VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Gonzalo Araya

ANDROID APPLICATION DEVELOPMENT AS A USER INTERFACE OF AN EMBEDDED DEVICE

Technology and Communication
2023

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Gonzalo Araya
Title	Android Application Development as a User Interface of an Embedded Device
Year	2023
Language	English
Pages	38
Name of Supervisor	Jani Ahvonen

This thesis project aims to develop a tablet (Android) application that will be used as a user interface to ring the bells of churches. Additionally, the application will have the feature of receiving SMS and trigger alarms with the objective of playing a song through USB serial communication.

The use of technologies will allow to improve and modernise church bell activities, such as playing bell songs and creating a bell sequence. This Android Java application uses serial (RS-232) over micro-USB to communicate with the embedded device, which will activate the relays that enable the ringing of the actual bell using automation. The customer is Techat Oy that develops the embedded device.

The thesis achieved it is main goal which is to create an Android application in order to communicate with the customers embedded device.

Keywords Application development, Java, serial communication.

CONTENTS

LIST OF FIGURES AND TABLES

LIST OF APPENDICES

LIST OF ABBREVIATIONS

OTG	On The Go
RN	React Native
SMS	Short Message Service
MVVM	Model-View-ViewModel
GUI	Graphical User Interface
UI	User Interface
API	Application Programming Interface
APK	Android Package Kit

1 INTRODUCTION

Nowadays, developing and innovating using new technologies has become an essential method for simplifying, enhancing inclusiveness, and automating processes. The thesis project will refer to the design, implementation, testing and deployment of an Android application which will be used as a user interface for an embedded device to automate the handling of church bells.

The Android application was designed to have an easy, quick-to-learn, intuitive and stable user interface to handle functionality, such as playing sequence bell songs, creating new bell songs, scheduling sequence bell songs, SMS-based interface and automated control of bell tower doors. The connection between an Android device and an embedded device will be established using the RS-232 protocol via a micro-USB cable.

The objective of this project was the development and deployment of an Android application which will satisfy the customer requirements and expectations to consolidate a business relationship.

2 TECHNOLOGIES SELECTION

2.1 Technologies to Communicate with an Embedded Device on Android

The selection of the technology stack was a critical factor in the Android application development process, which led to an analysis and comparison of different framework possibilities to ensure an efficient and effective project design.

2.1.1 React NativeReact

React Native is an open-source framework for building mobile applications based on React, Facebook's JavaScript library for building user interfaces. It presents a practical approach for developers to write code for both iOS and Android platforms with a single codebase, reducing development time and effort. The main feature is that written for the natural language of the computer, allowing native code to perform better and faster because they were created to suit their platforms. /1/

React Native provides a cost-effective and efficient solution for the creation of native cross-platform mobile applications. This framework is used for companies such as Meta, Pinterest, Microsoft, Salesforce, Airbnb, Tesla and Shopify. /2/

2.1.2 Java

Java is a widely used programming language for the development of Android mobile applications. It is officially recognized for use in Android development and it is supported by the Android operating system. Java offers a rich set of libraries, APIs and tools that are crucial for building robust and efficient Android apps. Its object-oriented programming paradigm makes it easy to write, maintain and extend code. Java has its own runtime environment.

Java was released in 1995 by Sun Microsystems, which is now owned by Oracle, and most of its features are available in open source and has an active community of developers. /3/

2.1.3 Kotlin

Kotlin was designed to improve the Java programming language. Released to the public as open source in the year 2012, Kotlin has become popular language to develop Android applications because is a statically typed, object-oriented programming language that is interoperable with the Java virtual machine (JVM), Java Class Libraries and Android. Kotlin can be compile to JavaScript or java class files.

Developers choose this language to develop Android applications because of its benefits, for example, interoperability with Java code, easy learning curve, reduced programming time, object-orientated and functional programming, cross-platform and flexibility.

The Kotlin Foundation, established by JetBrains and Google, is responsible for managing Kotlin and promoting its development and progression. /4/

2.1.4 Flutter

Flutter is a free and open-source mobile UI framework created by Google and released in May 2017. In a few words, it enables creating a native mobile application with only one codebase. This means that one programming language and one codebase can be used to create two different apps (for iOS and Android). Flutters uses Dart language as its base language. /5/

Developers choose this language to develop Android applications because it has benefits, for example, quick compilation (Hot-Reload), good documentation, cross-platform, and own rendering engine. /6/

This framework is used for companies, such as Google, Groupon, Toyota, BMW and eBay. /7/

2.2 Selection of the Framework Language

The initial framework selected to develop the project was React Native, utilizing JavaScript language. This decision was made due to its capability to produce fast-built, responsive and visually attractive applications with a relatively smaller development team. /8/

The initial phase of the Android application development was focused around creating and implementing the various layouts views based on the approved concept design. This was prioritized due to the delayed arrival of the serial device. Once the embedded device arrived, the attempt to establish serial communications with the device was unsuccessful, leading to the decision to reorganize the project.

The new framework language selected for the project was Java, which was a significant decision due to its impact on the complexity of the development of the Android application. The use of an OTG connection to establish an RS-232 communication with the embedded device is critical and Java provides the necessary resources and support to achieve this.

3 PROJECT DEFINITION

The project definition was discussed and agreed upon with the clients to determine the scope, features, functionalities, preliminary design and schedule of the project. During the project's design and implementation, modifications were made to enhance the user experience of the Android application and ensure timely completion within the project scope. These changes were necessary to ensure success and expanded on the original requirements.

3.1 Requirements

Based on the customer meetings, a list of functional requirements (Table 1.), external interfaces (Table 2.) and non-functional requirements (Table 3.) was elaborated for the application.

Table 1. Functional requirements

Reference	Description	Priority
F1	Create song sequence	2
F2	Save song sequence	2
F3	Play song sequence	1
F4	Open and close door	2
F5	Monitor client state	2
F6	Create an alarm to trigger play song	1
F7	Choose a song sequence from the tablet	1
F8	Start the chosen song from the tablet	1
F9	Play the bell sequence one time manually	2

F10	Start the play song through SMS	1
F11	Upkeep a whitelist of phone numbers	2
F12	Play and listen the play song	2
F13	Send a song list by SMS	2
F14	Open/close bell tower doors	2

Table 2. Non-functional requirements

Feature	Description
Recovery from a fault situation	When the tablet is restarted, the Android bell application can be run normally. Estimated recovery time is less then 5 minutes
Usability	The user interface must be easy for all potential users, having an intuitive and easy to learn user interface. Estimated time to learn it is 1 hour
Safety	Files are stored locally

Table 3. External Interfaces

Reference	Description
I1	Alarm handler
I2	Serial communication for an embedded device
I3	SMS receiver and send

3.2 Technical Communication with Serial Device

The communication between the Android application and the embedded device was established through a micro-USB connection to a USB port on the embedded device. The communication process using the RS-232 protocol will be facilitated by utilizing the OTG (On-The-Go) technology for Android devices. Table 4 details the protocols defined in the embedded device by Techat.

