Covid-19 Contact Tracing System Using MEAN Stack, Flutter Firebase Applications and NodeMCU Hardware

Neil Byrne

B.Sc.(Hons) in Software Development

May 9, 2021

Final Year Project - Covid-19 Contact Tracing System Using MEAN Stack, Flutter Firebase Applications and NodeMCU Hardware

Advised by: Mr. Gerard Harrison

Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)

Contents

Chapter 1

Introduction

1.1 Project Description

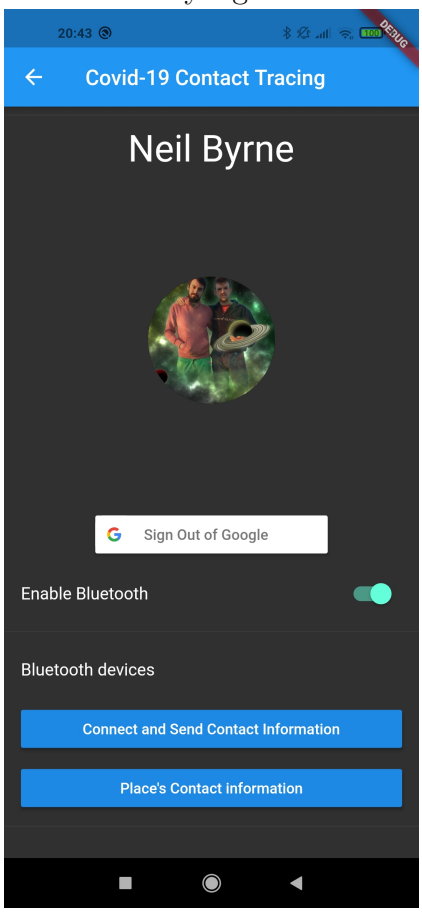
A Covid-19 Contact Tracing System to be used in the service industry and other businesses to track contact information of people in the case of a potential Coronavirus outbreak in the business establishment.

The system is broken down into three projects. An Android Flutter Application, connected to a Firebase Firestore for CRUD interactions of the user's contact information and Bluetooth capabilities to send the information. A NodeMCU ESP-32 module to retrieve the information via Bluetooth and parse it through HTTP requests. A MEAN stack web application to handle this contact information and perform CRUD operations, various filtering/sorting and serve as an email client.

1.1.1 Mobile

A Flutter Firebase application for the customer where they provide their contact information. The user opens the application and connects to the ESP-32 module via Bluetooth, types their contact information (first name, last name, phone number, email) and with a button press sends the contact information to the ESP-32. The user can also receive the placeID of the

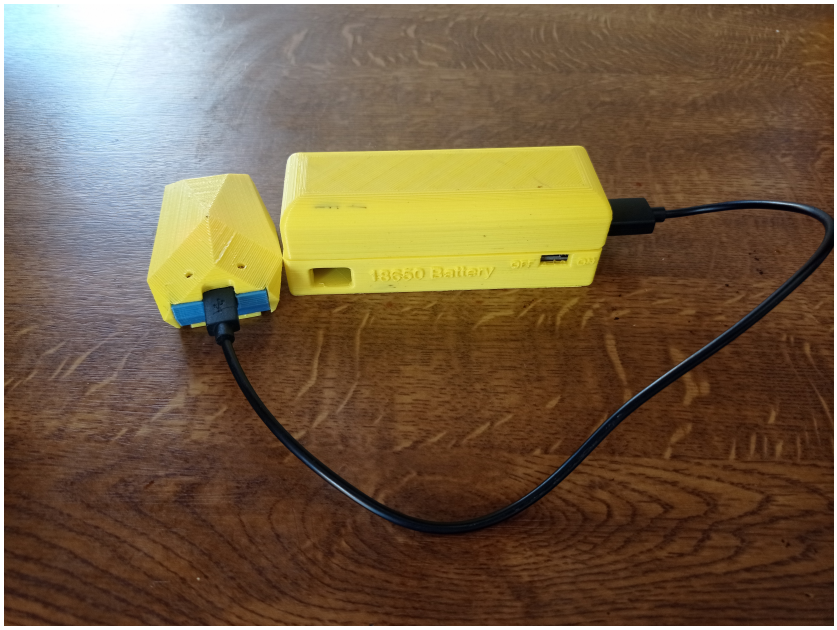
establishment from the ESP-32. This is then queried to the Google Services Place API for the business's contact information (name, formatted address, phone number, website). The app is connected to a Firebase Firestore so multiple user's contact information can be stored rather than typing it every time. The mobile app is developed with Flutter and the Dart language so it can be deployed to Android and iOS. Once the app is installed on a Android or iOS device it does not need any configuration work. The user can automatically sign in with Google or create an account.



1.1.2 NodeMCU

A NodeMCU ESP32 module is the middleman for communication of both apps. When a user's information is received via Bluetooth, a HTTP Post request will parse the data to the MEAN stack application. This information is also fed to a Telnet terminal on PuTTY via TCP/IP protocols. The Internet of Things (IoT) device is a cheap low powered Wi-Fi and Bluetooth enabled module and is the hardware the whole project was built around. Code can

be uploaded wireless and all serial communications can be monitored over the Telnet terminal.



1.1.3 MEAN Stack

A MEAN stack application for the business that must collect customer contact information. The app contains an organised list of all the people who have checked in. In the case of an outbreak in the business, the other people who were in the establishment that day can be searched. This provides a list of other people who may be affected as a result of the close contact. The email client can be used to then send a general email to all of the people at risk, informing them of the outbreak. This approach is much faster and more direct than the "pen and paper" method currently being implemented in businesses.

1.2 The Internet of Things

The Internet of Things (IoT) refers to a network comprised of physical objects capable of gathering and sharing electronic information. The IoT includes a wide variety of “smart” devices, from industrial machines that transmit data about the production process to sensors that track information about the human body.

1.2.1 How the Internet of Things Works

These devices use Internet Protocol (IP), the same protocol that identifies computers over the world wide web and allows them to communicate with one another. The goal behind the IoT is to have devices that self-report in real-time, improving efficiency and bringing important information to the surface more quickly than a system depending on human intervention.[1]

While "Internet of Things" is still very much a buzz word around technology there it no doubt it is has melded into everyday use. Smart watches and voice assistants are interesting to look into developing; by IoT nature they are to be explored to expand what devices communicate together and what they communicate about. My interests with hardware programming and IoT was the core idea of my final year project.

Chapter 2

Context

2.1 Project Motivation

Outside of college time I spent some time programming Arduinos and the various peripherals associated with it, so before I had an idea for a project I knew it would be based off some hardware. During college we studied a lot about IoT but we never got to play with the hardware itself. I saw this as an opportunity to further my interest in hardware and learn more about physically programming an IoT device. The ESP-32 looked to be a perfect fit for an IoT project given it's low cost and it's compatibility with the Arduino IDE and C++. NFC and RFID is used everywhere by most people. Due to it's very nature there is no extra thought about it when someone uses it to sign into work, connect to a Bluetooth speaker or even pay on your phone. The original motivation was to create a mobile app which uses NFC to communicate with the ESP-32, unfortunately due to hardware limitations discussed in the System Evaluation chapter, Bluetooth was used instead. While I now had motivation for what technology I would use. The kind of project to develop was not realised until I saw how contact tracing was being treated in businesses.

According to the HSE's Covid-19 guidelines, all restaurants, pubs and hotels must conduct contact tracing in the case of a Coronavirus outbreak in the establishment. While business are following the guidelines, the most common system for contact tracing is with pen and paper. Customers must recite aloud their full name, phone number and possibly email. There is no measure taken to ensure that this information written down is accurate or that it is properly stored. Data is the modern day currency, having this sort of personal information on display for anyone to copy or steal is in direct violation of General Data Protection Regulations (GDPR).

The Covid-19 pandemic demanded a radical change in our national infrastructure, how we work, learn and interact we each other is completely different largely because of technologies that were not available 20 years ago. Building a system which could take the place of "pen and paper" seems only logical and an interesting idea to make a project on.

2.2 Project Objectives

The main object of this project was to develop a more secure and easy to use system for contact tracing.

- **NFC and Bluetooth** Create an Android app that has NFC and Bluetooth capabilities. Learn how to parse information through with these mediums to other devices.
- **Registration** Allow multiple users to use the same app with their own data stored separately. Use the information in the registration to create a main page that is tailored to the user.
- **Places Contact** The user should be able to retrieve the contact information of the establishment by the same means they send contact information to the business. This information gives the user a timeline of where they have been.
- **Menu/Information** After sending contact information to the establishment, the user should receive a URL link or PDF of information. This information could be a menu for a restaurant or a digital pamphlet about the place.
- **Everything wireless** The ESP-32 should be packaged with any peripherals in a wireless manner. The ESP-32 should not need to be connected to a computer.
- **3D Printing** A 3D case should be printed for the ESP-32 and any peripherals. This will add a more enticing aesthetic to the project as a whole and give me the opportunity to learn about the process of rapid prototyping.
- **IoT** The NodeMCU should be the "middleman". There should be no direct communication with the Android and web app. All IoT features should be used to their fullest (Bluetooth, HTTP requests).

- **Date Filter** The list of contacts on the web app should be filterable by date. In the scenario of a Covid-19 outbreak, anyone who was in the establishment that day should be contacted. A list of contacts with no date reference makes contact tracing more difficult.

- **Email** After filtering contacts by date, each contact should have a button that enables the user to send a pre-written email to them.

- **Admin** There should be multiple users able to access the first and last name of the contact list. The admin should be the only one able to see the phone numbers and email addresses of the contacts.

2.3 Covid Tracker Ireland

The Irish Government and the Health Service Executive (HSE) released COVID Tracker Ireland on 7 July 2020 to a 862'000 download on the launch [2]. The app uses Bluetooth Low Energy (BLE) and the Exposure Notifications System developed by Google and Apple to generate anonymous IDs to log: Any user's phone that has the app installed and is within close contact. The distance between these two phones. The length of time the two phones were in close contact with.

The app downloads a list of anonymous codes provided by the HSE of people who use the app and have tested positive for Covid-19. The user will get a notification if they were closer than 2 metres for 15 minutes. The incorporation of Bluetooth is interesting as today Bluetooth is mostly associated with headphones and speakers. While Bluetooth file transfer speeds is 3 Mbit/s which is even slower than USB 1.1 which caps at 12 Mbit/s. Before the rise of smart phones, Bluetooth and Infrared was used to transfer files between phones. Now Bluetooth is much smarter, capable of mirroring a phone or laptop to a smart TV and tethering the network on one device to another. These less common uses remind us that Bluetooth is capable of excellent data transfers given a suitable task. Testing what Bluetooth can do on an ESP-32 that could provide some aid in combating Covid-19.

2.4 Outline of Chapters

The scope of the entire project and how each component works together. The development process is broken down into chapters.

2.4.1 Methodology

The methods the project was approached with for developing, testing and project management. The services and tools used during the development and reasons for their inclusion is described. This will give an overview of how the project grew and how it was managed to do so.

2.4.2 Technology Review

All technical aspects included in the project will be discussed. What these technologies do and why they were chosen over other technologies. Reasoning why other technologies were cut and how their inclusion would have fitted into the project.

2.4.3 System Design

The design and architecture of each component of the project, guides how to set-up, configure and deploy the Mobile, NodeMCU and Web projects. Code snippets and UI images demonstrates how the code works and the results from it. Errors encountered during development and the solutions to them as well as features that were dropped because of roadblocks.

2.4.4 System Evaluation

A reflection of the time spent developing the project and the end result. A critical analysis of its robustness, limitations and capability to scale further. Each objective set out in the this chapter will be revisited and the work done on them will be outlined.

2.4.5 Conclusion

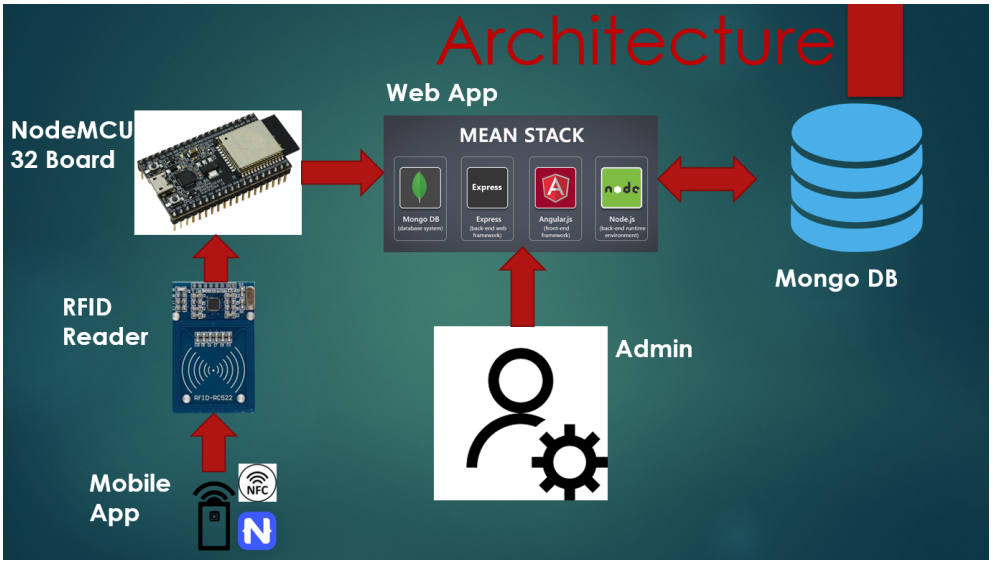
An overview of my thoughts on each of the projects of the system and of their respective software. Different lessons I learned with my time on a project so large. Plans for future development and personal final thoughts about my experience developing.

2.5 Structure of Project

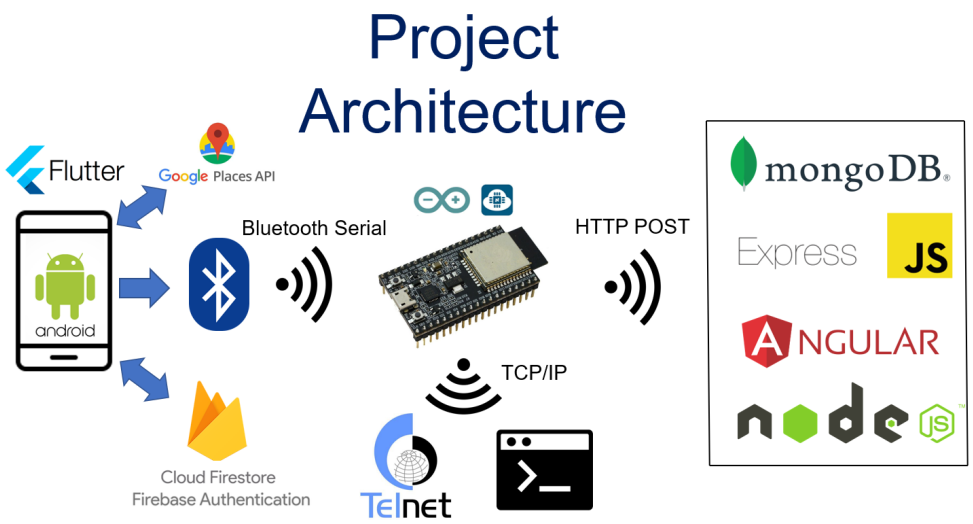
In December the proposal was sent to my supervisor Gerard Harrison. The proposal consisted of a PDF outlining my reasons for choosing this project, my work allocation to complete it and the technologies I plan to use. A short

presentation demonstrating all of these categories and high level view of the architecture was also presented. Below is the original architecture of the the project. Followed by the final complete architecture.

2.5.1 Original Architecture



2.5.2 Final Architecture



Chapter 3

Methodology

3.1 Overview of Methodology

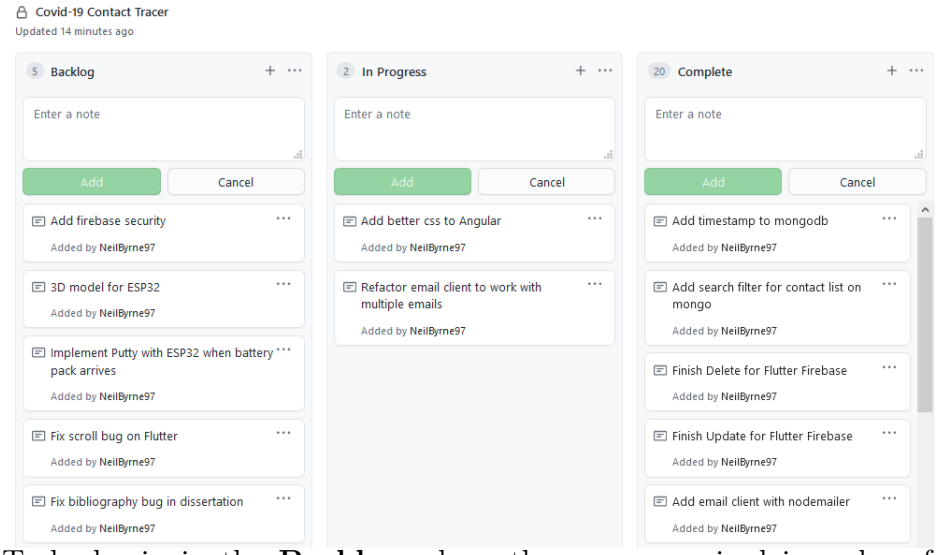
A methodology is an organised process to plan and control the development of software. Rapid Application Development (RAD), Extreme Programming (EXP) and Waterfall are various methodologies to chose from but they all lack flexibility. There was a lot of technologies in this project that I was not familiar with. There would be some learning curve to be taken into account. So an Agile methodology was implemented to be supported with Test Driven Development (TDD) and Unit testing.

3.2 Agile

Agile is an iterative approach to project management and software development encouraging continual improvement and incremental delivery of software. SCRUM and more Kanban are variations of Agile that all follow the same 12 principles outlined in the Agile Manifesto. For the purposes of this project I will be using Kanban.

3.2.1 Kanban

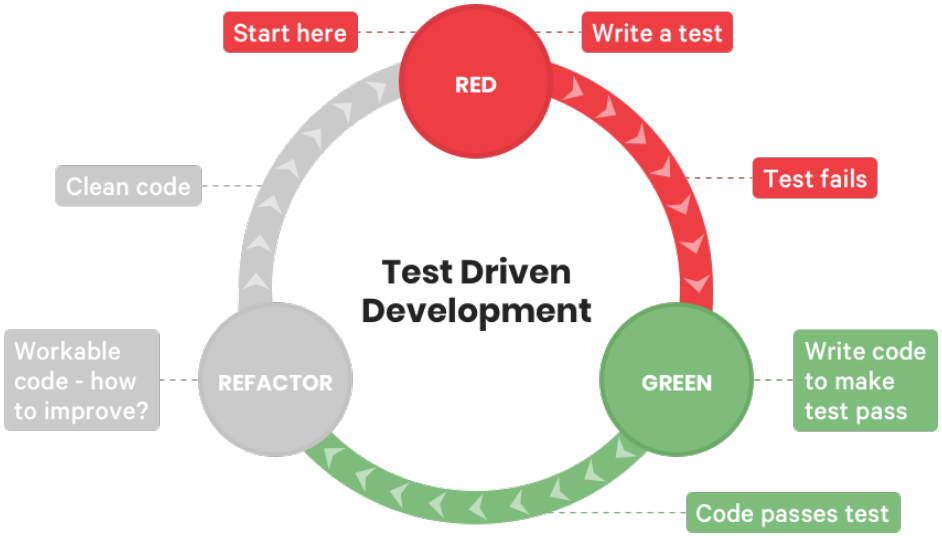
Kanban is a workflow management method that aims to support visibility in work, increase efficiency and improve continuously. A Kanban board is a visual tool to know what has to be done, what is being done and what is completed. More columns can be added for any given project. Having three separate projects to remember can be difficult, this visual tool gives a sense of what stage each project is in. GitHub has tools to create a Kanban board to organise workflow.



Tasks begin in the **Backlog** where they are organised in order of importance. When I decide to start on a task, it is moved into **In Progress**. It will remain there until it is completed and finally moved to **Complete**. The Kanban board helped create a visualisation of workflow which helped productivity and allowed me to prioritise what tasks needed to be complete.

3.2.2 Test Driven Development

Test Driven Development (TDD) is a form of testing involving coding, testing and refactoring. Tests are written for a function before any coding. Write enough code to make sure it passes the test and then refactor the code until it conforms with what is desired.



TDD is an approach I never took with any other project before. Initially it was tempting to start writing code when starting a project but I found the approach to awarding in the long run. As the codebase becomes more complicated some bugs can lay dormant until they are too integrated into the system. TDD exposed many of these bugs at an early stage.

