National College of Ireland
BSc in Computing
2016/2017

Richard Mangan
X13114514
richard.mangan@student.ncirl.ie

Rapid*ARM*

Technical Report

National College *of* Ireland

Submission of Thesis and Dissertation

National College of Ireland

Research Students Declaration Form

(Thesis/Author Declaration Form)

Name: Richard Mangan

Student Number: X13114514

Degree for which thesis is submitted: B.Sc Computing

Material submitted for award

(a) I declare that the work has been composed by myself.

(b) I declare that all verbatim extracts contained in the thesis have been distinguished by quotation marks and the sources of information specifically acknowledged.

(c) My thesis will be included in electronic format in the College Institutional Repository TRAP (thesis reports and projects)

(d) I declare that no material contained in the thesis has been used in any other submission for an academic award.

Signature of research student: ______________________________________

Date: ______________________

Table of Contents

EXECUTIVE SUMMARY **7**

1 INTRODUCTION **8**

1.1 DEFINITIONS, ACRONYMS AND ABBREVIATIONS 8
1.2 PURPOSE 8
1.3 PROJECT SCOPE 8
1.4 BACKGROUND 9
1.5 AIMS 9
1.6 TECHNOLOGIES 10
1.6.1 GitHub – Version control repository *10*
1.6.2 Firebase – Realtime database console *10*
1.6.3 JSON – Lightweight data format *10*
1.6.4 Bluetooth Low-Energy – Device communication *10*
1.6.5 Mobile Application *11*
1.6.5.1 Firebase Android API – Real-time Cloud Database 11
1.6.5.2 Splash Screen – Loading Animation 11
1.6.5.3 Material Intro Slider – Introduction Guide 11
1.6.5.4 MaterialDrawer - Material Design Navigation 11
1.6.5.5 Background Services – Receiving data from the smart-watch 11
1.6.5.6 Broadcast Receiver - Recent Activity Overview 11
1.6.5.7 MPAndroidChart - Visualising & Graphing Data 12
1.6.5.8 FingerprintManager - Fingerprint Authentication 12
1.6.5.9 TextToSpeech API - Creating audio from user details 12
1.6.5.10 Timer - Countdown to abort call 12
1.6.5.11 MediaPlayer – reading audio data to call receiver 12
1.6.6 Smart-watch Application *13*
1.6.6.1 Background service – Reading heart rate 13
1.6.6.2 SensorManager - Using heart rate sensor 13
1.6.6.3 Data API - Sending data via Bluetooth 13
1.6.7 Raspberry Pi - IoT Development Board *14*
1.6.7.1 Python IDLE – Programming language 14
1.6.7.2 Firebase Python API 14
1.6.7.3 Rpi.GPIO 14
1.6.7.4 8-Channel relay board 14
1.6.7.5 Threading 14
1.6.7.6 JSON library 14
1.6.7.7 Time library 14

2 DOCUMENT STRUCTURE **15**

3 SYSTEM **16**

3.1 DOCUMENT CONTROL 16
3.1.1 Revision history *16*
3.1.2 Distribution list *16*
3.1.3 Related documents *16*
3.2 REQUIREMENTS 17
3.2.1 User requirements *17*
3.2.2 Usability requirements *17*
3.2.3 Environmental requirements *17*
3.2.4 Non-Functional Requirements *18*
3.2.4.1 Performance/Response time 18
3.2.4.2 Availability 18
3.2.4.3 Recovery 18
3.2.4.4 Security 18
3.2.4.5 Reliability 18
3.2.4.6 Portability 18

3.2.4.7 Extendibility 18
3.2.5 Functional Requirements 19
3.2.5.1 Requirement 1 – Registration 20
3.2.5.2 Requirement 2 – Login 21
3.2.5.3 Requirement 3 – Post Profile Information 22
3.2.5.4 Requirement 4 – View Profile Information 23
3.2.5.5 Requirement 5 – Read heart rate 24
3.2.5.6 Requirement 6 – Update heart rate status 25
3.2.5.7 Requirement 7 – Phone call 26
3.2.5.8 Requirement 8 - Emergency call 27
3.2.5.9 Requirement 9 – Toggle home 28
3.2.5.10 Requirement 10 – View history 29
3.2.5.11 Requirement 11 – Night-mode 30
3.2.6 Data Requirements and Design 31
3.2.6.1 Required data 31
3.2.6.2 Database rules 31
3.2.6.3 Backup and recovery 31
3.2.6.4 Data limits 31
3.2.6.5 Performance monitoring 31
3.2.6.6 Logical Data Model 32
3.3 DESIGN AND ARCHITECTURE 33
3.3.1 Architecture Diagram 33
3.3.2 Sequence Diagram 34
3.3.3 Class Diagram – Mobile Application 35
3.3.4 Communication Diagram 36
3.3.5 Mobile Application Use-case Diagram 37
3.3.6 Smart Watch Application Use-case Diagram 38
3.3.7 Raspberry Pi Application Use-case Diagram 38
3.4 IMPLEMENTATION 39
3.4.1 Firebase – Realtime database console 39
3.4.2 JSON – Lightweight data format 39
3.4.3 Bluetooth – Device communication 39
3.4.4 Android Mobile Development 40
3.4.4.1 Firebase Android API – Cloud Database 40
3.4.4.2 Splash Screen – Loading Animated Screen 41
3.4.4.3 Intro Cards – Overview Guide 41
3.4.4.4 MaterialDrawer - Material Design Navigation 41
3.4.4.5 Background Service – Receiving data from the smart-watch 42
3.4.4.6 Recent Activity Overview – Broadcast Receiver 42
3.4.4.7 Heart rate gauge – Updating in real-time 43
3.4.4.8 Visualising & Graphing Data - MPAndroidChart 43
3.4.4.9 Fingerprint Authentication 44
3.4.4.10 Creating audio from user details – synthesizeToFile 44
3.4.4.11 Countdown to abort call 45
3.4.4.12 Night mode – Lowering expected heart rate during sleep 45
3.4.4.13 Loudspeaker – Reading audio data to call receiver 46
3.4.4.14 Unlock front door - Smart-home cloud control 46
3.4.5 Android Wear Smart-watch Development 47
3.4.5.1 Background service – Reading heart rate 47
3.4.5.2 Using heart rate sensor – SensorManager 47
3.4.5.3 Sending data to the phone – DataItems 48
3.4.5.4 Testing - Generating random heart rate data 49
3.4.6 Raspberry Pi - IoT Development Board 50
3.4.6.1 Python – Programming language 50
3.4.6.2 Firebase Python API 50
3.4.6.3 Rpi.GPIO 50
3.4.6.4 8-Channel relay board 51
3.4.6.5 Threading 51

3.4.6.6 JSON library ... 51
3.4.6.7 Time library ... 51
3.5 GRAPHICAL USER INTERFACE MOCK-UP ... 52
3.5.1 Login and main user interface layouts ... 52
3.5.2 Smartwatch user interface ... 52
3.6 TESTING AND EVALUATION ... 53
3.6.1 Unit Testing ... 53
3.6.1.1 Test 1 – checkRate ... 54
3.6.1.2 Test 2 – sendResult ... 55
3.6.1.3 Test 3 – emergencyCall ... 56
3.6.1.4 Test 4 – getData ... 57
3.6.1.5 Test 5 – countdownTimer ... 58
3.6.1.6 Test 6 – getFirebaseData ... 59
3.6.1.7 Test 7 – setSwitchState ... 60
3.6.1.8 Test 8 – sendNewToggleState ... 61
3.6.1.9 Test 9 – buildGraphs ... 62
3.6.1.10 Test 10 – updateCharts ... 63
3.6.2 Usability Testing ... 64
3.6.2.1 Usability Survey ... 64
3.6.2.2 Survey results ... 64
3.6.3 System Testing ... 73
3.7 RESEARCH ... 74
3.7.1 Medical Research ... 75
3.7.2 Potential Userbase ... 76
3.7.2.1 Persons over 65 living alone ... 76
3.7.2.2 Percentage of over 65's who are living alone ... 77

4 CONCLUSIONS ... 78

4.1 ADVANTAGES ... 78
4.2 FURTHER DEVELOPMENT OPPORTUNITIES ... 78
4.3 LIMITATIONS ... 78

5 REFERENCES ... 79

6 APPENDIX ... 80

6.1 APPLICATION SCREENSHOTS ... 80
6.1.1 Introduction Slider UI ... 80
6.1.2 Main User Interface ... 81
6.1.3 Profile Activity ... 81
6.1.4 Fingerprint Authentication UI ... 82
6.1.5 Smart-home Control UI ... 82
6.1.6 Profile Edit Activity ... 83
6.1.7 Heart Rate History UI ... 83
6.1.8 Cloud Database Console ... 84
6.2 MEDICAL PROFESSIONAL SURVEY TEMPLATE ... 85
6.3 MEDICAL SURVEY RESPONSES ... 87
6.4 UNIT TEST CLASSES ... 88
6.4.1 MyWearServiceTest ... 88
6.4.2 CallNotificationTest ... 89
6.4.3 ControllerActivityTest ... 90
6.4.4 GraphActivityTest ... 90
6.4.5 LoudspeakerActivityTest ... 91
6.4.6 MainActivityTest ... 92
6.5 USABILITY TEST SURVEY ... 93
6.6 PROJECT PROPOSAL ... 96
6.6.1 Objectives ... 96

6.6.2 Background *97*
6.6.3 Technical Approach *98*
6.6.4 Special Resources Required *98*
6.6.5 Technical Details *99*
6.6.6 Project Plan *100*
6.6.6.1 Idea Formulation: 100
6.6.6.2 Presentation Slides: 100
6.6.6.3 Project Pitch: 100
6.6.6.4 Reflective Journal - September: 100
6.6.6.5 Project Proposal: 101
6.6.6.6 Identify Supervisor: 101
6.6.6.7 Reflective Journal – October: 101
6.6.6.8 Project Prototype: 101
6.6.6.9 Reflective Journal – November: 101
6.6.6.10 Midpoint Presentation: 101
6.6.6.11 Reflective Journal - December: 101
6.6.7 Evaluation & Testing *102*
6.6.7.1 Unit testing 102
6.6.7.2 Integration Testing 102
6.6.7.3 System Testing 103
6.6.7.4 Evaluation Testing 103
6.7 MONTHLY JOURNALS 104
6.7.1 September *104*
6.7.2 October *105*
6.7.3 November *106*
6.7.4 January *107*
6.7.5 February *108*
6.7.6 March *109*

Executive Summary

This report has been produced to detail the research and development of a medical and home automation project, known herein as *RapidARM*.

RapidARM is primarily a health monitoring and alert system that autonomously calls for help if a user suffers with a cardiac emergency. The service continuously monitors the users heart rate activity through a smart watch application, and a smart phone application performs actions based on these readings when compared with heart rate parameters set by the user on initial setup. The service also functions as a smart-home hub in which a user can interact with common electrical devices in their home without the need to do so manually. This includes controlling lighting, television, radio and heating. The service also automatically unlocks the users' front door in the event of an emergency. The mobile application UI provides functions to instantly launch a call to the emergency services or to a selected contact, while also displaying an overview of the most recent heart rate activity to the user.

The system has been designed to automatically call the emergency services when it detects heart rate conditions that are deemed outside of the acceptable parameters, allowing a brief period for the user to interrupt the call. The heart rate parameters are configurable, based on the individual health requirements of the user, as advised by their doctor. User data is used to create an audio file that is saved to their device, and in the event of a trigger, this file is played to the emergency services when an automatic call is evoked. During an emergency trigger, the phone will display the users' identification details and medical history while also unlocking their front door.

RapidARM is built using an android mobile device, an Android Wear smart-watch, a RaspberryPi development board, and the Firebase API to provide authenticated user accounts and profile data, as well as saving a record of all events triggered within the system. This database exists in the cloud and can be accessed through the mobile application or through a web browser. An instance of the database is also stored within the mobile application itself for use when a network connection is not available. Events and data changes within the system occur in real-time, and a Raspberry Pi device and 8-channel relay board are used to receive control messages from the cloud and to interact with the users' home based these control messages.

Valuable contributions to the project were made by Professor. Richard Costello and Dr. Abir Alsaid including forming baseline figures for upper and lower safe heart rates, and in gauging the benefits of such a system with patients. Surveys completed by the contributors have been attached in the appendix.

Through research into the possible user base for the system conducted using data from the Census report 2011, It was identified that over 900'000 people above the age of 65 currently live alone in Ireland. Also highlighted in the research was the number of disabled people living in Ireland, at over 590'000 in the same 2011 report [3], [5].

Throughout testing, the system has been shown to address the problem consistently and reliably, providing a history of heart activity for the user to examine in the form of an events list, graphs and charts of data gathered from the smart-watch. Feedback was received on possible improvements during the usability testing and the results of this are contained in this document. *RapidARM* has been shown to provide a lifeline to the user who would otherwise be incapable of calling for assistance.

Research has been conducted into viable alternative options within the space and these have been evaluated to identify possible strengths, weaknesses or opportunities for the RapidARM project [1], [7]. The details of this research are included in the references section.

1 Introduction

1.1 *Definitions, Acronyms and Abbreviations*

Wear	The Operating system of the wearable device
Contact Person	The person nominated by the user for emergency contact
Mobile	Refers to any mobile device used for the service other than the smart watch
RapidARM	Used when referring to the application or service as a whole
Smart-watch	Android Wear device that is running the application
Development board	Embedded computer, used for smart-home features. Raspberry Pi etc.

1.2 *Purpose*

The purpose of this document is to define the technical details, the problem and the proposed solution for the *RapidARM* project. This document describes the service and its functions, as well as the requirements for the development of *RapidARM* and the technologies implemented within. This document will also include research conducted in the area, and testing methodologies used in the final deliverable.

1.3 *Project Scope*

This project will to create a health monitoring and smart-home application to alert users to dangerous heart activity and allow control of electrical devices in the home. *RapidARM* is intended to provide a level of independence particularly to elderly or disabled users by allowing them to interact with their home and to call for help without needing assistance.

The scope of the project is to develop a multi-device application, consisting of an embedded computer, mobile application and a smart-watch application to monitor the users' heartrate and to react to readings with the appropriate action by calling the emergency services and communicating an automated message to the receiver. The users' front door shall be unlocked by the application when an emergency call is initiated and their contact and medical details shall be outputted to the user interface of the mobile device.

A smart-watch application shall read the wearers heart rate at regular intervals, using an embedded optical heart rate sensor and transmit the readings to the mobile application for processing, using Bluetooth LE. The device shall require no interaction by the user and should transmit the readings in the background without waking the device screen or otherwise using excessive power unnecessarily.

The mobile application shall implement a real-time database with cloud backup and shall feature authenticated user accounts, user profiles, heart rate history and smart-home control features. User profiles shall contain contact details for the user, medical details, location and contact details for a nominated contact. The users' history shall contain lists, graphs or charts to display their heart rate events and shall require an authenticated fingerprint to secure access to this data. The mobile application shall generate audio files based on user entered data, for use during emergency calls.

An application written for an embedded device shall be used to provide smart-home features. This application shall interact with the cloud database to read control instructions in real-time and to control electrical devices using this data. This application shall allow the user to control lighting, heating, appliances and actuation of the front door lock, using a multi-channel relay board that receives input from the embedded device through GPIO pins. This application shall update the state information to the user to ensure that the mobile application accurately represents the current state of the devices that are being controlled.

1.4 Background

The inspiration for this project came from several sources as an attempt to address the problem of unaccompanied people calling for help in the event of an emergency.

This project idea was formulated because of the growing number of people, particularly the elderly or disabled who live alone and lack the ability to call for help in the event of an emergency. The project aims to bring a technological approach to monitoring the health of the user and initiating a call for assistance when the user is alone and unable to do so. The intention is to provide an extra layer of safety and security for a user and their families, that gives peace of mind and enables them to live longer and more independently in their own homes, reducing the need for them to move into supervised accommodation, where possible.

The author and developer is a former motor technician who has for several years been working to develop solutions that allow disabled users to interact with their world in ways that they could not before, with the aid of innovative technology. Because of this work it was identified that there was an opportunity to bridge these assisted technologies with the *Internet-of-Things* and to bring real quality-of-life benefits and independence to the user.

1.5 Aims

The project aims to allow the user to register secure accounts and backup their data to the cloud. The user shall be able to add profile data to their account and user data shall be used to create an audio file listing their contact details and location. This audio file should be stored on the local device and overwritten whenever changes are made to the users' profile information. Their heart rate should be taken automatically at a set interval and this data should be sent to the mobile device through a Bluetooth connection. The mobile device should backup this data to the cloud and decide if action is required based on the heart rate parameters set in the users' profile.

The service shall start an automated call if the users' heart rate is outside of safe parameters, allowing a brief period before completing so that the user can interrupt the call of necessary. During this call, the audio file created from the users' data shall be played to the emergency services.

The users' recent heart activity overview shall be presented in the main user interface and shall also contain buttons that allow the user to initiate a phone call with a single touch, either to a chosen contact or to the emergency services. A separate "History" activity shall present the user with a list of all heart activity in the database, including graphs and charts to visualise the data.

The application shall allow the user to control electrical devices in the home remotely, through their connection to an IoT development board. This should receive command messages through the Firebase cloud service and in turn should interact with a multi-channel relay board to turn on or off electrical devices. The relay board can be used as a switch for any electrical devices in the home, including lighting, heating, television and shall also include remote unlocking of the user's front door during an emergency situation.

1.6 Technologies

This section of the report provides a brief overview of the technologies used in the application and how they are implemented. A more detailed explanation of how these technologies are implemented and how they function is provided in the *Implementation* section of the document and includes screen shots and code snippets.

1.6.1 GitHub – Version control repository

GitHub was used throughout the development of the application as a method for managing version control, backup and recovery of the various applications within the service [5]. The service was developed across multiple machines and synchronized through GitHub, a process that would have been much more laborious without the use of such version control systems. Several issues occurred along the way that were remedied with the help of the analysis tools provided by GitHub to identify rouge changes in the code that had caused the problems and to enable to developer to revert to a known-good state.

1.6.2 Firebase – Realtime database console

Firebase was selected as the chosen database solution for the application, due to its excellent support across many mobile operating systems, web and IoT devices. Firebase is a JSON based cloud hosted object oriented database that consists of a nested tree-like structure, containing key-value pairs [11]. The application retains an instance on the local device that ensures that the service is available even when network connection is interrupted. When changes to the data occur either in the cloud or on the device, a notification of change is sent alongside a "*dataSnapShot*". This snapshot is essentially the parent node of the object that had been changed within the tree and is used to overwrite only the targeted set of data. The benefit of this method comes in the speed at which data is available and updated within the system. Instead of loading an entire record of an individual, which would include all their profile information and list of events in the system, the dataSnapShot can pass back and forward individual nodes within the profile and avoid loading in data that is unnecessary in the current context, resulting in smaller messages and faster load times.

By using a key-value database like this, the developer avoids the need to write complex and potentially buggy code to convert application data into a relational form for storage on the device or online. This structure allows greater freedom to alter the database and its data types quickly and easily, due to its schema-less design. Additionally, the availability of the database through a simple URL in the Firebase Developer Console means that opportunities exist to use the generated data to build dashboards or analytics tools on top, using easily available charting and graphing tools such as *HighCharts* and *Tableau* that are already designed to consume key-value data.

1.6.3 JSON – Lightweight data format

JSON is used extensively throughout all areas of the application, as it is a streamlined, efficient and human readable format for the data that is easily parsed, while containing only the required data and no unnecessary characters. All communication between the smart-watch, mobile application and Raspberry Pi use JSON. The database is also based on a JSON nested object.

1.6.4 Bluetooth Low-Energy – Device communication

The Android wear device is connected to the mobile application using Bluetooth Low Energy to extend the battery life of the watch as much as is possible [8]. The communication between the watch and mobile application is conducted using "*DataItems*". These DataItems are very small streams of JSON data that are limited in size to ensure that poorly constructed messages cannot hold open a communication channel unnecessarily, causing excessive power consumption. Larger messages can be sent by serializing multiple messages, if needed.

1.6.5 Mobile Application

The project is largely developed on the Android platform, using Android Studio 2.3.3 and Java libraries. The mobile application is written in Java with the UI written in XML. Various other libraries are used to perform specific functions within the application, details of which can be found below and in the *Implementation* section of this document.

1.6.5.1 Firebase Android API – Real-time Cloud Database

Database functions within the application have been implemented using the Firebase API [11]. Firebase is a cloud database solution for mobile and web. This application includes a local instance of the database on the mobile device that is used to communicate data changes back and forth between the cloud and the mobile application.

1.6.5.2 Splash Screen – Loading Animation

An animated splash screen is used on loading the application. This animation is written in XMP and uses the *RapidARM* logo to perform the animation for 3 seconds, during which the application is establishing a connection to the database and checking whether this is the first run of the application and if the user is authenticated or not.

1.6.5.3 Material Intro Slider – Introduction Guide

When the user is authenticated, they are presented with a "*FirstLoadActivity*" containing slider cards to guide them through the application and explain what the application does. These intro slides have been built using the *TangoAgency Material-Intro-Screen* library [14], [17].

1.6.5.4 MaterialDrawer - Material Design Navigation

Material design has been used throughout the application to create a seamless and intuitive experience to the user. This Google design philosophy aims to provide a unified experience across all platforms [#] and to provide a natural and fluent navigation through an application. This is done with the use of slider menus, dynamic animated layouts, shadowing and the use of responsive images and layouts that are changed dynamically, based on the DPI of the viewers' screen. *RapidARM* uses the *MaterialDrawer* library [14], [16] as a basis for all navigation throughout the application. This library allows the developer to produce sliding navigation menus and can include icons, widgets and account information within the navigation drawer.

1.6.5.5 Background Services – Receiving data from the smart-watch

When the smart-watch is connected, it will attempt to transmit data as soon as it is captured by the heart rate sensor. To allow this to happen reliably, it was required to develop a background service to receive this data. Using a background service means that the application does not need to be open to receive the incoming message, or the user can be using other activities within the application without interfering with the core functionality.

The background services contain many methods for transmitting data, receiving updates, persisting data to the database and broadcasting data to other activities within the application. These include implementations of *DataEventsReceiver*, *BroadcastManager*, *BroadcastReceiver* and *DataItems*. A more detailed explanation is provided in the Implementation section.

1.6.5.6 Broadcast Receiver - Recent Activity Overview

The BroadcastReceiver library in Android is used in the application to send data from background services to user interfaces [2]. This processes incoming heart rate readings and ensures the user interface always displays the latest reading accurately. Since background services cannot interact directly with the user interface, a broadcaster and receiver is required to make the data available to any class in the application that requires it.

1.6.5.7 MPAndroidChart - Visualising & Graphing Data

The application provides the ability to view all historic heart rate data that the service has received from the smart-watch and to view it in text or graph form [12]. The *MPAndroidChart* library is a set of tools to enabled the developer to build various chart types and to populate them with key-value data. These charts have been implemented in the "History" section of the application to display both the most recent data points and the list of all data points available. A line chart shows only the last 10 entries so that the view does not become overpopulated with data, making it difficult to interpret. Another chart features all the data points available in a bar-chart that allows the user to scroll and to zoom in on areas within the data.

1.6.5.8 FingerprintManager - Fingerprint Authentication

The fingerprint API is used in the mobile application to ensure that the users' historical heart rate data is not freely available to anyone who can access the device [2], [13]. This API enlists the operating systems encrypted fingerprint verification technology and prompts the OS to inform the application whether a user is authenticated or not. The sensitive fingerprint data is never accessible to developers or to the application, by design, and instead the API acts as a gatekeeper to identify if the current users' print matches the registered owner of the phone.

1.6.5.9 TextToSpeech API - Creating audio from user details

Data gathered from the user during signup is processed to create an audio file that is later read to the emergency services when a call is made [10]. The function uses the inbuilt *TextToSpeech* engine and synthesizeToFile function in Android to read the user entered text, while an input stream instance is used to save the resulting audio to local storage in the application. This process is programmed to repeat whenever the profile data is edited, to ensure that the audio file always contains the latest information.

1.6.5.10 Timer - Countdown to abort call

When a trigger of the system occurs, the user is presented with a countdown timer indicating how many seconds they have left to cancel the call before it is initiated. This activity also calls out the timer decrements through the phones' loudspeaker to ensure that the user is aware of the impending call. This function is created using an instance of *CountDownTimer* in Java, and a countdown duration is specified in milliseconds. A message *Handler* is used to allow the countdown to run on a separate thread to the main thread. Without the use of a new thread for the countdown, the main thread would not be accessible to run the cancel operation when the user interacts with the button. On completion of the timer, the call to the emergency services is started and after a short wait time, a *run* method begins a new *Intent* activity. This ends the existing process and starts the *LoudspeakerActivity* class.

1.6.5.11 MediaPlayer – reading audio data to call receiver

The *LoudspeakerActivity* class is responsible for loading the audio file from local storage and playing the audio to the call receiver [10]. This is performed with the use of a *MediaPlayer* instance which receives the Uri and path to the file and then proceeds to play the audio. This class is also responsible for remotely unlocking the users' front door and for outputting the users contact and medical details onscreen for the benefit of the first responders.

1.6.6 Smart-watch Application

An Android Wear smart-watch is used to take readings from the users’ wrist through an optical heart rate sensor. The Moto 360 1st generation watch was selected for the project due it its availability and its convenient wireless charging. An application was developed for the watch and consisted of a simple UI that displayed the last known heart rate of the user. As per *Google* development guidelines, all processing was handled in the mobile application, where a much more powerful processor and greater battery capabilities were available.

1.6.6.1 Background service – Reading heart rate

The smart-watch application consists of a main activity class, used for the user interface and a background service class. The background service contains the bulk of the functionality for the application. This background service uses the *googleAPIClient*, *SensorManager* interface and *DataItem* objects.

1.6.6.2 SensorManager - Using heart rate sensor

SensorManager is a generic class that is part of the *Google* API library used to access onboard sensors in mobile and wearable devices [2]. The background service includes an instance of the *SensorManager* and this instance is then assigned to the required sensor. The *SensorManager* provides methods for *onSensorChanged* and *onAccuracyChanged*. These methods are used to get values from the sensor and to measure how accurately the sensor can read the data from the user.

1.6.6.3 Data API - Sending data via Bluetooth

When data is successfully received from the sensor, it is passed to a method for transmission to the mobile device [8]. This is done using the *DataApi* and the *googleAPIClient* to make a connection with the mobile device. The raw data from the sensor is combined with an automatically generated timestamp from the system time into a *Map* object. This resulting *DataItem* object is then transmitted to the mobile device and a listener is used to check whether transmission was successful. If the message delivery failed or of the mobile device isn’t currently connected, the messages are queued and delivered as soon as a connection is available.

1.6.7 Raspberry Pi - IoT Development Board

A *Raspberry Pi* development board is used alongside the *Raspbian* Linux based operating system to run the code needed for the smart-home features. A smart-home kit has been built using the development board and an 8-channel relay board that can be tapped into the users' mains system at the breaker panel. This allows the user to control up to 8 traditional electrical devices within their home without the requirement for these devices to be "*smart*" enabled. The development board is accessed over the network using the *Putty* SSH client and file transfers to the device are done using *FileZilla* for Secure File Transfer Protocol.

1.6.7.1 Python IDLE – Programming language

A *Python* script was developed in the *Python IDLE* IDE, containing the code necessary for reading the cloud control messages and for interacting with the relay board. The script is written in *Python 2.7* and implements the following libraries.

1.6.7.2 Firebase Python API

The *Firebase API* provides the necessary libraries to access the database and to read and write data to it. A unique database URL is required from the *Firebase* console, alongside an *Auth* token used for authenticating the user, if necessary.

1.6.7.3 Rpi.GPIO

Rpi.GPIO is a library used for interacting with the *General-Purpose-Input/Output* pins on the *Raspberry Pi* development board. This library enables the developer to define a pin on the board, assign it to either an input or output and to use that pin to interact with a wide range of devices by feeding 3.3v through the pin.

1.6.7.4 8-Channel relay board

The relay board is connected to the *Raspberry Pi*'s +5v VCC, Ground and 8 individual GPIO pins. The relay state is set to "*ON*" when no voltage is received on the input pin. When the *Python* code triggers the GPIO pin to *"GPIO.output("pinNumber", 1)"*, the pin receives 3.3v and triggers the relay state to switch to the "OFF" position. Alternately, setting the GPIO pin to *"GPIO.output("pinNumber", 0)"*, results in the no voltage being supplied to the relay and the state being switched to "ON".

1.6.7.5 Threading

Threading is used in the *Python* application as a method of handling simultaneous publishing and subscribing to and from the *Raspberry Pi*. This benefits the system by insuring that processes are not blocked from completing while another process is using the main thread, and ensures that all state information in accurate at the time of viewing it.

1.6.7.6 JSON library

The *JSON Python* library is used to parse the response from *Firebase* into an object that can be iterated over, allowing for efficient extraction of individual pieces of data. As result of this, a single generic method that addressed all pins for switching the state can be written, instead of individual methods for each pin.

1.6.7.7 Time library

The *Time* library in *Python* is a simple function that enables the developer to write in a delay time and is very useful in situations where a pause is needed within a loop to prevent the script from looping too quickly and occupying valuable processor and memory resources.

2 Document Structure

Functional requirements

This section of the document details the functional requirements of the system. Each requirement is defined in terms of its description, scope and process flow.

Data requirements

This section of the document details the data requirements for the *RapidARM* system, including the database design, its rules and components, the types of data collected and how this data is handled in the application.

User requirements

The user requirements section contains a detailed breakdown of the requirements of the system from the end user perspective. This includes the performance the service must provide to the user and its availability.

Design and architecture

This section defines the system design in relation to the mobile. smartwatch and embedded device applications. This section includes the architecture, class, sequence and communication diagrams.

Use cases

This section contains the use case diagrams for the mobile and smartwatch application and the embedded device application, showing all use cases as listed. These use cases are addressed in the functional requirements.

Graphical User Interface

This section contains mock-ups of the user interfaces for the main application layout. Smart-watch layout and registration and login layouts.

Testing

This section contains the details and results of the testing phase. This includes Unit testing, integration testing and system testing.

Research

The research section contains details of required research into comparable products or services, potential user base and medical research for the function of the application.

Conclusion

The conclusion of the document, outlining the progress and achievements in addressing the proposed problem.

3 System

3.1 Document Control

3.1.1 Revision history

Date	Version	Scope of Activity	Prepared	Reviewed	Approved
14/10/2016	1	Create	--	X	X
21/10/2016	1.1	Update	--	X	X
16/02/2017	2	Add Requirements spec	--	X	X
06/03/2017	2.1	Add monthly reports	--	X	X
18/03/2017	3	Combine all for final report	--	X	X
20/03/2017	3.1	Added testing docs	--	X	X
06/04/2017	3.2	Edit and proof read	--	X	X
05/07/2017	4	Final edit and proof read	--	X	X

3.1.2 Distribution list

Name	Title	Version
Eamon Nolan	Lecturer	4.0
Richard Mangan	Student	4.0
Cristina Muntean	Supervisor	4.0

3.1.3 Related documents

Title	Comments
Mobile Application Diagram	
Smart watch Application Diagram	
Application Class Diagram	

3.2 Requirements

This section describes in detail, each functional requirement of the system, providing a description and scope of each requirement. The process flow for each functional requirement is detailed, including alternate and exceptional process flows, where required. The requirements have been associated with use cases as defined at the beginning of each functional requirement.

3.2.1 User requirements

1. The user requires the ability to register and log into a secure account on the mobile device.
2. The user requires the ability to monitor their heart rate to raise the alarm in the event of an emergency event.
3. The user requires the ability to view historic heart rate activity data in a graph or chart.
4. The user requires the service to perform the functions without requiring training or technical knowledge.
5. The user requires the service to provide a single, clear and easy to understand user interface.
6. The user requires the interface to indicate the *current status* of the users' heartrate in the main UI.
7. The user requires the mobile application to feature an emergency call feature and an outgoing call feature to a nominated person.
8. The user requires the application to allow for control of electrical devices in the home using cloud services and an IoT development board.

3.2.2 Usability requirements

1. The system shall require configuration only on first use, except when the user chooses to edit the settings in the system.
2. The system shall not require the user to sign in on relaunching the application if a user account has already been set up.
3. The system shall not require navigation through multiple levels when using the primary functions of the service.
4. The system shall present all the primary functions of the service in a single user interface to allow for quick and simple access by the user.
5. The smartwatch shall require no configuration on relaunching the application.
6. The smartwatch application shall present the user with a simplified *status* for their most recent heart rate reading e.g. "Status: Healthy", or "Status: Caution!" etc.
7. The user interface of the mobile application and smartwatch shall contain only minimal text to facilitate quick comprehension of the information on screen.
8. The user interface shall contain only sans serif fonts in sizes that enable easier reading.

