Diploma in

[Robotics and Mechatronics]

Overseas Internship Programme Report

Title:

[TemiChoco Questionnaire App]

NYP Supervisor:

[Dr Edwin Foo]

Student:

[Chua Choon Hsiang (191067F)]

Report Template

Acknowledgement

Table of Contents

1. Introduction
   * Background / System Overview
   * Problem
   * Project Aim

2. Objective(s)

3. Scope
   * Key Specifications & Features
   * Major Deliverables
   * Gantt Chart

4. Implementation
   * Hardware Development
   * Software Development
   * Integration and Testing
   * Key Technical Issues and Solutions
   * Flowchart
   * Specifications of Hardware and Software

5. Conclusion
   * Accomplishment
   * Future Enhancement

Appendices

Appendix A – Project Resources
Appendix B – Website References
Appendix C – Sample Code References
Appendix D – Video References

Introduction

Background Information

To continue a project done by a student in 2022, which is a 3D printed KitKat dispensing machine for Temi to attract students to learn more about NYP. Using Annikken Andee (Andee Android), a Annikken connected to an Arduino UNO, he created a quiz program, where a user inputs a correct answer, it will dispense a chocolate using Bluetooth communication, to control the servo. This project requires the replacement of Andee Android by using Android Studio to create a questionnaire App with Bluetooth feature using Java. Due to Andee Android limitations such as its limited UI app features and its usage of C program, Java is better choice for app development.

Annikken Andee is a hardware that used to connect the popular Arduino platform to smartphones and tablets. Annikken Andee have a few models which consist of the Andee U, Andee Android, and the Arduino 101 board. It is an Arduino shield (excluding Arduino 101) with a Bluetooth module, attached to the top of an Arduino board. It is capable of transmitting packets of data between the Arduino and smart devices using Bluetooth. Annikken Andee provides the convenience of creating an app without mobile app programming because it solves the complexities of displaying app widgets on a device and Bluetooth communication using their provided library source code. However, Annikken Andee has limited app widgets and its code is in C program which restricts outreach of app development for temi, Java and Kotlin are the preferred programming languages.

Built on JetBrains' IntelliJ IDEA software and created exclusively for Android development, Android Studio is the recognized integrated development environment (IDE) for Google's Android platform. On Windows, macOS, and Linux-based operating systems, it can be downloaded. As the main IDE for the development of native Android applications, supersedes the position of the Eclipse Android Development Tools (E-ADT). It is compatible with Java, Kotlin and C++ programming languages and more with extensions.

The Bluetooth Special Interest Group (Bluetooth SIG) created and marketed Bluetooth Low Energy (Bluetooth LE, also known as BLE and formerly marketed as Bluetooth Smart), a wireless personal area network technology intended for cutting-edge applications in the home entertainment, fitness, and security sectors. Although it lacks interoperability with conventional Bluetooth and is independent of it, Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR) and LE can coexist. The original specification, known as Wibree, was created by Nokia in 2006 and was incorporated as Bluetooth Low Energy in Bluetooth 4.0 in December 2009.

Bluetooth Low Energy is designed to offer significantly lower power and cost while keeping a similar communication range as compared to Classic Bluetooth. Native support for Bluetooth Low Energy is provided by the mobile operating systems iOS, Android, Windows Phone, BlackBerry, Linux, macOS, Windows 8, Windows 10, and Windows 11.

temi USA inc. is a multinational robotics company that specializes in Robot as a Service solutions (RaaS), autonomous platforms, Al, smart assistant, and cloud-based services. One of their current robots is temi three. It is cost effective, capable of 24/7 autonomous service for everyday tasks with focus on engagement efficiency. It is being used in many sectors such as retail, healthcare, business and more etc. NYP has an older version temi 2 that is being used to provide guided tours during open house to attract new students to NYP and spark their interest in robotics.

Problem

Temi is lacking an eye-catching app to interest new students about robotics and what NYP can provide the students during NYP open house.

Project Aim

Develop software modules Temi robot's outreach.

System Overview

- To develop hardware and software modules to activate the chocolate dispenser (Previous student work) using Bluetooth BLE.
- Integrate the chocolate dispenser into Temi robot.
- Program the Temi robot to interact with users.

Objectives

1st Step

Build a proficiency for app development by learning basic Java programming for coding apps and Android Studio usage for app creation.

2nd Step

To develop a testing questionnaire app interface to test if app works.

3rd Step

Research on how to develop a Bluetooth feature in an app using Android Studio.
Understand the differences between Bluetooth and Bluetooth Low Energy implementations.

4th Step

Develop a Bluetooth BLE testing app to evaluate Bluetooth communications between app and Arduino with the use Adafruit Bluefruit LE Friend UART (BLE)

5th Step

Improving the 3D printed dispenser.

- Build mounts for Arduino and Adafruit Bluefruit LE Friend UART (BLE)
- Secure Loose screws and add rubber tips for ground support.

Outcome

Compile and create app final build (Temi Choco). Testing and debugging the app to make sure it has a smooth-running questionnaire app with Bluetooth communication feature for dispensing chocolate.

Scope

Key Specifications & Features

The educational questionnaire interface app is created by using Android Studio with Java programming language. The app contains three types of questions, which are multiple choice question (MCQ), pictorial MCQ and short-structured question.

There is a total of ten questions, six MCQs, three pictorial MCQs and three short-structured questions. Five out of ten questions will be picked randomly for each quiz attempt to make sure a unique questionnaire for each user.

The app includes a Kahoot background music when no user is doing the quiz to attract students. App interface is also designed with a colourful design to make user interested. A results page is created at the end of five questions to show how the user did for the quiz. As an educational app, everyone would be given at least 1 chocolate to empower user's desires to try the quiz. While for knowledgeable users if they took less than 8 tries to answer 5 questions, they are rewarded 2 chocolates as an achievement of their hard work.

Using Adafruit Bluefruit LE Friend UART (BLE), a Bluetooth feature is added to the app. Which allows the app to be able to communicate with the Arduino board in the chocolate dispenser to control its servo to dispense chocolate. The Bluetooth feature is using Bluetooth Low Energy instead of Classic Bluetooth because its energy efficient, cost efficient and more suited for handling simple data communications such as dispensing chocolates.

3D printed Chocolate has a mount for the Arduino Board and the Bluefruit LE Friend UART is (BLE) is attached to the side of the dispenser in the wiring compartment. The dispenser's screws have been tightened and secured to prevent a structural integrity failure occurring during its usage. Rubber tips were added for better ground support.

Major Deliverables

At the end of this project, TemiChoco should be able to integrate into Temi and be able to be interacted with users. TemiChoco should have a random set of questions for each user's quiz attempt.

TemiChoco's Bluetooth feature (Bluetooth Low Energy) should be able to switch on/off Bluetooth, scan for Bluetooth compatible devices to connect with and lastly it should be able to communicate with Bluefruit LE Friend UART (BLE) using to activate the 3D printed Design to dispense chocolate.

The dispenser should be to stand flat on Temi robot's back and function normally, and the chocolates should be dispensed smoothly.

Project Gantt Chart

ChocoTemi Gannt Chart	WEEK																							
Tasks:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Selection of Project	🟥																							
	🟩																							
Download and Checking required Tools	🟥																							
	🟩																							
Learn Java Programming	🟥	🟥	🟥	🟥	🟥	🟥	🟥																	
	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	
Learn Android Studio Usage			🟥	🟥	🟥	🟥	🟥																	
			🟩	🟩	🟩	🟩	🟩	🟩																
Create App Interface Template							🟥	🟥	🟥	🟥														
							🟩	🟩	🟩	🟩	🟩													
Create Bluetooth Feature											🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥				
												🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩		
Create Final Build																		🟥	🟥	🟥	🟥	🟥	🟥	
																		🟩	🟩	🟩	🟩	🟩	🟩	
Creating presentation slides												🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	
												🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	
Gannt chart				🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
				🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Adapting to Japan				🟥																				
				🟩	🟩	🟩																		
Learning Japanese	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥										
	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩									
MILESTONES:																								
Mid Intern Presentation (Wk 13)													🟩											
Jap Intern Presentation (Wk 16)															🟩									
Final Intern Presentation (Week 23)																							🟩	
Final Intern Report (Week 24)																								🟩

🟥	Planned Duration
🟩	Actual Duration
🟪	Japan Arrival Week
🟪	Trip Week
🟦	Return to Singapore

Implementation

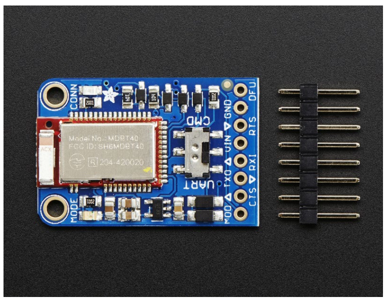
Hardware Development

Materials Used:

- Screws and nuts.
- Arduino UNO.
- Arduino mounting plate.
- Arduino USB cable.
- Male to male & male to female jumper wires.
- Adafruit Bluefruit LE Friend UART (BLE).
- 3D printed dispenser with servo.
- Chocolates.

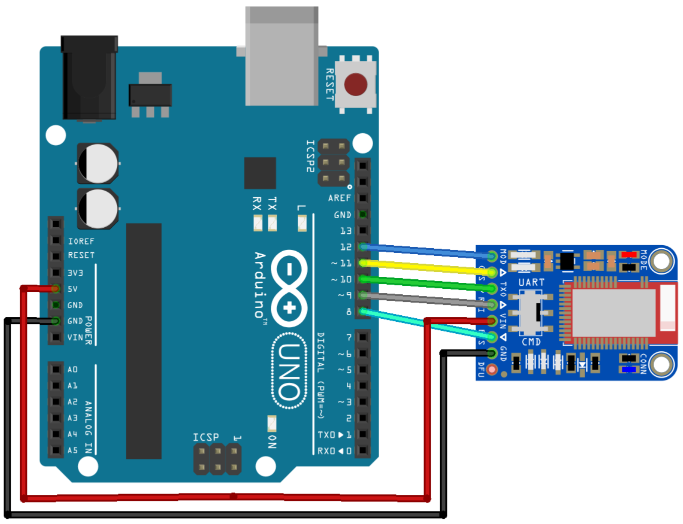
Adafruit Bluefruit LE Friend UART (BLE)'s connector pins were not welded for jumper wires to connect to the Arduino. Welding was required for the connecter pins to the ports of the equipment. Male to female jumper wires were required to connect the Bluefruit LE Friend UART (BLE) to Arduino UNO. Adafruit website and guidance from Dr Foo, can be used as reference to identify which port to connect the wires between the devices.

Adafruit Bluefruit LE Friend UART with connecter pins unwelded



Wiring of Adafruit Bluefruit LE Friend UART to Arduino UNO

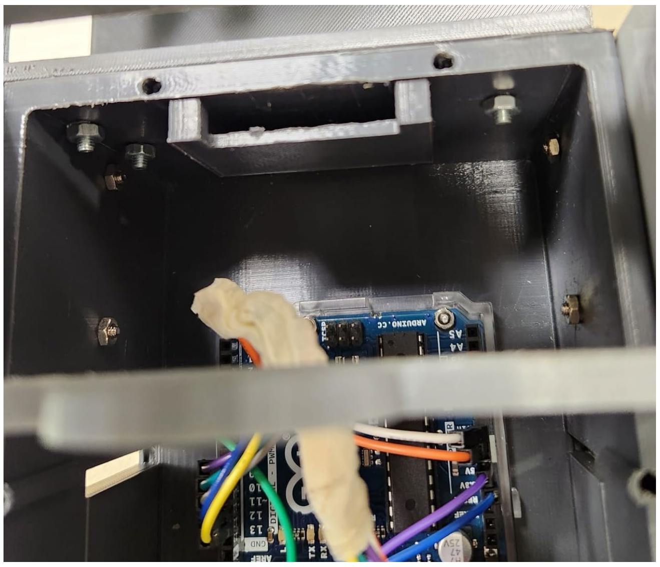
- **MOD** to **Pin 12**
- **CTS** to **Pin 11**
- **TXO** to **Pin 10**
- **RXI** to **Pin 9**
- **VIN** to **5V**
- **RTS** to **Pin 8**
- **GND** to **GND**



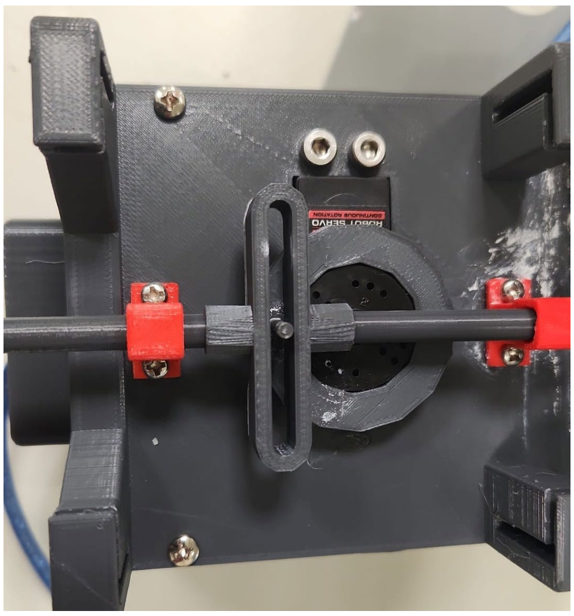
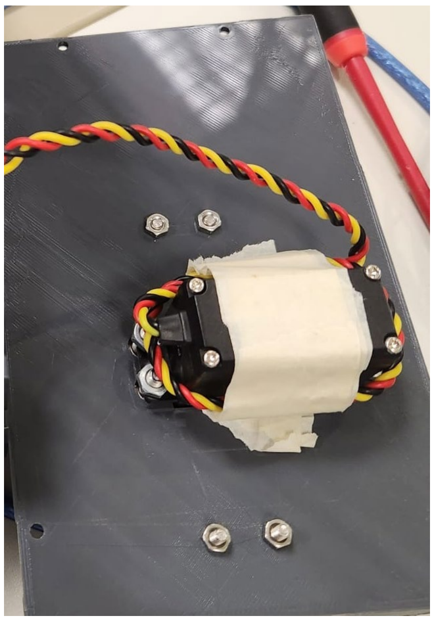
During inspection of the 3D printed dispenser, it was found that the screws that hold the parts together do not have nuts to keep them secure. Which after a period of wear and tear, it created a structure integrity failure. The loose screws on the dispenser were secured and tightened with nuts corresponding their screw size which are mostly M3 and M4.

<u>Images of secured screws</u>

Chocolate Container (Left) & Wiring compartment (Right)

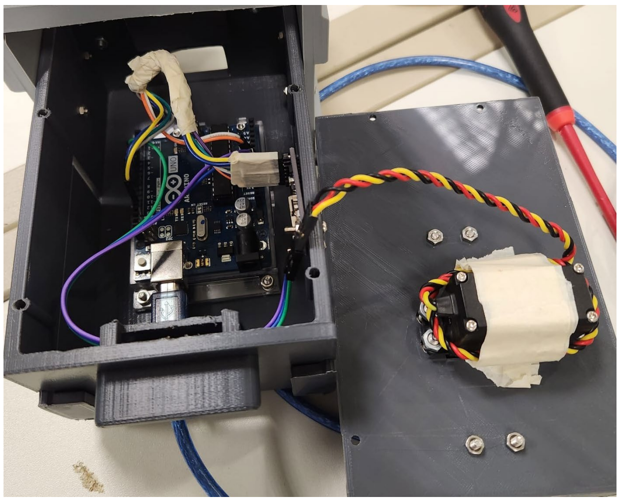


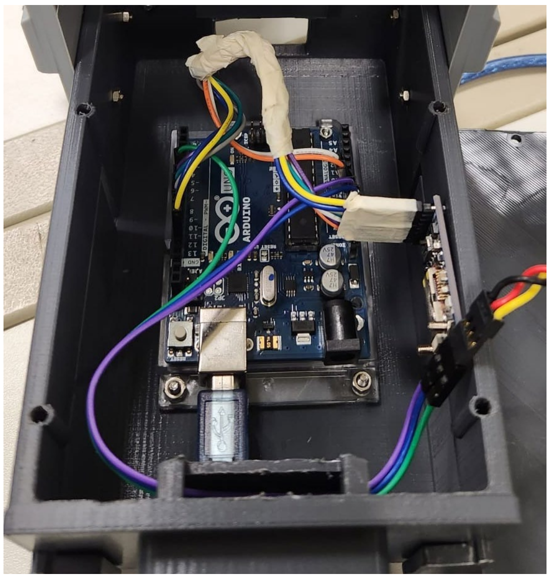
Top View (Left) & Bottom view (Right) of the lid of the wiring compartment(Right)

A mount was provided for the Arduino UNO, it attached to the middle of the dispenser's electrical and wiring compartment with screws. The Bluefruit LE Friend UART (BLE) was screwed to the right side of the wall of the compartment and connected it to the Arduino board. The servo to Arduino UNO were connected using male to male jumper wires, where the servo's signal wire (yellow) is connected to digital port 6 of the Arduino UNO's to send signals for servo movement. While the servo's power (red) and ground (black) wires were connected to Arduino UNO's 5V port for power and GND port for ground. To make the wiring have a neat presentation, the wires were tapped together. Lastly, a USB cable is connected to Arduino UNO as a power source and coding the servo movement.

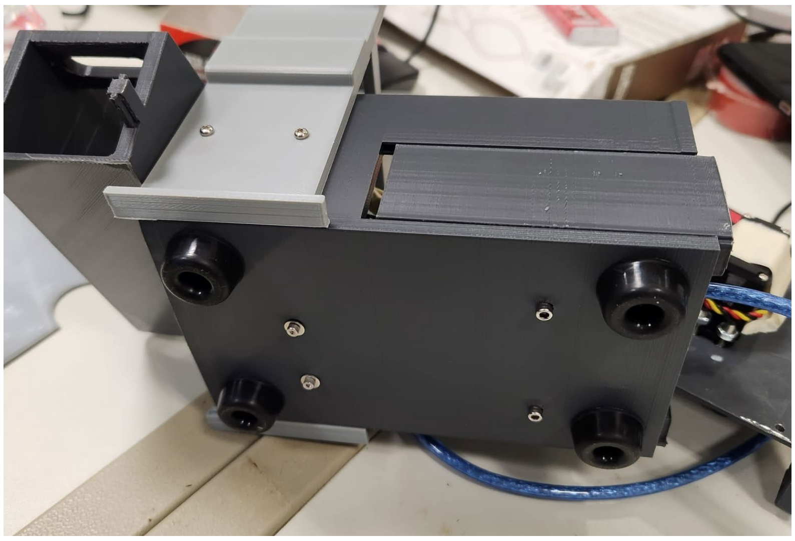
<u>Image of completed mounting of devices and their wirings.</u>





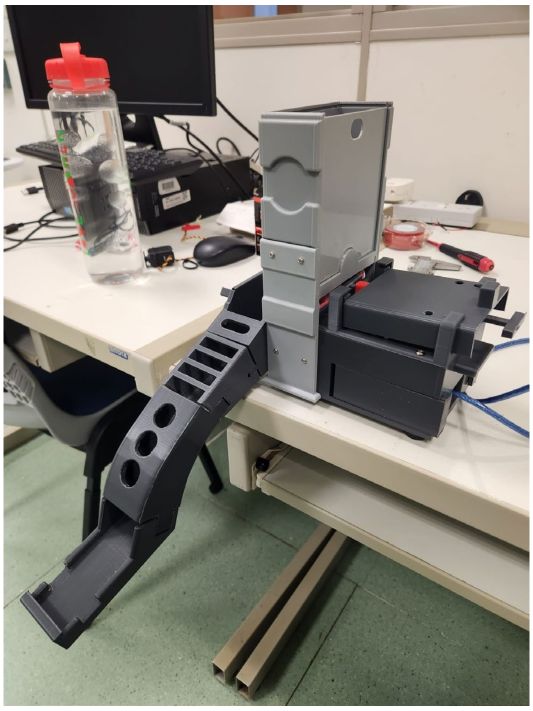
After the completion of wiring and tightening of screws, investigations found out the bottom of the dispenser can no longer be placed flat on the ground due to 2 mounting screws for the Arduino UNO. The solution was to add rubber tips to the edges of the dispenser.

