

Pneumatic

Pneumatic load cell uses air pressure and nozzle which is connected to a load cell. Air pressure is first applied at an end of the diaphragm then it will escape through the nozzle placed at the bottom of a load cell. The load cell is also made to regulate the balancing pressure.

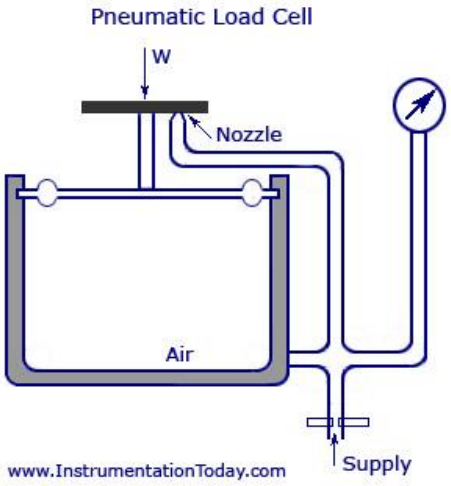


Figure 3.1: Picture of a pneumatic load cell

HX711 Amplifier

Due to the strain gauge value changes being extremely small, only by a few millivolts, the HX711 amplifier is also known as an ADC (analog to digital converter). The amplifier has a two wire interface, Clock and Data, for communication. The HX711 would be connected to the load cells and the microcontroller. Taking the reading from the load cells, amplifying the electrical signal to the microcontroller.

Features of the HX711 Amplifier

Operating voltage: 2.7V-5V

Operation current: < 1.5mA

Selectable 10SPS or 80SPS output data rate

Simultaneous 50 and 60HZ supply rejection

Figure 3.2: Picture of the HX711 amplifier used

How to connect the HX711 Amplifier

From the load cell connection, only 4 pins are used in this project which is the E+, E-, A-, A +. While connection from the HX711 Amplifier to the microcontroller uses the GND, DT, SCK and VCC.

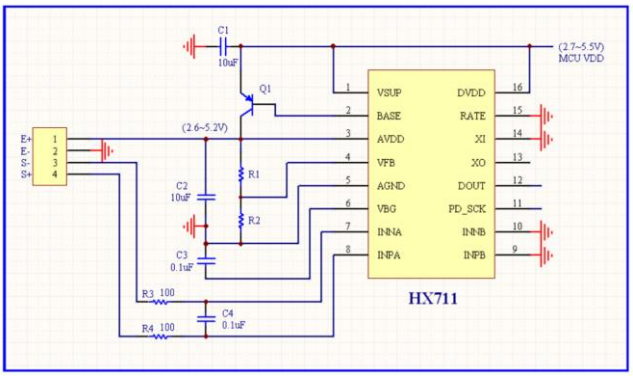


Figure 3.3: Picture of schematics of the HX711 Amplifier

Load Cell Connection to the HX711

The load cell are normally 4-wire however the load cell given to us was 3-wire. Thus we had to make the 3-wire load cell into a 4-wire through the use of the wheatstone bridge by using 2 1k Ohm resistors. The 1K resistors are used as the resistance difference between the load cell wires are 1K.

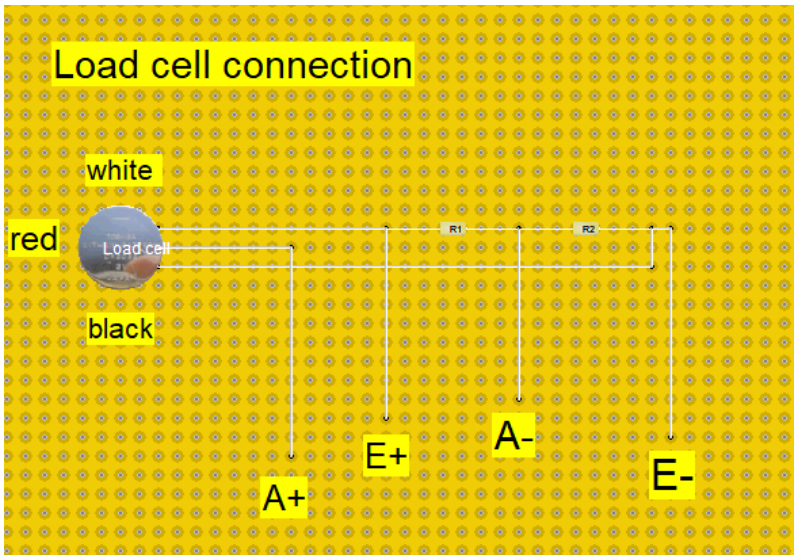


Figure 3.4: Picture of one load cell connection

Connection of HX711 to Arduino board

Yellow to pin 2

Blue to pin 3

Red to 5v

Black to ground

Load Cell Accuracy Factors

Placement

Placement of the load cells are extremely important. This is due to the fact that the placement and alignment of the load cells will affect the readings and accuracy of the

result. This is to ensure a more balance and accurate result overall by the principle of equilibrium. Thus In this project, the 4 load cells are placed at the 4 corners of the platform.

