

National College of Ireland

BSc in Computing

Internet of Things

2017/2018

Evan Masterson

x14426302

x14426302@student.ncirl.ie or evanm95@hotmail.com

Geofence Monitoring

Technical Report



National
College *of*
Ireland

Table of Contents

Executive Summary	4
1 Introduction.....	5
1.1 Definitions, Acronyms, and Abbreviations	5
1.2 Background	5
1.3 Aims	5
1.4 Technologies	6
2 System	7
2.1 Requirements	7
2.1.1 User Requirements.....	7
2.1.2 Functional requirements	12
2.1.2.1 Use Case Diagram.....	13
2.1.2.2 Requirement 1: Register User.....	14
2.1.2.3 Requirement 2: Login User	16
2.1.2.4 Requirement 3: Reset Password.....	18
2.1.2.5 Requirement 4: Update Profile	20
2.1.2.6 Requirement 5: Create Geofence	22
2.1.2.7 Requirement 6: Update Geofence.....	24
2.1.2.8 Requirement 7: View Location	26
2.1.2.9 Requirement 8: View Data	28
2.1.2.10 Requirement 9: Receive Notifications	30
2.1.2.11 Requirement 10: Receive Texts	32
2.1.3 Non-functional requirements.....	34
2.1.3.1 Performance/Response time requirement.....	34
2.1.3.2 Availability requirement.....	34
2.1.3.3 Recover requirement	34
2.1.3.4 Robustness requirement	34
2.1.3.5 Security requirement.....	34
2.1.3.6 Reliability requirement.....	34
2.1.3.7 Maintainability requirement	34
2.1.3.8 Portability requirement	35
2.1.3.9 Extendibility requirement.....	35
2.1.3.10 Reusability requirement.....	35
2.1.3.11 Resource utilization requirement	35

2.2	Design and Architecture	36
2.3	Implementation	37
2.3.1	Android	37
2.3.2	Tizen.....	51
2.3.3	Cloud Functions.....	55
2.4	Graphical User Interface (GUI) Layout	58
2.4.1	User Login/Registration Screen	58
2.4.2	User Profile Screen.....	59
2.4.3	Main Screen	60
2.4.4	Edit Geofences Screen.....	61
2.4.5	View Extra Info Screen	62
2.4.6	User Profile Screen.....	63
2.5	Application Programming Interfaces (API)	64
2.6	Testing.....	64
2.6.1	Unit Testing	64
2.6.2	Usability testing.....	65
2.7	Evaluation.....	65
2.8	Limitations	70
3	Conclusions	71
4	Further development or research.....	72
5	References	73
6	Appendix.....	74
6.1	Project Proposal	74
6.2	Project Plan	77
6.3	Monthly Journals	80
6.3.1	September	80
6.3.2	October.....	81
6.3.3	November	82
6.3.4	January.....	84
6.3.5	February	85
6.3.6	March.....	85
6.3.7	April	86
6.3.8	May.....	87

Executive Summary

This application addresses the issue of being able to reliably monitor the location and wellbeing of people using geolocation and geofencing techniques through a smartwatch.

Geolocation refers to the current real-world geographical location of a device like a smartwatch.

Geofencing allows us to create a virtual boundary defined by latitude and longitude coordinates in relation to real world geographical data, in this case the geolocation of our smartwatch.

This application is targeted at elderly people, more specifically those who are affected by Dementia or Alzheimer's disease. It will allow carers to accurately monitor patients through an Android application that will be receiving data from the smartwatch in real-time through the cloud.

The smartwatch will constantly be transmitting its data to the cloud, that includes its geolocation, heart rate, steps, sleep pattern and more. Using this data stored in the cloud, I can historically graph all the smartwatch users body vitals and display their current location. While having access to their location the user will be able to define geofence zones for the smartwatch. If the current location of the smartwatch travels into the geofence zone defined, then it will trigger an alert to notify the user.

1 Introduction

1.1 Definitions, Acronyms, and Abbreviations

API – Application Programming Interface

AWS – Amazon Web Services

Cloud Functions – Event driven, serverless functions deployed in the cloud, triggered by event changes in our cloud services such as database updates to carry out a defined action.

Geofence – A virtual boundary defined by longitude and latitude coordinates in relation to real world geographical data.

Twilio – A cloud platform which provides communication via SMS.

1.2 Background

I came up with the idea for this application after a conversation I had with a friend. We had been talking about family and he started talking about his grandad who suffers from Alzheimer's disease. Often, he doesn't remember people's names and has bursts of rage. There have also been cases where he was wandered from home unknowingly to others. He tends not to trust individuals he meets on the streets so if he's lost or confused he doesn't ask for help.

After our conversation I came to think about an application to best try solving issues like this. I mentioned his application idea to my friend, immediately he said he would love to have something like that to ensure his grandads safety. He also threw some additional idea's my way in terms of functionality.

1.3 Aims

The aim of my application is to create a fully functional application that can be used by anyone who meets the system requirements, which is an Android phone and a Samsung Gear S smartwatch.

It will use Google API for real-time geolocation data on Android. Firebase API as the user authentication system, database storage and Cloud Functions. Twilio API for automated text alerts.

If I had the time I would aim to make this application cross platform for use with iOS and various other smartwatches.

1.4 Technologies

Android Studio

I will be using Android Studio for the mobile application development part of my project. This will be written in Java. The Android application is the forefront of this project as it provides a full user interface that the user can work, although, for the most part my Android application will just be for pulling in information and displaying, it won't contain that much functionality, except for user registration/login and google maps for setting geofences.

Tizen Studio

I will be using Tizen Studio for my smartwatch development; the Samsung Gear S comes installed with Tizen OS. It will be my first time developing in this Operating System. It is written in HTML5 and JavaScript. I will be developing an application to take readings from the smartwatch user and sending them to the cloud, where the Android application will then take down these readings and process them.

Firebase

I will be using Google's Firebase for all my user authentication, login, registration, email verification, and password reset. It will also be used for verifying the users phone number if they wish to opt into receiving automated text alerts. Database storage will be maintained here with user information along with the deployment and use of Cloud Functions.

Twilio

Twilio is an external service that I'll be using for sending automated text alerts to the user if they wish to avail of this service. This will require the user to verify their phone number in advance which will be done through Firebase. This service will only be triggered by a Cloud Function if necessary.

2 System

2.1 Requirements

2.1.1 User Requirements

I created a Google Form survey to gather some insight into other people's thoughts on my idea, I shared my survey between friends, family, and my college peer's. I closed my survey after running for 5 days as I wasn't receiving any new responses and in total I received 41 responses which were mostly very positive. You can view the results of the survey in the screenshots below. The outcome of this survey has assured me that I'm on the right path to developing my desired application.

There have been numerous research papers written about using geofencing on humans, supporting the idea.

An interesting journal I found was the Journal of Assistive Technologies, ISSN: 1754-9540. In which discusses the area of using *"GPS tracking and location technologies to support vulnerable people at risk of becoming lost or threatened"*.

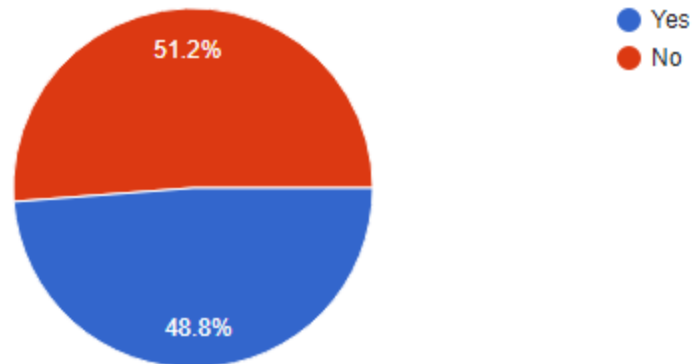
I've also supplied a quote below about making use of geofences for elderly care from a journal article listed in the references.

"It is difficult and tedious to track patients with memory loss. Such patients require extra care which leads to employing more people at the Geriatric centres. Setting up a digital boundary, allows the aged people to roam around freely. If they happen to cross the virtual fencing set by the geofencing, administration would receive the notification and will bring them back in the fencing zone. Geofencing allows the geriatric centres to make elderly people feel free and let them roam around without being under a constant spy."

Do you know anyone affected by Dementia/Alzheimer's disease

Yes

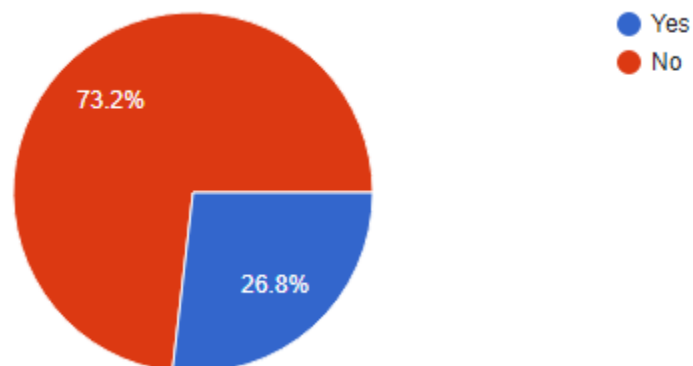
No



If Yes to Q1, have they ever wandered away from home unknowingly to others?

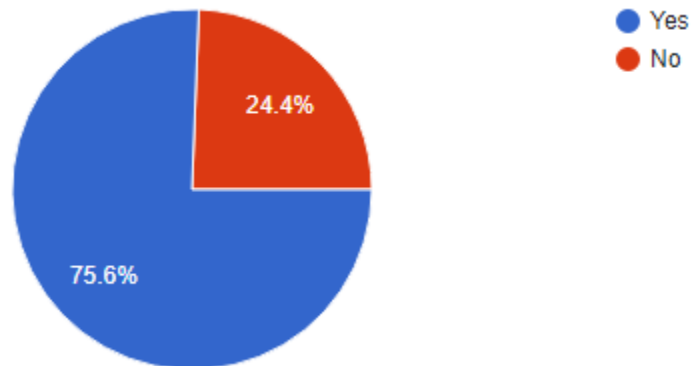
Yes

No



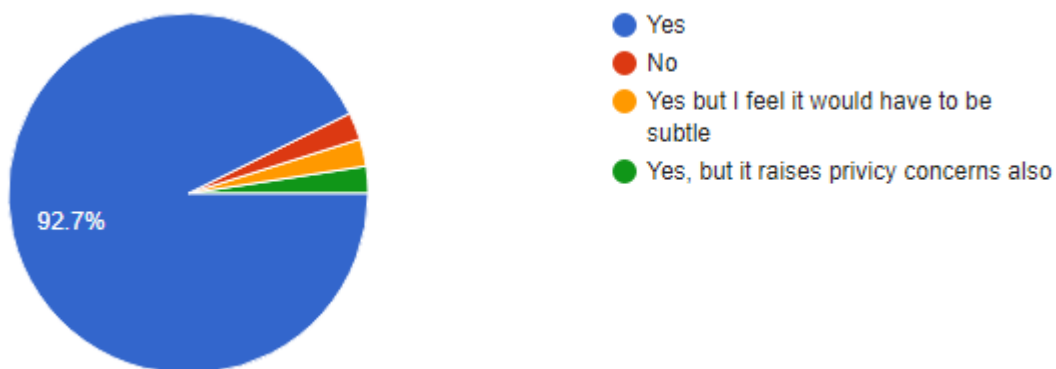
If Yes to Q1, but they have never wandered away from home, would you like to have the ability to monitor their location and be alerted in the case that they did wander in the future?

- Yes
- No



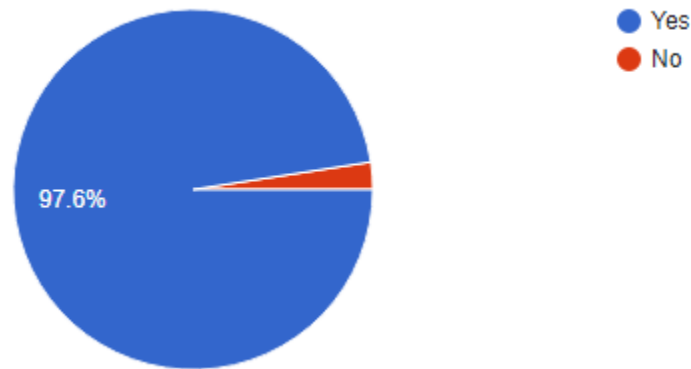
If you had the option to, would you like to monitor the safety of an elderly loved one, both location and body vitals whenever you want?

- Yes
- No
- Other...

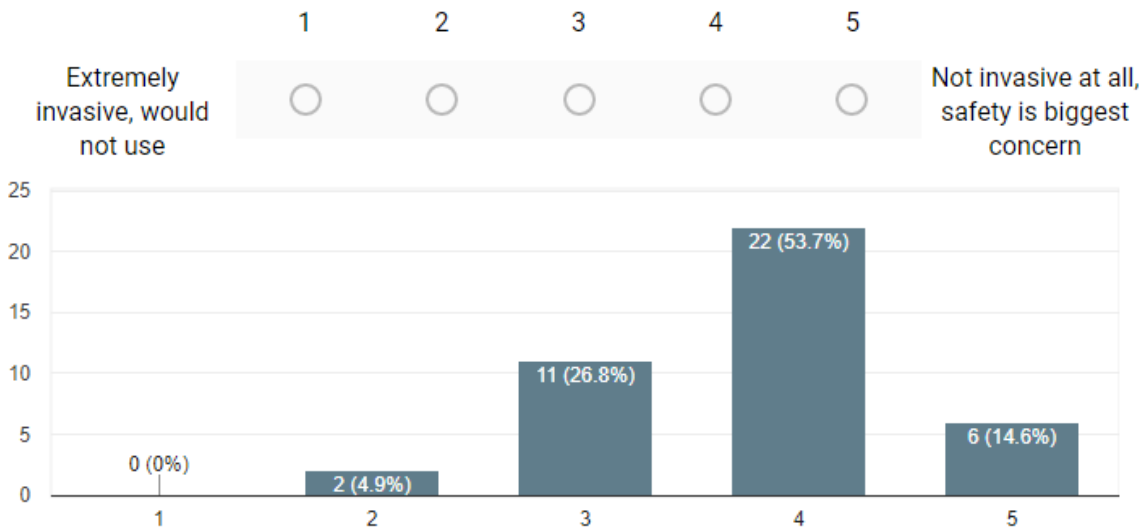


Do you think nursing homes for example, could benefit from an application like this?

- Yes
- No
- Other...



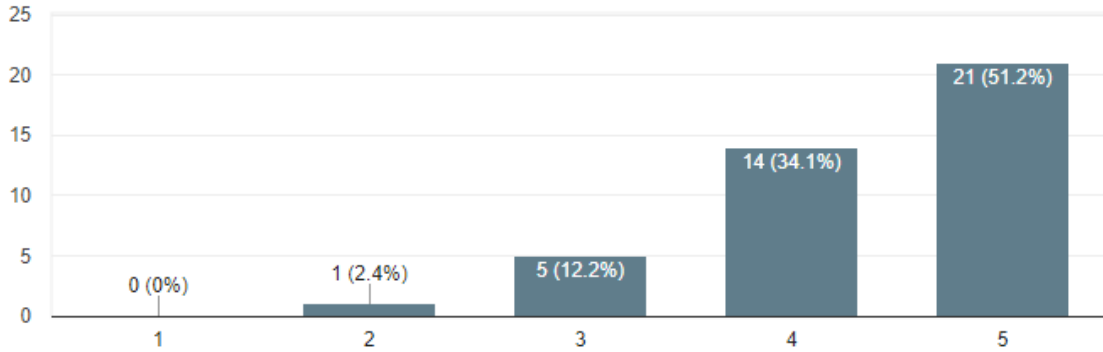
Do you consider an application like this invasive, even though it's for the safety and well being of others? *



How useful would you consider this application taking into consideration the panic button alert and viewing of historical body data which could be useful for catching early signs of illness? *

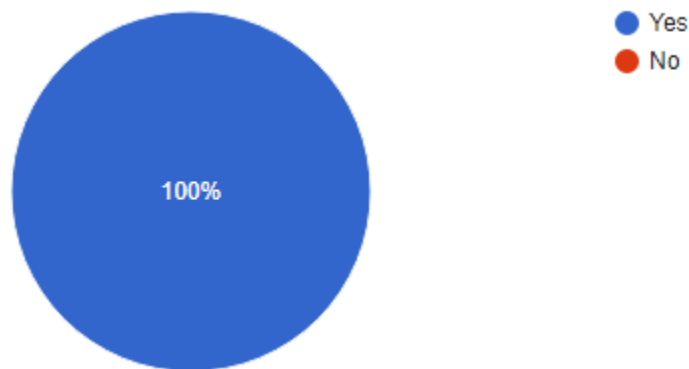
1 2 3 4 5

Not useful at all, would not use Extremely useful, could be hugely beneficial



Could you see this application potentially saving people's lives in severe cases?