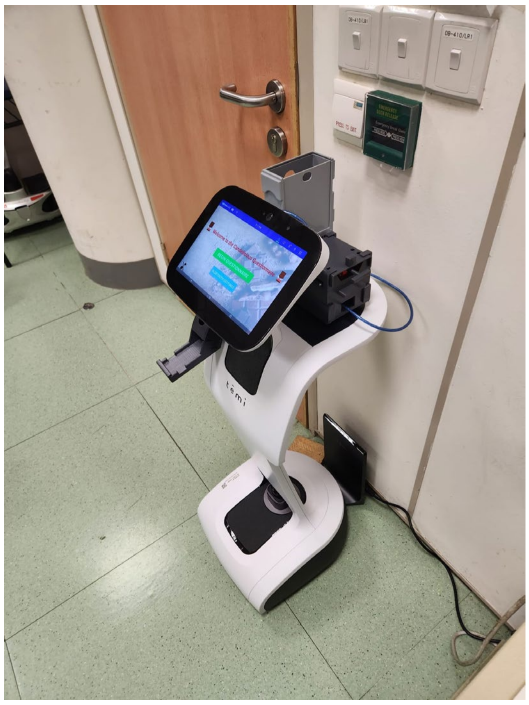
<u>Solution Image</u>



During testing of the code, the chocolates were dispensed smoothly from the dispenser.

Image and Video of 3D printed dispenser during final testing.



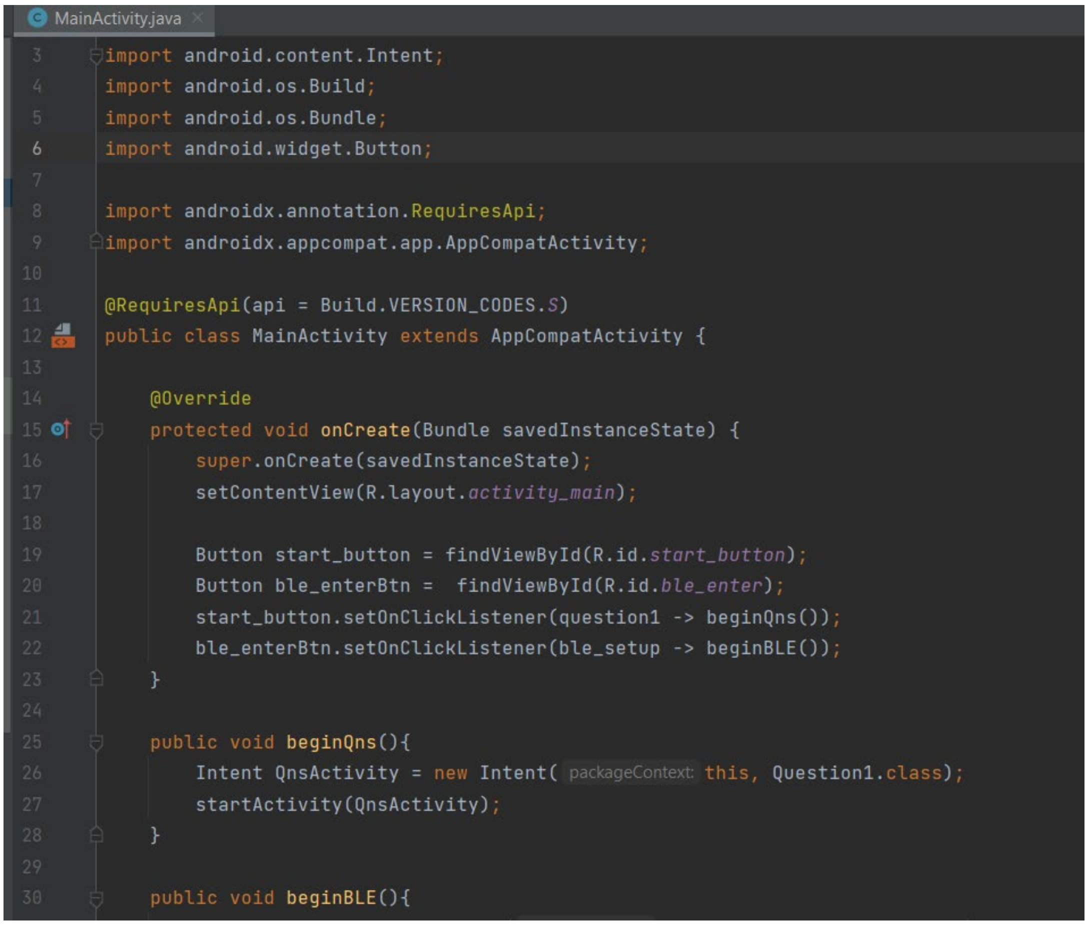


Software Development

Using Android Studio for app development and implementation of Bluetooth in app requires Android Studio and Java programming knowledge. For learning of Android Studio and programming in Java, YouTube videos and guide websites like Stack overflow, GitHub and Medium can be used for references.

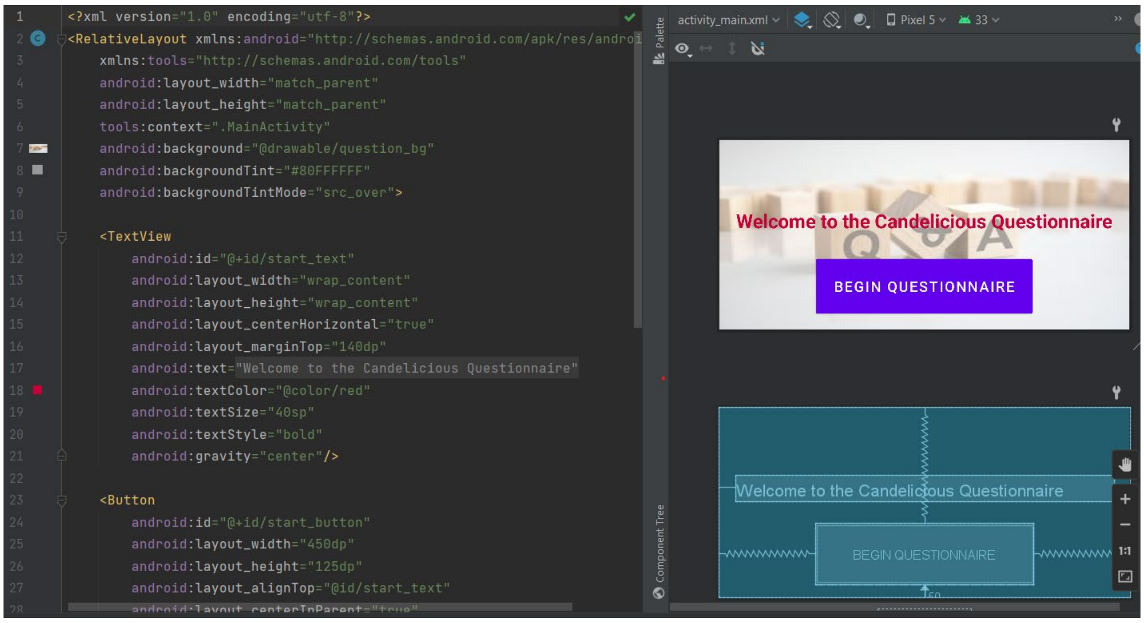
To test implementation of an android app, a test quiz app (EGR326 interface) was created. It contains a main menu, Bluetooth menu, 5 questions, a result page using Android Studio. It uses activities for the app's contents. An activity is the body code of an app's content, a xml file is required to design an app's appearance and a layout type such as relative layout, constraint layout and linear layout is used to store widgets like buttons, text display boxes, images, and more. In this testing app, relative layout is used because it provides the most flexibility in designing an app's appearance. XML resource files are also needed for images, colours (in hex code) used for design purpose and hardcoded text, theses files are stored in their respective resource folders. Example a layout folder is for layouts, colour folder is for colour, app theme style is in the themes folder and more etc. These files are separated from the activity as they belong to the resource folder of a project while activities are in a Java folder. For an activity to access a layout's widgets they use findViewById to find the pre-set resource ID of the specific widget. Lastly, imports are modules we use for coding widgets, functions, Bluetooth and more depending on what the code has, which belongs the library of the IDE which is Android Studio.

Image of an Activity for main menu


MainActivity.java

```java
import android.content.Intent;
import android.os.Build;
import android.os.Bundle;
import android.widget.Button;

import androidx.annotation.RequiresApi;
import androidx.appcompat.app.AppCompatActivity;

@RequiresApi(api = Build.VERSION_CODES.S)
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button start_button = findViewById(R.id.start_button);
        Button ble_enterBtn =  findViewById(R.id.ble_enter);
        start_button.setOnClickListener(question1 -> beginQns());
        ble_enterBtn.setOnClickListener(ble_setup -> beginBLE());
    }

    public void beginQns(){
        Intent QnsActivity = new Intent( packageContext: this, Question1.class);
        startActivity(QnsActivity);
    }

    public void beginBLE(){
```

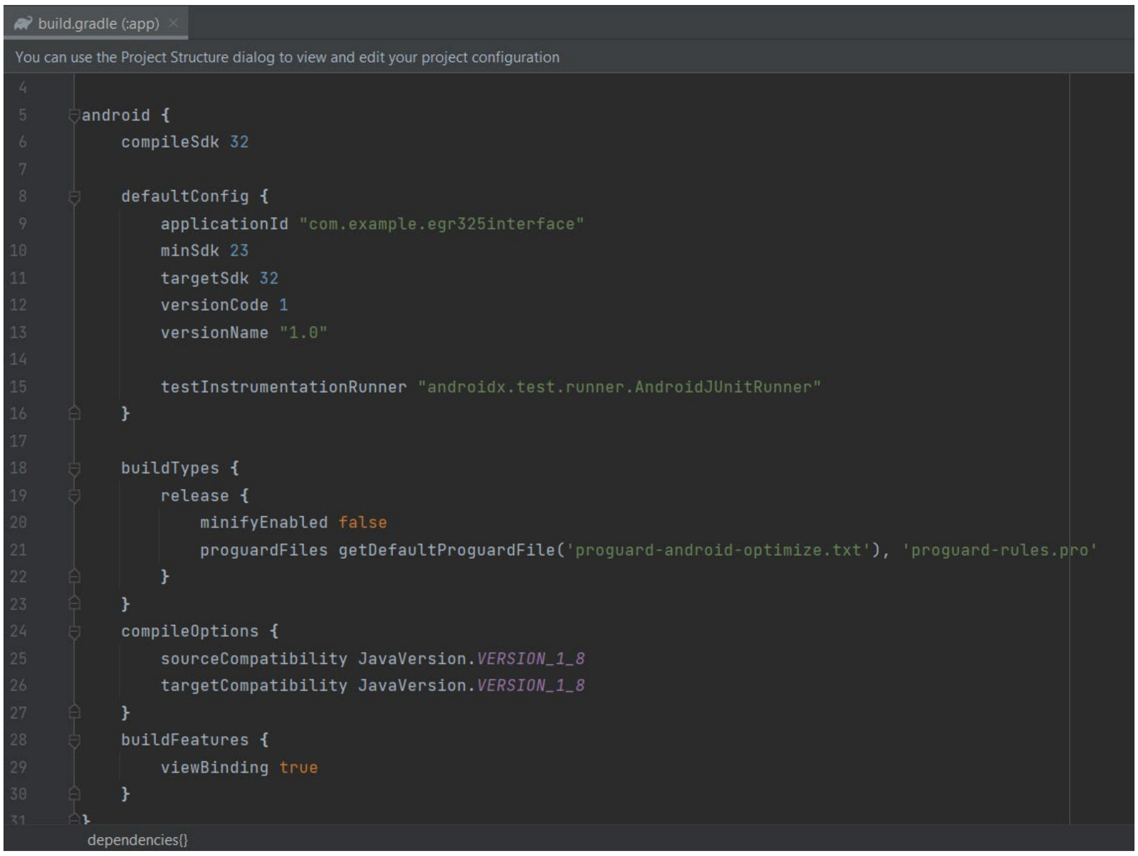
Image of a xml file using Relative Layout for main menu with design codes for widgets



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/androi
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:background="@drawable/question_bg"
    android:backgroundTint="#80FFFFFF"
    android:backgroundTintMode="src_over">

    <TextView
        android:id="@+id/start_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="140dp"
        android:text="Welcome to the Candelicious Questionnaire"
        android:textColor="@color/red"
        android:textSize="40sp"
        android:textStyle="bold"
        android:gravity="center"/>

    <Button
        android:id="@+id/start_button"
        android:layout_width="450dp"
        android:layout_height="125dp"
        android:layout_alignTop="@id/start_text"
        android:layout_centerInParent="true"
```

During the creation of the app, in a investigate to find out a compilation error of the application after creating an activity and layout (Main Menu). It was found that Android Studio uses Gradle, a build tool to create the app from the code. The Gradle needs to have the latest/compatible version of the project where the app is created, because some tools or code being used to create an app maybe outdated or required the latest version of its dependencies.

Image of testing app's project Gradle settings



build.gradle (:app)

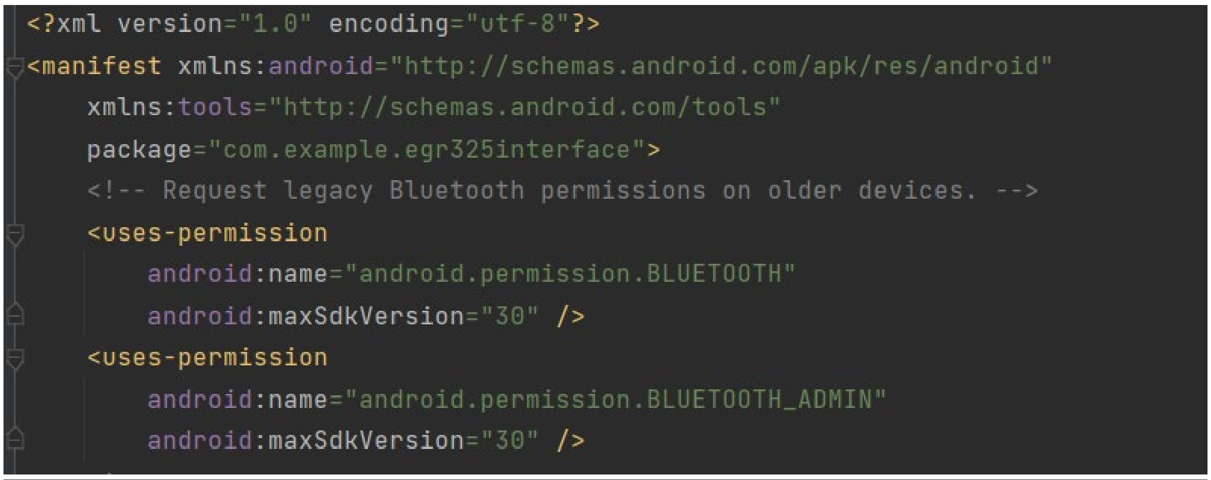
You can use the Project Structure dialog to view and edit your project configuration

```

android {
    compileSdk 32

    defaultConfig {
        applicationId "com.example.egr325interface"
        minSdk 23
        targetSdk 32
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    buildFeatures {
        viewBinding true
    }
}
```

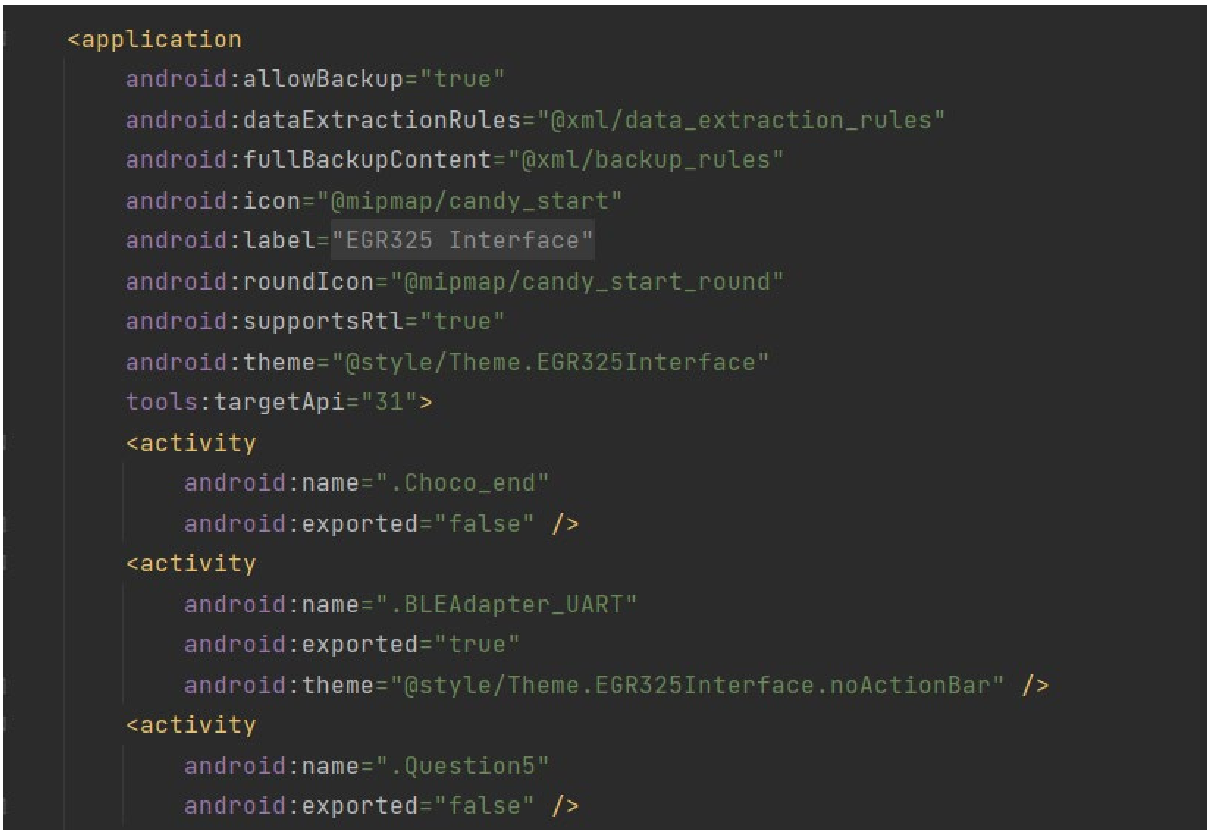
dependencies{}

Additionally, it was found that an Android manifest file is needed at the root of the project application source set because the activities' code was unable to be read by Gradle. The manifest file describes essential information about your app to the Android build tools, the Android operating system, and Google Play.

Snippets of Testing App's Android Manifest



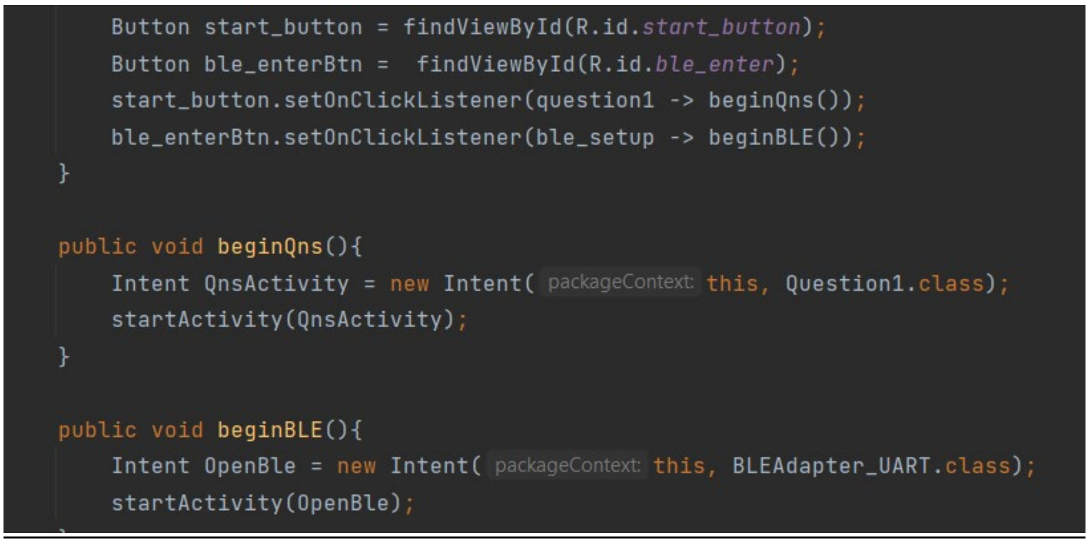
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.egr325interface">
    <!-- Request legacy Bluetooth permissions on older devices. -->
    <uses-permission
        android:name="android.permission.BLUETOOTH"
        android:maxSdkVersion="30" />
    <uses-permission
        android:name="android.permission.BLUETOOTH_ADMIN"
        android:maxSdkVersion="30" />
```



```
   <application
       android:allowBackup="true"
       android:dataExtractionRules="@xml/data_extraction_rules"
       android:fullBackupContent="@xml/backup_rules"
       android:icon="@mipmap/candy_start"
       android:label="EGR325 Interface"
       android:roundIcon="@mipmap/candy_start_round"
       android:supportsRtl="true"
       android:theme="@style/Theme.EGR325Interface"
       tools:targetApi="31">
       <activity
           android:name=".Choco_end"
           android:exported="false" />
       <activity
           android:name=".BLEAdapter_UART"
           android:exported="true"
           android:theme="@style/Theme.EGR325Interface.noActionBar" />
       <activity
           android:name=".Question5"
           android:exported="false" />
```

Buttons were used to monitor user inputs, to be able to execute functions like moving to next questions, answering a question and more.