Table 4. Serial command embedded device

Hex	Description
0x01 – 0x1E	Delay (1 second to 30 second)
0x65	Bell 1
0x66	Bell 2
0x67	Bell 1 and Bell 2
0x68	Bell 3
0x69	Bell 1 and Bell 3
0x6A	Bell 2 and Bell 3

0x6B	Bell 1, Bell 2 and Bell 3
0x6C	Bell 4
0x6D	Bell 1 and Bell 4
0x6E	Bell 2 and Bell 4
0x6F	Bell 1, Bell 2, and Bell 4
0x70	Bell 3 and Bell 4
0x71	Bell 1, Bell 3, and Bell 4
0x72	Bell 2, Bell 3, and Bell 4
0x73	Bell 1, Bell 2, Bell 3, and Bell 4
0x74	Stop track
0x75	Open door
0x76	Stop door
0x78	Define male in SMS burial (prefix)
0x79	Define female in SMS burial (prefix)
0xC9	Start message
0xCA	Stop message

3.3 Use Case Diagram

The use case diagram of the application represents user and user interaction scenarios with the system. Figure 1 describes the user as an actor, which will have access to all the application features. The user will manage the serial device through the Android application interface by playing a selected song directly, or indirectly, through an SMS receiver or setting a trigger allowing to play a selected song. Additionally, it will have the feature of opening/closing the bell tower doors, defining language and entering a trusted phone number.

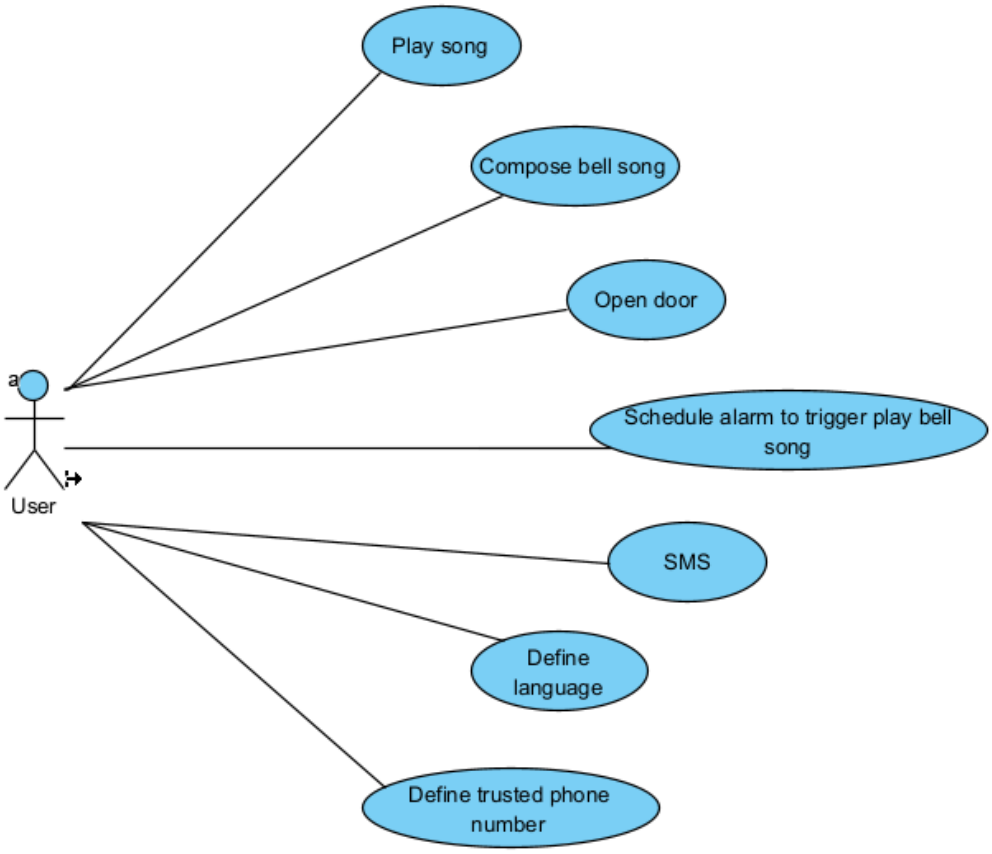

Figure 1. Use case diagram

3.4 Preliminary User Interface

The early user interface design was made creating two simple views. The purpose was to define and present a concept design idea allowing to fulfil customer expectations and develop a well-designed layout.

Figure 2 contains the play-compose layout view, divided into three horizontal sections. The upper section shows six buttons, representing a large bell, a small bell and four different time delays. The middle section presents a dropdown menu of song choices, which upon selection displays a view combining the chosen bell and delays. The lower section exhibits the composition list in plain text form.

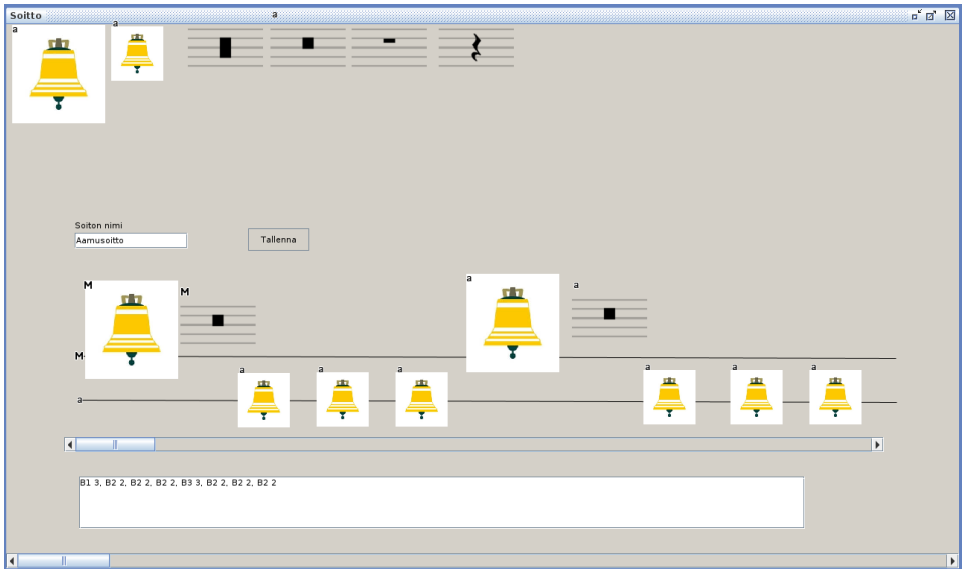

Figure 2. Android application play-create song layout

Figure 3 represents the set-display alarm layout view, divided into two sections. The top section includes a calendar, time input and song selection for setting a new alarm. The bottom section displays the current alarm settings and provides the option to delete them.