Weight Scale Comparison

To ensure an accurate reading of our weighing scale in this project, we went to Changi airport to compare, in terms of the weighing scale calibration. Using items such as bag, clothes and dumbbells.

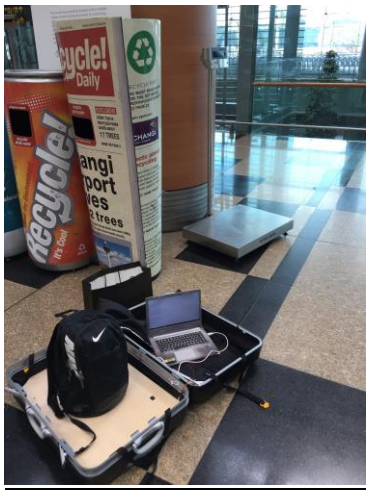


Figure 3.5: Picture comparing project weighing scale and airport weighing scale

Item	Weight of item	Project result	Airport result
Jacket	0.5kg	0.49kg	0.5kg
Bag-pack	3.2kg	3.17kg	3.2kg
5kg dumbbell	5kg	4.9kg	5.0kg

2kg dumbbell	2kg	2.05kg	2.0kg
Clothes	0.7kg	0.67kg	0.7kg

Arduino Coding For Weigh Scale

```
#include "HX711.h"
#define DOUT  3 // getting the data reading from the load cells to the pin
#define CLK  2 // getting data out of the data pin

HX711 scale(DOUT, CLK);
int i,j;
float calibration_factor = -7870; //calibration (compare to airport weighing scale)
float a,b,avg;

void setup()
{
  Serial.begin(9600);
  scale.set_scale();
  long zero_factor = scale.read_average(); //Get a baseline reading
  scale.set_scale(calibration_factor); //This value is obtained by using the SparkFun_HX711_Calibration sketch
   scale.tare(); //Assuming there is no weight on the scale at start up, reset the scale to 0
}
void loop()
{
   b=0;
   scale.get_units(); //get readings
if(scale.get_units()> 0.4) //get readings after threshold of 0.4kg )
{
for(i=0;i<50;i++) // get 50 readings
{
  a=scale.get_units(); //get readings
  b=b+a; //total value of all 50 readings
}
}
avg=(b/50);// average the first 50 readings , add 0.2 to make it accurate to start at 0.00kg
Serial.println("Average weight:");
Serial.println(avg);
}
```

DOUT is put at pin 3 of the arduino which is use to take the reading from the 4 load cells while CLK is put at pin 2 of the arduino which is use to get the data out of the pin.

Calibration factor is use to calibrate the load cell, to get an accurate reading which would depend on the type of load cell being use and position of the load cells, followed by comparing the results with other weighing scale, in our case it's the change airport weighing scale. Scale.set.scale() is used to reset the scale back to 0 readings, it only happens once as it is in the void setup.

In my coding, I have set 0.4kg to set as a threshold for the coding of getting the weight to start. Taking the first 50 readings, totalling them and then dividing it by 50. This is done to get an accurate result.

ALARM SYSTEM

In this project, we have an alarm system for the battery voltage which consist of a buzzer as an alarm. To ensure that the voltage of the battery does not overvoltage and destroy the microcontroller pin being used, a voltage divider circuit is placed. The alarm system is put in place to alert the user if the batteries is low, in these project case,3.5V and to change them.

Components being used are a piezo buzzer, 1 100 Ohm resistor, 1 160K Ohm resistor, 1 100K Ohm resistor.