- Yes
- No



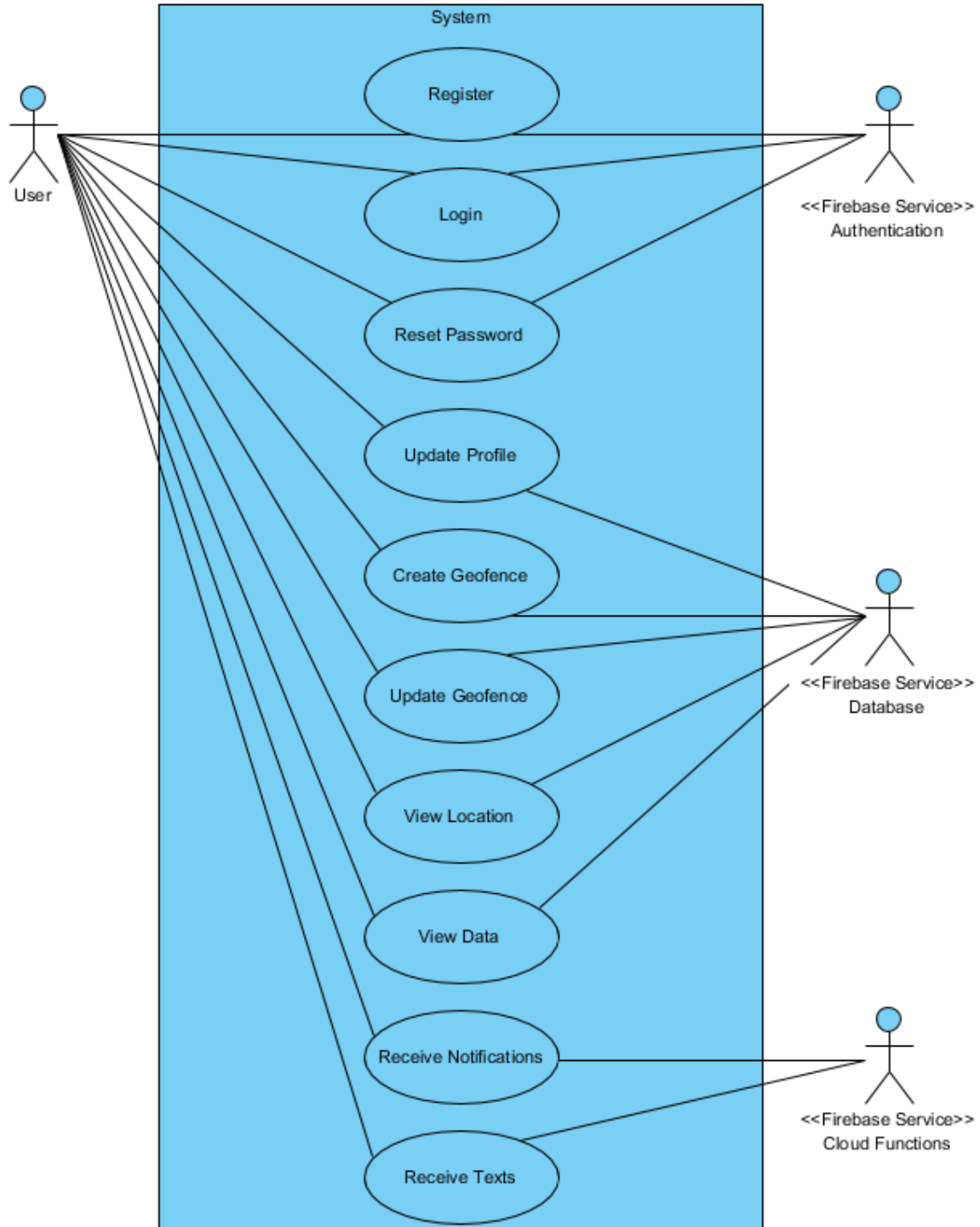
2.1.2 Functional requirements

The below requirements are listed in order from highest to lowest importance.

1. User Registration is required for the user to be able to access the application, it will allow for unique settings between each users account.
2. User Login comes after registration as they must have created an account before being able to login to access the applications features.
3. Password Reset needs to come next for the fact that if a user forgets their password they have no way of accessing the application.
4. Create Geofence is the most important interactable feature of the application, it's here where the user can start using geofences to monitor location activity of the smartwatch.
5. Receive Notifications is one of the most important features of the application as it sends push notifications to the user's phone when the smartwatch user has travelled outside of the defined geofence zone.
6. Receive Texts is similar to Receive Notifications but in this case a user has provided a valid phone number and they have accepted to receive automated text alerts when the smartwatch user has travelled outside of the define geofence zone.
7. View Location comes next, once a geofence has been setup the user will be able to view this on a map and the current location of the smartwatch user.
8. View Data is the ability to view all additional information being transmitted from the smartwatch like heart rate, sleep, steps etc.
9. Update Geofence comes down the line after the user has been able to setup initially, they can then return and edit any settings that they're not happy about or would like to tweak.
10. Update Profile I consider least important do to the fact that the user isn't required to complete their profile, but if they wish to toggle alerts on/off, change their email, update their password, or use automated text message alerts then they will be required to verify and supply a phone number.

2.1.2.1 Use Case Diagram

My Use Case diagram outlines the 8 functional requirements that my application needs to be capable of.



2.1.2.2 Requirement 1: Register User

2.1.2.2.1 Description & Priority

This requirement allows the user to create an account, so they can access the applications features. This will be essential for subsequent logins, password reset, setting up automated text alerts.

2.1.2.2.2 Use Case

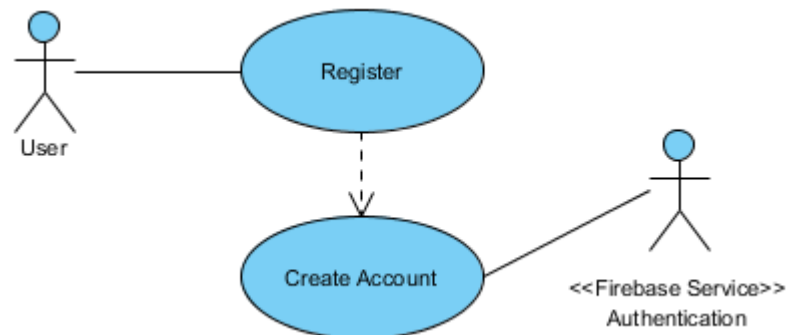
Scope

The scope of this use case is to allow the user to register a new account.

Description

This use case describes the process to register a new user.

Use Case Diagram



Flow Description

Precondition

The user has installed and launched the application.

Activation

This use case starts when a user opens the application and selects the signup option.

Main flow

1. The system presents the user the option to create an account.
2. The user enters valid email and password credentials and clicks the signup button. (See A1)
3. The system proceeds to create the new user through an external authentication service. (See A2)

4. The user's account is created, and they are automatically logged into the application. (See E1)

Alternate flow

A1: Invalid credentials

1. The system displays a message to the user that they have entered invalid credentials like an invalid email or their password didn't meet the specified requirements.
2. The user can re-enter the correct credentials.
3. The use case continues at position 3 of the main flow.

A2: Account already exists

1. The system displays a message to the user that an account already exists for the email address they provided.
2. The user can proceed to login, if they don't remember their password they can reset it.
3. The use case continues at position 3 of the main flow.

Exceptional flow

E1: Account creation error

1. The system receives an error creating the account, the authentication service may have had a slight glitch or could be offline or down.
2. The user accepts this message.
3. The use case continues at position 1 of the main flow.

Termination

The system presents the user with the main screen of the application.

Post condition

The system creates a new account for the user.

2.1.2.3 Requirement 2: Login User

2.1.2.3.1 Description & Priority

This requirement allows the user to login if they have an account already created. It will be important for accessing and displaying user specific settings.

2.1.2.3.2 Use Case

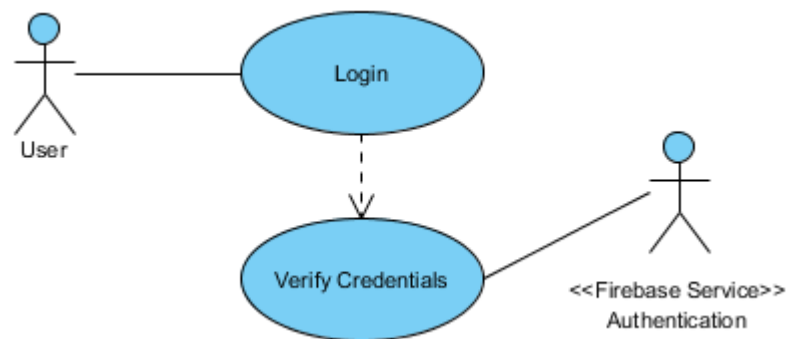
Scope

The scope of this use case is to allow a user to login.

Description

This use case describes the process of a user logging in.

Use Case Diagram



Flow Description

Precondition

The user has an existing account.

Activation

This use case starts when a user opens the application and selects the login option.

Main flow

1. The system presents the user the option to login.
2. The user enters valid credentials and clicks login. (See A1)
3. The system authenticates the credentials through the Firebase service. (See E1)
4. The users account is loaded, and they are brought to the main screen of the application.

Alternate flow

A1: Invalid credentials

1. The system displays a message to the user that they have entered invalid credentials like an invalid email or their password didn't meet the specified requirements.
2. The user can re-enter the correct credentials.
3. The use case continues at position 3 of the main flow.

Exceptional flow

E1: Account login error

1. The system receives an error authenticating the account, the email and password may not match, or the authentication service may have had a slight glitch or could be offline or down.
2. The user accepts this message.
3. The use case continues at position 1 of the main flow.

Termination

The system presents the user with the main screen of the application.

Post condition

The system logs the user in.

2.1.2.4 Requirement 3: Reset Password

2.1.2.4.1 Description & Priority

This requirement allows the user to reset their password in the case that they have forgotten it and cannot login. It reassures the user that will always be able to gain access to their account with a valid email.

2.1.2.4.2 Use Case

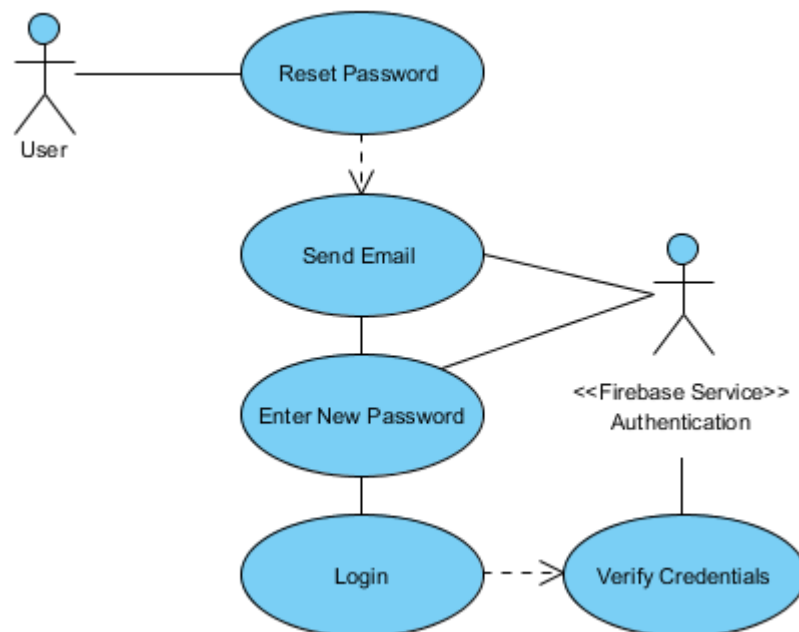
Scope

The scope of this use case is to allow the user to reset their password.

Description

This use case describes the situation where a user requests a password reset.

Use Case Diagram



Flow Description

Precondition

The user has an existing account.

Activation

This use case starts when a user opens the application and selects the reset password option.

Main flow

1. The system presents the user the option to reset password.
2. The user enters their email and clicks reset password. (See A1)
3. The system makes a call to an external authentication service to send a password reset email. (See E1)
4. The user can reset their password via the link provided in the email.
5. The user may now proceed to login.

Alternate flow

A1: Invalid credentials

1. The system displays a message to the user that they provided an invalid email address.
2. The user can re-enter the correct credentials.
3. The use case continues at position 3 of the main flow.

Exceptional flow

E1: Account does not exist

1. The system displays a message to the user that an account does not exist for the email provided.
2. The user can re-enter the correct credentials.
3. The use case continues at position 1 of the main flow.

Termination

The system presents a confirmation message that a password reset email was sent and returns to the login screen.

Post condition

The user can reset their password.

2.1.2.5 Requirement 4: Update Profile

2.1.2.5.1 Description & Priority

This requirement allows the user to complete or edit their existing profile. This requirement is not necessary but if the user would like to receive automated text alerts they will be required to supply a valid phone number.

2.1.2.5.2 Use Case

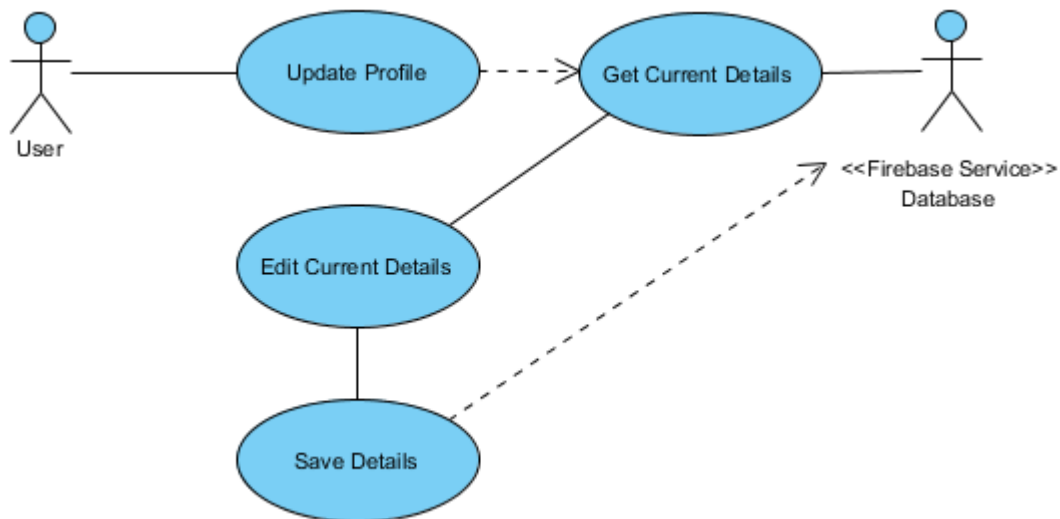
Scope

The scope of this use case is to allow the user to update their personal profile details.