3.2.3 Environmental requirements

1. The system shall operate on a mobile device running Android with a minimum version of 4.4 to support the APIs required for the smart watch.
2. The mobile device must have access to a cellular network, be capable of making phone calls and must have access either to a WIFI network or must have access to the mobile data network for accessing the cloud database.
3. The smart-watch must be based on the Android *Wear* operating system and must have an onboard optical heart rate sensor.

3.2.4 Non-Functional Requirements

The non-functional requirements define the requirements of the system in terms of its performance, quality, reliability and security.

3.2.4.1 Performance/Response time

The system shall check the users’ heart rate reading no less than once per hour.

The system shall respond to a dangerous reading by initiating an emergency call within 5 seconds of receipt of the data from the smart-watch.

The application shall update the users’ profile data in no more than 10 seconds after an external change to the data, where a network connection is active.

3.2.4.2 Availability

The system shall be available always while the user is wearing the device and within range of the mobile device.

The smart-watch shall queue messages for resubmission where a connection to the mobile device is not available.

3.2.4.3 Recovery

The system shall recover from termination of the application by reloading the existing user account and the latest data from the cloud database.

3.2.4.4 Security

The system shall provide secure user accounts with email recovery functionality

The users account shall be stored in the cloud and no passwords shall be stored on the local device.

The users’ heart rate data shall require authentication before allowing access to the detailed data.

3.2.4.5 Reliability

The system shall not be unavailable to the user for more than 5 minutes per day while the wearable device is active.

The system shall report fatal errors to the developer using the *Firebase* crash reporting functionality.

3.2.4.6 Portability

The system shall allow the user to move the equipment and service to a new location without affecting the performance of the system.

3.2.4.7 Extendibility

The system shall allow the extension of the service to further sensor nodes in the future. This may include additional sensors to compliment the smartwatch or to perform the function of the smart-watch while the watch is not being worn. The system may be extended further to provide a service to an organisation, who could distribute the devices to user/patients, allowing for monitoring of multiple users at once.

3.2.5 Functional Requirements

The following is a numbered list of the functional requirements. The subsequent pages of the report contain a detailed breakdown of each requirement, including its relationship to the use case diagrams, its scope, description and flow description.

Requirement	Description	
1	**Register:**	
		Register a new user account on the mobile device using an email address and password
2	**Login:**	
		Log into an existing user account on the mobile device using email and password
3	**Post profile info:**	
		Input profile information, contact number, address, nominated contact person, Eircode etc
4	**View Profile**	
		View the profile information of a registered user
5	**Read heart rate:**	
		Activate the heart rat sensor and read the current heart rate status on the watch
6	**Update heart rate status:**	
		Update the last known heart rate in the main UI.
7	**Make phone call:**	
		Initiate a call from the main UI to the users' chosen contact.
8	**Emergency call**	
		Automatic emergency call triggered by heart rate
9	**Toggle home:**	
		Switch a smart-home object on or off in the mobile application
10	**View history:**	
		View a history of all heart rate events received from the smart-watch
11	**Night mode:**	
		Toggle night mode so that system expects lower than normal heart rate

3.2.5.1 Requirement 1 – Registration

Requirement 1	**Register**
Use Case	Register new user
Scope	The scope of this use case is to create a new user account using an email address and password.
Description	This use case describes the process of registering a user account on the mobile device. The account will be used to store data about the user's location, medical history, contact person and a log of any triggers of the system.
Flow Description	
	Precondition
	The system is installed on the user's device and there is no currently active login.
	Activation
	This use case starts when the user runs the application for the first time and is not currently signed in, or has previously logged out of the system.
	Main flow
	1. The system checks for registered accounts. 2. The user enters the required fields 3. The system takes the users input and creates a user account 4. The user progresses to the next step in the application
	Alternate flow
	A1: <User already registered> 5. The system recognises the user's details and prompts the user to sign in. 6. The user enters previously registered account details 7. The use case continues at position 4 of the main flow
	Exceptional flow
	A2: <Registration error> 8. The system presents the user with an output message, informing them of the error
	Termination
	The system presents the next step when the user has successfully been logged into the system and the login process is terminated.
	Post condition
	The registration process is complete and will not be evoked again until the user has signed out.

3.2.5.2 Requirement 2 – Login

Requirement 2	**Login**
Use Case	Login
Scope	The scope of this use case is to allow the user to log into the system
Description	This use case describes the process of logging in with a user account on the mobile device. The account will be used to store data about the users' contact person and a log of any triggers of the system.
Flow Description	
	Precondition
	The application is installed on the user's device and the user is not signed. The user has opted to log in instead or registering a new account.
	Activation
	This use case starts when the user runs the application and is not logged in.
	Main flow
	1. The system presents the login interface 2. The user enters their login details 3. The system takes the users' input and checks against the user database. 4. The user progresses to the next step in the application
	Alternate flow
	A1: <User not found> 5. The system cannot find records of the user 6. The user is routed to the Registration use case 7. The use case continues at position 3 of the main flow
	Exceptional flow
	A2: <Login error> 8. The system presents the user with an output message, informing them of the error
	Termination
	This process is terminated when the system detects a successful login and the main user interface is launched.
	Post condition
	The login process is complete and will not be evoked again until the user has been signed out. The user is then presented with the main user interface.

3.2.5.3 Requirement 3 – Post Profile Information

Requirement 3	**Post Profile Info**
Use Case	Post Profile Information
Scope	The scope of this use case is to allow the user to input detailed profile information after the user has registered an account in the system.
Description	This use case describes the process of accepting user input and saving to the database for use in the users' profile.
Flow Description	
	Precondition
	The user is authenticated and has been directed to this process by the signup process. The user has previously entered user data and has chosen to edit the information.
	Activation
	This process begins when the user registration process has been completed or when the user has chosen to edit their profile information.
	Main flow
	1. The user successfully passes the registration process 2. The system presents the user with the data entry activity. 3. The user enters the required data and presses the save button. 4. The system saves the user input to their profile and progresses to the next activity
	Alternate flow
	A1: <Field not filled> 5. The user has not filled all fields in the activity 6. The system checks the user input and highlights the error to the user. 7. The user inputs the missing data and presses the save button
	Exceptional flow
	A2: <Login error> 8. The system is unable to save the user entered data and reverts to default data. 9. The system informs the user of the error.
	Termination
	This process ends when the user has filled in all required fields with valid data and has clicked the save button.
	Post condition
	The Profile Info process is complete and the user is progressed to the next activity.

3.2.5.4 Requirement 4 – View Profile Information

Requirement 4	**View Profile Info**
Use Case	View Profile Info
Scope	The scope of this use case is to allow an authenticated user to view the user profile.
Description	This use case describes the process of accepting user input and saving to the database for use in the users' profile.
Flow Description	
	Precondition
	The user is authenticated and has previously added profile information.
	Activation
	The user has selected the profile option from the navigation menu
	Main flow
	1. The user chooses to open the profile page 2. The system checks the user is authenticated and loads their data from the database 3. The user is presented with the profile overview screen
	Alternate flow
	A1: <No Profile Data> 4. The user chooses to open the profile page 5. The system finds no profile information for the user 6. The user is prompted to add profile data
	Exceptional flow
	A2: <Login error> 7. The system detects the user is not authenticated or has expired and loads the login activity 8. The user logs into the system
	Termination
	This process ends when the user opts to close the profile overview screen.
	Post condition
	The user is directed back to the main activity

3.2.5.5 Requirement 5 – Read heart rate

Requirement 5	**Read heart rate status**
Use Case	Read heart rate
Scope	The scope of this user case is to initiate a scanning of the users' heart rate by the smart-watch and to transmit the heart rate to the mobile device.
Description	This use case describes the process of taking a heart rate reading from the user.
Flow Description	
	Precondition
	The user is signed into the application on the mobile device and is wearing the device and has completed the initial setup steps.
	Activation
	This use case starts when the smart-watch background service calls the heart rate service at a set interval of once every hour.
	Main flow
	1. The smart-watch service detects that a heart rate reading is due to be taken. 2. The service begins the SensorManager process to get the reading 3. The SensorManager returns the reading response 4. The background service packages the heart rate in a DataItem and sends to the mobile device. 5. The mobile device receives the DataItem and processes the data within. 6. The mobile device checks the heart rate is within safe parameters 7. The main UI is updated with the status of the users' heart rate
	Alternate flow
	A1: <Not connected to the mobile device> 8. The smart-watch detects that it is not connected to the mobile device for transmission of the readings. 9. The readings are queued for transmission when the mobile device reconnects to the watch.
	Exceptional flow
	A2: <Error> 10. The system presents the user with an output message, informing them of the error and logs the error to the developer console.
	Termination
	This process ends when the smart-watch has successfully transmitted the heart rate reading to the mobile device.
	Post condition
	The background service closes the SensorManager connection and resumes sleeping.

3.2.5.6 Requirement 6 – Update heart rate status

Requirement 6	**Update heart rate info**
Use Case	Update heart rate info
Scope	The scope of this use case is to display the current heart rate status of the user on the main user interface of the mobile application.
Description	This use case describes the process of outputting the status of the users' latest heart rate reading in the main UI of the application.
Flow Description	
	Precondition
	The user is authenticated and the smart-watch is connected to the mobile device
	Activation
	The smart-watch sends heart rate data to the mobile device
	Main flow
	1. The mobile device receives the transmission and unpacks the data 2. The application checks the status against the users' heart rate parameters from the user profile. 3. The heart rate is broadcast to the MainActivity for output to the user. 4. The MainActivity rates the heart rate reading on the 3-position gauge.
	Alternate flow
	A1: <Heart rate dangerous> 5. The system detects that the users' heart rate is outside of the acceptable parameters set by the user. 6. The system triggers an emergency and starts a countdown to an emergency call. 7. The user does not cancel the countdown 8. The system initiates the call
	Exceptional flow
	A2: <Error> 9. The user cancels the countdown. 10. The system ends the activity and outputs the heart rate reading in the main UI.
	Termination
	The process terminates when the users' heart rate has been displayed in the MainActivity.
	Post condition
	The background service resumes sleeping and the heart rate is displayed to the user.

3.2.5.7 Requirement 7 – Phone call

Requirement 7	**Phone call**
Use Case	Phone call
Scope	The scope of this use case is to allow the user to initiate a call to a predefined contact in a single step, or for the mobile application to initiate a call if an emergency has been detected.
Description	This use case describes the process of initiating a phone call by the user or by the system when either the user has initiated the call through the UI, or the system detected a possible emergency through the heart rate sensor.
Flow Description	
	Precondition
	The user is signed into the application and has added a contact or emergency number.
	Activation
	This use case starts when the user presses the Call button, or the system detects an emergency.
	Main flow
	1. The system presents the user with the main UI with call buttons 2. The user presses the call button, for their contact or the emergency services 3. The application initiates the call for whichever option was selected by the user 4. The user can now interact with the recipient.
	Alternate flow
	A1: <Emergency Call> 5. The smartwatch detects an emergency and notifies the mobile application 6. The mobile application initiates a call to the emergency number and places the phone on loud speaker. 7. The system starts audio description of the users' location 8. The use case continues at position 4 of the main flow
	Exceptional flow
	A2: <Error> 9. The system presents the user with an output message, informing them of the error
	Termination
	The process is terminated after successful initiation of a phone call.
	Post condition
	The process will not be evoked again until either the user initiates the call from the UI or the system detects an emergency trigger.

3.2.5.8 Requirement 8 - Emergency call

Requirement 8	**Emergency call**
Use Case	Emergency call
Scope	The scope of this use case is to notify the user when an automated call to the emergency services is being initiated and all them to interrupt if required. The scope of this use case is to gather the medical information of the user and provide reminders through the smart watch.
Description	This use case describes the process of displaying a notification to the user when a phone call has been initiated automatically and to allow the user to interrupt the call before it is made.
Flow Description	
	Precondition
	The user is signed into system and is wearing the mobile device.
	Activation
	This use case starts when the mobile application initiates an automated call to the emergency services
	Main flow
	1. The mobile application initiates a phone call process and opens a connection to the smartwatch 2. The smartwatch receives the notification of intention to make a call and prompts the user to react 3. The user fails to act on the notification within a given timeframe 4. The mobile application makes the call
	Alternate flow
	A1: <User Interrupted Call> 5. The user interrupted the intent to make a call through the smartwatch UI 6. The smartwatch responds to the mobile application with a cancel notice 7. The mobile application cancels the call process and displays a notification of cancelation
	Exceptional flow
	A2: <Error> 8. The system presents the user with an output message, informing them of the error
	Termination
	The process is terminated after the user cancels the intent to initiate a call, or the user fails to respond within the timeframe.
	Post condition
	The process is terminated and will not be evoked again until it is initiated by an automated call from the mobile application.

3.2.5.9 Requirement 9 – Toggle home

Requirement 9	**Toggle home**
Use Case	Toggle home
Scope	The scope of this use case is to actuate the relays when the user interacts with the smart-home switches in the mobile application.
Description	This use case describes the process of actuating the relays when the user operates the switches in the mobile application. The user interaction is recorded and saved to the cloud database. The Raspberry Pi development board runs a Python script that is listening to the cloud database for changes. These control values are read into the python script and processed by the control methods to interact with the relays.
Flow Description	
	Precondition
	The user is logged in and has set up the Raspberry Pi device in the home.
	Activation
	This use case starts when the user navigates to the Smart-home activity in the mobile app.
	Main flow
	1. The user starts the smart-home activity. 2. The mobile app loads the switches and the state from the database. 3. The user interacts with the switches. 4. The mobile application updates the control values in the cloud database. 5. The Python script reads the values and processes them to the control methods.
	Alternate flow
	A1: <Raspberry Pi not online> 6. The Raspberry Pi is not connected and cannot update the control values. 7. The mobile app reads the database values and resets the toggle switch. 8. The user sees the switch state displayed in the UI. 9. The user connects the raspberry Pi device to the internet and proceeds
	Exceptional flow
	A2: <Error> 10. The system presents the user with an output message, informing them of the error
	Termination
	The process is terminated after the relay is actuated and the switch state updated.
	Post condition
	The process is terminated and will not be evoked again until the user interacts with the smart-home switches again.

3.2.5.10 Requirement 10 – View history

Requirement 10	**View history**
Use Case	View history
Scope	The scope of this use case is to allow the user to view the historic heart rate data collected by the system in a dedicated activity on the mobile device, using lists and graphs.
Description	This use case describes the process of starting the history activity, loading the historic data and outputting it to the user in list or graph form.
Flow Description	
	Precondition
	The user is logged in on the mobile application and has provided their fingerprint.
	Activation
	This use case starts when the user navigates to "History" activity in the mobile app.
	Main flow
	1. The user starts the History activity 2. The mobile application queries he database for historic data 3. The database response is parsed and output to the user in lists and graphs. 4. The user is presented with the data and can interact with the graphs.
	Alternate flow
	A1: <User not authenticated> 5. The user selects the History activity but fails to provide a fingerprint. 6. The system informs the user that authentication is required. 7. The user provides a fingerprint and is taken to the history activity. 8. The application outputs the users' history data
	Exceptional flow
	A2: <Error> 9. The system presents the user with an output message, informing them of the error
	Termination
	The process is terminated after the user navigates to another activity in the system or presses the back button.
	Post condition
	The process is terminated and will not be evoked again until the user navigates to the history activity again.

3.2.5.11 Requirement 11 – Night-mode

Requirement 11	**Night mode**
Use Case	Night mode
Scope	The scope of this use case is to allow the user to activate a night mode feature. This feature enables the system to know that the user is sleeping and that their heart rate is expected to lower.
Description	This use case describes the process of toggling the night mode feature. This informs the system that the user expects their heart rate to slow, as is normal during sleep.
Flow Description	
	Precondition
	The user is logged in on the mobile device and is presented with the main UI.
	Activation
	This use case starts when the user interacts with the night mode toggle switch on the main UI.
	Main flow
	1. The user interacts with the toggle switch. 2. The system reads the Boolean value of the switch and inverts it before sending back to the database. 3. The user is presented with the switch in the new state. 4. The background service reads the new state before acting on triggers and allows a deviation from the lower limit.
	Alternate flow
	A1: <Rate below limit> 5. The background service detects night mode is enabled and allows a percentage deviation from the lower limit. 6. The watch sends a heart rate reading that is below the night mode deviation 7. The background service disregards the night mode setting and triggers an event. 8. The user is notified of the trigger and the process for an emergency call begins.
	Exceptional flow
	A2: <Error> 9. The system presents the user with an output message, informing them of the error
	Termination
	This process is terminated when the night mode switch displays the new switch state to the user in the main UI.
	Post condition
	The new state is represented to the user in the main UI. This process is terminated and will not be evoked again unless the user interacts with the night mode switch.

3.2.6 Data Requirements and Design

The service is built on top a *Firebase* database [11], a cloud based object oriented NOSQL database provided by *Google*. The application requires that the user can register and sign into a secure account, to record profile data including their name, address, contact details and a nominated point of contact. The database shall also persist all events that are recorded by the smart-watch application and associate them with a timestamp of when they occurred.
The database consists of separated authenticated user accounts and a JSON database that are linked to provide dynamic, secure data storage for each user, based on their uniquely generated user ID. The database is accessed using *ValueEventListeners* in *Java* and an instance of the database is stored on the local device to provide fault tolerance during network downtime, without affecting the service.

The local instance and cloud database interact using *dataSnapShots*, an instance of the newly changed data that is sent from one to the other. This ensures synchronicity of the data in the system while providing an efficient, low latency solution to data consistency.

3.2.6.1 Required data

Name, email, phone number, address, Eircode, nominated contact, contact's phone number, contacts address, heart rate upper and lower limits, heart rate events including timestamp and heart rate itself, smart-home switch state.

3.2.6.2 Database rules

The database rule set enables *Firebase* to provide secure and private storage for each user by dynamically generating a database instance, based on their unique user identification value. The "*$uid*" property is passed into the rules declaration in *Firebase* and read and write privileges are set to allow the user to access only those nodes in the database tree that match their unique ID. The following example shows how the rules accept the *uid* parameter and limit the user to seeing only data relevant to their account.

```
"rules": {
  "users": {
    "$uid": {
      ".read": "$uid === auth.uid",
      ".write": "$uid === auth.uid"
    }
  }
}
```

3.2.6.3 Backup and recovery

Firebase database provides a seamless integrated data backup solution in its paid subscription. Alternatively, the database can be duplicated and backed up to external devices quickly and simply by accessing the database URL. Additionally, the developer can export the entire database in JSON format through the *Firebase* Developer Console. These backups can also be imported into the console if a recovery is necessary.

3.2.6.4 Data limits

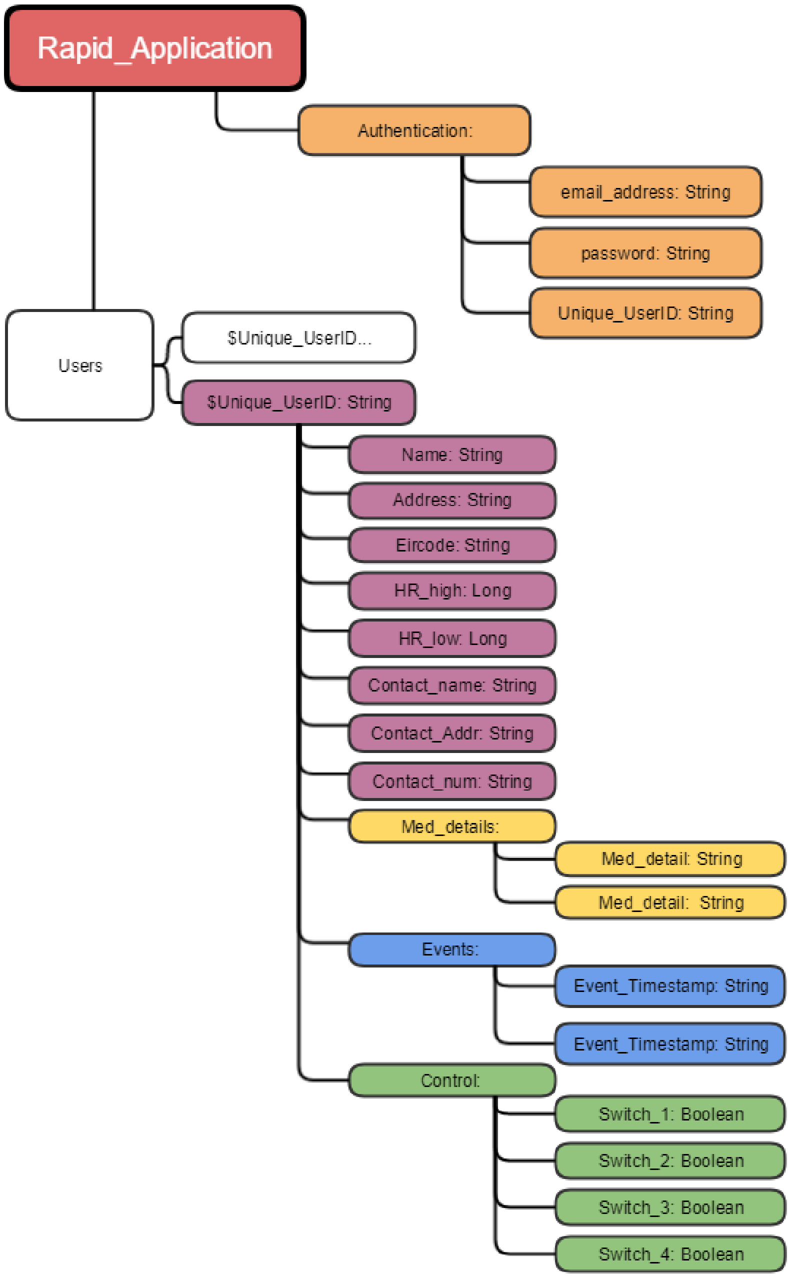
The database can handle up to 100'000 simultaneous connections using the free tier. Additional resources are available through the paid tier.

3.2.6.5 Performance monitoring

Using *Firebase Performance Monitoring*, detailed data can be gathered to analyse user interaction with the application. This data enables the developer to identify performance issues within he application. These features can be used to measure performance down to individual activities within the application, measure the start-up time and the time it takes to obtain required data from the cloud.

3.2.6.6 Logical Data Model

The diagram below is a representation of the NoSQL database design and shows the separate authentication and data storage nodes of the tree. These nodes are then linked by the unique user identifier that is generated during registration, as shown in he "*Database rules*" section. This ensures that data can only be accessed within the system by a user who has a matching authentication token to that of the data itself.

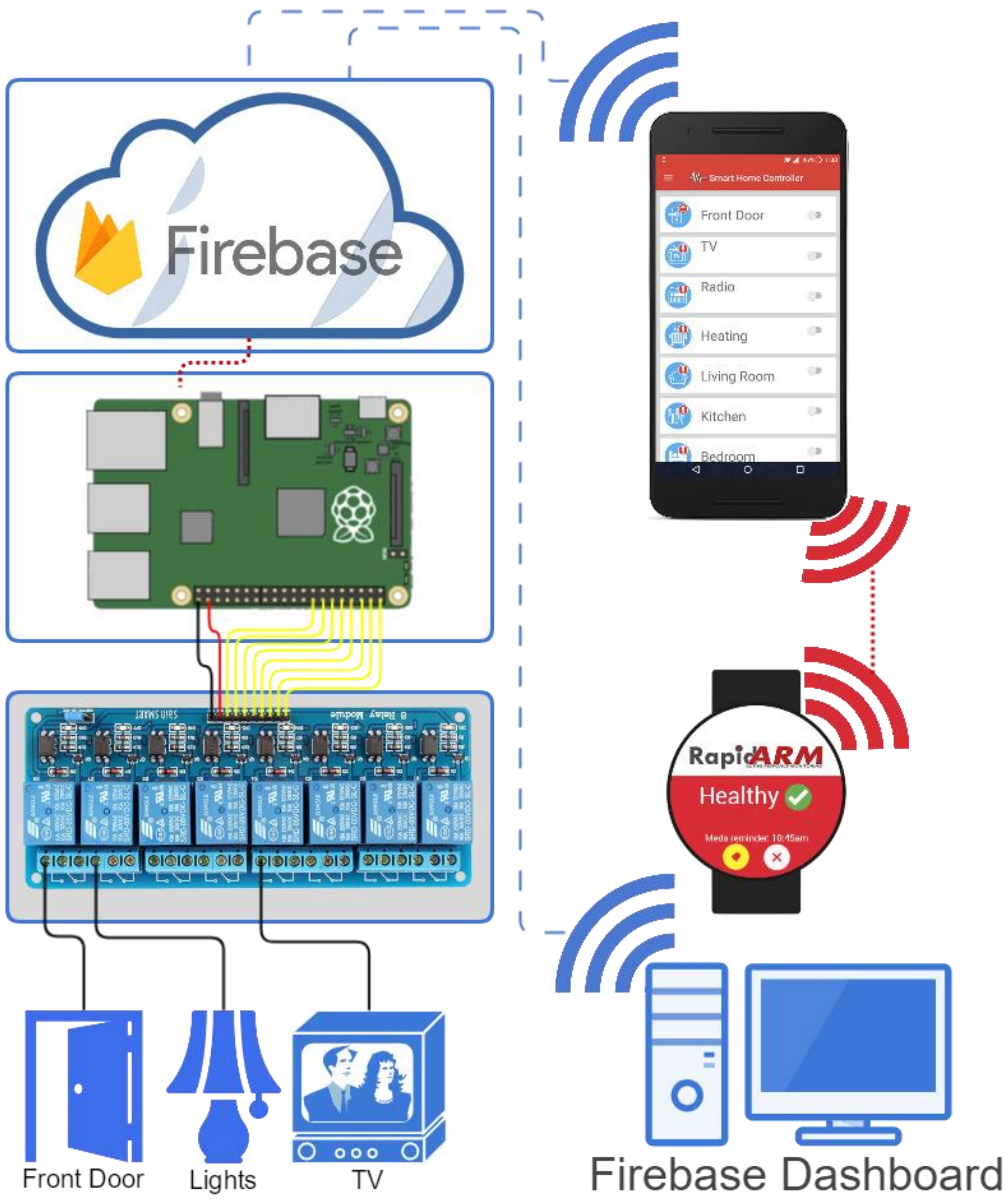


3.3 Design and Architecture

This section provides a detailed insight into the design and architecture of the *RapidARM* service and includes architecture, sequence, class and communication diagrams. Further details on each diagram is contained within each heading, where necessary.

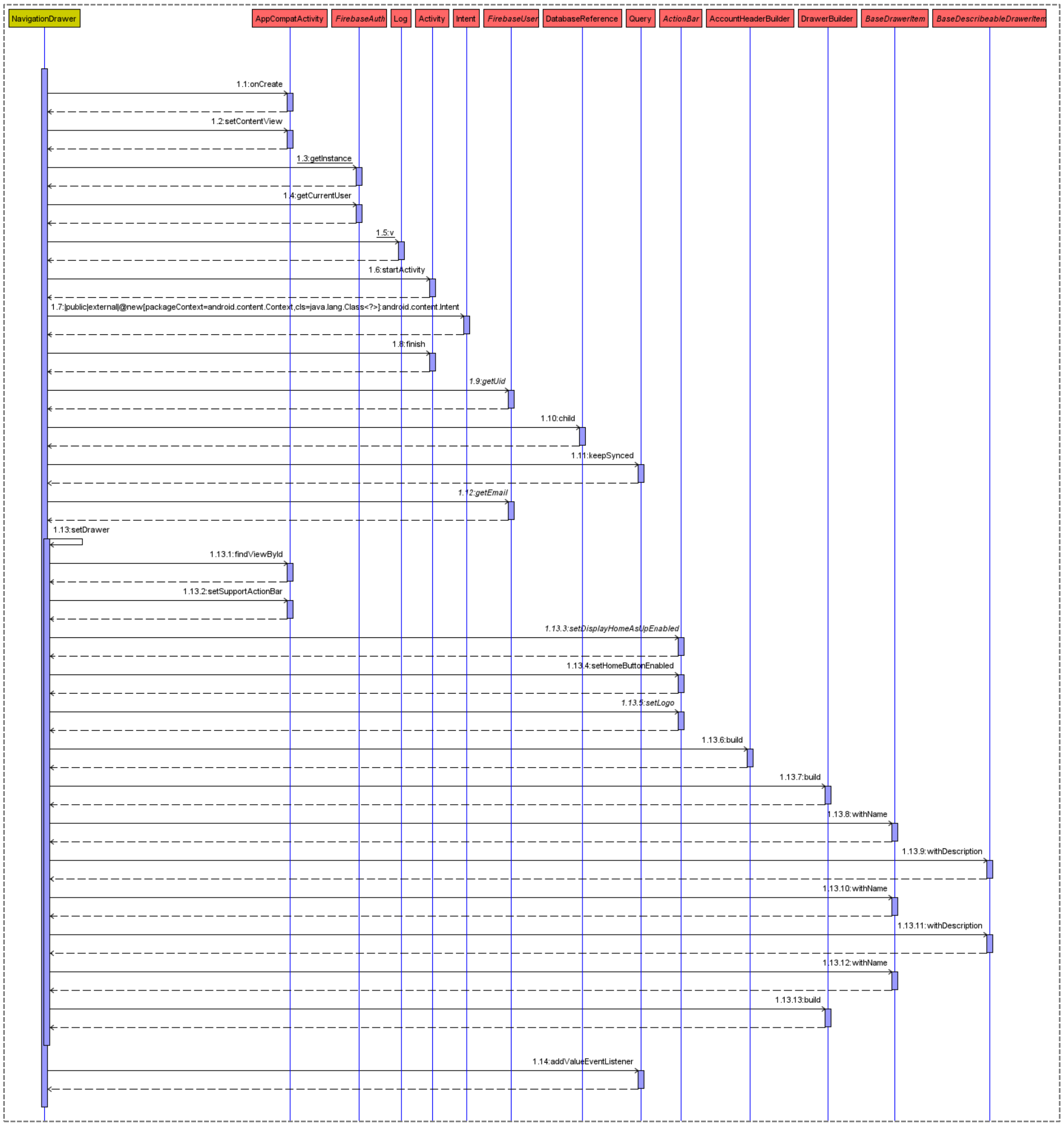
3.3.1 Architecture Diagram

The below architecture diagram shows how the mobile application interacts with the *Raspberry Pi* through the *Firebase* cloud service to control various elentrical items in the home, including the lights, TV and remotely unlocking the door.