Formula:

R1/RT =100 /100+160 =2.6

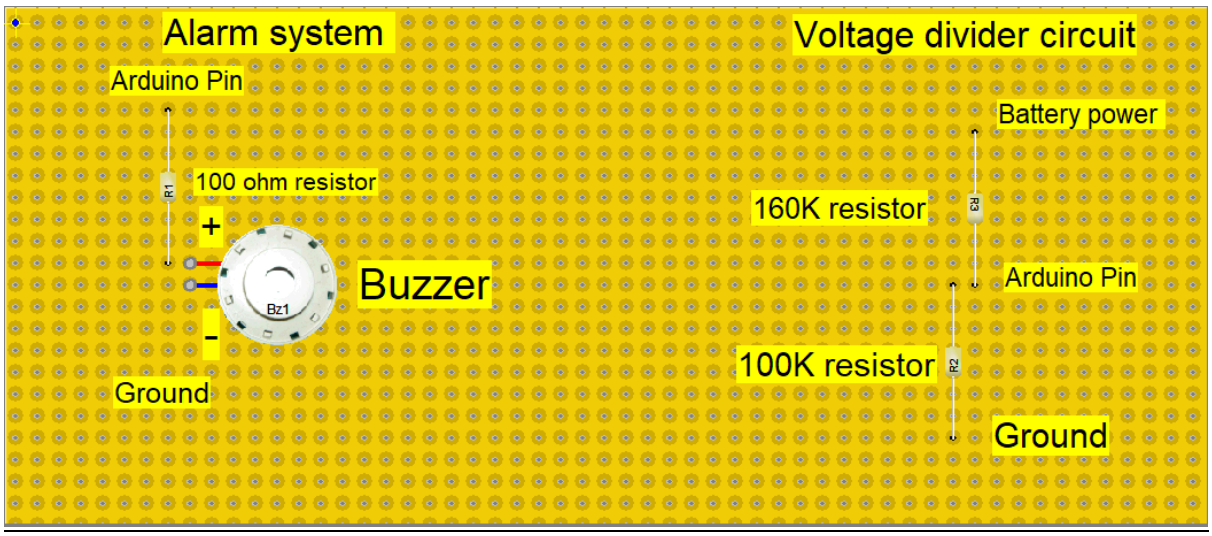


Figure 3.6: Connection of the Alarm System and Voltage Divider

Connection to Arduino

Buzzer positive (orange color) connected to the 100 ohm resistor then to 1 PWM pin and negative connected to ground

Voltage divider(arduino pin uses an analog input pin (yellow color wire) and ground

Arduino coding for Alarm system

```
//Specify digital pin on the Arduino that the positive lead of piezo buzzer is attached.
int piezoPin = 7;
int myCounter = 0;
int voltagePin = A6;
int Battery=0;



void setup()
{
 pinMode(piezoPin,OUTPUT); //Set output for piezo buzzer
 Serial.begin(9600);

}  //close setup
```

```
 void loop()
 {
  while(1)
 {
  Battery = analogRead(voltagePin); //set the analog to read the voltage , set to a name:Battery
   float Batterylife = (((Battery/1024.0)*5)*2.6; //formula varies due to the resistors value
  Serial.println(Batterylife);
   if(Batterylife < 3.5 )
   {

   digitalWrite(piezoPin,HIGH);  //On the buzzer

   }
   else
     {
        digitalWrite(piezoPin,LOW);  //Off the buzzer
     }

   }

 }
```

BATTERY

In this project, we have decided to use the external power source for the microcontroller which needs 7V-12V. With the regulations of airport safety on batteries into consideration too, we have research and compared different type of batteries.

These are some of the considerations, power (Wh), capacity (mAh), voltage, if commonly found.

Type of battery	Power(Wh) (Ah X V)	Voltage (V)	capacity (mAh)	common
AA	3.90	1.5	1800-2600	common
AAA	1.3-1.8	1.5	860-1200	common
CR123A	4.2	3	1400	common
CRV3	9	3	3000	common
18650	13.32	3.7	1500-3600	uncommon
CR-P2	9	6	1500	uncommon
9 volt or E	5.625	9	625	common

CR2	2.25	3	750	common
RCR123A	2.04	3	680	common
16340	3.7	3.7	500-1000	uncommon
HR14	3	1.2	2500	uncommon
2CR5M	4.8	3	1600	uncommon

After much consideration and research, we have chosen the CR123A as our battery source as it has a high capacity which would allow the systems to last longer, voltage of 3V, thus only needing 3 of the batteries as compared to AA or AAA which would need 6 batteries, it is also common to find in stores and shopping malls.

DISPLAY SCREEN

In this project, we have chosen the waveshare 4.3inch e-ink display screen as our chosen screen. The reason why we did so was due to the fact that it is able to display numbers, words and picture. It is also extremely low in power consumption. It can also store images without needing any power. It comes with 6 wires in the colors of red, black, white, green, yellow, blue.