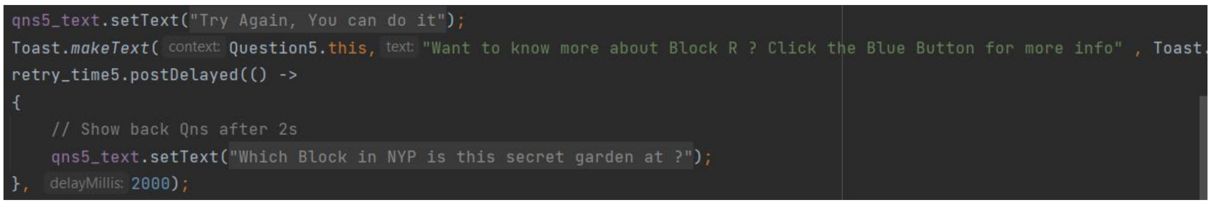
<u>Image of Main Menu code to move to either Bluetooth menu or start Quiz using Button.</u>



```
        Button start_button = findViewById(R.id.start_button);
        Button ble_enterBtn =  findViewById(R.id.ble_enter);
        start_button.setOnClickListener(question1 -> beginQns());
        ble_enterBtn.setOnClickListener(ble_setup -> beginBLE());
    }

    public void beginQns(){
        Intent QnsActivity = new Intent( packageContext: this, Question1.class);
        startActivity(QnsActivity);
    }

    public void beginBLE(){
        Intent OpenBle = new Intent( packageContext: this, BLEAdapter_UART.class);
        startActivity(OpenBle);
```

Textview, Android Studio's text display boxes, were used to display text to users, to display questions, welcoming text, text for the reading and more. A Handler is used as a delay to show a text for a given period, it uses postdelay( ) function. Handlers is also used in other parts of the apps, as a time related issue is involved.

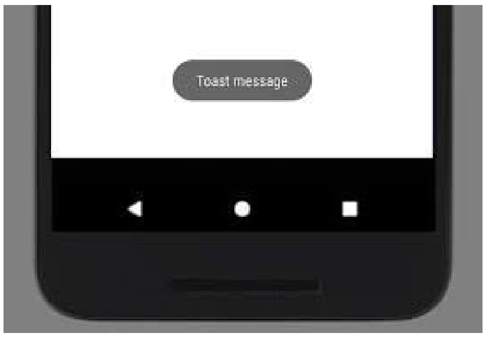
<u>Image of Textview being use for notifying user's results on a question.</u>



```
qns5_text.setText("Try Again, You can do it");
Toast.makeText( context: Question5.this, text: "Want to know more about Block R ? Click the Blue Button for more info" , Toast.
retry_time5.postDelayed(() ->
{
    // Show back Qns after 2s
    qns5_text.setText("Which Block in NYP is this secret garden at ?");
}, delayMillis: 2000);
```

Toast messages was used to display extra information to user for this project one of its uses is to display a reason for wrong answer.

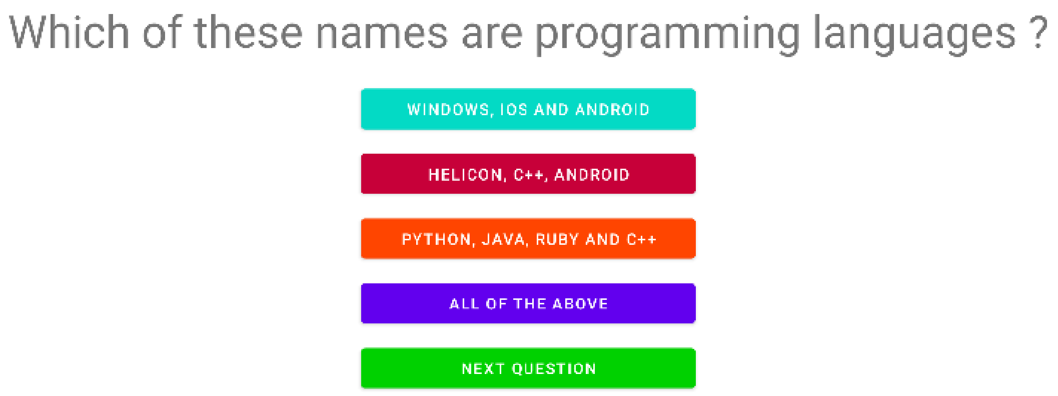
<u>Example of Toast message</u>



There are three types of questions in the test quiz app and their example images.

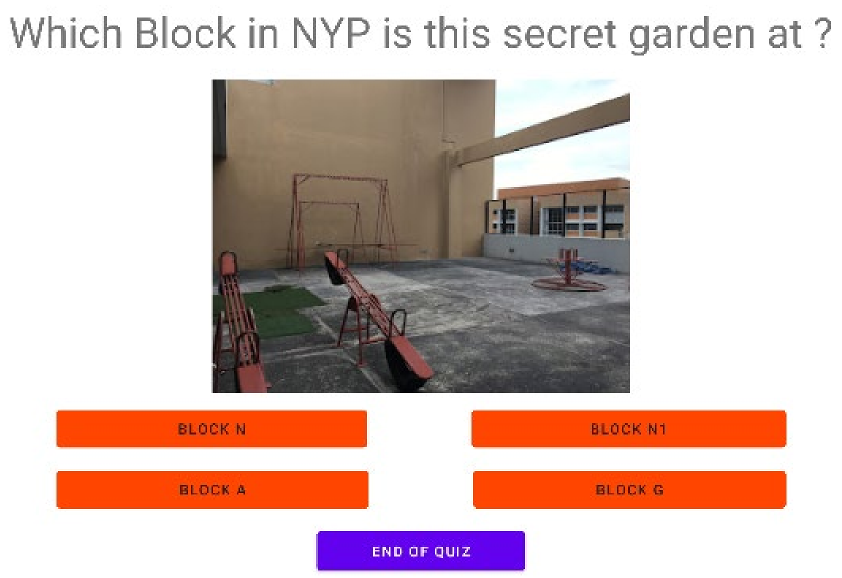
As the questionnaire is for educational purposes, user must answer correctly to proceed to next question.

- Multiple choice questions (MCQ)



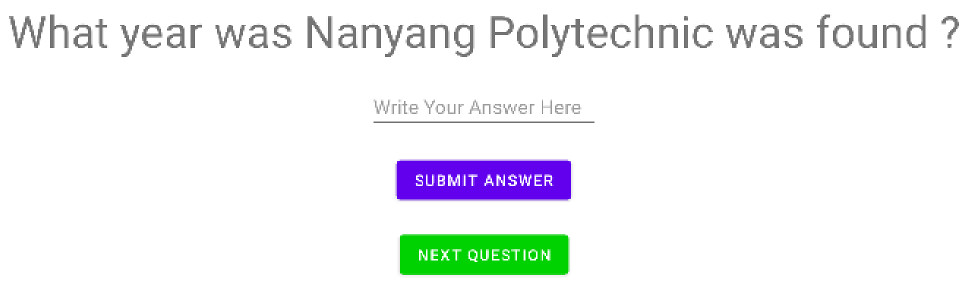
Which of these names are programming languages ?

- Pictorial Multiple-choice questions (PMCQ)



Which Block in NYP is this secret garden at ?

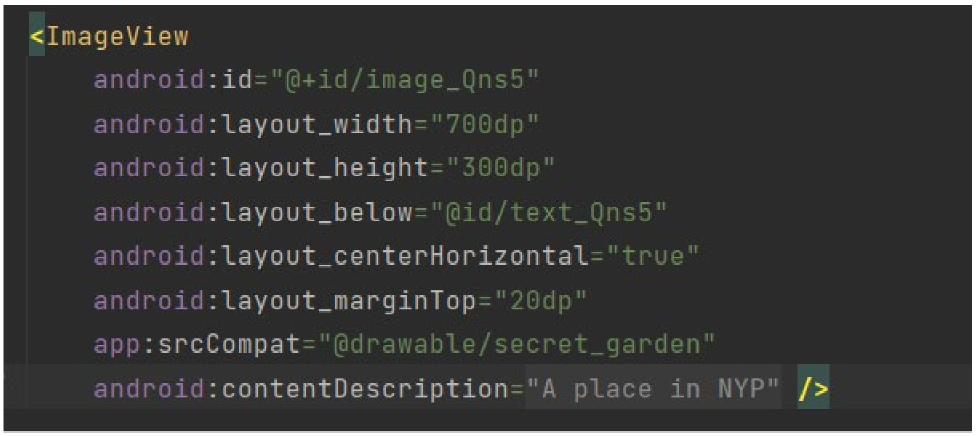
- Short Structure questionnaires (SSQ)



What year was Nanyang Polytechnic was found ?

The difference between the MCQ and PMCQ is the usage of an image which is done by using ImageView, Android Studio's app widget for showing images to users. Images used are stored in an XML resource folder named drawable, where images and vector assets are stored.

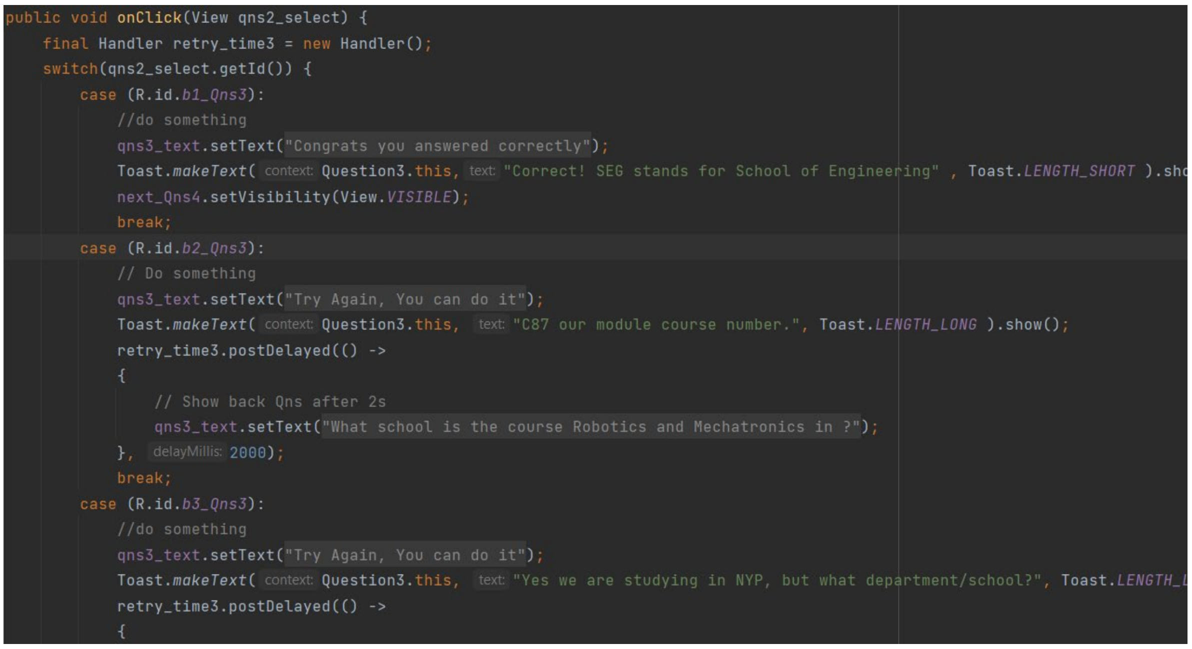
Image of ImageView being use for showing a PMCQ.



```xml
<ImageView
    android:id="@+id/image_Qns5"
    android:layout_width="700dp"
    android:layout_height="300dp"
    android:layout_below="@id/text_Qns5"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="20dp"
    app:srcCompat="@drawable/secret_garden"
    android:contentDescription="A place in NYP" />
```

The MCQ types of questions uses a switch case method to determine if the user input is correct. As it each app widget has its own data ID, using a switch case method to identify if the button pressed by the user is either correct or wrong.

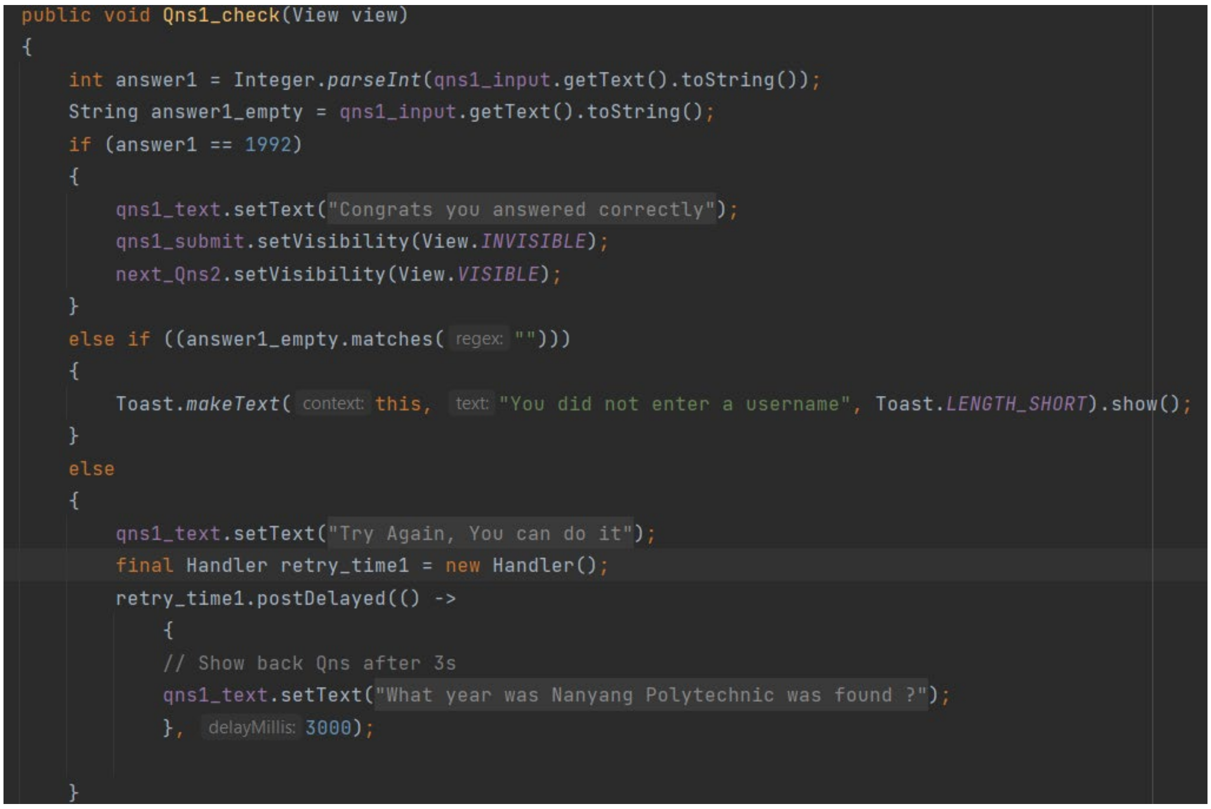
Image of a MCQ question code



```java
public void onClick(View qns2_select) {
    final Handler retry_time3 = new Handler();
    switch(qns2_select.getId()) {
        case (R.id.b1_Qns3):
            //do something
            qns3_text.setText("Congrats you answered correctly");
            Toast.makeText( context: Question3.this, text: "Correct! SEG stands for School of Engineering" , Toast.LENGTH_SHORT ).sho
            next_Qns4.setVisibility(View.VISIBLE);
            break;
        case (R.id.b2_Qns3):
            // Do something
            qns3_text.setText("Try Again, You can do it");
            Toast.makeText( context: Question3.this,  text: "C87 our module course number.", Toast.LENGTH_LONG ).show();
            retry_time3.postDelayed(() ->
            {
                // Show back Qns after 2s
                qns3_text.setText("What school is the course Robotics and Mechatronics in ?");
            },  delayMillis: 2000);
            break;
        case (R.id.b3_Qns3):
            //do something
            qns3_text.setText("Try Again, You can do it");
            Toast.makeText( context: Question3.this,  text: "Yes we are studying in NYP, but what department/school?", Toast.LENGTH_L
            retry_time3.postDelayed(() ->
            {
```

While for short-structure question it uses a pre-set answer and compare it with the user's input using the app widget Editview, which takes user text inputs.

Image of a short-structure question code (Numbered answer)



```
public void Qns1_check(View view)
{
    int answer1 = Integer.parseInt(qns1_input.getText().toString());
    String answer1_empty = qns1_input.getText().toString();
    if (answer1 == 1992)
    {
        qns1_text.setText("Congrats you answered correctly");
        qns1_submit.setVisibility(View.INVISIBLE);
        next_Qns2.setVisibility(View.VISIBLE);
    }
    else if ((answer1_empty.matches( regex: "")))
    {
        Toast.makeText( context: this,  text: "You did not enter a username", Toast.LENGTH_SHORT).show();
    }
    else
    {
        qns1_text.setText("Try Again, You can do it");
        final Handler retry_time1 = new Handler();
        retry_time1.postDelayed(() ->
            {
            // Show back Qns after 3s
            qns1_text.setText("What year was Nanyang Polytechnic was found ?");
            },  delayMillis: 3000);

    }
```

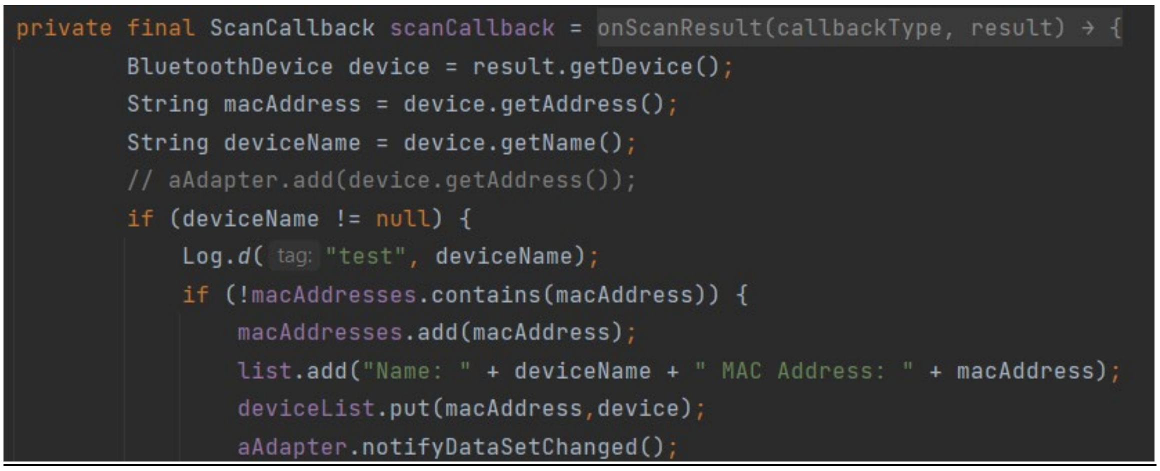
A Bluetooth test app (Blelist) was created for testing Bluetooth communication implementation in an Android app to make sure the quiz app codes would not collide and crash with the Bluetooth code. After the testing, the apps were integrated into in one using quiz testing app.

The app widgets used in the Blelist app are buttons and a Listview, and a custom-made tool bar with buttons for going back main page and assessing the dispenser and a Textview for the app's title in Bluetooth menu.

The buttons are used for listening to user inputs, such as toggling Bluetooth, request of dispensing chocolate by sending data to Arduino UNO via Bluetooth communications and requesting of scanning devices.