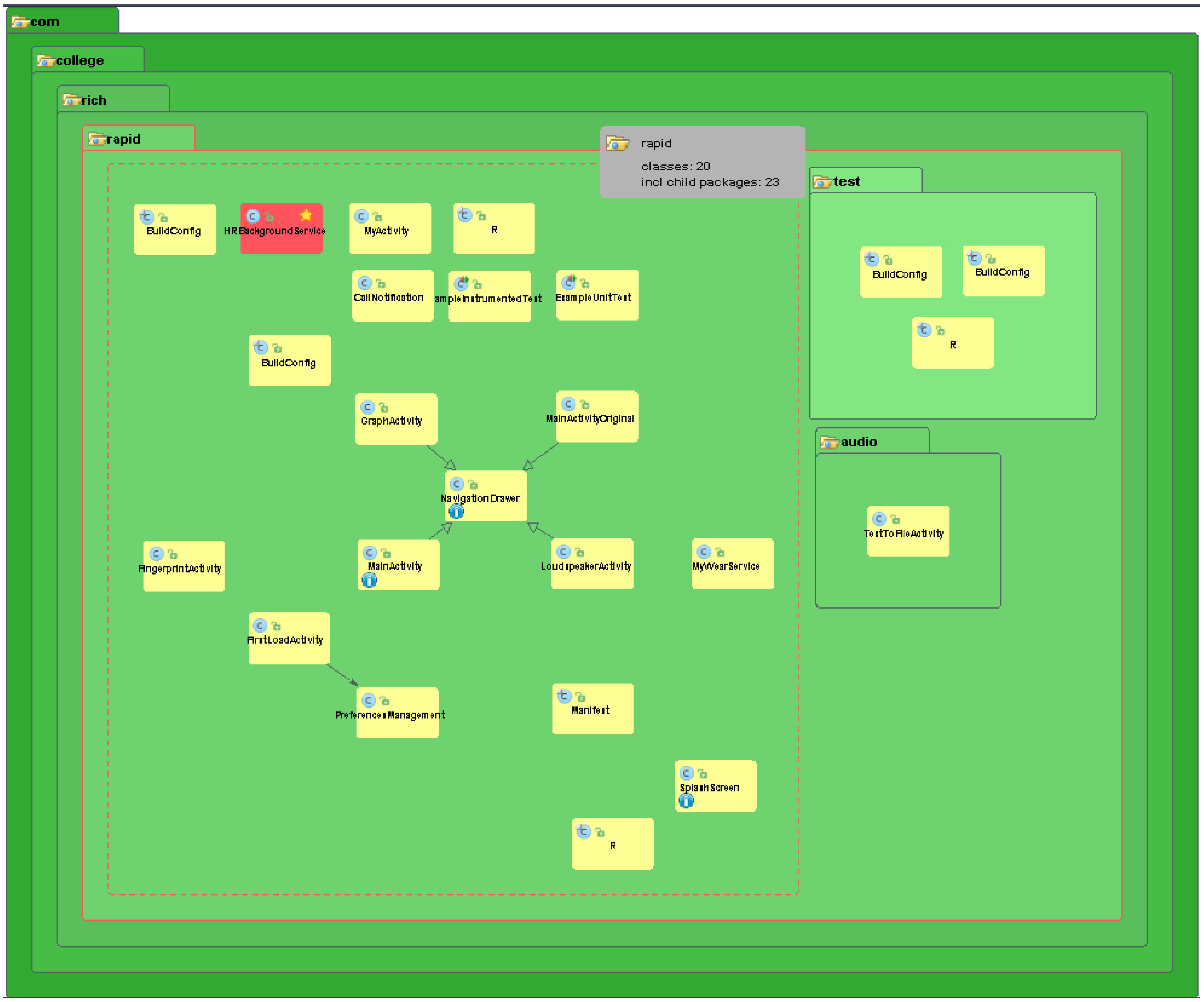
3.3.2 Sequence Diagram

The sequence diagram is produced using Android Studio and the "*SequenceDiagramReload"* plugin. This diagram shows the sequence of events within the application from initial load of the MainActivity when the "*SplashScreen*" activity has finished.



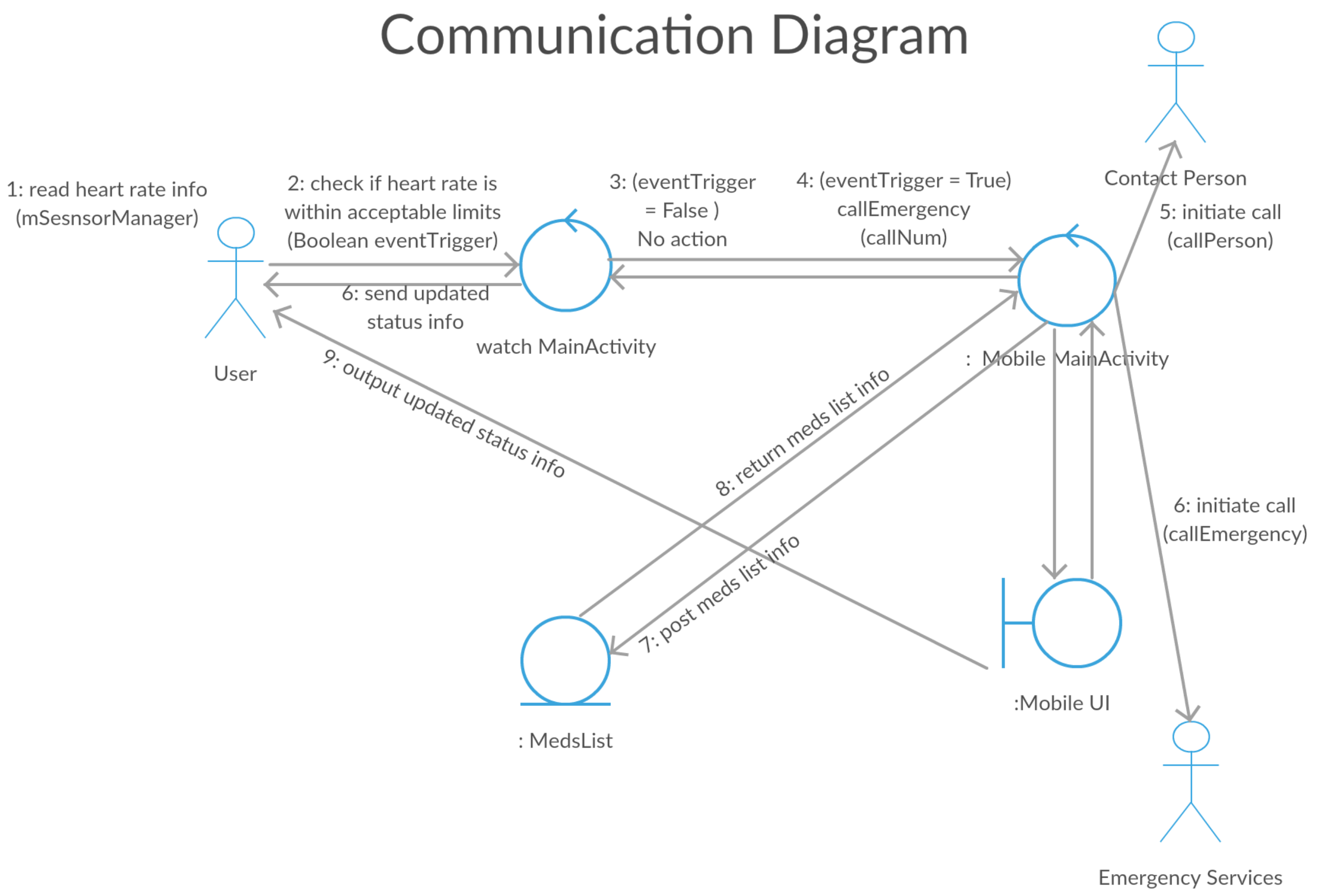
3.3.3 Class Diagram – Mobile Application

This class diagram was generated using the *Code Iris* plugin with Android Studio 2.3.3. This diagram shows the classes and packages within the *RapidARM* project. Implemented libraries and API's are not represented in this diagram.



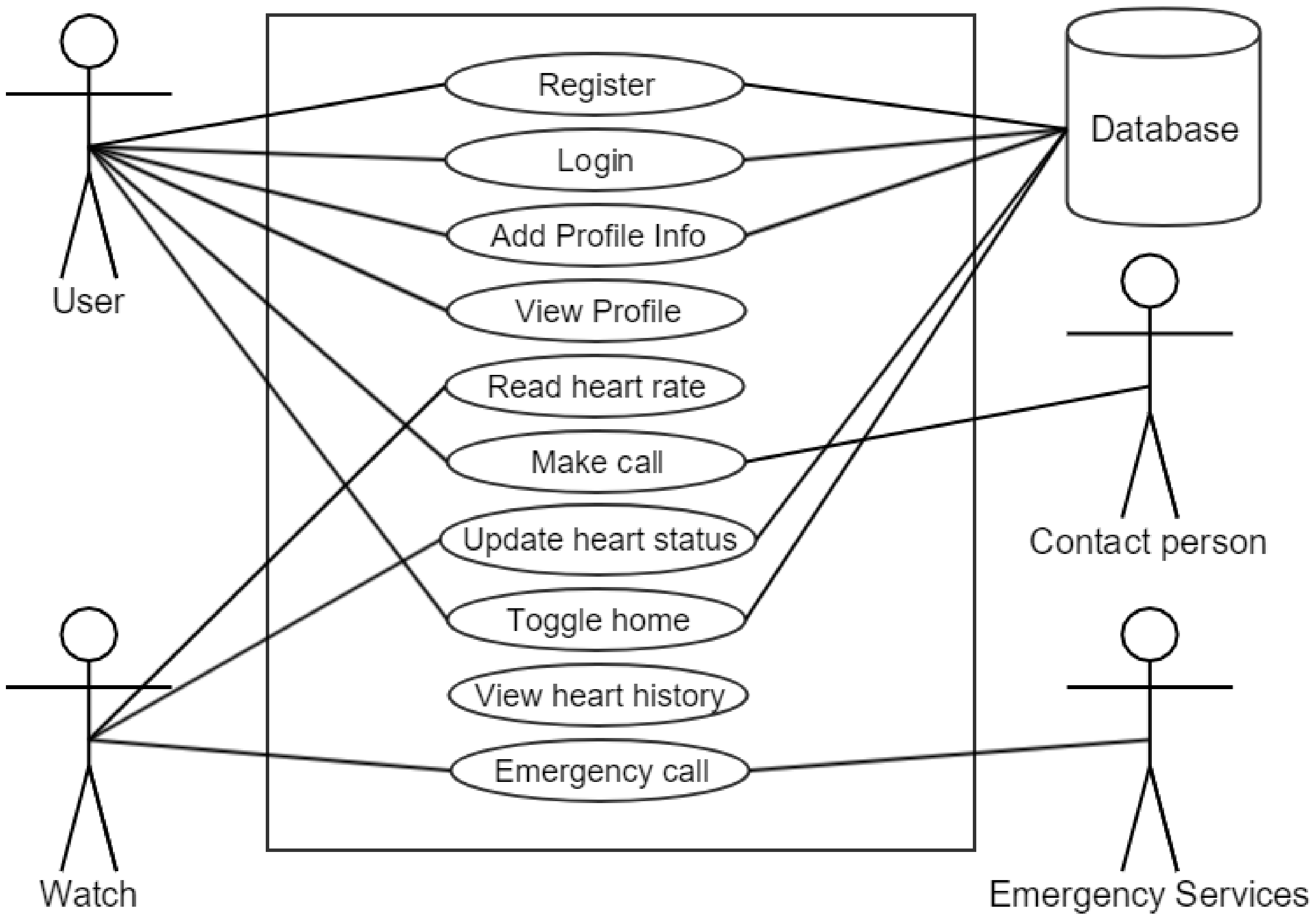
3.3.4 Communication Diagram

The communication diagram gives a conceptual overview of the flow of events within the system during operation. Each state shows the flow direction to another process or to an actor.

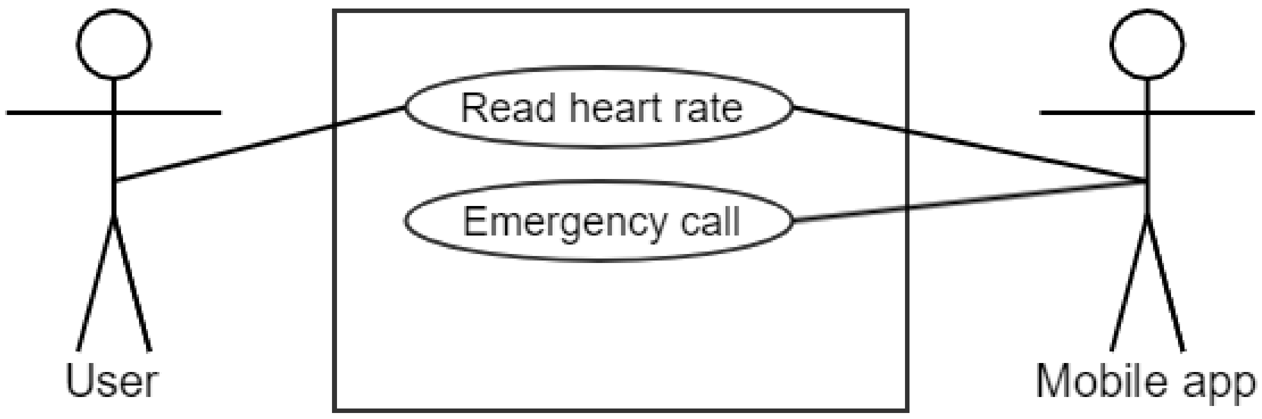


Communication Diagram
1: read heart rate info (mSesnsorManager)
2: check if heart rate is within acceptable limits (Boolean eventTrigger)
3: (eventTrigger = False) No action
4: (eventTrigger = True) callEmergency (callNum)
Contact Person
5: initiate call (callPerson)
User
6: send updated status info
watch MainActivity
: Mobile MainActivity
9: output updated status info
8: return meds list info
7: post meds list info
6: initiate call (callEmergency)
: MedsList
:Mobile UI
Emergency Services

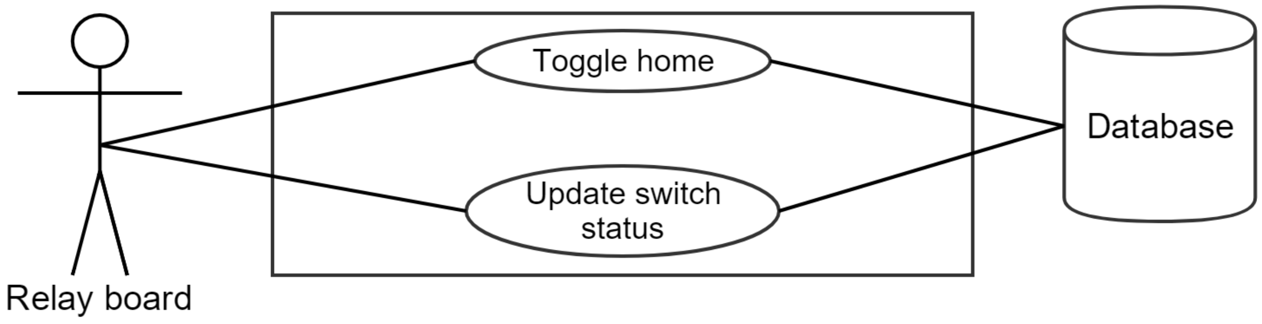
3.3.5 Mobile Application Use-case Diagram



3.3.6 Smart Watch Application Use-case Diagram



3.3.7 Raspberry Pi Application Use-case Diagram



3.4 *Implementation*

This section of the document provides a detailed breakdown of important technologies and how they are implemented in the system. Code snippets are provided to explain how some problems were solved. These code snippets have been simplified with some boilerplate code removed to make the examples more clear and concise.

3.4.1 Firebase – Realtime database console

Firebase was selected as the chosen database solution for the application, due to its excellent support across many mobile operating systems, web and IoT devices. *Firebase* is a JSON based cloud hosted object oriented database that consists of a tree-like structure, containing key-value pairs [11]. The database retains an instance on the local device that ensures that the service is available even when network connection is interrupted. When changes to the data occur either in the cloud or on the device, a notification of change is sent alongside a "*dataSnapShot*". This snapshot is essentially the parent node of the object that had been changed in the tree and is used to overwrite only the targeted set of data. The benefit to this method comes in the speed at which data is available and updated within the system. Instead of loading an entire record of an individual, which would include all their profile data and list of events in the system, *Firebase* can pass back and forward individual nodes within their profile to avoid loading in data that is unnecessary in the current context.

By using a key-value database like this, the developer avoids the need to write complex and potentially buggy code to convert application data into a relational form for storage on the device or online. This structure allows greater freedom to alter the database and its data types quickly and easily, due to its schema-less design. Additionally, the availability of the database through a simple URL in the *Firebase* Developer Console means that opportunities exist to take the generated data to build dashboards or analytics tools on top, using easily available charting and graphing tools such as *HighCharts* and *Tableau* that are already designed to consume key-value data.

3.4.2 JSON – Lightweight data format

JSON is used extensively throughout all areas of the application, as it is a streamlined, efficient and human readable format for the data that is easily parsed, while containing only the required data and no unnecessary characters.

The smart-watch reads data from the heart-rate sensor and packages this reading along with the timestamp into a JSON object that is then transmitted to the mobile application in the form of a "*DataItem*".

Throughout various sections of the mobile application, JSON is used in reading or writing to *Firebase* where only a small sub section of the database needs to be updated. This includes using *Map* objects to package multiple *key-value* pairs and dump them in one go, or individual map methods to put keys and values into single nodes of the database without other nodes in the database.

The *Raspberry Pi* also uses JSON when parsing the response from *Firebase* with the control information needed for providing the smart-home functionality. This data is then iterated over to extract the individual switch commands and their *True* or *False* values, before being passed to specific functions for actuation.

3.4.3 Bluetooth – Device communication

The Android Wear device is connected to the mobile application using *Bluetooth Low Energy* to extend the battery life of the watch as much as is possible [4]. The communication between the watch and mobile application is conducted using "*DataItems"*. These DataItems are very small streams of JSON data that are limited in size to ensure that poorly constructed messages cannot hold open a communication channel unnecessarily, causing excessive power consumption. Larger messages can be sent by serializing multiple messages, if needed.

3.4.4 Android Mobile Development

The project is largely developed on the Android platform, using Android Studio and Java. The mobile application is written in Java with the UI written in XML.

3.4.4.1 Firebase Android API – Cloud Database

Database functions within the application have been implemented using Firebase. Firebase in a cloud database solution for mobile and web. This database also includes a local instance on the mobile device that is used to communicate data changes back and forth between the cloud and the mobile application. The database is based on JSON and does not require a schema. A *DatabaseReference* is created in the application and this reference is a pointer to the highest-level node in the database tree for which the user has access and can also be set to lower levels within the tree, when it is necessary to retrieve individual values. Data is retrieved from the server using an *eventListener* and the resulting response is a *dataSnapShot* object.

This *dataSnapShot* is a copy of the database from the parent node of the changed data, meaning the application does not need to load in all the data in its entirety to read or write to specific locations. The developer can program the snapshot to synchronise the entire database, or smaller sub-sets of the data contained within. When the application receives a new *dataSnapShot* object, it is then used to create a new *Map* object, containing the key-value pairs within. The values can be fetched from the Map object and assigned to a local variable, as shown in the code snippet below.

```
mDb.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        Map<String, String> myMap = (Map<>) dataSnapshot.getValue();

        mName = myMap.get("name");
        contactNum = myMap.get("contact_num");
    }
    @Override
    public void onCancelled(DatabaseError databaseError) {
    //Handle error here:
    }
});
```

The above example shows how a users' name is obtained using a *valueEventListener* and how the response from Firebase is processed into meaningful data by the application. This approach obtains an object containing multiple pieces of data and the "*.get()*" method is used to obtain a specific piece of data from within. A alternative approach can be used to obtain single values from the database without the need to load all the above data, as shown in the below method for obtaining only a "name".

```
mDb.child("name").addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot snapshot) {
        name = (String) snapshot.getValue();
    }
});
```

3.4.4.2 Splash Screen – Loading Animated Screen

An animated splash screen is used on loading the application. This uses the *RapidARM* logo and performs an animation for 3 seconds, during which the application is establishing a connection to the database and checking whether this is the first run of the application and if the user is authenticated or not.

On finishing this animation, the "*SplashScreen"* Activity loads the "*RegisterActivity*" when it is detected that the application had not previously been run. The *PreferencesManagement* class is used to identify if the application had previously run. If the application had previously been run, the activity evokes the *MainActivity* class instead, where it will then attempt to load the main UI of the application. If the user has previously signed in and a valid auth token is present, the main UI will be displayed.

If a user auth token was not found when the "*MainActivity"* was launched, the user is directed to the *LoginActivity*.

3.4.4.3 Intro Cards – Overview Guide

On first loading of the application, the user is prompted to *register* and *sign-in* [17]. When the user is authenticated, they are presented with a *FirstLoadActivity* containing slider cards to guide them through the application and explain its function. These cards are designed to be simple so as not to be overly confusing, while explaining the applications functions through images and short descriptions in a UI, requiring only swipes to navigate through it.

3.4.4.4 MaterialDrawer - Material Design Navigation

Material design has been used throughout the application to create a seamless and intuitive experience for the user. This *Google* design philosophy aims to provide a unified experience across all platforms [14] and to provide a natural and fluent navigation through an application. This is done with the use of slider menus, dynamic animated layouts, shadowing and the use of responsive images and layouts that are changed dynamically, based on the DPI of the viewers screen.

RapidARM uses the *MaterialDrawer* library as a basis of all navigation throughout the application [16]. This library allows the developer to produce sliding navigation menus and can include icons and account information within the slider.

This was done by first producing test activity with the navigation drawer, before writing the code necessary for each subsequent activity that the drawer should navigate to. This led to code repetition, as each activity that should have the navigation drawer would need to build it over again. The solution to this was to develop a generic *NavigationDrawer* class in which all the navigation functions where contained. This included writing methods to obtain data from Firebase to populate an account header with the name and email of the current user.

This generic class was then implemented by all further activities that required navigation so that any changes that were necessary in the navigation menu where programmed only once before being inherited by all sub-class activities.

3.4.4.5 Background Service – Receiving data from the smart-watch

When the smart-watch is connected, it will attempt to transmit data as soon as it is captured by the heart rate sensor. To allow this to happen reliably, it was required to develop a background service to receive this data. Using a background service meant that the application does not need to be open to receive the incoming message, or the user can be using other activities within the application without interfering with the core functionality.

The background service uses a "*DataEventsReceiver*" method to receive transmissions from the smart-watch. The resulting "*dataEvent*" object is traversed to obtain the timestamp and heart rate values from within.

This background service also retrieves required data from the Firebase database, such as the heart rate limits for the user, emergency phone number etc, and uses this data to compare the incoming values against. The values are passed to a method that checks if the user's heart rate requires an action. If the service detects the heart rate is at dangerous levels, a call to the emergency services is triggered, otherwise the gauge and heart rate in the main UI is updated to reflect this new event.

Background services are not capable of directly interacting with the UI, so the transmission of the data to the main UI is done using a "*BroadcastManager*". A new *Intent* is declared and the heart rate details are passed into this intent. The intent is then broadcast throughout application using the "*BroadcastManager*" to any other activity that is listening with a "*BroadcastReceiver*" for the incoming data.

The following code snippet shows how a broadcast is created. The previously obtained heart rate is passed into this method alongside a message string and an intent is created, before being passed to the broadcaster object for transmission.

```
public void sendBroadcast(float rate, String message) {
    Intent intent = new Intent(rate);
    if (message != null)
        intent.putExtra(HR_MESSAGE, message);
    broadcaster.sendBroadcast(intent);
}
```

3.4.4.6 Recent Activity Overview – Broadcast Receiver

On loading the main UI, the user is presented with their most recent heartrate and a 3-position gauge, displaying how that reading is categorised into either of the three segments for "*Safe*" (Green), "*Warning*" (Orange) and "*Emergency*" (Red). This gauge is updated using a "*BroadcastReceiver*" to obtain new data from a background service [2]. This broadcast is evoked whenever the smart-watch records new data and sends it to the mobile device, resulting in the gauge updating in real-time. The code snippet below shows the broadcast receiver method in the *MainActivity*. The incoming intent is cast to a local variable and outputted to the main UI *TextView*. The reading is then fed into the "*updateGauge*" method for ranking the reading on the 3-position gauge.

```
receiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        int rateOutput = intent.getStringExtra(MyWearService.HR_MESSAGE);

        Log.d(TAG, "<<<RECEIVED from Service:>>> " +rateOutput);
        bpmTv.setText(rateOutput);        //Output latest rate in textview

        updateGauge(rateOutput);          //Update gauge with latest heart rate
    }
};
```

3.4.4.7 Heart rate gauge – Updating in real-time

When data has been received from the background service, it then passed into a method to update the gauge in the *MainActivity*. This happens every time new data is received. In the example below, the *latest* variable is the latest heart rate reading and the *rateLow* and *rateHigh* variables are the upper and lower limits as set by the user and pulled from the Firebase database.

```
public void updateGauge(long latest) {
    if (latest >= rateLow && latest <= rateHigh) {
        if (latest - rateLow > 10 && rateHigh - latest > 10) {
            statusIcon.setImage(R.drawable.gauge_green);
        } else {
            statusIcon.setImage(R.drawable.gauge_yellow);
        }
    } else if (latest > rateHigh || latest <= rateLow && latest != 0) {
        statusIcon.setImage(R.drawable.gauge_red);
    } else {
        statusIcon.setImage(R.drawable.error_small);
    }
}
```

3.4.4.8 Visualising & Graphing Data - MPAndroidChart

The application provides the ability to display all historic heart rate data that the service has received from the smart-watch and to view it in text or graph form [12]. The *MPAndroidChart* library is a set of tools to enabled the developer to build various chart types and to populate them with key-value data.

These charts have been implemented within the *History* section of the application to display both the most recent data points and the list of all data points available. A line chart shows only the last 10 entries so that the view does not become overpopulated with data, making it difficult to interpret. A second chart features all the data points available in a bar-chart that allows the user to scroll and to zoom in on areas within the data. The following snippet shows how the line data chart is built.

```
public void buildGraphs() {
    ArrayList<Entry> yVals = new ArrayList<>();
    final Map<String, Float> treeMap = new TreeMap<>(myMap);
    Set set = treeMap.entrySet();
    Iterator it = set.iterator();
    Date myDate = null;
    SimpleDateFormat dateFormat = new SimpleDateFormat("dd:MM:yy:HH:mm");
    int counter = 0;
    String timestamp;

    while (it.hasNext()) {
        Map.Entry entry = (Map.Entry) it.next();
        timestamp = (String) entry.getKey();
        Double tmpHR = (Double) entry.getValue();
        Float tmpHR2 = tmpHR.floatValue();
        myDate = dateFormat.parse(timestamp);
        yVals.add(new Entry(counter, tmpHR2));
        counter = counter + 1;
    }
    LineDataSet dataSet = new LineDataSet(yVals, "Heart Activity Data");
    dataSet.setMode(LineDataSet.Mode.STEPPED);
    LineData newData = new LineData(dataSet);

    lineChart.animateXY(2000, 2000);
    lineChart.getXAxis().setDrawGridLines(false);
    lineChart.getXAxis().setAxisMaximum(10);
    lineChart.setData(newData);
}
```

3.4.4.9 Fingerprint Authentication

The Fingerprint API is used in the mobile application to verify that the users' historical heart rate data is not freely available to anyone who can access the device[2], [13]. This API enlists the operating systems encrypted fingerprint verification technology and prompts the OS to inform the application whether a user is authenticated or not. The sensitive fingerprint data is never accessible to developers or to the application, by design, and instead the API acts as a gatekeeper to identify if the current users' print matches the registered owner of the device. The below snippet shows how responses from the fingerprint handler are processed to either progress the user through the application, or to notify them of a failed login.

```
@Override
public void onAuthenticationFailed() {
    this.update("Fingerprint Authentication failed.", false);
}

@Override
public void onAuthenticationSucceeded(FingerprintManager.AuthResult result) {
    ((Activity) context).finish();
    Intent intent = new Intent(context, GraphActivity.class);
    context.startActivity(intent);
    Toast.makeText(context, "Authentication successful", Toast.LENGTH_SHORT).show();
}
```

3.4.4.10 Creating audio from user details – synthesizeToFile

Data gathered from the user during signup is processed to create an audio file that is later played to the receiver of an emergency call [10]. The function uses the inbuilt *TextToSpeech* engine in Android to read the user entered text, while an input stream instance is used to save the resulting audio to local storage in the application. This process is programmed to repeat whenever the profile data is edited, to ensure that the audio file always contains the latest information. The users' name, address, contact details and Eircode are read to the receiver of the phone call along with an announcement to inform them that the call is automated. This is then looped until the call is ended. This snippet shows how the file is created and saved to the local device.

```
public void makeAudioFile(){
    path = new File(Environment.getExtStorageDir().getAbsoPath() +"/"+
getResources().getString(R.string.app_name));
    path.mkdir();
    Filename = path + "/eircode.wav";
    soundFile = new File(Filename);

    if (soundFile.exists()){
        soundFile.delete();
    }
    if (myTTS.synthesizeToFile(forEmergServ(), null, soundFile, Filename) ==
TextToSpeech.SUCCESS) {

        Toast.makeText(getApplicationContext(), "Sound file created")
    } else {
        Toast.makeText(getApplicationContext(), "Oops! Sound file not created");
    }
}
```

3.4.4.11 Countdown to abort call

When a trigger of the system takes place, the user is presented with a countdown timer indicating how many seconds they have left to cancel the call before it is initiated. This activity also calls out the timer through the phones loudspeaker to ensure that the user is aware of the impending call. This function is created using an instance of *CountDownTimer* in Java, and a countdown duration is specified in the form of milliseconds. A message Handler is used to allow the countdown to run on a separate thread to the main thread. Without the use of a new thread for the countdown, the main thread would not be accessible to run the *cancel* operation when the user interacts with the button. On completion of the timer, the call to the emergency services is started and after a short wait time, a run method begins a new *Intent* activity. This ends the existing process and starts the *LoudspeakerActivity* class.

The following snippet shows how the countdown timer is created, including converting the increments to speech and using a new thread to launch the resulting activity. This ensures the phone application goes to the background.

```java
public void countdownTimer() {
    myCounter = new CountDownTimer(10000, 1000) {
        public void onTick(long millisUntilFinished) {
            int tempIndex = (int) (millisUntilFinished / 1000);
            countdown.setText("" + millisUntilFinished / 1000);
            mySpeech.speak(String.valueOf(tempIndex), TextToSpeech.QUEUE_FLUSH, null);
        }
        public void onFinish() {
            countdown.setText("00");
            Intent intent = new Intent(Intent.ACTION_CALL);
            intent.setData(Uri.parse("tel:" + emergencyNum));
            intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            intent.addFlags(Intent.FLAG_FROM_BACKGROUND);

            startActivity(intent);

            final Handler handler = new Handler();
            handler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    startActivity(new Intent(this, LoudspeakerActivity.class));
                    finish();
                }
            }, 1000);
        }
    }.start();
}
```

3.4.4.12 Night mode – Lowering expected heart rate during sleep

A function has been developed in the main activity that allows the user to toggle on and off a night-mode. This mode adjusts the response to lower heart rate readings to account for the expected drop that occurs during normal sleep. The user can toggle this function and a Boolean value is sent to the database to indicate their chosen mode. The background service polls the database every time a new reading is received and alongside the upper and lower limits, this service reads in the state of the night-mode switch. If the night-mode is active, the application will allow a deviation from the lower limit that the user has specified in their profile. If a reading is received that exceeds this deviation limit, the service will respond as normal by triggering an emergency call.

3.4.4.13 Loudspeaker – Reading audio data to call receiver

This class is responsible for loading in the audio file from local storage and playing the audio to the call receiver [10]. This is performed with the use of a *MediaPlayer* instance which receives the Uri and path to the file and then proceeds to play the audio. This class is also responsible for remotely unlocking the users' front door and for outputting the users contact and medical details onscreen for the benefit of the first responders. The following snippet shows how the MediaPlayer object is used to play the audio to the call receiver having already received all the required data.

```
public void startSound() {
    Thread thread = new Thread() {
        @Override
        public void run() {
            try {
                while (true) {
                    sleep(1000);
                    mediaPlayer.start();
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    };
    thread.start();
}
```

3.4.4.14 Unlock front door - Smart-home cloud control

The system has been programmed to automatically unlock the users' front door when an emergency call is triggered. This is done by calling the following method in the LoudspeakerActivity class when a call is active. This method repeatedly activates and deactivates the locking mechanism. This was done to avoid the chances of the door not unlocking reliably. The values being set below are sent to Firebase, where they are received by the Raspberry Pi and used to trigger the locking and unlocking.

```
public boolean openFrontDoor() {
    Thread thread = new Thread() {
        @Override
        public void run() {
            try {
                while (isAlive) {
                    controlDbRef.child("switch1").setValue(true);
                    sleep(2000);

                    controlDbRef.child("switch1").setValue(false);
                    sleep(2000);
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    };
    thread.start();
    return true;
}
```

3.4.5 Android Wear Smart-watch Development

An Android Wear smart-watch is used to take readings from the users' wrist, through an optical heart rate sensor. The Moto 360 1st generation watch was selected for the project due it its availability and its convenient wireless charging. The watch runs on the 1st generation of Android Wear, an operating system developed specifically for wearable devices. As a result of this watch being an initial release device and not receiving Android Wear 2.0, there were many challenges in the development process due to poor documentation and support in several areas.

An application was developed for the watch and consisted of a simple UI that displayed the last known heart rate of the user. Since the battery life of the device was limited to a single day, it was important to ensure that the application was as efficient as possible with no unnecessary wake locks. A background service was written to obtain the users' heart rate once every hour without waking the screen, and to transmit this data to the mobile application. As per *Google* development guidelines, all processing was handled in the mobile application, where a much more powerful processor and greater battery capabilities were available.

3.4.5.1 Background service – Reading heart rate

The smart-watch application consists of a main activity, used for the user interface and a background service. The background service contains the bulk of the functionality for the application. This background service uses the *googleAPIClient*, *SensorManager* interface and *DataItem* objects.