Description

This use case describes the process a user undergoes when updating information in their profile.

Use Case Diagram



Flow Description

Precondition

The user is logged in.

Activation

This use case starts when a user navigates to the profile page through the options menu.

Main flow

1. The system presents the user the option to edit their current profile details.
2. The user may update their name, email, phone and save changes. (See A1)
3. The user may change their current password and save changes. (See A2)
4. The system pushes these changes to an external database and authentication service. (See E1)

Alternate flow

A1: User verification

1. The system sends a verification email or text if the user changes their email or phone.
2. The user verifies these changes.
3. The use case continues at position 3 of the main flow.

A2: Invalid credentials

1. The system alerts the user that their current password did not match, or their new password did not meet the specified requirements.
2. The user can re-enter the correct credentials.
3. The use case continues at position 4 of the main flow.

Exceptional flow

E1: External service error

1. The system receives an error from the external service and alerts the user it encountered an error updating their details.
2. The user accepts this message.
3. The use case continues at position 1 of the main flow.

Termination

The system presents a message confirming the user details have been updated.

Post condition

The user has updated their profile details.

2.1.2.6 Requirement 5: Create Geofence

2.1.2.6.1 Description & Priority

This requirement allows the user to setup geofence zones for the smartwatch. This is extremely important as it is the main component of the application.

2.1.2.6.2 Use Case

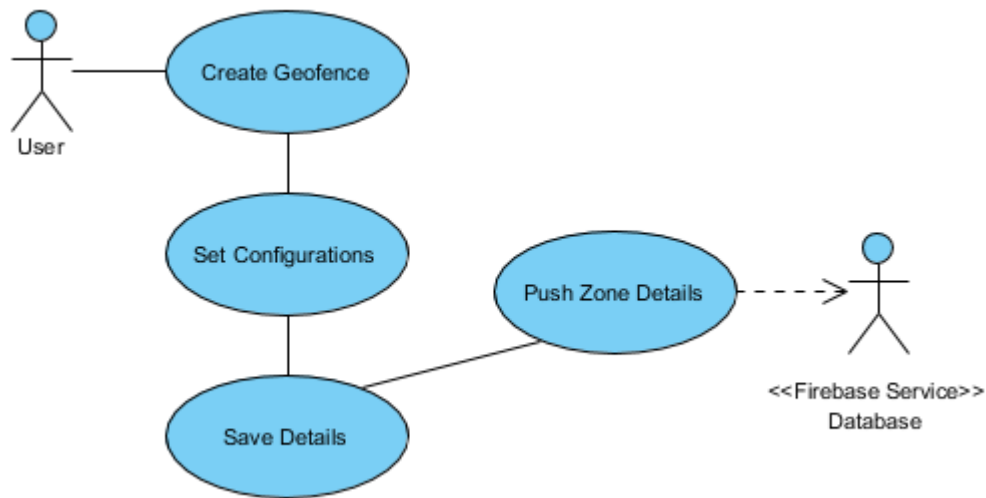
Scope

The scope of this use case is to allow the user setup a geofence for the smartwatch.

Description

This use case describes the process of a user creating a new geofence zone.

Use Case Diagram



Flow Description

Precondition

The user is logged in and has a smartwatch paired.

Activation

This use case starts when a user navigates to the setup geofence locations page from the main screen.

Main flow

1. The system presents the user with a blank map and the option to add a geofence zone.
2. The user can select a point on the map and select multiple configurations, including zone colour, radius and when they want the geofence active.
3. The user can save these configurations when finished.
4. The system accepts these configurations and saves them to an external database. (See E1)

Alternate flow

None

Exceptional flow

E1: External service error

1. The system receives an error from the external service and alerts the user it encountered an error updating their details.
2. The user accepts this message.
3. The use case continues at position 1 of the main flow.

Termination

The system returns to the main screen where the user can view the new map containing the new geofence zone.

Post condition

The user has created a geofence zone for the smartwatch.

2.1.2.7 Requirement 6: Update Geofence

2.1.2.7.1 Description & Priority

This requirement allows the user to update any configurations of an existing geofence. They can also remove it if they wish.

2.1.2.7.2 Use Case

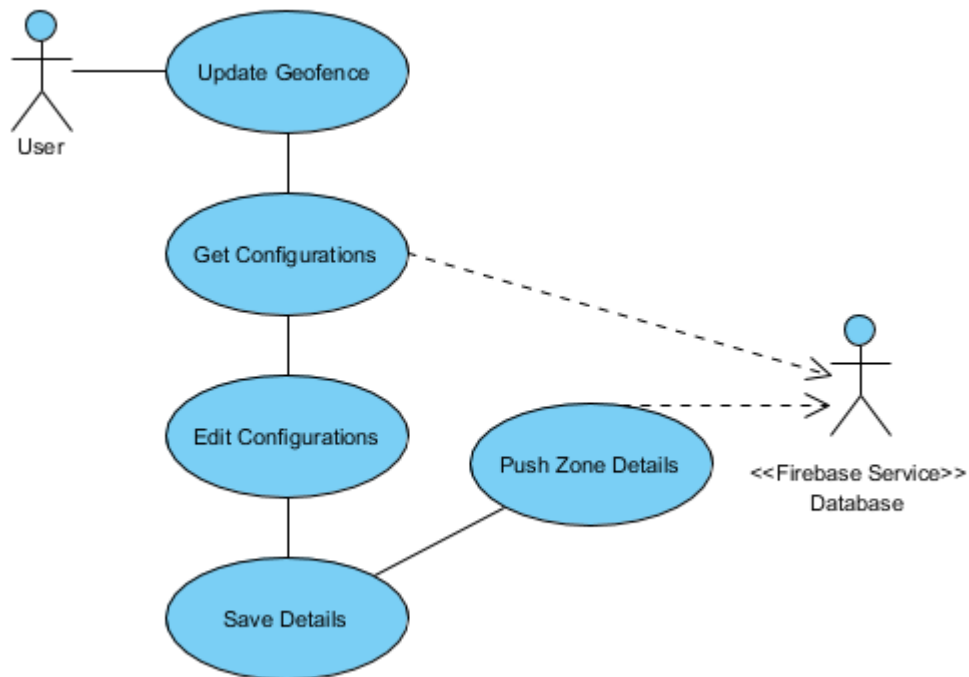
Scope

The scope of this use case is to allow a user to update an existing geofence zone.

Description

This use case describes the process of a user editing configurations of an existing geofence zone.

Use Case Diagram



Flow Description

Precondition

The user already has a geofence created.

Activation

This use case starts when a user navigates to the setup geofence locations page from the main screen.

Main flow

1. The system presents the user a map of existing geofence zones and the option to click on them and update any configurations.
2. The user can select an existing zone and update the zone colour, radius, and when they want it active.
3. The user can save these configurations when finished.
4. The user can also select an existing zone and remove it. (See A1)
5. The system accepts these changes and updates it in an external database. (See E1)

Alternate flow

A1: Confirmation of removal

1. The system presents the user with an alert if they confirm they wish to remove the existing zone.
2. The user accepts this message.
3. The use case continues at position 5 of the main flow.

Exceptional flow

E1: External service error

1. The system receives an error from the external service and alerts the user it encountered an error updating the details of the geofence.
2. The user accepts this message.
3. The use case continues at position 1 of the main flow.

Termination

The system returns to the main screen where the user can view the new map containing the updated geofence zone.

Post condition

The parameters of the existing geofence has been updated.

2.1.2.8 Requirement 7: View Location

2.1.2.8.1 Description & Priority

This requirement allows the user to view the current smartwatch location and any geofence zones associated with it. This is important as it makes up the main screen of the application, it's the first screen the user see's after logging in.

2.1.2.8.2 Use Case

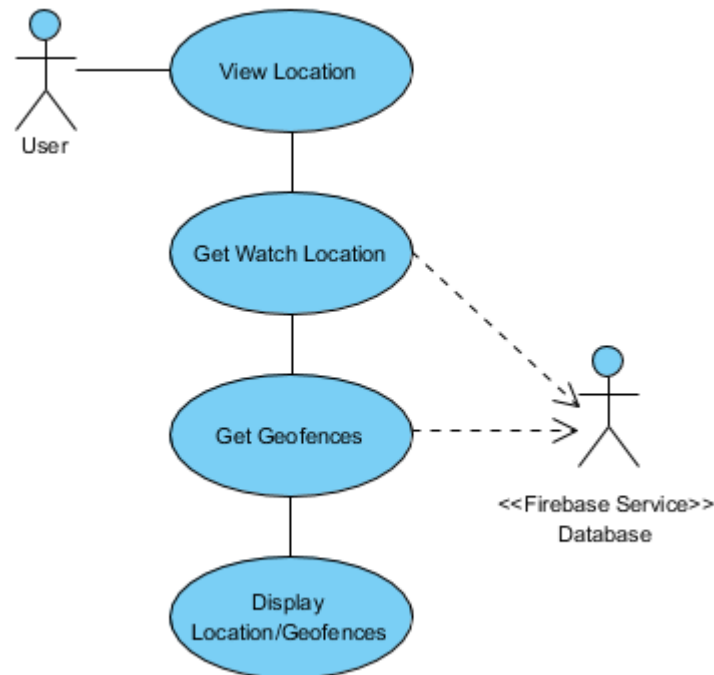
Scope

The scope of this use case is to allow the user view location information and geofence zones associated with the smartwatch.

Description

This use case describes the process of a user viewing current smartwatch location information.

Use Case Diagram



Flow Description

Precondition

The user is logged in and has a smartwatch paired.

Activation

This use case starts when a user logs in and is presented with the main screen of the application.

Main flow

1. The system presents the user with a map of the current smartwatch location and all geofence zones associated with it.
2. The user can navigate to other parts of the application such as setting up geofences or to view additional smartwatch data from this screen.
3. The system presents the user with a new screen if they have chosen to select one.

Alternate flow

None

Exceptional flow

None

Termination

The system presents the user with the main screen of the application.

Post condition

The user has viewed the current smartwatch location and any geofences defined.

2.1.2.9 Requirement 8: View Data

2.1.2.9.1 Description & Priority

This requirement allows the user to view any additional information about the smartwatch user, such as heart rate, steps, sleep patterns.

2.1.2.9.2 Use Case

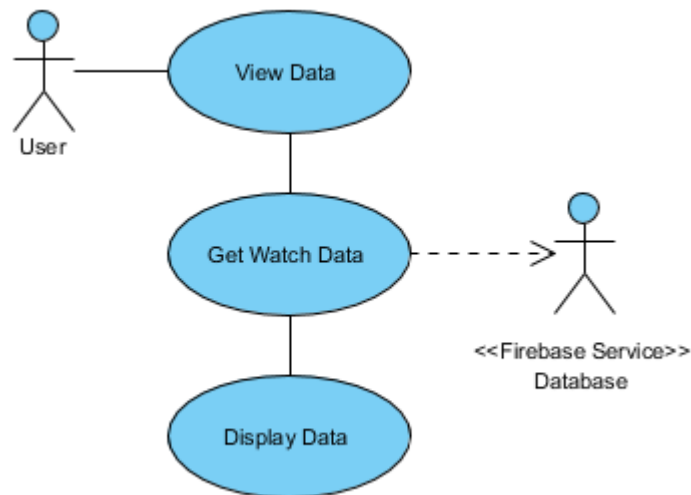
Scope

The scope of this use case is to view additional important data outside of location.

Description

This use case describes the process of viewing additional information on the smartwatch user.

Use Case Diagram



Flow Description

Precondition

The user is logged in and has a smartwatch paired.

Activation

This use case starts when a user clicks the view extra information button from the main screen.

Main flow

1. The system presents additional information about the smartwatch such as heart rate, steps, sleep patterns in forms of graphs.
2. The user may select historical data by selecting a date. (See E1)

Alternate flow

None

Exceptional flow

E1: No data

1. The system alerts the user that there is no information recorded for the date selected.
2. The user accepts this message.
3. The use case continues at position 1 of the main flow.

Termination

The system presents the user with the main screen of the application.

Post condition

The user is displayed with graphs containing relevant data.

2.1.2.10 Requirement 9: Receive Notifications

2.1.2.10.1 Description & Priority

This requirement allows the user to receive notification alerts about the whereabouts of the smartwatch if it has travelled beyond a geofence zone. Notification alerts is one of the most important features of the application.

2.1.2.10.2 Use Case

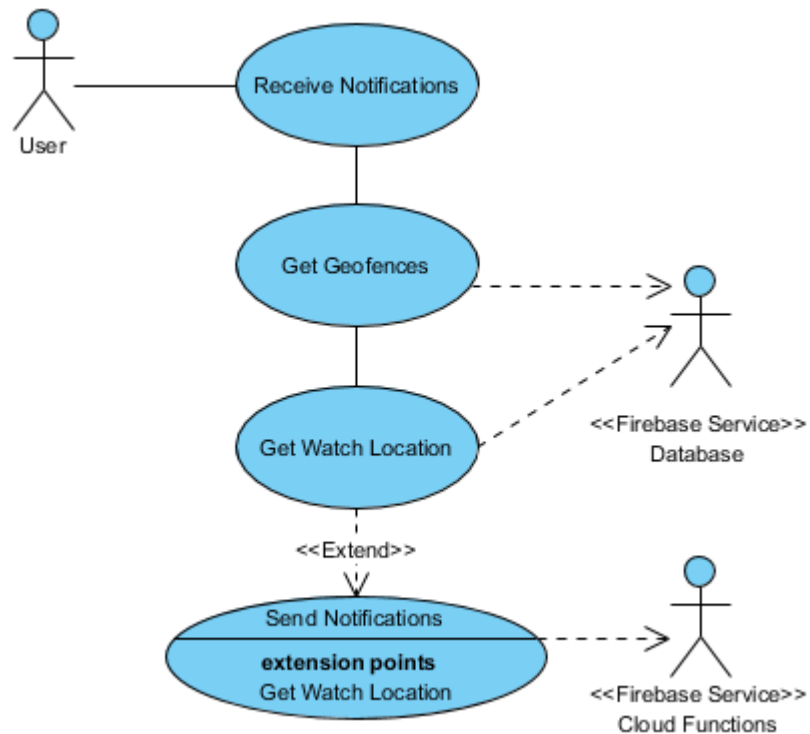
Scope

The scope of this use case is to receive notification alerts.

Description

This use case describes the process of receiving push notification alerts.

Use Case Diagram



Flow Description

Precondition

The user has accepted to receive notification alerts.

Activation

This use case starts when the smartwatch location triggers a notification alert.

Main flow

1. The system receives an alert that the smartwatch has travelled beyond a geofence zone.
2. The system sends a push notification to alert the user. (See E1)
3. The user can click the notification and it opens the main screen of the application showing the current location of the watch.

Alternate flow

None

Exceptional flow

E1: Phone is switched off

1. The users phone is switched off, so they cannot receive a notification.
2. The notification will show up when the phone is switched back on.
3. The user case continues at position 1 of the main flow.

Termination

The system presents the user with the main screen of the application.

Post condition

The user receives a push notification alert.

2.1.2.11 Requirement 10: Receive Texts

2.1.2.11.1 Description & Priority

This requirement allows the user to receive automated text alerts about the whereabouts of the smartwatch if it has travelled into a geofence zone. Notification alerts is one of the most important features of the application.