3.2.3 Meetings and Presentation

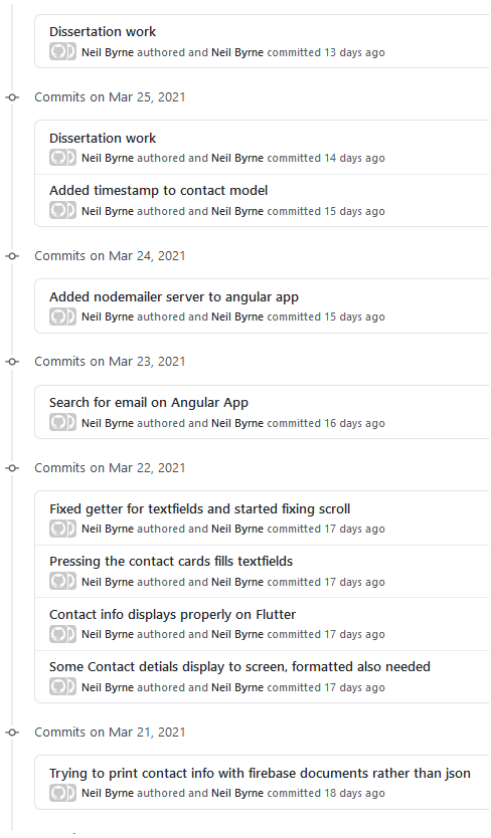
Throughout the year weekly meetings were held between myself, my supervisor Mr. Gerard Harrison and fellow students. These meetings served as a general progress report and any followed by a QA with Mr. Gerard Harrison. There was some overlap in the project ideas being developed which led to some early problem solving. The timely meetings also provided a chance for me to review my Kanban board and update it.

3.3 Version Control

3.3.1 GitHub

GitHub is code hosting platform for version control and collaboration. Everything involved in the project was included in the repository. Source code from all of the projects implemented or unimplemented, dissertation work,

images, presentation slides etc. Github has two primary functions for development, tracking code changes and the security of having a back up of everything in case of any issues with my local machine. In Subsection 3.2 I discuss how Github also contributed to the management of the project with some of it's tools.



3.4 Testing

For the most part TDD was a continuous part with the development. In Notepad I brainstormed the type of tests I wanted to take. Unit testing was performed on the Flutter and MEAN app. As of today, there is no well supported way to create Unit tests for Arduino Studio. There was no indication that this would cause an issue because there is little changing logic in the code to arise any error. The serial monitor often is enough to debug. Unit tests are useful for verifying the behaviour of a single function, method or class.

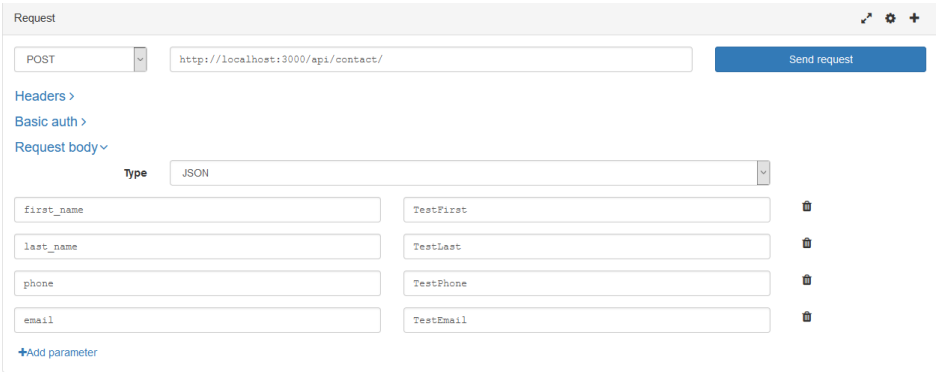
3.4.1 Flutter Unit Testing

Flutter's package manager contains a Test library to test code written in Dart. After including the dependency, the tests just need to be written and run.

```dart
import 'package:atyun_flutter/environment.dart';
import 'package:flutter/material.dart';
import 'package:flutter_test/flutter_test.dart';
import 'package:atyun_flutter/components/button.component.dart';


void main() {
  testWidgets('should create button success', (WidgetTester tester) async {
    final testkey = Key('test');
    final button = Button(
      'TestButton',
      key: testkey,
    );
    await tester.pumpWidget(Container(
      child: button,
    ));
    expect(find.byWidget(button), findsOneWidget);
  });

  testWidgets('Environment should be singleton', (WidgetTester tester) async {
    final env1 = Environment(appId: 'a');
    final env2 = Environment(appId: 'a');
    expect(env1.appId, equals(env2.appId));
  });

}
```

3.4.2 RESTED Tester Add-On

API tests are important early in development to ensure that a MEAN app is functional. RESTED was used, a powerful HTTP client support add-on for Firefox similar to Chrome's Postman. It has an easy to use query designer for writing test cases, managing data and measure response time for effective testing and management of test cases.



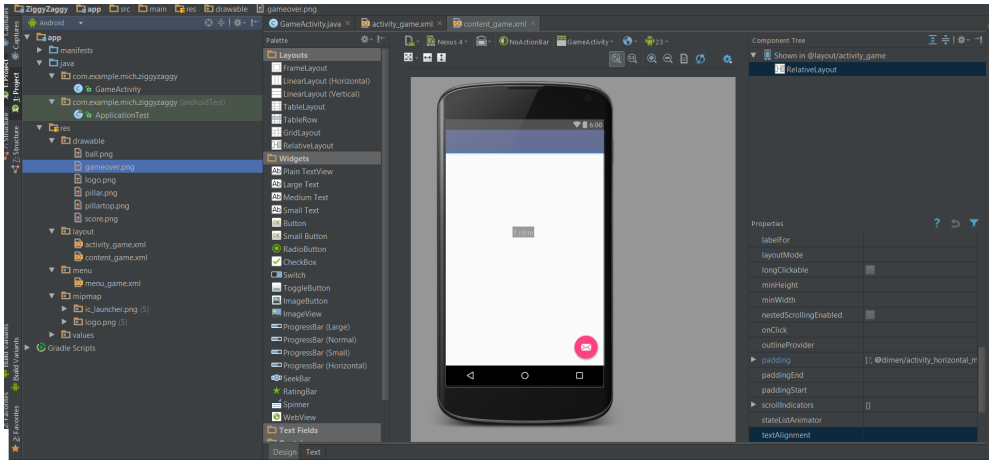
The ability to create and save HTTP requests and most importantly read their responses helped narrow down where issues could be. At the early stages of development the RESTED add-on was used to test the MongoDB server.

Chapter 4

Technology Review

4.1 Android Development

Out of all of the technologies used to develop this system, the Android project is the one I had no experience in. I was not aware of the IDEs, frameworks or languages used but was excited to learn how to develop on Android. It is also the project which I refactored the language several times early in development. Initially I was going to develop two Angular apps, one for the web and one for mobile and use NativeJS framework to build the mobile app. However I would be using hardware on the mobile, NFC/Bluetooth, these sensors must be programmed directly. NativeJS simply allows an Angular app to run natively on mobile and would not allow access or functionality to the sensors. Java and Kotlin are the two languages I used to develop a NFC app. The documentation on Android's developer site provided code snippets for both languages [3]. Both languages have years of development behind them and the UI layout editor is a intuitive way to design the UI without messing the style. However, when the project moved to used Bluetooth instead I discovered Flutter to be the best way to do this. Although none of the codebase from the Java or Kotlin projects were used for the Flutter project, the time spent developing them made me more comfortable with the IDE and Android as a whole.



4.1.1 Flutter

Flutter [4] is an open-source UI development kit created by Google and initially released May 2017. Flutter uses the Dart programming language also developed by Google to create apps that can be deployed to Android, iOS and as of recently with Flutter 2 to web from the same codebase. While for this app a web counterpart would not work, it is good to see this new way of building apps: Putting the design first while the app runs natively. Flutter and Dart are relatively new but thanks to their backing with Google it can be sure that more developers will be using them in the future. The packager manager pub.dev has lots of libraries to get an app started [5].

The flutter-bluetooth-serial package supplied an easy to use bluetooth messaging app to be used between two devices. This code is easily understood and to develop on top of so I did not have to "reinvent the wheel".

4.1.2 Android Studio

Android Studio [6] works well with similarities to VisualCode. Its ability to emulate Android or iOS devices make it quick to debug with. Rather than use an emulated device, I debugged on my own mobile phone, a Xiaomi Redmi 4 by turning on the developer options. It's also important to note that a smart phone was needed to test the Bluetooth capabilities.

4.1.3 Gradle

Gradle is a build automation tool for several languages. Android Studio will automatically sync the gradle file and build the project.

4.1.4 Pub.dev Package Manager

Flutter package manager with a growing community for the Dart language. The **pub get** command will install any packages for Flutter.

4.1.5 Libraries

The four following packages are all required for Firebase integration.

firebase_core

Plugin to use the core API for Firebase enables to connect to multiple Firebase apps.

firebase_analytics

An app measurement solution that provides insight on app usage and user engagement. When setting up the Firebase Firestore you can opt out to installing this plugin. I ran the app on three different Android phones which is visualised on the Firebase console.

firebase_auth

A versatile password authenticator using passwords, phone numbers and identity providers like Google, Facebook and Twitter.

cloud_firestore

Flutter plugin for Cloud Firestore, a cloud-hosted, NoSQL database with live synchronization and offline support.

cupertino_icons

Default icons asset for Cupertino widgets based on Apple styled icons, used for Send, Save, Update, Delete buttons.

flutter_bluetooth_serial

A basic Bluetooth serial library for Classic Bluetooth implementation. Provides starter code for basic serial connection.

scoped_model

A Widget that passes a Reactive Model to all of it's children. Handles the Animation and Text Editing Controllers.

flutter_signin_button

Generates sign in buttons for different social media account including Google, Facebook and Github. Google sign in is used for this app, retrieving user information to tailor a more personalised experience.

google_maps_webservice

Allows integration with Google services such as Maps, Routes and Places through the various APIs.

http

A set of high-level functions and classes to consume HTTP resources.

get_it

Service locator to decouple the interface from a concrete implementation and to access the concrete implementation from anywhere in the app. Required to access phone call and SMS functions of the phone.

Flutter Bluetooth Serial

Flutter Blue [7] is a more popular Bluetooth library for Flutter but much heavier. Rather than adding functionality for the sake of more functionality I used Flutter Bluetooth Serial [8]. The starter code comes complete with a messaging terminal to send messages to another connected Bluetooth device.

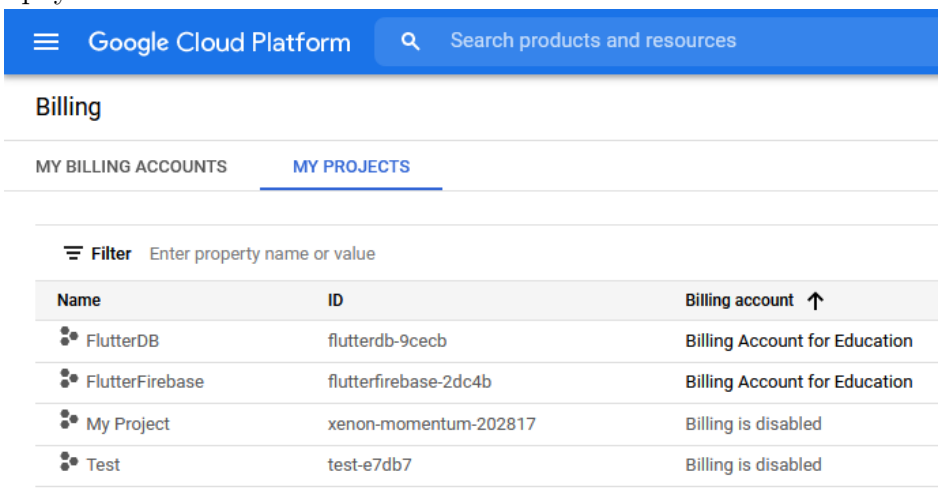
Firebase Firestore

Some configuration is needed to link to a Firebase database. The documentation provides a detailed guide on how to apply the plugins on every platform. Plugins are available for real-time and cloud Firestore depending on the type of database needed. A Firestore instance has methods to perform actions on Collections or Documents.

4.1.6 Firebase

Having finally chosen a front-end I was happy with, I decided to fully subscribe with the Google ecosystem for my mobile app. Although I looked into using AWS Amplify [9], having an app that is solely based off Google products creates an opportunity I never had in the course. Of course, integration with Flutter is simplified and documentation is more widely available.

During development I received emails from Firebase informing me that my free trial was expiring and unless I upgraded from the free tier I would not be able to modify my data. Fortunately, my lecturer Ian McLoughlin got Google Platform coupons for use in our projects. This $100 coupon allowed me to continue with the databases and create new ones under the new payment scheme.

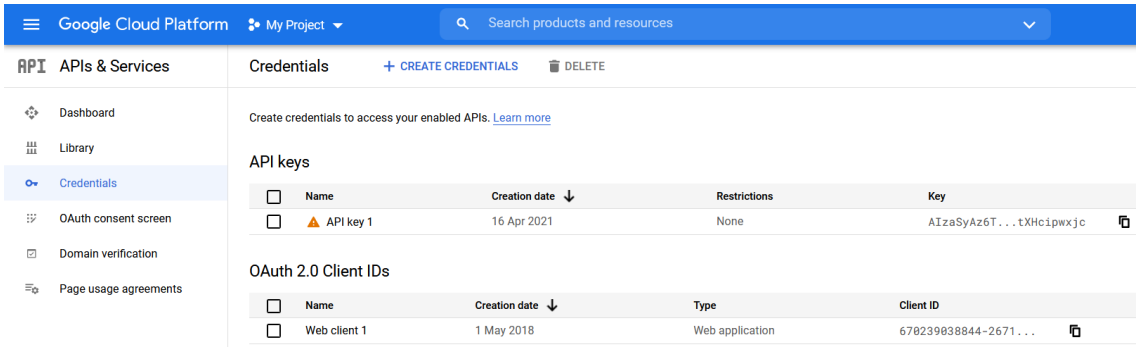


4.1.7 Google Maps Platform

The Google Maps Platform [10] is a set of APIs and SDKs that allow developers to embed Google Maps features in their app. APIs for Maps, Routes and Places are all available but for this project only the Places API was used. The Places API is a service that returns information about places using HTTP requests. Places are defined within the API as establishments, geographical locations, or prominent points of interest. To access any of the Google API's you need an API key which can be generated for free on the Google Console page.

Generated API key for Project:
AIzaSyAz6TJpPOpuhahblOebTaiCmtXHcipwxjc

Restrictions can be applied to keys to block requests with this key from any websites or apps declared. This key was only used for development not production so there are no restrictions.



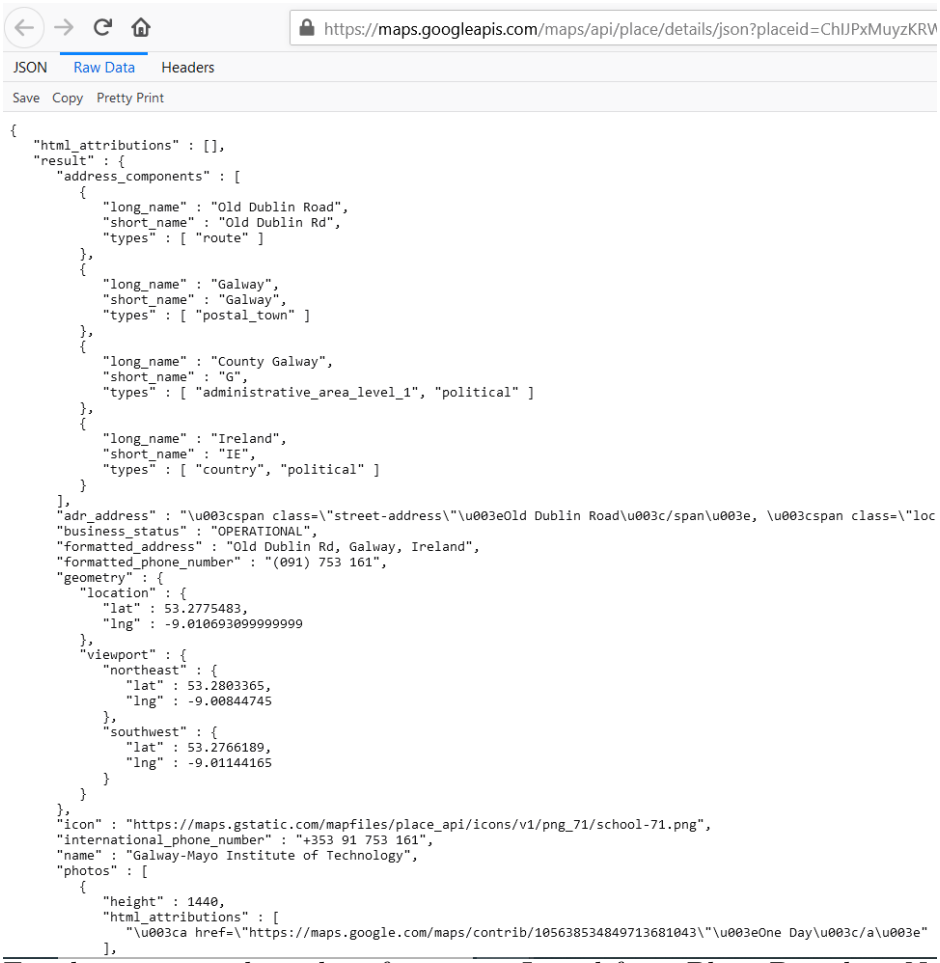
Place Details

Each location on Google Maps has it's own unique ID that can be queried for more information about the location.

GMIT's Place ID: ChIJk5qFk8iWW0gRmwunZPv7a_w

With an API key and Place ID, all the Google Maps data can be retrieved by putting the credentials in the URL format below.

https://maps.googleapis.com/maps/api/place/details/json?placeid=PLACEIDkey=APIKEY



For this project, the only information I used from Place Details is Name, Address and Phone Number.

4.2 NodeMCU Development

A project implementing some sort of an IoT device was the intention from the beginning. Although we do not cover any Arduino or micro-controller programming in the course, I had some of my own experience with an Arduino Uno and programming in the Arduino IDE.

NodeMCU is an open-source firmware and development kit that helps to prototype or build IoT products. It includes firmware that runs on the ESP-8266 and ESP-32 WiFi-SoC from Espessif System, and hardware which is based on the ESP-32 module. The firmware uses the Lua scripting language, programming the board itself is done with C++ and in the Arduino IDE similar to Arduino. Although development began on a ESP-8266 board it

moved to an ESP-32 which proved crucial when the primary function moved to Bluetooth.

4.2.1 Telnet

Telnet is an application protocol for bi-directional interactive text oriented communication via a virtual terminal connection. It is still prominently used for RF terminals, bar code scanners and other data collection devices. This is the solution to not having a serial output for the ESP-32. The ESP-32 must be plugged into the computer for serial output but this data can be accessed through this way instead.

4.2.2 Arduino IDE

The Arduino IDE is a cross platform app written in C and C++. It allows us to write and deploy code for Arduino compatible boards and other vendor boards such as the ESP-32. The Arduino IDE is very basic, since most the scripts wrote on it are single page this is okay. Common problems encountered developing on hardware often have to do with connecting to the board itself. A USB cable with a data wire must be used to transfer the script to the board and for serial output.

4.2.3 Arduino Library Manager

The Library Manager is lightweight but contains most of the libraries anyone would need for Arduino development.

4.2.4 Libraries

Bluetooth Serial

Provided by Arduino, establishes a Bluetooth client which can use the Serial Input to parse data. This library does little work than wait for a connection to be available, it is the Flutter library which does the heavy lifting.

WiFi

Built in by Arduino, establishes a network connection (local and Internet) using Arduino WiFi shields or in this case using the WiFi chip in the board. This library is the driving force of the ESP-32, without it the board would be redundant. Capable of instantiating Servers, Clients and send/receive User Datagram Protocol (UDP) packages through Wi-Fi.

HTTP Client

While the WiFi library will create a connection to a network, this library is used to perform HTTP protocols GET, POST, UPDATE and DELETE (CRUD) to a web server. Along with Arduino JSON library we can specify a URL from local or the web and send requests.

Arduino JSON

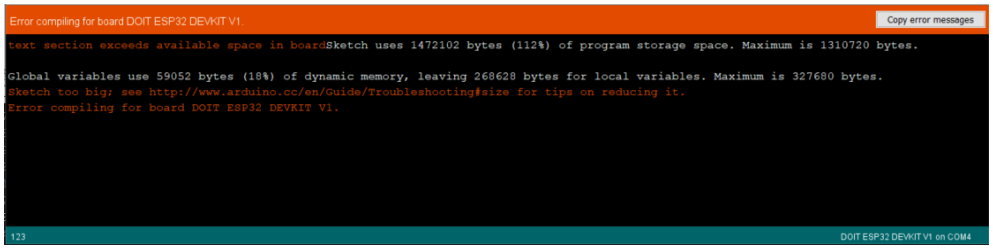
The only third party used on the ESP-32, a more simplified and efficient library to deal with serialization, deserialization and streams. In fact, it accomplishes these things so much better than it's first-party counterpart that it is the most popular Arduino libraries on GitHub.