Display Screen Connection To The Arduino

Arduino	Display screen
5V	Red
Ground	Black
RX	White
TX	Green
Arduino pin (e.g. D2)	Yellow
Reset (leave it disconnected)	Blue

Figure 3.7: Picture of the 4.3inch E-ink display screen

ARDUINO

Arduino is a microcontroller that allows open source hardware and software programming. It is relatively easy to use and learn for students that are new to electronic and programming background. The arduino have pins which allows input and output of data and signals to be sent. In this project, we have compare arduino boards and end up picking the Arduino DUE.

We picked the Arduino DUE due to its memory and processor having a higher capacity than compare to the Arduino bluno mega 2560.

Arduino Board	Processor	Memory	Analog I/O	Digital I/O
Arduino DUE	84Mhz	96KB SRAM, 512KB flash	12 input, 2 output	54
Arduino bluno mega 2560	16Mhz	Memory capacity of EEPROM: 4KB	16 input	54 14 PWM

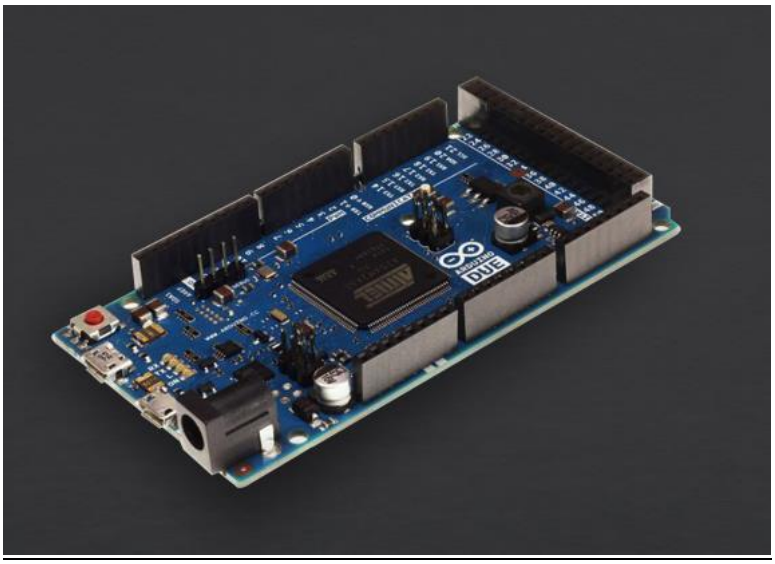


Figure 3.8: Picture of Arduino DUE

BLUETOOTH

For this project, we have decided to use bluetooth as our method of communication. Thus, we have chosen the HM-11 bluetooth module with a working frequency of 2.4 Ghz ISM band, ability to send and receive no byte limit. Using 4 pins to connect to the arduino, 5v, ground, TX, RX. Using an arduino shield to directly connect to the the Bluetooth to the shield.

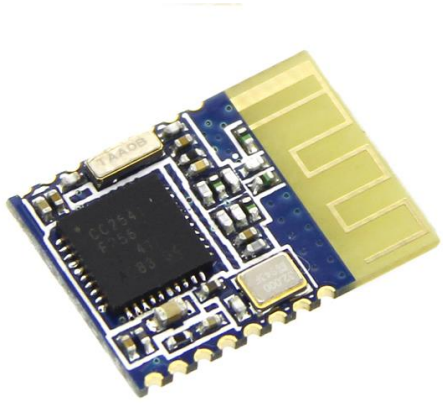


Figure 3.9: Picture of HM11 bluetooth module

CASING

Due to how luggages are treated when transporting, casing are made in this project to protect the component. Using the software Autodesk inventor and Makerspace's equipments to make the casings.

Equipments and materials used in Makerspace

Soldering tool: use to connect wires permanently

Up-Box 3D Printer: use to print out the load cell casing

Scroll saw: use to cut the wood for the foundation of the load cell and to fit into the luaggage