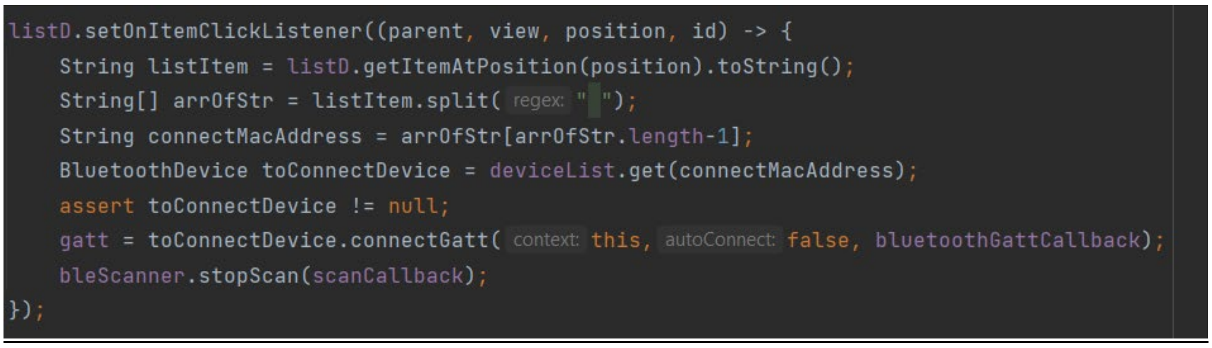
Listview is Android's Studio's app widget to provide a list of data the users can view, for this project it is used to display what devices can be connected to such as Adafruit Bluefruit LE friend UART and another other device that has Bluetooth Low Energy module in it. The data is extracted from Bluetooth Call-back codes, and Listview uses a Array Adapter to populate the list with data. The Listview can also be used to listen to user inputs, which in this app is used to connect to Bluefruit LE Friend UART, to allow the app to communicate with the Arduino. The Listview has a custom text layout created to change the colour and size of the text in Listview.

Image of the Listview code to get Data for users to view.



```
private final ScanCallback scanCallback = onScanResult(callbackType, result) → {
        BluetoothDevice device = result.getDevice();
        String macAddress = device.getAddress();
        String deviceName = device.getName();
        // aAdapter.add(device.getAddress());
        if (deviceName != null) {
            Log.d( tag: "test", deviceName);
            if (!macAddresses.contains(macAddress)) {
                macAddresses.add(macAddress);
                list.add("Name: " + deviceName + " MAC Address: " + macAddress);
                deviceList.put(macAddress,device);
                aAdapter.notifyDataSetChanged();
```

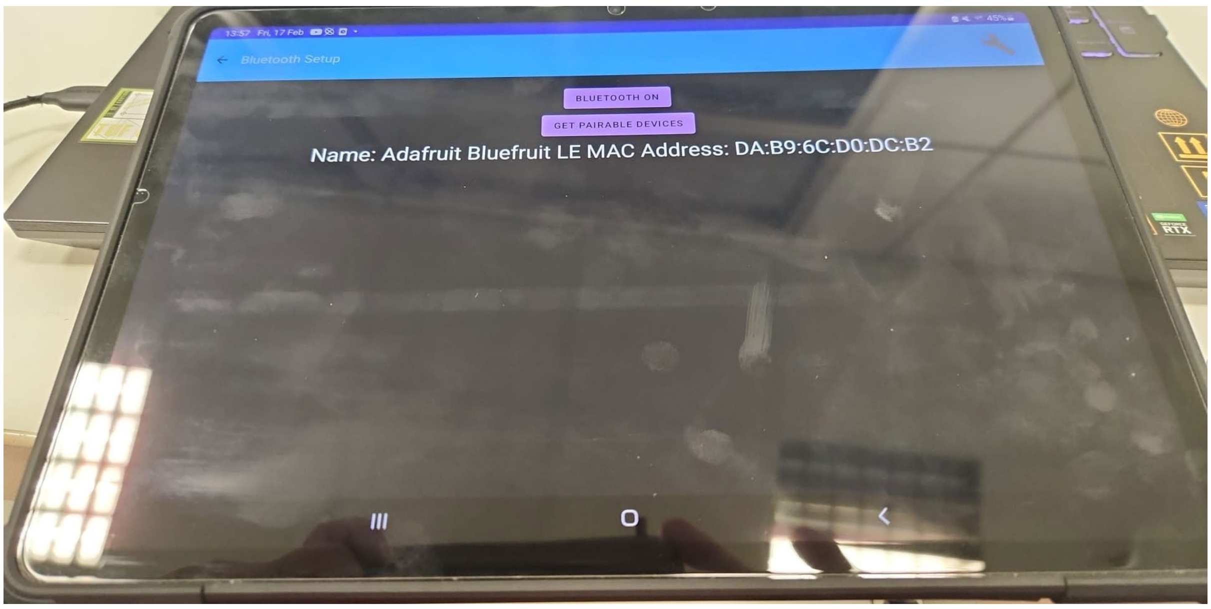
Image of the Listview code listening to user input to connect to a device.



```
listD.setOnItemClickListener((parent, view, position, id) -> {
    String listItem = listD.getItemAtPosition(position).toString();
    String[] arrOfStr = listItem.split( regex: " ");
    String connectMacAddress = arrOfStr[arrOfStr.length-1];
    BluetoothDevice toConnectDevice = deviceList.get(connectMacAddress);
    assert toConnectDevice != null;
    gatt = toConnectDevice.connectGatt( context: this, autoConnect: false, bluetoothGattCallback);
    bleScanner.stopScan(scanCallback);
});
```

A custom tool was added to be able to create a button to test the implementation of sending data to Arduino UNO to dispense chocolate and receive information for it. A back button was added when being integrated to test quiz app to go back main menu.

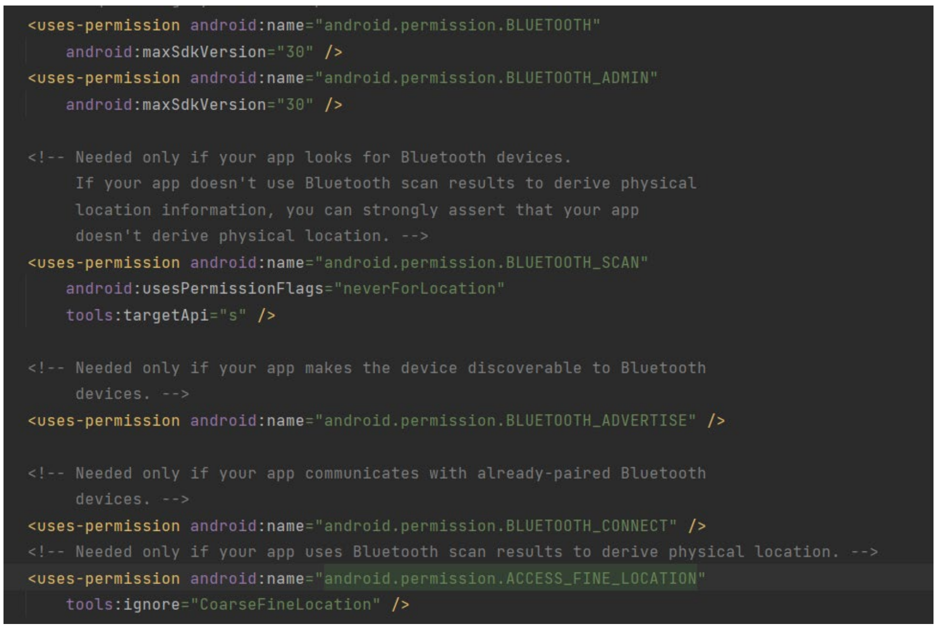
Image of how Blelist when integrated into Quiz test app looks like during testing phase.



Name: Adafruit Bluefruit LE MAC Address: DA:B9:6C:D0:DC:B2

To establish a Bluetooth connection, Blelist's android manifest require Bluetooth permissions to access Bluetooth API, which allows for a Bluetooth feature in app by using its imports.

<u>Snippet of Bluetooth Permissions in the Android manifest</u>



```
<uses-permission android:name="android.permission.BLUETOOTH"
    android:maxSdkVersion="30" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"
    android:maxSdkVersion="30" />

<!-- Needed only if your app looks for Bluetooth devices.
     If your app doesn't use Bluetooth scan results to derive physical
     location information, you can strongly assert that your app
     doesn't derive physical location. -->
<uses-permission android:name="android.permission.BLUETOOTH_SCAN"
    android:usesPermissionFlags="neverForLocation"
    tools:targetApi="s" />

<!-- Needed only if your app makes the device discoverable to Bluetooth
     devices. -->
<uses-permission android:name="android.permission.BLUETOOTH_ADVERTISE" />

<!-- Needed only if your app communicates with already-paired Bluetooth
     devices. -->
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
<!-- Needed only if your app uses Bluetooth scan results to derive physical location. -->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
    tools:ignore="CoarseFineLocation" />
```

More details on the Bluetooth Permissions:

BLUETOOTH_SCAN permission allows the app to look for Bluetooth devices, such as BLE peripherals.

BLUETOOTH_ADVERTISE permission allows the app's device to be discoverable by other devices with Bluetooth.

BLUETOOTH_CONNECT permission allow the app to communicate with already-paired Bluetooth devices.

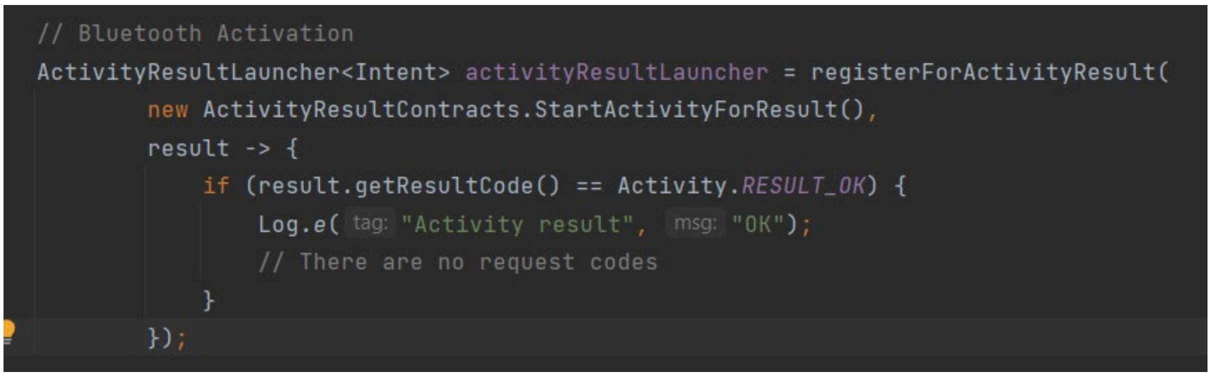
ACCESS_FINE_LOCATION permission uses Bluetooth scan results to derive physical location. It is possible to assert that your app does not derive physical location, without asserting, it will be required in the manifest. However, it's not a necessity for this project, hence it was added in the manifest due to the permission requirements.

BLUETOOTH ADVERTISE, BLUETOOTH CONNECT, and BLUETOOTH SCAN are all runtime permissions. As a result, before you may search for Bluetooth devices, make a device discoverable to other devices, or communicate with Bluetooth devices that are already paired, you must expressly ask the user's permission in your app, by having the device system to prompt the user.

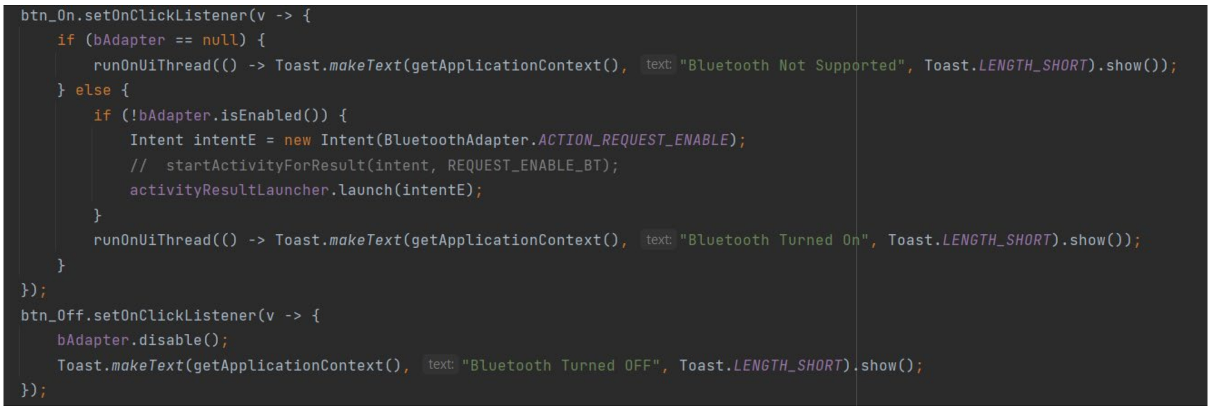
After permission implementation, through the research on how to setup Bluetooth feature in a device. A BluetoothAdapter is required for all Bluetooth activity. The BluetoothAdapter represents the device's own Bluetooth adapter (the Bluetooth radio). BluetoothAdapter can be used as a Boolean, if its null it means device is not Bluetooth supported. To obtain a BluetoothAdapter, using getDefaultAdapter( ) or with the use BluetoothManager's getAdapter( ).

To be able to enable a device's Bluetooth radio, Call isEnabled( ) to check whether Bluetooth is currently enabled. If this method returns false, then Bluetooth is disabled. To request that Bluetooth be enabled, call ActivityResultLauncher a call function that start an action. Using ACTION_REQUEST_ENABLE intent action. This call issues a request to enable Bluetooth through the system settings (without stopping your app). To check if Bluetooth is enabled use isEnabled( ) function and to switch off Bluetooth use BluetoothAdapter.disable( ).

Images of codes to activate Bluetooth in a device.



```
// Bluetooth Activation
ActivityResultLauncher<Intent> activityResultLauncher = registerForActivityResult(
        new ActivityResultContracts.StartActivityForResult(),
        result -> {
            if (result.getResultCode() == Activity.RESULT_OK) {
                Log.e( tag: "Activity result",  msg: "OK");
                // There are no request codes
            }
        });
```

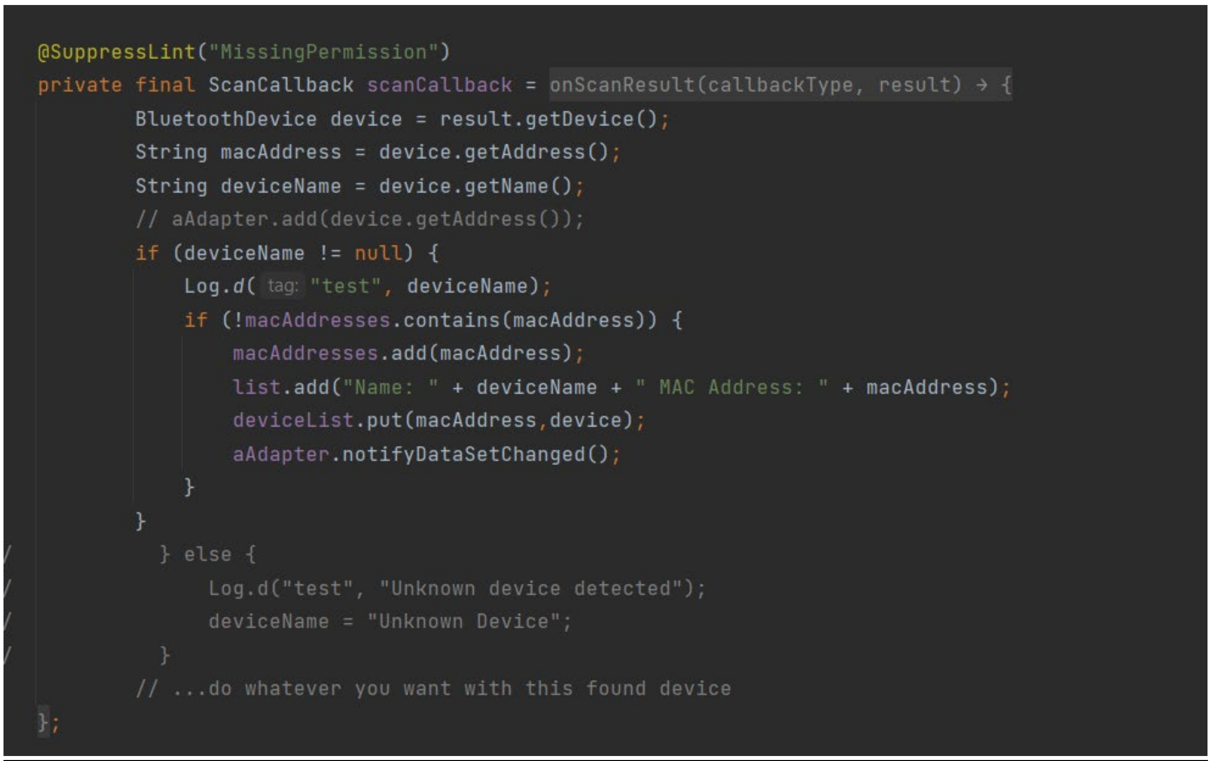


```
btn_On.setOnClickListener(v -> {
    if (bAdapter == null) {
        runOnUiThread(() -> Toast.makeText(getApplicationContext(),  text: "Bluetooth Not Supported", Toast.LENGTH_SHORT).show());
    } else {
        if (!bAdapter.isEnabled()) {
            Intent intentE = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            //  startActivityForResult(intent, REQUEST_ENABLE_BT);
            activityResultLauncher.launch(intentE);
        }
        runOnUiThread(() -> Toast.makeText(getApplicationContext(),  text: "Bluetooth Turned On", Toast.LENGTH_SHORT).show());
    }
});
btn_Off.setOnClickListener(v -> {
    bAdapter.disable();
    Toast.makeText(getApplicationContext(),  text: "Bluetooth Turned OFF", Toast.LENGTH_SHORT).show();
});
```

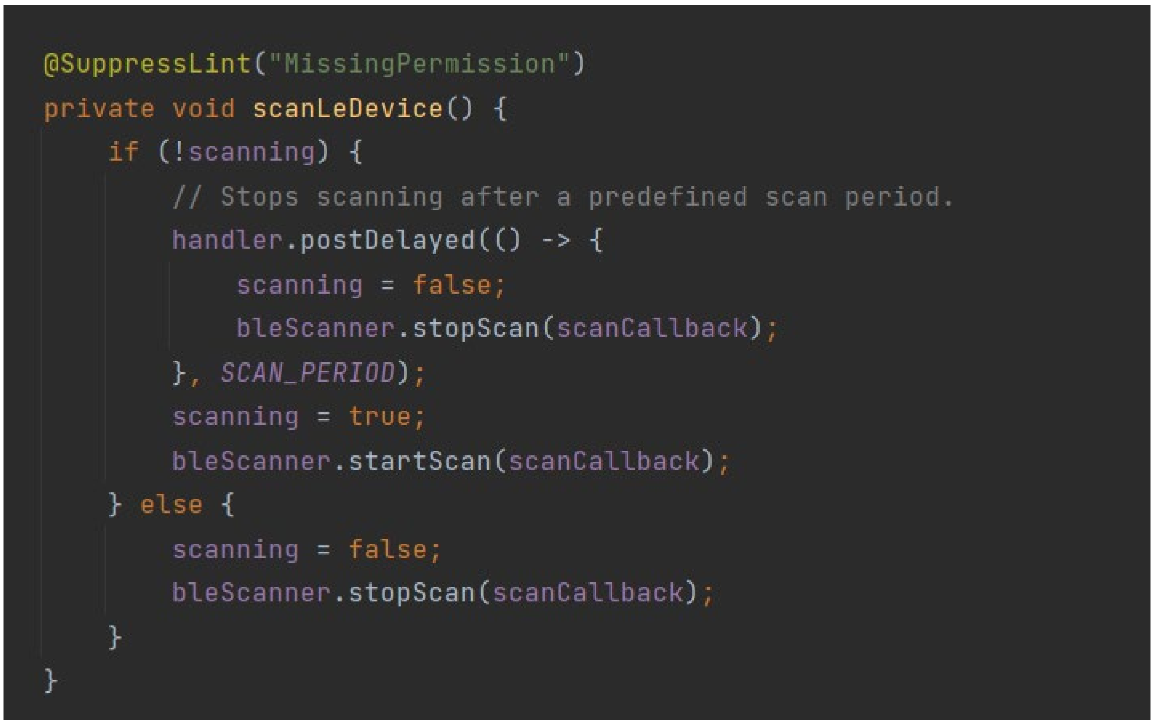
After enabling Bluetooth, the device is capable of using Classic Bluetooth and Low Energy Bluetooth, From reference of Andriod developer website both Bluetooth have their own set of codes to use when implementing their usages. Since Bluetooth Low Energy is used, references for Bluetooth Low Energy (BLE) is used only.