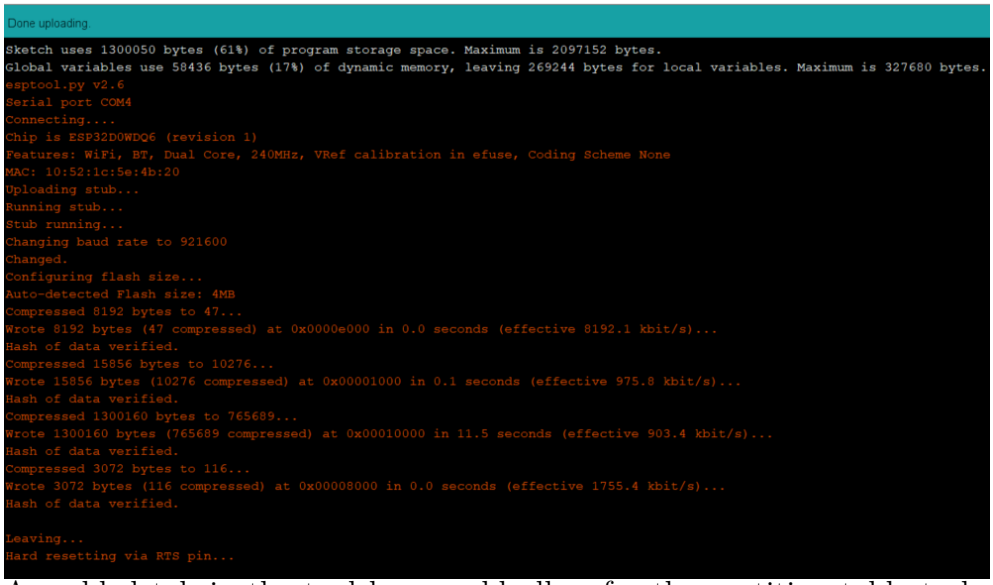
RFID RC522

While subsequently being commented out from the project, the Arduino community have a very simple library to use RFID peripherals. Once the hardware is wired and starter code is ran Read/Writes work instantly with RFID tags and cards. I am hopeful that when NFC peripherals become available, the community will continue to contribute and innovate.

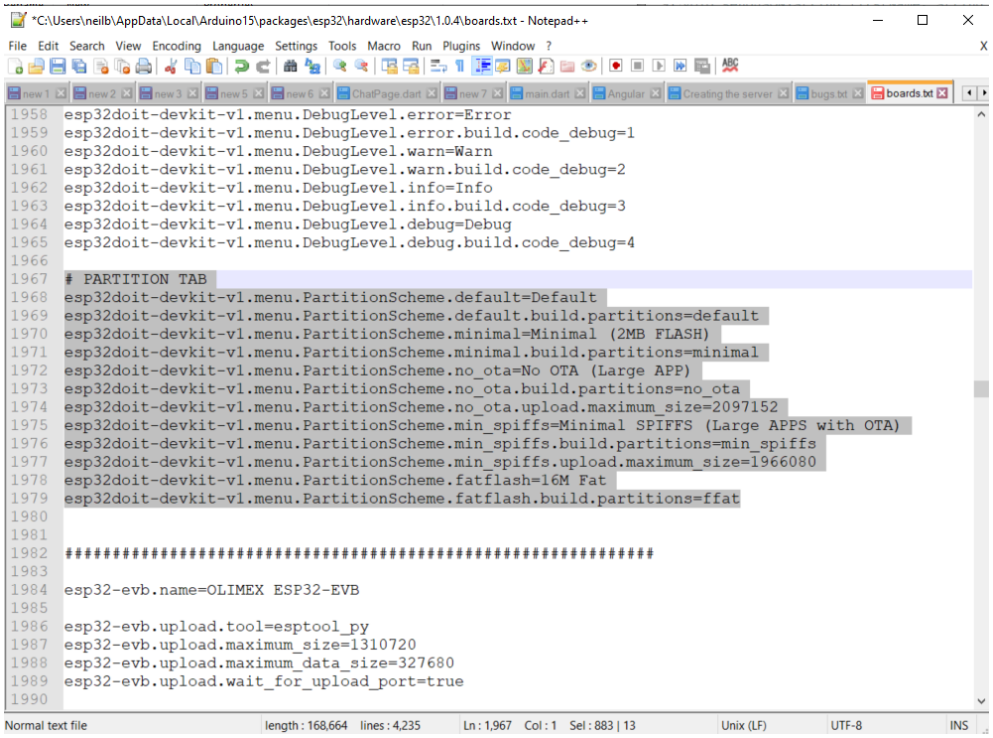
4.2.5 Memory Partitioning

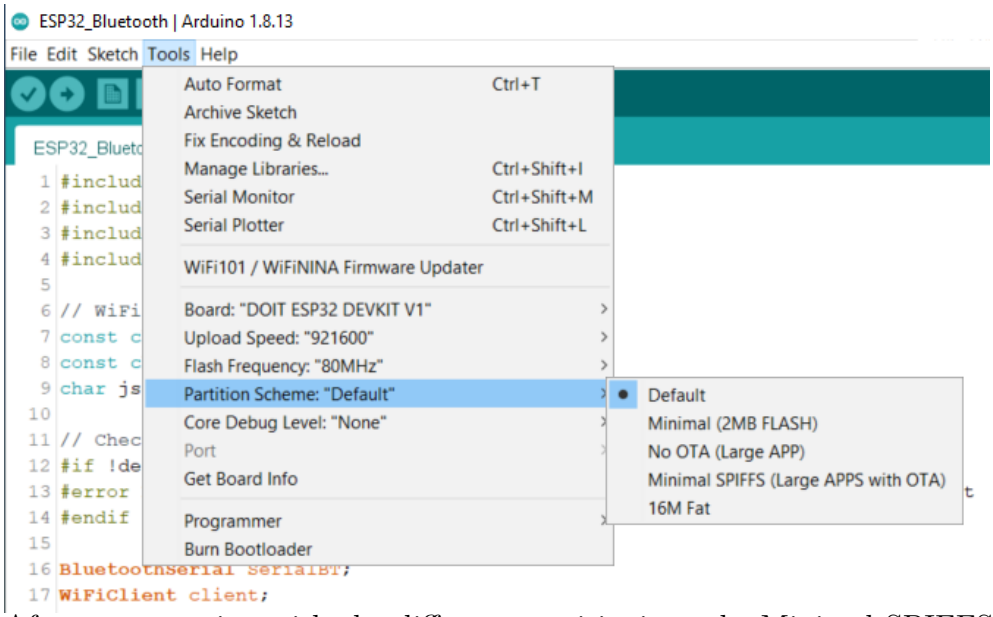
After including BluetoothSerial and HTTPClient libraries the ESP-32 ran out of memory. Despite the IoT capabilities of the ESP-32, it's on board memory is only 4MB. The number of lines of code was too much to be compiled. After some research I discovered this to be a common problem for complex projects. With some added configuration to the board settings, the memory partitions can be altered.





An added tab in the tool bar would allow for the partition table to be altered. After this change in board configuration all the code code be compiled and uploaded.