2.1.2.11.2 Use Case

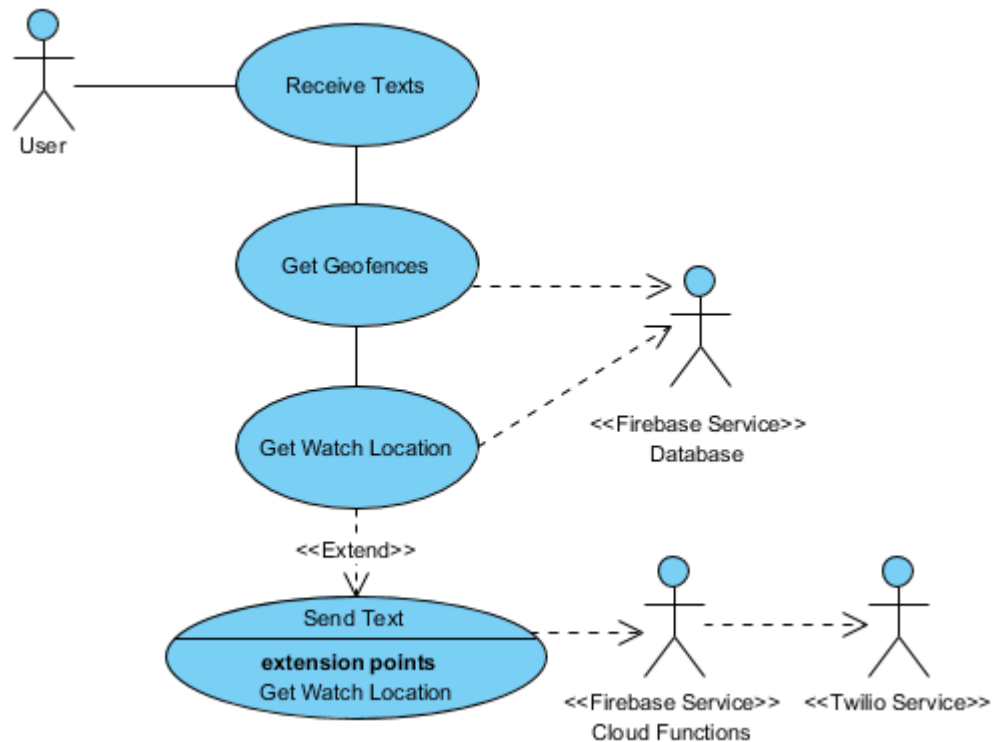
Scope

The scope of this use case is to receive automated text alerts.

Description

This use case describes the process of receiving automated text alerts.

Use Case Diagram



Flow Description

Precondition

The user has supplied a valid phone number and has accepted to receive automated text alerts.

Activation

This use case starts when the smartwatch location triggers a notification alert.

Main flow

1. The system receives an alert that the smartwatch has travelled beyond a geofence zone.
2. The system checks if the user has enabled text alerts.
3. The system sends an automated text alert through an external service. (See E1)
4. The user receives a text notifying them of the situation. (See E2)

Alternate flow

None

Exceptional flow

E1: External service error

1. The system receives an error from the external service and alerts the user it encountered an error updating their details.
2. The user accepts this message.
3. The use case continues at position 1 of the main flow.

E2: Phone is switched off

1. The users phone is switched off, so they cannot receive a notification.
2. The notification will show up when the phone is switched back on.
3. The user case continues at position 1 of the main flow.

Termination

The system presents the user with the main screen of the application.

Post condition

The user receives an automated text alert.

2.1.3 Non-functional requirements

2.1.3.1 Performance/Response time requirement

For the smartwatch part of this application, it will be constantly running in the background and sending information on a cycle so its performance and response time will be constant. In the case of the android application, it will update in real-time the location of the watch as its moving so its performance and response time is based off how often the smartwatch is transmitting data.

All GUI's should retrieve and display information within 2 seconds or quicker.

2.1.3.2 Availability requirement

As this application will require an active internet connection and will rely on Firebase being available to host and transmit data between the smartwatch and smartphone. If the user does not have an internet connection or because I cannot guarantee that the Firebase service will always be available, then I cannot guarantee my application will always be available. I also rely on Google Maps and Twilio to be available.

2.1.3.3 Recover requirement

In the case where a user forgets the password to their account they can request to have it reset where they will be sent an automated email. For all extra watch information such as the heart rate data, it will be all be backed up with duplicate copies in the case the original becomes corrupt.

2.1.3.4 Robustness requirement

My application will handle all exceptions that occur so that the system does not crash on the user. If an exception occurs, the system will notify the user through a subtle alert and will resolve the application to a point where the user can continue to interact.

2.1.3.5 Security requirement

The security for my application is that a user needs a working internet connection and a correct set of credentials to access their personal profile. The user will have to verify their email when registering and only the user will be able to reset their password via their own email. The password of each user will also be encrypted by the Firebase Authentication service. This will prevent others from accessing user accounts that are not their own. Also, if a user opts into text alerts they will be required to enter a verification code sent to their phone.

2.1.3.6 Reliability requirement

I plan to have the application being able to run 24/7 if the smartwatch is turned on and transmitting data. This will provide reliable up to date data. There will be times where it is out of my control and an external service is not running or is offline such as Google Maps, Firebase, or Twilio.

2.1.3.7 Maintainability requirement

I will maintain my application through constant testing and listen to user feedback about any bugs or glitches encountered so that I can rectify any issues.

2.1.3.8 Portability requirement

As this application is built for Android, it will have to support as many versions as possible to allow a wide range of devices to be compatible. It will support devices from Android 4.0.3+ which is any device running Ice Cream Sandwich or higher.

2.1.3.9 Extendibility requirement

The application I'm creating is built with a Samsung Gear S and a Samsung Galaxy S5, in the future it could be extended to support all Android and iOS devices, and to work across a wide range of smartwatches for wide spread use.

2.1.3.10 Reusability requirement

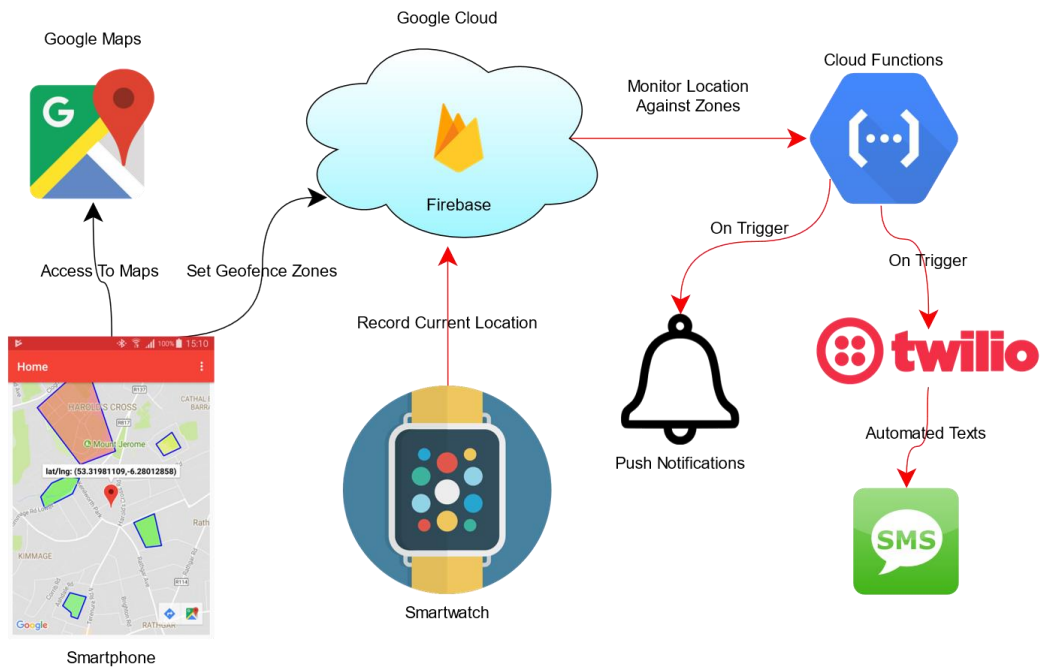
Some of the code I write will be reused in various parts of my application. For example, my options menu in Android will be written once but extended into every page. I'll also make use of Google Maps in my main screen and my edit geofences screen. I'll have a UserInfo class that can be called from anywhere in my application to get data on the user from the database.

2.1.3.11 Resource utilization requirement

My application will not be very resource heavy on the user's personal device, as I'm making use of a lot of external services to process information, this reduces the load of the application in run time.

2.2 Design and Architecture

I've chosen this architecture for my system because I've got more than just a simple standalone android application and to keep the heavy workload off my Android and smartwatch applications. This will reduce latency and load time will be more efficient to allow the applications to perform to their highest standard. I've got various separate components which all interconnect and rely on each other. My smartwatch application isn't interactable, it's purely running in the background and transmitting its data to Firebase on a looped cycle. My android application is the main application and will be interacting with multiple API's, from here I will be accessing smartwatch data from Firebase and interacting with Google Maps. The cloud functions take the heavy computational load off my application for comparing current location against each of the defined zones and responsible for triggering alerts. I've listed all external systems as services in my diagram as I'm purely making external calls while either providing some parameters or just retrieving information.



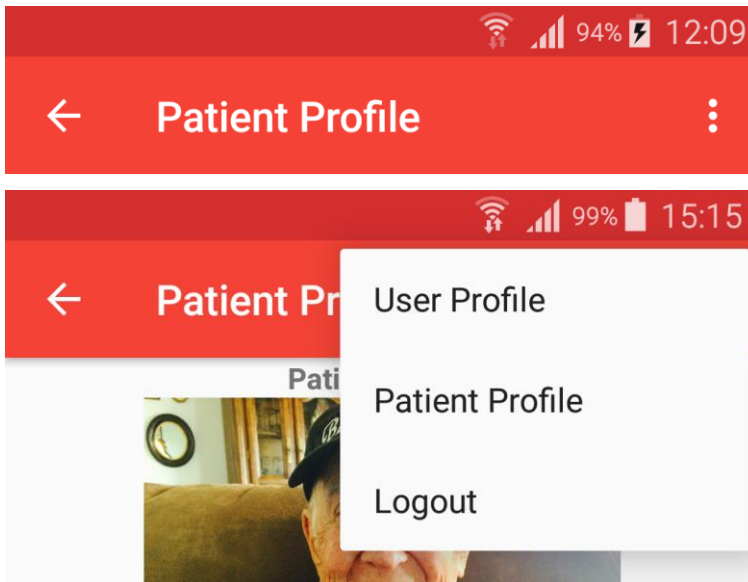
2.3 Implementation

2.3.1 Android

The Android application is the heart of this project, it provides all the necessary information the user needs and additional functionality to enable this application to meet the requirements set out.

Base Setup

It's worth noting that all of the activity's implemented in my application extend a base activity that I have defined. This allows each activity to have a uniform view in terms of the action bar at the top of the application. This includes backwards navigation and an options menu to which the user can navigate to user profile, patient profile or logout.



I achieved this through inflating an options menu and defining what each of the selected items can do, navigating to other activities through intents.

```

@Override
public boolean onOptionsItemSelected(MenuItem item){
    switch(item.getItemId()){
        case R.id.home:
            NavUtils.navigateUpFromSameTask( sourceActivity: this);
            return true;
        case R.id.menu_patient_profile:
            Intent patientProfileIntent = new Intent(getApplicationContext(), PatientProfileActivity.class);
            startActivity(patientProfileIntent);
            return true;
        case R.id.menu_user_profile:
            Intent profileIntent = new Intent(getApplicationContext(), UserProfileActivity.class);
            startActivity(profileIntent);
            return true;
        case R.id.menu_logout:
            auth.signOut();
            Toast.makeText(getApplicationContext(), text: "Logged out", Toast.LENGTH_SHORT).show();
            Intent logoutIntent = new Intent(getApplicationContext(), LoginRegistrationActivity.class);
            startActivity(logoutIntent);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

```

Login Registration Activity

The implementation of this part of my project is crucial and the focal point to allowing subsequent functionality to be implemented. I've developed it in such a way that it acts and provides all functionality you would need in a commercially viable application. Its functions include user registration, login, password reset and resending verification emails. These functions include important email and password validation.

Before any user can begin to benefit from this application they must sign up and verify their email to proceed further. With this account setup they can then login to the smartwatch to begin transmitting its data. All information is stored under the user's unique identifier (UID) created by Firebase upon account creation.

From here on out each activity will make use of the FirebaseAuth API to allow us access to the current user object.

On creating a user account, we make a call to *validateEmail* and *validatePassword*. These functions are responsible for ensuring the user is entering the correct criteria for emails and passwords. We then make a call to *sendVerificationEmail* which is responsible for sending a verification email which the user needs to activate their account to access the application. We also set some default information for the user, such as enabling notifications where they can toggle on/off later if they wish.

```

private void createAccount(String email, String password) {
    if (!validateEmail(email) || !validatePassword(password)) {
        return;
    }
    showProgressDialog();

    auth.createUserWithEmailAndPassword(email, password).addOnCompleteListener( activity: this, new OnCompleteListener<AuthResult>() {
        @Override
        public void onComplete(@NonNull Task<AuthResult> task) {
            if (task.isSuccessful()) {
                sendVerificationEmail();
                user = auth.getCurrentUser();
                FirebaseDatabase database = FirebaseDatabase.getInstance();
                DatabaseReference dbRef = database.getReference().child(user.getId());
                dbRef.child("notifications").setValue(true);
            } else {
                Toast.makeText(getApplicationContext(), text: "Authentication failed.", Toast.LENGTH_SHORT).show();
                confirmUser(null);
            }
            hideProgressDialog();
        }
    });
}

```

By signing in we are assuming the user has created an account, if they haven't a simple authentication failed message is shown, if they have then we make a call to *checkIfEmailVerified* and this function checks if the user has verified their account before proceeding.

```

public void signIn(String email, String password) {
    showProgressDialog();

    auth.signInWithEmailAndPassword(email, password).addOnCompleteListener( activity: this, new OnCompleteListener<AuthResult>() {
        @Override
        public void onComplete(@NonNull Task<AuthResult> task) {
            if (task.isSuccessful()) {
                checkIfEmailVerified();
            } else {
                Toast.makeText(getApplicationContext(), text: "Authentication failed.", Toast.LENGTH_SHORT).show();
                confirmUser(null);
            }
            hideProgressDialog();
        }
    });
}

```

As mentioned above I've implemented a function to ensure the user is entering the specific criteria set for emails and passwords, both functions are identical except for the pattern being checked. Declared at the top of my class I have regex patterns defined for *EMAIL_PATTERN* and *PASSWORD_PATTERN*. Using a matcher, we can check if the entered field matches the criteria.

```

public boolean validateEmail(String email) {
    boolean valid = false;
    Pattern pattern = Pattern.compile(EMAIL_PATTERN);
    Matcher matcher = pattern.matcher(email);

    if (matcher.matches()) {
        valid = true;
    } else {
        emailET.setError("Invalid Email");
        Toast.makeText(getApplicationContext(), text: "Please enter valid email", Toast.LENGTH_LONG).show();
    }
    return valid;
}

```

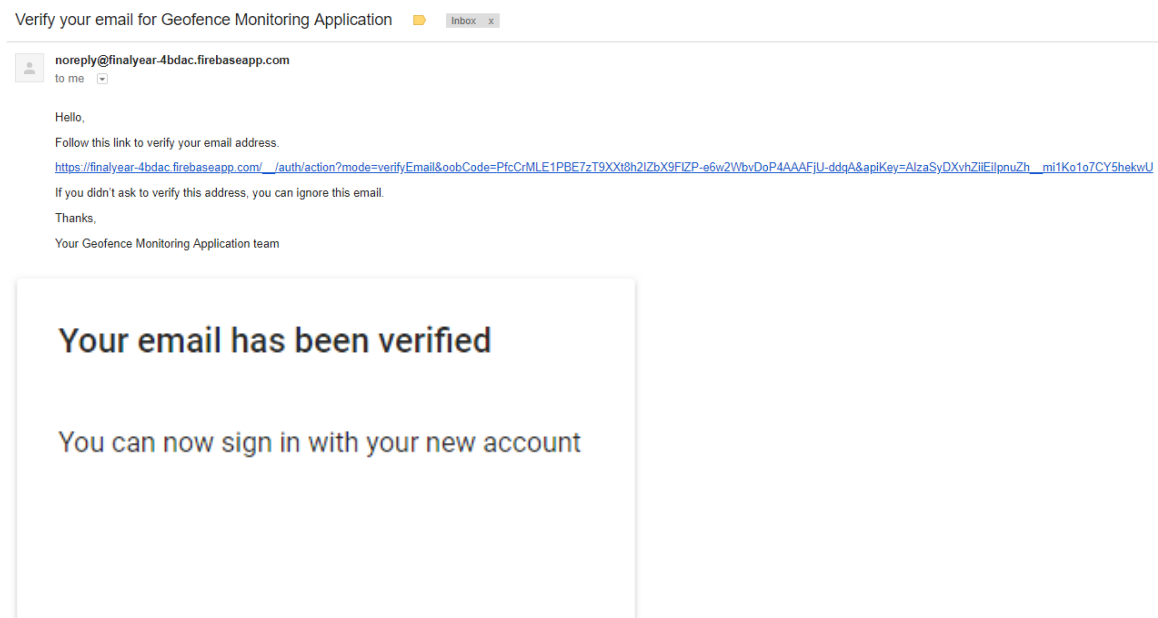
Sending verification emails is also done through the Firebase API. The user is signed in at this stage but still stuck on the activity, it's only when they have verified their email that it will start a new intent.

```

private void sendVerificationEmail() {
    user = auth.getCurrentUser();
    if (user != null) {
        user.sendEmailVerification().addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                if (task.isSuccessful()) {
                    auth.signOut();
                    Toast.makeText(getApplicationContext(), text: "Verification Email sent...", Toast.LENGTH_SHORT).show();
                } else {
                    Toast.makeText(getApplicationContext(), text: "Unable to send email", Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
}
}

```

The email a user will receive with a link to complete verification.