To scan BLE devices, it requires a function to check status of scan and a callback function using a abstract class called ScanCallback to implemet a scanner callback method, with the use of a handler to allow the scanner to scan for a period of time. During the scanning process the information obtain through the Bluetooth radio will be send to the callback method and the information can be stored in variables and be used in the code by using functions like getDevice( ), getAddress( ), getName( ) and more, and be shown in the app. In this project stated earlier, a listview in the test app is used to display information received from the callback method. The scanner also check if there are duplicates of devices by checking if there are duplicate device addresses with the use of a array making sure only there is no duplicate devices in the the listview. After that it will using notifyDataSetChanged( ) to update the listview throught its array adapter.

Images of functions used to scan a BLE device and filtering duplicates



```java
@SuppressLint("MissingPermission")
private final ScanCallback scanCallback = onScanResult(callbackType, result) → {
        BluetoothDevice device = result.getDevice();
        String macAddress = device.getAddress();
        String deviceName = device.getName();
        // aAdapter.add(device.getAddress());
        if (deviceName != null) {
            Log.d( tag: "test", deviceName);
            if (!macAddresses.contains(macAddress)) {
                macAddresses.add(macAddress);
                list.add("Name: " + deviceName + " MAC Address: " + macAddress);
                deviceList.put(macAddress,device);
                aAdapter.notifyDataSetChanged();
            }
        }
          } else {
              Log.d("test", "Unknown device detected");
              deviceName = "Unknown Device";
          }
        // ...do whatever you want with this found device
};
```



```java
@SuppressLint("MissingPermission")
private void scanLeDevice() {
    if (!scanning) {
        // Stops scanning after a predefined scan period.
        handler.postDelayed(() -> {
            scanning = false;
            bleScanner.stopScan(scanCallback);
        }, SCAN_PERIOD);
        scanning = true;
        bleScanner.startScan(scanCallback);
    } else {
        scanning = false;
        bleScanner.stopScan(scanCallback);
    }
}
```

From the investiagtions of Bluetooth data communication using some references like Adafruit App source code, Andriod Developers website and guide from Medium website. For Bluetooth Low Energy it uses a GATT for communication.

GATT is an acronym for the Generic ATTribute Profile, and it defines the way that two Bluetooth Low Energy devices transfer data back and forth using concepts called Services and Characteristics.

To connect to a device a class BluetoothDevice is required to create a object to represent a remote Bluetooth device. A BluetoothDevice lets you create a connection with the respective device or query information about it, such as the name, address, class, and bonding state. In Blelist app, the BluetoothDevice uses the address obtained from the Listview data which is obtain through the ScanCallback callback scanner method.

Using a BluetoothGatt method call connectGatt( ) to connect the Bluefruit LE Friend UART by using the BluetoothDevice. In the test app a gobal variable "dgatt" using BluetoothGatt import is created to maintain the connection of the Bluefruit LE Friend UART Bluetooth profile, to be able to communicate with it.

In order to use connectGatt( ), a abstract class called BluetoothGattCallback is needed. It creates a object that contains BluetoothGatt methods which monitors any Bluetooth related actions running in the app. BluetoothGatt is the public API for the Bluetooth GATT Profile, one of Andriod Studio's imports. The gatt in the method are independent local variables they do not affect the gobal variable "dgatt" created to store the connected device's Bluetooth profile.

<u>Explainations of used BluetoothGatt methods:</u>

public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) is used to check for any connection changed between two Bluetooth Low energy devices. The integer newState can be BluetoothProfile.STATE_CONNECTED or BluetoothProfile.STATE_DISCONNECTED, which is the connection status of the device. Its found that after a device is connected the method discoverServices( ) should be used immediately as the Gatt server have been established which the connection between devices.

A Service is a container for logically related Bluetooth data items. Those data items are in fact called Characteristics. A Service can be thought of as the owner of the Characteristics inside it

public void onServicesDiscovered(BluetoothGatt gatt, int status) is used when method discoverServices( ) is called and services are discovered. This method is used for obtaining a specific service using a specific UUID as a device can have many services. After getting the specific service, its characteristics are obtainable using a specific UUID.

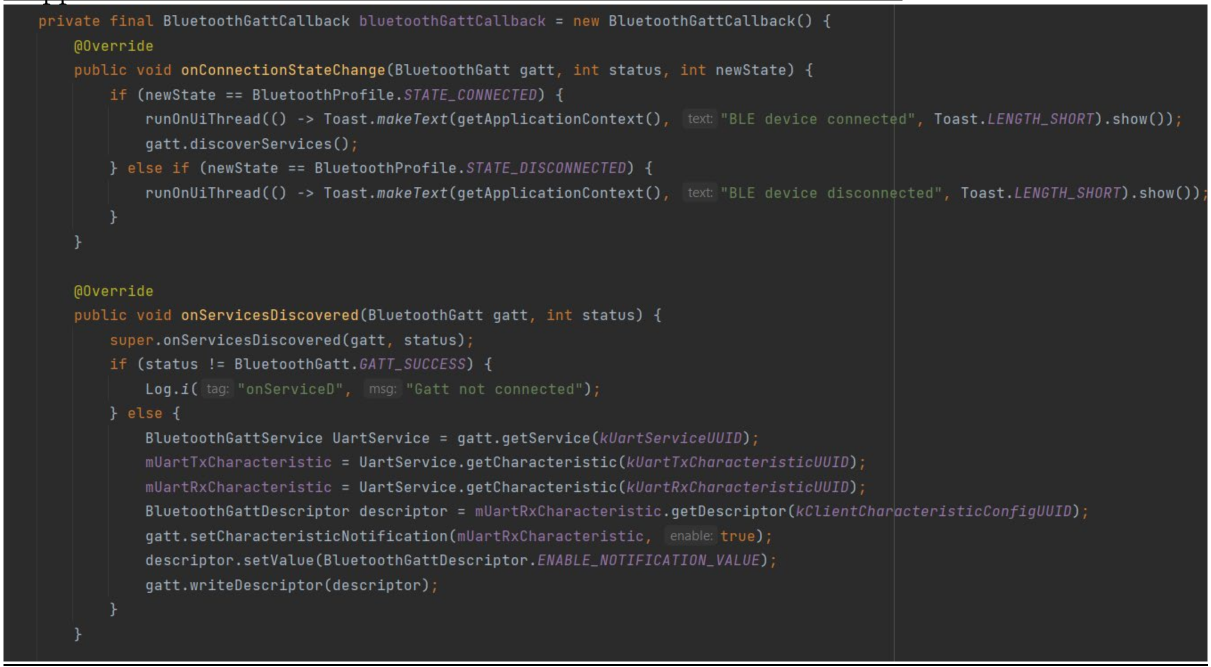
A universally unique identifier (UUID) is a 128-bit (16 bytes) number that is guaranteed to be globally unique. UUIDs are used in many protocols and applications other than Bluetooth, and their format, usage, and generation is specified in ITU-T Rec. X.667, alternatively known as ISO/IEC 9834-8:2005.

In Blelist app they are use for Bluetooth, mainly for data transfer. Once the characteristics have been obtained it is possible for the app to write data to a another device or read data from it. The UUIDs used in this project are:

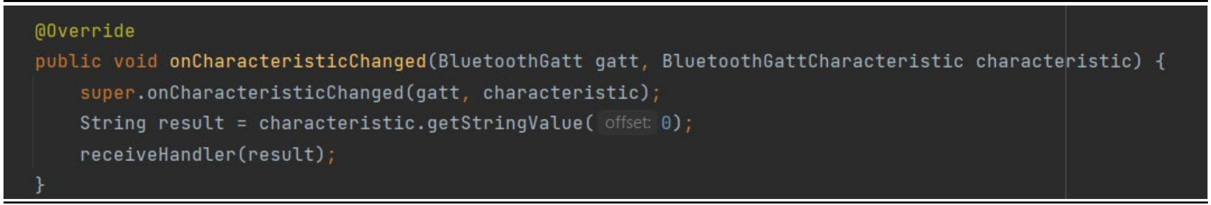
1. 6e400001-b5a3-f393-e0a9-e50e24dcca9e for UART Service (Base UUID)
2. 6e400002-b5a3-f393-e0a9-e50e24dcca9e for writing characteristics using UART Service
3. 6e400003-b5a3-f393-e0a9-e50e24dcca9e for reading characteristics using UART Service
4. 00002902-0000-1000-8000-00805f9b34fb for ClientCharacteristicConfig (To enable notifications or indications for reading data)

public void onCharacteristicChanged(BluetoothGatt gatt, BluetoothGattCharacteristic characteristic) is used when data is received in the app.

<u>Snippet of used BluetoothGattCallback and the BluetoothGatt methods:</u>



```java
private final BluetoothGattCallback bluetoothGattCallback = new BluetoothGattCallback() {
    @Override
    public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) {
        if (newState == BluetoothProfile.STATE_CONNECTED) {
            runOnUiThread(() -> Toast.makeText(getApplicationContext(),  text: "BLE device connected", Toast.LENGTH_SHORT).show());
            gatt.discoverServices();
        } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
            runOnUiThread(() -> Toast.makeText(getApplicationContext(),  text: "BLE device disconnected", Toast.LENGTH_SHORT).show());
        }
    }

    @Override
    public void onServicesDiscovered(BluetoothGatt gatt, int status) {
        super.onServicesDiscovered(gatt, status);
        if (status != BluetoothGatt.GATT_SUCCESS) {
            Log.i( tag: "onServiceD",  msg: "Gatt not connected");
        } else {
            BluetoothGattService UartService = gatt.getService(kUartServiceUUID);
            mUartTxCharacteristic = UartService.getCharacteristic(kUartTxCharacteristicUUID);
            mUartRxCharacteristic = UartService.getCharacteristic(kUartRxCharacteristicUUID);
            BluetoothGattDescriptor descriptor = mUartRxCharacteristic.getDescriptor(kClientCharacteristicConfigUUID);
            gatt.setCharacteristicNotification(mUartRxCharacteristic,  enable: true);
            descriptor.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
            gatt.writeDescriptor(descriptor);
        }
    }
```

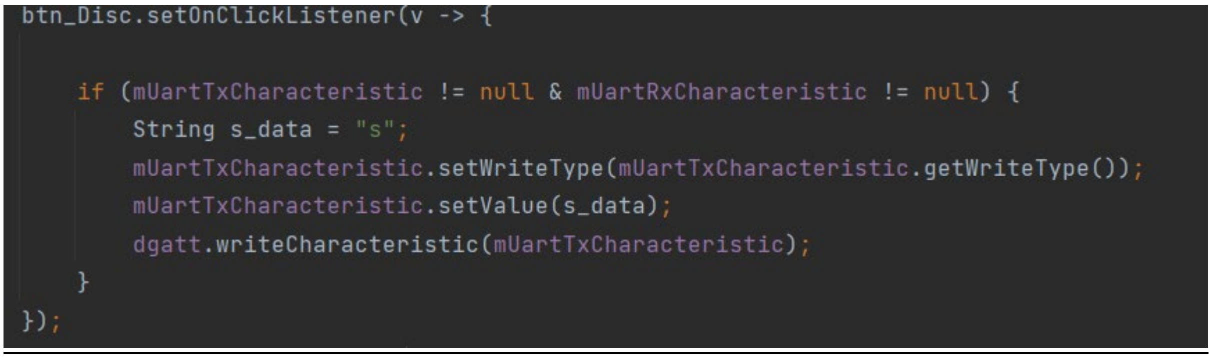


```java
@Override
public void onCharacteristicChanged(BluetoothGatt gatt, BluetoothGattCharacteristic characteristic) {
    super.onCharacteristicChanged(gatt, characteristic);
    String result = characteristic.getStringValue( offset: 0);
    receiveHandler(result);
}
```

The implementation of writing data to Arduino to dispense a chocolate, it require a few methods in order which are setWriteType( ), setValue( ) and writeCharacteristic( ). Function setWriteType( ) set the write type for the charactristic, setValue( ) is used to set value of the data to be send in the characteristic, the data is a string value of "s" in the app.

Lastly, writeCharacteristic( ) is to write the characteristic to the Bluefruit LE Friend UART for it to read the characteristic and dispense a chocolate.

<u>Image of writing characteristic code with a button to trigger the code</u>



```java
btn_Disc.setOnClickListener(v -> {

    if (mUartTxCharacteristic != null & mUartRxCharacteristic != null) {
        String s_data = "s";
        mUartTxCharacteristic.setWriteType(mUartTxCharacteristic.getWriteType());
        mUartTxCharacteristic.setValue(s_data);
        dgatt.writeCharacteristic(mUartTxCharacteristic);
    }
});
```

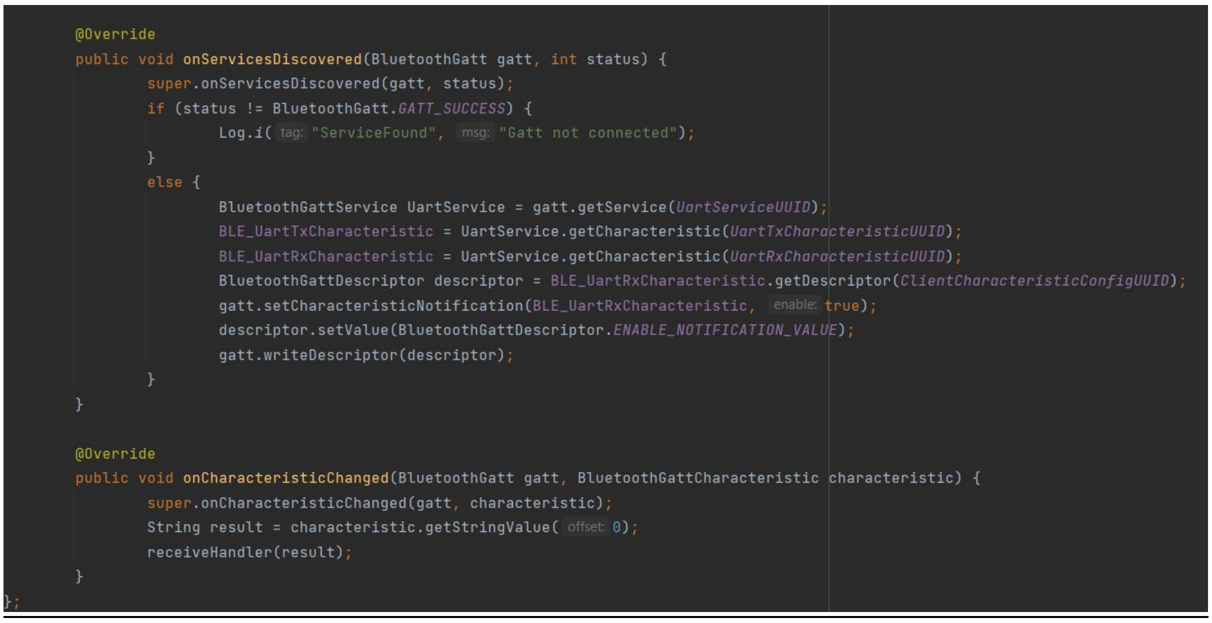
While for the implementation of reading data from Arduino, requires a public class BluetoothGattDescriptor to represents a Bluetooth GATT Descriptor. GATT Descriptors contain additional information and attributes of a GATT characteristic, which in this test app is using the UUID for reading characteristic, the descriptor is able to receive data coming from Arduino.

The BluetoothGattDescriptor variable requires ClientCharacteristicConfig UUID by using getDescriptor( ) to enable notifications for read characteristics by using setCharacteristicNotification( ). The descriptor variable will be set with a value using setvalue( ) to enable notification for descriptor. The variable will be sent Arduino, which will trigger Arduino to send back a notification which is the characteristic that the app is suppose to read.

The BluetoothGattDescriptor code is written BluetoothGatt method onServiceDiscovered( ) to make sure Arduino will be ready to write characteristic to the app when Bluefruit LE Friend UART is connected.

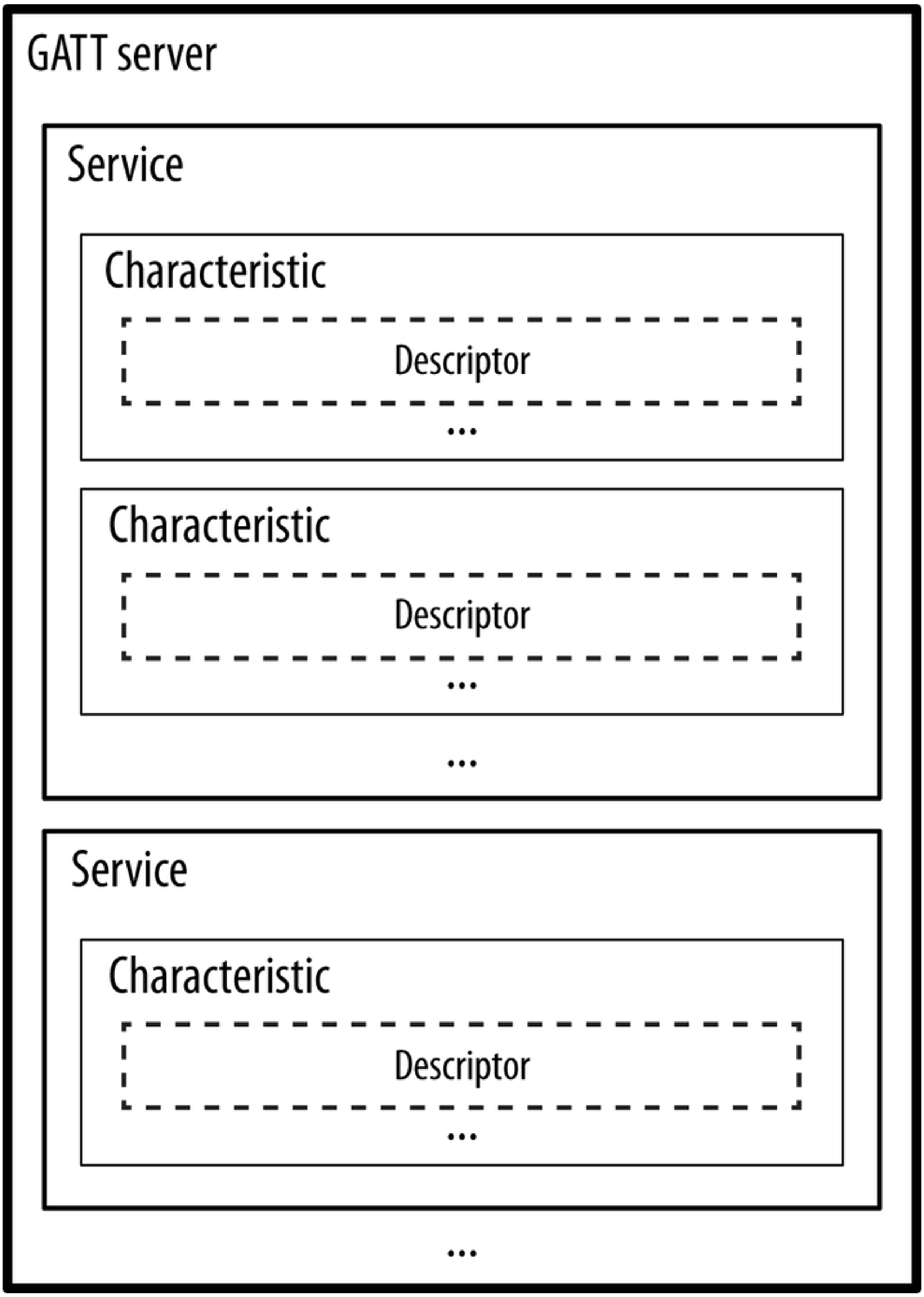
The reading of characteristics is triggered by the BluetoothGatt method onCharacteristicChanged( ) and it will excute a function to read the characteristic using getStringValue( ), and sent to a string builder to create string variable for the characteristic to be used in the app.