Bench drilling machine: use to drill holes

Belt and disc sander: use to send the excess wood and making it smooth

Multimeter: to check connectivity of the wires

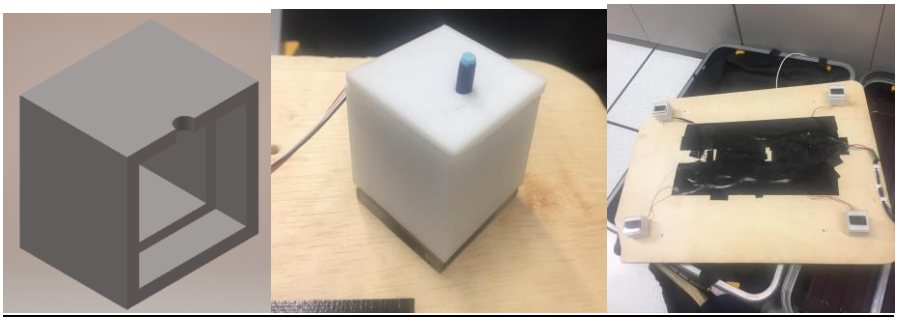


Figure 4.0: Casing for the weighing scale

The casing of the load cell are structured as shown on figure 13 as only the middle portion of the load cell are sensitive, being push down by pressure to get the reading. This ensures that the pressure is centralise.

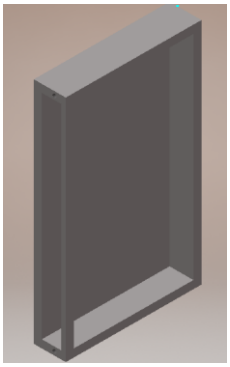


Figure 4.1: Casing for display screen holder

Casing	Length	Breath	Height
Load cell holder	35mm	35mm	35mm
Display screen holder	160mm	100mm	20mm

Wire Connection

To connect the wires together for the circuitry in the weighing scale, the cable wires are connected using soldering.

The wires use to connect the arduino board are male to male wires. They are used as the chance of hurting the pin and the connection being loose are lowered.

Wire Protection

To protect the wires from being damaged, we decided to use insulation tape and cable duct. Insulation tape is use to keep the wires organised and keep it waterproof. Cable duct is used as a casing for the wrapped wires to prevent any external damages that may happen.

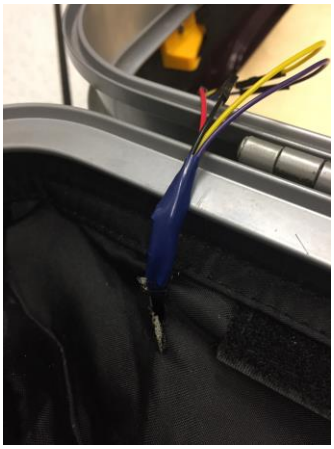


Figure 4.2: Picture of wires wrap with insulation tape

Figure 4.3: Picture of cable duct as a casing

Firmware component

Firmware Component Overview

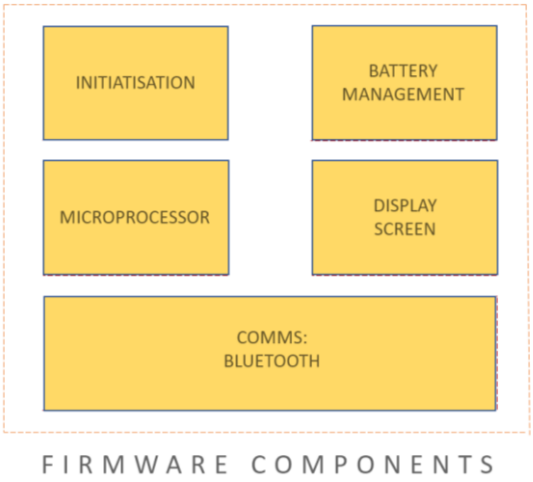


Figure 4.4: Firmware System Diagram

1.1 Arduino

Arduino was used as the base for all my codes in the firmware part of this project. Arduino is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board. The Arduino platform has become quite popular with people just starting out with electronics, and for good reason.

Unlike most previous programmable circuit boards, the Arduino does not need a separate piece of hardware (called a programmer) in order to load new code onto the board – you can simply use a USB cable. The boards are able to transmit inputs into outputs by just computing the instructions or coding using the Arduino programming language and its software called Integrated Development Environment (IDE) to the microcontroller that is already available on the board. Due to its accessible knowledge, Arduino has been used in a lot of different kinds of projects from everyday objects to complex scientific instruments for both beginners and experts.

There are also different types of features to an Arduino boards so that users are able to choose the board that matches the type of project they are handling and whether it requires more input ports or larger memory or processor.

The table below shows the comparison of the different features to an Arduino board.