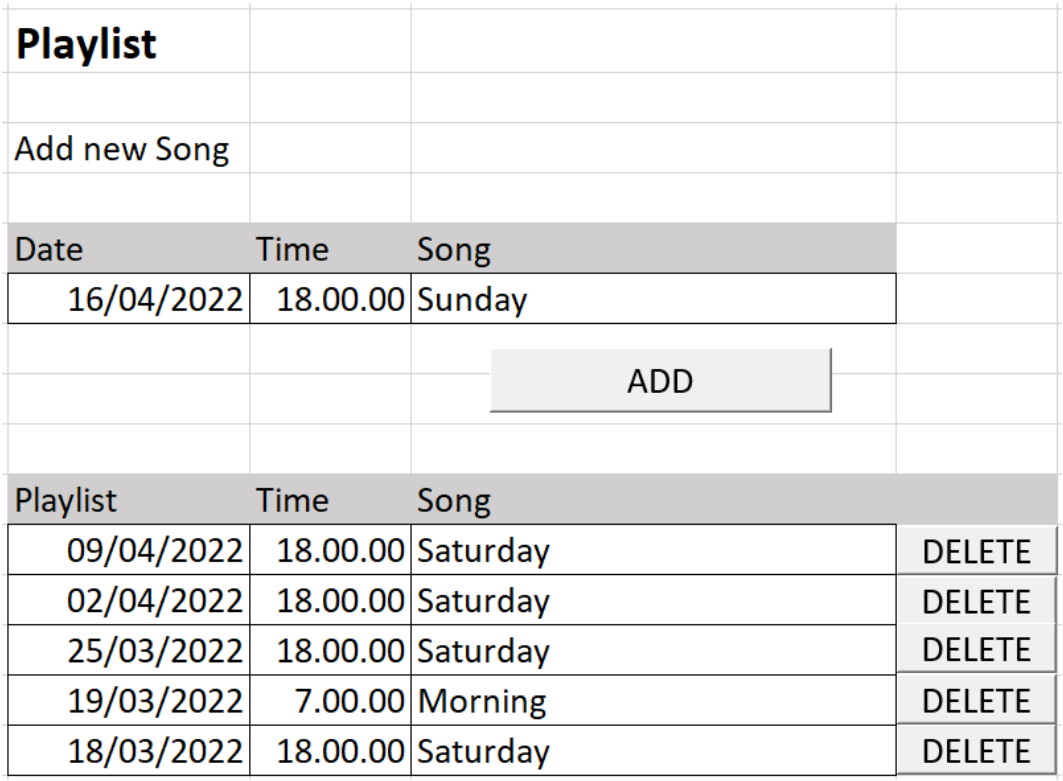

Figure 3. Android application set-display alarm layout

4 DESIGN

The design of the application was developed following the requirements described in the previous chapter. The architecture for the project was the Model-View-ViewModel (MVVM) pattern. The layout view was designed to create an easy and intuitive user interface.

4.1 Android Tablet Wireframes

The layout was designed according to the specifications provided by the customer. Figure 4 contains the Play layout view where the user can play a song from the song list. The upper section contains 3 buttons: play, edit and delete. The following section displays the songs list created by the user.

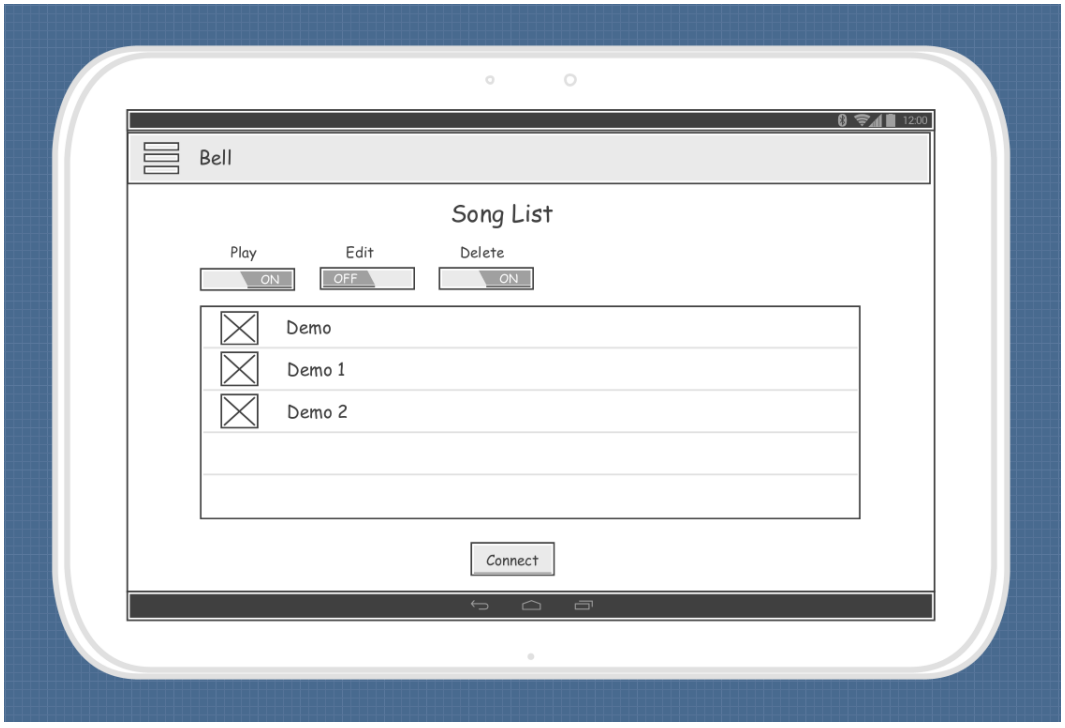

Figure 4. Play layout view

Figure 5 contains the Create layout view to create a new song using bell icon-buttons and a slide delay with range between 1 to 30 seconds. The sequence created will be handled as a string and the name of the song will be unique.

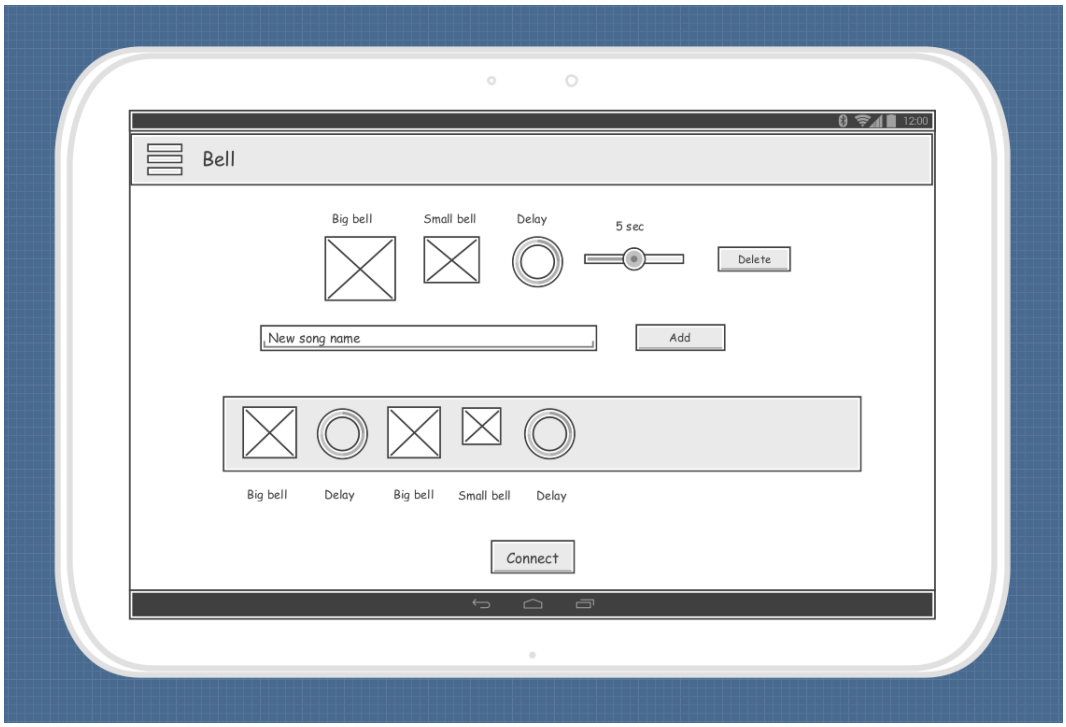

Figure 5. Create layout view

Figure 6 contains the Alarm Create layout view to create a new alarm, that can be a single, daily or weekly alarm, which can be chosen in the upper section from the view. In the lower section the date, time and a song can be picked from the songs list.