3.4.5.2 Using heart rate sensor – SensorManager

SensorManager is a generic class that is part of the *Google* API library used to access onboard sensors in mobile and wearable devices [2]. The background service includes an instance of the *SensorManager* and this instance is then assigned to the required sensor. The *SensorManager* provides methods for *onSensorChanged* and *onAccuracyChanged*.

When the background service starts, a listener is registered on the selected sensor. The *onSensorChanged* method receives raw updates from the sensor in a primitive array and it is checked to ensure that the data is not null before the heart rate data is extracted and sent to a method for transmitting to the mobile device. When complete, the listener is unregistered to free up the sensor and will not be registered again until the required interval between scans is reached, or the user chooses to check their heart rate manually.

The code snippet below is called by the listener when heart activity is detected. It shows how the *onSensorChanged* method creates the timestamp, gets the reading from the sensor event, checks if the reading is greater than zero and unregisters the listener, before sending the newly obtained data to a method for transmission to the mobile device.

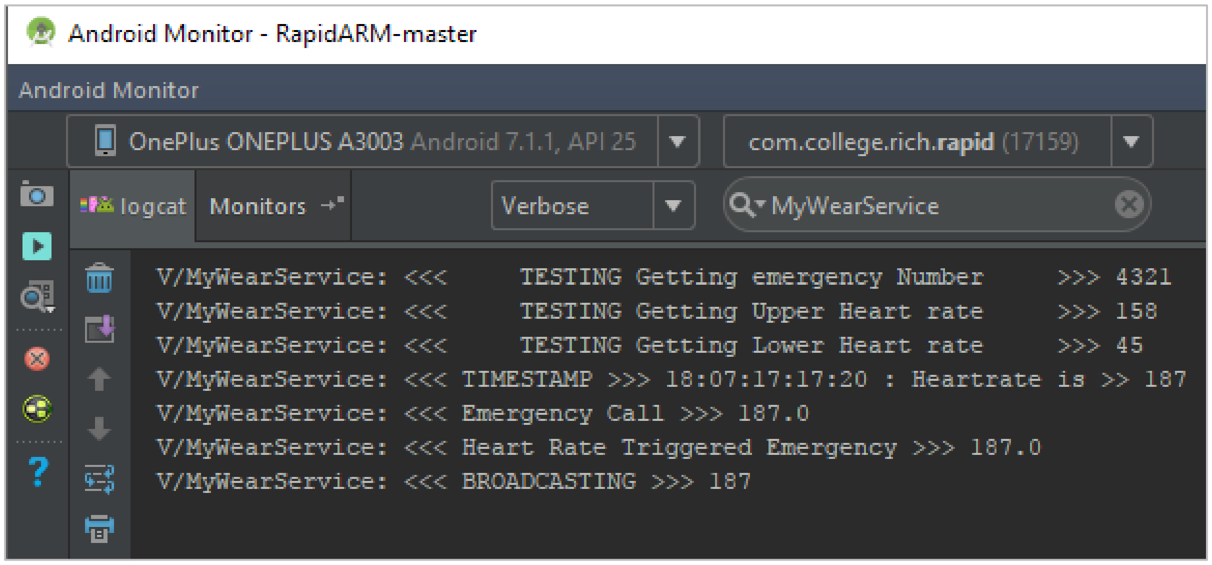
```
public void onSensorChanged(SensorEvent event) {
    long timestamp = event.timestamp;                   //making the timestamp
    float value = event.values[0];                      //getting sensor value

    if (value > 0) {                                    //checking if the rate is valid
        mSensorManager.unregisterListener(this);
        stopSelf();
        Log.d(TAG, "*** Value is: " + value);           //logging to console for testing
        sendHRCount(value, timestamp);                  //sending data to mobile

    } else {
        Log.d(TAG, "*** Value is 0: " + value);         //logging to console when failed
    }
    stopSelf();                                         //stop SensorManager
}
```

3.4.5.3 Sending data to the phone – DataItems

When data is successfully received from the sensor, it is passed to a method for transmission to the mobile device. This is done using the *DataApi* and the *googleAPIClient* to make a connection with the mobile device [2]. The raw data from the sensor is combined with an automatically generated timestamp from the system time into a *Map* object. This resulting DataItem object is then transmitted to the mobile device and a listener is used to check whether transmission was successful or not. If the message delivery failed or of the mobile device isn't currently connected, the messages are queued and delivered as soon as a connection is available. The code snippet below shows the method for receiving the heart rate and timestamp for processing and sending to the mobile device. The mobile device will receive a JSON object with the parent node of "*HR-Rate"* and two children nodes of "*HR*" and "*timestamp*".

```
public void sendHRCount(float rate, long timestamp) {
    PutDataMapRequest mapReq = PutDataMapRequest.create("/HR-Rate");
    mapReq.getDataMap().putFloat("HR", rate);
    mapReq.getDataMap().putLong("timestamp", timestamp);
    PutDataRequest req = mapReq.asPutDataRequest();

    Wearable.DataApi.putDataItem(mGoogleApiClient, req)
            .setResultCallback(. . . {
                    if (dataItemResult.getStatus().isSuccess())
                        Log.d(TAG, "DATA SUCCESS: ");
                    else
                        Log.d(TAG, "DATA FAIL: ");
                }
            });
```

The mobile device receives this transmission by listening for incoming DataItems and checking if the parent node matches its expected "*/HR-Rate*" before unpacking the data and processing in the background service. The following screenshot of the Android debugging monitor shows the incoming data from the smartwatch when it was received by the "*MyWearService"* class in the mobile application. On this instance, the heart rate reading exceeded the normal limits and an emergency response was triggered by the background service. The reading was then broadcast to the main UI.


Android Monitor - RapidARM-master
Android Monitor
OnePlus ONEPLUS A3003 Android 7.1.1, API 25
com.college.rich.rapid (17159)
logcat
Monitors
Verbose
MyWearService

```
V/MyWearService: <<<     TESTING Getting emergency Number     >>> 4321
V/MyWearService: <<<     TESTING Getting Upper Heart rate     >>> 158
V/MyWearService: <<<     TESTING Getting Lower Heart rate     >>> 45
V/MyWearService: <<< TIMESTAMP >>> 18:07:17:17:20 : Heartrate is >> 187
V/MyWearService: <<< Emergency Call >>> 187.0
V/MyWearService: <<< Heart Rate Triggered Emergency >>> 187.0
V/MyWearService: <<< BROADCASTING >>> 187
```

3.4.5.4 Testing - Generating random heart rate data

For testing purposes, a method for generating randomised heart rate data has been developed. This method is contained in the main activity of the application and is started by tapping on the heart rate output text field. The *Random* class in Java is used to generate a value between a specific range and this is combined with a *timestamp* before being transmitted to the mobile device. This method is essential for development purposes, as the heart rate sensor used could at times be unreliable and slow. The snippet below shows this process, including formatting the timestamp to make it more human readable before being saved to the database.

```java
public void testData(View view) {
    Random r = new Random();
    int Low = 30;                                           //Lower limit
    int High = 190;                                         //Upper limit
    int randomResult = r.nextInt(High-Low) +Low;            //Random between lower/upper

    Calendar cal = Calendar.getInstance();                  //Formatting the timestamp
    SimpleDateFormat dateFormat = new SimpleDateFormat("dd:MM:yy:HH:mm");
    String timestamp = dateFormat.format(cal.getTime());

    PutDataMapRequest mapReq = PutDataMapRequest.create("/HR-Rate");
    mapReq.getDataMap().putFloat("HR", randomResult);
    mapReq.getDataMap().putString("timestamp", timestamp);
    PutDataRequest req = mapReq.asPutDataRequest();         //Sending map to mobile

    Wearable.DataApi.putDataItem(mGoogleApiClient, req)
            .setResultCallback(. . .{
            if (dataItemResult.getStatus().isSuccess()) {
                    Log.e(TAG, "Sending DataItem Succeeded: " +randomResult);
                } else {
                    Log.e(TAG, "Sending DataItem Failed: ");
                }
            }
       });
}
```

3.4.6 Raspberry Pi - IoT Development Board

A *Raspberry Pi* development board is used alongside the *Raspbian* Linux based operating system to run the code needed for the smart-home features. A smart-home kit has been built using the development board and an 8-channel relay board that can be tapped into the users' mains system at the breaker panel. This allows the user to control up to 8 traditional electrical devices within their home without the requirement for these devices to be "*smart*" enabled. The development board runs *Python* code that connects to the *Firebase* Cloud Database to obtain control commands and performs these actions by setting the GPIO pins of the board either high or low, depending on which switch state is desired. The *Raspberry Pi* can be accessed over the network using the *Putty* SSH client and files are copied to the device using *FileZilla* for Secure File Transfer Protocol.

3.4.6.1 Python – Programming language

A *Python* script contains the code necessary for reading the cloud control messages and for interacting with the relay board. The script is written in *Python 2.7* and implements the following libraries,

3.4.6.2 Firebase Python API

The *Firebase* API provides the necessary libraries to access the database and to read and write data to it. A unique database URL is required from the Firebase console, alongside an *Auth* token used for authenticating the user, if necessary. Using this data, the developer creates a *databaseReference* that is addressed whenever interaction with the database is required. E.g.

```
firebase = FirebaseApplication("www.fake-url.com", authentication="authDataHere")
```

When the developer wants to access data from *Firebase*, this *databaseReference* is used, as shown below.

```
myData = firebase.get("/controlMessage", None
```

The resulting *myData* object is a JSON object containing all the data from the node "*controlMessage*" and its children.

3.4.6.3 Rpi.GPIO

Rpi.GPIO is a library used for interacting with the *General-Purpose-Input/Output* pins of the *Raspberry Pi* development board. This library enables the developer to define a pin on the board, assign it to either an input or output and to use that pin to interact with a wide range of devices by feeding 3.3v through the pin. The data obtained from *Firebase* is used to dictate what signals are sent across these pins. 8 GPIO pins in total have been used in the project, each connected to a different channel on the relay board and used to actuate a different electrical device. The following snippet shows the method that receives the switch ID, switch state and the pin number it is assigned to, before acting on the pin using the GPIO methods.

```
def makeAction(switchID, switchState, pinNum):
        if switchState == True:
                print "Switching GPIO", pinNum, switchID, "to ON"
                GPIO.output(pinNum, 0)
        elif switchState == False:
                print "Switching GPIO pin", pinNum, switchID, "to OFF"
                GPIO.output(pinNum, 1)
        pinNum = 0
```

3.4.6.4 8-Channel relay board

The relay board is connected to the *Raspberry Pi*'s +5v VCC, Ground and 8 individual GPIO pins. The relay state is set to "*ON"* when no voltage is received on the input pin. When the Python code triggers the GPIO pin to "GPIO.output("pinNumber", 1)", the pin receives 3.3v and triggers the relay state to switch to the "*OFF*" position. Alternately, setting the GPIO pin to "GPIO.output("pinNumber", 0)", results in no voltage being supplied to the relay and the state being switched back to "*ON*".

3.4.6.5 Threading

The *Threading* library is used in the Python application as a method of handling simultaneous publishing and subscribing to and from the *Raspberry Pi*. This benefits the system by insuring that processes are not blocked from completing while another process is using the main thread, and ensures that all state information is accurate at the time of viewing.

3.4.6.6 JSON library

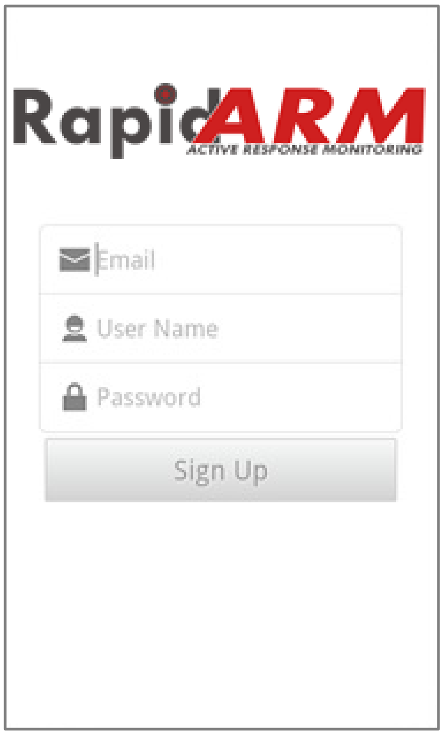
The *JSON* Python library is used parse the response from Firebase into an object that can be iterated over, allowing for efficient extraction of individual pieces of data. As result of this, a single method that addresses all pins for switching the state can be written, instead of individual methods for each pin. Using iteration, a GPIO pin number can be assigned to an incoming message and this pin number is passed alongside the control message received, to the generic method that will perform the action. This results in fewer lines of code, improving readability and avoids repetition.

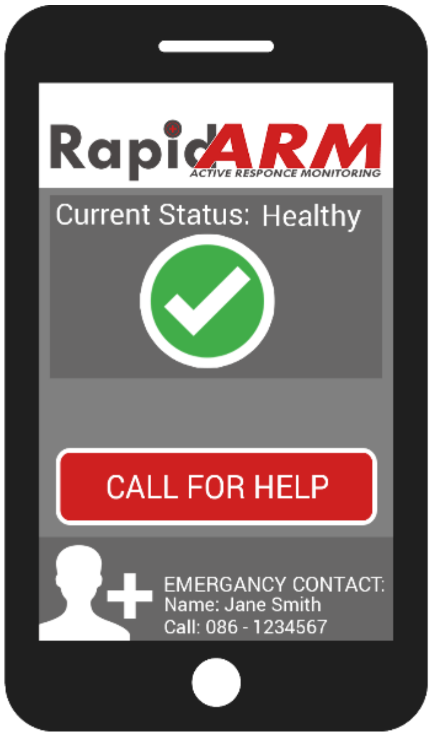
3.4.6.7 Time library

The *time* library in Python is a simple function that enables the developer to write in a delay time and is useful in situations where a pause is needed within a loop. In the project, an infinite loop is used to constantly check for updates from the database, but this happens faster than the human user can observe, leading to many more queries than is necessary to perform the required action. A short delay at the end of the loop allows the application to process the most recent commands received without the need for unnecessary network calls and processing that otherwise waste valuable resources and affect the reliability of the application.

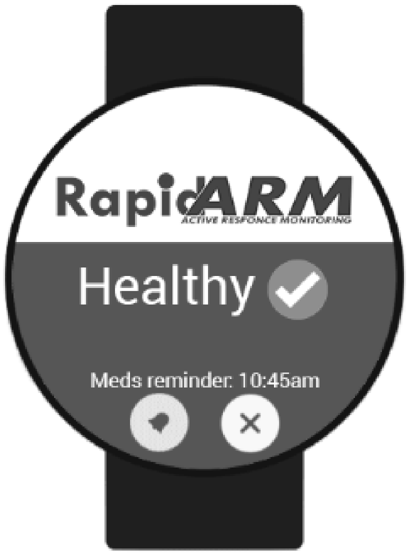
3.5 Graphical User Interface Mock-up

3.5.1 Login and main user interface layouts





3.5.2 Smartwatch user interface



3.6 Testing and Evaluation

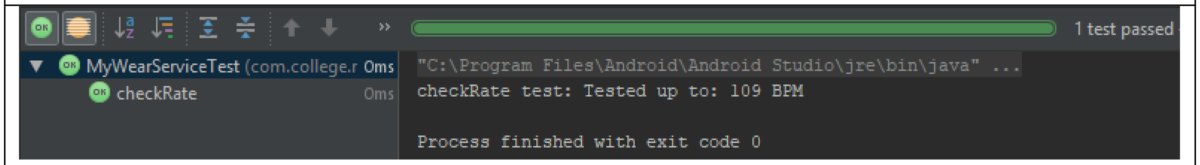
3.6.1 Unit Testing

These tests have been performed in Android Studio using the *JUnit* framework and *Mockito*. Unit tests are carried out on key methods within the system using *JUnit4*. *Mockito* is used to mock dependencies of the various classes so that the tests can be carried out in the virtual machine.

Tested By	Rich Mangan
Tested On	29/04/2017

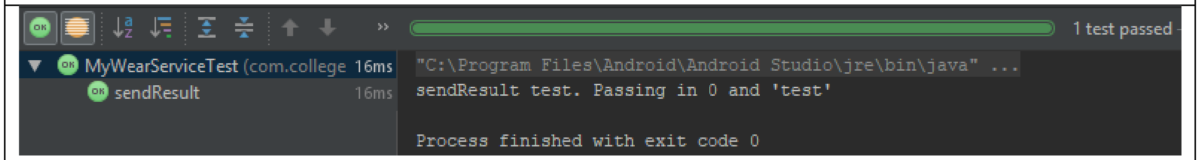
Test	**Test case Description**	**Expected Result**	**Result**	**Remarks**
1	Test *checkRate* method	All results return false	PASS	
2	Test *sendResult* method	All results return false	PASS	
3	Test *emergencyCall* method	All results return true	PASS	
4	Test *getData* method	All results return true	PASS	
5	Test *countdownTimer* method	100’000 values return successfully	PASS	
6	Test *getFirebaseData* method	All results return true	PASS	
7	Test *setSwitchState* method	All results return true	PASS	
8	Test *sendNewToggleState* method	All results return true	PASS	
9	Test *buildGraphs* method	All results return true	PASS	
10	Test *updateCharts* method	All results return true	PASS	

3.6.1.1 Test 1 – checkRate

Test name:	
	checkRate
Target class:	
	MyWearService.java
Description:	
	The purpose of this test is to test the applications ability to trigger an event based on the users' heart rate. This is performed on the *checkRate* method in the *MyWearService* class. The test contains test values for upper and lower heart rate. A "*rate*" variable is passed to the method and is iterated until it is equal to the upper value.
Procedure:	
	The test class instantiates the target class and sets the *rate* variable equal to the *lower* variable. The *rate*, *upper* and *lower* variables are passed into the method and the method returns a Boolean. The test iterates the *rate* until it is equal to the *upper* variable, testing at each iteration. The method should always return false when the *rate* is less than the *upper* and greater than the *lower* variable.
Expected result:	
	False
Actual result	
	False
Test status:	
	PASS
Output:	
	

```
"C:\Program Files\Android\Android Studio\jre\bin\java" ...
checkRate test: Tested up to: 109 BPM

Process finished with exit code 0
```

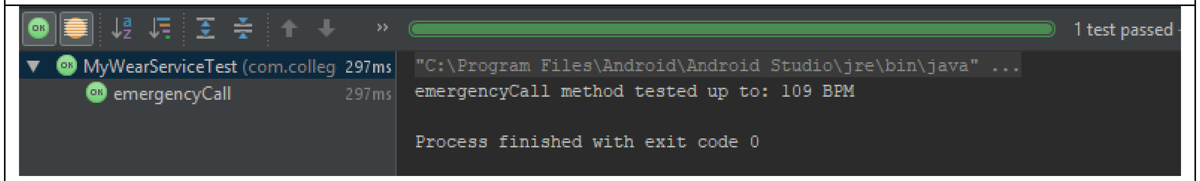
3.6.1.2 Test 2 – sendResult

Test name:	
	sendResult
Target class:	
	MyWearService.java
Description:	
	The purpose of this test is to test the applications ability to send incoming heart rate data to the main UI via broadcasts. This is performed on the *sendResult* method in the *MyWearService* class.
Procedure:	
	The test class instantiates the target class and passes the *rate* variable and a test string in the parameters. The method returns a Boolean. If the string is not null, the Boolean will be false, else it will be true.
Expected result:	
	False
Actual result	
	False
Test status:	
	PASS
Output:	

```
MyWearServiceTest (com.college 16ms
    sendResult                 16ms
```

```
1 test passed
"C:\Program Files\Android\Android Studio\jre\bin\java" ...
sendResult test. Passing in 0 and 'test'

Process finished with exit code 0
```

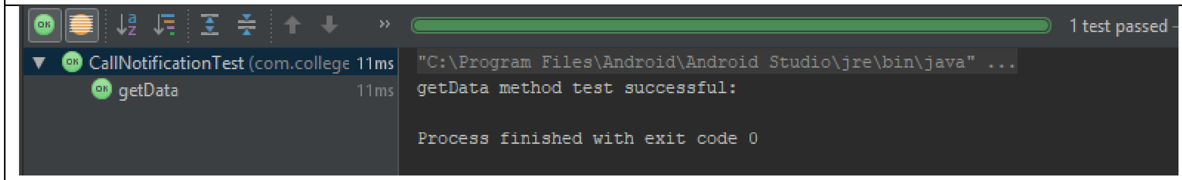
3.6.1.3 Test 3 – emergencyCall

Test name:	
	emergencyCall
Target class:	
	MyWearService.java
Description:	
	The purpose of this test is to test the applications ability to trigger an emergency call and start the call activity. This method checks the users' heart rate in the *checkRate* method and returns a Boolean value. This is performed on the *emergencyCall* method in the *MyWearService* class.
Procedure:	
	The test class instantiates the target class and creates local variables for *rate*, *upper* and *lower*. The test consists of a loop that sets the value of *rate* equal to the *lower* reading, iterating it until it is equal to the *upper* variable. The will only return "*true*" when the input passes the *checkRate* call within, otherwise it will return "*false*".
Expected result:	
	True
Actual result	
	True
Test status:	
	PASS
Output:	
	

```
MyWearServiceTest (com.colleg 297ms
    emergencyCall             297ms
```

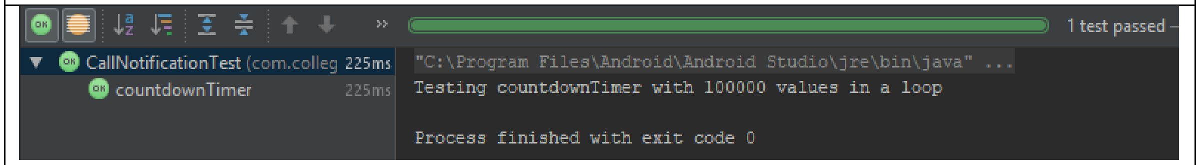
```
"C:\Program Files\Android\Android Studio\jre\bin\java" ...
emergencyCall method tested up to: 109 BPM

Process finished with exit code 0
```

3.6.1.4 Test 4 – getData

Test name:
getData
Target class:
CallNotificationTest.java
Description:
The purpose of this test is to check if the application can successfully obtain required data from the cloud database using listeners. This method listens to individual nodes within Firebase and assigns their values to local variables. This method is called by the *onCreate* method when the *CallNotification* activity begins.
Procedure:
The test class instantiates the target class and calls the getData method. This method contains a try/catch statement and returns "*true*" if successful or "*false*" if the method call fails.
Expected result:
True
Actual result
True
Test status:
PASS
Output:


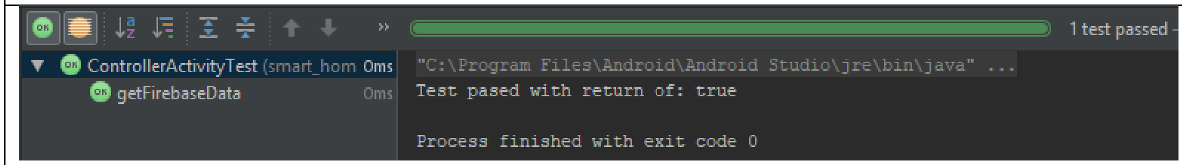
```
"C:\Program Files\Android\Android Studio\jre\bin\java" ...
getData method test successful:

Process finished with exit code 0
```

3.6.1.5 Test 5 – countdownTimer

Test name:	
	countdownTimer
Target class:	
	CallNotificationTest.java
Description:	
	The purpose of this test is to verify that the application can successfully begin a countdown timer when an event is triggered, and to evoke the phone call activity when the time has elapsed. This method returns a Boolean of “*true*” when successfully executed, otherwise it returns “*false”*. The test passes in a *long* value, starting at *0* and incrementing to *100’000*, testing at each iteration.
Procedure:	
	The test class instantiates the target class and calls the countdownTimer method. This method accepts a *long* value duration time in the parameters and using a loop, it tests the *countdownTimer* method with all values from *0* up to *100’000*. A successful method call will return a “*true”* value, otherwise it will return a “*false”* value in the return statement.
Expected result:	
	True
Actual result	
	True
Test status:	
	PASS
Output:	
	

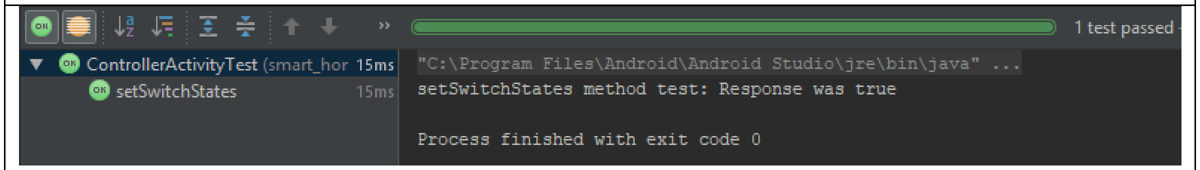
```
"C:\Program Files\Android\Android Studio\jre\bin\java" ...
Testing countdownTimer with 100000 values in a loop

Process finished with exit code 0
```

3.6.1.6 Test 6 – getFirebaseData

Test name:	
	getFirebaseData
Target class:	
	ControllerActivityTest.java
Description:	
	The purpose of this test is to verify that the ControllerActivity can obtain the state information from the database. This method expects a Boolean return value of true when successful and false otherwise.
Procedure:	
	The test class instantiates the target class and calls the *getFirebaseData* method. This method connects to the database and obtains the required data inside a try/catch statement. If the method response is successful, the return value will be true, otherwise it will be false.
Expected result:	
	True
Actual result	
	True
Test status:	
	PASS
Output:	
	

```
"C:\Program Files\Android\Android Studio\jre\bin\java" ...
Test pased with return of: true

Process finished with exit code 0
```

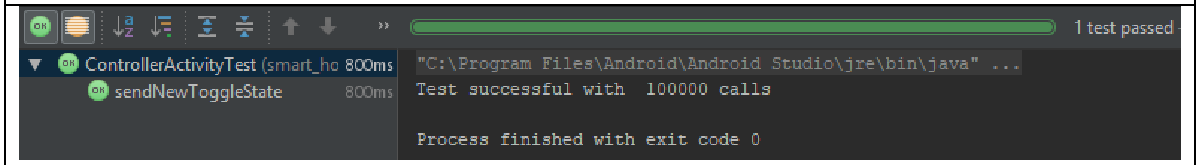
3.6.1.7 Test 7 – setSwitchState

Test name:	
	setSwitchStates
Target class:	
	ControllerActivityTest.java
Description:	
	The purpose of this test is to verify that the application can successfully represent the accurate state of the smart-home controller switches on load of the activity. This method is contained in the *ControllerActivity* and returns a Boolean value of true when successful and false otherwise.
Procedure:	
	The test class instantiates the target class and calls the *setSwitchState* method on loading of the activity. This class is called by the *getFirebaseData* method when data has been obtained successfully. The *setSwitchState* method returns a Boolean value of true when successful and false otherwise.
Expected result:	
	True
Actual result	
	True
Test status:	
	PASS
Output:	
	

```
"C:\Program Files\Android\Android Studio\jre\bin\java" ...
setSwitchStates method test: Response was true

Process finished with exit code 0
```

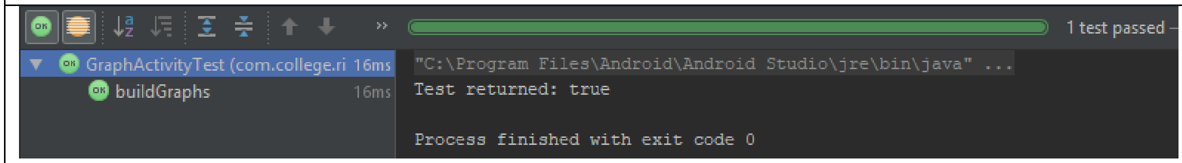
3.6.1.8 Test 8 – sendNewToggleState

Test name:	
	sendNewToggleState
Target class:	
	ControllerActivityTest.java
Description:	
	The purpose of this test is to verify that the application sends the new switch state to the cloud database. This method accepts a string and a Boolean value in the parameters. The test was conducted by passing in 100’000 random combinations of string and Boolean. This method is called by the onClick events in the UI.
Procedure:	
	The test class instantiates the target class and calls the *sendNewToggleState* method. A random string is generated, consisting of alphanumeric data and is paired with a randomly generated Boolean value. An iterator is used to run the test 100’000 times, passing the newly generated random data in at each iteration.
Expected result:	
	True
Actual result	
	True
Test status:	
	PASS
Output:	



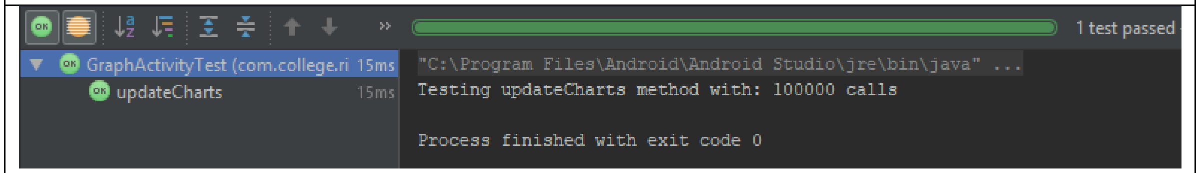
```
"C:\Program Files\Android\Android Studio\jre\bin\java" ...
Test successful with  100000 calls

Process finished with exit code 0
```

3.6.1.9 Test 9 – buildGraphs

Test name:	
	buildGraphs
Target class:	
	GraphActivityTest.java
Description:	
	The purpose of this test is to verify that the *GraphActivity* can successfully generate the graphs from data obtained from the database. This method returns a Boolean value of true when successful and false otherwise.
Procedure:	
	The test class instantiates the target class and calls the *buildGraphs* method. This method returns a Boolean value of true if successful and false otherwise.
Expected result:	
	True
Actual result	
	True
Test status:	
	PASS
Output:	
	

```
"C:\Program Files\Android\Android Studio\jre\bin\java" ...
Test returned: true

Process finished with exit code 0
```

3.6.1.10 Test 10 – updateCharts

Test name:	
	updateCharts
Target class:	
	ControllerActivityTest.java
Description:	
	The purpose of this test is to verify that the application can successfully update the graphs and charts in the UI. The test was conducted using an iterator to run the test 100’000 times.
Procedure:	
	The test class instantiates the target class and calls the *updateCharts* method. This method returns a Boolean value of true when successful and a false value otherwise. The test uses a while loop and an iterator to repeat the test 100’000 times.
Expected result:	
	True
Actual result	
	True
Test status:	
	PASS
Output:	

```
"C:\Program Files\Android\Android Studio\jre\bin\java" ...
Testing updateCharts method with: 100000 calls

Process finished with exit code 0
```

3.6.2 Usability Testing

The system has been delivered to an end user for final evaluation testing and has been worn for a few days to evaluate its ability to regularly take readings from the wearer, to act on the readings and how it can deal with possible false alarm scenarios. This testing phase has provided feedback on final modifications required before finalising the project.

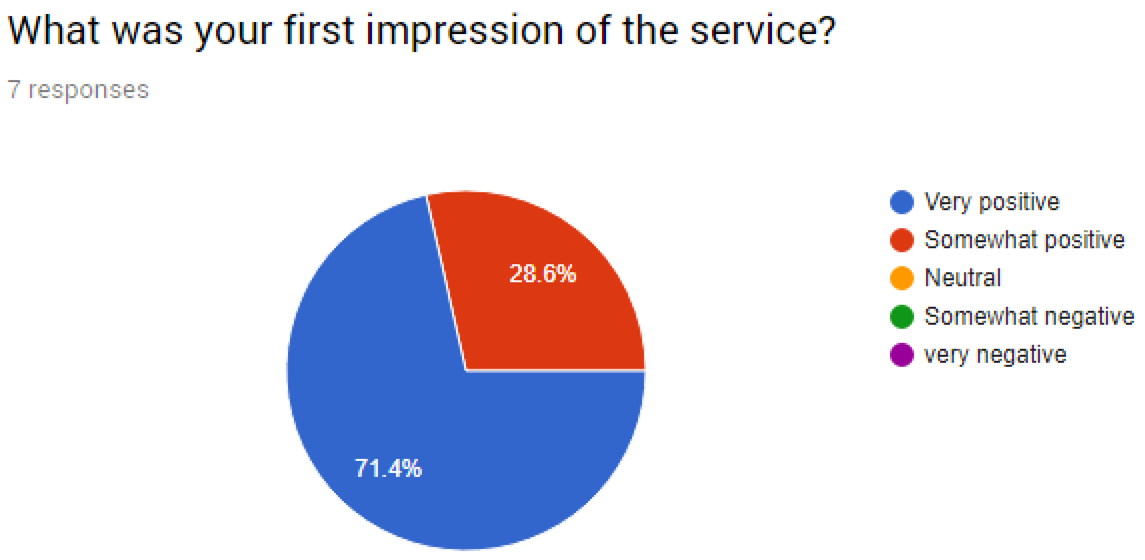
3.6.2.1 Usability Survey

The application was presented to 7 users for testing and their feedback gathered using a usability survey. This survey was created in *Google Forms* and distributed online. The responses are presented in pie charts and can be exported in *csv* format for analysis. The user was presented with 17 questions on the idea, function, responsiveness, and reliability of the project. Additionally, the user was asked for specific feedback on what they liked best and what they thought needed improvement in the application. The survey that was presented to the testers has been included in the appendix of this document.