Arduino Board	Memory	Processor	Analogue I/O	Digital I/O
Arduino UNO	2KB SRAM, 32KB flash	16Mhz ATmega328	6 input, 0 output	14
Arduino DUE	96KB SRAM, 512KB flash	84Mhz AT91SAM3X8E	12 input, 2 output	54
Arduino MEGA 2560	8KB SRAM, 256KB flash	16Mhz ATmega2560	16 input, 0 output	54

After much consideration, we decided to use Arduino Bluno Mega 2560 board. However, in the later part of our project, we had to change our board to Arduino DUE as it has more RAM which we needed for our transfer of image from the app side to the display screen. This will be further discussed in the Bluetooth segment later.

1.2 Battery Management

Purpose of batteries: Without the minimum voltage required, the circuit can't be powered up to function the different components of our project. The minimum voltage required to power up the entire system without any problems is 3.5 Volts. Any voltage lesser than that will cause the alarm to be turned on to alert the user that it's time to change the batteries to power the whole system. The alarm will go on till the user changes the batteries for the program to work. On the other hand, if the voltage is above 3.5Volts, then the program will run as per normal.

After researching the various batteries and its limitations with bringing it on board in the plane such as following by the rules of IATA, we finally concluded to use the CR123A batteries.

As a smart alternative, we are planning to use a power bank as a alternative for batteries as it will be both rechargeable and you can remove it to use for the smartphone on the plane as well. This way, we can avoid the idea of exploding batteries on board the plane.

Figure 4.5 Picture of CR123A batteries

1. 【SCOPE】

This specification applies to the following 3.0v lithium cell CR123A manufactured by AA Portable Power Corp (http://www.batteryspace.com)

2. 【RATINGS】

TABLE Ⅰ:

ITEM		UNIT	SPECIFICATIONS	CONDITIONS
Nominal voltage		V	3.0	Standard discharge
Nominal capacity		mAh	1300	Standard discharge
Instantaneous short–circuit current		A	⩾10	Time⩽0.5 second
Off-load voltage		V	⩾3.2	
Storage temperature		℃	-40~60	
Standard weight		g	16	Unit cell
Service output	Initial	Standard	1000 h	Continuous discharge with load 15kΩ, till 2.0v end-voltage
		Minimum	900 h	
	After 12 months storage	Standard	950 h	
		Minimum	850 h	

Figure 4.6: Table of Battery Specifications

1.3 Bluetooth Connectivity

Bluetooth is a short-range wireless communication technology that allows devices such as mobile phones, computers, and peripherals to transmit data or voice wirelessly over a short distance. The purpose of Bluetooth is to replace the cables that normally connect devices, while still keeping the communications between them secure.

Bluetooth Technology

Bluetooth was intended as a wireless replacement for cables. It uses the same 2.4GHz frequency as some other wireless technologies in the home or office, such as cordless phones and WiFi routers. It creates a 10-meter (33-foot) radius wireless network, called a personal area network (PAN) or piconet, which can network between two and eight

devices. This short-range network allows you to send a page to your printer in another room, for example, without having to run an unsightly cable.

Bluetooth uses less power and costs less to implement than Wi-Fi. Its lower power also makes it far less prone to suffering from or causing interference with other wireless devices in the same 2.4GHz radio band.

Bluetooth range and transmission speeds are typically lower than Wi-Fi (the wireless local area network that you may have in your home). Bluetooth v3.0 + HS—Bluetooth high-speed technology—devices can deliver up to 24 Mbps of data, which is faster than the 802.11b Wi-Fi standard, but slower than wireless-a or wireless-g standards. As the technology has evolved, however, Bluetooth speeds have increased.

Bluetooth version 4.0 features include low energy consumption, low cost, multivendor interoperability, and enhanced range. The hallmark feature enhancement to the Bluetooth 4.0 spec is its lower power requirements; devices using Bluetooth v4.0 are optimized for low battery operation and can run off of small coin-cell batteries, opening up new opportunities for wireless technology. Instead of fearing that leaving Bluetooth on will drain your cell phone's battery, for example, you can leave a Bluetooth v4.0 mobile phone connected all the time to your other Bluetooth accessories.

Connection via Bluetooth

The process of connecting two Bluetooth devices is called "pairing." Generally, devices broadcast their presences to one another, and the user selects the Bluetooth device they want to connect to when its name or ID appears on their device. As Bluetooth-enabled devices proliferate, it becomes important that you know when and to which device you're connecting, so there may be a code to enter that helps ensure you're connecting to the correct device.

Bluetooth Limitations

There are some downsides to Bluetooth. The first is that it can be a drain on battery power for mobile wireless devices like smartphones, though as the technology (and battery technology) has improved, this problem is less significant than it used to be.