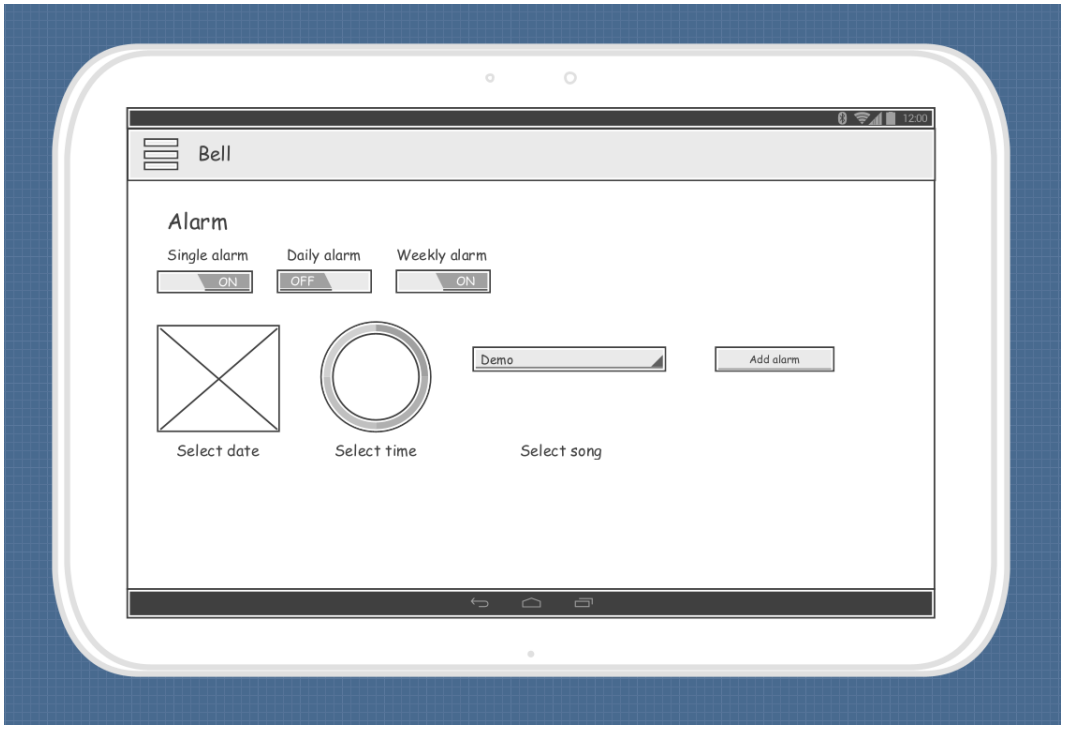

Figure 6. Set alarm layout view

Figure 7 represents the Alarm layout view to display the existing alarms, which is divided into 2 sections. The upper section contains a delete button and the lower section which displays a list of alarms. These display alarms are divided into two vertical sections, the left-hand side displays a single alarm and the right-hand side represents recurring alarms.

4.2 Architecture

The project was developed using the architecture pattern MVVM. Figure 8 details the package diagram distribution and communication between View, Model, ViewModel, PureJavaClass and Adapters. The view package holds the different layout views shown on the screen, Model is the package where all models are located where the models will receive the input data from the view and will process it. The ViewModel package holds ViewModel files which will work as an intermediary of changes of state of the model state. PureJavaClass package

contains a custom Java class and adapters hold SMS receivers, Alarm receivers and list adapters. /9/

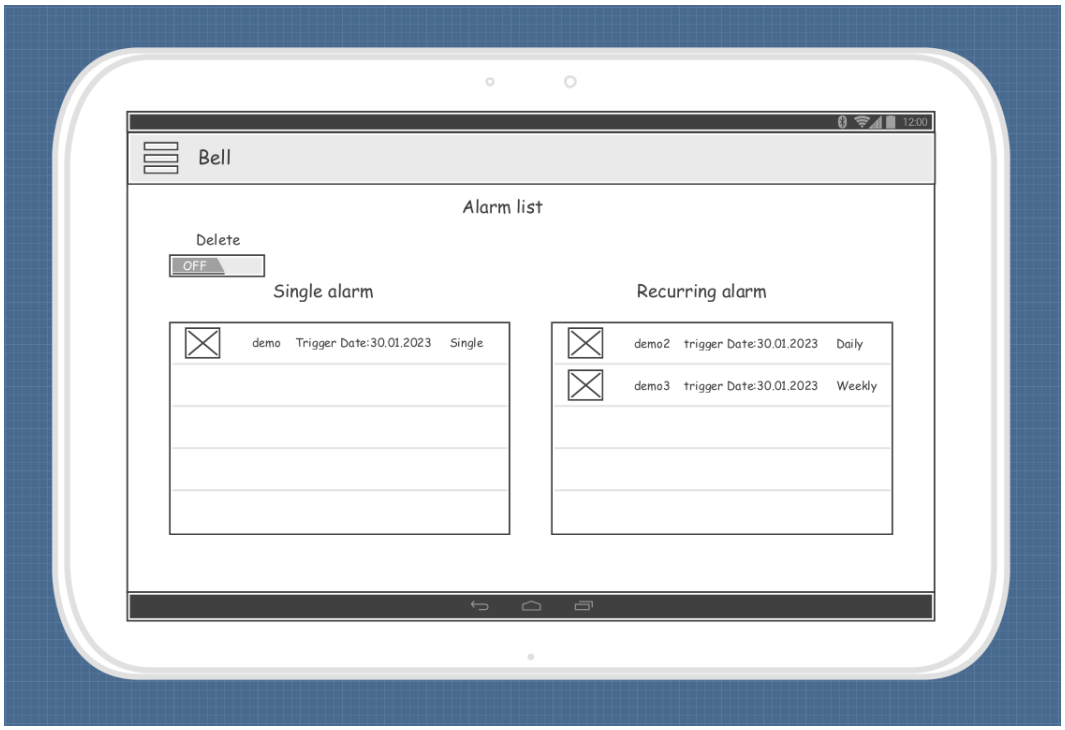

Figure 7. Alarm list layout view

4.3 SMS Class Diagram

Figure 9 details classes related to the SMS feature. When the Android application is launched, MainActivity will initiate the SMS Receiver adapter to wait for a trigger action. When an SMS is received on the Android device, the action will be broadcast to the Android application and handled by the SMS Receiver. The message will then be processed using the smsHandle method, followed by sending a response SMS. If the response SMS is validated, the message will be converted from plain text to the protocol format of the embedded device and send it through serial communication with the embedded device, which concludes the SMS process.