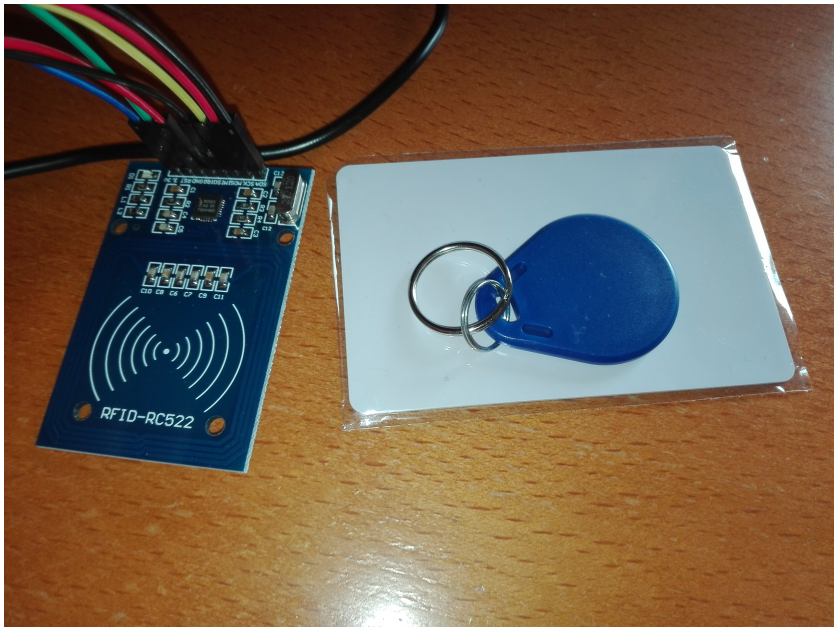


After some testing with the different partitioning tabs Minimal SPIFFS (Large APPS with OTA) was the only one to compile and upload all of the script to the ESP-32.

```
Done uploading.
Sketch uses 1300050 bytes (61%) of program storage space. Maximum is 2097152 bytes.
Global variables use 58436 bytes (17%) of dynamic memory, leaving 269244 bytes for local variables. Maximum is 327680 bytes.
esptool.py v2.6
Serial port COM4
Connecting....
Chip is ESP32D0WDQ6 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
MAC: 10:52:1c:5e:4b:20
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 921600
Changed.
Configuring flash size...
Auto-detected Flash size: 4MB
Compressed 8192 bytes to 47...
Wrote 8192 bytes (47 compressed) at 0x0000e000 in 0.0 seconds (effective 8192.1 kbit/s)...
Hash of data verified.
Compressed 15856 bytes to 10276...
Wrote 15856 bytes (10276 compressed) at 0x00001000 in 0.1 seconds (effective 975.8 kbit/s)...
Hash of data verified.
Compressed 1300160 bytes to 765689...
Wrote 1300160 bytes (765689 compressed) at 0x00010000 in 11.5 seconds (effective 903.4 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 116...
Wrote 3072 bytes (116 compressed) at 0x00008000 in 0.0 seconds (effective 1755.4 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

4.2.6 RFID-RC522

Initially the project specification used a RFID peripheral for the ESP-32. RFID seemed it would be a perfect fit for parsing data from an Android app to an IoT device. Given that the parsed information would occupy little memory. Although the reader worked well for ID tags and cards implementing it in the project proved not to be possible.



Radio Frequency Identification (RFID)

RFID was first developed in the 1980s to improve on the barcode system used to track inventory in all types of businesses. Similar to barcodes, RFID store and transmit identifiers (ID), but does not require line of sight to scan the ID. While RFID serves a purpose for inventory tracking and simple key tags, it does not have the ability to transmit a large string of formatted data and thus not suitable for this project. Through hard coding ID values to a database of contact information onto a RFID card or tag is a possible way to make this type of project work with this system. The goal was to develop a mobile app without any the requirement of tags for a better user experience.

Near-Field Communication (NFC)

In 2002, NFC was created by Nokia, Phillips and Sony [11]. Allowing contactless data transfer on low powered mobile devices. NFC uses radio frequency waves like Wi-Fi and Bluetooth hence it's involvement in the IoT ecosystem. NFC can be integrated into most hardware to enhance the experience. NFC's conjunction with Bluetooth pairing on speakers is another great example of IoT in our world. Modern smartphones now feature NFC card emulation allowing for contactless payment. NFC card emulation and NFC reader templates in Java and Kotlin written Android apps are widely available on **Github**.

Even though the use for the mobile app this was promising, a peripheral to let the ESP-32 read this parsed information was not available. Hardware restrictions of the RFID and NFC forced a revaluation of the project.

There are NFC readers on the market but most come pre-installed with software and are not compatible with the ESP-32. New peripherals and software is being developed for ESP-32 and other IoT devices so perhaps in a further iteration of the hardware it may be possible to create a project of this design.

4.3 MEAN Stack Development

The main reason that the MEAN stack was chosen is because of my familiarity with it during this course and it provides a framework for a full stack application. MEAN stands for Mongo - Back-end database. Express - Communication between front-end and back-end. Angular - Front-end. NodeJS - Provides the server and package manager.

4.3.1 Visual Studio Code IDE

Visual Studio Code is the lite version of Visual Studio IDE. All the bells and whistles are striped away to leave room for what is most important, the code editor. To its core that's all it is, an editor. Extensions for different languages or features can be added as needed which is why many developers prefer the lite version. Personally, it is my go-to code editor after having used it for programming C++, C#, Python and Typescript. A powerful IntelliSense, git support, multi-windowing, navigation, npm package manager support, and even a debugger make it well rounded for most programming needs.

VS Code Extensions

4.3.2 Node Package Manager

Node Package Manager (npm) is the package manager for the Javascript runtime environment for Node.js. Dependencies can be installed with **npm install** in the command line. Express will be installed this way when setting up a MEAN stack app.

4.3.3 Libraries

angular/cli

Command line interface tool to initialize, develop, scaffold and maintain Angular apps directly from a command shell. **ng new** builds a new Angular app that will run. **ng generate** generates components, routes, services and pipes and create test shells for them. **ng serve** will deploy the app locally.

body-parser

Body parsing middware. It is responsible for parsing the incoming request bodies in a middleware before you handle it.

cors

Cross-Origin Resource Sharing (Cors) allows restricted resources on a web page to be requested from another domain. Required to make HTTP requests from localhost:4200 (**Angular**) to localhost:3000 (**Mongo**).

express

Required to run express as middleware for the stack. Express.js serves as the de facto backend framework for Node.js.

jquery

A tool to simplify HTML Document Object Model (DOM) tree traversal and manipulation.

mongoose

MongoDB's object modeling tool designed to work in an asynchronous environment.

nodemailer

Simple email client used in app to send emails to selected contacts from the connected gmail account.

nodemon

Extremely useful tool that automatically restarts node app when a file change in the directory is detected.

4.3.4 MongoDB

Similar to Firestore, MongoDB is a NoSQL Document based database that can store JSON documents created by Angular or directly from Express. For new developers or large companies MongoDB is very popular because of it's flexibility and ease of use. It leaves much of the responsibility about the data model to the developer. This would suggest there is more work required. Rather the opposite is true. MongoDB will not breakdown if the developer wants to add a new table entry or a new Document that conflicts with the schema. This flexible data model lets the developer write an app and not force it to fit a set schema. For this app all the contact information (first name, last name, phone number, email) is stored to the database in JSON format.

4.3.5 ExpressJS

ExpressJS is a web framework built around Node.js that provides a set of features for web and mobile development. ExpressJS will set up and handle

the server and routing. This is a huge time saver for the developer. Components from other frameworks such as Angular, React and Vue can also be handled, processed and updated. The ease of use and integration with other frameworks makes ExpressJS a powerful full stack framework.

4.3.6 Angular

Angular is a front-end web framework used to develop Single-Page web apps. Single Page web frameworks (Angular, React, Vue) are dynamic pages that, instead of leading the user through multiple pages, will update and rewrite the current web page. This means the web page does not refresh, the content on the page just changes. This creates a more enjoyable, fast and fluid experience. Angular was first release in 2010 as AngularJS. But in 2016, the whole framework was overhauled and rewritten. Within all full stack frameworks, the front-end is the most mutable. While React and Vue are equally good. Angular is the most mature framework and in recent years has grown in popularity (since the overhaul in 2016). Angular integrates with ExpressJS and Node.js very well allowing for rapid prototyping. Originally for the mobile Android app, I wanted to create it with Angular implementing NativeJS. But due to the inability access the phone's I/O hardware forced a shift in framework.

4.3.7 NodeJS

Since NodesJS's inception in 2009 it has been the de facto Javascript web server for developers. NodeJS is a cross-platform runtime environment that allows JavaScript to run outside the browser. Its vast library of modules provides functionality that can be easily incorporated. The most popular feature for NodeJS is its ability to preform HTTP requests. This means it's entirely possible to develop an app that uses JavaScript for the front-end and back-end. Thanks to the HTTP request capabilities and it's versatile package manager, the process of just building apps is swift and of a high standard.

4.4 Wi-Fi

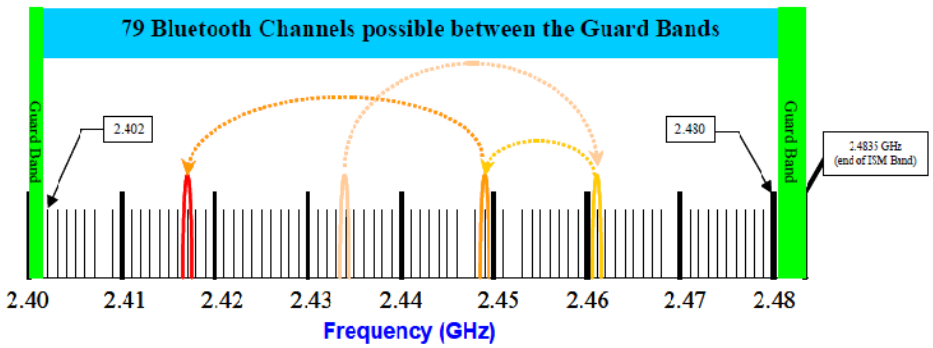
In 1991, NCR Corporation and ATT invented the precursor to 802.11 for cashier teller systems. After years of development the first 802.11 protocol was released providing 2 Mb/s [12] . In 1999 the 802.11 WiFi standard was updated to 11 Mb/s link speeds across the 2.4GHz and 5GHz bandwidth.

This coincided with the release of Apples iBook which featured the Wi-Fi technology, popularising it at the time. Today Wi-Fi is built into any smart device and is ubiquitous with the internet. A WiFi router is connected to the internet via physical connection. End devices such as phones, laptops, tablets can use this WiFi connection to use the internet connection provided by the router [13].

The ESP-32 WiFi module can connect to a known wireless network and perform HTTP requests to a local server or one deployed on the web with the HTTPClient library.

4.5 Bluetooth

Ericsson Mobile developed the **"short link"** radio technology Bluetooth in 1989. But the first consumer Bluetooth device was released in 1999 [14]. Bluetooth uses radio waves to transport data between short distances similar to Wi-Fi. They both operate at 2.4GHz frequency but Bluetooth uses much weaker signals (around 1mW power) at much shorter distances. Bluetooth is much better at connecting to multiple devices at once that don't interfere with each other because of 79 designated Bluetooth channels. A radio technology known to have been used since 1903 called Frequency-Hopping Spread Spectrum (FHSS)[15] is used. The flavour used by Bluetooth called Adaptive Frequency Hopping (AFH) [16], divides the data into packets and transmit them across each of the 79 channels. These channels are switched between 1600 times per second.



The ESP-32 features Bluetooth 4.2 and Bluetooth Low Energy BLE. Since the Android app will be sending a continuous data stream Bluetooth 4.2 will be used with the Bluetooth Serial library in the Arduino IDE.

4.6 Rapid Prototyping

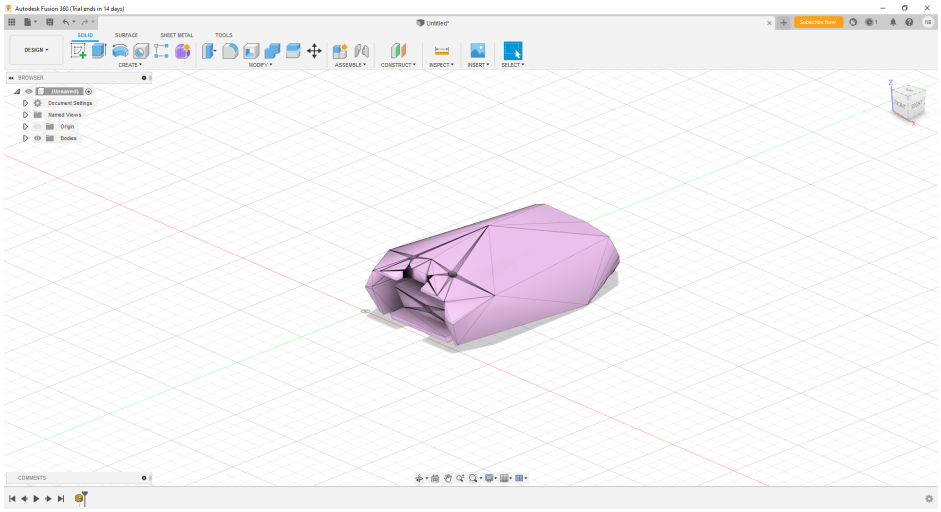
Rapid Prototyping is the fast fabrication of a physical part, model or assembly using 3D computer aided design (CAD). In this fast-moving modern-day consumer market, companies need to develop and introduce new products faster to remain competitive. Since faster product development and technology innovation are key to a company's success, rapid prototyping becomes the most important element of new product development. The following objectives are achieved through rapid prototyping [17].

- Prototyping plays a vital role in the process of creating successful products because it speeds up the new product development process.
- Early stage design/concept validation of form, fit, and function of the design.
- Final stage product verification against the technical requirement and business objectives.
- It allows functionality testing to test the objectives of the concept and to finalise the specification.
- Prototype gives the end user, client, customer, user participants hands-on user experience to get feedback.

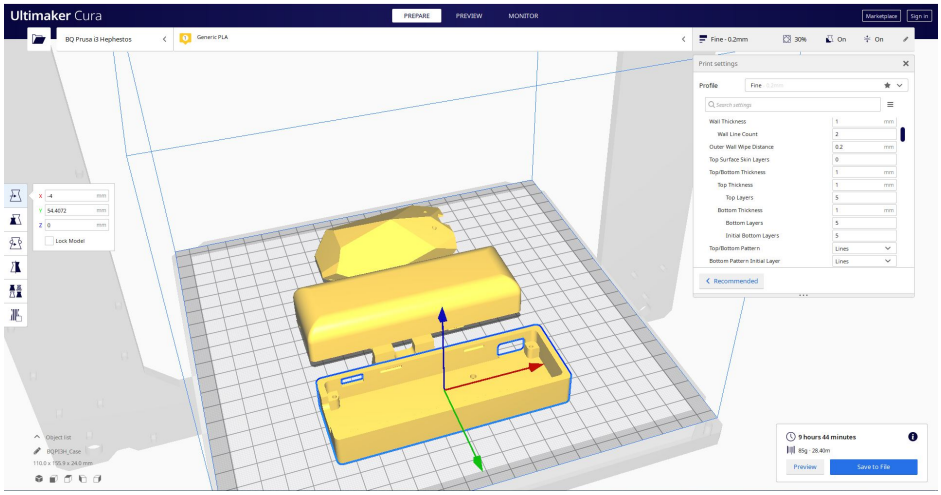
A designer has a variety of prototyping technologies available. Each of them has their benefits and drawbacks and some are specialised to a particular job. This project uses Fused Deposition Modelling (FDM) to print a case for the ESP-32 and battery module.

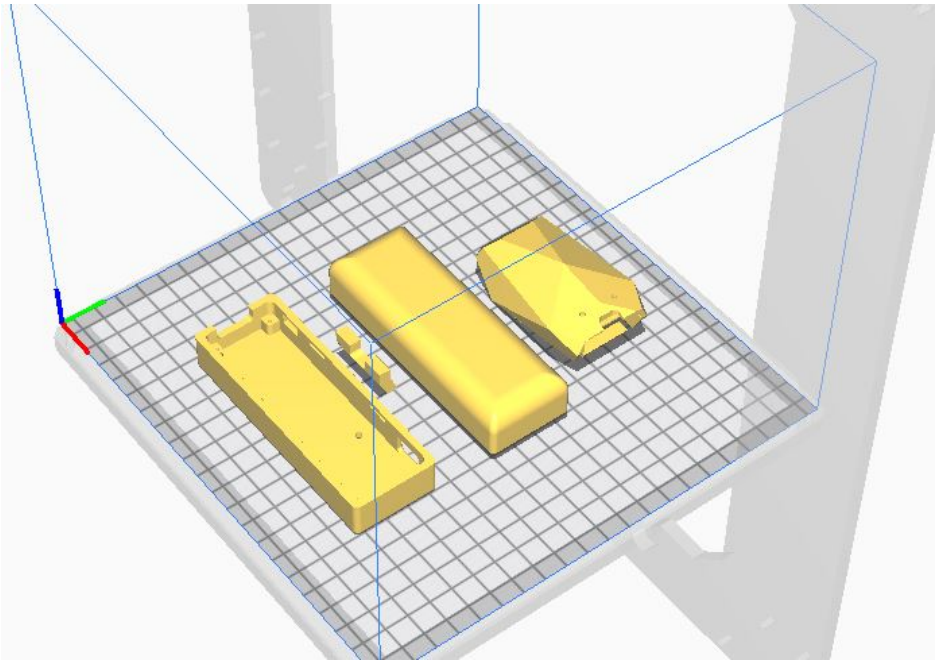
4.6.1 3D Modelling

Before any printing can start, a 3D model must be created. **Thingiverse** is an excellent website where people share their 3D models. For the battery pack, a model found on the site was used [18]. But for the ESP-32, a custom design was created in AutoDesk Fusion 360 [19]. AutoDesk Fusion 360 is a paid software but fortunately offers a 30 day free trial.



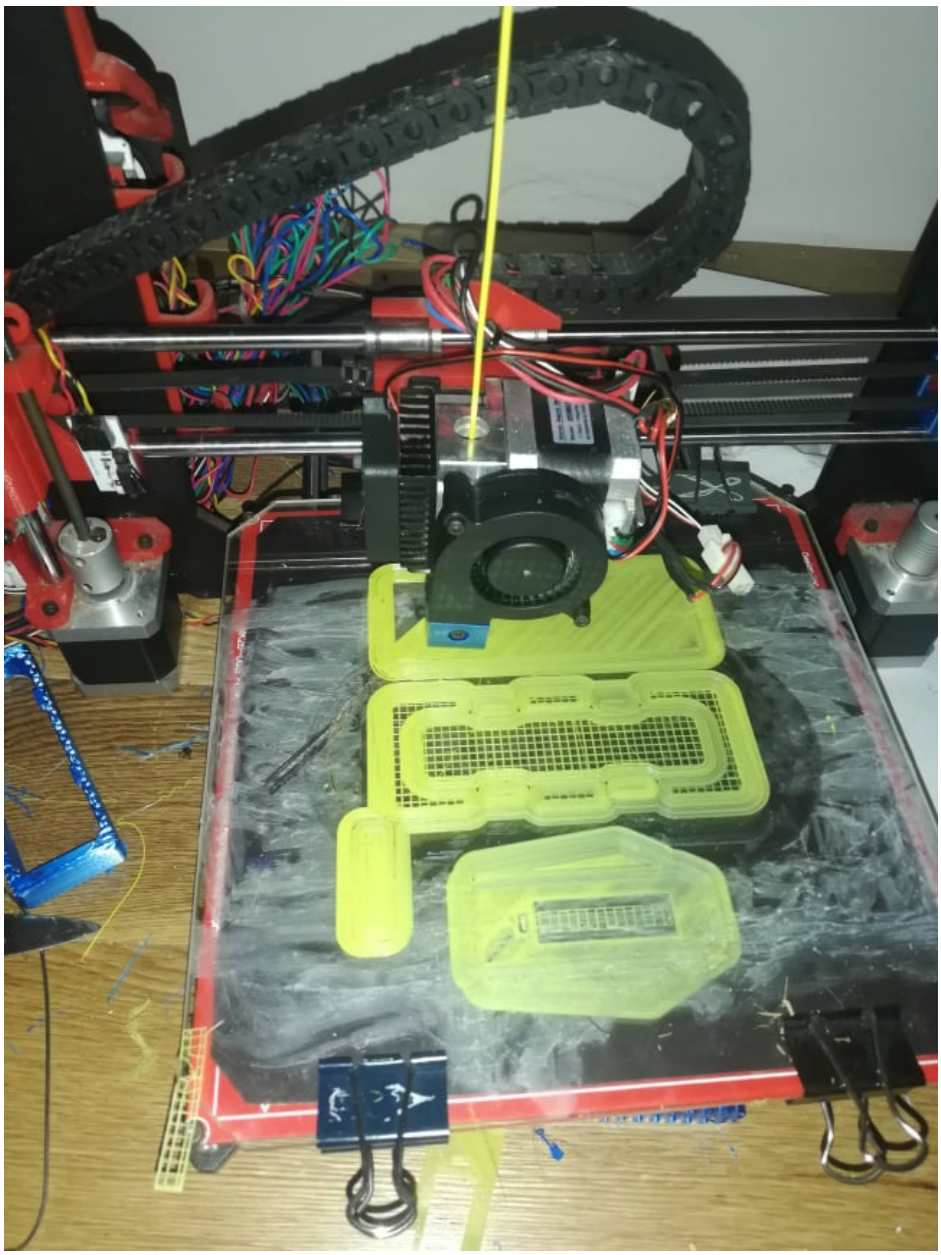
Once the 3D model is created it must be parsed through a slicing software. This is where the density of the print and other variables are inputted. **Ultimaker Cura** was the chosen software but most slicing softwares are offer a similar tool set.

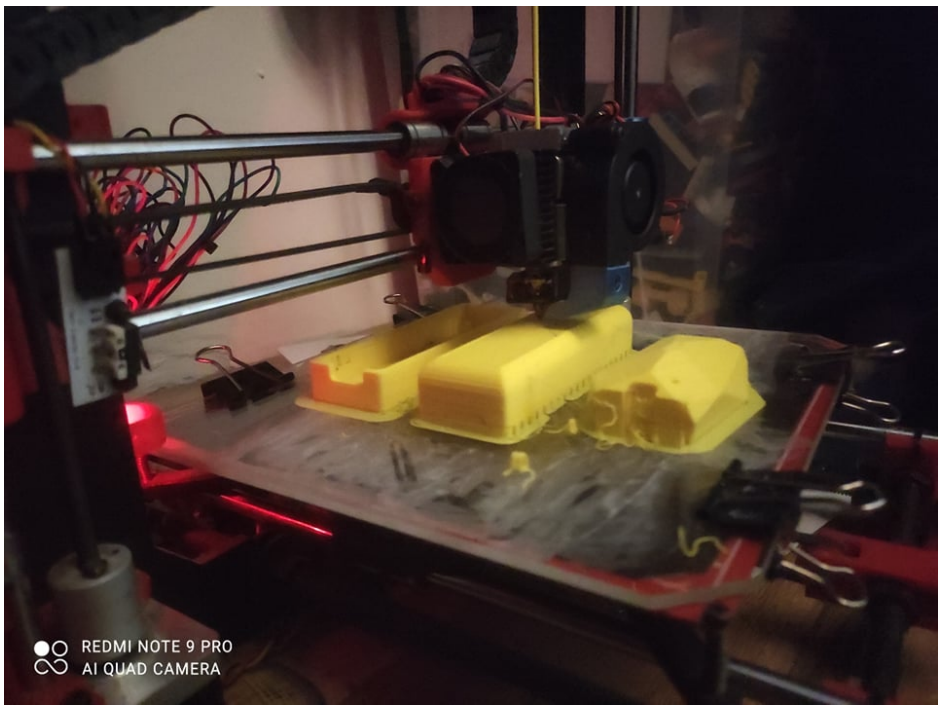




4.6.2 3D Printing

The 3D printer used is a Prusa i3 mk3 Three printing attempts were made for the models. Each of them totaling 9 hours and 44 minutes print time.

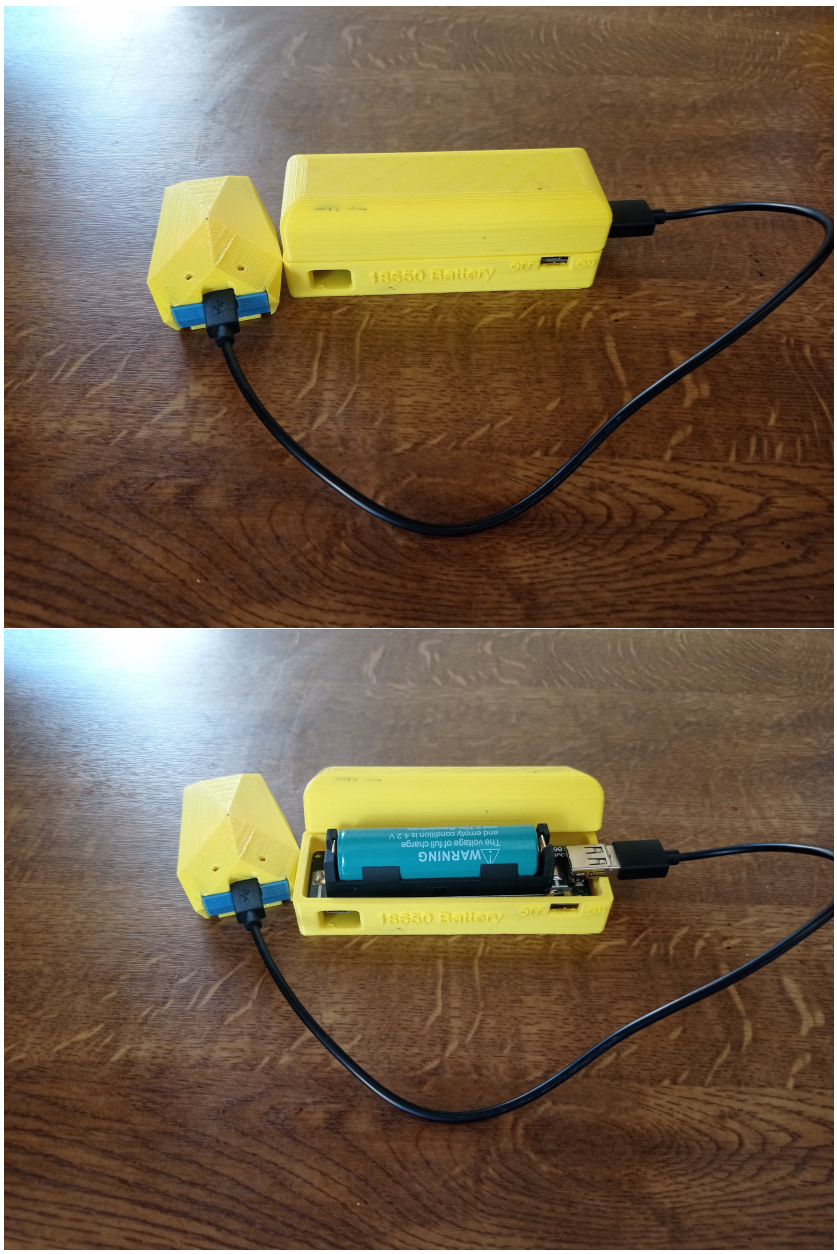



REDMI NOTE 9 PRO
AI QUAD CAMERA

The two failed attempts were due to heat lost during the print from inspecting the progress. This is the longest print undertaken by the printer. Rather than printing everything together, each model should of been it's own print. The second print was 90% done before it failed. If each of the models was printed separately the print time would of been drastically smaller.


REDMI NOTE 9 PRO
AI QUAD CAMERA

4.6.3 Final Prints



4.7 Latex

LaTeX is a document preparation system used for the communication and publication of scientific documents [20]. Although it is not as user-friendly or intuitive as Microsoft Word, anyone with some knowledge of programming would find the interface familiar. While Microsoft Word or Apple Pages follow the "What you see is what you get" method. Latex uses plain text

and tags which will be compiled to the finished page. This may seem like unnecessary extra steps. But since it's release in 1985, Latex's ability to render a finished page identically on all operating systems is something no word processors can do today. The universality of the mathematical expression in Latex creates a worldwide standard for scientists writing academic papers. Although for the case of this project, Latex was used to write the Dissertation on the cloud based editor Overleaf [21].

Chapter 5

System Design

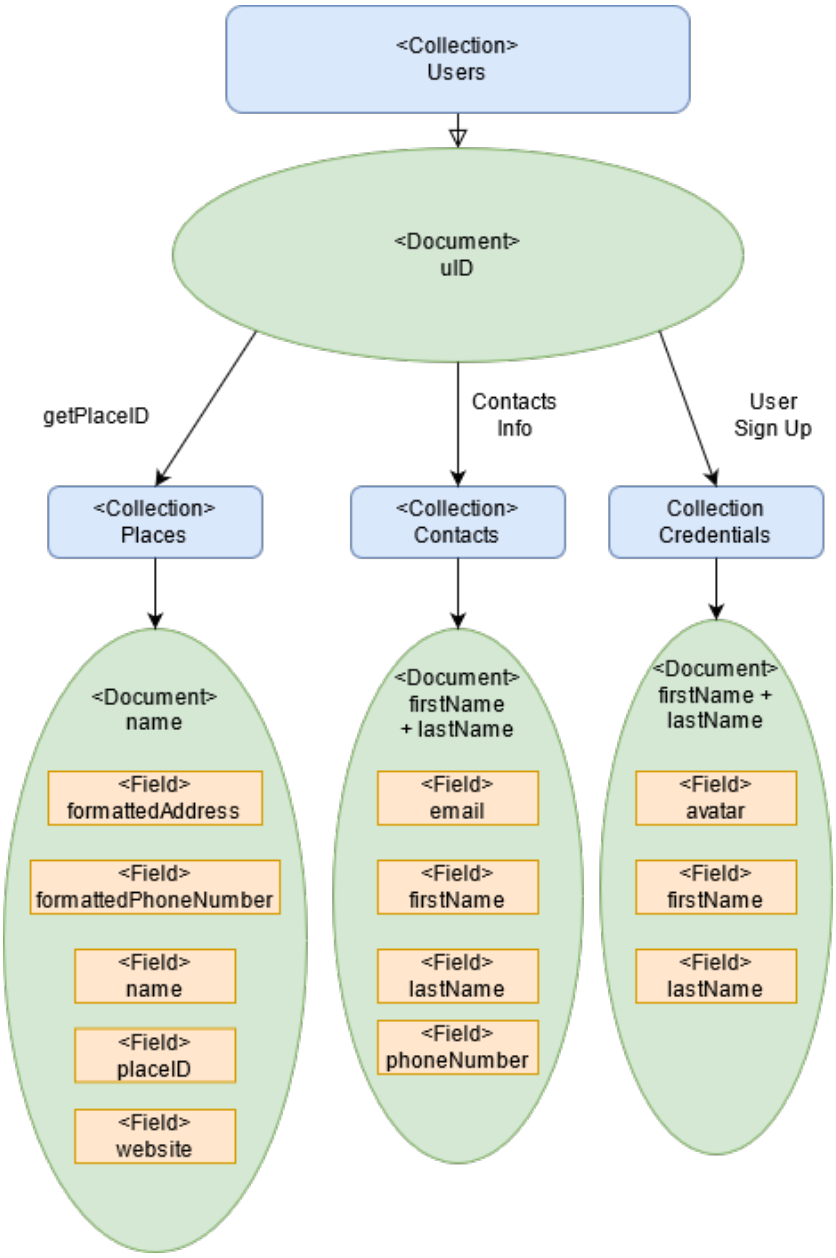
5.1 Chapter Introduction

In this section I will discuss the architecture of the project as a whole and each of the three components (Mobile, NodeMCU, Web) individually in detail.

5.2 Mobile

In the early stages of development I dabbled with Java and Kotlin as my two options for Android development. I was interested in building the database with Firebase. The Firebase homepage mentioned Flutter as an alternative language for multi-platforms. Firebase and Flutter are both developed by Google, explaining the heavy advertising of one another on their respective sites. Nevertheless, this presented an opportunity to develop with Google architecture for front and backend.

5.2.1 Firestore UML



5.2.2 Firebase Setup

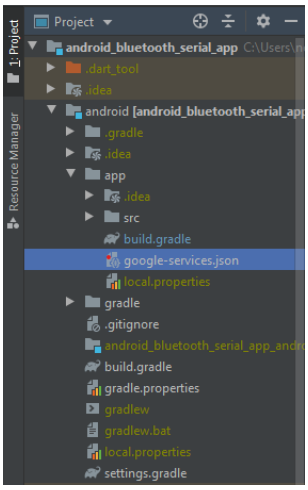
Setting up Firebase is quick and easy with a step by step guideline for each language.

1. When creating a Firebase project there is some configuration required

on the Firebase and Flutter sides. Once a Firebase project is set up, the Android app must be registered to it.