Also, the range is fairly limited, usually extending only about 30 feet, and as with all wireless technologies, obstacles such as walls, floors, or ceilings can reduce this range further. The pairing process may also be difficult, often depending on the devices involved, the manufacturers, and other factors that all can result in frustration when attempting to connect.

We chose Bluetooth over wifi due to a few reasons:

Bluetooth	**Wifi**
Faster information relay timing	Connection Setup: 1) High Complexity 2) Time Consuming
Easier & intuitive setup	

The Bluetooth module that we are using is HM-11 and the specifications are as below:

- BT Version: Bluetooth Specification V4.0 BLE
- Send and receive no bytes limit.
- Working frequency: 2.4GHz ISM band
- Modulation method: GFSK(Gaussian Frequency Shift Keying)
- RF Power: -23dbm, -6dbm, 0dbm, 6dbm, can modify through AT Command AT+POWE.
- Speed: Asynchronous: 6K Bytes
  Synchronous: 6K Bytes
- Security: Authentication and encryption
- Service: Central & Peripheral UUID FFE0,FFE1
- Power: +3.3VDC 50mA
- Long range: Open space have 100 Meters with iphone4s
- Power: In sleep mode 400uA~1.5mA, Active mode 8.5mA.
- Working temperature:–5 ~ +65 Centigrade
- Size: HM- 10 26.9mm x 13mm x 2.2 mm; HM-11 18*13.5*2.2mm

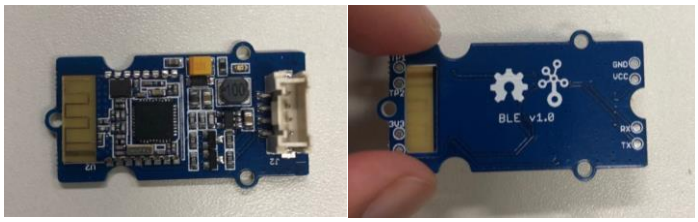


Fig 4.7 & 4.8: HM-11 Bluetooth module

1.4 Display Screen

We did a comparison between 2 screens- E-Ink and E-Paper screens. We settled on E-Paper screen due to its ability to display the last received image even after powering it down. This meant that it was perfectly suited to our needs in terms of energy conservation as it was able to keep our energy consumption down and reliably resolve the issue of the time taken before the image was cleared from the screen. The name of the display screen was Waveshare 4.3 inch e-paper screen.

Name	Colors	Grey Level	Resolution	Display Size (mm)	Outline Dimension (mm)	Full Refresh Time(s)	Partial Refresh Time(s)	Interface	Pi Header
4.3" E-Paper UART Module	Monochrome	4	800 × 600	88.00 × 66.00	118.0 × 75.0	1.5	N/A	UART	N/A

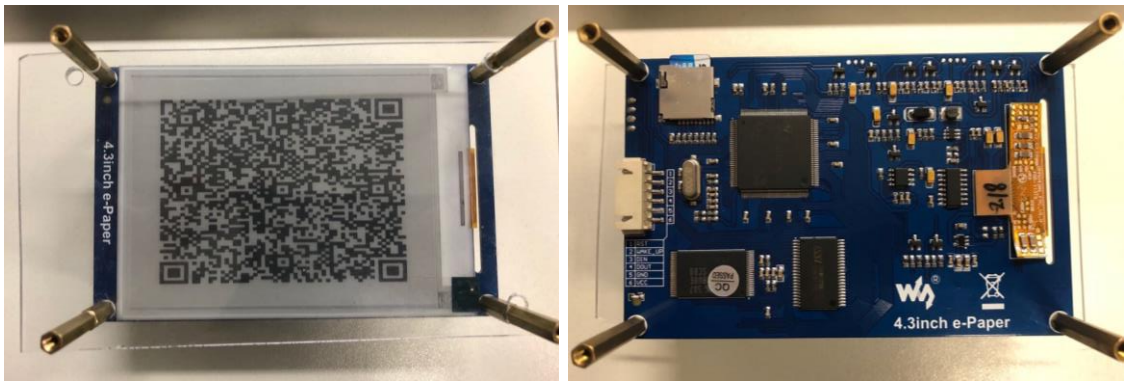


Fig 4.9 & 5: Display Screen with the QR code

However, due to the problem on the app side with sending over files more than the limit of 88 Bytes, we were unable to send the image over to be displayed on the display screen. This is the extent to which we stopped our project. We hope that the next batch will complete this section to transfer the image over from the app to the display screen and complete the whole project successfully.