3.6.2.2 Survey results

Analysis of the responses revealed a few issues in the usability of the application. While the pool of test users was relatively low, a number of those tested suggested that updating the user profile was not as obvious as intended. In particular, the "settings" button in the UI possibly didn't suggest that it contained links to edit the profile. This may be addressed with a popup tab on first load, or possibly by loading the activity with these options already expanded.

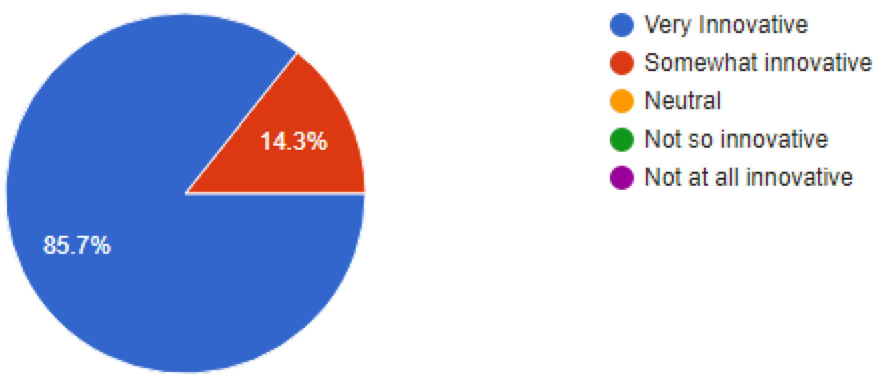
Of those tested, 71.4% had a very positive impression of the service, with the remaining having a somewhat positive impression.



85.7% of those tested rated the application as “Very innovative”, with the remainder rating it “Somewhat positive”.

How innovative is the application?

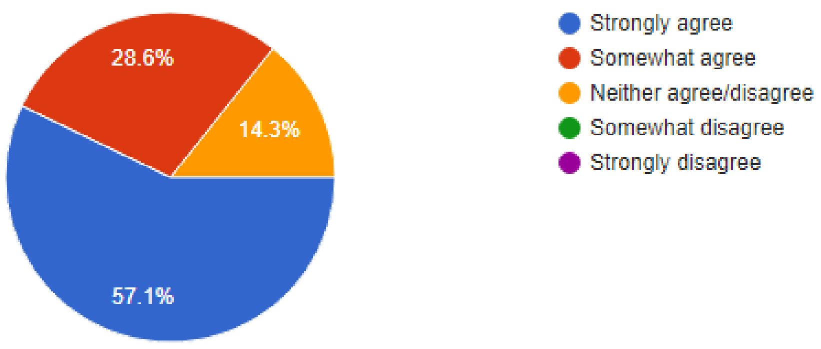
7 responses



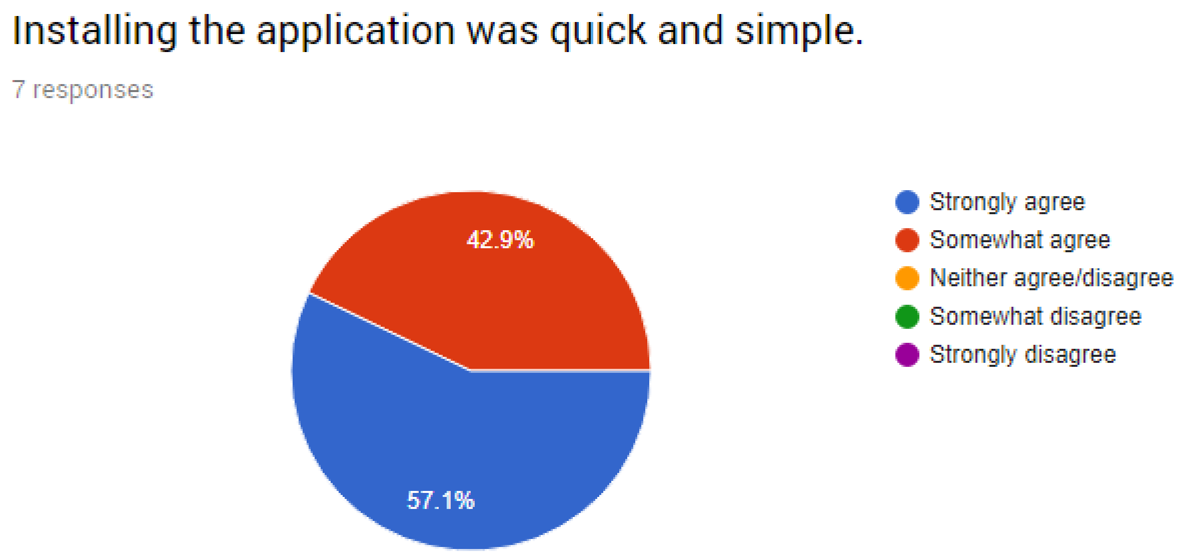
When asked whether they knew someone who would benefit from the service, the resulting data shows more variability than previous questions. 57.1% of those tested strongly agreed with the statement, while 28.6% somewhat agreed. The remaining 14.3% neither agreed nor disagreed. This may be interpreted as a miscommunication of the target market for the application when pitching it to the users, or possibly an ambiguous question.

When you think about the service, you know at least one person who could benefit from it.

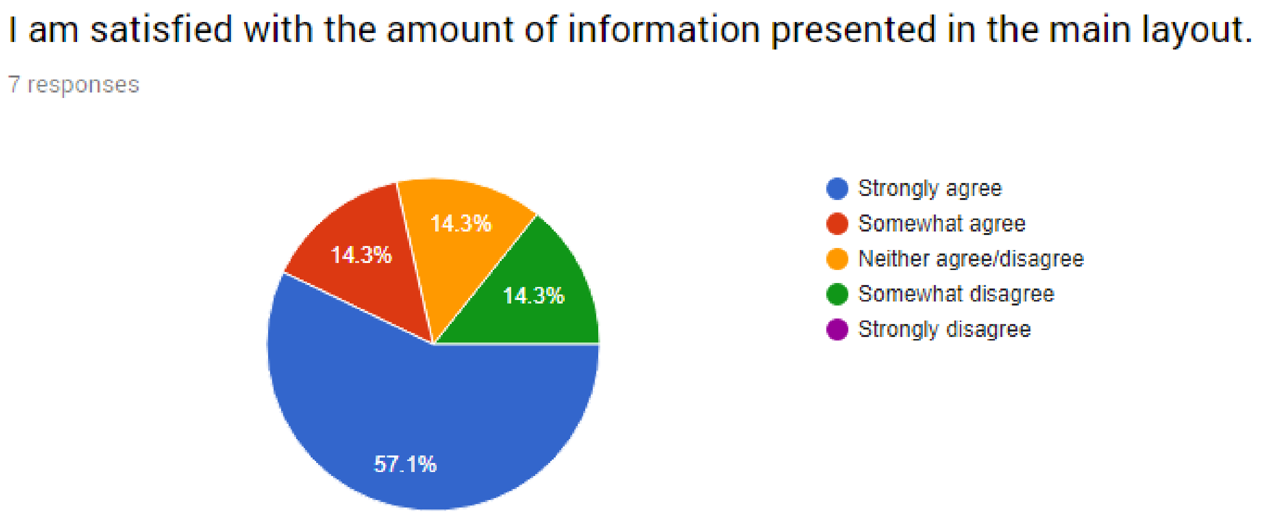
7 responses



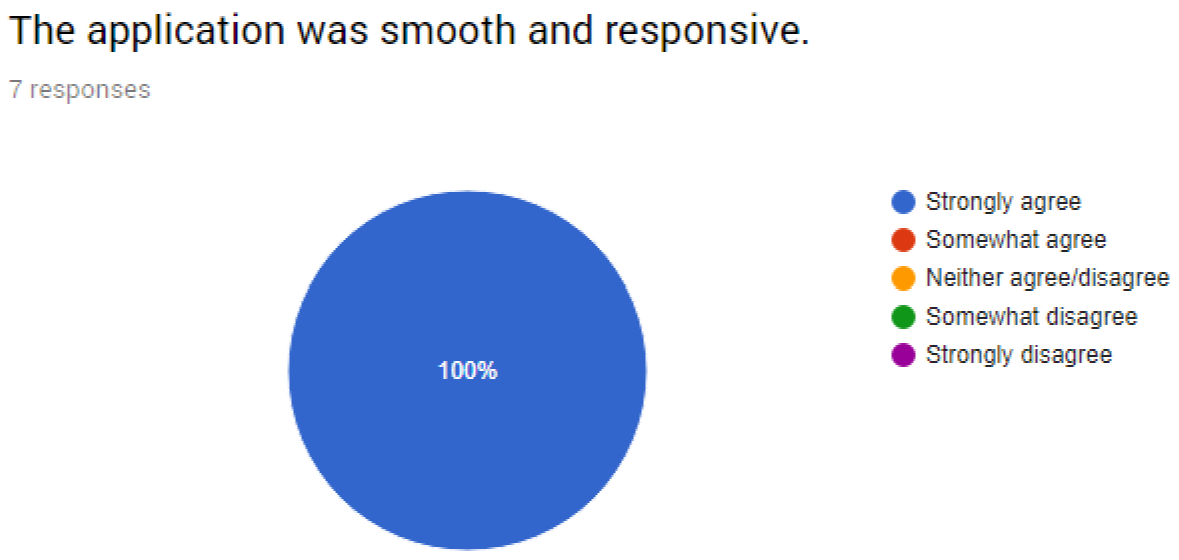
When asked about how simple the application was to install, the response was positive overall, with a relatively even split between "strongly agree" and "somewhat agree".



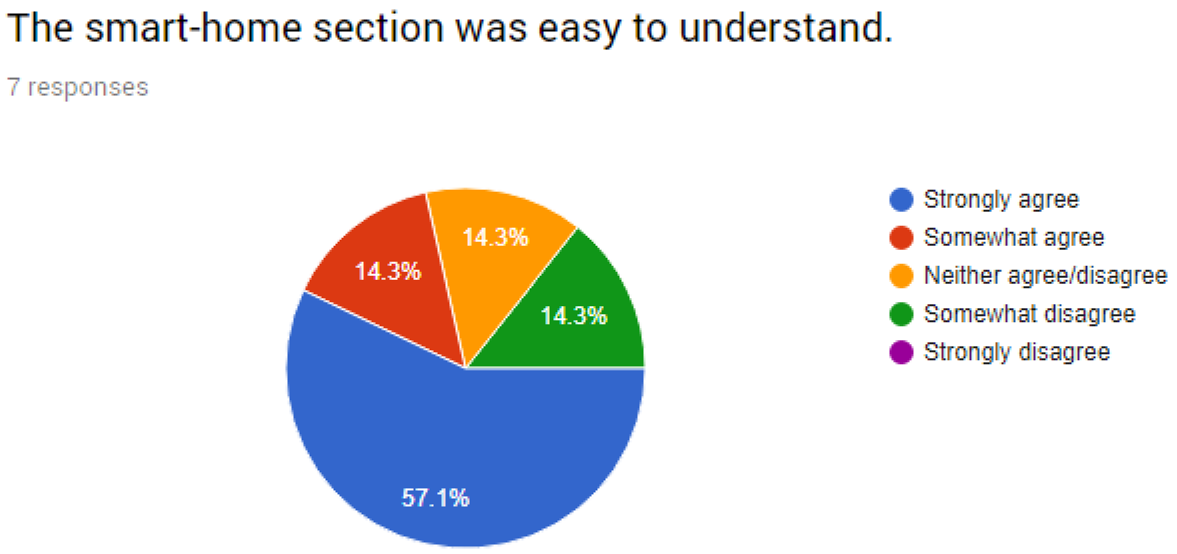
When asked if they were satisfied with the amount of information presented in the main layout, 57.1% of those tested indicated that they were very satisfied, while the remaining users were split evenly between "*somewhat satisfied*", "*neither*" and "*somewhat dissatisfied*".



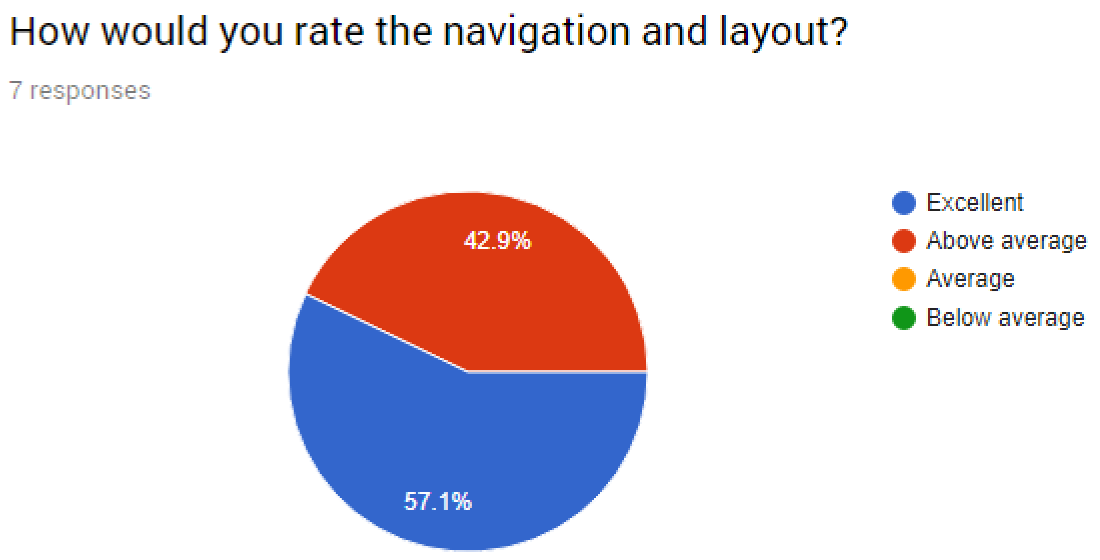
When asked about the smoothness and responsivity of the application, the testers were unanimous in saying that they strongly agreed that the application was smooth and responsive.



When presented with the suggestion that the smart-home section of the application was easy to understand, it seems that the testers differed somewhat. 57.1% strongly agreed, while the remaining users were split evenly in somewhat agreeing, somewhat disagreeing and having neither opinion. This may suggest that the user was confused with the infrastructure elements of the system rather than the actual smart-home section of the application. This could possibly have been avoided with an improved wording of the question. In the section for improvement suggestions, one user highlighted the smart-home section when commenting with "*The smart bit could maybe be explained a bit more*".

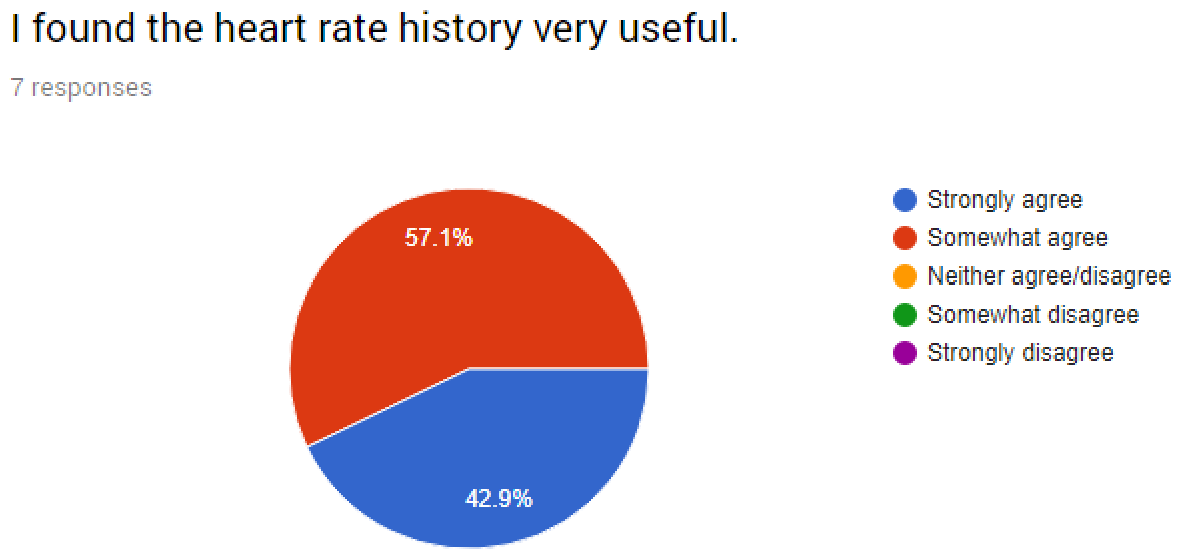


When asked to rate the navigation and layout of the application, the response from the users was overall positive, with a split between "*excellent*" and "*above average*".


How would you rate the navigation and layout?

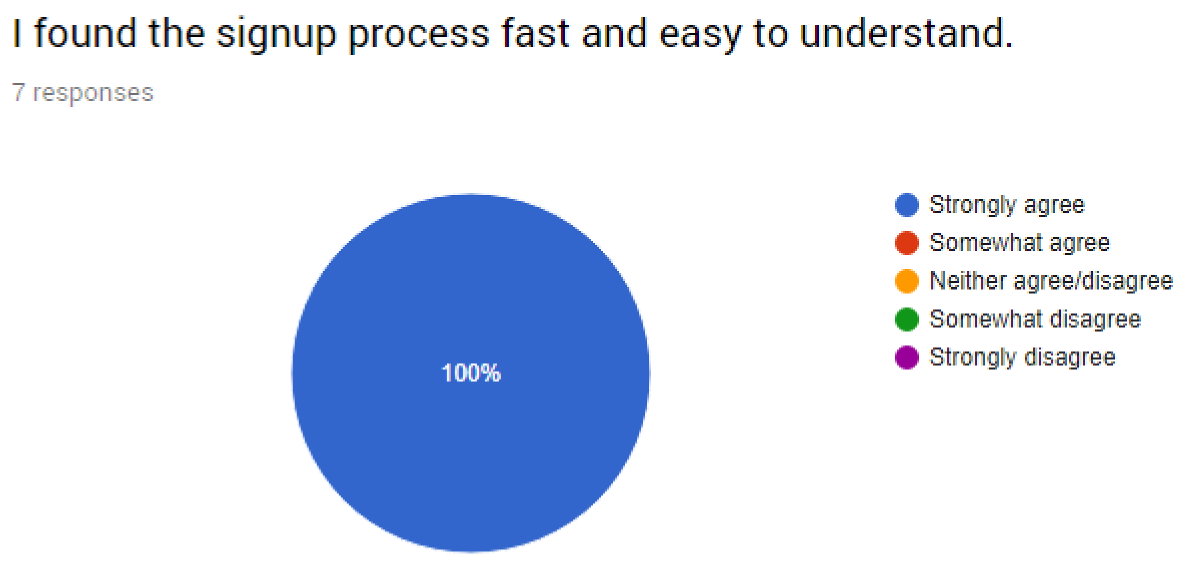
7 responses

The users were presented the proposition that they found the heart rate history section very useful. While all responses were positive, 57.1% of them somewhat agreed, with the remainder strongly agreeing. This may highlight an issue with explaining the purpose of the history section and pitching the benefits to the user. Additionally, one user gave specific feedback in which they suggested adding a function to email the history directly to their doctor.

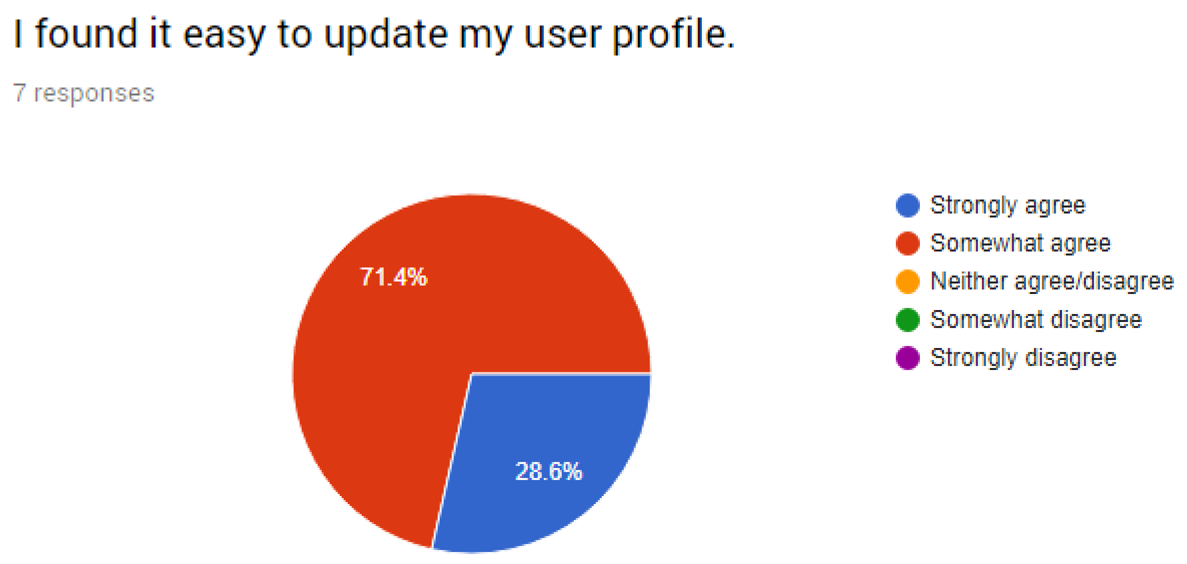

I found the heart rate history very useful.

7 responses

The response from the users in relation to the ease of signing up was unanimously positive.



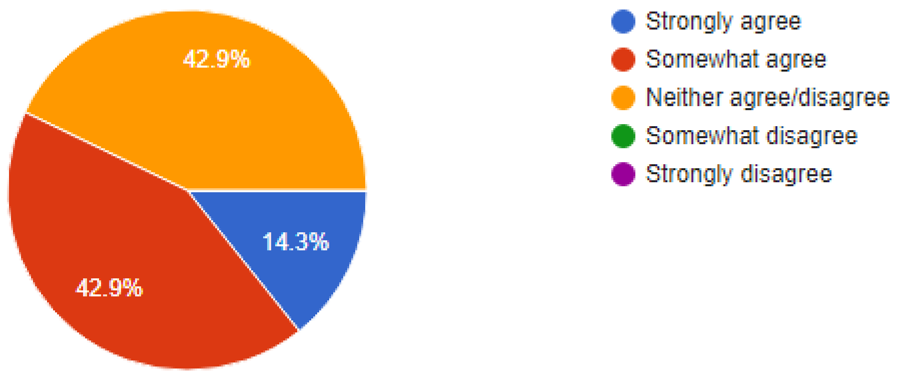
When presented with the suggestion that they found updating their profile easy, 71.4% of the users responded by indicating that they somewhat agree with the statement. The remainder of the users strongly agreed. Additional feedback was received with reference to the settings button used in the profile overview activity. The user suggested that navigating to the edit profile screen was not as obvious as it should be.



When asked if they agreed that the start-up guide was sufficiently detailed, the users were split at 42.9% either somewhat agreeing or neither agreeing/disagreeing, with the remaining 14.3% strongly agreeing. A number of those tested gave specific feedback on improving the user guide and instructions. While the start-up guide had been designed specifically to be simplistic to avoid being confusing, this may need to be revisited to provide more information to the user on first use.

The startup guide was detailed enough.

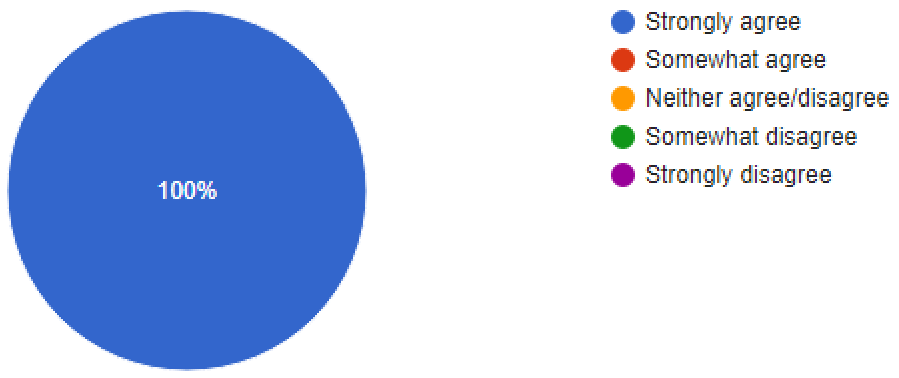
7 responses



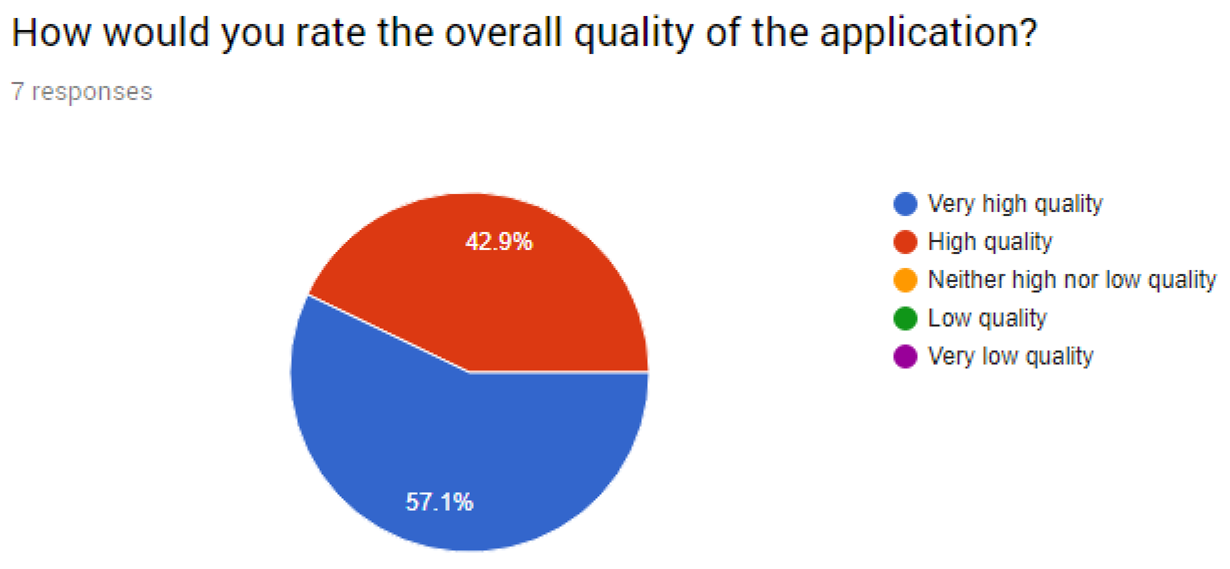
The users were unanimous in agreement that the service functioned without any technical difficulties.

The service functioned without technical problems

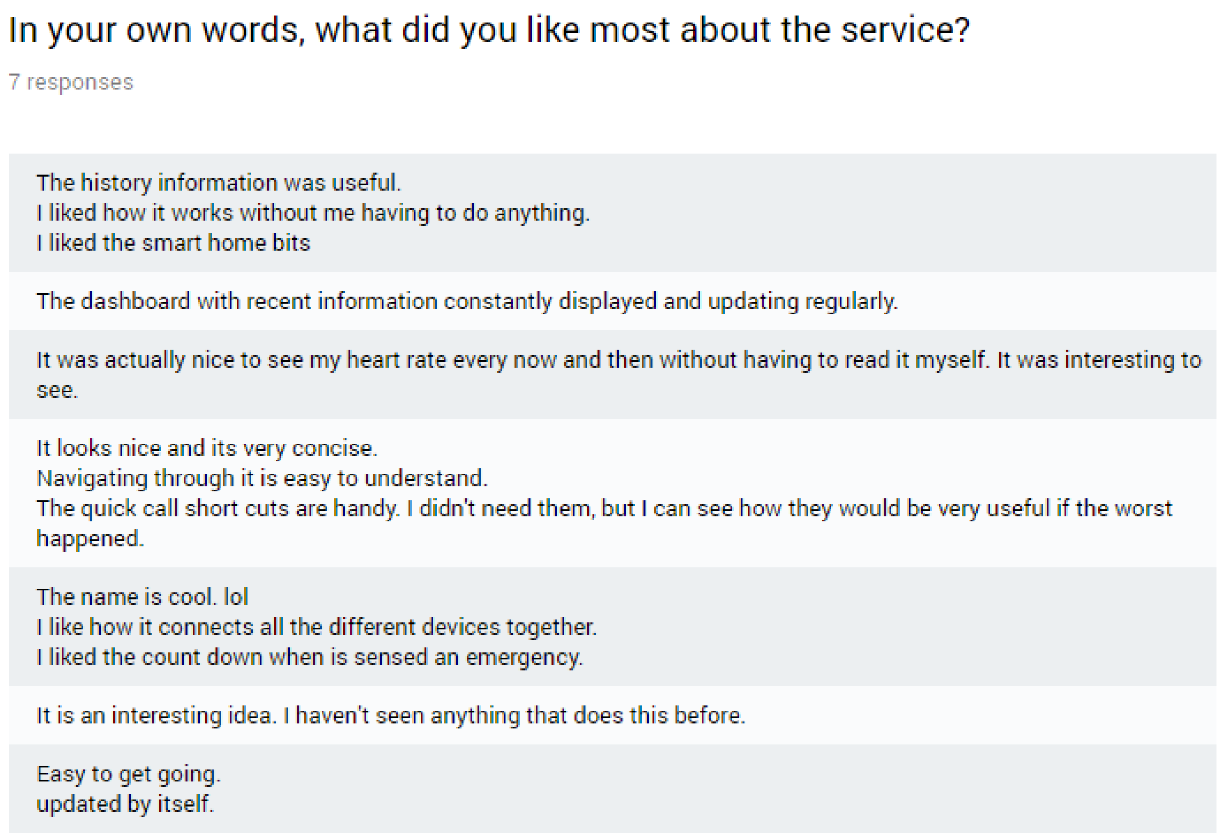
7 responses



When asked to rate the overall quality of the application, 57.1% of those tested rated the application as “very high quality”, while the remaining users rated it “high quality”.



When asked about what they liked most about the application, several users gave specific feedback about the autonomy of the service, how it connects multiple different devices and its smart-home functions.



In your own words, what did you like most about the service?

7 responses

The history information was useful.
I liked how it works without me having to do anything.
I liked the smart home bits

The dashboard with recent information constantly displayed and updating regularly.

It was actually nice to see my heart rate every now and then without having to read it myself. It was interesting to see.