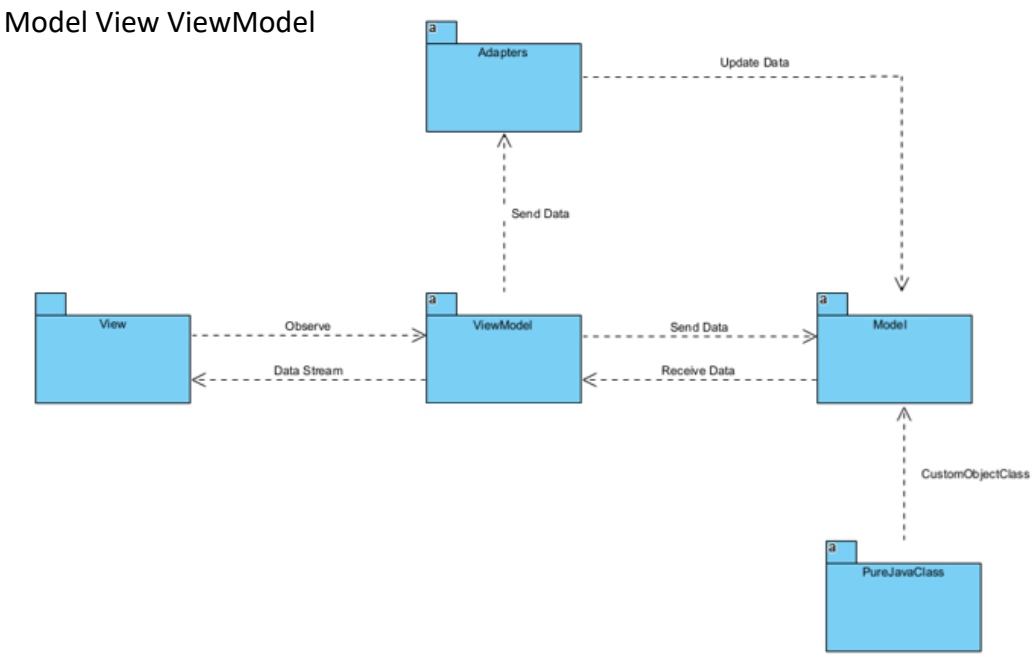

Figure 8. Package diagram

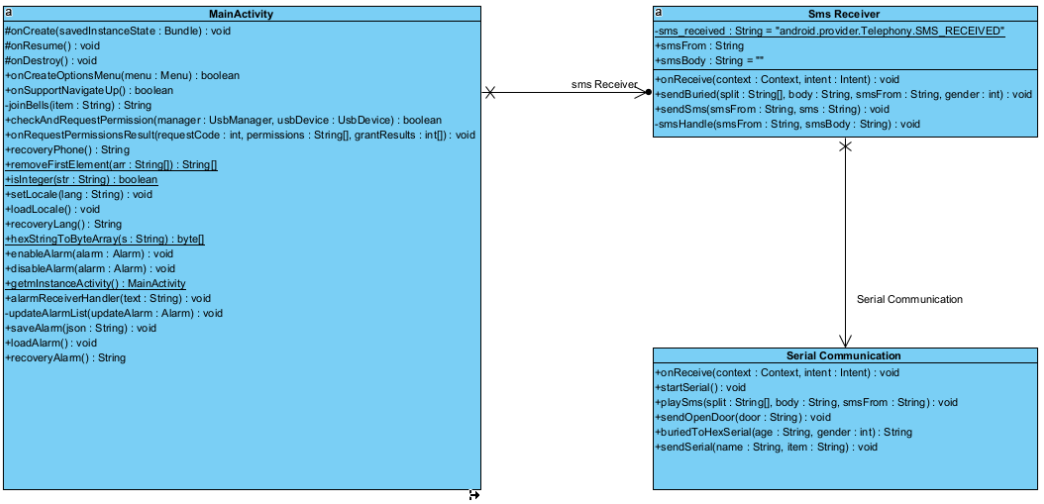

Figure 9. SMS class diagram

4.4 Play Bell Song Sequence Diagram

The play sequence diagram shows the main feature of the application. Figure 10 contains the diagram that represents the user play song interaction. When the user clicks the play button and selects a song in playView, the selected song will be received in the playFragment. The playFragment will handle the action using the playListAdapter to update the playFragment and play view through the playView Model. Additionally, playListAdapter will communicate with SerialCommunication that will start the communication process with the embedded device through USB connection.

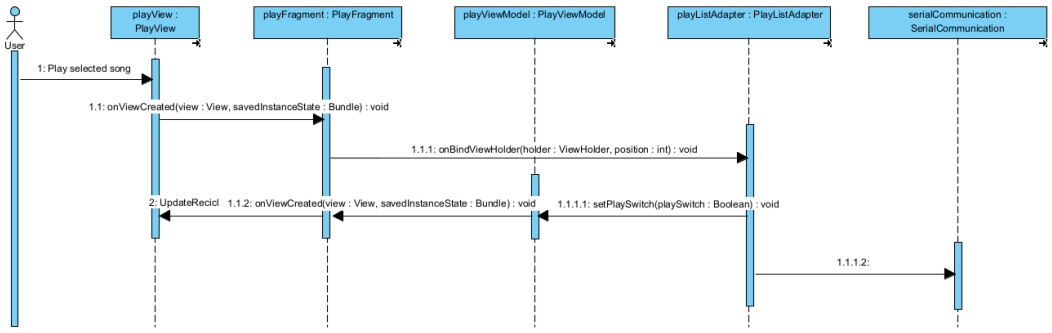

Figure 10. Play bell sequence diagram

4.5 Compose Bell Song Sequence Diagram

Creating new bell songs is an important feature to the project. Figure 11 sequence diagram show how the song create sequence is processed. When a user interacts with the create layout view, they will typically input various parameters, such as the name of a song, the type of bell to be used, and the desired delays. These inputs are received and processed by the createFragment. The createFragment then communicates with the playViewModel to update the song list in the playFragment through the playListAdapter. This allows the updated list of songs, which includes the newly created layout, to be displayed in the playView.

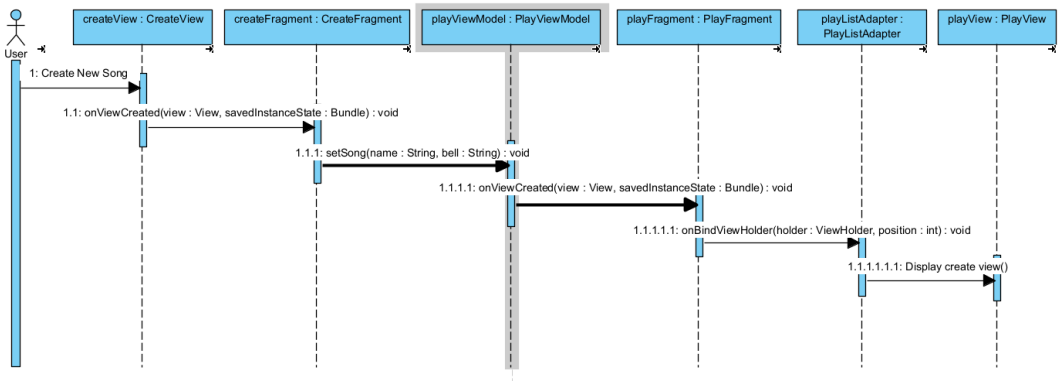

Figure 11. Create bell sequence diagram

5 IMPLEMENTATION

5.1 Android Studio – Java Application

The implementation started with creating a new Java Android application utilizing Android Studio, which initializes a base code structure for a new application. Following steps were to identify, analyse and development user interface and serial communication as major priorities.