```
39      defaultConfig {
40          // TODO: Specify your own unique Application ID (https://
41          applicationId "com.example.android_bluetooth_serial_app"
42          minSdkVersion 21
43          targetSdkVersion 29
44          versionCode flutterVersionCode.toInteger()
45          versionName flutterVersionName
46      }
```

2. The app id will generate a google-services.json to be added to the Android app module root directory. This configuration file contains the unique details of the Firebase project.



3. Android still needs the Firebase SDK to understand how to use this configuration.

Project-level build.gradle (<project>/build.gradle):

```
buildscript {
  repositories {
    // Check that you have the following line (if not, add it):
    google()  // Google's Maven repository
  }
  dependencies {
    ...
    // Add this line
    classpath 'com.google.gms:google-services:4.3.5'
  }
}

allprojects {
  ...
  repositories {
    // Check that you have the following line (if not, add it):
    google()  // Google's Maven repository
    ...
  }
}
```

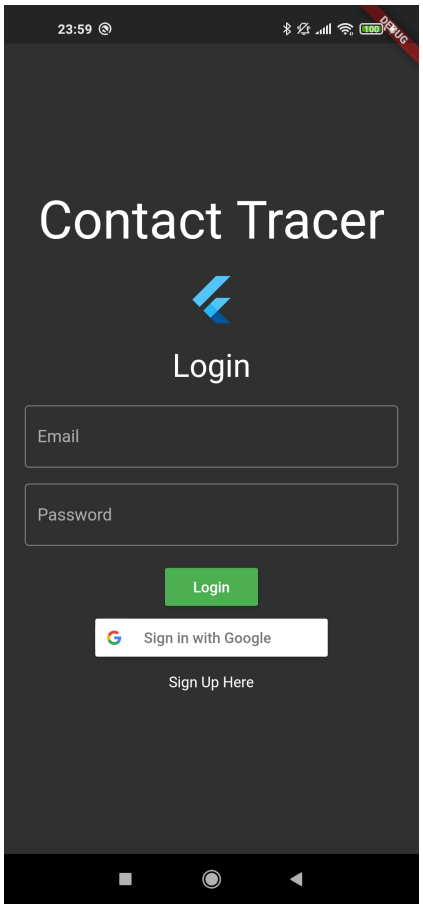
App-level build.gradle (<project>/<app-module>/build.gradle):

```
apply plugin: 'com.android.application'
// Add this line
apply plugin: 'com.google.gms.google-services'

dependencies {
  // Import the Firebase BoM
  implementation platform('com.google.firebase:firebase-bom:26.8.0')

  // Add the dependencies for the desired Firebase products
  // https://firebase.google.com/docs/android/setup#available-libraries
}
```

Once these steps are complete, Gradle will pick up the configuration and sync the project and the Android app is Firebase enabled.

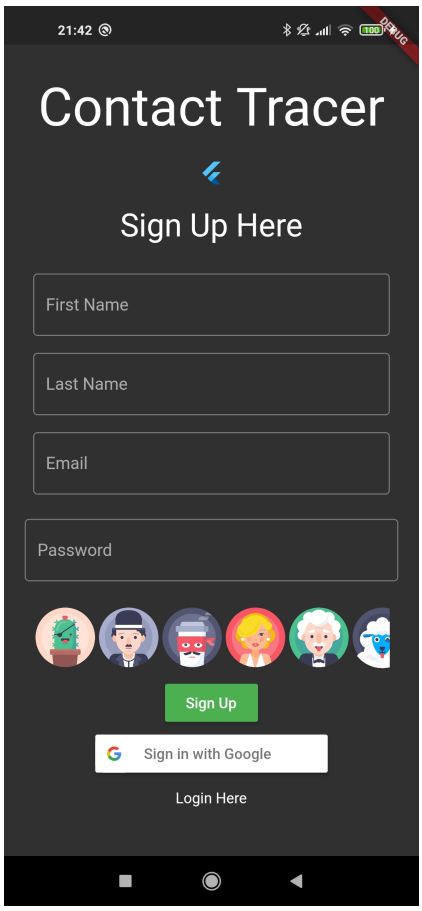
5.2.3 User Registration

Upon opening the app, the user is presented with a login screen. "Sign Up Here" will open the Sign Up Page. Alternatively, the user can sign in with Google.



Sign Up

The Sign Up Page requires basic credentials: first name, last name, email address, password and an icon. Although no email verification is required, a valid email is. Passwords are encrypted by Firebase's authentication service and cannot be viewed but can be reset. Once an account is created, the user will be redirected to the Login Page.

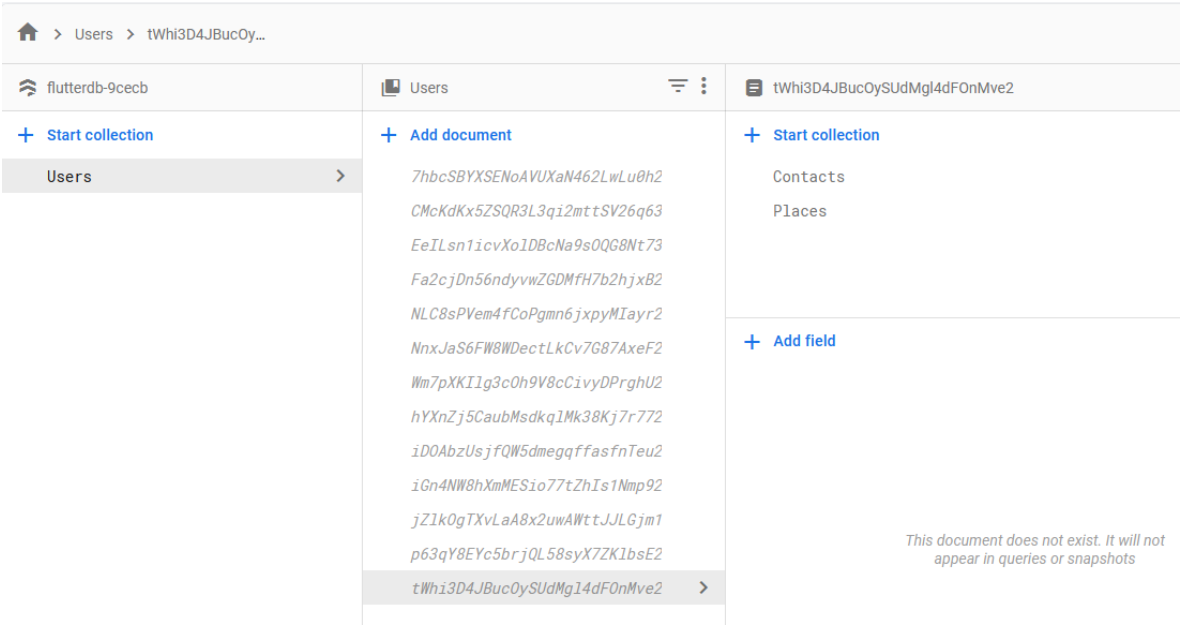


Search by email address, phone number or user UID

Add user

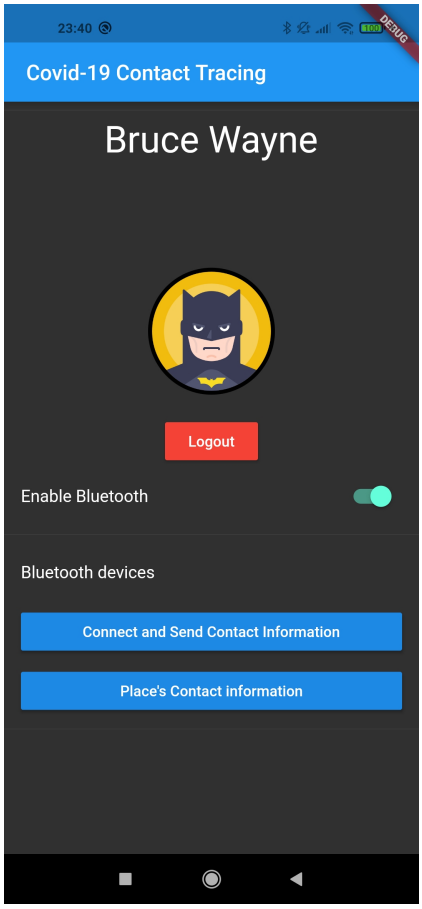
Identifier	Providers	Created	Signed in	User UID
neilbyrne19@gmail.com		16 Apr 2021	29 Apr 2021	tWhi3D4JBucOySUdMgl4dFOnMve2
g00343624@gmit.ie		29 Apr 2021	29 Apr 2021	psAH4OwuA1Z4p0cVtB2gJlwFXOl1
gbleach@glanagua.com		23 Apr 2021	29 Apr 2021	p63qY8EYc5brjQL58syX7ZKlbsE2
neilb@gmail.com		22 Apr 2021	22 Apr 2021	cz7TRDqRqCePpt3sbXCrh8R0e303
batman@gmail.com		26 Apr 2021	29 Apr 2021	Wm7pXKIlg3cOh9V8cCivyDPrghU2
ina@gmail.com		29 Apr 2021	29 Apr 2021	NnxJaS6FW8WDectLkCv7G87Axe...
neilbyrne999@gmail.com		22 Apr 2021	23 Apr 2021	NLC8sPVem4fCoPgmn6jxpyMIayr2
john@gmail.com		28 Apr 2021	28 Apr 2021	EeILsn1icvXolDBcNa9sOQG8Nt73
mayak@gmail.com		27 Apr 2021	27 Apr 2021	CMcKdKx5ZSQR3L3qi2mttSV26q63
thomas@gmail.com		23 Apr 2021	23 Apr 2021	7hbcSBYXSENoAVUXaN462LwLu0...

A UID is generated to uniquely identify the user. The UID will be the name of the document within Firestore to store all of the contacts, places, and credentials for each user. If the user signs in with Google, they will still have a UID but will not have a credentials collection in Firestore. Instead the name and icon will be retrieved from Google.



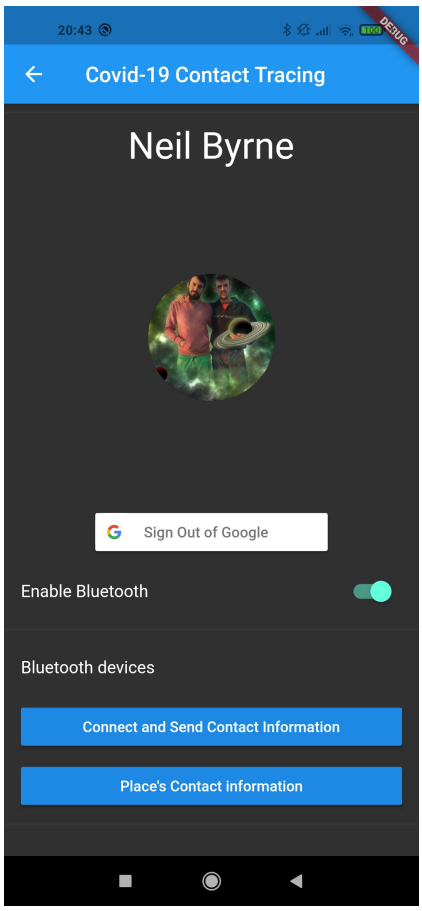
Sign In (Email)

The credentials registered with populate the page for a personal aesthetic.



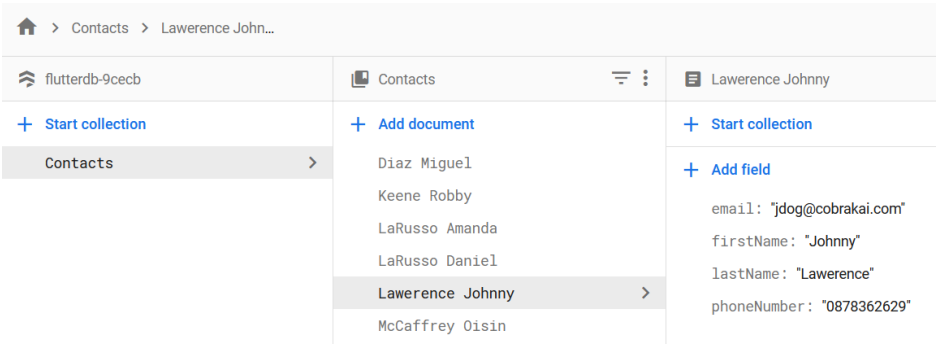
Sign In (Gmail)

Name and profile picture are retrieved from Google for a personal aesthetic.



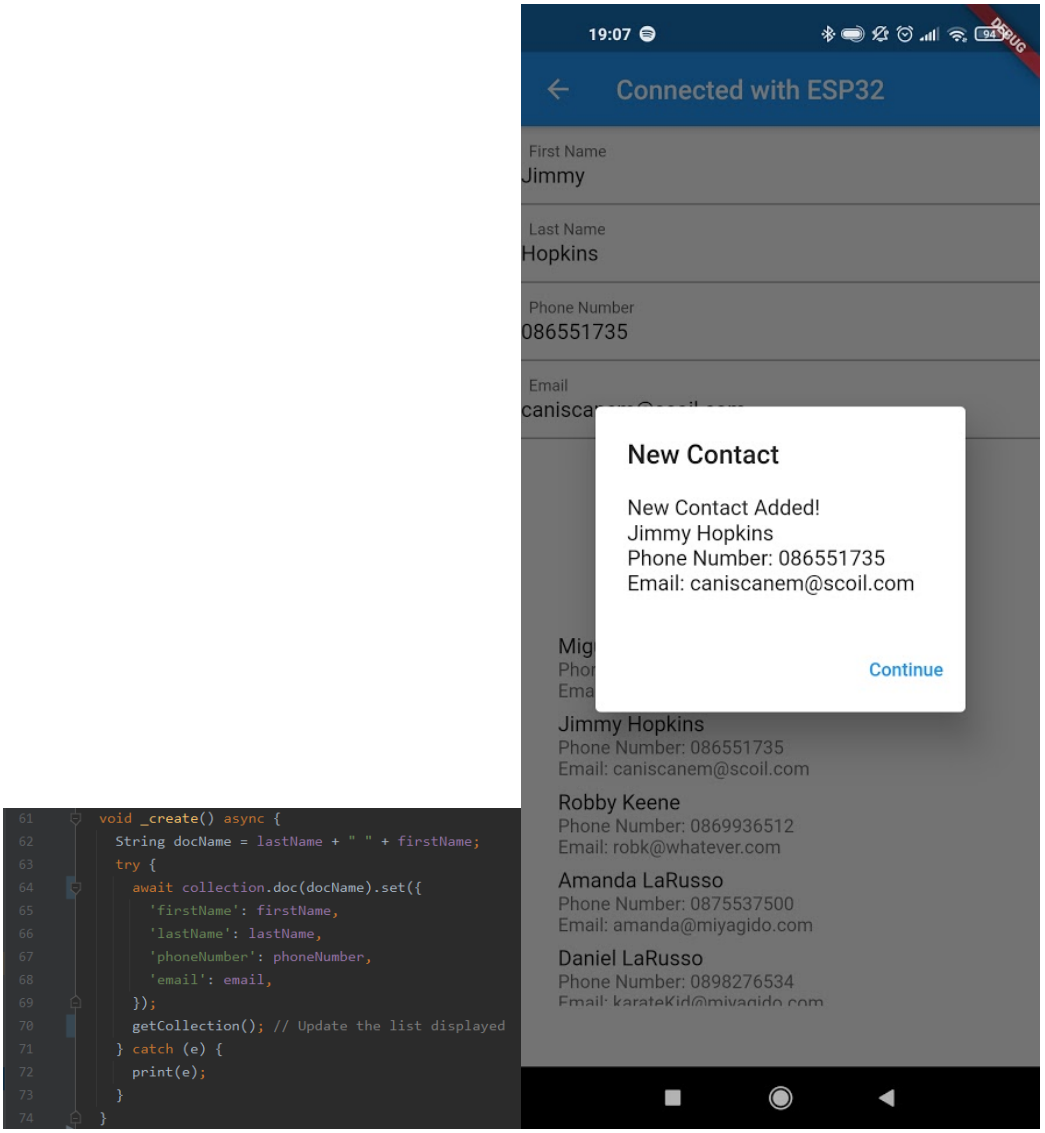
5.2.4 Firestore CRUD operations

With Firebase configured to the Android app, I'm going to discuss the methods on how to perform Create, Read, Update and Delete operations on the Collections saved to Firestore. All Collections and Documents are stored under the Users Collection. Collections within the Users Collection will store the Contacts, Places and Credentials. To add a Document to a Collection a string name for the Document must be decided. While Firebase has capabilities to generate random IDs. For the purpose of creating a contact list, a concatenation of **lastName** and **firstName** is used. A similar concatenation occurs for Places and Credentials Collections for easier querying.



Create

Collection.set will create a Document given a string and set each of the fields to it's counterpart in the text fields. Any new Document will be under the same Collection declared at the beginning.



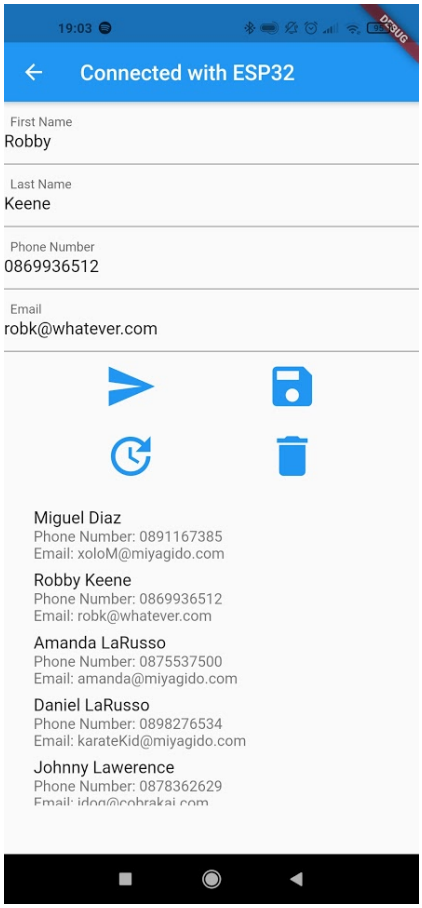

```
void _create() async {
  String docName = lastName + " " + firstName;
  try {
    await collection.doc(docName).set({
      'firstName': firstName,
      'lastName': lastName,
      'phoneNumber': phoneNumber,
      'email': email,
    });
    getCollection(); // Update the list displayed
  } catch (e) {
    print(e);
  }
}
```

Read

Futures are a way in Flutter to asynchronously retrieve data or throw an error. Although the purpose of this Future and **getCollection()** are only for debugging, to ensure that changes have applied to Firestore.

```
final FirebaseFirestore firestore = FirebaseFirestore.instance;
CollectionReference collection = FirebaseFirestore.instance.collection('Contacts');
Future fetchDetails() async {
  collection.get().then((querySnapshot) {
    querySnapshot.docs.forEach((result) {
      print(result.data());
    });
  });
}
```

```
374   void getCollection() {
375     collection.get().then((QuerySnapshot querySnapshot) {
376       querySnapshot.docs.forEach((DocumentSnapshot documentSnapshot) {
377         print(documentSnapshot.data().toString());
378
379       });
380     });
381   }
```



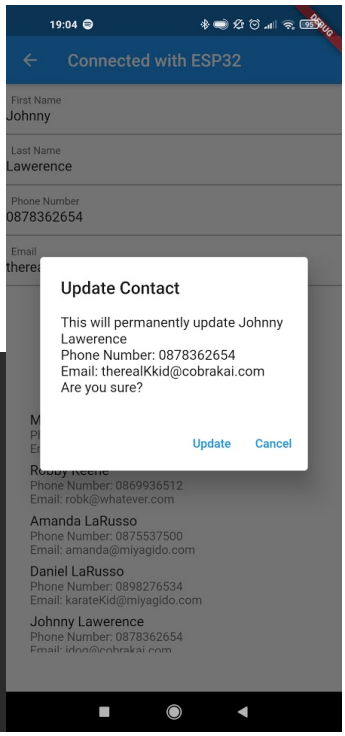
To display each Document individually a StreamBuilder is used. The StreamBuilder builds itself based on the latest snapshots (Document) provided by Collection and returns it in a ListView. Each snapshot is mapped to a DocumentSnapshot that contains the data from the snapshot and therefore can be extracted. All details are displayed for each Document in its own ListTile which can act like a button with the onTap property. When tapped the TextFields at the top are populated and the get methods are updated to avoid any legacy data mismatching what's in the TextField.