It's the responsibility of *checkIfEmailVerified* which is called in *signIn* to either allow the user to proceed or stop them from going any further. If verified, we make a call to *confirmUser* with the user object and this function starts a new intent bringing the user to the main activity.

```

private void checkIfEmailVerified() {
    user = auth.getCurrentUser();
    if (user != null) {
        if (user.isEmailVerified()) {
            confirmUser(user);
        } else {
            sendVerificationEmail();
            Toast.makeText(getApplicationContext(), text: "Email not verified.", Toast.LENGTH_SHORT).show();
        }
    }
}
}

```

If the user has forgotten their password and cannot gain access to their account, they have the option to reset their password given an email address that already has an account created.

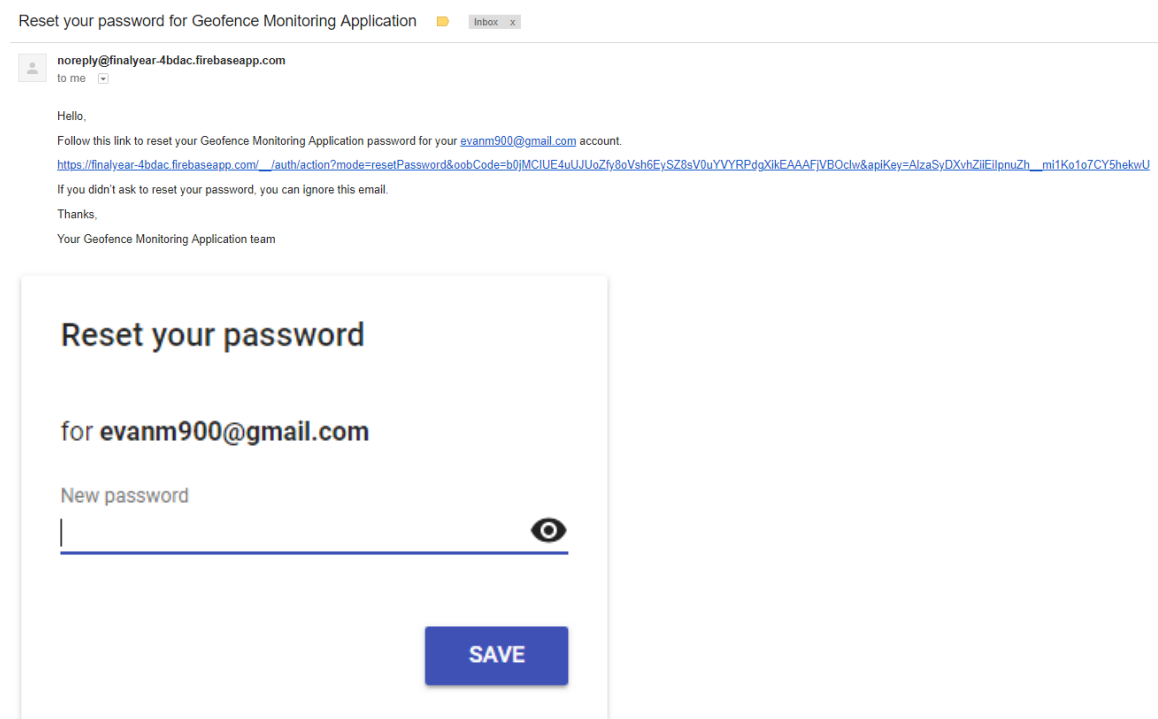

```

private void resetPassword(final String email) {
    if(email == null) {
        emailET.setError("Required.");
        return;
    }
    showProgressDialog();

    auth.sendPasswordResetEmail(email).addOnCompleteListener(new OnCompleteListener<Void>() {
        @Override
        public void onComplete(@NonNull Task<Void> task) {
            if(task.isSuccessful()) {
                Toast.makeText(getApplicationContext(), text: "Password Reset Email sent...", Toast.LENGTH_LONG).show();
            } else {
                Toast.makeText(getApplicationContext(), text: "User: " + email + " does not exist.", Toast.LENGTH_LONG).show();
            }
            hideProgressDialog();
        }
    });
}
}

```

The email a user will receive with a link to reset their password.



User Info

Once a user has access to the application we need some user object to be able to read/write any information associated with the user. As we need access to user information through every aspect of the application, I've separated out all concerns regarding user information to an instantiable class.

In this class I make use of the *ValueEventListener* attached to a *DatabaseReference* object, which allows me to retrieve all data in real-time as it updates. This functionality lives under the *getUserData* method where I've added my own custom event listener, so each activity has access to the event listener.

This means inside *onDataChange* function of the *ValueEventListener* I make a call to my own listener that an event has occurred.

```
public void getUserData() {
    dbRef.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {...}

        @Override
        public void onCancelled(DatabaseError databaseError) {
            // Failed to read value
            Log.w( tag: "Failed to read value.", databaseError.toException());
        }
    });
}
```

This is how all activities will gain full exposure to user info getters.

```
userInfo.getUserData();
userInfo.setEventListener(new UserInfoListener() {
    @Override
    public void onEvent() {
```

Now we have created a *userInfo* object with a listener attached to retrieve information on update. With this we have complete access to all getters and setters defined.

```
public void addZone(final List<LatLng> zone, final String zoneColour){...}
public void deleteZone(final List<LatLng> zone){...}
public void setPhone(String phone) { dbRef.child("phone").setValue(phone); }
public void setAlertStatus(Boolean status) { dbRef.child("notifications").setValue(status); }
public void setPatientName(String name) { dbRef.child("patient_name").setValue(name); }
public void setPatientAge(String age) { dbRef.child("patient_age").setValue(age); }
public void setPatientAbout(String about) { dbRef.child("patient_about").setValue(about); }
public FirebaseAuth getAuth() { return auth; }
public String getPhone() { return phone; }
public Double getLatitude() { return latitude; }
public Double getLongitude() { return longitude; }
public ArrayList<ArrayList<LatLng>> getZones() { return completeZoneList; }
public ArrayList<String> getZoneColours() { return completeZoneColourList; }
public JSONArray getHeartRateInfo() { return heartRateInfo; }
public String getPatientName() { return patientName; }
public String getPatientAge() { return patientAge; }
public String getPatientAbout() { return patientAbout; }
```

Main Activity

The main activity in this application makes use of the Google Maps API to allow us to show the current watch location and any defined geofence zones.

Using our *userInfo* event listener, we can just retrieve the information we need to show in this activity. Here we just want the smartwatch's current location and pre-defined geofence zones if there are any.

```
userInfo.setEventListener(new UserInfoListener() {
    @Override
    public void onEvent() {
        latitude = userInfo.getLatitude();
        longitude = userInfo.getLongitude();
        completeZoneList = userInfo.getZones();
        completeZoneColourList = userInfo.getZoneColours();

        if(latitude != null && longitude != null) {
            gMap.clear();
            LatLng watch = new LatLng(latitude, longitude);
            gMap.addMarker(new MarkerOptions().position(watch).title(watch.toString()));
            gMap.moveCamera(CameraUpdateFactory.newLatLngZoom(watch, 16.0f));
        }

        if(!completeZoneList.isEmpty() && !completeZoneColourList.isEmpty() && (completeZoneList.size()==completeZoneColourList.size())){
            for(int i=0; i<completeZoneList.size(); i++){
                createPolygon(completeZoneList.get(i), completeZoneColourList.get(i));
            }
        }
    }
});
```

If we have pre-defined geofence zones then we want to draw them on the Google Map to display to the user, so for each zone in our list of zones we call the *createPolygon* method given the zone details in the form of an *ArrayList<LatLng>* and the zone colour. Then we just add our polygon to the map given our *PolygonOptions*.

```
public void createPolygon(ArrayList<LatLng> zoneList, String zoneColour) {
    int fillColor = 0;
    if(zoneColour.equals("Green")){
        fillColor = 0x5000ff00;
    } else if(zoneColour.equals("Yellow")) {
        fillColor = 0x50ffff00;
    } else if(zoneColour.equals("Red")){
        fillColor = 0x50ff0000;
    }
    PolygonOptions polygonOptions = new PolygonOptions();
    polygonOptions.addAll(zoneList);
    polygonOptions.strokeColor(Color.BLUE);
    polygonOptions.fillColor(fillColor);
    polygonOptions.strokeWidth(7);
    gMap.addPolygon(polygonOptions);
}
```

Geofence Activity

Slightly similar to the main activity we too also create our Google Map object and plot the current smartwatch location and any pre-defined zones. In this activity we want to allow the user to create additional geofence zones. Making use of listeners associated with Google Maps such as *onMapLongClick*, *onMarkerClick* and *onPolygonClick* listeners.

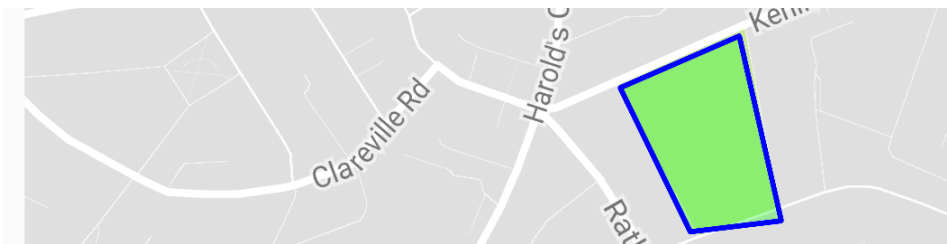
I've made use of the *onMapLongClick* listener to allow the user plot out markers in which they want to draw their geofence zone. When each extra marker is plotted we make a call to *reDrawPolylines* which clears any existing lines and reconnects the points in the list, it then makes a call to *createPolygon* when there are enough points available.

```
@Override
public void onMapLongClick(LatLng location) {
    point = location;
    listGeofencePoints.add(point);
    mMap.addMarker(new MarkerOptions().position(point).title(point.toString()));

    reDrawPolylines();
    Toast.makeText(getApplicationContext(), "Click marker's to remove", Toast.LENGTH_SHORT).show();
}
```

```
public void reDrawPolylines() {
    PolylineOptions polylineOptions = new PolylineOptions().color(Color.BLUE);
    Polyline polyline;

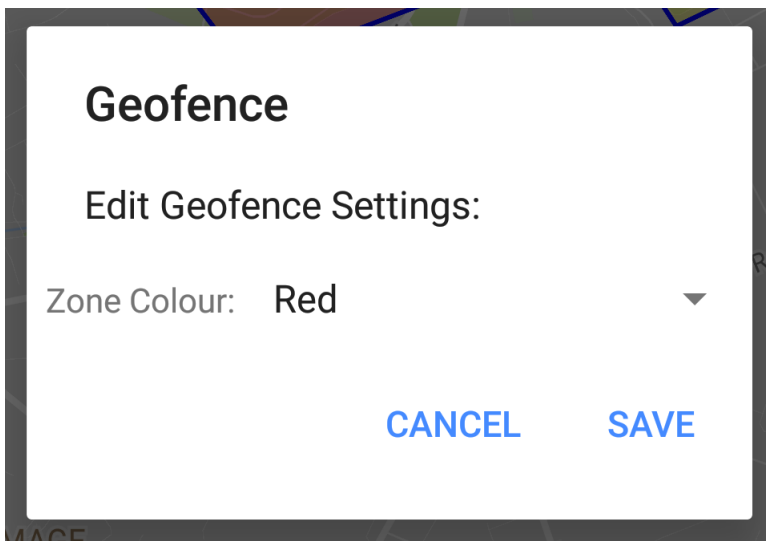
    for(Polyline line : polylineList) {
        line.remove();
    }
    polylineList.clear();
    for(int i = 0; i < listGeofencePoints.size(); i++) {
        polylineOptions.add(listGeofencePoints.get(i));
        polyline = mMap.addPolyline(polylineOptions);
        polylineList.add(polyline);
    }
    if(listGeofencePoints.size() > 2) {
        createPolygon(listGeofencePoints, "Green");
    }
}
```



Once a user has plotted their desired zone and they are satisfied, we have provided access to the *onPolygonClick* listener. With this we inflate an *AlertDialog* display which allows the user to select their zone colour from green, yellow or red. Upon clicking save we make a call to *savePolygon* so that geofence zone is now stored as a defined zone in Firebase.

```
@Override
public void onPolygonClick(Polygon polygon) {
    final List<LatLng> listPoints = polygon.getPoints().subList(0, polygon.getPoints().size()-1);
    view.setVisibility(View.VISIBLE);

    AlertDialog.Builder alert = new AlertDialog.Builder( context: this);
    alert.setTitle("Geofence");
    alert.setMessage("Edit Geofence Settings:");
    alert.setView(view);
    alert.setPositiveButton( text: "Save", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            ((ViewGroup) view.getParent()).removeView(view);
            dialog.dismiss();
            savePolygon(listPoints, zoneColour);
        }
    });
    alert.setNegativeButton( text: "Cancel", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            ((ViewGroup) view.getParent()).removeView(view);
            dialog.dismiss();
        }
    });
    alert.setNeutralButton( text: "Delete", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            ((ViewGroup) view.getParent()).removeView(view);
            dialog.dismiss();
            deleteZone(listPoints);
        }
    });
    alert.show();
}
```



Extra Info Activity

This activity is responsible for displaying any additional information about the smartwatch user such as heart rate info. As mentioned earlier due to hardware and software complications I've had to mock this data, but the implementation doesn't change. Once again, we are going to define a *userInfo* event listener to read our data. The heart rate data is stored as a *JSONArray*, that is an array made up of a list of json objects. Each json object holds the *heart_rate* which is an integer and *date_time* as an *epoch*, that is the number of milliseconds that have elapsed since January 1st, 1970. I turn the *epoch* into a *Date* object so that I can format it later to just strip out the timestamp. Each time the heart rate object in the database gets updated we call *updateGraph*.

```
userInfo.setEventListener(new UserInfoListener() {
    @Override
    public void onEvent() {
        heartRateInfo = userInfo.getHeartRateInfo();
        ArrayList<Integer> heartRate = new ArrayList<>();
        ArrayList<Date> dateTime = new ArrayList<>();
        for(int i=0; i<heartRateInfo.length(); i++){
            try {
                int hr = Integer.valueOf(heartRateInfo.getJSONObject(i).getString( name: "heart_rate"));
                String dt = heartRateInfo.getJSONObject(i).getString( name: "date_time");
                long epoch = Long.parseLong(dt);
                Date date = new Date(epoch*1000);

                heartRate.add(hr);
                dateTime.add(date);
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
        updateGraph(heartRate, dateTime);
    }
});
```