5.2 User Interface

The user interface was developed to be intuitive and easy to learn. The main view was designed to play a song from the play list (Figure 12).

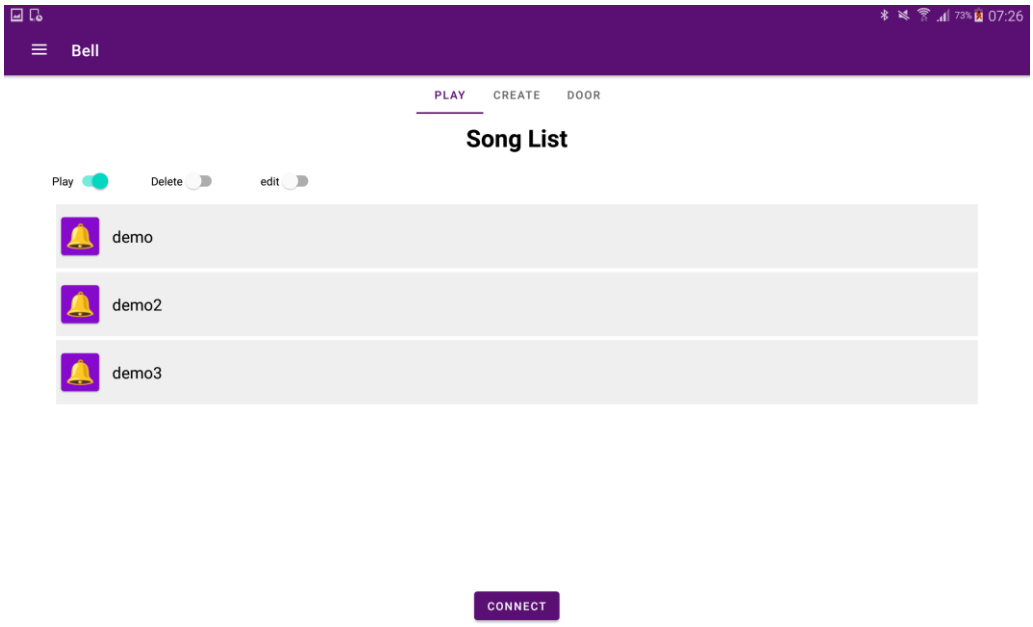

Figure 12. Play layout view

The implementation of create song was updated from the requirements to handle delays from 1 to 30 seconds utilizing sliders (Figure 13). When user adds a new note or delay, they will be displayed in the lower section of the view, and the name will be validated to be unique. When the user adds a new song, the view will be switched to play view with the updated song list.

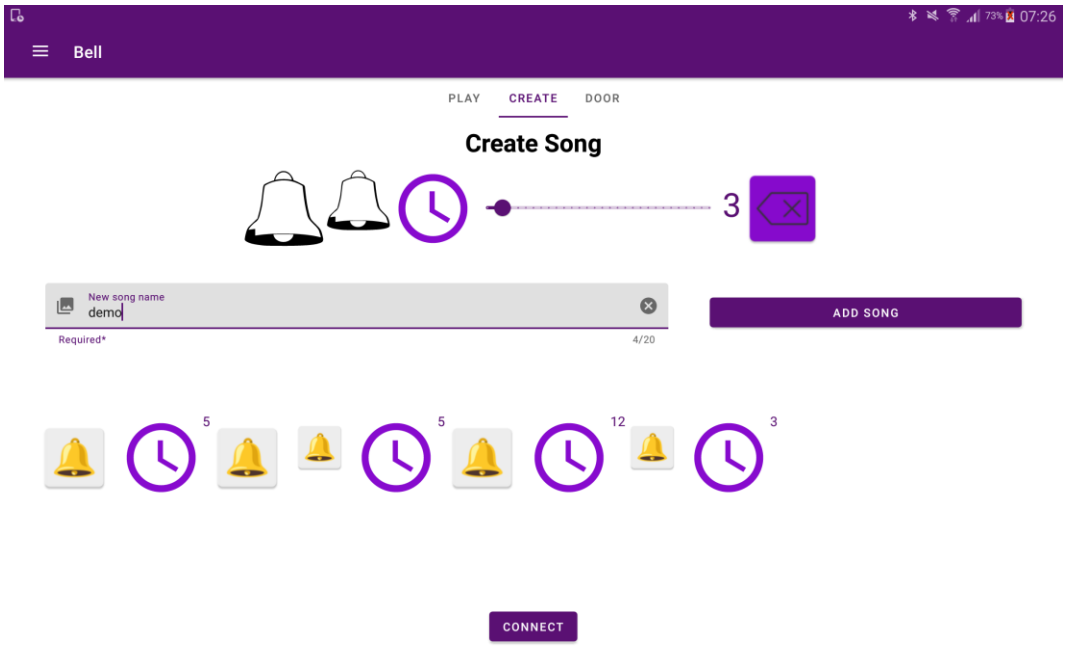

Figure 13. Create layout view

The Create new alarm view was implemented, as per design (Figure 14). The user will input a date, time and song from the view, and it will be added to the alarm list view (Figure 15).

5.3 Serial Communication

The Android application was developed to communicate with an embedded device. The connection between an Android device and an embedded device is established using the RS-232 protocol via a micro-USB cable.