```
return new ListView(
  shrinkWrap: true,
  children: snapshot.data.docs
      .map((DocumentSnapshot document) {
    return new ListTile(
      title: new Text(document['firstName'] + " " + document['lastName']),
      subtitle: new Text("Phone Number: " + document['phoneNumber'] + "\nEmail: "  + document['email']),
      onTap: (){
        firstNameField.text = document['firstName'];
        lastNameField.text = document['lastName'];
        phoneNumberField.text = document['phoneNumber'];
        emailField.text = document['email'];
        getFirstName(document['firstName']);
        getLastName(document['lastName']);
        getPhoneNumber(document['phoneNumber']);
        getEmail(document['email']);
        print(document['firstName'] + " " + document['lastName']);
      }
    );  // ListTile
  }).toList(),
);  // ListView
```

Update

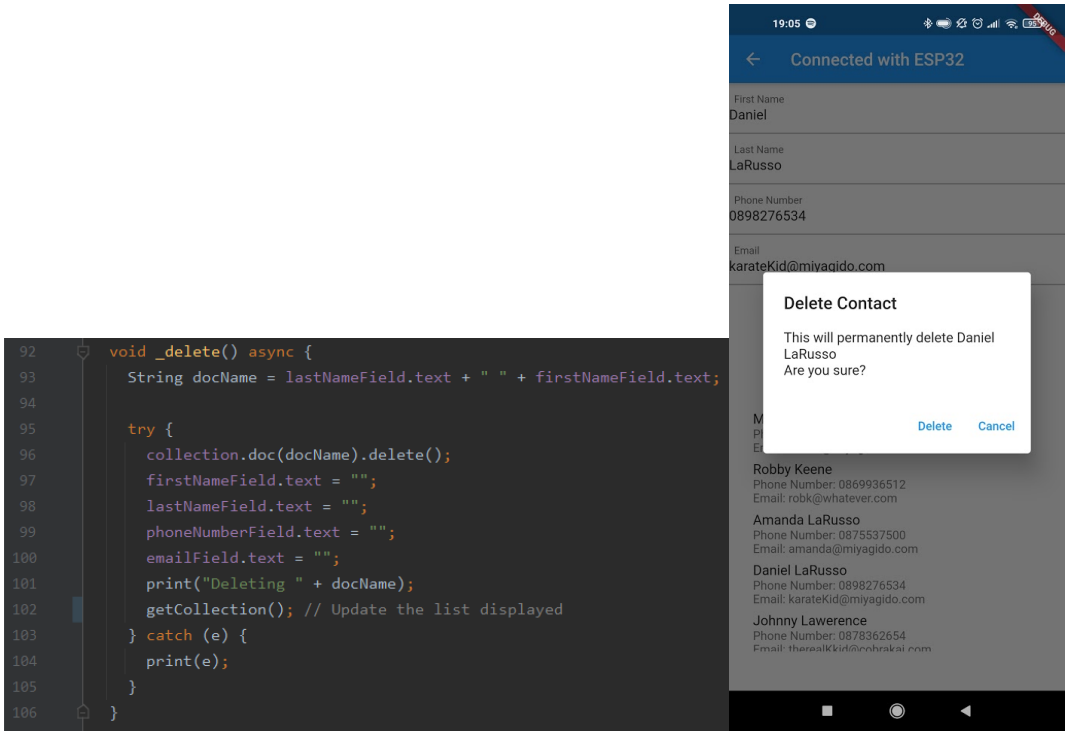
The text fields are used to set field for the given Document. Since the Document is set to be the last and first name concatenated, if either of those are changed a new Document is created. This allows a single person to have several Documents e.g Personal and Professional contacts.

```
76    void _update() async {
77      String docName = lastNameField.text + " " + firstNameField.text;
78      try {
79        collection.doc(docName).update({
80          'firstName': firstNameField.text,
81          'lastName': lastNameField.text,
82          'phoneNumber': phoneNumberField.text,
83          'email': emailField.text,
84        });
85        print("Updating " + docName);
86        getCollection(); // Update the list displayed
87      } catch (e) {
88        print(e);
89      }
90    }
```



Delete

Collection has a delete property to delete any Document given the name. The text fields are then updated to an empty string to display the default labelText which is just the fields name.



Text Fields

When performing any **CRUD operations** it is the data within the text fields that are performed on. Each text field is of class Flexible which controls how the widget flexes. The value of the flex determines the amount of space the widget will occupy as a fraction of the total amount of flex. Each text field is 6 flex (6 flex * 4 text fields = 24 flex) and the StreamBuilder is 24 flex so they are the same size. Splitting the horizontal right in the middle for a cleaner UI.

```
Flexible(
  flex: 6,
  child: TextFormField(
    // FirstName
    controller: firstNameField,
    decoration: InputDecoration(
        labelText: "  First Name",
        fillColor: Colors.white,
        focusedBorder: OutlineInputBorder(
            borderSide:
                BorderSide(color: Colors.blue, width: 2.0))),
    onChanged: (String firstName) {
      getFirstName(firstName);
    },
  ), // TextFormField
), // Flexible
], // <Widget>[]
), // Row
```

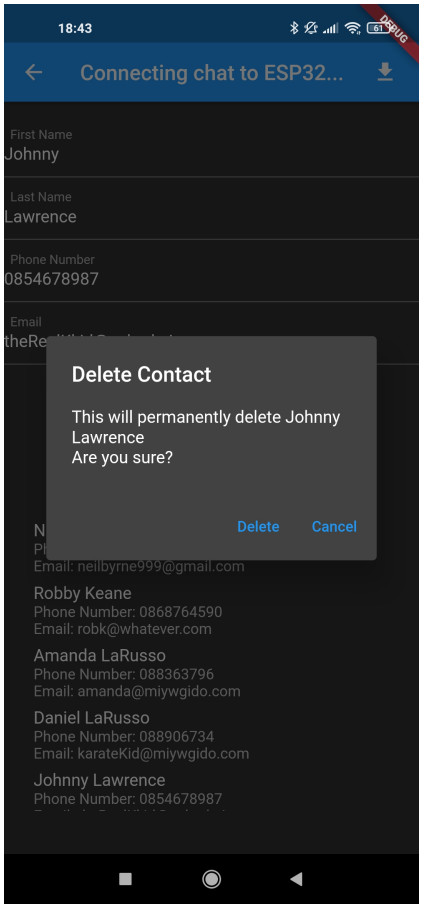
Dialog Alerts

In Flutter everything is considered a Widget and different layouts can be used with each other. The IconSize is the a float value of the logical pixels. When any of the icon buttons are pressed, a showDialog is returned.

```
IconButton(
  color: Colors.blue,
  iconSize: 50,
  icon: const Icon(Icons.delete),
  onPressed: _deleteAlert,
), // IconButton
]), // Row
```

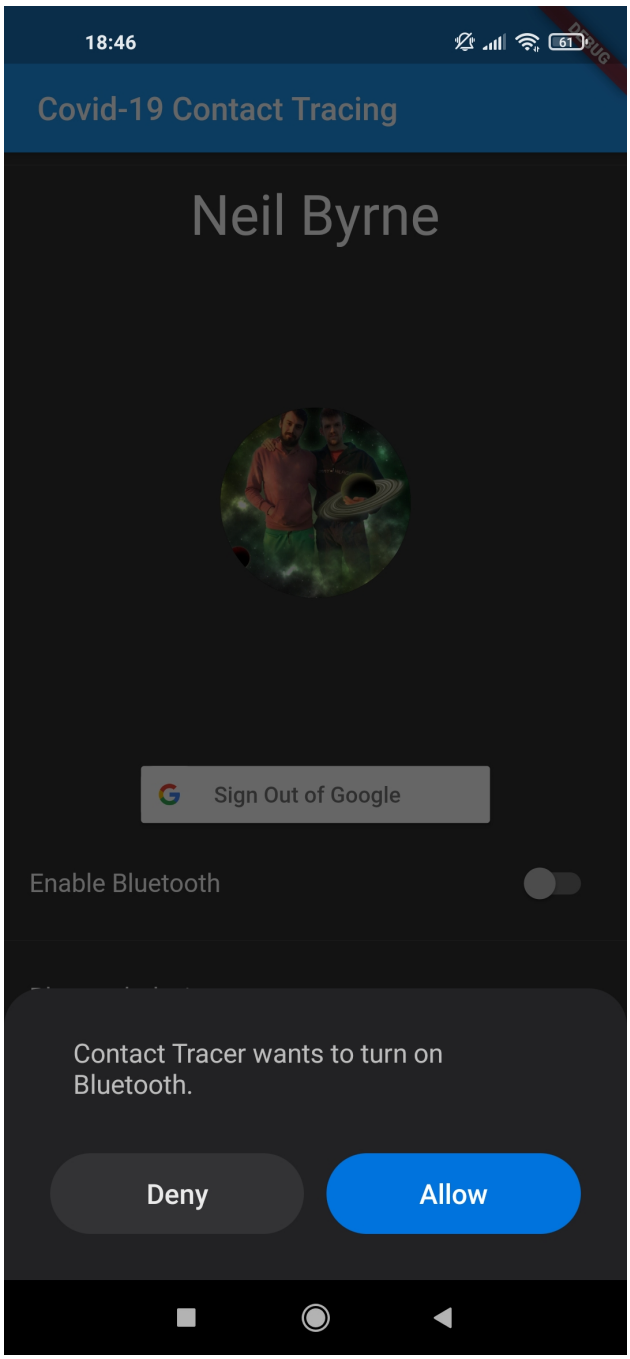
A showDialog displays a Material dialog above the current content on the app. An AlertDialog is returned prompting the user with two TextButtons. Only these two buttons work while the showDialog is active, both close the showDialog when pressed.

```
Future<void> _deleteAlert() async {
  return showDialog<void>(
    context: context,
    barrierDismissible: false, // User must tap button!
    builder: (BuildContext context) {
      return AlertDialog(
        title: Text('Delete Contact'),
        content: SingleChildScrollView(
          child: ListBody(
            children: <Widget>[
              Text('This will permanently delete ' + firstNameField.text + " " + lastNameField.text),
              Text('Are you sure?'),
            ],  // <Widget>[]
          ),  // ListBody
        ),  // SingleChildScrollView
        actions: <Widget>[
          TextButton(
            child: Text('Delete'),
            onPressed: () {
              Navigator.of(context).pop();  // Close dialog box
              _delete();
            },
          ),  // TextButton
          TextButton(
            child: Text('Cancel'),
            onPressed: () {
              Navigator.of(context).pop();  // Close dialog box
            },
          ),  // TextButton
        ],  // <Widget>[]
      );  // AlertDialog
    },
  );
}
```



5.2.5 Bluetooth Serial

The Bluetooth Serial plugin provides all of the Bluetooth functionality. An instance of FlutterBluetoothSerial requests the Bluetooth interface on the phone for permission to enable. This is prompted to the user if Bluetooth is not already on.


18:46
Covid-19 Contact Tracing
Neil Byrne
Sign Out of Google
Enable Bluetooth
Contact Tracer wants to turn on Bluetooth.
Deny
Allow

```
SwitchListTile(
  title: const Text('Enable Bluetooth'),
  value: _bluetoothState.isEnabled,
  onChanged: (bool value) {
    // Do the request and update with the true value then
    future() async {
      // async lambda seems to not working
      if (value)
        await FlutterBluetoothSerial.instance.requestEnable();
      else
        await FlutterBluetoothSerial.instance.requestDisable();
    }
    future().then((_) {
      setState(() {});
    });
  },
), // SwitchListTile
```

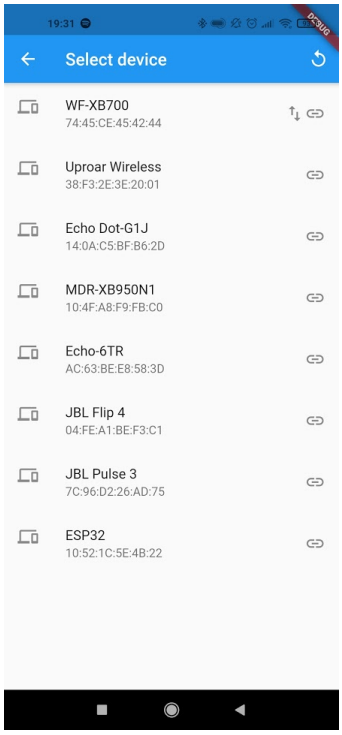
Three things are needed to connect to another Bluetooth device:

1. An instance of **BluetoothDevice** which represents various information about the device. Such as the name of the device, the MAC address and if it has already been paired with before.
2. The **availability** of the device. Is it turned on? Is it within proximity?
3. The **Received Signal Strength Indication (RSSI)** which is a measure of the power present in a received radio signal.

```
24  class _DeviceWithAvailability extends BluetoothDevice {
25    BluetoothDevice device;
26    _DeviceAvailability availability;
27    int rssi; // Received Signal Strength Indication
28
29    _DeviceWithAvailability(this.device, this.availability, [this.rssi]);
30  }
```

If two Bluetooth devices have connected before then they are bonded and the relevant information is usually saved for quicker connecting next time. A list of these devices is saved and can be added to by pressing the refresh button.

```
78  void _startDiscovery() {
79    _discoveryStreamSubscription =
80        FlutterBluetoothSerial.instance.startDiscovery().listen((r) {
81      setState(() {
82        Iterator i = devices.iterator;
83        while (i.moveNext()) {
84          var _device = i.current;
85          if (_device.device == r.device) {
86            _device.availability = _DeviceAvailability.yes;
87            _device.rssi = r.rssi;
88          }
89        }
90      });
91    });
```



Once the device is selected **startChat** will build the page to communicate with the ESP-32 starting a server with it. The passed server provides the MAC address to establish a connection which remains connected until the page is closed. Theoretically seven Android devices can be connected to the ESP-32 simultaneously creating a **Piconet: An ad hoc network linking several Bluetooth devices to one device.** Due to limited hardware I only tested three Android devices simultaneously connected to the ESP-32 and noted no obvious dropout.

```dart
void _startChat(BuildContext context, BluetoothDevice server) {
  Navigator.of(context).push(
    MaterialPageRoute(
      builder: (context) {
        return ChatPage(server: server);
      },
    ), // MaterialPageRoute
  );
}
```

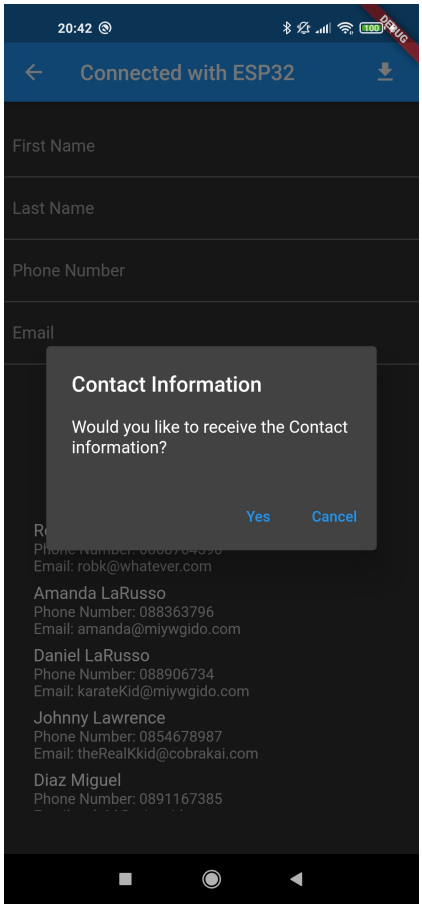
```dart
void initState() {
  super.initState();
  BluetoothConnection.toAddress(widget.server.address).then((_connection) {
    print('Connected to the device');
    connection = _connection;
    setState(() {
      isConnecting = false;
      isDisconnecting = false;
    });
```

Since a Bluetooth connection is serial there is no need for any special wrapper or conversion like with NFC to send messages. The String is UTF-8 encoded and sent to the ESP-32 to be parsed into a HTTP Post request.

```dart
void _sendMessage(String text) async {
  text = text.trim();
  textEditingController.clear();

  if (text.length > 0) {
    try {
      connection.output.add(utf8.encode(text));
      await connection.output.allSent;
```

5.2.6 Place Contact Information

On the top of the chat page, the download button will prompt the user to receive the contact information of the place. This will send a unique string "placeID" to the ESP-32. Once the ESP-32 received this string it will send the hard-coded place ID.

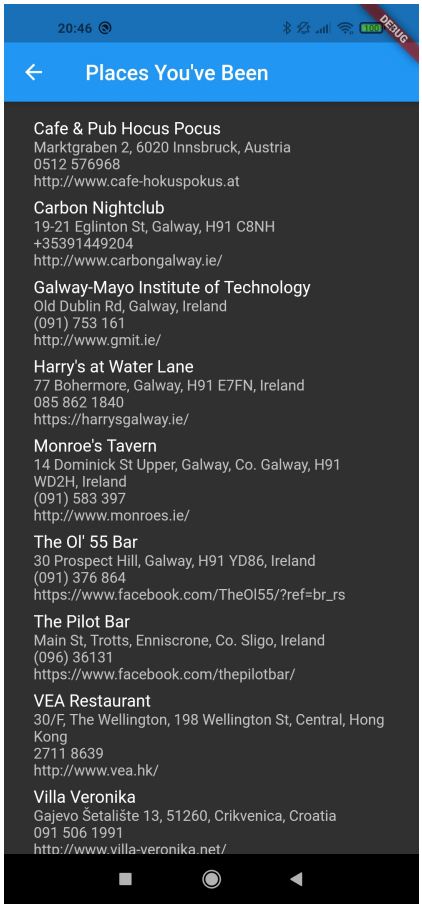


With the place ID, Google maps services can return the location details: name, formatted phone number, formatted address and website. All of this information including the place ID is then added to the Places collection for the user.

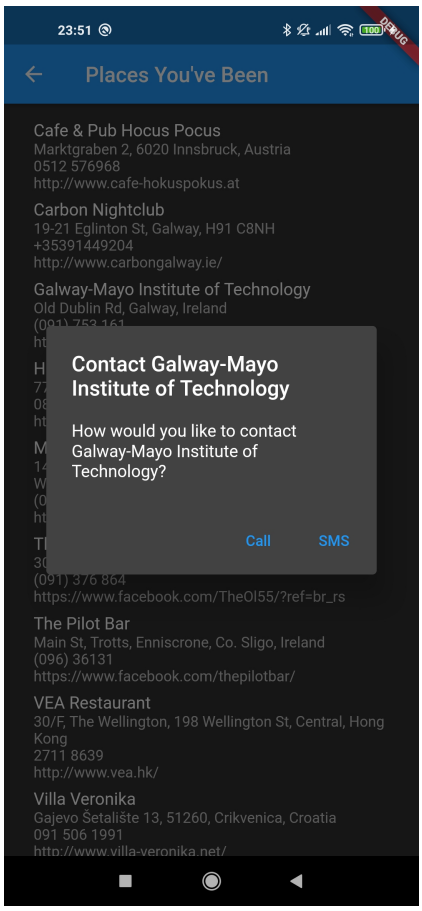
```dart
void _addPlaceID(String placeID) async {
  final places =
  new GoogleMapsPlaces(apiKey: "AIzaSyAz6TJpPOpuhahblOebTaiCmtXHcipwxjc");
  PlacesDetailsResponse response = await places.getDetailsByPlaceId(placeID);

  String name = response.result.name;
  String formattedAddress = response.result.formattedAddress;
  String formattedPhoneNumber = response.result.formattedPhoneNumber;
  String website = response.result.website;

  try {
    await usersCollection.doc(uid).collection('Places').doc(name).set({
      'name': name,
      'formattedAddress': formattedAddress,
      'formattedPhoneNumber': formattedPhoneNumber,
      'website' : website,
      'placeID' : placeID,
    });
    getCollection(); // Update the list displayed
  } catch (e) {
    print(e);
  }
}
```

From the Main Page the user can navigate to the Places Page. Here all of the contact information regarding each location the user has been is displayed. This data is fed through a StreamBuilder from the Firestore. This is the same method the contacts in the Chat Page are displayed.

```
child: ListView.builder(
  controller: listScrollController,
  padding: const EdgeInsets.all(12.0),
  itemBuilder: (context, index) {
    _getUID();
    return StreamBuilder<QuerySnapshot>(
      stream: usersCollection.doc(uid).collection('Places').snapshots(),
      builder: (BuildContext context,
          AsyncSnapshot<QuerySnapshot> snapshot) {
        if (snapshot.hasError)
          return new Text('Error: ${snapshot.error}');
        switch (snapshot.connectionState) {
          case ConnectionState.waiting:
            return new Text('Loading...');
          default:
            return new ListView(
              shrinkWrap: true,
              children: snapshot.data.docs
                  .map((DocumentSnapshot document) {
                return new ListTile(
                  title: new Text(document['name']),
                  subtitle: new Text(
                      document['formattedAddress'] +
                          "\n" +
                          document['formattedPhoneNumber'] +
                          "\n" +
                          document['website']),  // Text
                  onTap: () {
                    name = document['name'];
                    formattedPhoneNumber =
                        document['formattedPhoneNumber'];
                    getName(document['name']);
                    getFormattedPhoneNumber(
                        document['formattedPhoneNumber']);
                    _contactAlert();
                  },
                );  // ListTile
              }).toList(),
            );  // ListView
```



Tapping one of the places will trigger an alert pop up asking the user how they would like to contact the location. The Calls and Message Service will pass the number to the phone call app the user chooses (Whatsapp, Telegram, phone's first party app).