Using a library called Androidplot, I can create a series of heart rate data. I've implemented a custom formatter and overridden the default configuration of labelling the Y axis in increments of 1,2,3 etc. I've set the labels to be the timestamp of each heart rate value. We then call *redraw* on the plot object.

```
public void updateGraph(ArrayList<Integer> heartRate, final ArrayList<Date> dateTime){
    plot.clear();
    XYSeries series = new SimpleXYSeries(heartRate, SimpleXYSeries.ArrayFormat.Y_VALS_ONLY, title: "Series");
    LineAndPointFormatter seriesFormat = new LineAndPointFormatter(Color.RED, Color.DKGRAY, fillColor: null, pH: null);
    seriesFormat.getVertexPaint().setStrokeWidth(30f);
    seriesFormat.setInterpolationParams(new CatmullRomInterpolator.Params( pointPerSegment: 20, CatmullRomInterpolator.Type.Centripetal));
    /*
     Custom formatter to override default configuration
     Replaces Y axis label values of 1,2,3... etc with the Time value associated with each X value point
    */
    plot.getGraph().getLineLabelStyle(XYGraphWidget.Edge.BOTTOM).setFormat(new Format() {
        @Override
        public StringBuffer format(Object obj, StringBuffer toAppendTo, FieldPosition pos) {
            int i = Math.round(((Number) obj).floatValue());
            Date date = dateTime.get(i);
            SimpleDateFormat sdf = new SimpleDateFormat( pattern: "HH:mm:ss");
            return sdf.format(date, toAppendTo, pos);
        }
    });
    @Override
    public Object parseObject(String source, ParsePosition pos) { return null; }
});
plot.addSeries(series, seriesFormat);
plot.redraw();
}
```

User Profile Activity

This activity is responsible for the users account configuration. They have the ability to toggle alerts on/off, opt into text alerts, change their account email, or change their account password.

Toggling alerts on and off is done through a simple `ToggleButton`, if it is checked it sets the alert status to be true, or else it must be false. This status is checked in our Cloud Functions and it defines whether we will send notification alerts or not.

```
alertTglBtn.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if(isChecked){
            userInfo.setAlertStatus(true);
            Toast.makeText(getApplicationContext(), text: "Notification alerts turned on.", Toast.LENGTH_LONG).show();
        } else {
            userInfo.setAlertStatus(false);
            Toast.makeText(getApplicationContext(), text: "Notification alerts turned off.", Toast.LENGTH_LONG).show();
        }
    }
});
```

If the user wishes to opt into text alerts which uses Twilio API they can activate the click listener on the opt in button which makes a call to `verifyNumber`.

Using the `PhoneAuthProvider` from Firebase we can make a call to `verifyPhoneNumber` with the number that needs to be verified, a timeout for the verification code to be entered and the callback for this function.

```
public void verifyNumber(){
    PhoneAuthProvider.getInstance().verifyPhoneNumber(phoneNumber, 60, TimeUnit.SECONDS, activity, this, phoneCallback);
    auth.setLanguageCode("IE");
}
```

Once the user enters the verification code and activates the click listener on the verify button then we make a call to `verifyPhoneNumberWithCode` given the `verificationId` provided in the callback and the `code` they entered.

```
public void verifyPhoneNumberWithCode(String verificationId, String code){
    PhoneAuthCredential credential = PhoneAuthProvider.getCredential(verificationId, code);
    phoneCallback.onVerificationCompleted(credential);
}
```

The callback is defined here with multiple states.

```

phoneCallback = new PhoneAuthProvider.OnVerificationStateChangedCallbacks() {
    @Override
    public void onVerificationCompleted(PhoneAuthCredential phoneAuthCredential) {
        Toast.makeText(getApplicationContext(), text: "Successfully Verified Number: " + phoneNumber, Toast.LENGTH_SHORT).show();
        userInfo.setPhone(phoneNumber);
        finish();
        startActivity(getIntent());
    }

    @Override
    public void onVerificationFailed(FirebaseException e) {
        Toast.makeText(getApplicationContext(), text: "Error encountered verifying code", Toast.LENGTH_LONG).show();
    }

    @Override
    public void onCodeSent(String verificationId, PhoneAuthProvider.ForceResendingToken token) {
        Toast.makeText(getApplicationContext(), text: "Verification code sent to: " + phoneNumber, Toast.LENGTH_SHORT).show();
        codeVerification = verificationId;
        phoneET.setVisibility(View.GONE);
        phoneBtn.setVisibility(View.GONE);
        verifyCodeET.setVisibility(View.VISIBLE);
        verifyCodeBtn.setVisibility(View.VISIBLE);
    }
};

```

If the user wishes to change their account email they need to provide their new email address and their current password. On button click of change email we make a call to *sendVerificationEmail*, similar to what we do when creating a user account. This time it's slightly different.

We need to *reauthenticate* the current user first, so we have access to the *updateEmail* function provided by Firebase. Upon successful change of their email we then proceed to *sendEmailVerification* and on success of that we sign the current user out as they must now verify their new email.

```

public void sendVerificationEmail(final String email, String password) {
    user = auth.getCurrentUser();
    AuthCredential credential = EmailAuthProvider.getCredential(user.getEmail(), password);
    user.reauthenticate(credential).addOnCompleteListener(new OnCompleteListener<Void>() {
        @Override
        public void onComplete(@NonNull Task<Void> task) {
            if(task.isSuccessful()) {
                user.updateEmail(email).addOnCompleteListener(new OnCompleteListener<Void>() {
                    @Override
                    public void onComplete(@NonNull Task<Void> task) {
                        if(task.isSuccessful()) {
                            user.sendEmailVerification().addOnCompleteListener(new OnCompleteListener<Void>() {
                                @Override
                                public void onComplete(@NonNull Task<Void> task) {
                                    if(task.isSuccessful()) {
                                        auth.signOut();
                                        Intent logoutIntent = new Intent(getApplicationContext(), LoginRegistrationActivity.class);
                                        startActivity(logoutIntent);
                                        Toast.makeText(getApplicationContext(), text: "Verification Email sent...", Toast.LENGTH_SHORT).show();
                                    } else {
                                        Toast.makeText(getApplicationContext(), text: "Unable to send email", Toast.LENGTH_SHORT).show();
                                    }
                                }
                            });
                        } else {
                            Toast.makeText(getApplicationContext(), text: "Error updating email", Toast.LENGTH_LONG).show();
                        }
                    }
                });
            } else {
                Toast.makeText(getApplicationContext(), text: "Error authenticating user", Toast.LENGTH_LONG).show();
            }
        }
    });
}

```

If the user wishes to change their password, they can do so by providing their current password and their new password. The change password button makes a call to *changePassword*. Here we need to validate that the new password they entered matches our criteria. We must also *reauthenticate* the user to gain

access to `updatePassword`. On success we just refresh the activity to clear any fields.

```
public void changePassword(){
    if(validatePasswordFields(newPasswordET.getText().toString(), repeatPasswordET.getText().toString())) {
        user = userInfo.getAuth().getCurrentUser();
        AuthCredential credential = EmailAuthProvider.getCredential(user.getEmail(), currentPasswordET.getText().toString());
        user.reauthenticate(credential).addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                if(task.isSuccessful()){
                    user.updatePassword(newPasswordET.getText().toString()).addOnCompleteListener(new OnCompleteListener<Void>() {
                        @Override
                        public void onComplete(@NonNull Task<Void> task) {
                            if(task.isSuccessful()){
                                finish();
                                startActivity(getIntent());
                                Toast.makeText(getApplicationContext(), text: "Password has been changed.", Toast.LENGTH_LONG).show();
                            } else {
                                Toast.makeText(getApplicationContext(), text: "Error changing password.", Toast.LENGTH_LONG).show();
                            }
                        }
                    });
                } else {
                    Toast.makeText(getApplicationContext(), text: "Error authenticating user", Toast.LENGTH_LONG).show();
                }
            }
        });
    }
}
```

Patient Profile Activity

The patient profile does not have much impact on the functionality of the application but rather it exists for future development. It allows a user to save information about the patient, such as name, age, and anything about them which could maybe contain information on any health problems they might have etc. The idea is that this information could be used in emergency circumstances, in the case this application was incorporated with the emergency services, they would receive all this information on call.

On loading this activity, we need to check that the user has permissions enabled to gain read access to their devices external storage. This only occurs in the case of the user's device being higher than SDK 22. If permissions aren't enabled, they are prompted to enable them.

```
public boolean checkPermissions(){
    if (Build.VERSION.SDK_INT >= 23) {
        if (checkSelfPermission(Manifest.permission.READ_EXTERNAL_STORAGE) == PackageManager.PERMISSION_GRANTED) {
            return true;
        } else {
            ActivityCompat.requestPermissions( activity this, new String[]{android.Manifest.permission.READ_EXTERNAL_STORAGE}, requestCode: 1);
            return false;
        }
    }
    else {
        return true;
    }
}
```

We have a `FloatingActionButton` with an on click listener which toggles the editable fields on/off. This makes a call to `updateInfo` with values from each `EditText` field.

```
public void updateInfo(String name, String age, String about){
    userInfo.setPatientName(name);
    userInfo.setPatientAge(age);
    userInfo.setPatientAbout(about);
}
```

We can also set photo for the patient for identification purposes. I've used an `ImageButton` with an on click listener to allow the user to select an image from the gallery on their device.

```
imageButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent gallery = new Intent(Intent.ACTION_PICK, MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
        gallery.setType("image/*");
        startActivityForResult(gallery, requestCode);
    }
});
```

This triggers the `onActivityResult` which contains the intent data containing the Uri of the image selected. We set the `ImageButton` to the Uri and make call to `updatePicture` to handle saving the image to Firebase.

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data){
    super.onActivityResult(requestCode, resultCode, data);
    if(requestCode == 0 && resultCode == RESULT_OK){
        Uri uri = data.getData();
        imageButton.setImageURI(uri);
        updatePicture(uri);
    }
}
```

This is responsible for setting the photo associated with the user. We can create a `UserProfileChangeRequest` object to `setPhotoUri` of the user, we then make a call to `updateProfile` with our change request.

```
public void updatePicture(Uri uri){
    UserProfileChangeRequest profile = new UserProfileChangeRequest.Builder().setPhotoUri(uri).build();
    user.updateProfile(profile).addOnCompleteListener(new OnCompleteListener<Void>() {
        @Override
        public void onComplete(@NonNull Task<Void> task) {
            if(task.isSuccessful()) {
                Toast.makeText(getApplicationContext(), text: "Updated patient image", Toast.LENGTH_LONG).show();
            } else {
                Toast.makeText(getApplicationContext(), text: "Unable to update patient image", Toast.LENGTH_LONG).show();
            }
        }
    });
}
```

2.3.2 Tizen

The smartwatch application runs on Tizen OS but is written purely in HTML and JavaScript as it is considered a web application.

Setup

Just like a normal web application with scripting, I've got to link all the relevant external scripts that I'm using. In this case being Firebase provided with some configuration to initialise the application. This will allow me to access Firebase freely in my JavaScript functions.

```
<script src="https://www.gstatic.com/firebasejs/4.9.0/firebase-app.js"></script>
<script src="https://www.gstatic.com/firebasejs/4.9.0/firebase-auth.js"></script>
<script src="https://www.gstatic.com/firebasejs/4.9.0/firebase-database.js"></script>
<script>
// Initialize Firebase
var config = {
  apiKey: "AIzaSyDXvhZiiEiIpnuZh__mi1Ko1o7CY5hekWU",
  authDomain: "finalyear-4bdac.firebaseio.com",
  databaseURL: "https://finalyear-4bdac.firebaseio.com",
  projectId: "finalyear-4bdac",
  storageBucket: "finalyear-4bdac.appspot.com",
  messagingSenderId: "542107190898"
};
firebase.initializeApp(config);
</script>
```

View

The presentation of my smartwatch application is very simple as it's only functionality is to transmit data in the background. To do so we must first allow the user to login. They must have created an account through the Android application first. Using a simple form calling JavaScript functions, they are provided with this screen.

```
<form id="signin">
  <input class="menu" type="email" name="Email" placeholder="Email here" id="email"></br>
  <input class="menu" type="password" name="Password" placeholder="Password here" id="password"></br>
  <input class="menu" type="button" value="Signin" onclick="signin()">
</form>
<div id="textbox"></div>
<button class="menu" id="signout" onclick="signout()">Signout</button>
```

Initialisation

Before I begin implementing any functionality to my application I need to tell Tizen to keep the CPU awake for my application to run whilst in the background.

```
tizen.power.request("CPU", "CPU_AWAKE");
```

Using a *window.onLoad* I initialise my application using Firebase's *onAuthStateChanged* function. Upon a user existing, I make a call to *setLocation* and create a *setInterval* for this function to be called every 30 seconds thereafter.

```
function initApp(){
  firebase.auth().onAuthStateChanged(function(user) {
    if (user) {
      // User is signed in.
      userid = user.uid;
      console.log(userid);
      document.getElementById('signin').style.display = 'none';
      document.getElementById('signout').style.display = 'block';
      getLocation();
      // Get current location every 30 seconds
      setInterval(getLocation, 30000);
    } else {
      document.getElementById('signout').style.display = 'none';
      console.log("Not signed in.");
    }
  });
}
```

Location

The *setLocation* function is the core part of my smartwatch application. Using the navigator object in JavaScript I can get access to the current geolocation of the smartwatch. Upon being successful I read in the latitude and longitude coordinates and then store them under the user's id in the Firebase Realtime Database. These values will be read in by the Android application to display current user's location and also read in by the Cloud Functions to check if the user has entered a zone or not.

```
function getLocation(){
    var location = document.getElementById("textbox");

    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(function(position){
            var latitude = position.coords.latitude;
            var longitude = position.coords.longitude;
            location.innerHTML = "Latitude: " + latitude + "<br>Longitude: " + longitude;
            firebase.database().ref(userid+'/').update({
                latitude: latitude,
                longitude: longitude
            });
            console.log("Latitude: " + latitude + " - " + "Longitude: " + longitude);
        });
    } else {
        console.log("erro");
        location.innerHTML = "Geolocation is not supported by this browser.";
    }
}
```

Heart Rate

Due to complications I encountered revolving around the smartwatch, I could not implement the heart rate monitor as I had originally desired. Due to the hardware and software I had access to, it was no longer supported. To work around this, I decided to mock the implantation of the heart rate just to demonstrate how it should work given the correct resources. I used Mockaroo's API to make calls to a schema I had pre-defined and, on each call, I would read the mocked data and then push it to Firebase. Due to additional complications with the smartwatch I could not make the HTTPS requests from it as the watch could not decode and verify the SSL Certificates, it worked fine with HTTP requests, but Mockaroo uses HTTPS. I then decided to put the mocking functionality into my Cloud Functions as they only get triggered when the watch location updates, so the mocked heart rate would also update at the same time. I make a call to Mockaroo using *fetch* and read in the values, I then check any existing values I had previously pushed for the user, if the *epoch* timestamp is over 24 hours old they are ignored, if not then we build up a *JSONArray* and push all the values to Firebase.