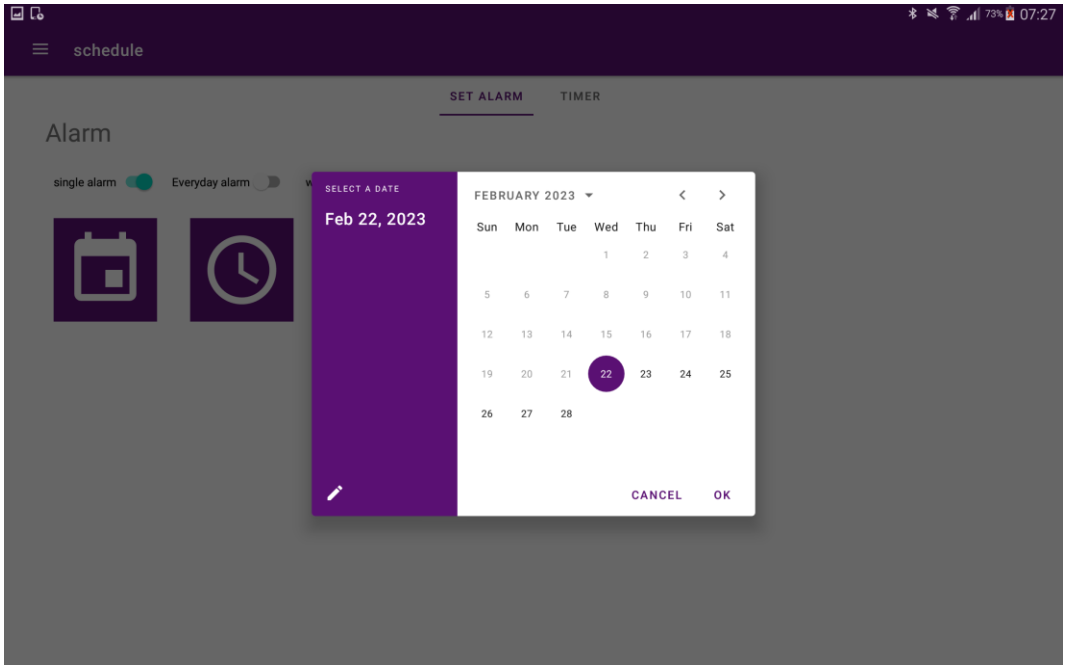

Figure 14. Set alarm view

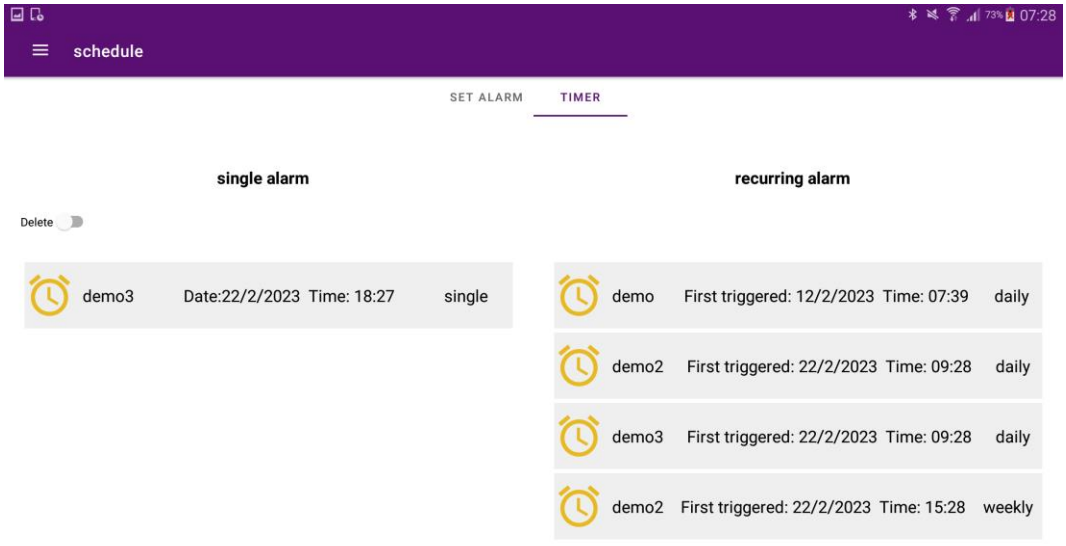

Figure 15. Alarm list layout view

The current version need is physically connected using a microUSB – USB cable, USB multiport HUB, and USB extension. The driver is CH340, and was specified by the customer.

5.3.1 Start Serial Communication

In order for the application to communicate with the embedded device, a serial port connection must be opened between them. The dependency imported to the project included a list of defaults drivers. When an embedded device is connected, it will automatically detect the connection and set the driver as CH340 (Code Snippet 1). The startSerial function will be initiated automatically when the application is launched. The application also has a button to establish connection manually. The parameters for RS-232 connection was baud rate 9600, 1 stop bit and none parity.

```
public void startSerial() {
    if (usbSerialPort == null) {
        UsbSerialDriver driver;
        UsbManager manager = (UsbManager) getSystemService(Context.USB_SERVICE);
        List<UsbSerialDriver> availableDrivers = UsbSerialProber.getDefaultProber().findAllDrivers(manager);
        if (availableDrivers.isEmpty()) {
            Toast.makeText(getApplicationContext(), "No serial USB devices!", Toast.LENGTH_SHORT).show();
            return;
        }
        try {
            driver = availableDrivers.get(0);
        } catch (Exception ignored) {
            Toast.makeText(getApplicationContext(), "No correct serial USB devices!", Toast.LENGTH_SHORT).show();
            return;
        }
        UsbDeviceConnection connection = manager.openDevice(driver.getDevice());
        if (connection == null) {
            Toast.makeText(getApplicationContext(), "Error opening USB device!", Toast.LENGTH_SHORT).show();
            return;
        }
        try {
            usbSerialPort = driver.getPorts().get(0);
            usbSerialPort.open(connection);
            usbSerialPort.setParameters(9600, 8, UsbSerialPort.STOPBITS_1, UsbSerialPort.PARITY_NONE);
            Toast.makeText(getApplicationContext(), "setting devices attached in ports", Toast.LENGTH_LONG).show();
            Toast.makeText(getApplicationContext(),  "Connection successful!", Toast.LENGTH_SHORT).show();
        } catch (IOException e) {
        }
    }
}
```

Code Snippet 1. Start Serial Communication

5.3.2 Write Data through Serial Communication

Code Snippet 2 represents a function that converts and writes data over a serial port after a connection has been established. The function takes two arguments: name, a string representing the name of the song, and item, a string containing bells and delays. The item string is first converted to a byte array and then each byte in the byte array is transformed into 8 bits using bitwise AND operation with the value 0xFF. The resulting 8-bit values are then stored in a buffer. Finally, the buffer is written to the serial port using write() method.

```
public void sendSerial(String name, String item) {
    byte[] body;
    String temp = "";
    int counter = 1;
    String[] parts = itemTemp.split("!");
    serialBuffer[0] = (byte) ((0xC9) & 0xFF);
    for (int i = 0; i < parts.length; i++) {
        Integer number = Integer.parseInt(parts[i]);
        parts[i] = String.valueOf(number);
        String hex = "";
        if (Integer.toHexString(Integer.parseInt(parts[i])).length() < 2) {
            hex += "0";
        }
        hex += Integer.toHexString(Integer.parseInt(parts[i]));
        temp += hex;
    }
    body = hexStringToByteArray(temp);
    for (byte bytes : body) {
        serialBuffer[counter] = (byte) (bytes & 0xFF);
        counter++;
    }
    serialBuffer[counter] = (byte) ((0xCA) & 0xFF);
    try {
        Toast.makeText(getApplicationContext(), "Playing " + name, Toast.LENGTH_SHORT).show();
        usbSerialPort.write(serialBuffer, 0);
    } catch (Exception e) {
        Toast.makeText(getApplicationContext(), "Error writing to serial port!", Toast.LENGTH_SHORT).show();
    }
}
```

Code Snippet 2. Send Serial Communication

5.4 SMS

Code Snippet 3 shows the function when the SMS is received by the Android application. When the Android device receives the SMS, the onReceive() method of a BroadcastReceiver is triggered. This method receives an intent with the SMS information, which can be decomposed into details related to the sender and body of the SMS.