```
192  class CallsAndMessagesService {
193    void call(String number) => launch("tel:$number");
194
195    void sendSms(String number) => launch("sms:$number");
196  }
```

While the Calls and Message Service can also work with emails, storing email addresses is against Google's terms of use with the Place API.

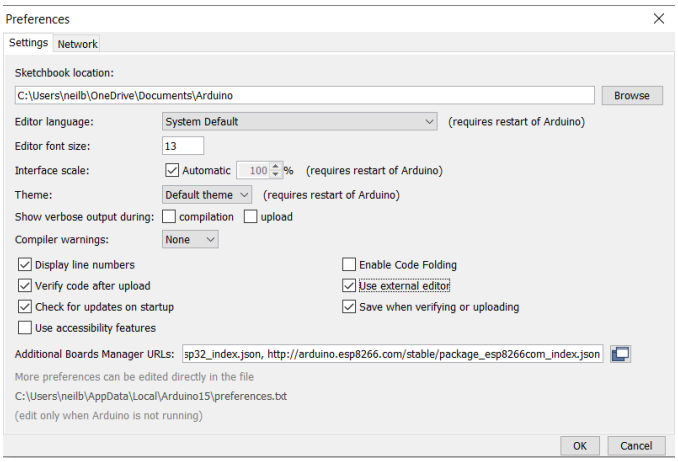
5.3 NodeMCU

5.3.1 Arduino IDE Setup

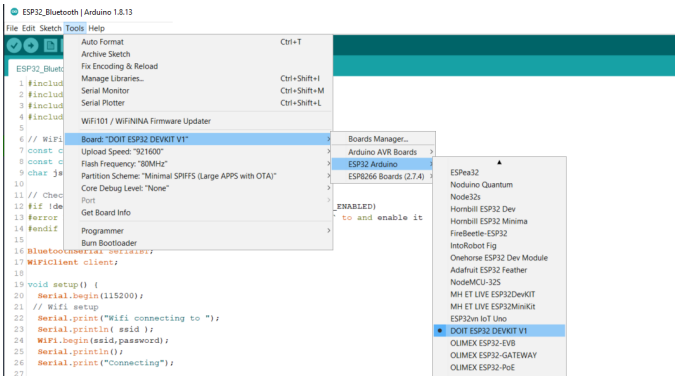
To set up the environment for the Arduino IDE to allow programming for the NodeMCU board there are a few configuration steps that need to be

followed:

1. The Arduino IDE must be informed of the board that is being used. This is done by adding the board to the Board Manager in File -> Preferences. The relevant packages can than be installed for the ESP-32 Board [22] and the ESP-8266 Board [23]

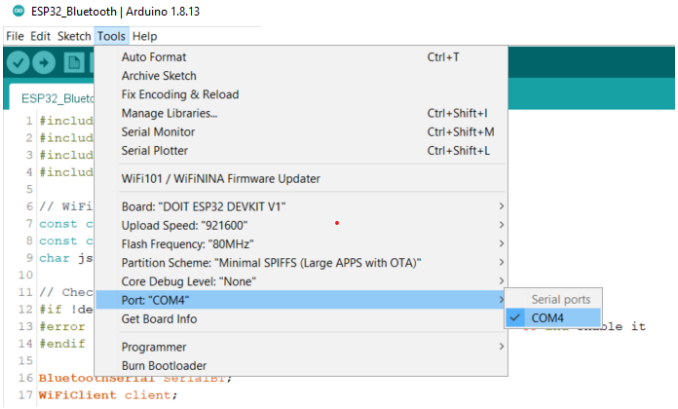


2. Now we can install the board. Go to Tools -> Board -> Board Manager and search for ESP-32. "ESP-32 by Espressif Systems" is the board we are looking for.



3. Once installed we can select the board Tools -> Board

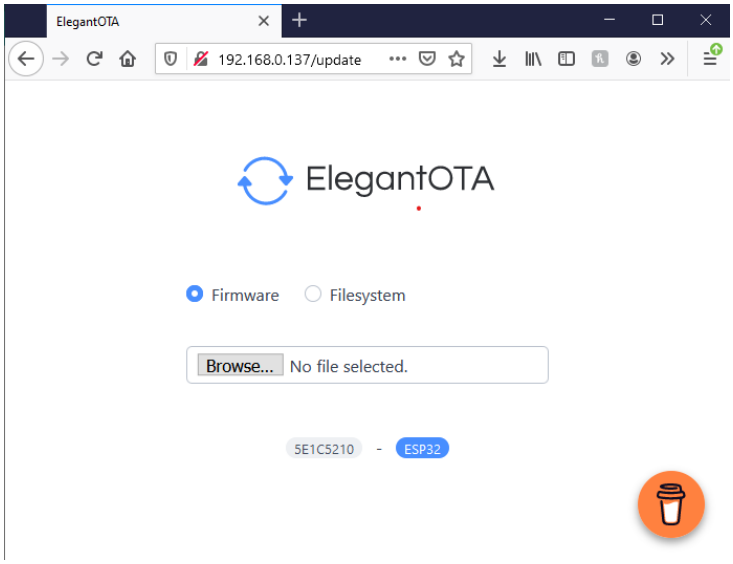
4. Plug the ESP-32 into the computer via USB with a data wire and select the port Tools -> Port -> COM4



On first installation your computer may be missing the required drivers. Silicon Labs provide downloads for any drivers needed [24].

5.3.2 Over The Air (OTA) Uploads

Typically the code is uploaded to the Arduino devices via serial USB cable. However, the NodeMCU has an Over The Air (OTA) feature to allow code upload via Wi-Fi. ElegantOTA provides a simple UI to upload binary exports of sketches so long as the initial setup code has been uploaded. Once OTA is enabled, the NodeMCU becomes a real IoT device not wired to anything but a power source and fully accessible over the network.



5.3.3 WiFi

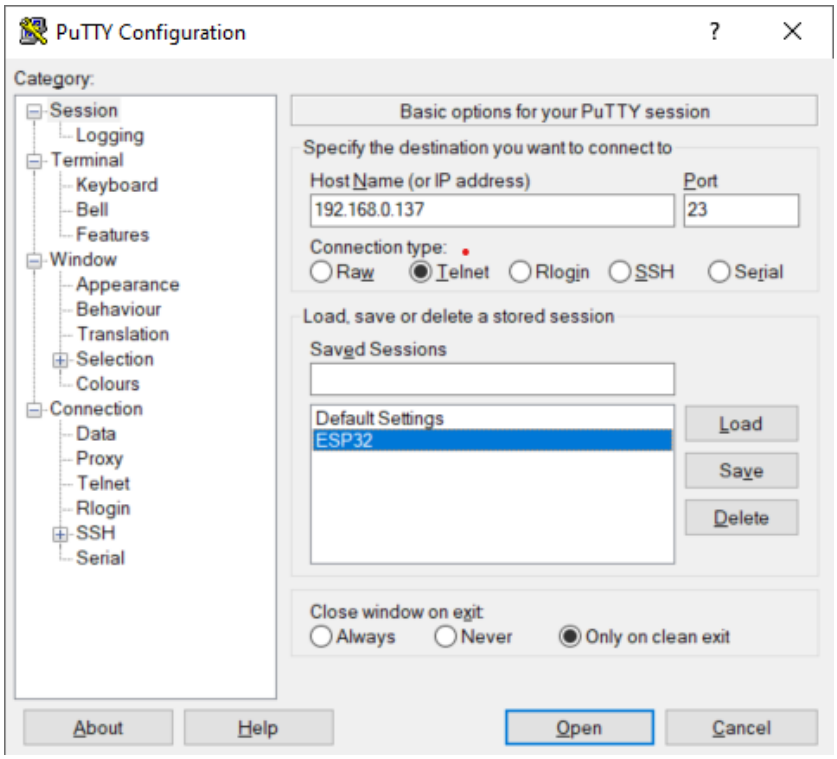
To allow the NodeMCU to connect to Wi-Fi we must explicitly state the network and provide the WEP key.

```
10 // WiFi config
11 const char* ssid="Infinitang IV";
12 const char* password = "PowPowPow1";
```

In the **setup()** function we connect to the internet by calling the WiFi.begin method passing in the credentials. If a connection is established, **WiFi.status()** will check for success!

```
25 void setup() {
26   Serial.begin(115200);
27   WiFi.mode(WIFI_STA);
28   WiFi.begin(ssid, password);
29   Serial.println("");
30
31   // Wait for connection
32   while (WiFi.status() != WL_CONNECTED) {
33     delay(500);
34     Serial.print(".");
35   }
36   Serial.println("");
37   Serial.print("Connected to ");
38   Serial.println(ssid);
39   Serial.print("IP address: ");
40   Serial.println(WiFi.localIP());
41
```

After the Over The Air (OTA) code has been uploaded we no longer need to use the serial input but now lose the serial print log. To aid in debugging and transparency when the app is running, the serial output is also routed to a **Telnet** connection which can be viewed in a **PuTTY** terminal.



5.3.4 Bluetooth

An instance of BluetoothSerial is created and waits until a connection is made. When a connection is made, any data sent to the Serial Monitor is parsed to **convertString(infoString)** where the contact information can be retrieved.

```
// Check if bluetooth is enabled
#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it
#endif

BluetoothSerial SerialBT;
WiFiClient client;
```

```
  if (Serial.available()) {
    SerialBT.write(Serial.read());
  }
  if (SerialBT.available()) {
    String infoString=SerialBT.readString();
    convertString(infoString);
  }
  delay(20);
}
```

5.3.5 RESTful

In **convertString()**, the delimiter which was added in the mobile app is removed and the data is saved to local variables on the NodeMCU.

```
78  char *token = strtok(splitString, "-");
79
80  while(token != NULL){
81    if(firstName == ""){
82      firstName = token;
83    }
84    else if(lastName == ""){
85      lastName = token;
86    }
87    else if(phoneNumber == ""){
88      phoneNumber = token;
89    }
90    else if(email == ""){
91      email = token;
92    }
93    //printf("%s\n", token);
94    token = strtok(NULL, "-");
95  }
```

Finally the information must be put into a JSON format to be accepted by the API. The IP address of the computer is the first part of the address because the MEAN project is running on the computer not the NodeMCU. Followed by the server NodeJS is running on and then the route to the API. Since we have access to the internet on the network, this could be altered to any API that is served online.

```
116   client.begin("http://192.168.0.80:3000/api/contact/");
117   client.addHeader("Content-Type", "application/json");
```

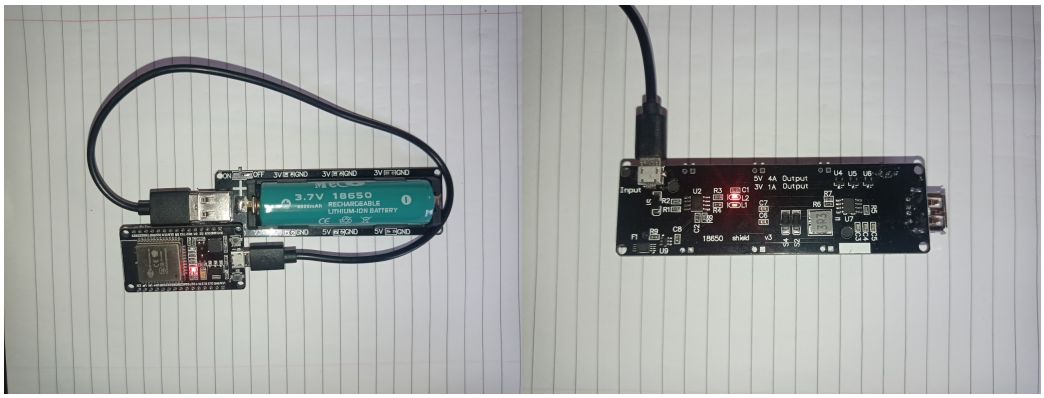
```
Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . : home
   IPv4 Address. . . . . . . . . . . : 192.168.0.80
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.0.1
```

Each field has an associated object which is predetermined in the API. Once we serialise the JSON object document, the HTTP POST is complete.

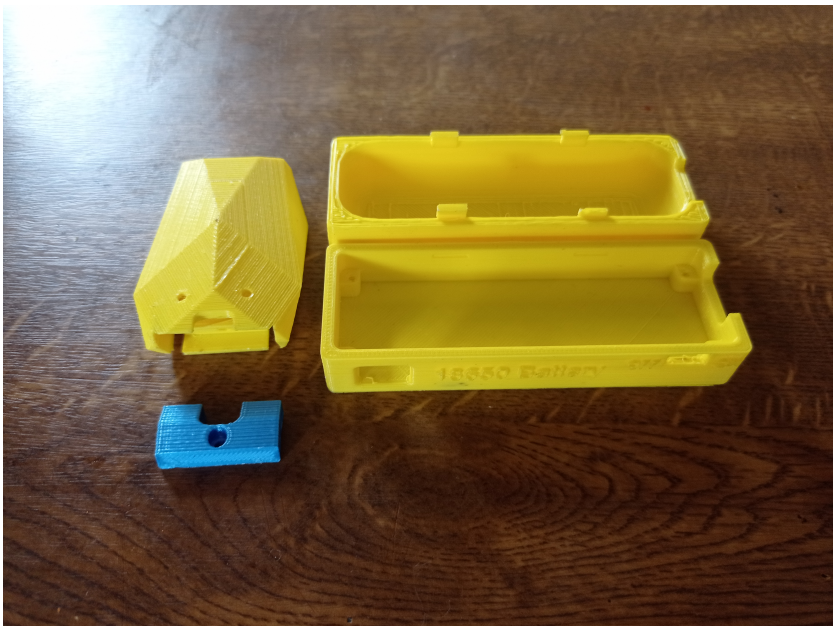
```
123     JsonObject object = doc.to<JsonObject>();
124     object["first_name"] = firstName;
125     object["last_name"] = lastName;
126     object["phone"] = phoneNumber;
127     object["email"] = email;
128
129     serializeJson(doc, jsonOutput);
130     Serial.println(String(jsonOutput));
131     TelnetStream.println(String(jsonOutput));
132
133     int httpCode = client.POST(String(jsonOutput));
```

5.3.6 Battery Pack

In the interest in building a portable system, some sort of a battery pack was added to the specification. Anything that produces 3V of power would work with the NodeMCU. Portable power banks are an overkill because of their weight and large capacity. Instead I opted with a Battery Charging Shield [25]. This module is fitted with a 4000mAh rechargeable lithium-ion battery which can be recharged from the port on the back.



5.3.7 3D Printed Cases



5.4 MEAN Stack

The web app is modelled after the MEAN stack (MongoDB, ExpressJS, Angular, NodeJS). This model can be broken down into the three tier architecture. Data Tier, Logic Tier and Presentation Tier. In development these different tiers are described in the Model View Controller (MVC). This is a common architecture for building dynamic web pages. The first tier a user will interact with is the the Presentation tier. Angular will process the request made on the client side and pass it to the Logic Tier which is a NodeJS server. The server then uses ExpressJS to make the request to the Data Tier. ExpressJS retrieves the data from MongoDB. The data makes its way back through the NodeJS server and onto the Presentation Tier where the result will be displayed.

5.5 Data Tier

5.5.1 Mongoose

The Mongoose package enables CRUD (Create, Read, Update, Delete) operations to be performed on MongoDB. To do this Mongoose (Data Tier) must connect to NodeJS (Logic Tier). MongoDB is set up on port 3000.

```
12  // Connect to mongoDB
13  mongoose.connect('mongodb://localhost:27017/contactlist');
14  // On Connection
15  mongoose.connection.on('connected', ()=>{
16      console.log('Connected to database mongodb @ 27017');
17  });
18
19  mongoose.connection.on('error', (err)=>{
20      if(err){
21          console.log('Error in database connection: ' + err);
22      }
23  });
24
25  // Port number
26  const port = 3000;
```

5.6 Logic Tier

The Logic Tier refers to the business logic of the app. This is where the data is manipulated through CRUD operations. NodeJS allows a unified language to be used between the three tiers. Due to NodeJS's non-blocking IO calls and light-weight implementation, it enables the application to scale handling thousands of concurrent connections.

5.6.1 Connecting to Data Tier

As mentioned, Mongoose is how the Data and Logic Tier are connected. Mongoose can define a schema with strongly typed data in JavaScript. The data can then be manipulated through CRUD operations. After connecting to MongoDB, the data model must be declared for each type. The universality of JavaScript makes these interactions easy. first_name, last_name, phone and email are all String types and the timestamp is type Date. When a contact is created, the timestamp will save the currant date. This date will be used as a filter performed in the Presentation Tier.

```
var express = require('express');
var mongoose = require('mongoose');
var bodyparser = require('body-parser');
var cors = require('cors');
var path = require('path');

var app = express();

const route = require('./routes/route');
var emailRouter = require('./routes/emailRouter');

// Connect to mongoDB
mongoose.connect('mongodb://localhost:27017/contactlist');
```

```
const ContactSchema = mongoose.Schema({
    first_name:{
        type: String,
        required: true
    },
    last_name:{
        type: String,
        required: true
    },
    phone:{
        type: String,
        required: true
    },
    email:{
        type: String,
        required: true
    },
    timestamp : {
        type: Date, default: Date.now
    }
});
```

5.6.2 Connecting to Presentation Tier

Port 3000 which was set up connected with Mongoose is where the server will send and retrieve data to and from the Presentation Tier. Using the IP address, port number and routing address, the presentation tier will send HTTP requests back to the server. The routing address will determine what action is to be performed on the data.

5.6.3 CRUD Operations

Once the Data Tier is connected to the other tiers, the routes can be used to perform CRUD operations. Contacts can be added via HTTP POST requests from the ESP-32 but they can also be added on the web app.

Create

To save a new contact addContact() will read the text fields and parse their data. The time stamp will save the current DateTime. A pipe will be applied to this later to print the date in the correct format. The route deals with the kind of HTTP request being dealt with. Then the API will save and return the update database.

```
addContact(){
  const newContact ={
    first_name: this.first_name,
    last_name: this.last_name,
    phone: this.phone,
    email: this.email,
    timestamp: this.now.toDateString()
  }
  this.contactService.addContact(newContact)
  .subscribe(contact =>{
    this.contacts.push(contact);

    this.contactService.getContacts() // Refresh when new contact added
    .subscribe( contacts =>
      this.contacts = contacts);
  });
}
```

```
// Add contact
router.post('/contact', (req, res, next)=>{
    console.log(req.body)

    let newContact = new Contact({
        first_name: req.body.first_name,
        last_name: req.body.last_name,
        phone: req.body.phone,
        email: req.body.email,
    });
    newContact.save((err, contact)=>{
        if(err){
            res.json({msg: 'Failed to add contact'});
        }
        else{
            res.json({msg: 'Contact added successfully'});
        }
    });
});
```

```
// Add contact
addContact(newContact: any){
  var headers = new Headers()
  headers.append('Content-Type', 'application/json')
  return this.http.post('http://localhost:3000/api/contact/', newContact, {headers:headers})
  .pipe(map((res: any) => res.json()));
}
```

Read

On initialise of the project the contact service performs a GET request on all of the contacts saved to the API. There is no associated method for the READ because no data is being manipulated.

```
ngOnInit() {
  this.contactService.getContacts().subscribe( contacts =>this.contacts = contacts);
  this.nodeMailerForm = this.formBuilder.group({emailr:[null,[Validators.required]]})
}
```

```
13   getContacts(){
14     return this.http.get('http://localhost:3000/api/contacts/').pipe(map((res: any) => res.json()));
15   }
16
```

Update

While there is no reason for a contact to be edited, the functionality is available and can be easy implemented by added the action on the Presentation Tier.

```
94   updateContact(id:any){
95     var contacts = this.contacts;
96     this.contactService.deleteContact(id)
97       .subscribe(data =>{
98           if(data.n==1){
99             for(var i = 0;i< contacts.length;i++){
100              if(contacts[i]._id == id){
101                contacts.splice(i, 1);
102              }
103            }
104          }
105    });
106  }
```

Delete

Once a contact is created it is assigned an ID. This ensures no contact is accidentally overwritten and enables a simple method to delete the contact. The ID will be passed through the method were the route will handle the HTTP request. The API will delete the record and return the updated database.

```
80    deleteContact(id:any){
81      var contacts = this.contacts;
82      this.contactService.deleteContact(id)
83        .subscribe(data =>{
84            if(data.n==1){
85              for(var i = 0;i< contacts.length;i++){
86                if(contacts[i]._id == id){
87                  contacts.splice(i, 1);
88                }
89              }
90            }
91        });
92    }
```

```
33  router.delete('/contact/:id',(req, res, next)=>{
34      Contact.remove({_id: req.params.id}, function(err, result){
35          if(err){
36              res.json(err);
37          }
38          else{
39              res.json(result);
40          }
41      });
42  });
```

```
25    // Delete contact
26    deleteContact(id: any){
27      return this.http.delete('http://localhost:3000/api/contact/' + id)
28      .pipe(map((res: any) => res.json()));
29    }
```

5.6.4 Date Filter

The datepicker allows the user to enter a date either through text input, or by choosing a date from the calendar. When a date is picked, an event is created with that date. This is used in the Presentation layer as a filter. So only the contacts with the given date will be presented.