```
function updateHeartRate(userId) {
  let heart_rate;
  let date_time;
  let existingData = [];
  fetch(`http://${domain}/heart_rate.json`, { headers })
    .then(response => response.json())
    .then(data => {
      heart_rate = data.heart_rate;
      date_time = data.date_time;
      let json = {
        "heart_rate": heart_rate,
        "date_time": date_time
      };
      return admin.database().ref(userId+'/heart_rate').once('value', snapshot => {
        if(snapshot.val() !== null) {
          for (let i = 0; i < snapshot.val().length; i++) {
            let json = {
              "heart_rate": snapshot.val()[i].heart_rate,
              "date_time": snapshot.val()[i].date_time
            };
            let time = new Date().getTime()/1000;
            if((time-json.date_time) < 86400){
              existingData.push(json);
            }
          }
        }
        existingData.push(json);
        return admin.database().ref(userId).child('heart_rate').set(existingData);
      });
    });
}
```

2.3.3 Cloud Functions

Setup

Initial setup for cloud functions, requires in all the npm package's that I need, along with loading the Twilio and Mockaroo credentials that have been pre-set in the Firebase functions configuration.

```
// Require firebase functions/auth
const functions = require('firebase-functions');
const admin = require('firebase-admin');
// npm package to check if latlng point is inside polygon
const checkInside = require('point-in-polygon');
// npm package to fetch api data from mockaroo
const fetch = require("node-fetch");
const domain = 'my.api.mockaroo.com';
const headers = { "X-API-Key": functions.config().mockaroo.api };
}

/*
  Require Twilio service
  Firebase Flame or Blaze plan required to make calls to external api's
*/
const twilio = require('twilio');
// Set these credentials using cmd line with: firebase functions:config:set twilio.sid="CHANGE_ME" twilio.token="CHANGE_ME"
const accountSid = functions.config().twilio.sid;
const authToken = functions.config().twilio.token;
const client = new twilio(accountSid, authToken);

// Initialise app
admin.initializeApp();
```

Event Trigger

This is the cloud function I've written to trigger on changes in the database, when the watch location updates, this function triggers every time and plucks the relevant data from the database.

```
exports.checkLatLng = functions.database.ref('{userId}/latitude').onUpdate((snapshot, context) => {
  const userId = context.params.userId.toString();
  updateHeartRate(userId);
  let latitude;
  let longitude;
  let notifications;
  let tokenId;
  let phone;

  return admin.database().ref(userId).once('value', snapshot => {
    snapshot.forEach(data =>{...});
  });
});
```

Once triggered it reads in all the defined zones from the database, creating a new array list containing the latitude and longitude of each point. For each zone it will subsequently activate the *checkLocation* function only if the users notifications are toggled on. If a phone number has been found for the user which was checked earlier in the code, then both notifications and text alerts could get triggered.

```

if(key==='zones'){
  data.forEach(zoneData =>{
    let zone=[];
    let colour;
    console.log("ZoneKey: "+zoneData.key.toString());
    for(let i=0; i<zoneData.val().length; i++){
      if(i===zoneData.val().length-1){
        colour = zoneData.val()[i].colour;
      } else {
        let lat = zoneData.val()[i].latitude;
        let lng = zoneData.val()[i].longitude;
        zone.push([lat, lng])
      }
    }
    // If notification alerts are turned on then proceed
    if(notifications==='true') {
      if (latitude && longitude && zone && colour && tokenId && phone) {
        // Notification & text
        checkLocation(latitude, longitude, zone, colour, tokenId, phone);
      } else if (latitude && longitude && zone && colour && tokenId) {
        // Just Notification
        checkLocation(latitude, longitude, zone, colour, tokenId);
      }
    }
  });
}

```

The *checkLocation* method takes in multiple parameters and uses the variable *checkInside* we had set earlier which uses the *point-in-polygon* npm package. This package accepts in the latitude and longitude point we want to check against the list of points of each zone, returning true or false. On true alerts are triggered, if the *phone* parameter was passed in initially then a text alert is also triggered.

```

function checkLocation(latitude, longitude, zone, colour, tokenId, phone){
  console.log(latitude, longitude, zone, colour);
  if(checkInside([latitude, longitude], zone)) {
    console.log("Inside zone");
    if(colour === "Green") {
      sendNotification(tokenId, colour);
    } else if(colour === "Yellow" || colour === "Red") {
      sendNotification(tokenId, colour);
      if (phone) {
        sendTextMessage(phone, colour);
      }
    }
  } else {
    console.log("Not in zone");
  }
}

```


Notifications

This method is responsible for sending push notifications to the user's device using the *tokenId* which is pushed and stored from the Android device in the database. We use *admin* which is a variable we set at the top of the file to access the *firebase-admin* functions which include Cloud Messaging, used for push notifications.

```
function sendNotification(tokenId) {
  let notificationPayload = {
    token: tokenId,
    notification: {
      title: "Geofence Monitor",
      body: "ALERT !!!\nUser has entered an unsafe area.\nOpen to view current status."
    },
    android: {
      notification: {
        sound: "default"
      }
    }
  };
  admin.messaging().send(notificationPayload).then((response) =>{
    console.log("Success sending: " + response);
  }).catch((error) =>{
    console.log("Error sending message: " + error);
  });
}
```

Text Alerts

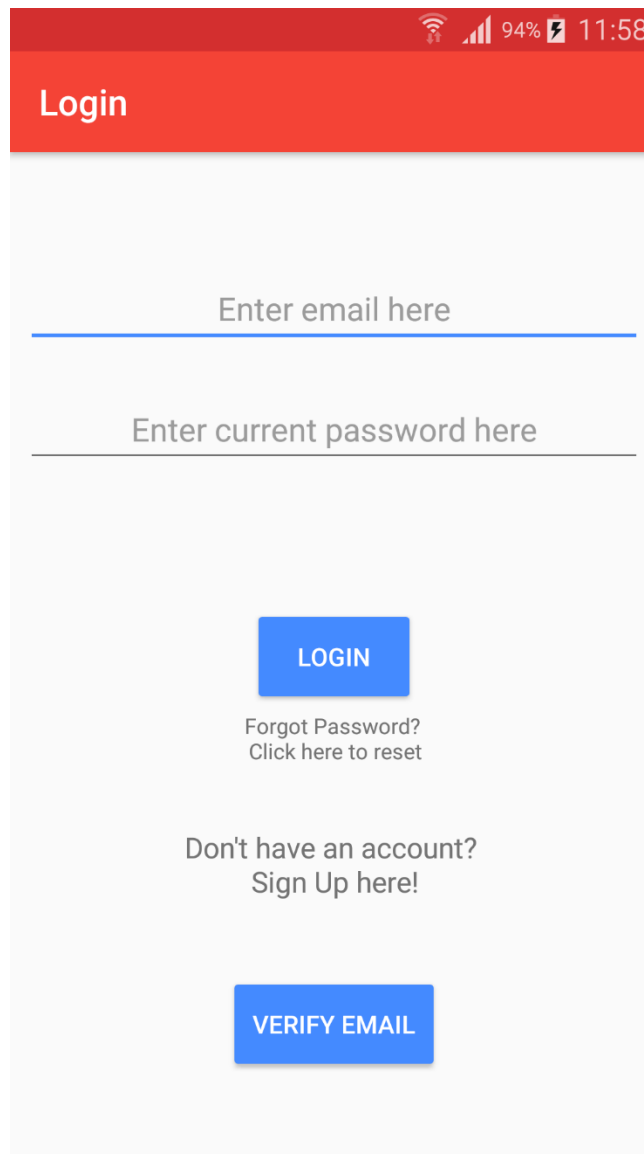
This method is responsible for sending text alerts through the Twilio API, only activated if the user has a phone number associated with their account. The *from* number is the number assigned to my Twilio account.