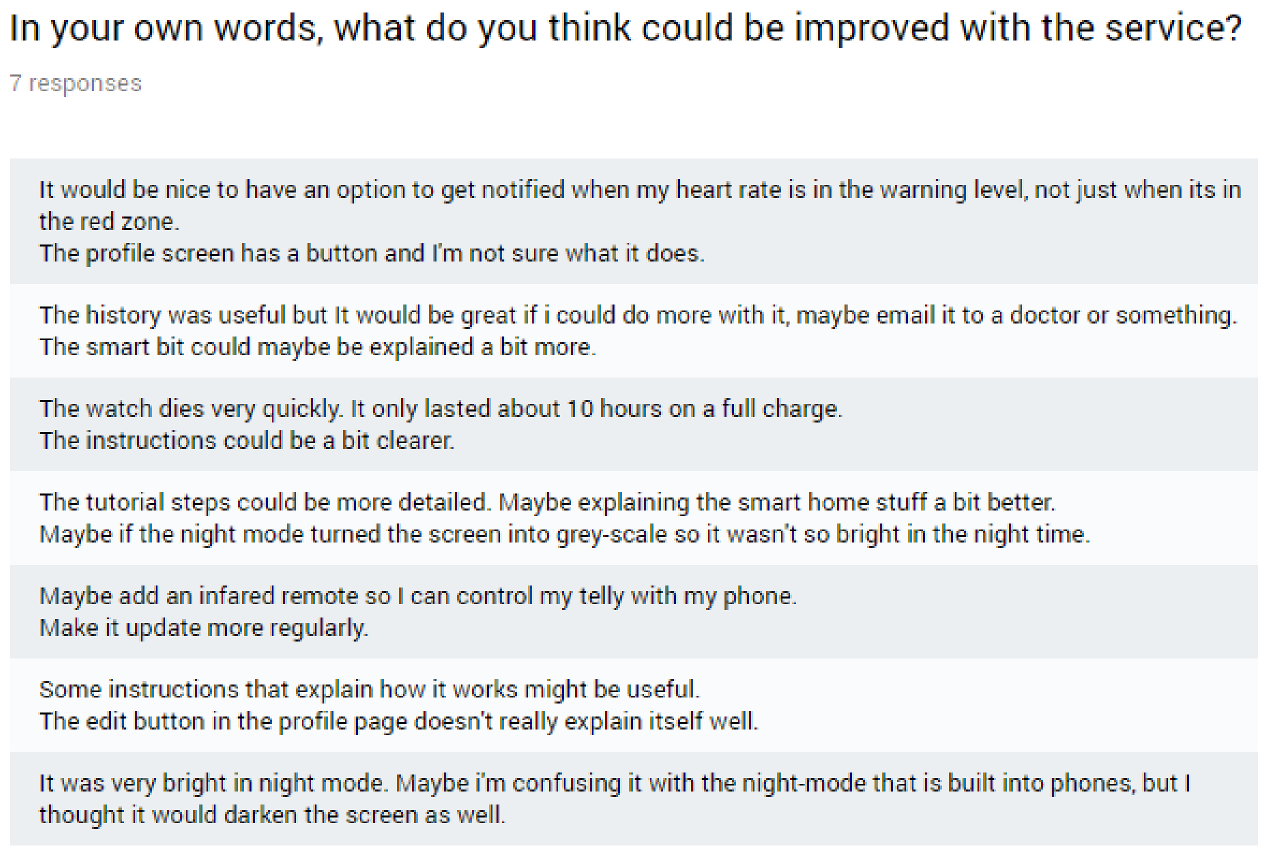
It looks nice and its very concise.
Navigating through it is easy to understand.
The quick call short cuts are handy. I didn't need them, but I can see how they would be very useful if the worst happened.

The name is cool. lol
I like how it connects all the different devices together.
I liked the count down when is sensed an emergency.

It is an interesting idea. I haven't seen anything that does this before.

Easy to get going.
updated by itself.

When asked for suggestions for improvement, some users provided useful suggestions such as the option to be notified when the heart rate was in the warning range, not just in the dangerous range. Another useful suggestion was the addition of a function to email the heart rate history to the users' doctor. One of the common suggestions from the users was to improve the instructions to better explain what the application does and how it works. One suggestion pointed out some confusion with the night mode function where the user was expecting this to alter the UI rather than just the triggering of emergency events. This is a particularly good point and one that could be added to the application quite easily. Given that the user is expected to be sleeping, there is no need for the UI to be displayed in full colour/brightness.



In your own words, what do you think could be improved with the service?

7 responses

It would be nice to have an option to get notified when my heart rate is in the warning level, not just when its in the red zone.
The profile screen has a button and I'm not sure what it does.

The history was useful but It would be great if i could do more with it, maybe email it to a doctor or something.
The smart bit could maybe be explained a bit more.

The watch dies very quickly. It only lasted about 10 hours on a full charge.
The instructions could be a bit clearer.

The tutorial steps could be more detailed. Maybe explaining the smart home stuff a bit better.
Maybe if the night mode turned the screen into grey-scale so it wasn't so bright in the night time.

Maybe add an infared remote so I can control my telly with my phone.
Make it update more regularly.

Some instructions that explain how it works might be useful.
The edit button in the profile page doesn't really explain itself well.

It was very bright in night mode. Maybe i'm confusing it with the night-mode that is built into phones, but I thought it would darken the screen as well.

3.6.3 System Testing

A system test was devised to assess the system's ability to process readings from the smart watch and to receive these readings in the mobile application. The aim of this test is to verify that the various classes and methods work in conjunction with each other to reliably initiate a trigger of the system.

The test was carried out by feeding randomised readings to the smart-watch application and counting those received by the mobile application. This was conducted using the Android monitor in Android Studio to view debug information. The test was performed twice, once with only save heart rate readings and a second time with only excessive readings.

```
public void testData(View view) {
    Random r = new Random();
    int Low = 30;                                          //Lower limit
    int High = 190;                                                //Upper limit
    int randomResult = r.nextInt(High-Low) +Low;           //Random between lower/upper

    Calendar cal = Calendar.getInstance();                 //Formatting the timestamp
    SimpleDateFormat dateFormat = new SimpleDateFormat("dd:MM:yy:HH:mm");
    String timestamp = dateFormat.format(cal.getTime());

    PutDataMapRequest mapReq = PutDataMapRequest.create("/HR-Rate");
    mapReq.getDataMap().putFloat("HR", randomResult);
    mapReq.getDataMap().putString("timestamp", timestamp);
    PutDataRequest req = mapReq.asPutDataRequest();        //Sending map to mobile

    Wearable.DataApi.putDataItem(mGoogleApiClient, req)
            .setResultCallback(. . .{
                  if (dataItemResult.getStatus().isSuccess()) {
                    Log.e(TAG, "Sending DataItem Succeeded: " +randomResult);
                 } else {
                    Log.e(TAG, "Sending DataItem Failed: ");
                 }
            }
       });
}
```

3.7 Research

Research areas for the project included examining the number of elderly and disabled people living in Ireland, as detailed in the information from the Central Statistics Office and compiled by the All Ireland Research Observatory.

Research was also conducted to identify similar services to *RapidARM*, and while some services exist that satisfy some of the requirements, there was difficulty in finding direct comparisons with the *RapidARM* service.

Examples of services offering similar features include *Kardia Mobile* [1], a service that documents a users' heart activity and collects this data to inform the user of potential difficulties and for presenting a doctor with information about a specific situation. The comparison with this service was difficult due to the nature in which *Kardia* is used. This service is intended to be used by a user in a deliberate fashion in which the user must open the application and actively place their fingers on the sensor pad to obtain a reading.

The *RapidARM* service differs from this by being a service that operates in the background, only requiring input from the user when a situation is detected in which the user must be informed. While *Kardia* requires a user to be an active participant in the monitoring of their heart activity, *RapidARM* is capable of being used in a situation where the user may have no knowledge of the service, and is instead administered by a carer or family member.

Other services researched include *AED Alert* [7], a mobile application that sends out a beacon for local helpers when a cardiac arrest is reported. This application employs useful features that could be added to *RapidARM* in the future. The service requires that a user is already aware of the incident and that there are nearby users of the application that are equipped to assist in this type of emergency. This service does not offer any kind of sensor monitoring, but combined with the *Kardia* service, could offer a feature set like that of RapidARM.

3.7.1 Medical Research

It was identified that in order to provide such a service in the medical field, advice on the various parameters involved in the project would need to be sought from medical professionals. The project received invaluable input from two doctors, Professor. Richard Costello and Dr. Abir Alsaid, in the field of general medicine, sleep and respiratory, to gauge opinions on the feasibility of the system and to seek specialist input in relation to identifying key indicators and in interpreting heart rate information.

An online survey was then produced to gather information and to document it for use later in the application. The completed surveys can be found in the appendix of this document.

The survey consists of 9 statements on a 1 to 5 scale in which the subject can strongly disagree or strongly agree. 2 further questions ask the subject to estimate at what point they would have cause for concern in relation to a patient’s fast heart rate and their slow heart rate.

The survey then asks for specific input from the doctor on any other factors that they would consider possible indications of cardiac arrest or heart attack, which have not been included in the prior questions.

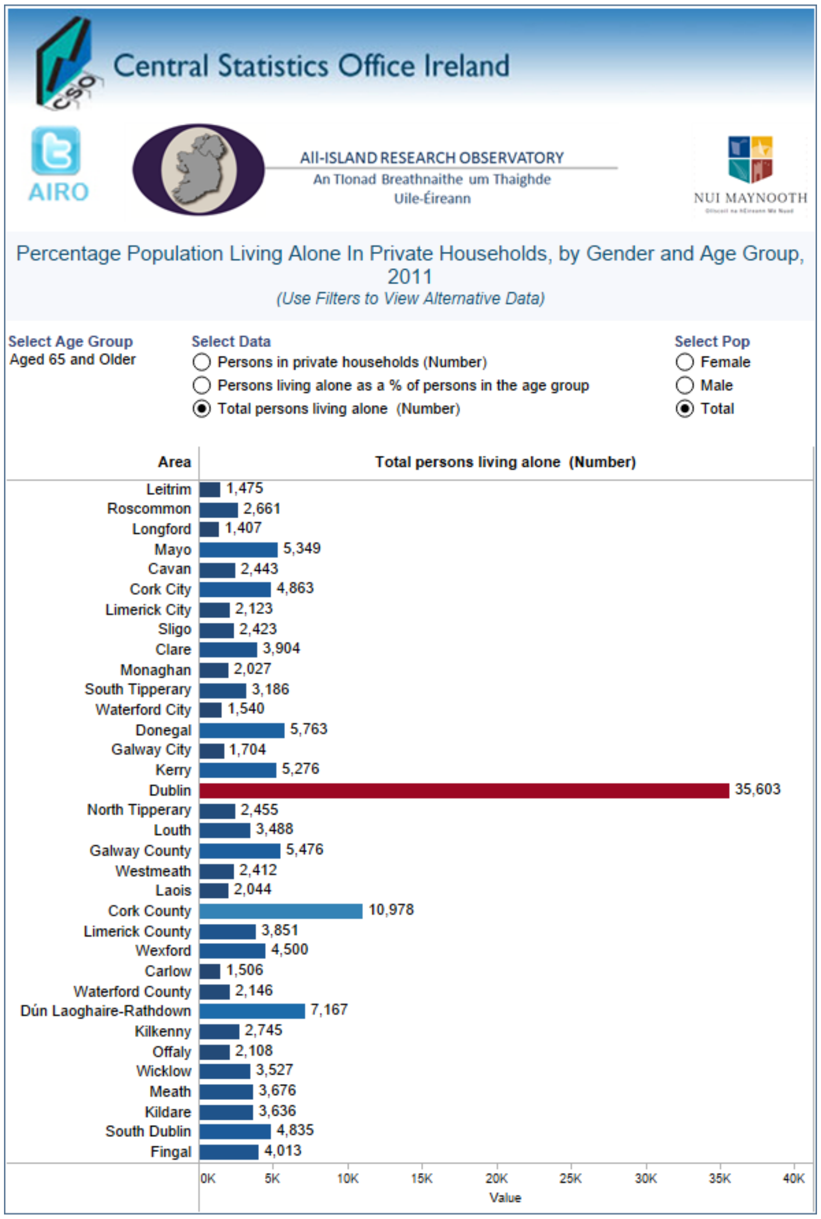
Finally, the survey asks for any comments and recommendations from the doctor, to catch any noteworthy information that had not been addressed.

3.7.2 Potential Userbase

3.7.2.1 Persons over 65 living alone

According to the 2011 Census statistics, the number of elderly people living alone in Ireland has reached over 950’000.

Figures showing the number of people over 65 living alone by county.


Central Statistics Office Ireland
AIRO
All-ISLAND RESEARCH OBSERVATORY
An Tlonad Breathnaithe um Thaighde Uile-Éireann
NUI MAYNOOTH

Percentage Population Living Alone In Private Households, by Gender and Age Group, 2011

(Use Filters to View Alternative Data)

Select Age Group: Aged 65 and Older

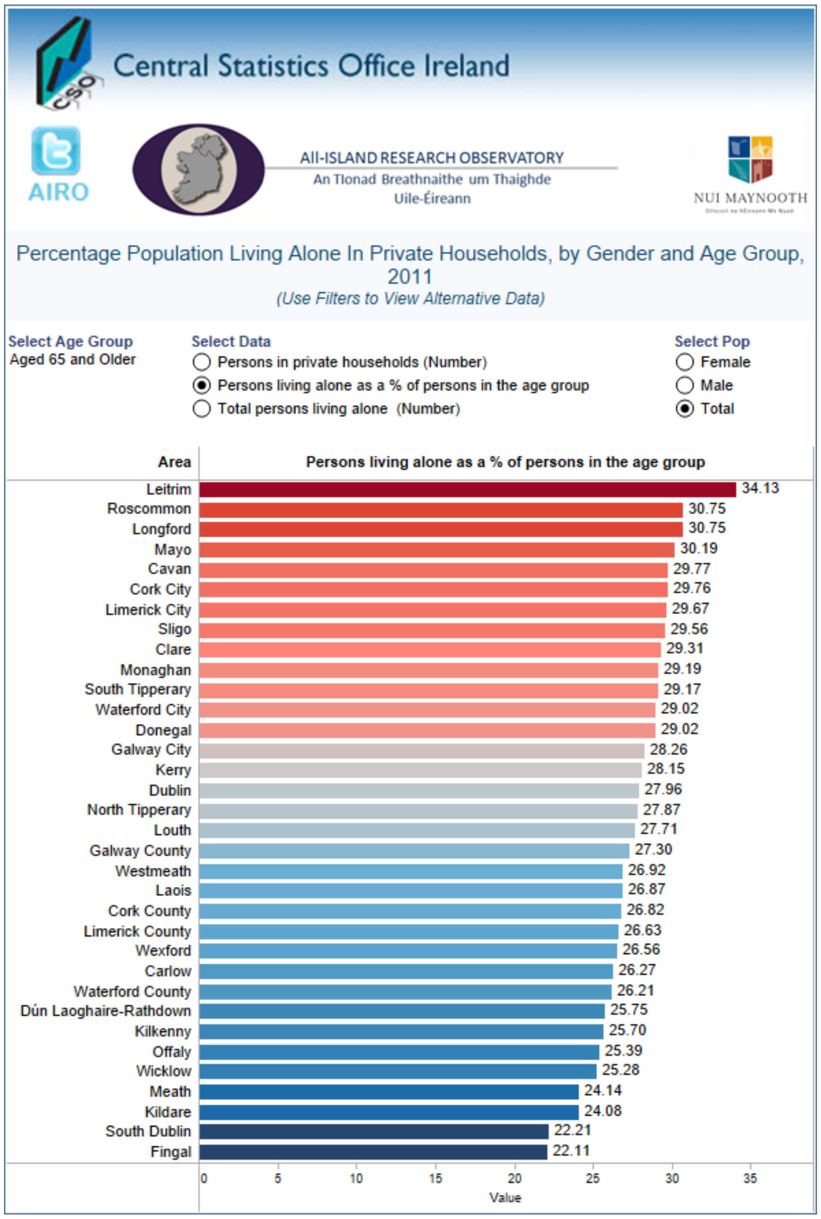
Select Data:
- ○ Persons in private households (Number)
- ○ Persons living alone as a % of persons in the age group
- ● Total persons living alone (Number)

Select Pop:
- ○ Female
- ○ Male
- ● Total

Source: All-Island Research Observatory. (2014)

3.7.2.2 Percentage of over 65's who are living alone

Figures showing the percentage of people over 65 living alone by county.



Central Statistics Office Ireland

AIRO

All-ISLAND RESEARCH OBSERVATORY
An Tlonad Breathnaithe um Thaighde Uile-Éireann

NUI MAYNOOTH

Percentage Population Living Alone In Private Households, by Gender and Age Group, 2011
(Use Filters to View Alternative Data)

Select Age Group: Aged 65 and Older

Select Data:
- ◯ Persons in private households (Number)
- [✓ Persons living alone as a % of persons in the age group]
- ◯ Total persons living alone (Number)

Select Pop:
- ◯ Female
- ◯ Male
- [✓ Total]

Source: All-Island Research Observatory. (2014)

4 Conclusions

4.1 Advantages

The project can be viewed as a safety net that provides a level of independence to the user and assists vulnerable people to remain living at home for longer, while also providing some peace of mind to relatives or carers. This can help in providing a better quality of living in the later years of a users' life, where alternative living arrangements would be a path for many. Indirectly, this can reduce costs to the user and their family if the user opts to remain living at home instead of moving into assisted living arrangements.

From the user perspective, the service provides an ever-present method for communication with the outside world that is particularly important where mobility issues are present. The smart-home features provide comfort and ease of access, while providing wide ranging opportunities for automation as an evolution of the system.

The system exists on hardware that is easy to obtain and is relatively inexpensive when compared with medical devices that perform similar tasks. The system could be offered as a service in which the user provides their own equipment and installs the *RapidARM* application on this hardware.

4.2 Further Development Opportunities

The system provides an opportunity for further research and development in the medical IoT field with specific reference to the elderly and disabled. Such research areas may include technology such as occupancy, temperature, motion sensors and electricity usage monitoring that could provide an external entity with detailed information on the activity of the user or patient. This would reduce the reliance of the system on a single sensor, mounted to the users arm and would provide a much greater insight into the activity level of that user, while being more fault tolerant than the existing system.

The evolution of this system could see it developed into a suite of tools that would be used by an organisation responsible for assisting multiple patients, such as a carer or to a doctor who could monitor multiple patients at the same time. This could be implemented in the homes of individuals, or in the rooms of patients in independent living facilities.

Additionally, the use of the Firebase cloud database provides many opportunities for automation and artificial intelligence by making use of Firebase Cloud Functions. In its current state, Firebase cloud functions can be used to activate and deactivate electrical devices over the network. This could be useful in regulating the temperature in the home of a user, or to turn off lighting and entertainment when the watch has detected that the user is asleep, or possibly the randomising of lighting when the user is away, to give the impression that the home is not empty.

4.3 Limitations

In its current form, the system is heavily reliant on a single sensor to provide the required data for the system to function. Due to cost and time constraints, it was not possible to obtain additional equipment that could duplicate the functionality of the smartwatch, enabling more accurate and reliable monitoring of a user.

The smartwatch requires the user to have mounted the strap firmly, without excess movement so that the optical sensor can obtain a reliable reading. Given greater resources and budget, it would be preferable to find a sensor that is better suited to obtaining these readings without being uncomfortable for the user.

5 References

[1] Alivecor.com. (2016). *Kardia Mobile*. [online] Available at: https://www.alivecor.com/en/ [Accessed 4 Nov. 2016].

[2] APIs, A. (2017). *Android 6.0 APIs | Android Developers*. [online] Developer.android.com. Available at: https://developer.android.com/about/versions/marshmallow/android-6.0.html [Accessed 15 Mar. 2017].

[3] All-Island Research Observatory. (2014). *Population Aged 65 and Over, 2011*. [online] Available at: http://airo.maynoothuniversity.ie//external-content/table-16-population-aged-65-and-over-2011 [Accessed 23 Sep. 2016].

[4] Bluetooth, D. (2016). *Debugging over Bluetooth | Android Developers*. [online] Developer.android.com. Available at: https://developer.android.com/training/wearables/apps/bt-debugging.html [Accessed 25 Sep. 2016].

[5] Disability-federation.ie. (2012). *Disability in Ireland: Some Facts and Figures*. [online] Available at: http://www.disability-federation.ie/index.php?uniqueID=10598 [Accessed 4 Nov. 2016].

[6] GitHub. (2016). TangoAgency/material-intro-screen. [online] Available at: https://github.com/TangoAgency/material-intro-screen/ [Accessed 28 Feb. 2017].

[7] Hartveilig Wonen - AED Alert System. (2016). *Hartveilig Wonen*. [online] Available at: http://heartsafeliving.com/ [Accessed 13 Nov. 2016].

[8] Developer.android.com. (2016). *BluetoothManager | Android Developers*. [online] Available at: https://developer.android.com/reference/android/bluetooth/BluetoothManager.html [Accessed 2 Nov. 2016]

[9] Developer.android.com. (2016). *Intent | Android Developers*. [online] Available at: https://developer.android.com/reference/android/content/Intent.html#ACTION_CALL [Accessed 2 Nov. 2016].

[10] Developer.android.com. (2016). *AudioManager | Android Developers*. [online] Available at: https://developer.android.com/reference/android/media/AudioManager.html [Accessed 2 Nov. 2016].

[11] Firebase. (2017). *Firebase Realtime Database | Firebase*. [online] Available at: https://firebase.google.com/docs/database/ [Accessed 7 Feb. 2017].

[12] Jahoda, P. (2014). *PhilJay/MPAndroidChart*. [online] GitHub. Available at: https://github.com/PhilJay/MPAndroidChart [Accessed 16 Feb. 2017].

[13] Khaleel, S. (2016). *Android How to Add Fingerprint Authentication – AndroidHive*. [online] Androidhive.info. Available at: https://www.androidhive.info/2016/11/android-add-fingerprint-authentication/ [Accessed 12 Mar. 2017].

[14] Material design guidelines. (2016). *Introduction - Material design - Material design guidelines*. [online] Available at: https://material.io/guidelines/#introduction-goals [Accessed 10 Feb. 2017].

[15] P. Regmi, P. (2016). *Android Material Design Profile Screen XML UI Design*. [online] Viral Android – Tutorials, Examples, UX/UI Design. Available at: http://www.viralandroid.com/2016/03/android-material-design-profile-screen-xml-ui-design.html [Accessed 15 Feb. 2017].

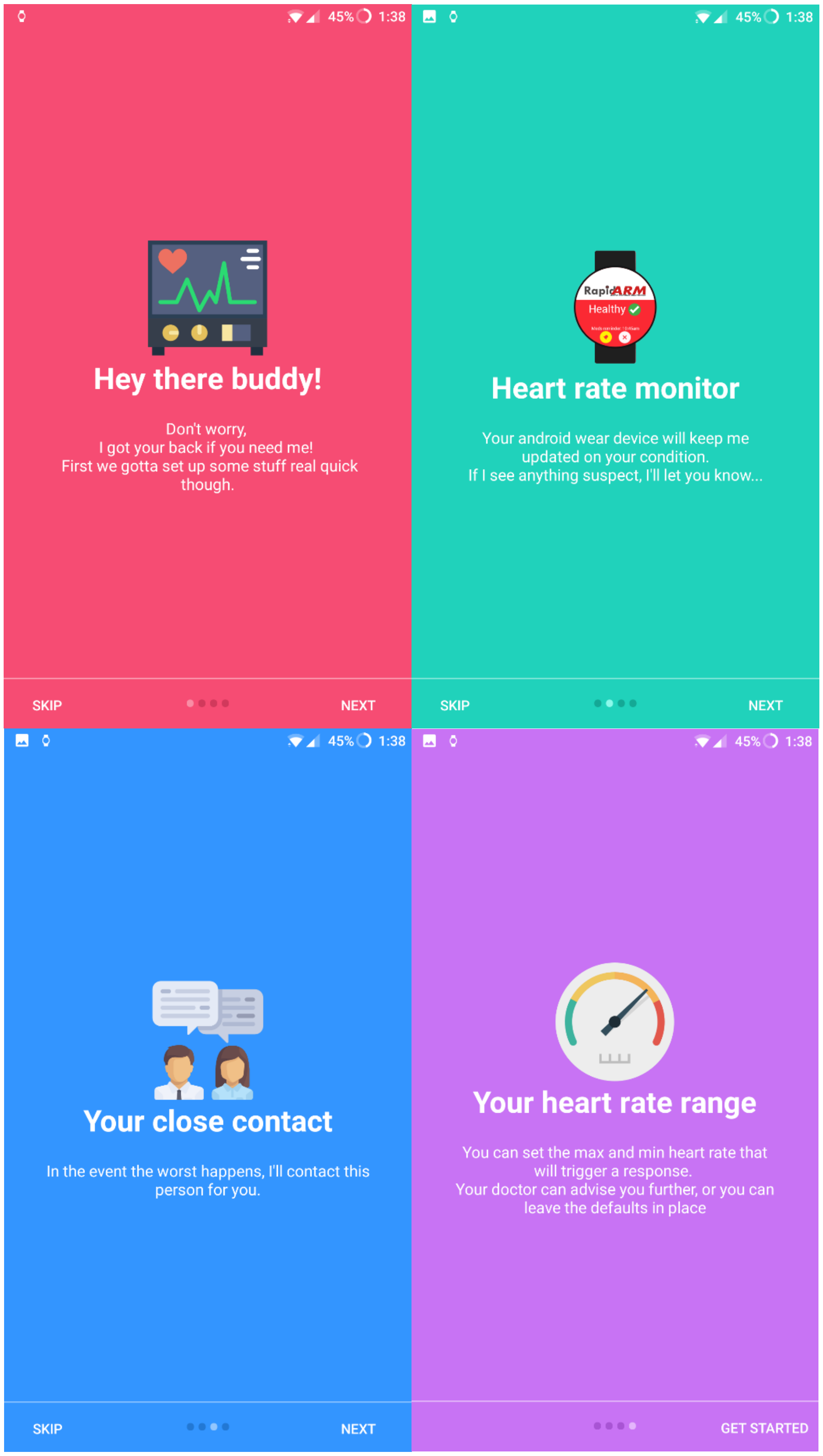
[16] Penz, M. (2014). *mikepenz/MaterialDrawer*. [online] GitHub. Available at: https://github.com/mikepenz/MaterialDrawer [Accessed 5 Feb. 2017].

[17] Tamada, R. (2016). *Android How to Build Intro Slider for your App – AndroidHive*. [online] Androidhive.info. Available at: http://www.androidhive.info/2016/05/android-build-intro-slider-app/ [Accessed 17 Oct. 2016].

6 Appendix

6.1 Application Screenshots

6.1.1 Introduction Slider UI



Hey there buddy!

Don't worry,
I got your back if you need me!
First we gotta set up some stuff real quick though.

Heart rate monitor

Your android wear device will keep me updated on your condition.
If I see anything suspect, I'll let you know...

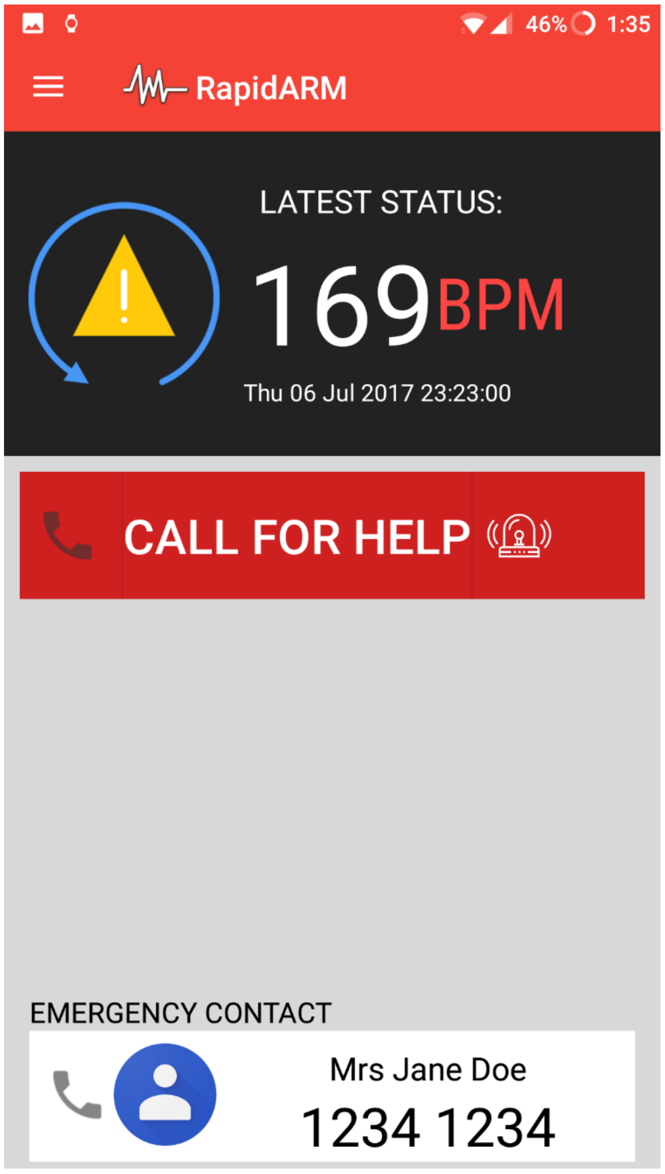
Your close contact

In the event the worst happens, I'll contact this person for you.