```
29    addEvent(event: MatDatepickerInputEvent<Date>) {
30      var test1 = `${event.value}`;
31      var test  = new Date (test1.toString()).toDateString();
32
33      this.events = test
34      console.log(test)
35      console.log(String(test))
36    }
```

```
export class FilterPipe implements PipeTransform {
  transform(items: any[], test: string): any[] {

    if (!items) {
      return [];
    }
    if (!test) {
      return items;
    }
   test = test;
    console.log('SEARCH TEXT: ', test)
    console.log('SEARCH ITEMS: ',items)

    return  items.filter(o => {
      console.log('SEARCH timestamps: ',o.timestamp)
     // return o.timestamp.toDateString().includes(test)
      return new Date(o.timestamp).toDateString().includes(test)
    });
```

5.6.5 Email Client

NodeMailer serves as the email client to send a pre-written to. The method passes an email string onto the service. In order to allow emails to be sent from a given email, the setting must be enabled in the the Gmail account.

```
  sendMail(email:any){
    //debugger
    alert("Email Sent!");
    let emailr  = this.nodeMailerForm.value.emailr;
    console.log(email)
    let reqObj = {
      emailr:emailr,
      email:email
    }
    this.emailService.sendMessage(reqObj).subscribe(data=>{
      console.log(data);

    })
  }
```

```
transporter.sendMail(mailOptions, function(error, info){
  if (error) {
    console.log(error);
    res.send('error') // if error occurs send error as response to client
  } else {
    console.log('Email sent: ' + info.response);
    res.send('Sent Successfully')//if mail is sent successfully
  }                            //send Sent successfully as response
});
})
```

An email for this purpose was created: **neilbyrne999@gmail.com**. MailOp-

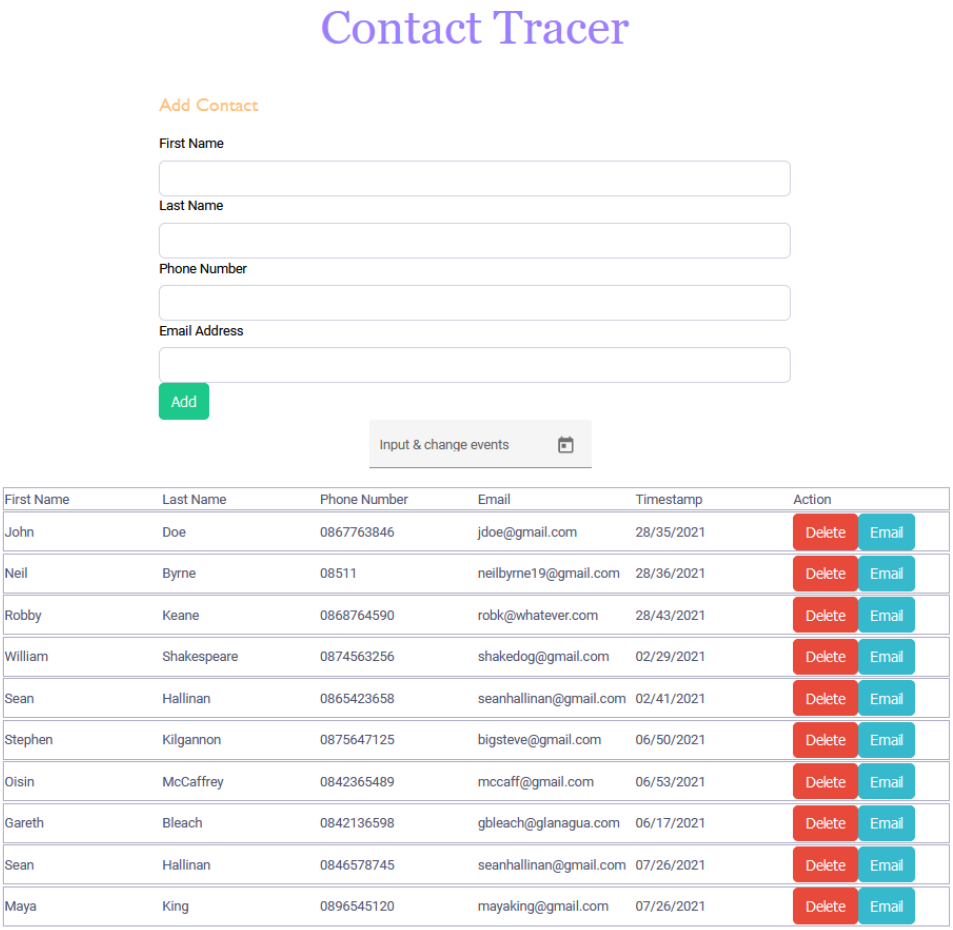
tions details with the entire structure of the email.

```
21  var transporter = nodemailer.createTransport({
22    service: 'gmail',
23    auth: {
24      user: 'neilbyrne999@gmail.com',//replace with your email
25      pass: '#########'//replace with your password
26    }
27  });
```

```
35  var mailOptions = {
36    from: 'neilbyrne999@gmail.com',//replace with your email
37    to: req.body.email,//replace with your email
38    subject: `Covid-19 Outbreak`,
39    text:`There has been as Covid-19 outbreak at _______`
40  };
```

5.7 Presentation Tier

The Presentation Tier is at the topmost level of the three-tier architecture. It supplies a graphical interface for the user to interact with. Angular creates Single Page Apps (SPA) so any data changed on the app will only refresh the content not the page.



The add contact method will read the text fields and pass it onto the contact service.

```
<div style="padding-left: 20%; padding-right: 20%;">
  <h2 class = "page-header"> Add Contact</h2>
  <form (submit) = "addContact()">
      <div class ="form-group">
          <label>First Name</label>
          <input type="text" [(ngModel)] = "first_name" name = "first_name" class = "form-control">
      </div>
      <div class ="form-group">
          <label>Last Name</label>
          <input type="text" [(ngModel)] = "last_name" name = "last_name" class = "form-control">
      </div>
      <div class ="form-group">
          <label>Phone Number</label>
          <input type="text" [(ngModel)] = "phone" name = "phone" class = "form-control" required>
      </div>
      <div class ="form-group">
          <label>Email Address</label>
          <input type="text" [(ngModel)] = "email" name = "email" class = "form-control" required>
      </div>
      <input type = "submit" class = "btn btn btn-success" value = "Add">
  </form>
</div>
```

Each contact is iterated with their details. When the user selects a date from the date picker, the filter will ensure only contacts with the specified date is displayed. A pipe on the date formats it to DD/MM/YYYY. The delete button retrieves the ID of the contact to be deleted. While the email button retrieves the email and passes it to the method and then onto the service.

```
<div *ngFor = "let contact of contacts  | filter: events" style="padding-left: 5%;padding-right: 5%" >

  <table >
    <tr >
      <td>{{contact.first_name}}</td>
      <td>{{contact.last_name}}</td>
      <td>{{contact.phone}}</td>
      <td>{{contact.email}}</td>
      <td>{{contact.timestamp | date:'dd/mm/yyyy' }}</td>
      <td>
        <input type= "button" (click) = "deleteContact(contact._id)" value = "Delete" class = "btn btn-danger">
        <input type= "button" (click) = "sendMail(contact.email)" value = "Email" class = "btn btn-info">

      </td>
    </tr>
  </table>
```

The Date picker will pass the changed date as an event to be handled in the Logic Tier.

```
30          <div style="padding-left: 40%;">
31            <mat-form-field appearance="fill">
32              <mat-label>Input & change events</mat-label>
33              <input matInput [matDatepicker]="picker"
34                     (dateChange)="addEvent($event)">
35              <mat-datepicker-toggle matSuffix [for]="picker"></mat-datepicker-toggle>
36              <mat-datepicker #picker></mat-datepicker>
37            </mat-form-field>
38          </div>
```

Chapter 6

System Evaluation

6.1 Robustness

Overall each aspect of the project works well individually and coupled together. There are no known bugs or glitches to cause any errors or crashes in the projects. However, there are some minor issues and other features that were cut because of time constraints fixing their errors.

- When signing out of the Flutter app a null error may appear for brief moment and then the app resumes. This is caused because the login is null when signed out and the error is thrown before the login page loads. While this error causes no actual problem it is immersion breaking and may lead the user to believe there is a more underlying issue.

- A feature where the user could receive a URL or PDF from an establishment that displayed a menu or information the business wanted to pass on was to be added. The flutter_web_plugin was to handle the URLs and open a browser in app. After building the project with this new package a list of errors occurred. After searching for a solution to this problem I discovered it to be a bug within Android Studio [26]. There seems to be no one fix for the problem. After trying several with no result. I decided enough time was lost and I cut the feature entirely to concentrate on polishing what was done. Even after removing the package, I had to roll back the project to a previous version for a clean build.

6.2 Testing

Testing was done continuously through the project. Careful consideration was taken for the features that controlled the communication between the different devices. Measures to ensure that edge cases and unexpected data did not cause any unwanted behaviour was tested heavily. For the most part, in college we design our apps to work a certain way and avoid what might break them. Going into industry this is no longer true. We must try to break our apps and create scenarios that test the limitations of them.

- When requesting for a place's contact information, the Flutter app simply sends "PlaceID" to the ESP-32. Checks on Flutter and the ESP-32 are in place to prevent the user from triggering this from other means.
- The idea of the project is to send relevant contact information. While there is no way to ensure that the information being sent is accurate, checking if it is valid is possible. Checks on the Flutter, ESP-32 and Web apps have some sort of validation for the email and phone number.

6.3 Scalability

Each project of the system was built in mind of each other. But each project can stand on its own. The Firestore free tier has a quota more than big enough for the app. If the app were to be published to the app store, this

can easily be upgraded to a paid tier. As discussed in REFERENCE, I faced some issues when adding more feature to the ESP-32. The script in its current form has the 4Mb of memory on the ESP-32 almost maxed out. While the module is great, it has been 5 years since the last release. In order to scale the app with more functionality such as more HTTP requests or added peripherals an upgrade in hardware is required. Thanks to the thousands of simultaneous connections NodeJS can handle, scaling the Web app is painless. The MVC nature and architecture of Angular means new components can be added non-obtrusively.

SOFTWARE ENGINEERING APP

6.4 Limitations

6.4.1 NFC support

The ability to use NFC to transfer contact information to the ESP-32 was the feature I was most interested in. After research and testing I found that there is no hardware peripheral for the ESP-32 that would allow this. If the NFC was to be used with two phones it would be possible. There are also NFC readers for computers that would accomplish this task but at the cost of scrapping the ESP-32. Perhaps the next iteration of the module will rectify this.

6.4.2 iOS Deployment

Flutter has the capability to deploy on Android, iOS and Web from the same codebase. The Bluetooth features meant that the code could only be tested on physical devices and not with an emulator. I tested it on three separate Android devices (each with a different flavour of Android) and they all worked with no hiccups. Unfortunately, Apple has restrictions that make it testing on iOS difficult.

6.5 Objectives

In the Context Chapter, I listed the functionality and features I wanted in this project. Most of these objectives were accomplished some with better than expected results. A couple did not make the final project for different reasons. I plan to return to the project after submitting to find solutions to these problems.

6.5.1 NFC and Bluetooth

NFC support was cut from the project at an early stage due to hardware limitations. This was disappointing because I also find NFC to be a fascinating technology that is underappreciated. I learned a lot about how the NFC Data Exchange Format (NDEF) data works and I'm even more interested in developing a project with it in the future. The Bluetooth features work exceptionally well. I'm very happy with how I incorporated it in my project. Again, I believe Bluetooth is underappreciated, the average user would believe it us just an audio tool. In this project, I have demonstrated that it is capable of more interesting tasks and should be investigated further.

6.5.2 Registration

Firebase was a perfect database for the Flutter app. Having user authentication handled by Firebase as well made incorporating the two a breeze. The UIDs generated by Firebase made retrieving the user's credentials, contacts and place details easier. Since Google's acquisition of Firebase, it has been their flagship for app development. It will be interesting to see how Google grows the platform in the coming years.

6.5.3 Places Contact

The Google Places API surprised me with how much of Google's data I could use in my own app. Initially I was going to use Geo-tags to accomplish this objective. But after reading through the documentation, the Places API seemed a better fit. The API also stores photos of locations which I had starting including in the app. Unfortunately, due to time constraints this feature was cut.

6.5.4 Menu/Information

This feature was cut because of an error explained in Robustness 6. This was the last feature to be added and as such it was the closest to the deadline. I spent two days debugging the problem, the more a looked into it, the more I found it to be a prevalent error. Although this was the last feature to be added, it was one of the first I wanted. Perhaps I should of started it's implementation at an earlier stage.

6.5.5 Everything wireless

I really wanted to accomplish this objective as a demonstration that the ESP-32 is a true IoT device. The battery pack works as expected, since I bought pack, I have not plugged the ESP-32 into the computer. The Elegant OTA package enables scripts to be uploaded wirelessly.

6.5.6 3D printing

The 3D prints provide some protection to the hardware and add to the project aesthetic. AutoDesk Fusion 360 and Ultimaker Cura are excellent softwares with a plethora of tutorials to create any 3D model desired.

6.5.7 IoT

Bluetooth, HTTP requests, Telnet and Over The Air uploads are all IoT communications that work homogeneously on the ESP-32. There is little configuration to enable them and great documentation to explain how to. Although with all these I/Os the ESP-32's memory is maxed out. Nevertheless, the ESP-32 has proven it is a capable IoT module and is the centre-point of the project.

6.5.8 Date Filter

Angular is difficult framework when first learning it. It's modular architecture means adding new features takes more work than simply dumping what is on the documentation into your own code. I had a couple iterations of this feature to get it to where it is. But in the end date filter works exactly how I wanted it to.

6.5.9 Email

I was unsure how I would achieve this objective. But thanks to the universality of JavaScript within the MEAN stack, there are several solutions to it. NodeMailer is a light package that fitted this need perfectly. It's implementation is what makes the Web app well-rounded.

6.5.10 Admin

Unfortunately, due to time constraints this feature was dropped entirely. I had made some progress on it with but not enough where I felt comfortable adding to the final project.

Chapter 7

Conclusion

7.1 Overview

Over all I am exceptionally happy with the outcome of this project. Working on three separate projects (Android app, NodeMCU, Web app) was a challenge. It's very easy to concentrate too much on one app and neglect the others. The Kanban board kept me on track to have some work done on each of them every two weeks. There is never going to be enough time for any project. No developer will ever say that they are "finished". So it's important to manage your time well accounting for errors and roadblocks. And in the inevitability, that you run out of time. Have the courage to make cuts and alterations for the benefit of the project as a whole. A common problem with final year projects is people leave the dissertation too late. I tended to keep my dissertation updated as I developed. This proved to be an excellent learning tool and gave me time to reflect on what I have done and maybe make alterations before it's too late. This is the largest project I have every undertaken, the management lessons I learned could not have been thought in a lecture or lab. This was the most freedom we have been given for any project and I enjoyed the challenge of making it my own.

7.1.1 Android App

Building an app and running it on my own phone was something I was excited about and it did not disappoint. I was worried about restrictions for Bluetooth and phone calls but Android's open source software encourages developers to use them. I am glad I chose Flutter because of it's ability to deploy to different operating systems from the same codebase. In future projects I will consider Flutter as an option. Firebase is on the rise with

developers, if Google continues to push the two of them as a package. I believe it will become one of the big players for mobile and web.

7.1.2 NodeMCU

The ESP-32 is an impressive piece of hardware especially for its low cost. I tended to opt for third party packages as they typically do the job better. This is a testament to the community behind it. All IoT communications work as expected and are very reliable.

7.1.3 Web App

The MEAN stack is one of the most popular developer ecosystems for its ability to produce quick, clean web pages with the awesome power of JavaScript. The app runs smoothly and is easy to interface with the ESP-32. Even though there is a steep learning curve, it is very rewarding. The flexibility of design and modular components in Angular interests me to develop more projects with the stack.

7.2 Future Developments

With national Covid-19 cases declining and no real intent on releasing the system, this project serves as a proof of concept. I still plan on returning to the project to implement unfinished features and complete all of the set objectives. After this I will publish the repository link on various forums for others the tweak and develop. While reading documentation and tutorials is how I started this project, it is the websites and forums fueled by other developers that drove me through to the end. The developer community is a fascinating ecosystem of like minded people sharing, debugging and teaching one another. Several times during development I opted for the third party alternatives because the community knows what tools and features we want and need. I look forward to using what I've learned here to build more exciting projects.

7.3 Learning Outcomes

Spanning three separate projects, my skills in all languages involved has improved. Dart and mobile development was completely new area for me. I am happy I chose the Flutter framework because I believe it is going to

grow in popularity in the coming years. The Kanban board was an essential tool for keeping track of such a large project. Learning to put time into the management of the project as well as the development is very important.

Aside from new programming skills , what I learned from this project is: Do not underestimate the length of time some features take to accomplish. This means remembering that there will be errors and unforeseen roadblocks that time must be allocated to. Even if it is an unrelated part of the project, there is a ripple effect on all the other parts. Perhaps if I spent more time considering which features would take the most time and allocate the time accordingly. Then all of the objectives would be completed. Nevertheless, I look forward to implementing these lessons in future projects.

7.4 Final Thoughts

Reflecting over the past few months, I am happy with the work I accomplished. I enjoyed all of my modules in my final year and learned a lot from them all, the freedom and scope of The Applied Project and Minor Dissertation made it the most enticing and exciting. My advice for anyone who is starting their final year is to pick a project that they are excited to develop. Start early to give the project time to mature. The project you develop may not be the project you originally set out to do. There will be unforeseen roadblocks ahead. How you react and respond to these roadblocks will largely influence the project outcome. It is a culmination of four years in college and should be treated as such. Do something your are interested in and have fun with it. Thank you to my supervisor Mr.Gerard Harrison for advising me during development and keeping me on course for the year. Thank you Dr.John Healy for the inspiration and motivation in the past two years. And thank you to all my friends and family who helped and supported me through the year.

Bibliography

[1] Will Kenton. The internet of things (iot), Feb 2020. "https://www.investopedia.com/terms/i/internet-things.asp.

[2] Health Service Executive. Covid tracker app. "https://covidtracker.gov.ie/.

[3] Google. Android bluetooth docs. "https://developer.android.com/guide/topics/connectivity/bluetooth.

[4] Google. Flutter. "https://flutter.dev/.

[5] Don Norman. Pubdev. "https://pub.dev/.

[6] Google. Android studio. "https://developer.android.com/studio.

[7] Paul DeMarco. Flutter blue. "https://pub.dev/packages/flutter_blue.

[8] Patryk Ludwikowski Ed Ufolly. Flutter bluetooth serial. "https://pub.dev/packages/flutter_bluetooth_serial.

[9] Amazon Web Service. Aws amplify. "https://aws.amazon.com/amplify/.

[10] Google. Google maps platform. "https://developers.google.com/maps/documentation/places/web-service/overview.

[11] Matt Hamblen. A short history of nfc. https://www.computerworld.com/article/2493888/a-short-history-of-nfc.html.

[12] CableFree. The history of wifi: 1971 to today. "https://www.cablefree.net/wireless-technology/history-of-wifi-technology/.

[13] Tracy V. Wilson Bernadette Johnson Marshall Brain. How wifi works. "https://computer.howstuffworks.com/wireless-network.htm.

[14] Robert Triggs. A quick history of bluetooth. "https://www.androidauthority.com/history-bluetooth-explained-846345/.

[15] Bluetooth. How bluetooth technology uses adaptive frequency hopping to overcome packet interference. "https://www.bluetooth.com/blog/how-bluetooth-technology-uses-adaptive-frequency-hopping-to-overcome-packet-

[16] Near Communications. Adaptative frequency hopping. https://sites.google.com/site/nearcommunications/adaptative-frequency-hopping.

[17] TWI Global. What is rapid prototyping - definition, method and advantages. https://www.twi-global.com/technical-knowledge/faqs/faq-manufacturing-what-is-rapid-prototyping.

[18] Jorge Peralta. Thingiverse: 18650 battery shield case. https://www.thingiverse.com/thing:4672332.

[19] AutoDesk. Autodesk 360. https://www.autodesk.com/.

[20] Latex. The latex project. "https://www.latex-project.org/.

[21] Overleaf. Overleaf. "https://www.overleaf.com/.

[22] Espressif Systems. Esp-32 package download. https://dl.espressif.com/dl/package_esp32_index.json.

[23] Espressif Systems. Esp-8266 package download. http://arduino.esp8266.com/stable/package_esp8266com_index.json.

[24] Silicon Labs. Usb drivers. https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers.

[25] Bangood. Battery pack. https://www.banggood.com/ESP32S-ESP32-0_5A-Micro-USB-Charger-Board-18650-Battery-Charging-Shield-witl
html?cur_warehouse=CN.

[26] GitHub Forums. Error: Not found: 'dart:html'. https://github.com/flutter/flutter/issues/53005.