<u>Image of reading characteristic code with a button to trigger the code</u>



```
        @Override
        public void onServicesDiscovered(BluetoothGatt gatt, int status) {
                super.onServicesDiscovered(gatt, status);
                if (status != BluetoothGatt.GATT_SUCCESS) {
                        Log.i( tag: "ServiceFound",  msg: "Gatt not connected");
                }
                else {
                        BluetoothGattService UartService = gatt.getService(UartServiceUUID);
                        BLE_UartTxCharacteristic = UartService.getCharacteristic(UartTxCharacteristicUUID);
                        BLE_UartRxCharacteristic = UartService.getCharacteristic(UartRxCharacteristicUUID);
                        BluetoothGattDescriptor descriptor = BLE_UartRxCharacteristic.getDescriptor(ClientCharacteristicConfigUUID);
                        gatt.setCharacteristicNotification(BLE_UartRxCharacteristic,  enable: true);
                        descriptor.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
                        gatt.writeDescriptor(descriptor);
                }
        }

        @Override
        public void onCharacteristicChanged(BluetoothGatt gatt, BluetoothGattCharacteristic characteristic) {
                super.onCharacteristicChanged(gatt, characteristic);
                String result = characteristic.getStringValue( offset: 0);
                receiveHandler(result);
        }
};
```

Image of how a Gatt Server looks like

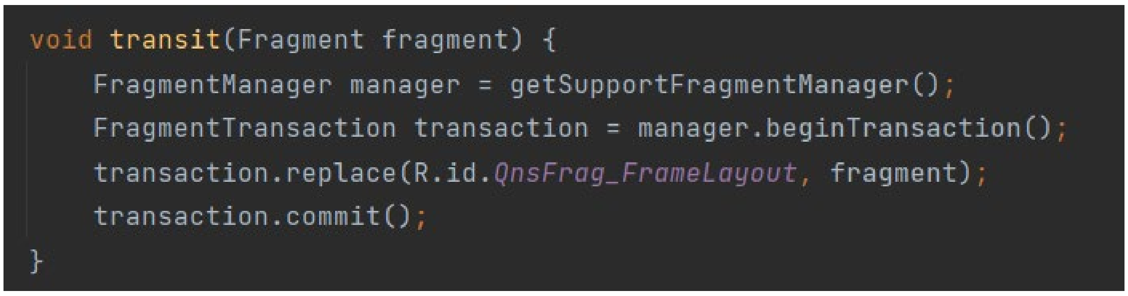


After the testing for implementation of Andriod app and Bluetooth communications, a final build app is created named TemiChoco, to meet the objectives of creating a questionnaire app with Bluetooth (BLE) capabilities for communication between Temi 2 and Arduino UNO, using Bluefruit LE Friend UART, and to make a neat presentation of code.

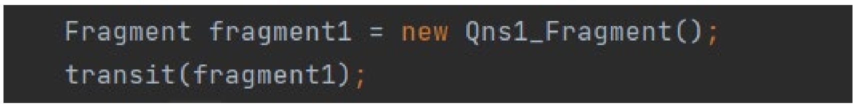
During the development of TemiChoco, the questions activities were replaced with Fragments, which is essentially a subactivity of a activity. Due fragment's reusability and the benefit of removing the need to replace a activity, reducing the computing power of the app and improvement the aesthetics of the app as fragment have better transition. Fragments are not required in Andriod Manifest as its part of a activity, fragments also provide a more flexible management of the project(app) structure. It is also possible to have multiple fragments in a activity creating a multi-screen UI for future improvements. Lastly, Google recommended developers to use fragments.

The difference in coding while Activity only needs a intent to use the function startActivity( ) to move to another activity, fragments require a FragmentManager and Fragment Transaction to navigate to another fragment.

<u>Image of the code for Fragment navigation</u>



```
void transit(Fragment fragment) {
    FragmentManager manager = getSupportFragmentManager();
    FragmentTransaction transaction = manager.beginTransaction();
    transaction.replace(R.id.QnsFrag_FrameLayout, fragment);
    transaction.commit();
}
```



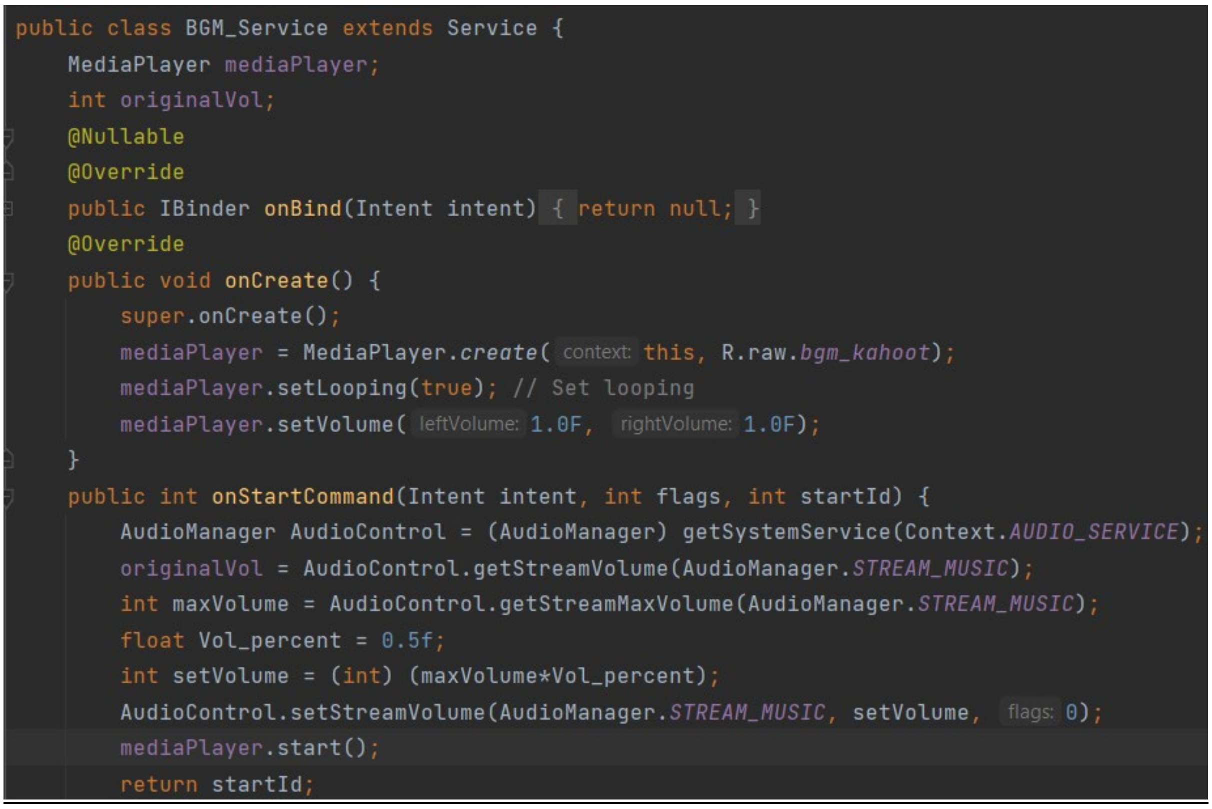
```
Fragment fragment1 = new Qns1_Fragment();
transit(fragment1);
```

AlertDialog one of Android Studio's Dialog box to prompt user of a notification, is used to replace most of the toast messages used to display information to users. Information includes the result of their answers when answering a question, device connected reminder, Bluetooth connection notifications or Bluetooth inactive notifications and the reason of error when a user entered the wrong answer, to catch their attention and to help to learn more about NYP and robotics. This is because after research on the usage of toast message in Android apps its found that it cannot appear in the middle of the text for devices that has Android 11 and above, Temi robot is equipped with Android 11. Toast message are only used for debug messages which are Bluetooth connection status and chocolate dispensing status.

To strengthen new student's interest, kahoot background music was added into the TemiChoco to be played when the main menu is showed, with a toggle button if the music get excessive. The reason for Kahoot music is because of nostalgia because Kahoot is usually used for quizzes in school, which could attract students more effectively.

During development of the music player, service abstract class was used to able to run the music in the background, and Android Studio's import MediaPlayer is used to play music from the resource file under the raw folder which contains a downloaded kahoot mp4 file. Import Audio Manager was used to set the volume of the music when being played and will set the device volume to its original when the music stops. When the app is closed or not in the main menu, the music is stopped.

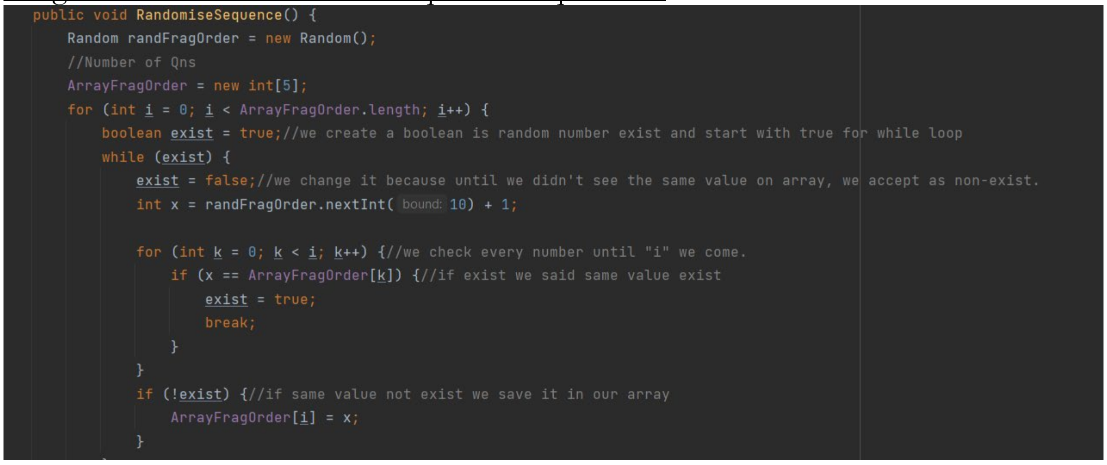
Snippet of music player code



```
public class BGM_Service extends Service {
    MediaPlayer mediaPlayer;
    int originalVol;
    @Nullable
    @Override
    public IBinder onBind(Intent intent) { return null; }
    @Override
    public void onCreate() {
        super.onCreate();
        mediaPlayer = MediaPlayer.create( context: this, R.raw.bgm_kahoot);
        mediaPlayer.setLooping(true); // Set looping
        mediaPlayer.setVolume( leftVolume: 1.0F,  rightVolume: 1.0F);
    }
    public int onStartCommand(Intent intent, int flags, int startId) {
        AudioManager AudioControl = (AudioManager) getSystemService(Context.AUDIO_SERVICE);
        originalVol = AudioControl.getStreamVolume(AudioManager.STREAM_MUSIC);
        int maxVolume = AudioControl.getStreamMaxVolume(AudioManager.STREAM_MUSIC);
        float Vol_percent = 0.5f;
        int setVolume = (int) (maxVolume*Vol_percent);
        AudioControl.setStreamVolume(AudioManager.STREAM_MUSIC, setVolume,  flags: 0);
        mediaPlayer.start();
        return startId;
```

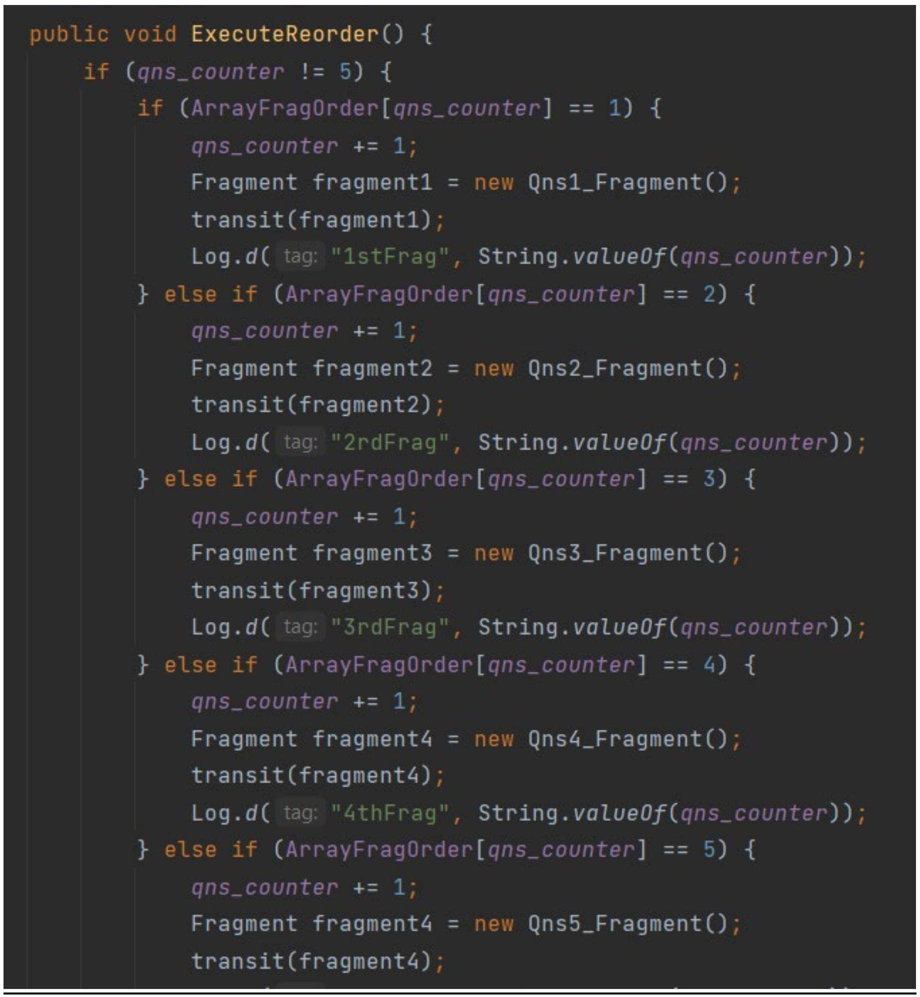
To ensure every user who uses the app have a unique set of questions, the number of questions increased to ten, using the three types of questions, six MCQs, three PMCQs and three short, structured questions. A randomiser was added to randomly pick out five out of ten questions for a user to answer in sequence. A question counter is added to track the user progress and show the user the question according to the randomised sequence of questions. Every quiz attempt will be randomised and the counter will reset at a new attempt.

Image of Randomiser code for sequence of questions



```java
public void RandomiseSequence() {
    Random randFragOrder = new Random();
    //Number of Qns
    ArrayFragOrder = new int[5];
    for (int i = 0; i < ArrayFragOrder.length; i++) {
        boolean exist = true;//we create a boolean is random number exist and start with true for while loop
        while (exist) {
            exist = false;//we change it because until we didn't see the same value on array, we accept as non-exist.
            int x = randFragOrder.nextInt( bound: 10) + 1;

            for (int k = 0; k < i; k++) {//we check every number until "i" we come.
                if (x == ArrayFragOrder[k]) {//if exist we said same value exist
                    exist = true;
                    break;
                }
            }
            if (!exist) {//if same value not exist we save it in our array
                ArrayFragOrder[i] = x;
            }
```

Image of user question tracker and function to navigate to question.



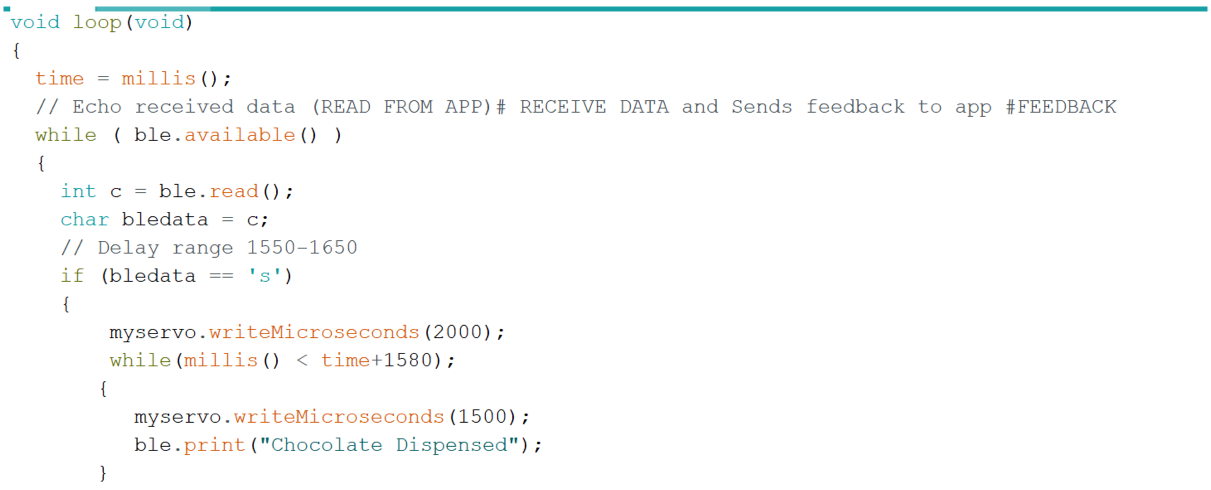
```java
public void ExecuteReorder() {
    if (qns_counter != 5) {
        if (ArrayFragOrder[qns_counter] == 1) {
            qns_counter += 1;
            Fragment fragment1 = new Qns1_Fragment();
            transit(fragment1);
            Log.d( tag: "1stFrag", String.valueOf(qns_counter));
        } else if (ArrayFragOrder[qns_counter] == 2) {
            qns_counter += 1;
            Fragment fragment2 = new Qns2_Fragment();
            transit(fragment2);
            Log.d( tag: "2rdFrag", String.valueOf(qns_counter));
        } else if (ArrayFragOrder[qns_counter] == 3) {
            qns_counter += 1;
            Fragment fragment3 = new Qns3_Fragment();
            transit(fragment3);
            Log.d( tag: "3rdFrag", String.valueOf(qns_counter));
        } else if (ArrayFragOrder[qns_counter] == 4) {
            qns_counter += 1;
            Fragment fragment4 = new Qns4_Fragment();
            transit(fragment4);
            Log.d( tag: "4thFrag", String.valueOf(qns_counter));
        } else if (ArrayFragOrder[qns_counter] == 5) {
            qns_counter += 1;
            Fragment fragment4 = new Qns5_Fragment();
            transit(fragment4);
```

A user counter for track number of users who played the app was also added to track how many users played for admin purposes if needed.

Lastly, the result page improved drastically because the test app's design was bland. It only has a button to restart the quiz and it automatically dispense a chocolate when the activity opens with a Textview showing the user results. In TemiChoco, it has a Editview to obtain the name of the user and Textview to display the users results with his name and instructions to end the quiz. AlertDialog was used to notify if a chocolate has been dispensed and a Textview for the title page was added. A checker was added to check how many tries did the user take to answer all the questions, if the user took less than seven tries, two chocolates would be dispensed to reward for the user's achievement, else everyone would be rewarded with one chocolate. Lastly, two buttons were used to either end the quiz which would reset the whole app, close Bluetooth, reset user counter and be on standby mode in main menu or restart app with the connection still active. A chocolate counter is also added to track the number of chocolates dispensed, if there were no chocolate left, a dialog box would prompt for a refill of chocolate and reset the counter.

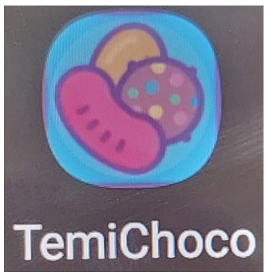
During the implementation of Arduino code, using a given code from Arduino for Adafruit Bluefruit LE Friend UART (BLE), the codes that involved serial were all removed to enable automatic running of Arduino for Bluetooth, and code for data communication were changed to dispense chocolate when receiving read characteristic from the app and sending write characteristics to the app after dispensing and added servo code into Arduino to control servo movement. Most of the code is handed by a configuration file included with given code.

Arduino code to dispense chocolate.



```
void loop(void)
{
  time = millis();
  // Echo received data (READ FROM APP)# RECEIVE DATA and Sends feedback to app #FEEDBACK
  while ( ble.available() )
  {
    int c = ble.read();
    char bledata = c;
    // Delay range 1550-1650
    if (bledata == 's')
    {
        myservo.writeMicroseconds(2000);
        while(millis() < time+1580);
       {
          myservo.writeMicroseconds(1500);
          ble.print("Chocolate Dispensed");
       }
```

After the creations of the TemiChoco, the app icon was changed to a candy theme icon for aesthetics purposes.



Integration and Testing

For the integration and testing of TemiChoco, Blelist and EGR326 interface were evaluated on a tablet which is on Android 13. As it provides easy access, quicker response, and updates to my code, as it only requires a USB cable to connect to my computer to access the codes and developer mode on my tablet to allow testing of the applications.

In the final rounds of testing TemiChoco, it was tested on Temi to achieve the object of having a questionnaire app in Temi and for user to interact with Temi.

During testing for app development, for legacy Bluetooth-related permission declarations, a requirement of setting android: maxSdkVersion to thirty in the Gradle settings. This app compatibility step helps the system grant your app only the Bluetooth permissions that it needs when installed on devices that run Android 12 or higher, any device lower is will not be affected after going through testing on Temi.

Key Technical Issues and Solutions

During the development of Bluetooth testing app, Google released an update to the permissions required for Bluetooth, it causes sample app's used for references to fails and outdated codes no longer works.