Your heart rate range

You can set the max and min heart rate that will trigger a response.
Your doctor can advise you further, or you can leave the defaults in place

6.1.2 Main User Interface



LATEST STATUS:

169BPM

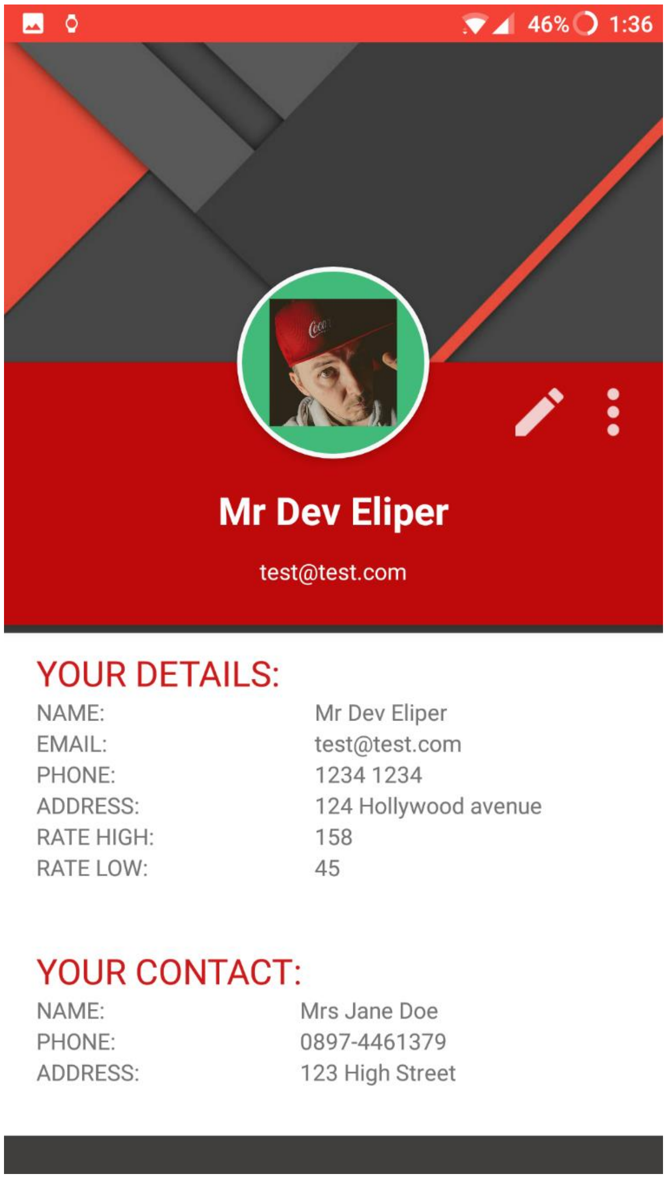
Thu 06 Jul 2017 23:23:00

EMERGENCY CONTACT

Mrs Jane Doe

1234 1234

6.1.3 Profile Activity



Mr Dev Eliper

test@test.com

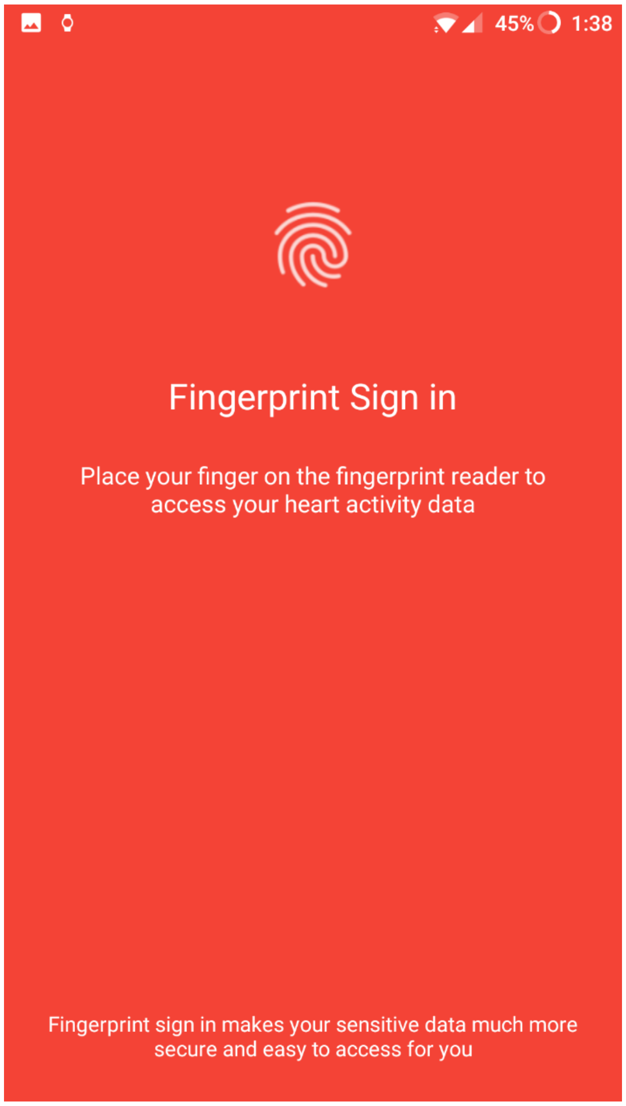
YOUR DETAILS:

NAME:	Mr Dev Eliper
EMAIL:	test@test.com
PHONE:	1234 1234
ADDRESS:	124 Hollywood avenue
RATE HIGH:	158
RATE LOW:	45

YOUR CONTACT:

NAME:	Mrs Jane Doe
PHONE:	0897-4461379
ADDRESS:	123 High Street

6.1.4 Fingerprint Authentication UI



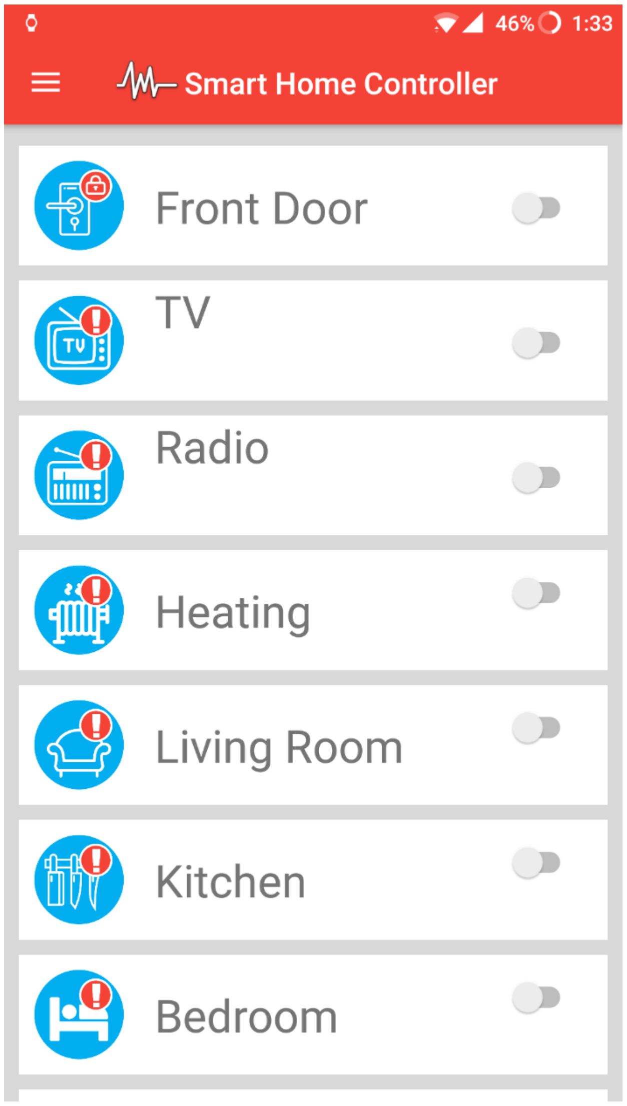
45% 1:38

Fingerprint Sign in

Place your finger on the fingerprint reader to access your heart activity data

Fingerprint sign in makes your sensitive data much more secure and easy to access for you

6.1.5 Smart-home Control UI



46% 1:33

Smart Home Controller

Front Door

TV

Radio

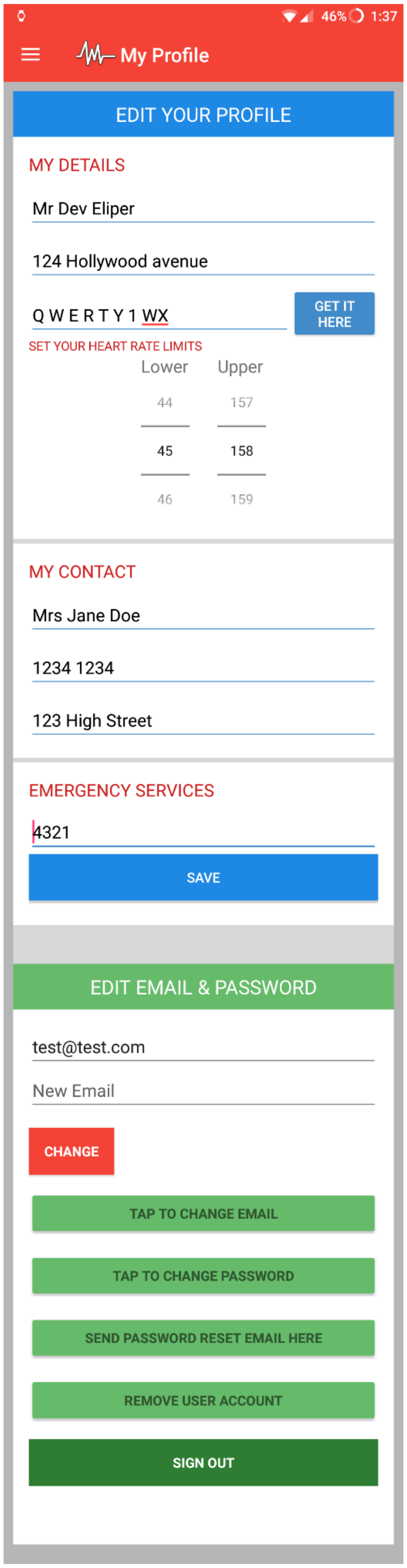
Heating

Living Room

Kitchen

Bedroom

6.1.6 Profile Edit Activity



46% 1:37

My Profile

EDIT YOUR PROFILE

MY DETAILS

Mr Dev Eliper

124 Hollywood avenue

Q W E R T Y 1 WX

GET IT HERE

SET YOUR HEART RATE LIMITS

Lower	Upper
44	157
45	158
46	159

MY CONTACT

Mrs Jane Doe

1234 1234

123 High Street

EMERGENCY SERVICES

4321

SAVE

EDIT EMAIL & PASSWORD

test@test.com

New Email

CHANGE

TAP TO CHANGE EMAIL

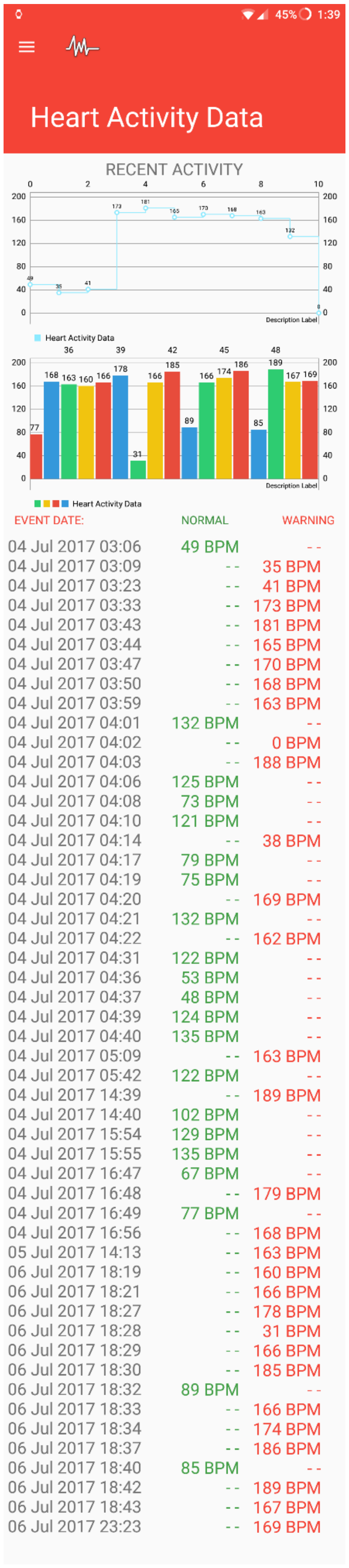
TAP TO CHANGE PASSWORD

SEND PASSWORD RESET EMAIL HERE

REMOVE USER ACCOUNT

SIGN OUT

6.1.7 Heart Rate History UI



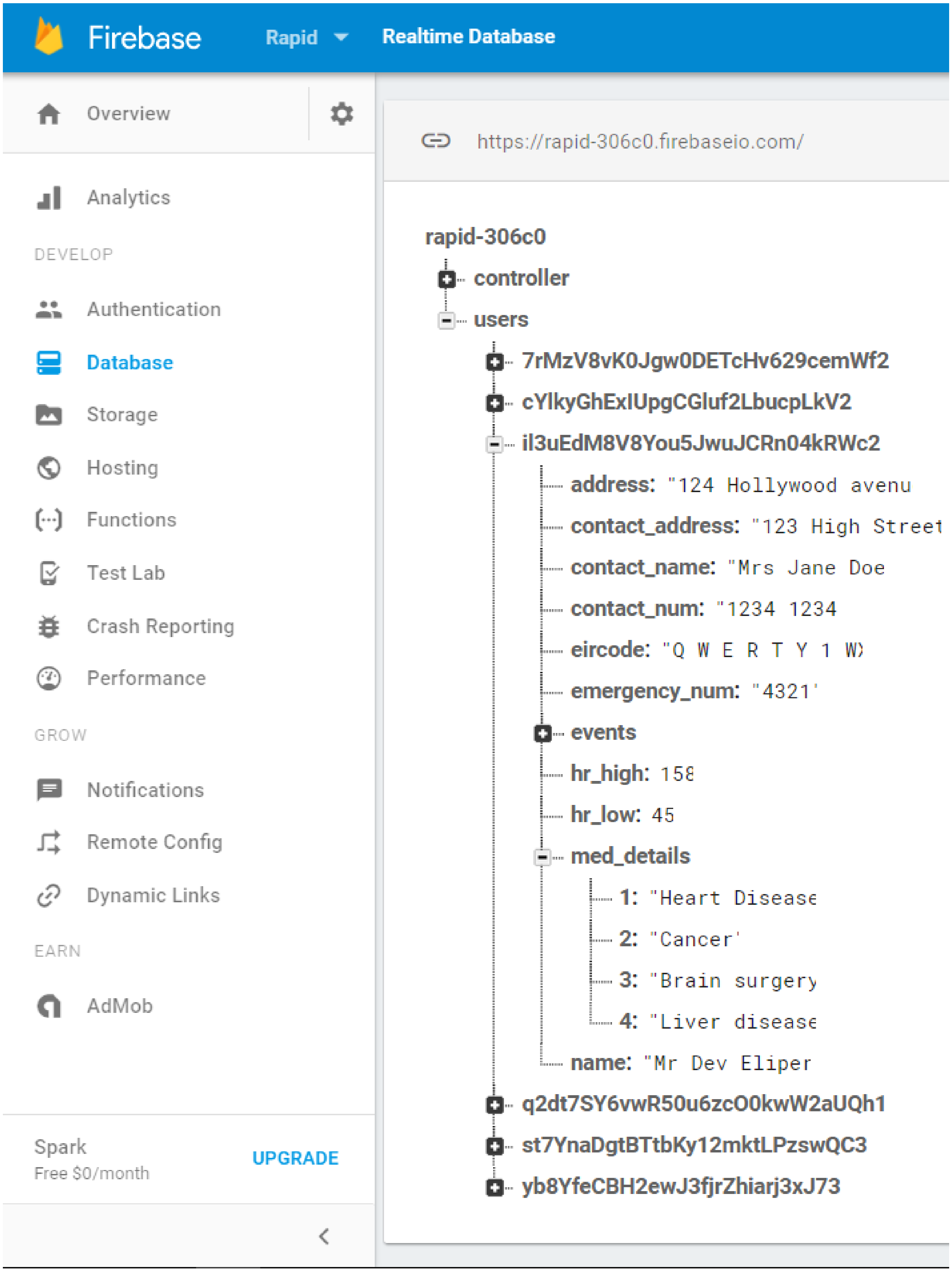
45% 1:39

Heart Activity Data

RECENT ACTIVITY

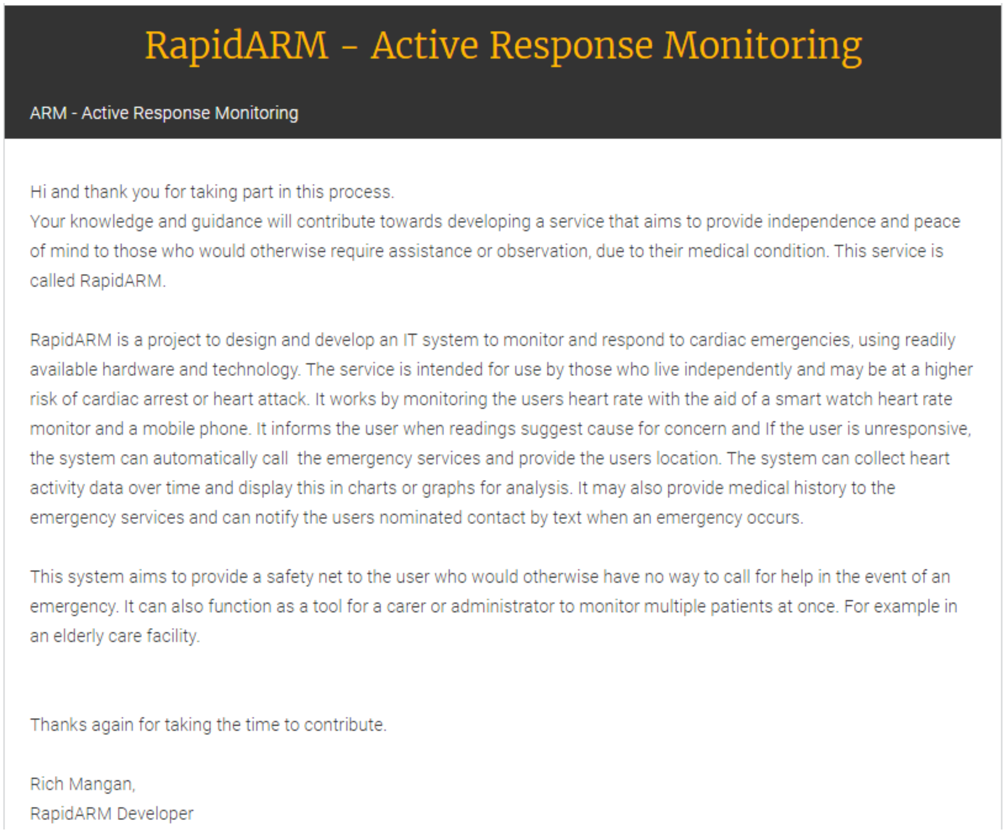
EVENT DATE:	NORMAL	WARNING
04 Jul 2017 03:06	49 BPM	- -
04 Jul 2017 03:09	- -	35 BPM
04 Jul 2017 03:23	- -	41 BPM
04 Jul 2017 03:33	- -	173 BPM
04 Jul 2017 03:43	- -	181 BPM
04 Jul 2017 03:44	- -	165 BPM
04 Jul 2017 03:47	- -	170 BPM
04 Jul 2017 03:50	- -	168 BPM
04 Jul 2017 03:59	- -	163 BPM
04 Jul 2017 04:01	132 BPM	- -
04 Jul 2017 04:02	- -	0 BPM
04 Jul 2017 04:03	- -	188 BPM
04 Jul 2017 04:06	125 BPM	- -
04 Jul 2017 04:08	73 BPM	- -
04 Jul 2017 04:10	121 BPM	- -
04 Jul 2017 04:14	- -	38 BPM
04 Jul 2017 04:17	79 BPM	- -
04 Jul 2017 04:19	75 BPM	- -
04 Jul 2017 04:20	- -	169 BPM
04 Jul 2017 04:21	132 BPM	- -
04 Jul 2017 04:22	- -	162 BPM
04 Jul 2017 04:31	122 BPM	- -
04 Jul 2017 04:36	53 BPM	- -
04 Jul 2017 04:37	48 BPM	- -
04 Jul 2017 04:39	124 BPM	- -
04 Jul 2017 04:40	135 BPM	- -
04 Jul 2017 05:09	- -	163 BPM
04 Jul 2017 05:42	122 BPM	- -
04 Jul 2017 14:39	- -	189 BPM
04 Jul 2017 14:40	102 BPM	- -
04 Jul 2017 15:54	129 BPM	- -
04 Jul 2017 15:55	135 BPM	- -
04 Jul 2017 16:47	67 BPM	- -
04 Jul 2017 16:48	- -	179 BPM
04 Jul 2017 16:49	77 BPM	- -
04 Jul 2017 16:56	- -	168 BPM
05 Jul 2017 14:13	- -	163 BPM
06 Jul 2017 18:19	- -	160 BPM
06 Jul 2017 18:21	- -	166 BPM
06 Jul 2017 18:27	- -	178 BPM
06 Jul 2017 18:28	- -	31 BPM
06 Jul 2017 18:29	- -	166 BPM
06 Jul 2017 18:30	- -	185 BPM
06 Jul 2017 18:32	89 BPM	- -
06 Jul 2017 18:33	- -	166 BPM
06 Jul 2017 18:34	- -	174 BPM
06 Jul 2017 18:37	- -	186 BPM
06 Jul 2017 18:40	85 BPM	- -
06 Jul 2017 18:42	- -	189 BPM
06 Jul 2017 18:43	- -	167 BPM
06 Jul 2017 23:23	- -	169 BPM

6.1.8 Cloud Database Console



```
https://rapid-306c0.firebaseio.com/

rapid-306c0
  controller
  users
    7rMzV8vK0Jgw0DETcHv629cemWf2
    cYlkyGhExIUpgCGluf2LbucpLkV2
    il3uEdM8V8You5JwuJCRn04kRWc2
      address: "124 Hollywood avenu
      contact_address: "123 High Street
      contact_name: "Mrs Jane Doe
      contact_num: "1234 1234
      eircode: "Q W E R T Y 1 W
      emergency_num: "4321'
      events
      hr_high: 158
      hr_low: 45
      med_details
        1: "Heart Disease
        2: "Cancer'
        3: "Brain surgery
        4: "Liver disease
      name: "Mr Dev Eliper
    q2dt7SY6vwR50u6zcO0kwW2aUQh1
    st7YnaDgtBTtbKy12mktLPzswQC3
    yb8YfeCBH2ewJ3fjrZhiarj3xJ73
```

6.2 *Medical Professional Survey Template*



RapidARM – Active Response Monitoring

ARM - Active Response Monitoring

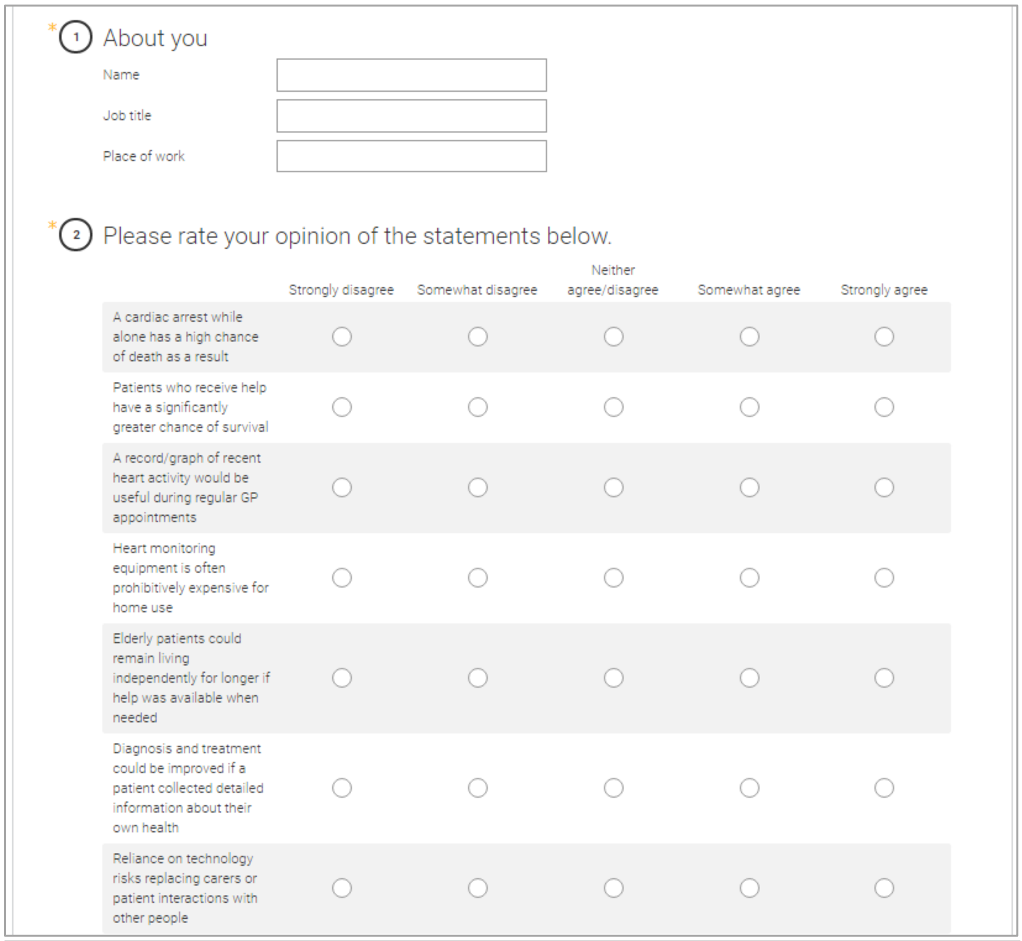
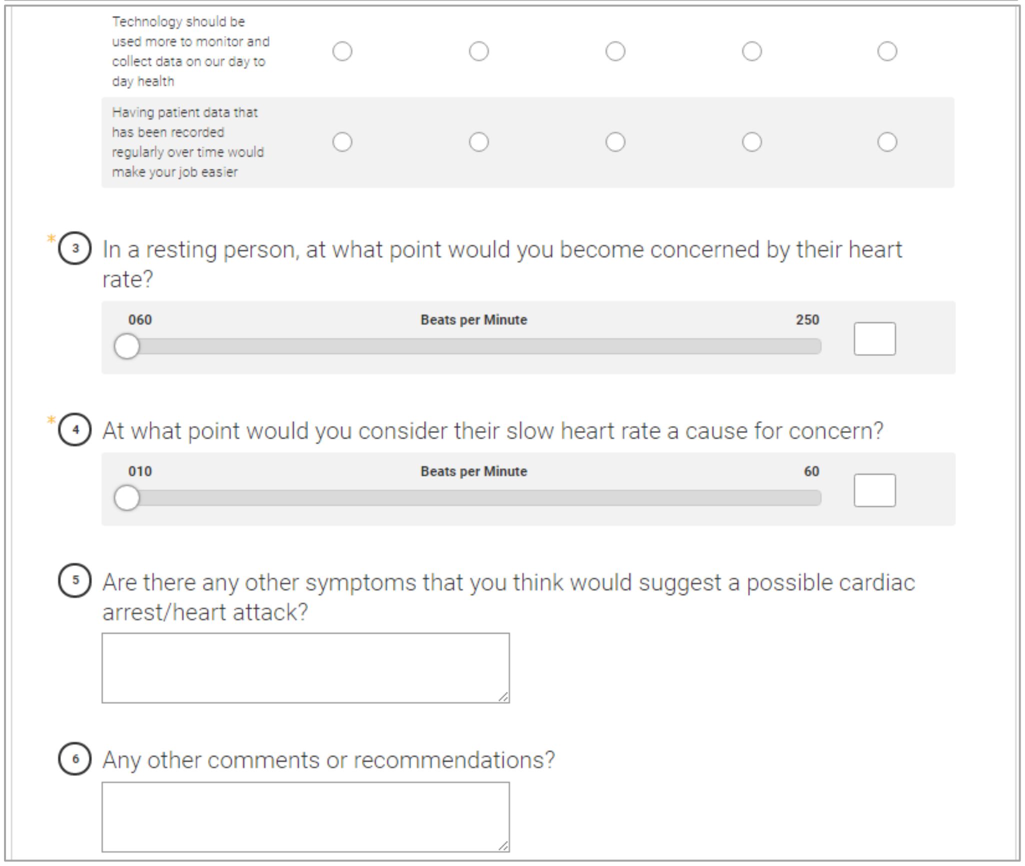
Hi and thank you for taking part in this process.
Your knowledge and guidance will contribute towards developing a service that aims to provide independence and peace of mind to those who would otherwise require assistance or observation, due to their medical condition. This service is called RapidARM.

RapidARM is a project to design and develop an IT system to monitor and respond to cardiac emergencies, using readily available hardware and technology. The service is intended for use by those who live independently and may be at a higher risk of cardiac arrest or heart attack. It works by monitoring the users heart rate with the aid of a smart watch heart rate monitor and a mobile phone. It informs the user when readings suggest cause for concern and If the user is unresponsive, the system can automatically call the emergency services and provide the users location. The system can collect heart activity data over time and display this in charts or graphs for analysis. It may also provide medical history to the emergency services and can notify the users nominated contact by text when an emergency occurs.

This system aims to provide a safety net to the user who would otherwise have no way to call for help in the event of an emergency. It can also function as a tool for a carer or administrator to monitor multiple patients at once. For example in an elderly care facility.

Thanks again for taking the time to contribute.

Rich Mangan,
RapidARM Developer

* 1 About you

Name	
Job title	
Place of work	

* 2 Please rate your opinion of the statements below.

	Strongly disagree	Somewhat disagree	Neither agree/disagree	Somewhat agree	Strongly agree
A cardiac arrest while alone has a high chance of death as a result	○	○	○	○	○
Patients who receive help have a significantly greater chance of survival	○	○	○	○	○
A record/graph of recent heart activity would be useful during regular GP appointments	○	○	○	○	○
Heart monitoring equipment is often prohibitively expensive for home use	○	○	○	○	○
Elderly patients could remain living independently for longer if help was available when needed	○	○	○	○	○
Diagnosis and treatment could be improved if a patient collected detailed information about their own health	○	○	○	○	○
Reliance on technology risks replacing carers or patient interactions with other people	○	○	○	○	○
Technology should be used more to monitor and collect data on our day to day health	○	○	○	○	○
Having patient data that has been recorded regularly over time would make your job easier	○	○	○	○	○

* 3 In a resting person, at what point would you become concerned by their heart rate?

060 **Beats per Minute** **250**

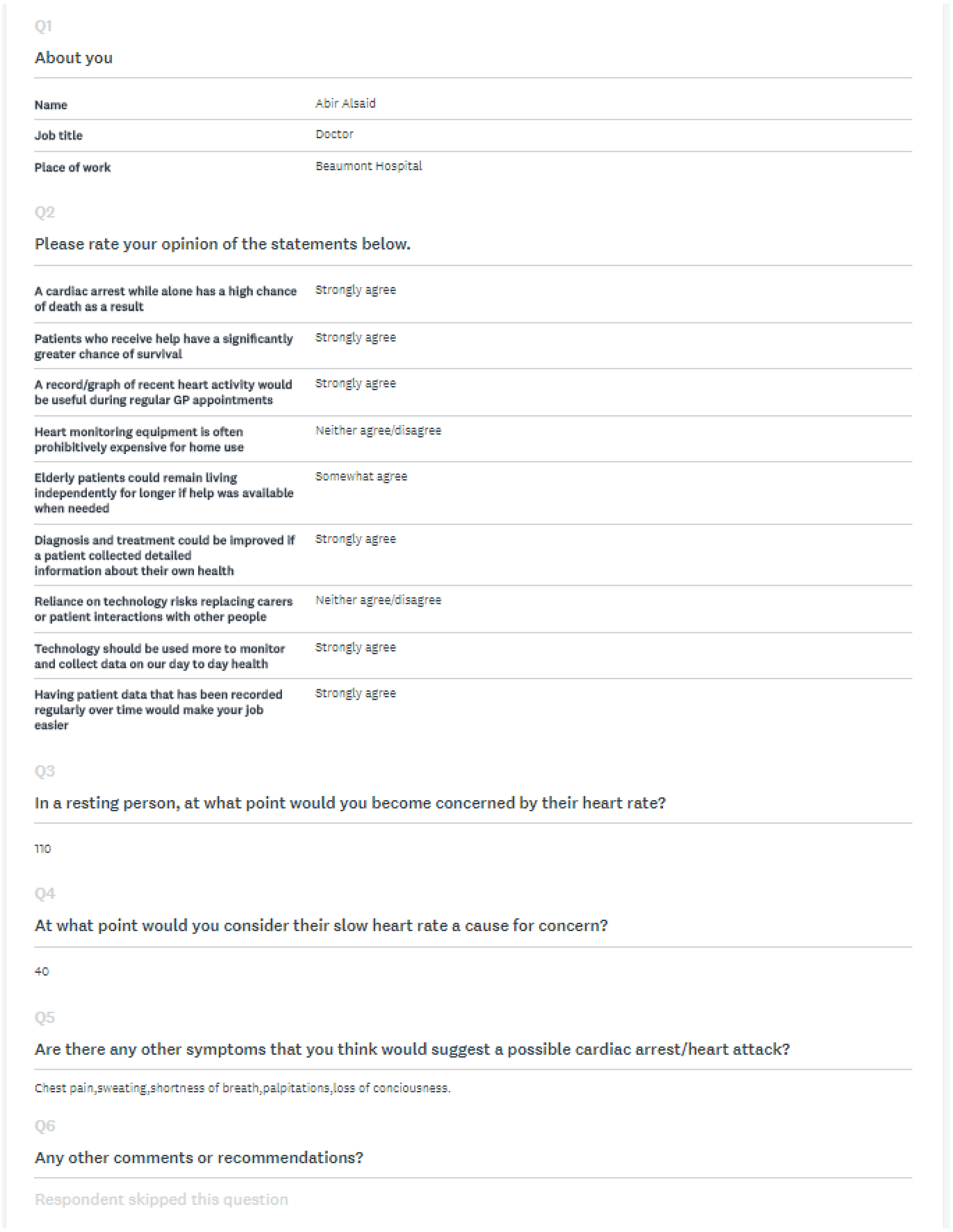
* 4 At what point would you consider their slow heart rate a cause for concern?

010 **Beats per Minute** **60**

5 Are there any other symptoms that you think would suggest a possible cardiac arrest/heart attack?

6 Any other comments or recommendations?

6.3 Medical Survey Responses



Q1

About you

Name	Abir Alsaid
Job title	Doctor
Place of work	Beaumont Hospital

Q2

Please rate your opinion of the statements below.

A cardiac arrest while alone has a high chance of death as a result	Strongly agree
Patients who receive help have a significantly greater chance of survival	Strongly agree
A record/graph of recent heart activity would be useful during regular GP appointments	Strongly agree
Heart monitoring equipment is often prohibitively expensive for home use	Neither agree/disagree
Elderly patients could remain living independently for longer if help was available when needed	Somewhat agree
Diagnosis and treatment could be improved if a patient collected detailed information about their own health	Strongly agree
Reliance on technology risks replacing carers or patient interactions with other people	Neither agree/disagree
Technology should be used more to monitor and collect data on our day to day health	Strongly agree
Having patient data that has been recorded regularly over time would make your job easier	Strongly agree

Q3

In a resting person, at what point would you become concerned by their heart rate?

110

Q4

At what point would you consider their slow heart rate a cause for concern?

40

Q5

Are there any other symptoms that you think would suggest a possible cardiac arrest/heart attack?

Chest pain,sweating,shortness of breath,palpitations,loss of conciousness.

Q6

Any other comments or recommendations?