To listen for SMS events, the application needs to register a BroadcastReceiver in the AndroidManifest.xml file with an intent filter for the "android.provider.Telephony.SMS_RECEIVED" action. /10/

```
public class ReceiveSms extends BroadcastReceiver {

    1 usage
    private static final String sms_received = "android.provider.Telephony.SMS_RECEIVED";
    1 usage
    public String smsFrom, smsBody = "";
    @Override
    public void onReceive(Context context, Intent intent) {
        if(intent.getAction().equals(sms_received)){
            Bundle bundle = intent.getExtras();
            SmsMessage[] messages;
            if(bundle != null) {
                try{
                    Object[] pdus = (Object[]) bundle.get("pdus");
                    messages = new SmsMessage[pdus.length];
                    for(int i=0; i<messages.length; i++){
                        messages[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
                        smsFrom = messages[i].getOriginatingAddress();
                        smsBody = messages[i].getMessageBody();
                    }
                } catch (Exception e){
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Code Snippet 3. SMS receiver function

6 TESTING

The objective of this chapter is to analyse and test the system functionality by creating a set of test cases. The impact of the system implementation will also be evaluated based on the testing results to ensure its reliability and performance. Table 5 show the information results from the unit, integration and UI test performed, testing code are giving in Code Sniet (appendix A).

Table 5. Testing cases table

S/N	Test Case Description	Input	Expected Result	Pass/Fail
1	Transform bell song format to embedded device protocol format	301!5!301!5!302!5!301!5!301!302!5!	101!5!101!5!102!5!101!5!103!5!	Pass
2	Receive SMS and divide content to gender and age	M12	men m12	Pass
3	Receive SMS and divide content in gender and age	N23	women N23	Pass
4	Transform gender and age to serial device protocol format	1.23 (age) 2. 1 (gender)	120!301!2!301!2!302!2!302!2!302!2!	Pass
5	Receive SMS with content 'H' and return SMS response	H	playlist	Pass
6	Receive SMS with content 'P2' and play indicated song	P2	play p2	Pass

7	Receive SMS with content 'D1' and open indicated door	D1	door d1	Pass
8	Receive SMS with undefined command content and re-turn SMS response	test	unknown command	Pass

7 DEPLOY

After thorough testing and customer approval, the final release of the application was produced by creating an APK file using the Android Studio IDE's built-in func- tion- ality. The APK file was then delivered to the customer for use, ensuring that the application met their requirements and was thoroughly tested.

The deployment process included building the APK file, creating and signing a re- lease certificate and installing and running the application on the target Android device.

The release certificate was signed to the final version of the application for distri- bution to our customer. This certificate will help to ensure that the application has not been modified for a third party, providing security and trust for users.

8 CONCLUSIONS

The thesis led to the successful creation of an Android application that enables communication with an embedded device, meeting the expectations of the customer. The project was delivered on schedule and received positive feedback from the customer.

The project implementation was considered successful. Nevertheless, the selection of the language-framework could have been enhanced through better communication with the customer. This, in turn, would help to identify and prevent similar issues in the future.

The end result has been satisfactory, as the customer expressed a desire to add new features to the Android application and the embedded device that was developed. Furthermore, there are plans to expand the partnership between the companies.

REFERENCES

/1/ Eisenman, Bonnie. (2016). Learning React Native. 1st Ed. O'Reilly Media Inc, the USA.

/2/ Melnyk, Olga. 2022. Top 10 Big Companies Using React Native. Accessed 16.3.2023. https://careerkarma.com/blog/companies-that-use-react-native/

/3/ Vaguez, Levi. 2022. Kotlin vs. Java for Android development. Accessed 14.2.2023. https://blog.logrocket.com/kotlin-vs-java-android-development/

/4/ Android Developers. Kotlin overview. Accessed 12.2.2023. https://developer.android.com/kotlin/overview

/5/ Brain Mentors. Flutter Overview. Accessed 12.2.2023. https://brain-mentors.com/flutter-overview/

/6/ Azumo Research. 2023. What is Flutter?. Accessed 16.3.2023. https://azumo.com/insights/what-is-flutter

/7/ Khomutova, Sofia. 2022. Top 24 famous apps built with Flutter Framework. Accessed 16.3.2023. https://apexive.com/post/top-apps-built-with-flutter-framework

/8/ Clark, Mariana. 2022. Top 10 Advantages of React Native. Accessed 16.3.2023. https://blog.back4app.com/react-native-advantages/

/9/ Sehgal, Anmol. 2018. Common Android Architectures (MVC vs MVP vs MVVM). Accessed 16.3.2023. https://anmolsehgal.medium.com/common-android-architectures-mvc-vs-mvp-vs-mvvm-afd8461e1fee

/10/ Stack Overflow. Sending and Receiving SMS and MMS in Android (pre Kit Kat Android 4.4). Accessed 20.2.2023. https://stackoverflow.com/quetions/14452808/sending-and-receiving-sms-and- mms-in-android-pre-kit-kat-android-4-4

APPENDIX A – Test code

```
@Test
public void joinBells() {
    String input = "301!5!301!5!302!5!301!5!301!302!5!";
    String expected = "101!5!101!5!102!5!101!5!103!5!";
    assertEquals(expected, MainActivity.joinBells(input));
}

@Test
public void buriedToHexSerial() {
    String age = "23";
    int gender = 1;
    String expected = "120!301!2!301!2!302!2!302!2!302!2!";
    assertEquals(expected, MainActivity.buriedToHexSerial(age, gender));
}

@Test
public void smsSongM12() {
    String inputBody = "m12";
    String expected = "men m12";
    assertEquals(expected, MainActivity.smsSong(inputFrom, inputBody, inputFrom));
}

@Test
public void smsSongN23() {
    String inputBody = "N23";
    String expected = "women n23";
    assertEquals(expected, MainActivity.smsSong(inputFrom, inputBody, inputFrom));
}

@Test
public void smsSongH() {
    String inputBody = "H";
    String expected = "playList";
    assertEquals(expected, MainActivity.smsSong(inputFrom, inputBody, inputFrom));
}

@Test
public void smsSongP() {
    String inputBody = "P2";
    String expected = "play p2";
    assertEquals(expected, MainActivity.smsSong(inputFrom, inputBody, inputFrom));
}

@Test
public void smsSongD() {
    String inputBody = "D1";
    String expected = "door 1";
    assertEquals(expected, MainActivity.smsSong(inputFrom, inputBody, inputFrom));
}

@Test
public void smsSongTest() {
    String inputBody = "test";
    String expected = "unknown command";
    assertEquals(expected, MainActivity.smsSong(inputFrom, inputBody, inputFrom));
}
```

Code Snippet A-1. Unit and Integrated testing

```java
@Test
public void createSong(){
    onView(withId(R.id.tab_layout))
            .perform(click());
    onView(withId(R.id.textInput))
            .perform(typeText( stringToBeTyped: "UITest"));
    onView(withId(R.id.bigBell))
            .perform(click());
    onView(withId(R.id.slider))
            .perform(click());
    onView(withId(R.id.btn_add_tag))
            .perform(click());
}

@Test
public void useAppContext() {
    // Context of the app under test.
    Context appContext = InstrumentationRegistry.getInstrumentation().getTargetContext();
    assertEquals( expected: "fi.ekseli.bell", appContext.getPackageName());
}
```

Code Snippet A-2. Unit and Integrated testing