A permission checker was added to deal with a permission error that kept appearing during the start of Blelist, its found that its Google that required developers to add a checker to enhance security of a app. The checker is only used on first use of the app after installation. However, whenever a function involves Bluetooth, permissions lint warning is given even if the permission checker is in the app. Using @SuppressLint("MissingPermission") helped solved the issue.

Some widgets used in the app requires Android SDK 31, but this project uses a minimum SDK 21, which causes errors. Hence, @RequiresApi(api = Build.VERSION_CODES.S) is required.

During the Final testing of TemiChoco in Temi, investigations showed permission checker prompts a user to allow a permission and the user denies. If a user input requires Bluetooth, the app will instantly crash. Research is needed to fix the solution, currently its only known that the checker will use a system prompt, it can't be edit and the code has not been found through research, hence no solution can be provided.

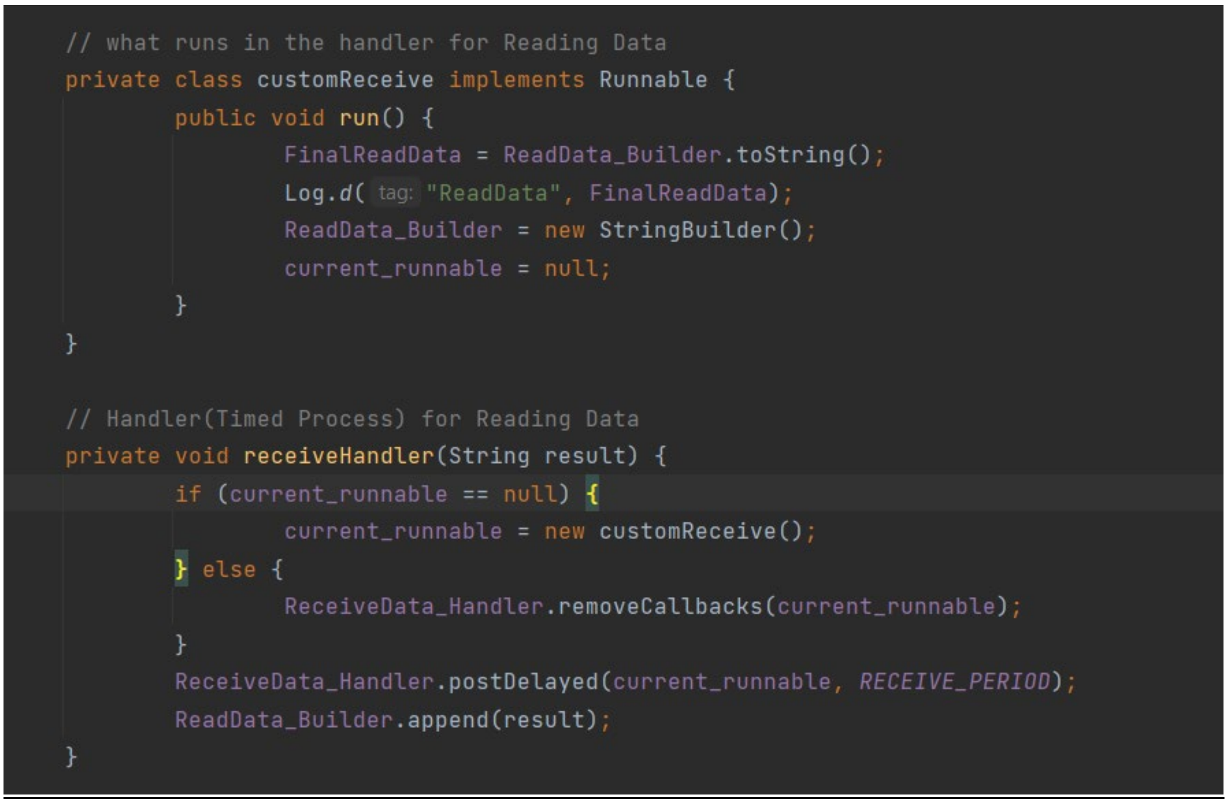
The issues were solved by referring to Android Developer website and stack overflow, which was to update the permissions in the manifest. This project uses the updated permission for Bluetooth, refer to Appendix for website to view android 11 or lower permissions.

Most references to research on app development and Bluetooth implementation were outdated or too complex to understand. There is no solution, only through time of understanding of multiple sample codes, using websites to seek help or uses solved solutions and videos will help.

This project requires strong understanding of Java programming and Android Studio usage, only with time and experience will solve this problem.

During the implementation of reading characteristics, the data transfer was extremely fast as the call-back function was called very frequently, causes the data to be corrupted, so a handler and runnable was created to give time for the characteristic to be send into the app and preventing the functions to keep triggering. The runnable is an action used to show what the data received is in a string variable. Function removeCallbacks( ) is used to removes any duplication of data by removing previous data received from the runnable. The action cannot be written in the handler because in the start of the app there is no characteristic to read which will crash the app, hence a runnable is used.

Image of solution for reading characteristic issue



```
// what runs in the handler for Reading Data
private class customReceive implements Runnable {
        public void run() {
                FinalReadData = ReadData_Builder.toString();
                Log.d( tag: "ReadData", FinalReadData);
                ReadData_Builder = new StringBuilder();
                current_runnable = null;
        }
}

// Handler(Timed Process) for Reading Data
private void receiveHandler(String result) {
        if (current_runnable == null) {
                current_runnable = new customReceive();
        } else {
                ReceiveData_Handler.removeCallbacks(current_runnable);
        }
        ReceiveData_Handler.postDelayed(current_runnable, RECEIVE_PERIOD);
        ReadData_Builder.append(result);
}
```

After the implementation of reading characteristic in the Blelist app, it was crashing. It was found that because writing and reading characteristics, are asynchronous which means if they run both writing and reading characteristics are running at the same time, the app will crash, hence a delay is added to make sure the data transfer is complete first before running the next transfer.

During the testing of the 3D printed dispenser, it is found that the chocolate sitting on the pusher and the two holders of the pusher causes friction to the pusher, with affects the servo revolution when it's dispensing a chocolate which causes a deviation in the revolution. This is because the servo is moving at a fixed timing, it does not have a feedback system to fix the deviation, and the material is causing then friction. Creating a new dispenser model or adding ball bearing to the holder are solutions but they have not been implemented due to time constraint. The time range to reduce the deviation to a minimum is 1550 to 1650 milliseconds.

Previously, the dispenser had loose screws for keeping the structure together. Additionally with wear and tear of the 3D printed material used for the dispenser, it created a structure integrity failure. The screws were tightened with nuts.

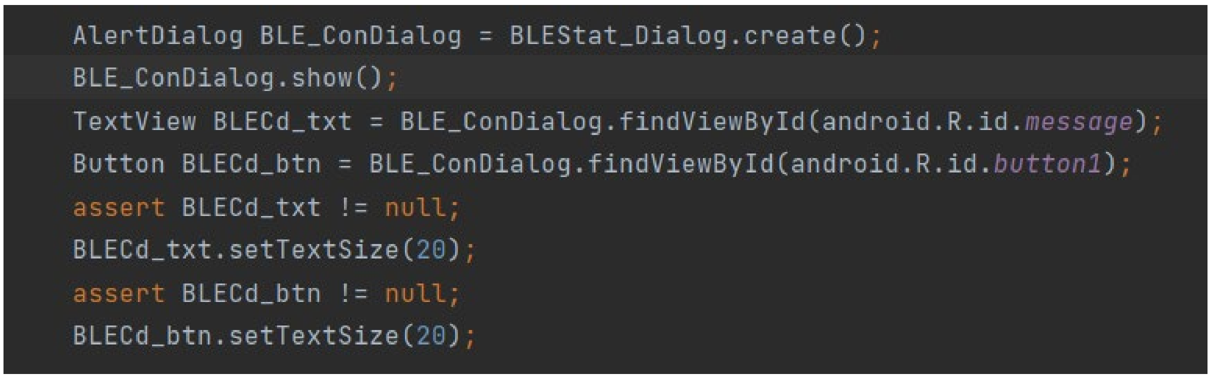
The first chocolate is guaranteed to be dispensed but the slide of the dispenser is not smooth, the chocolate may get stuck in the process of dispensing. Solution is to file and smooth the slider's surface.
Video of Chocolate getting stuck

During testing of apps in Temi, investigations found that special fonts used were not applicable to Temi, because of it did not have the fonts downloaded in its storage. Themes used in development were also not applicable, as it does not have the downloaded theme. This causes the app design to look ugly. Editing the layout design by moving widgets, changing colours, size of text and alignment. Evaluating the edited layout in Temi solved the problem.

ADB connection, the method of connect to download the app into Temi, did not work sometimes. Restarting Android Studio or Temi will fix the issue as the connection restarted.

During the implementation of AlertDialog, it was found that the text is too small to read. Hence, to increase the font size. Access to the AlertDialog widgets is required to change the font size. However, the widgets must be asserted non null because the Dialog uses a builder import and have not been created and shown, meaning the widget does not exist.

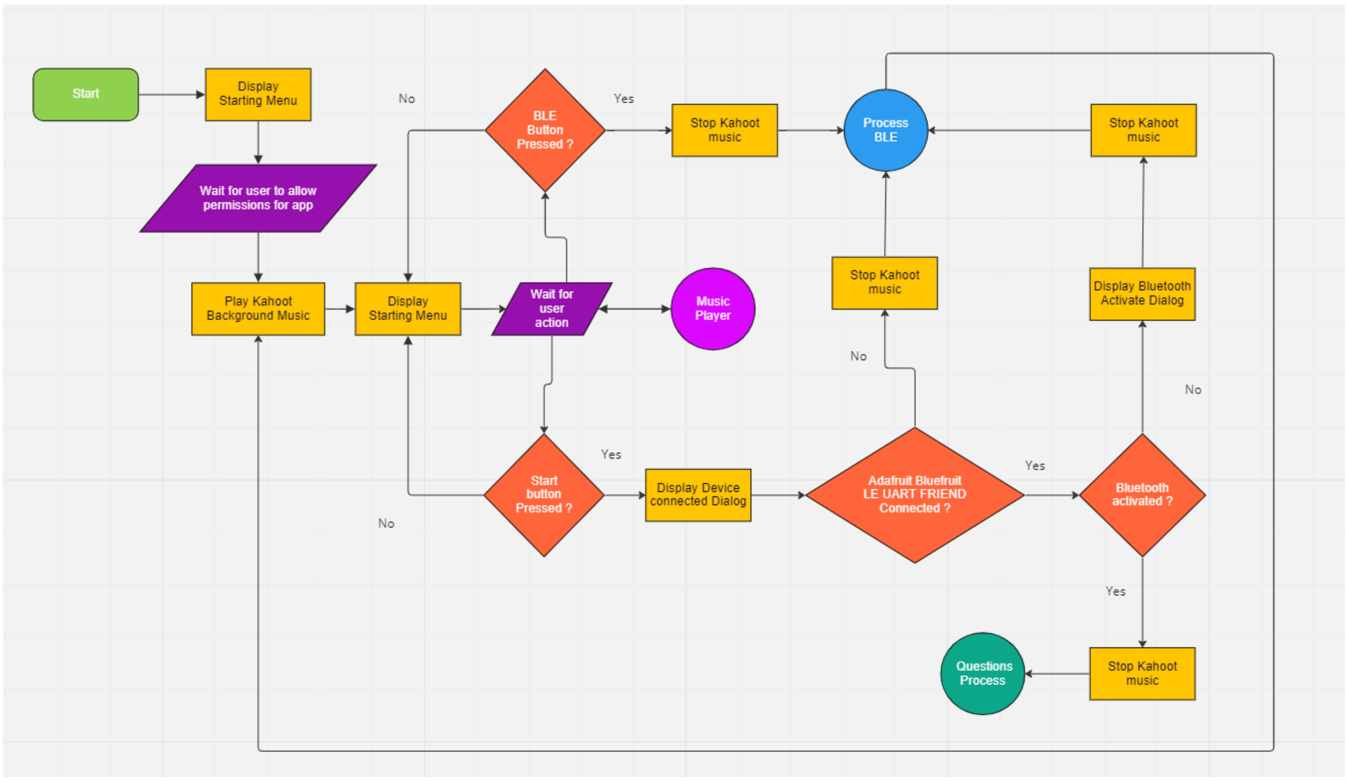
Image for AlertDialog creation and font size adjustments.



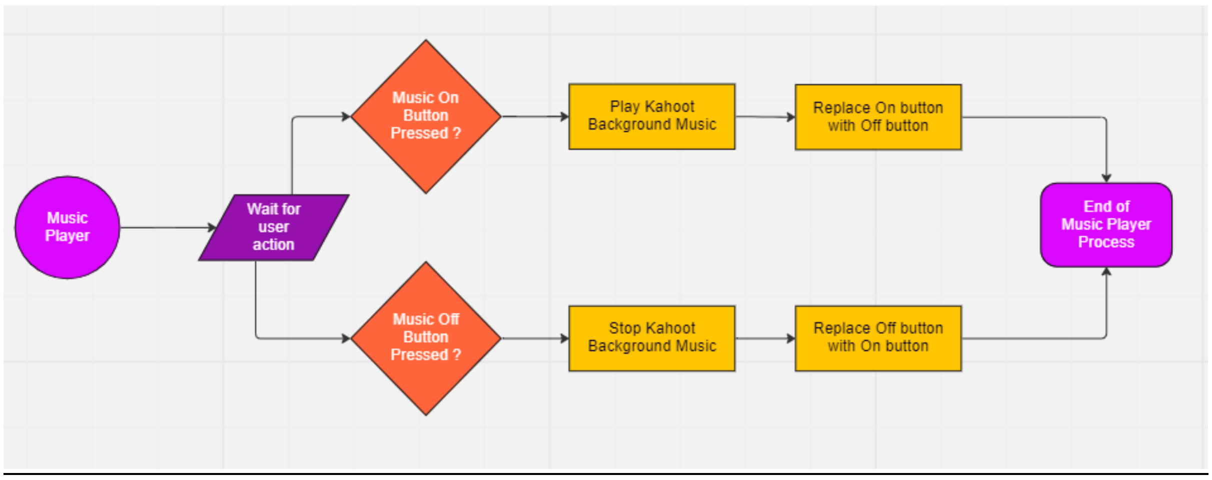
```
AlertDialog BLE_ConDialog = BLEStat_Dialog.create();
BLE_ConDialog.show();
TextView BLECd_txt = BLE_ConDialog.findViewById(android.R.id.message);
Button BLECd_btn = BLE_ConDialog.findViewById(android.R.id.button1);
assert BLECd_txt != null;
BLECd_txt.setTextSize(20);
assert BLECd_btn != null;
BLECd_btn.setTextSize(20);
```

Flowchart of TemiChoco

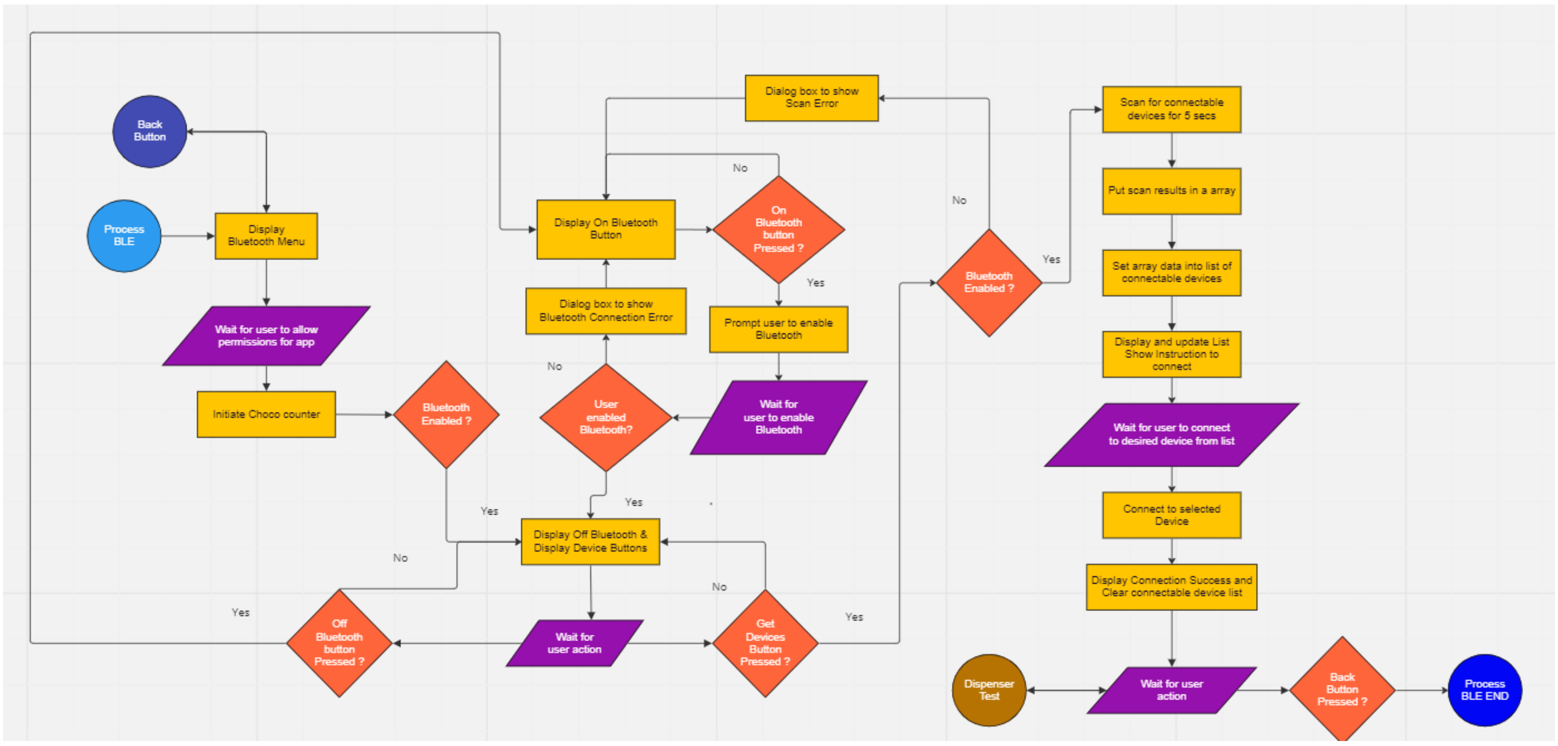
Main Menu Flowchart



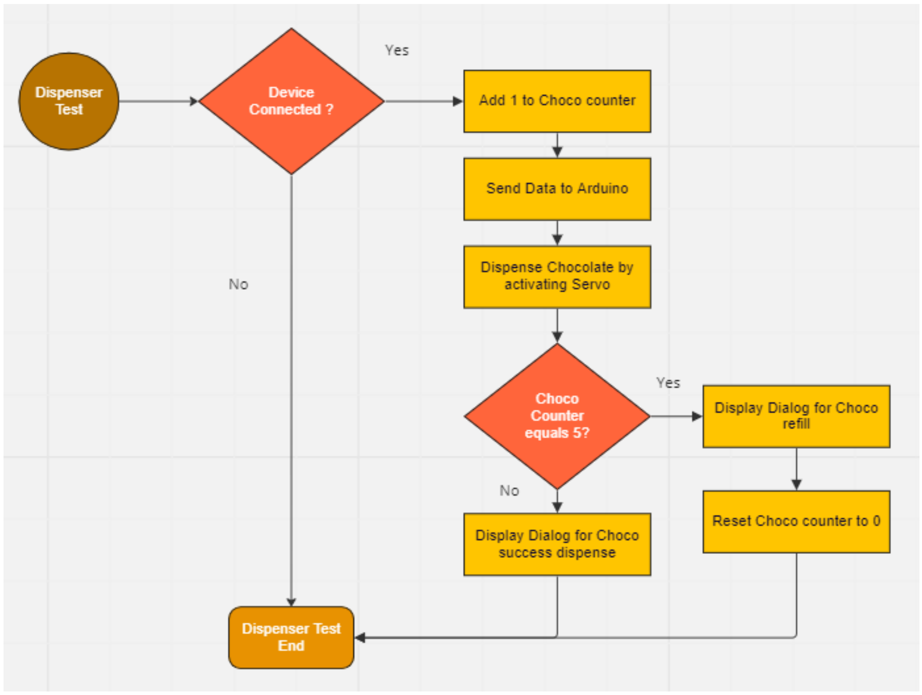
Music Player Flowchart

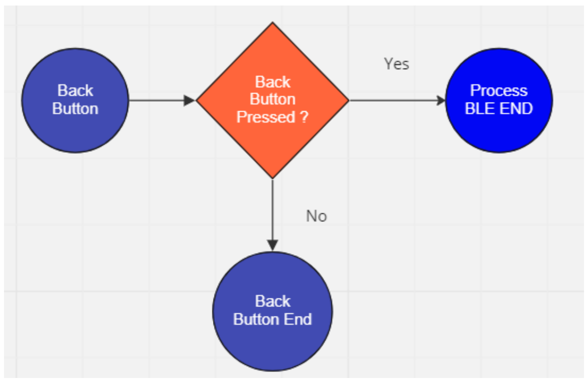

Music Player
Wait for user action
Music On Button Pressed ?
Play Kahoot Background Music
Replace On button with Off button
Music Off Button Pressed ?
Stop Kahoot Background Music
Replace Off button with On button
End of Music Player Process