Conclusion

Overall, when we finish the smart suitcase project, we were exposed to various platforms, technologies, APIs, gained more knowledge and insights which improved our programming abilities. However, we were unable to see this project to completion due to the limitations of Polidea's Bluetooth Library, which restricted our maximum data transmission capacity to 88 bytes. If there is a phase 2, we hope that the upcoming batch will be able to further improvise on our current technologies and bring the project to greater heights.

References

https://en.wikipedia.org/wiki/AAA_battery

https://en.wikipedia.org/wiki/AA_battery

https://www.google.com.sg/search?q=Battery+casing+to+arduino+for+CR123A&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjO1e_MpO3ZAhUDuY8KHXwWDhkQ_AUICigB&biw=1366&bih=662#imgrc=Buw12WhX5za9VM:

https://www.google.com.sg/search?q=how+to+solder+batteries+together&oq=how+to+solder+battries+to&aqs=chrome.1.69i57j0l5.9932j0j9&sourceid=chrome&ie=UTF-8

https://industrial.panasonic.com/cdbs/www-data/pdf/AAA4000/AAA4000COL23.pdf

https://industrial.panasonic.com/ww/products/batteries/primary-batteries/lithium-batteries/cylindrical-type-lithium-batteries-cr-series/CR123A

https://www.dfrobot.com/category-195.html

https://www.dfrobot.com/product-1218.html

https://www.baldengineer.com/five-wireless-modules-for-wireless-projects.html

http://www.st.com/en/wireless-connectivity.html

http://www.ti.com/product/cc2640r2f

https://www.dfrobot.com/wiki/index.php/Bluno_Mega_2560_(SKU:DFR0323)

https://www.dfrobot.com/product-1175.html?gclid=CjwKCAiA24PVBRBvEiwAyBxf-f3HwK5QvsbEvT3dbEAhs8vsTBv-86UD5PqtxJmaRDF8Tx3vFeO6kxoCS4wQAvD_BwE

https://www.waveshare.com/wiki/2.13inch_e-Paper_HAT#Working_with_Arduino

https://www.waveshare.com/wiki/4.3inch_e-Paper_UART_Module

http://www.suretorque.eu/all-smartloadcell/wireless-s-load-cell.html

https://blog.beamex.com/weighing-scale-calibration-how-to-calibrate-weighing-instruments

http://www.instructables.com/id/Arduino-Load-Cell-Scale

https://www.lazada.sg/digoo-smart-led-bluetooth-large-body-weight-fat-scale-monitor-app-400lb180kg-silver-intl-101020839.html?spm=a2o42.searchlist.list.37.34bc68eb1bw7n9

https://learn.sparkfun.com/tutorials/getting-started-with-load-cells?_ga=2.113922434.42553505.1520397039-172166054.1520231560

https://learn.sparkfun.com/tutorials/load-cell-amplifier-hx711-breakout-hookup-guide

https://create.arduino.cc/projecthub/MOHAN_CHANDALURU/hx711-load-cell-amplifier-interface-with-arduino-fa47f3

https://github.com/sparkfun/Load_Sensor_Combinator

https://electronics.stackexchange.com/questions/18669/how-to-wire-up-a-3-wire-load-cell-strain-gauge-and-an-amplifier

https://camo.githubusercontent.com/927cbcaa8d0ab2418db13ee40ebffee534c8df1b/687474707337a2f2f63646e2e737061726b66756e2e636f6d2f722f3630302d3630302f6173736574732f6c6561726e5f7475746f7269616c732f332f382f332f48583731315f616e645f436f6d62696e61746f725f626f6172645f686f6f6b5f75705f67756964652d30392e6a7067

https://www.reddit.com/r/arduino/comments/2ib6au/limiting_the_number_of_times_the_loop_repeats/

https://github.com/sparkfun/HX711-Load-Cell-Amplifier/blob/master/firmware/SparkFun_HX711_PowerTest/SparkFun_HX711_PowerTest.ino

https://www.youtube.com/watch?v=Y64Ev-pWqFs

https://www.youtube.com/watch?v=QsYcYknKbB0&t=1273s

http://www.batteryspace.com/prod-specs/cr123A_1300.pdf

https://www.waveshare.com/4.3inch-e-paper.htm

http://embedio.com.au/projects/WaveshareEpaper/index.htm

https://photogear.co.nz/panasonic-cr123a-lithium-battery-3v.html