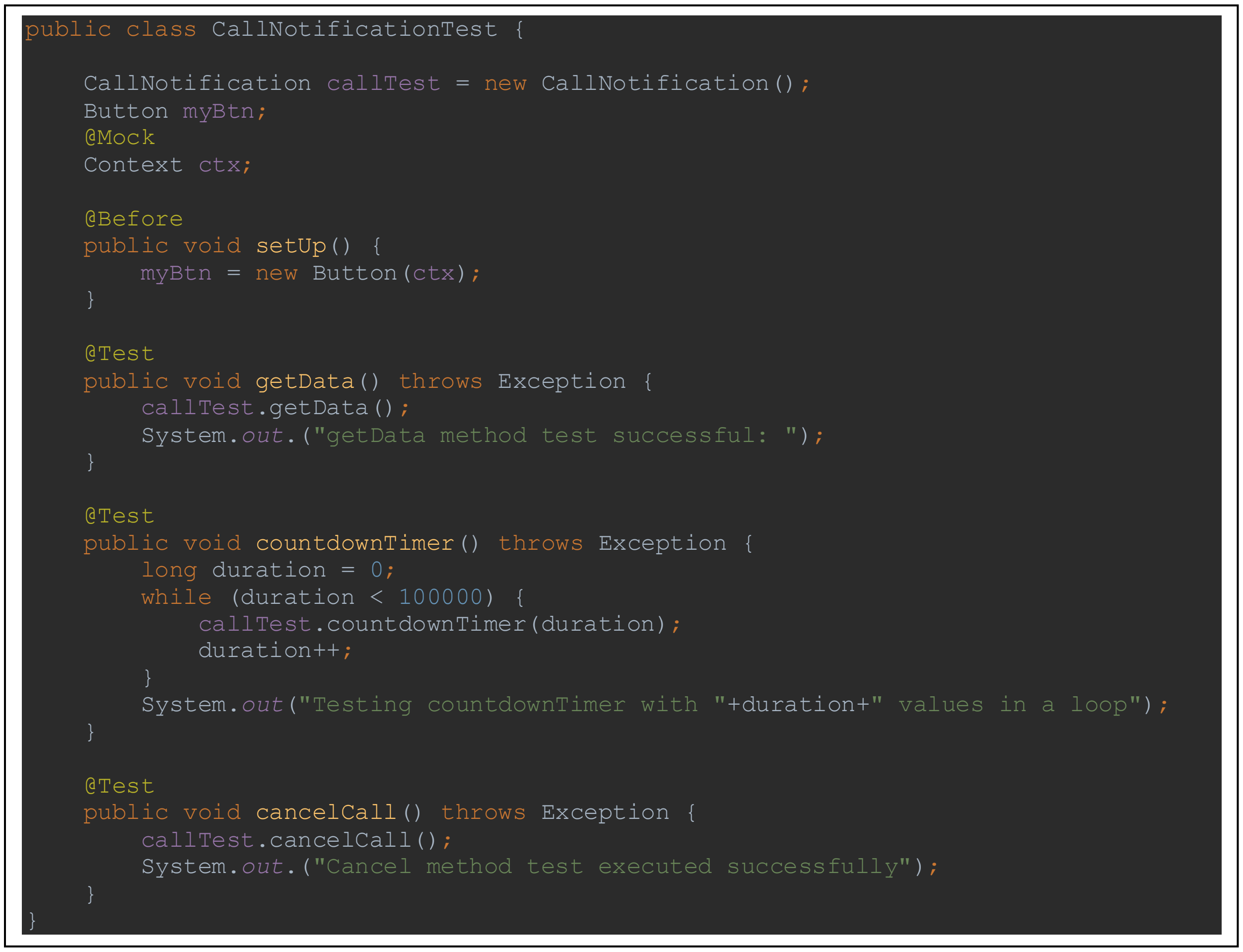
Respondent skipped this question

6.4 Unit Test Classes

6.4.1 MyWearServiceTest

```java
@RunWith(MockitoJUnitRunner.class)
public class MyWearServiceTest {

    private static final String TAG = "MyWearServiceTest";

    long rate;
    long upper = 110;
    long lower = 40;

    MyWearService mWear = new MyWearService();

    @Mock
    private Context ctx;
    private Intent intent;

    @Test
    public void checkRate() throws Exception {

        for (int i = (int) lower; i < upper; i++) {
            rate = i;
            Boolean result = mWear.checkRate(rate, upper, lower);
            assertFalse(result);
        }
        System.out.println("checkRate test: Tested up to: " + rate + " BPM");
    }

    @Test
    public void sendResult() throws Exception {
        boolean result = mWear.sendResult(rate, "test");

        assertFalse(result);
        System.out.println("sendResult test. Passing in " + rate + " and 'test'");
    }

    @Test
    public void emergencyCall() throws Exception {

        boolean result;
        for (int i = (int) lower; i < upper; i++) {
            rate = i;
            result = mWear.emergencyCall(rate);
            assertTrue(result);
        }
        System.out.println("emergencyCall method tested up to: " + rate + " BPM");
    }
}
```

6.4.2 CallNotificationTest



```
public class CallNotificationTest {

    CallNotification callTest = new CallNotification();
    Button myBtn;
    @Mock
    Context ctx;

    @Before
    public void setUp() {
        myBtn = new Button(ctx);
    }

    @Test
    public void getData() throws Exception {
        callTest.getData();
        System.out.("getData method test successful: ");
    }

    @Test
    public void countdownTimer() throws Exception {
        long duration = 0;
        while (duration < 100000) {
            callTest.countdownTimer(duration);
            duration++;
        }
        System.out("Testing countdownTimer with "+duration+" values in a loop");
    }

    @Test
    public void cancelCall() throws Exception {
        callTest.cancelCall();
        System.out.("Cancel method test executed successfully");
    }
}
```

6.4.3 ControllerActivityTest

```java
public class ControllerActivityTest {

    ControllerActivity controlTest = new ControllerActivity();

    @Test
    public void getFirebaseData() throws Exception {
        Boolean success = controlTest.getFirebaseData();
        System.out.println("Test passed with return of: " + success);
    }

    @Test
    public void setSwitchStates() throws Exception {
        Boolean success = controlTest.setSwitchStates();
        System.out.println("setSwitchStates method test: Response was " + success);
    }

    @Test
    public void sendNewToggleState() throws Exception {

        SecureRandom random = new SecureRandom();
        Random randBool = new Random();

        long iterator = 0;
        while (iterator < 100000){
            String randString = new BigInteger(130, random).toString(32);
            Boolean switchState = randBool.nextBoolean();

            System.out.println("Random string: " + randString);
            System.out.println("Random Boolean: " + switchState);

            controlTest.sendNewToggleState(randString, switchState);
            iterator++;
        }
        System.out.println("Test succeeded with  " + iterator + " calls");
    }
}
```

6.4.4 GraphActivityTest

```java
public class GraphActivityTest {

    GraphActivity graphTest = new GraphActivity();


    @Test
    public void buildGraphs() throws Exception {
        Boolean result = graphTest.buildGraphs();
        System.out.println("Test returned: " + result);
    }

    @Test
    public void updateCharts() throws Exception {
        long iterator = 0;
        while (iterator < 100000){
            graphTest.updateCharts();
            iterator++;
        }
        System.out.println("Testing updateCharts method with: "+iterator +" calls");
    }
}
```

6.4.5 LoudspeakerActivityTest

```java
public class LoudspeakerActivityTest {

    LoudspeakerActivity speakerTest = new LoudspeakerActivity();

    @Test
    public void getData() throws Exception {
        boolean result = speakerTest.getData();
        System.out.println("Test succeeded with return of: " + result);
    }

    @Test
    public void openFrontDoor() throws Exception {
        Boolean result = speakerTest.openFrontDoor();
        System.out.println("Test succeeded with a return of: " + result);
    }

    @Test
    public void startSound() throws Exception {
        long iterator = 0;
        while (iterator < 100000) {
            speakerTest.startSound();
            iterator++;
        }
        System.out.println("Test succeeded after " + iterator + " iterations");

    }

}
```

6.4.6 MainActivityTest

```java
public class MainActivityTest {

    MainActivity mainTest = new MainActivity();

    @Test
    public void updateGauge() throws Exception {

        SecureRandom random = new SecureRandom();

        long iterator = 0;
        while (iterator < 100000){
            Long randLong = random.nextLong();
            mainTest.updateGauge(randLong);
            iterator++;
        }
        System.out.println("Test succeeded with  " + iterator + " calls");
    }

    @Test
    public void callEmergency() throws Exception {
        mainTest.callEmergency();
    }

    @Test
    public void callPerson() throws Exception {
        mainTest.callPerson();
    }

    @Test
    public void makeCall() throws Exception {

        SecureRandom random = new SecureRandom();

        long iterator = 0;
        while (iterator < 100000){
            String randString = new BigInteger(130, random).toString(32);
            mainTest.makeCall(randString);
            iterator++;
        }
    }
}
```

6.5 Usability test survey

Usability testing survey

Usability testing the RapidARM application.

* Required

What was your first impression of the service? *

- o Very positive
- o Somewhat positive
- o Neutral
- o Somewhat negative
- o Very negative

How innovative is the application? *

- o Very Innovative
- o Somewhat innovative
- o Neutral
- o Not so innovative
- o Not at all innovative

When you think about the service, you know at least one person who could benefit from it. *

- o Strongly agree
- o Somewhat agree
- o Neither agree/disagree
- o Somewhat disagree
- o Strongly disagree

Installing the application was quick and simple. *

- o Strongly agree
- o Somewhat agree
- o Neither agree/disagree
- o Somewhat disagree
- o Strongly disagree

I am satisfied with the amount of information presented in the main layout. *

- o Strongly agree
- o Somewhat agree
- o Neither agree/disagree
- o Somewhat disagree
- o Strongly disagree

The application was smooth and responsive. *

- o Strongly agree
- o Somewhat agree
- o Neither agree/disagree
- o Somewhat disagree
- o Strongly disagree

The smart-home section was easy to understand. *

- o Strongly agree
- o Somewhat agree
- o Neither agree/disagree
- o Somewhat disagree
- o Strongly disagree

How would you rate the navigation and layout? *

- o Excellent
- o Above average
- o Average
- o Below average
- o Below average

I found the heart rate history very useful. *	
	o Strongly agree o Somewhat agree o Neither agree/disagree o Somewhat disagree o Strongly disagree
I found the signup process fast and easy to understand. *	
	o Strongly agree o Somewhat agree o Neither agree/disagree o Somewhat disagree o Strongly disagree
I found it easy to update my user profile. *	
	o Strongly agree o Somewhat agree o Neither agree/disagree o Somewhat disagree o Strongly disagree
The start-up guide was detailed enough. *	
	o Strongly agree o Somewhat agree o Neither agree/disagree o Somewhat disagree o Strongly disagree
The service functioned without technical problems *	
	o Strongly agree o Somewhat agree o Neither agree/disagree o Somewhat disagree o Strongly disagree
How would you rate the overall quality of the application? *	
	o Very high quality o High quality o Neither high nor low quality o Low quality o Very low quality
In your own words, what did you like most about the service?	
	Your answer
In your own words, what do you think could be improved with the service?	
	Your answer
Do you have any other comments, suggestions or feedback?	
	Your answer

Page 1 of 1
Never submit passwords through Google Forms.

SAVE

Project Proposal

September 2016

6.6 *Project Proposal*

6.6.1 Objectives

To design and develop a service that will act as a safety net for patients in the high-risk category for cardiac arrest and call for help if the person is incapacitated. This service targets those who desire to remain living in their own homes, as opposed to in assisted living facilities.

The service consists of a node and base-station was will communicate with each other over a Bluetooth Low Energy connection.

- The node is an android powered smartwatch with a built-in heartrate monitor.
- The Base station is an android tablet or phone with the ability to make phone calls.

The heartrate monitor will regularly check the patients' heartrate and compare against an upper and lower acceptable limit. If the reading falls within the limit then the system does nothing. If the system detects a reading outside of the acceptable range, it will send a Boolean flag to the base station. At this point, the job of the node is complete and the base station will handle the processes from here on.

The base station will consist of a singular user interface that presents the "current status" of the patient as either "healthy" or "Attention Needed". The interface also contains a panic button and an "Emergency contact" person. The user will be able to set the emergency services contact number on first load of the application and this is recorded permanently in the system. The user will also be prompted to add a contact persons' name and contact details and have this permanently recorded.

If the watch triggers an alarm, the base station will initiate a phone call to the emergency services and will play an audio message, calling for help and listing the patients' location. When the watch initially detects an issue, it will notify the wearer and allow them to cancel the alert within a short time period after the detection. This state is intended to avoid unintentional triggering of the system.

The base station will display an area with the contact details of the users' previously saved contact person and pressing this will initiate either a call or text directly to that person without the need for manual searching of the contacts list or initiating a call. This feature is intended for use in a scenario where the user requires non-urgent assistance from a family member or carer, and should not be used as an emergency contact.

6.6.2 Background

I originally came up with this idea in 3rd year of college but did not have the necessary skill set at the time to develop it as my semester project at the time. I was inspired with the idea after a conversation with a disabled friend who is in the high-risk category for cardiac arrest and requires the assistance of a carer on a daily basis, making him vulnerable when unaccompanied.

Though he requires round-the-clock attention, he is often left unaccompanied for short periods without the presence of a carer and due to the nature of his disability, would not be capable of calling for help by himself if an incident occurred. This service would provide a safety net to those in the same situation, acting as an intermediate for those who want to remain living in their own homes and to retain their independence despite their medical needs, when carers or family members are not present.

Many elderly people would prefer to remain in their homes as an alternative to living in care facility. The project aims to make this process safer by providing a service that a patient, family member or carer could install and set up so that the patient is monitored and able to call for assistance in the event of an emergency while they are alone or incapacitated. This enables the patient to receive medical assistance that they otherwise would not have been able to summon and could potentially be lifesaving.

The need for such a service has been highlighted recently by the discovery of the tragic death of two elderly brothers in Dublin who had been living together for decades, while remaining isolated from their community. One of the brothers had been deceased for several weeks while the second man, who was dependant on his brother for assistance, was unable to raise the alarm throughout, leading to unimaginable suffering and eventually the death of the remaining brother.

Through the use of existing technologies in heartrate monitoring and through a smartphone, this project aims to prevent such tragedies from happening, and allow the patient to receive medical assistance as soon as possible. The project aim is to provide this service and produce a single simple user interface that provides all the relevant information, without the need to traverse multiple different level of a UI, or search through contacts etc. so that the use of the service is intuitive to people who may have no technical knowledge.

6.6.3 Technical Approach

The project task is to create a service by which the users' heartrate is pulled regularly and compared against a set of acceptable parameters. This will include research into android development for smartphone and smart wearables, making automated calls from a users' mobile device, using Google API's and methods for reading sensor data from a wearable device, inter-device communication over BLE etc.

- Define the functional requirements, non-functional requirements and scope of the project, and develop UI wireframes.
- Research Android Developer console and identify processes and methods to gather required data and to communicate with wearable devices.
    - Research methods to interact with sensors and to gather data from them.
- The project will involve researching and creating an Android Wear application that sends a request to *SensorManager* for the heartrate.
- Checks the response against a scale and returns a Boolean result to the smartwatch app.
- If the result is outside the norm, the watch must evoke an action on the smartwatch. This will happen over Bluetooth LE.
- The smartphone application must be listening out for the trigger from the wearable device and if given a trigger, must initiate a call to emergency services by invoking an *Intent.ACTION_CALL* method.
- Once a connection is established, the mobile all must play an audio stream, detailing the need for help.
- The application will use *AudioManager.MODE_IN_CALL* to play the audio while in an active call, and *audioManager,setSpeakerPhoneOn(true)* to output the audio through the phones' speaker.

6.6.4 Special Resources Required

The project will require several items of specific hardware devices and will be developed using the Android Studio development environment, Python IDLE and the Firebase Console. A mobile device, smart-watch and a development board is required for the main functions of the application. Additionally, an 8-channel relay board is required for the development board.

Resource	Model	Requirements	Cost
Smart-watch	Moto 360 Gen 1	• Running Android Wear 1.0	€179.99
		• Optical heart rate sensor.	
		• Bluetooth Low Energy	

Android mobile device	OnePlus 3T	• Running Android version > 4.3 • Bluetooth • Fingerprint Sensor • Wi-Fi connectivity • Mobile data access	€479.99
Development board	Raspberry Pi 3 Model B	• Running Rasbian OS Pixel • Wi-Fi Connectivity	€69.00
8-Channel relay board	WAVGAT	• 5V power input • Mains voltage switchable relay	€14.99
Cloud database	Firebase – Free tier	• Handling user accounts • Saving user profile & event data	€0.00
		Total:	744.96

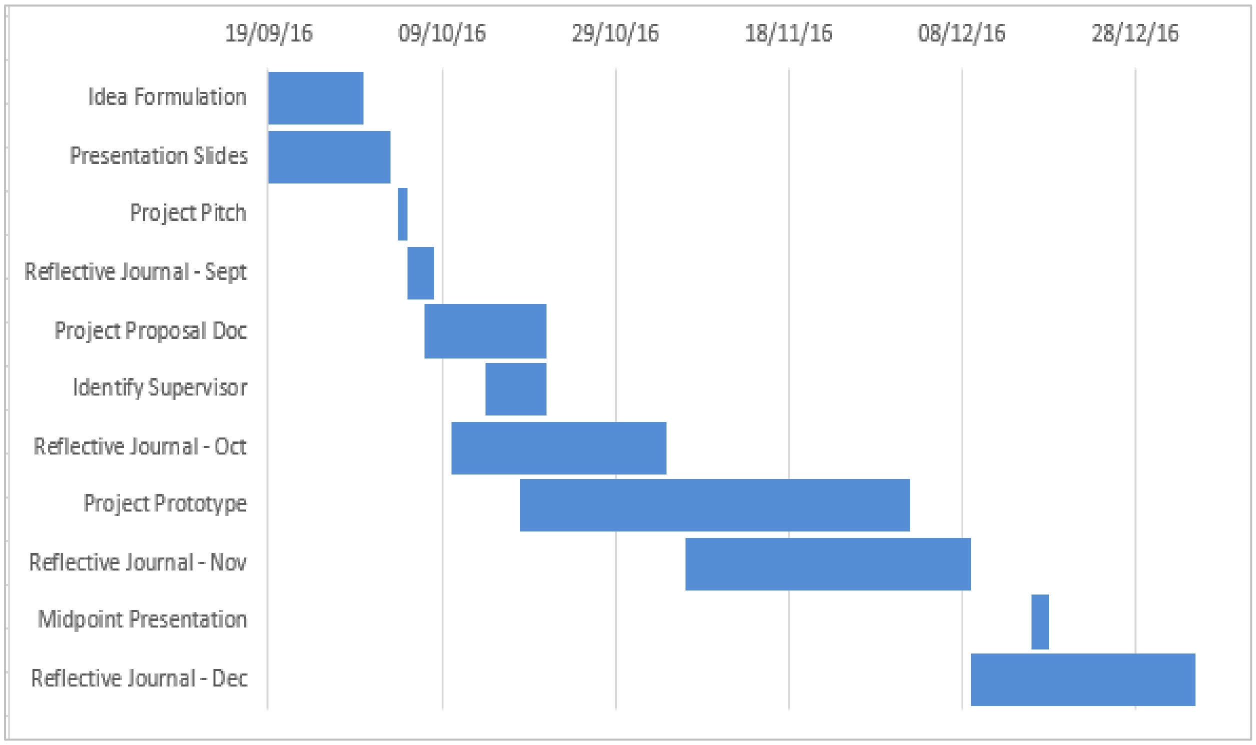
6.6.5 Technical Details

The project will consist primarily of three applications, a smartphone and a smartwatch app and a Python application to be run on the RaspberryPi. The Android applications will be developed using Java and XML for the logic and UI. The application will also contain a local database to store information on the smartphone. This will be created using Firebase. The Python code will be written in Cloud9 and will run in Raspbian OS on the RaspberryPi.

The project will involve the development of android background services and components that allow the device to lie idle until a change is detected, before evoking an action when prompted. The mobile application and the smart-watch will both require background services. These services shall be written in Java using *IntentService* classes.

6.6.6 Project Plan

The Gantt chart outlays a timeframe in which the project requirements shall be prioritised in the first phase of the module. This includes delivering a basic prototype of the application in December.



6.6.6.1 Idea Formulation:

The idea needed to be finalised by the upload deadline for the presentation slides on the 4th of October. I began considering possible ideas during the summer, when I had number of ideas shortlisted. I had 11 days from the beginning of the semester until the presentation day to formulate an idea and a set of core functions to incorporate in the project.

6.6.6.2 Presentation Slides:

The deliverable consisted of a maximum of two slides, briefly detailing the project idea and the functions it would perform. I had 2 weeks to identify exactly what the project would be, why it would be challenging, whom it should target and why I wanted to attempt the project.

6.6.6.3 Project Pitch:

The project pitch was due for the 4th of October and was a requirement to get the go ahead to continue with the project idea. The pitch was scheduled to run for approximately 5 minutes, in which I would detail the general purpose of the idea and to explain its purpose and defining characteristics. I received a positive response from the examiners, and received immediate confirmation that the project had been accepted.

6.6.6.4 Reflective Journal - September:

The September journal was due for submission on the 8th October. As this date was very close to the presentation pitch, there was a very small window of time to cover in the document, and this mainly consisted of detailing the formulation of the idea and the project slides and pitch.

6.6.6.5 Project Proposal:
The project proposal deliverable is due for the 20th of October and will include a more detailed breakdown of the project objectives, its functions, technologies used, project plan etc. I have allowed 14 days to complete this document.

6.6.6.6 Identify Supervisor:
The supervisors will be allocated to students in week 6 (24th – 28th Oct) however I have contacted a lecturer with the aim of securing my supervisor early in the project, and to best match my project with a supervisor who would have particular knowledge in the IoT and mobile development areas. I am aiming to have my supervisor by the end of week 5.

6.6.6.7 Reflective Journal – October:
I have allocated 25 days to complete the October journal, as this will cover many more details of the project development to date, including the proposal document, supervisor meeting and the early progress on development of the prototype. This journal is due for submission on the 4th of November.

6.6.6.8 Project Prototype:
The project prototyping phase has been allocated the maximum possible time allowable between confirmation of the project pitch and the delivery deadline of the prototype. This phase of the project is expected to take 45 days to deliver a working prototype, before further development into a fully functioning application. The prototype is due for submission on the 2nd of December 2016.

6.6.6.9 Reflective Journal – November:
The November journal is allocated 33 days to complete in time for submission on the 9th of December 2016. The journal is expected to detail progression of the project prototype and any possible setbacks, revisions or further research along the way. The journal will be written in short notes throughout the month, before being structured into a final document, ready for submission before the deliverable deadline.

6.6.6.10 Midpoint Presentation:
The midpoint presentation is due to take place over 2 days from the 16th December to the 17th of December 2016 and will examine the progress of the project this far.

6.6.6.11 Reflective Journal - December:
The December reflective journal will be the final journal before the end of the semester and will likely contain the results of the building of the project this far, including the midpoint presentations and the prototype deliverable, as well as detailing areas for possible improvement in the lead up to semester 2. The journal will be written in short notes throughout the month, before being structured into a final document, ready for submission before the deliverable deadline.

6.6.7 Evaluation & Testing

6.6.7.1 Unit testing

Unit testing will be implemented to run tests on individual functions or classes within the application, separately from the overall application. These unit tests will be used to highlight potential areas where bugs may occur later in the integration testing. Unit tests will be done locally within the development environment and through instrument tests on a physical device and android emulator.

6.6.7.2 Integration Testing

The system will undergo integration testing on completion of the unit tests. This phase will test a combination of various parts of the system together to highlight possible bugs before moving on to full system tests. This may include integration tests on service and components of the application, as well as user intractable functions. The integration testing will be conducted in the Android Studio IDE, on an android Virtual device and on a physical device.

6.6.7.3 System Testing

System testing will be implemented by pre-setting the saved emergency phone number to a test mobile phone number. The smart watch will be provided with sample sensor data that is outside acceptable parameters and should trigger a response from the smartwatch to the mobile application. This data may be alternated to trigger the smartwatch to switch states and force a recurring connection to the smartphone, while monitoring for failures.

The mobile application will be tested further with the user of a BLE simulator, an application for android devices that provides incoming pseudo data to an application for testing

6.6.7.4 Evaluation Testing

The system will be delivered to an end user for final evaluation testing and will be worn for several days to evaluate its ability to regularly take readings from the wearer, how to act on the readings and how it can deal with possible false alarm scenarios. This testing phase will provide feedback on final modifications that may be required before finalising the project.

Monthly Progress Journals

September 2016 – April 2017

6.7 Monthly Journals

6.7.1 September

Student name:	Rich Mangan
Programme:	BSHC4-IOT
Month:	September

My Achievements

As this is the second week of the semester, my only deliverable this far has been to pitch my software project idea to a panel of lecturers (*Paul Hayes, Lisa Murphy* and *Vikas Sahni*). I completed that process on Wednesday and was given the green light to continue with the project idea. Some suggestions were made on possibly removing some elements of the idea to simplify it, but I'm not sure I like the idea of pulling functionality out of it and I'd be a little fearful of a slimmed down version not being impressive enough to keep me on course for a 1st class honours. I failed to mention a couple of other points after one of the lecturers asked me a question and I forgot to come back to it. I should maybe insist on holding questions until the end for future reference.

My Reflection

I have been playing around with several different ideas and was a bit lost as to which to finally settle on but I ended up going with an idea I had during my second-year team project. We didn't take it up at the time as it seemed too complex for our skill level at the time but I feel with a lot more experience in mobile development now because of personal projects and my internship that I can do it. There's still a few areas where it's going to require further learning to proceed with my idea but I have a clearly defined plan in my head for what the service will do and in some ways more importantly, what it won't or isn't intended to do so I hope that helps in the documentation phase.

Intended Changes

Over the next month I will be working towards delivering a basic prototype of the idea to get some feedback on what areas might require more thought. I'll be refreshing on my software engineering class project to get an idea on where to start with the requirements spec/documentation etc., although I feel that If I can get a semi functional version of my idea up and running early, then it'll help greatly in knowing what to include in my UML, requirements spec etc.

I am currently researching development for wearable devices and communication with mobile apps. Development on a basic mobile app is going ok so far but there's a bit of a learning curve in doing the same for a smart watch.

Supervisor Meetings

Date of Meeting: *No meetings as of yet.

6.7.2 October

Student name:	Rich Mangan
Programme:	BSHC4-IOT
Month:	October

My Achievements

I have received my project supervisor as of last week and I have been assigned Dr. Cristina Muntean. She responded to me yesterday to organise our first meeting so we'll have to wait and see what will happen there, as I'm not quite sure yet as to what we will be doing. I have also made progress on the project specification document and should have that finished off in the coming days.

My Reflection

Other than the above, I haven't really had much development on the project itself, as we have been heavily burdened with the addition of the AI module and it requires way more time than most of our other modules. Its reading week right now and I've yet to take a break from this overloaded schedule and yet still need to prepare for 2 upcoming CA's at the start of the week on my return.

Intended Changes

-

Supervisor Meetings

Date of Meeting:	First meeting scheduled, but has not yet occurred
Items discussed:	*
Action Items:	*

6.7.3 November

Student name:	Rich Mangan
Programme:	BSHC4-IOT
Month:	November

My Achievements

Research was conducted in accessing sensor data from the android Wear watch and using this data to estimate the users' heartrate. The mobile application has been started, and includes the user interface and the main elements that the user will interact with, including the callout buttons and current status indicator on the main UI. The sensor can take readings from the wearer and the current task to process this data and output in some form to the user.

The technical report has been completed and must be uploaded before midnight on the 8th of December.

My Reflection

Time has been a limiting factor in the ability to make progress on this project over the past month because of additional load from other modules. Some progress has been made on the mobile application and early developments in the smart watch are underway. Currently, it is hoped that the collection of the sensor data from the watch can be completed in the coming week, ready for presentation of the prototype on the 20th of December.

Intended Changes

Some limitations have been identified with the watch in terms of its ability to regularly read the users heartrate accurately. The heartrate sensor requires that the watch is mounted firmly on the wrist to get an accurate reading. Other options for sensors are currently being researched with the aim of identifying a sensor that can more reliably monitor the users' heart activity. Ideally, another Android Wear smartwatch would be used, as this would make the transition from existing hardware more straightforward in terms of the application development.

Supervisor Meetings

Date of Meeting: 8th December 2016

Items discussed:

The next supervisor meeting is due to take place on the 8th of December. This will be the second meeting and it is hoped that the technical report can be examined by the supervisor, prior to uploading on the 9th of December.

Action Items:

-

6.7.4 January

Student name:	Rich Mangan
Programme:	BSHC4-IOT
Month:	January

My Achievements

Throughout the month of January, the main objective has been to fix irregularities in the readings from the heartrate sensor. This process was complicated due to poor documentation on the sensor service within android wear, as there are depreciated methods that don't have a clear path to an alternative. The launch of Android Wear 2 last week may also have another effect. The hope is that the documentation on the methods is updated or improved now that the new OS has been released. The sensor readings are now being gathered with an acceptable accuracy and further plans may include purchasing a second smartwatch with a more capable sensor on-board.

Work has begun on implementing the Firebase database within the application for storing the users' details, contact person etc. This is being developed with possible expansion in mind, if time and costs allows. This may be tweaked as a admin/user type system that could be rolled out to multiple elderly residents by a carer etc. This is not part of the original project idea, but is a possible expansion of the proposed idea, if time and cost allows.

Research is also being conducted into the feasibility of building a web dashboard that can display the data from the Firebase database in graph form for use by an admin.

My Reflection

Progress on the smart watch application is making progress now, after a lot of issues getting reliable readings from the sensor.

The database framework is made now and the next step is to integrate the user authentication into the app and to collect and store the data that the user needs, e.g. contact person names, phone numbers etc.

Intended Changes

Examine options for an alternative heartrate sensor/smartwatch. The current sensor is not reliable enough.

Supervisor Meetings

Date of Meeting: 3rd February 2017

Items discussed:

Marking scheme for the midpoint presentation

Complexity of projects

Action Items:

Consider doctor advice on cardiac activity

Get ethics form from Cristina

6.7.5 February

Student name:	Rich Mangan
Programme:	BSHC4-IOT
Month:	February

My Achievements

The user management and authentication has been completed and integrated into the Firebase database. Development is ongoing on the collection of data from the user on first sign up, and a user guide has been developed to guide the user through the functions and features of the application on first load.

The heart rate sensor on the watch is functional and has been set to activate only for a brief moment, to extend the battery life during development. This will later activate at a regular interval and stay active until a reliable reading is obtained.

Research has begun into developing a dashboard UI that could allow further extension of the service for use by an administrator who would monitor multiple people at once through the UI. A test was conducted that successfully showed that the live data from the Firebase database could be queried using HTTP GET requests that would return a JSON string of the required data. This data can then be fed into the desired charts for representing to the end user.

Contact has been made with a few doctors who are enthusiastic about the project and are keen to give their input in the areas where medical professional advice is required. I.e. the ranges at which a user should be concerned by their heart activity and the points at which a trigger is evoked within the system. A questionnaire is being assembled to document their input for the final project documentation.

My Reflection

The existing heart rate monitor may not be suitable for a real-world scenario and may need to be replaced at a later point with a more reliable option.

Intended Changes

Examine the feasibility of fingerprint authentication within the application.

Examine the options for displaying a user's heart activity in graph form within the application.

Supervisor Meetings

Date of Meeting: 10th March 2017

Items discussed:

- Adding user graphs in the mobile application
- Possibility of adding fingerprint authentication
- Bringing a prototype to the next meeting

Action Items:

- Complete survey and send to doctors.
- Start work on voice reading of address
- Complete data collection in app
- Build web dashboard for administrator

6.7.6 March

Student name:	Rich Mangan
Programme:	BSHC4-IOT
Month:	March

My Achievements

User management is finished and each user has a unique entry in the database for saving data about themselves, events etc.

I have received feedback from 2 doctors so far and hope to add another soon.

I have implemented the fingerprint authentication that was suggested by Cristina at the last meeting. Additionally, I have completed the graphs for user activity that was suggested also. The graphs contain only test data now but the aim over the coming week is to finalise graphs to accurately display user activity data.

I have been able to successfully retrieve data from the users account when making an automated call. I have a bug in the existing code that returns a null on first attempt but works as expected on subsequent try's. I suspect this is down to the network call to the cloud database not returning before the phone activity is launched. I can add a simple check here to make sure the number is retrieved before evoking the call.

My Reflection

In the most recent meeting, I was able to talk at length with Cristina about the project and provide a demo of the progress, including the features she had previously suggested adding. We talked about prioritising the mobile app and watch app instead of focusing on the web dashboard, which we could come back to at a later date if time allowed. We talked about adding in a night mode for the application so that users' heartrate lowest point could account for the user being asleep.

Intended Changes

Examine the feasibility of fingerprint authentication within the application.

Examine the options for displaying a users' heart activity in graph form within the application.

Supervisor Meetings

Date of Meeting: 31st March 2017

Items discussed:

- Project progress and application demo
- Adding a night mode to the application
- Focusing on application instead of web interface

Action Items:

- Add in a night mode
- Start work on voice reading of address