Bluetooth Menu Flowchart

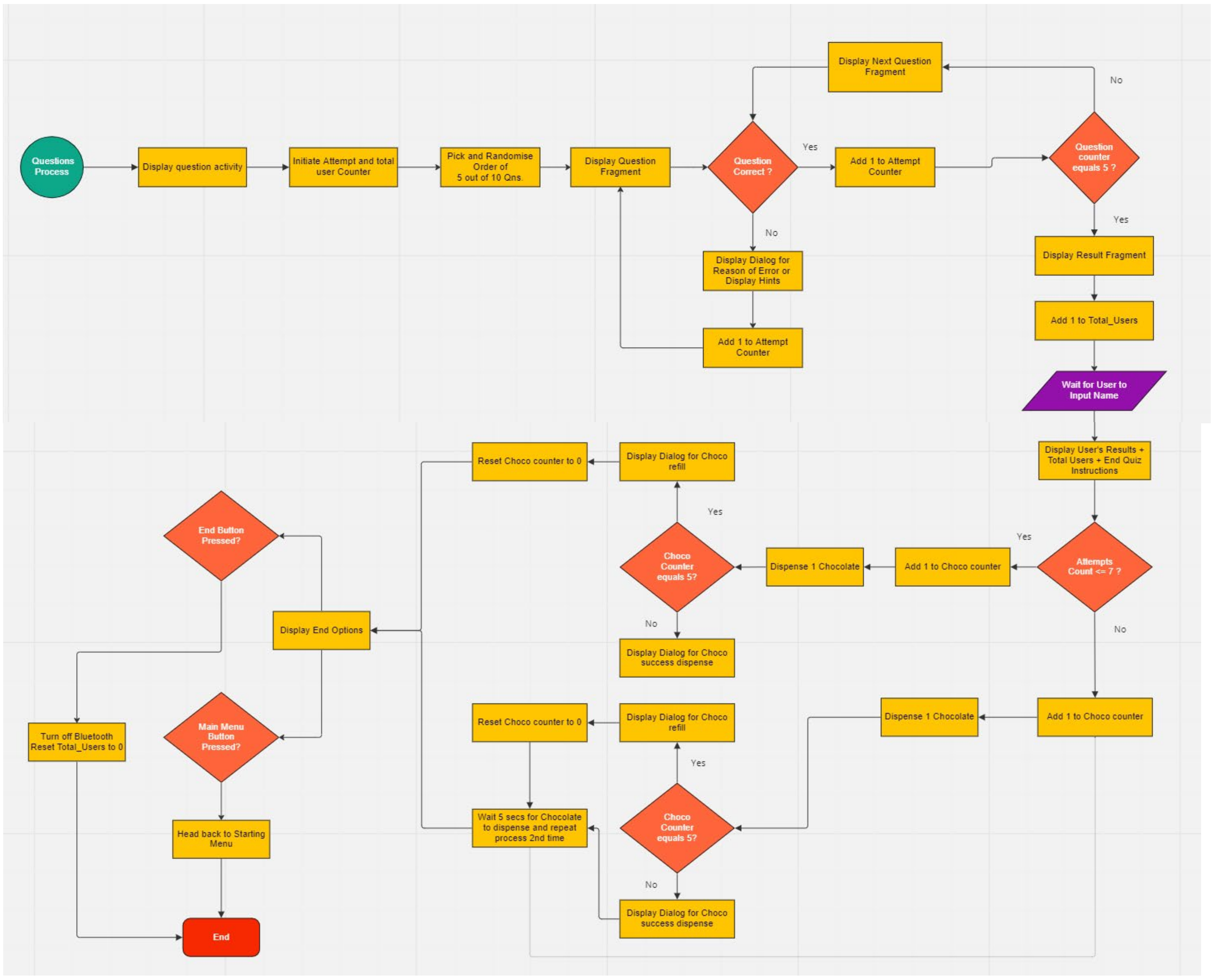

Back Button
Process BLE
Display Bluetooth Menu
Wait for user to allow permissions for app
Initiate Choco counter
Bluetooth Enabled ?
Yes
No
Display On Bluetooth Button
On Bluetooth button Pressed ?
No
Yes
Dialog box to show Bluetooth Connection Error
Prompt user to enable Bluetooth
Wait for user to enable Bluetooth
User enabled Bluetooth?
No
Yes
Display Off Bluetooth & Display Device Buttons
Wait for user action
Off Bluetooth button Pressed ?
Yes
Get Devices Button Pressed ?
No
Yes
Dialog box to show Scan Error
Bluetooth Enabled ?
No
Yes
Scan for connectable devices for 5 secs
Put scan results in a array
Set array data into list of connectable devices
Display and update List Show Instruction to connect
Wait for user to connect to desired device from list
Connect to selected Device
Display Connection Success and Clear connectable device list
Dispenser Test
Wait for user action
Back Button Pressed ?
Process BLE END

Bluetooth Menu Toolbar's Dispenser and Back Button Flowchart





Flowchart for questions and results page


Questions Process
Display question activity
Initiate Attempt and total user Counter
Pick and Randomise Order of 5 out of 10 Qns.
Display Question Fragment
Question Correct ?
Yes
No
Display Dialog for Reason of Error or Display Hints
Add 1 to Attempt Counter
Add 1 to Attempt Counter
Display Next Question Fragment
Question counter equals 5 ?
No
Yes
Display Result Fragment
Add 1 to Total_Users
Wait for User to Input Name
Display User's Results + Total Users + End Quiz Instructions
Attempts Count <= 7 ?
Yes
No
Add 1 to Choco counter
Dispense 1 Chocolate
Choco Counter equals 5?
Yes
No
Display Dialog for Choco refill
Reset Choco counter to 0
Display Dialog for Choco success dispense
Add 1 to Choco counter
Dispense 1 Chocolate
Choco Counter equals 5?
Yes
No
Display Dialog for Choco refill
Reset Choco counter to 0
Display Dialog for Choco success dispense
Wait 5 secs for Chocolate to dispense and repeat process 2nd time
Display End Options
End Button Pressed?
Main Menu Button Pressed?
Turn off Bluetooth Reset Total_Users to 0
Head back to Starting Menu
End

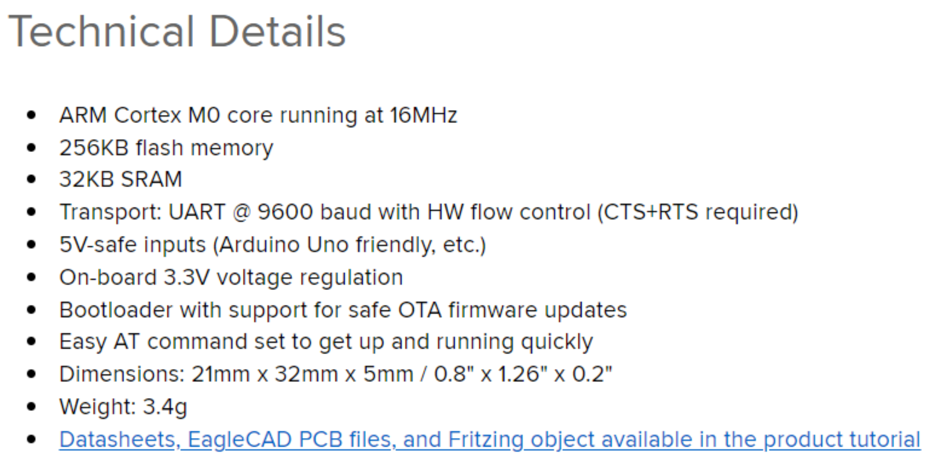
The link for flowcharts is in Appendix A

Specifications of Hardware and Software

Temi: https://www.robotemi.com/specs/

Adafruit Bluefruit LE UART Friend BLE:

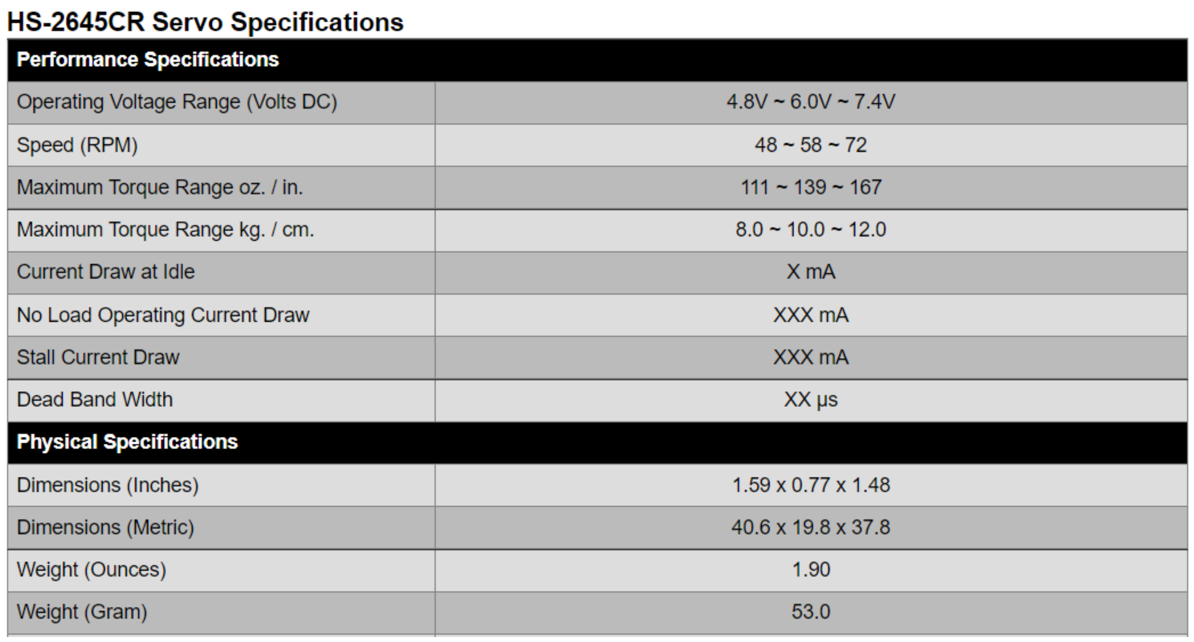
https://www.adafruit.com/product/2479



Technical Details

- ARM Cortex M0 core running at 16MHz
- 256KB flash memory
- 32KB SRAM
- Transport: UART @ 9600 baud with HW flow control (CTS+RTS required)
- 5V-safe inputs (Arduino Uno friendly, etc.)
- On-board 3.3V voltage regulation
- Bootloader with support for safe OTA firmware updates
- Easy AT command set to get up and running quickly
- Dimensions: 21mm x 32mm x 5mm / 0.8" x 1.26" x 0.2"
- Weight: 3.4g
- Datasheets, EagleCAD PCB files, and Fritzing object available in the product tutorial

Servo:



HS-2645CR Servo Specifications

Performance Specifications	
Operating Voltage Range (Volts DC)	4.8V ~ 6.0V ~ 7.4V
Speed (RPM)	48 ~ 58 ~ 72
Maximum Torque Range oz. / in.	111 ~ 139 ~ 167
Maximum Torque Range kg. / cm.	8.0 ~ 10.0 ~ 12.0
Current Draw at Idle	X mA
No Load Operating Current Draw	XXX mA
Stall Current Draw	XXX mA
Dead Band Width	XX µs
Physical Specifications	
Dimensions (Inches)	1.59 x 0.77 x 1.48
Dimensions (Metric)	40.6 x 19.8 x 37.8
Weight (Ounces)	1.90
Weight (Gram)	53.0

Android Studio Version used is Chipmunk 2021.2.1, the code using a minimum Android 6, and a minimum SDK of 23 and Java language is used.

Arduino UNO:
https://store.arduino.cc/products/arduino-uno-rev3

Conclusion

Accomplishment

TemiChoco is an appealing questionnaire application with Kahoot background music capable of Bluetooth Low Energy communication with Arduino UNO using Bluefruit LE Friend UART to activate the chocolate dispenser. TemiChoco application can be installed into the Temi robot and be interacted with Temi robot's interface, the chocolate dispenser is able to dispense chocolate by using the app in Temi robot.

TemiChoco can function smoothly, capable of picking five out of ten questions randomly, enrich new students with information about NYP during the quiz and dispensing one or two chocolates depending on user results as a reward when is used during NYP open house.

However, there are limitations which is the user cannot deny device's Bluetooth permission requests which will cause the app will crash, data transfer has a delay and the dispenser experience revolution deviation.

Temi's outreach is achieved as user can interact with TemiChoco using Temi's interface. For future development, Temi's outreach can be improved by integrating Temi's SDK into the app.

The 3D printed chocolate dispenser have been improved with secured screw and mounts secure the Arduino UNO and Bluefruit LE Friend UART, to prevent a structure failure and electronics hazard.

This project can give a basic understanding of Java programming, Android Studio and Bluetooth Low Energy implementations.

Future Enhancement

As Android Studio is a software designed for app development. The app design can be improvement further by using custom layouts for custom design of app widgets and the design of the UI. Advanced features and widgets that Android Studio provides like Snackbar, Recyclerview, Videoview and more can be developed to be integrated into a app. Examples of improvements are ranking list, timer for user to answer questions and more.

To further enhance Temi's outreach, the app could be integrated with Temi SDK and use Temi's features such as Mapping, Ai chat and movement. This will enhance the appeal that Temi must interest students.

Bluetooth code can be improved and polished, such as reducing the delay for data transfer, auto connect to a Bluefruit LE Friend UART and more.

The dispenser can be improved by adding feature such as LED for indications, IR sensor to monitor if chocolate has run out and buzz to provide user results on a question. These features can be implemented with the use of the Arduino UNO board used for the dispenser.

The dispenser model can be improved by using a material for 3D printing are better than 3D printed materials used for the current Dispenser or purchasing parts to build the dispenser. The material must be strong enough to hold the electronics and chocolates and smooth surfaces to dispense chocolate without friction affecting the dispensing process.

Appendix

Appendix A – Project Resources

Flowchart & Gantt Chart of ChocoTemi, done by Chua Choon Hsiang
https://miro.com/app/board/uXjVPnUwX-s=/?share_link_id=266134304488

Google Slides used for TemiChoco Presentation, done by Chua Choon Hsiang, Feb 2023
https://docs.google.com/presentation/d/1u5KK33OHmer3KooX4ajFzZ_u8U6AcQUZvnRnFlwjmOI/edit?usp=sharing

TemiChoco Project Codes Repository done by Chua Choon Hsiang.
https://github.com/elthef/Temi_NYP

Appendix B – Websites References

Android Developer Website created by Google Developers, Bluetooth overview.
https://developer.android.com/guide/topics/connectivity/bluetooth

Android Developer Website created by Google Developers, Set up Bluetooth.
https://developer.android.com/guide/topics/connectivity/bluetooth/setup

Android Developer Website created by Google Developers, Bluetooth permissions.
https://developer.android.com/guide/topics/connectivity/bluetooth/permissions

Android Developer Website created by Google Developers, Bluetooth Low Energy.
https://developer.android.com/guide/topics/connectivity/bluetooth/ble-overview

Android Developer Website created by Google Developers, Find BLE devices.
https://developer.android.com/guide/topics/connectivity/bluetooth/find-ble-devices

Android Developer Website created by Google Developers, Connect to a GATT server.
https://developer.android.com/guide/topics/connectivity/bluetooth/connect-gatt-server

Android Developer Website created by Google Developers, Transfer BLE data.
https://developer.android.com/guide/topics/connectivity/bluetooth/transfer-ble-data

Martijn van Welie, March 23, 2019, Making Android BLE work — part 1.
https://medium.com/@martijn.van.welie/making-android-ble-work-part-1-a736dcd53b02

Martijn van Welie, Apr 6, 2019, Making Android BLE work — part 2.
https://medium.com/@martijn.van.welie/making-android-ble-work-part-2-47a3cdaade07

Martijn van Welie, Apr 16, 2019, Making Android BLE work — part 3.
https://medium.com/@martijn.van.welie/making-android-ble-work-part-3-117d3a8aee23

Stuart Kent, January 11, 2018, Bluetooth Low Energy on Android: Top Tips for the Tricky Bits v4 (CodeMash).
https://speakerdeck.com/stkent/bluetooth-low-energy-on-android-top-tips-for-the-tricky-bits-v4-codemash

Adafruit Industries, Adafruit Bluefruit LE UART Friend - Bluetooth Low Energy (BLE).
https://www.adafruit.com/product/2479

Kevin Townsend, Nov 19, 2014, Introducing the Adafruit Bluefruit LE Friend (UART service).
https://learn.adafruit.com/introducing-adafruit-ble-bluetooth-low-energy-friend/uart-service

Kevin Townsend, Carles Cufí, Akiba, Robert Davidson, Getting Started with Bluetooth Low Energy.
https://www.oreilly.com/library/view/getting-started-with/9781491900550/ch04.html

Annianni, 10 Mar, 2021, Difference Between a Fragment and an Activity in Android.
https://www.geeksforgeeks.org/difference-between-a-fragment-and-an-activity-in-android/

Tedris, March 10, 2013, how to display Toast at centre of screen.
https://stackoverflow.com/questions/15321186/how-to-display-toast-at-center-of-screen

Dave, Dec 31, 2021 , BLE writing to a characteristic (Android Studio)
https://stackoverflow.com/questions/70545166/ble-writing-to-a-characteristic-android-studio

Waza_Be, Jul 3, 2011, Changing font size into an AlertDialog.
https://stackoverflow.com/questions/6562924/changing-font-size-into-an-alertdialog

sushma1008, Feb 11, 2017, Read data from ble in android app continuously.
https://stackoverflow.com/questions/42173412/read-data-from-ble-in-android-app-continuously

Androider, Apr 9, 2011, Android Preventing Double Click On A Button
https://stackoverflow.com/questions/5608720/android-preventing-double-click-on-a-button

user14381409, Oct 13, 2020, Create an array with random integers but with no duplicates [duplicate]
https://stackoverflow.com/questions/64340800/create-an-array-with-random-integers-but-with-no-duplicates

Rohit, Nov 21, 2011, Android background music service
https://stackoverflow.com/questions/8209858/android-background-music-service

Mustafa Kuloğlu, May 27, 2021, Android 12 New Bluetooth Permissions
https://stackoverflow.com/questions/67722950/android-12-new-bluetooth-permissions

knts, Feb 11, 2022, startActivityForResult is deprecated, I'm trying to update my code [duplicate]
https://stackoverflow.com/questions/71082372/startactivityforresult-is-deprecated-im-trying-to-update-my-code

Appendix C – Sample Codes References

Antonio García, Dec 3, 2021, Java language, Bluefruit LE Connect Android source code (V2), licensed by Adafruit.
https://github.com/adafruit/Bluefruit_LE_Connect_Android_V2

AnisurRahmanApel, May 11, 2022, Java language, Simple Quiz Application in Android Using Fragment
https://github.com/CodeForBoss/SimpleQuizApp-using-fragment

Michael(MichaelsPlayground), on Nov 6, 2022, Java language, Android BluetoothLeGatt Sample
https://github.com/MichaelsPlayground/BluetoothLeGattSdk33

Amir yazdanmanesh, Mar 11, 2022, Java and Kotlin languages, Bluetooth Low Energy Terminal
https://github.com/Amir-yazdanmanesh/Bluetooth-Low-Energy-Terminal-Android-BLE-Library

Appendix D – Videos References

Android Development for Beginners - Full Course, done by Meisam from freeCodeCamp.org YouTube Channel
https://www.youtube.com/watch?v=fis26HvvDII

Android Development for Beginners - Full Course (Part 2), done by Meisam from freeCodeCamp.org YouTube Channel
https://www.youtube.com/watch?v=RcSHAkpwXAQ

Java Tutorial for Beginners, done by Mosh from Programming with Mosh YouTube Channel
https://www.youtube.com/watch?v=eIrMbAQSU34

End of Report