```
function sendTextMessage(phone) {
  console.log(phone);
  const client = new twilio(accountSid, authToken);
  client.messages.create({
    body: "Automated text ALERT !!!\nUser has entered an unsafe area.\nOpen application for more info.",
    to: phone,
    from: '+16308668643',
  }).then((message) => {
    console.log(message.sid);
  }).catch((error) => {
    console.log(error);
  });
}
```

2.4 Graphical User Interface (GUI) Layout

2.4.1 User Login/Registration Screen

This is in the initial screen the user will see when opening the application for the first time. This screen is used for user logins, user registration, password reset, and sending verification emails, it will just accept their email and password, if their credentials are correct they will be brought to the main screen. If the user has forgotten their password, they can click the password reset option and they will be sent an email to reset their password. If the user has registered but not verified their email they can enter their details and resend the email. If the user hasn't registered yet they can choose the signup option where it will automatically sign them up with the email and password provided. If the user is already logged in, they will be brought straight to the main screen.



94% 11:58

Login

Enter email here

Enter current password here

LOGIN

Forgot Password?
Click here to reset

Don't have an account?
Sign Up here!

VERIFY EMAIL

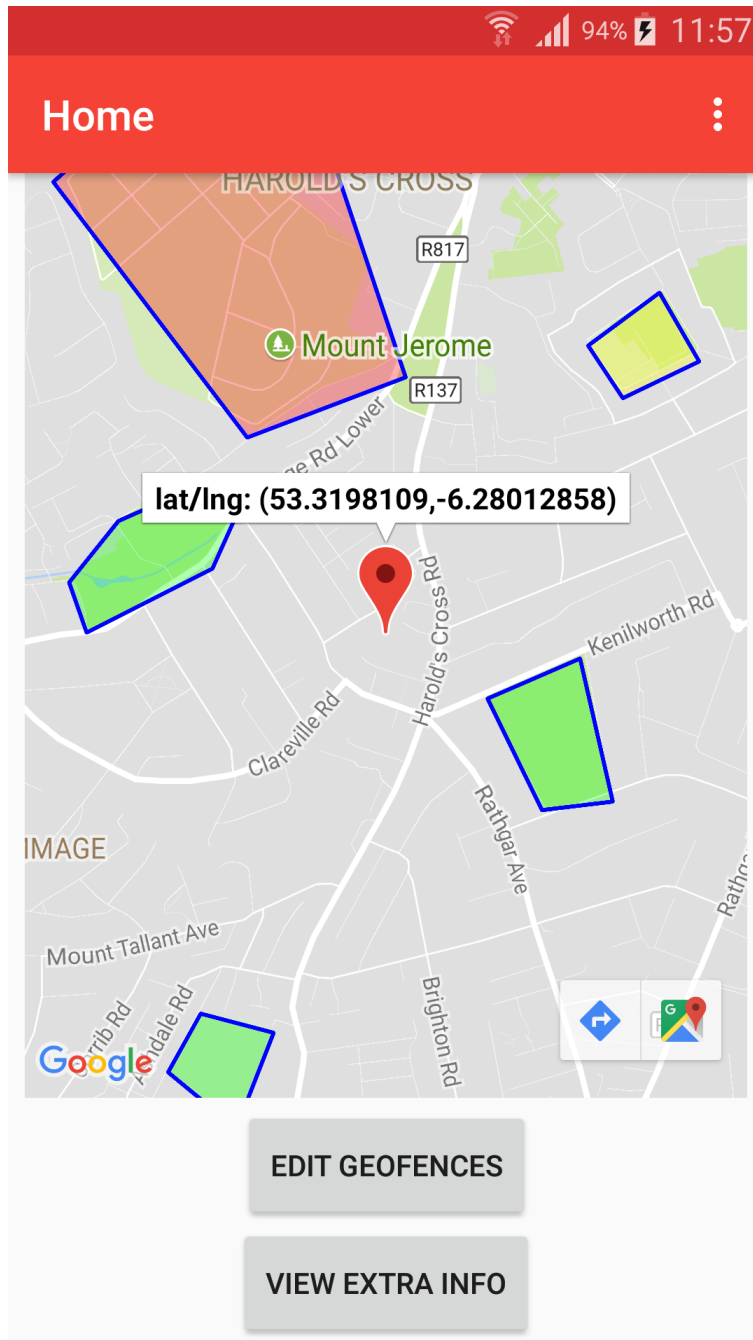
2.4.2 User Profile Screen

In here the user will be able to update their user profile. Here they can opt into text alerts if they wish to, they will have to provide their phone number in which they will receive a verification code to then enter. They can change their email and change their password if they wish to, given the correct credentials.

The screenshot shows a mobile application interface for the 'User Profile' screen. At the top, there is a red header bar with a back arrow on the left, the text 'User Profile' in the center, and a three-dot menu icon on the right. Below the header, the main content area is white. It starts with the heading 'Update Your Profile!'. Underneath, there is a toggle switch for 'Receive Notification Alerts?' which is currently set to 'ON'. Below this is the text 'Text Alerts?' followed by a text input field with the placeholder 'Enter in the format of: +353860000000'. A large grey button labeled 'OPT IN' is positioned below the input field. Further down, there is a section for 'Change Email?' with a text input field containing the placeholder 'Enter email here'. Below that is another text input field with the placeholder 'Enter current password here'. A large grey button labeled 'CHANGE EMAIL' is located below the second input field. At the bottom, there is a section for 'Change Password?' with a text input field containing the placeholder 'Enter current password here'.

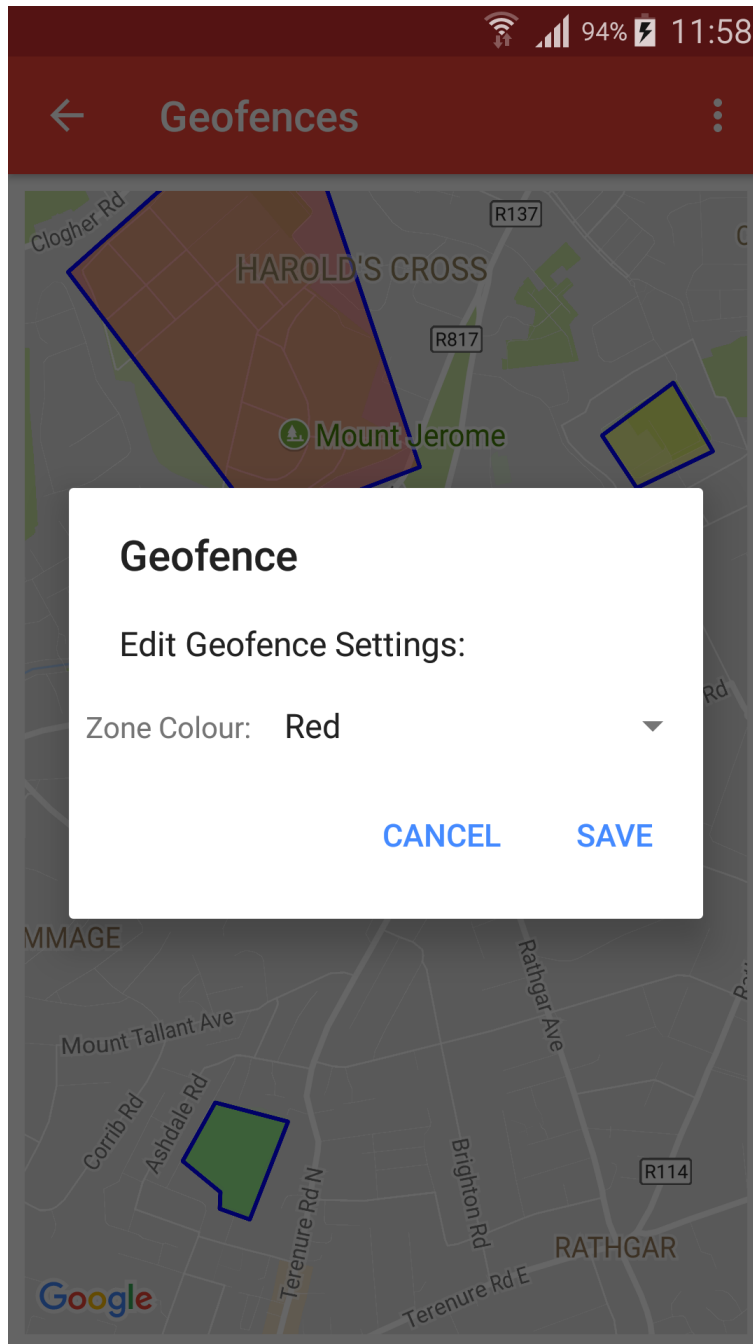
2.4.3 Main Screen

Here the main screen of the application is where the user can access every feature. Once setup they will be able to see the current location of the watch and all their geofences when they've been setup. They will be able to access the menu bar dropdown and two other screens, one to configure their geofences and the other to view additional information from the watch such as heart rate.



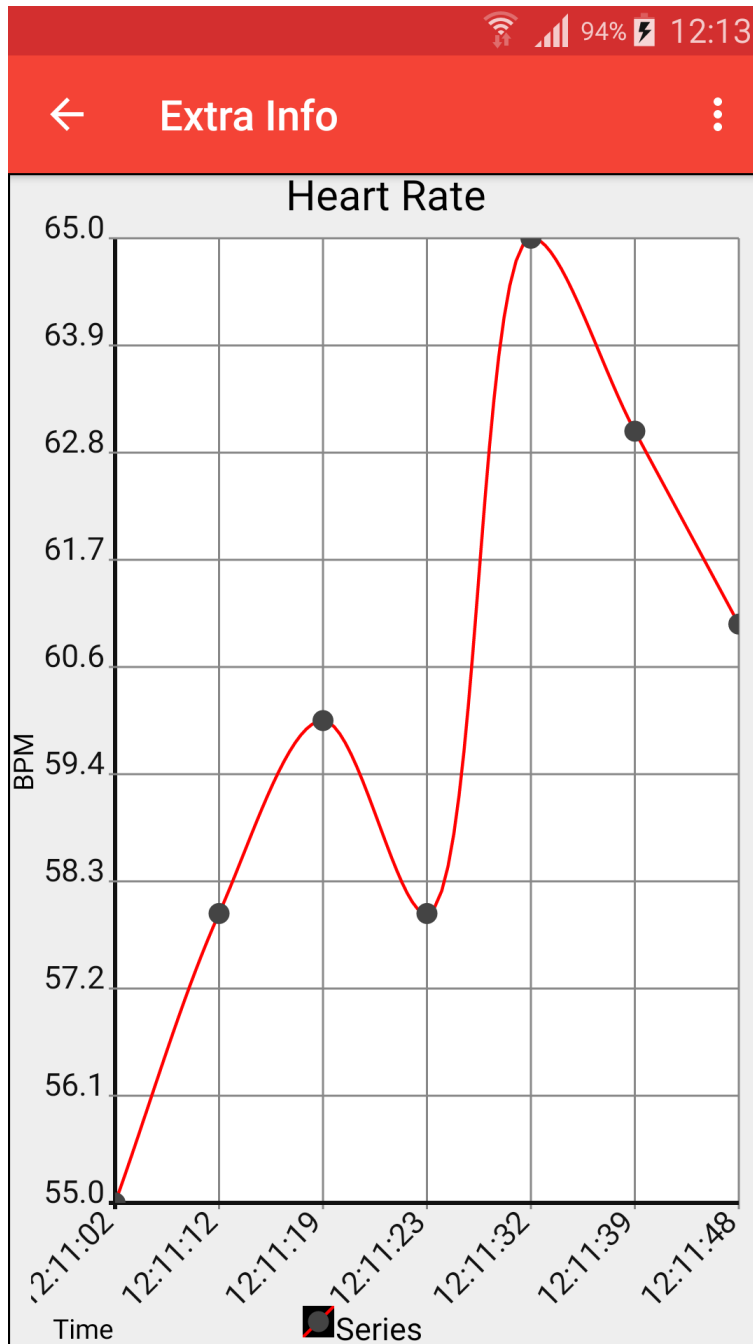
2.4.4 Edit Geofences Screen

This screen is the most important part of the application as it allows the user to configure their geofences. They will be able to select multiple points on the map where it will create a polygon zone and define it's setting by choosing the zone colour from green/yellow/red, the days of the week they would like the geofence to be active, it may not be necessary to have alerts 24/7 for example. Any other geofences that exist on this map they can click on and edit or remove them.



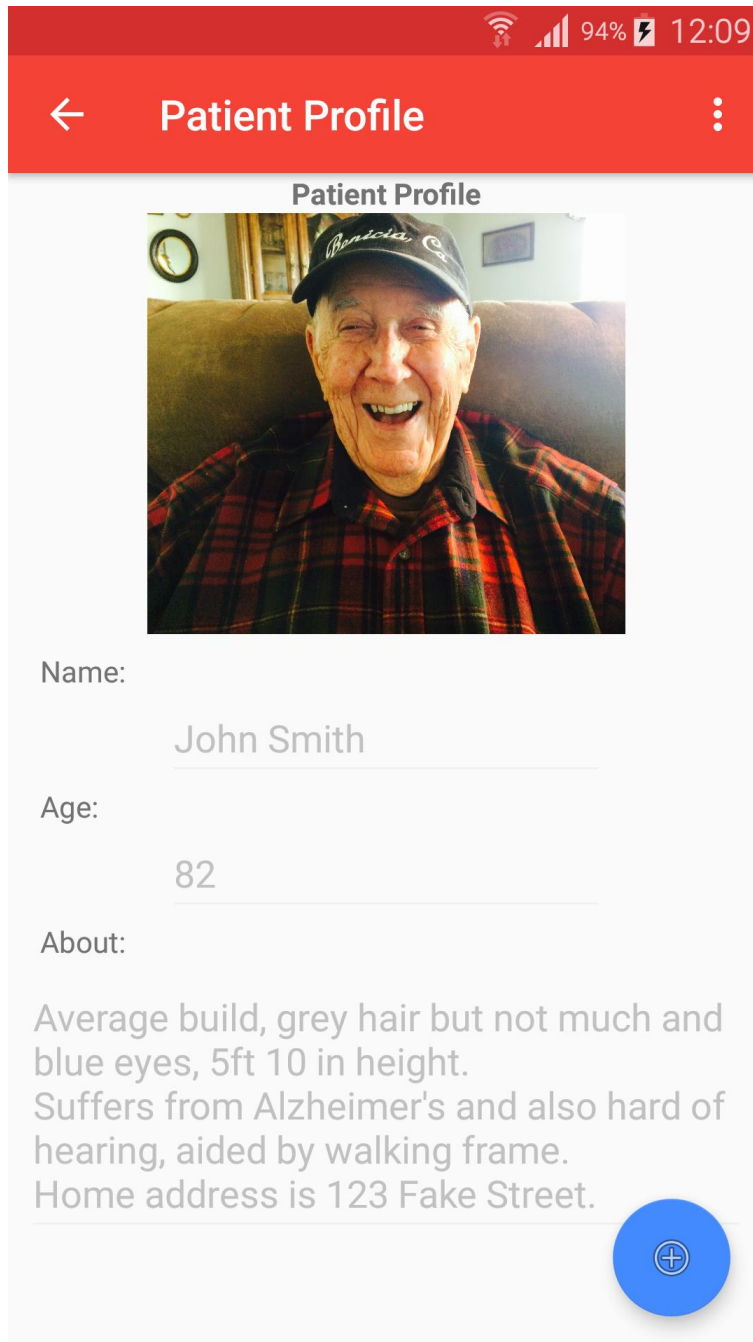
2.4.5 View Extra Info Screen

This part of the application will allow the user to view historical information up to the last month on the watch user such as their daily heart rate, sleep schedule, a step counter. I may add other functionality as I develop, possibly a map containing data of where the watch user was located on that day.



2.4.6 User Profile Screen

This part of the application allows a user to customise the profile of the patient being monitored. This could be useful in future developments of the application, if it were incorporated with the emergency services, this information would be sent to them etc. Using the FloatingActionButton they can toggle editing on/off.



2.5 Application Programming Interfaces (API)

- Google Maps API
My application will make use of the Google Maps API allowing the ability for defining geofences and displaying location information of the smartwatch.
- Firebase API
Will be used for my user registration/login authentication, email verifications, password reset, real-time database and cloud functions.
- Twilio API
Twilio's service will be used for automated text alerts, if the user has opted in, the user will receive a text message if the watch wearer has gone beyond a red zone or activated the panic button option.
- Mockaroo API
I've had to make use of Mockaroo to mock the heart rate data that would have been generated by the smartwatch given the appropriate hardware and software.

2.6 Testing

2.6.1 Unit Testing

I have written a select amount of unit tests for my Android application. Due to the application relying heavily on external services like Google Maps API and Firebase API, it was impossible for me to test raw implementation of my code. So I have used Mockito as my mocking framework to do this.

Due to all my tests being mocked I have not unit tested my whole application, instead I have unit tested the more important parts of my application like authentication and receiving user data. The tests that I have implemented demonstrates with mock objects how the system should interact.

To provide an example of the sign in tests below, using mock objects it demonstrates the workflow of the real implementation. The first test is verifying we have an interaction with the method in our mocked activity. The second test verifies an interaction with the Firebase function and asserts that the mock call we made returned the value we had expected. The third test verifies an interaction with the Task object that gets returned from Firebase on success, in this case a mockTask object, we also assert the mockTask returns the expected value.


```

@Test
public void SignIn_VerifyInteraction(){
    mockActivity.signIn( email: "name@email.com", password: "Password1");
    verify(mockActivity).signIn( email: "name@email.com", password: "Password1");
}

@Test
public void SignIn_FirebaseListener_ReturnsTask(){
    when(mockAuth.signInWithEmailAndPassword( $: "name@email.com", $s1: "Password1")).thenReturn(mockTask);
    mockAuth.signInWithEmailAndPassword( $: "name@email.com", $s1: "Password1");
    verify(mockAuth).signInWithEmailAndPassword( $: "name@email.com", $s1: "Password1");
    assertThat(mockAuth.signInWithEmailAndPassword( $: "name@email.com", $s1: "Password1"), is(mockTask));
}

@Test
public void SignIn_FirebaseTaskIsSuccessful_ReturnsTrue(){
    when(mockTask.isSuccessful()).thenReturn(true);
    mockTask.isSuccessful();
    verify(mockTask).isSuccessful();
    assertThat(mockTask.isSuccessful(), is( value: true));
}

```

2.6.2 Usability testing

I have been personally testing my application throughout development to ensure I didn't run into any unwanted bugs or glitches. It has helped me find null pointer exceptions triggered by buttons where sufficient information wasn't supplied. It also allowed me to create a fluid working application with easy to use design. I believe anyone could pick up this application and understand each of the functions in it. Towards the end of development of my application I have asked members of my family and friends to interact with the Android application so to provide me with some feedback, I got five users in total.

Those who participated were given a survey to answer some questions regarding feedback and user experience. You can view the survey questions and answers under the evaluation section below. The feedback was quite good, with some constructive criticism and ideas for future development.

2.7 Evaluation

As mentioned briefly in the usability testing section, I have gotten some feedback from all users involved in helping me test my application so not to create a bias review from only my perspective.

I have performed stress tests and system integration tests with my external API's such as Firebase to ensure my application can handle the desired number of requests when processing location data. I have found that when sending data from my smartwatch to Firebase I could send information as quick as I wanted when I had it running locally on my machine as it is a Web Application, but due to hardware limitations again, the smartwatch could not keep up with the constant requests, so I was forced to set an interval of 30 seconds between each request.

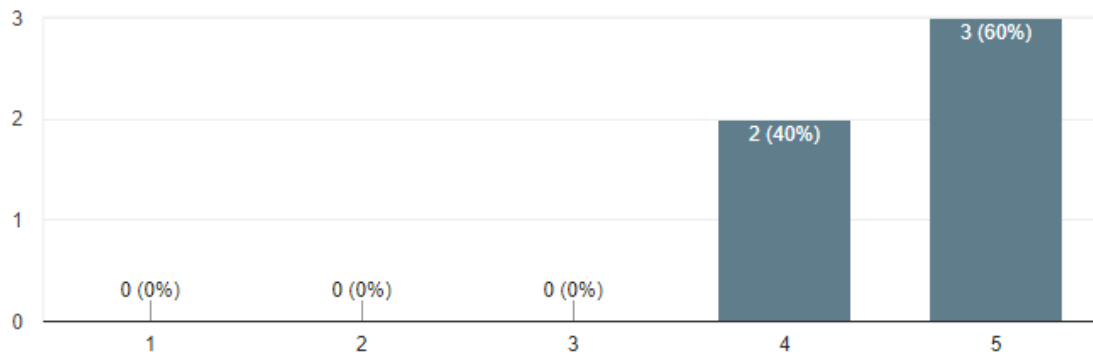
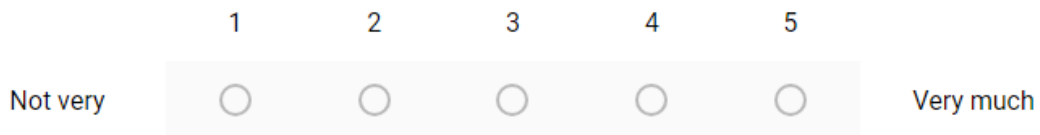
As shown above briefly in the example unit test, I have implemented some system integration tests, that is verifying that I can make a connection to Firebase and perform the relevant tasks.

The core principal and objective of this project was to create an application whereby you could monitor patients through use of a smartwatch device through integration of Google Maps and geofences, and to be alerted when patients entered prohibited zones.

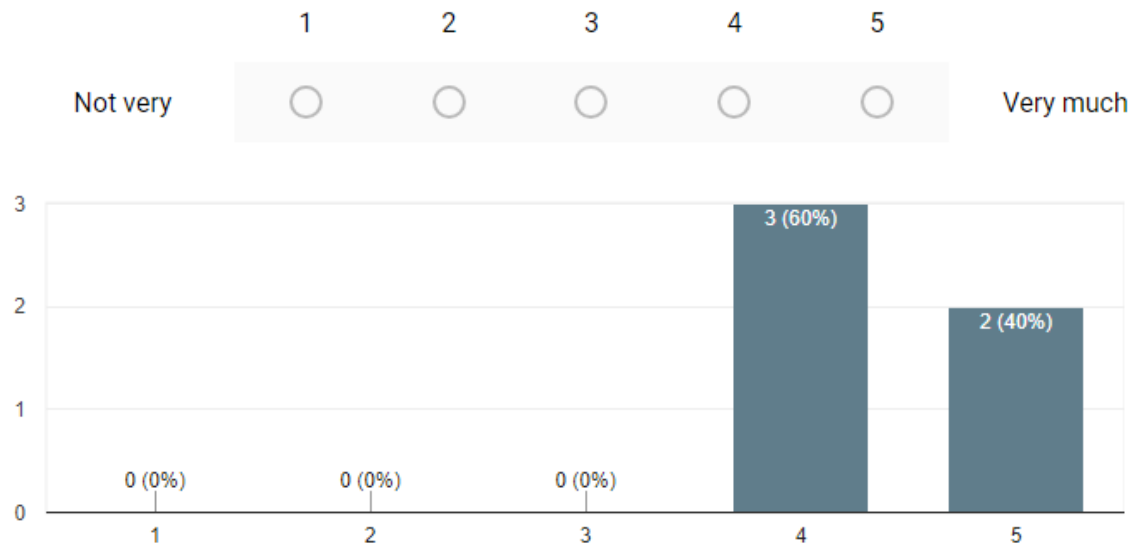
I look back on my initial requirements and prototype application and I compare them to what I have developed now, and I believe I have fully achieved what I set out to do and a little bit more. Of course, there could be so many more improvements given the correct hardware, software, time and resources, there are an endless list of possibility's that could be achieved in an application like this, but from what I set out to accomplish, I have successfully achieved that goal.

Evaluation survey questions and answers.

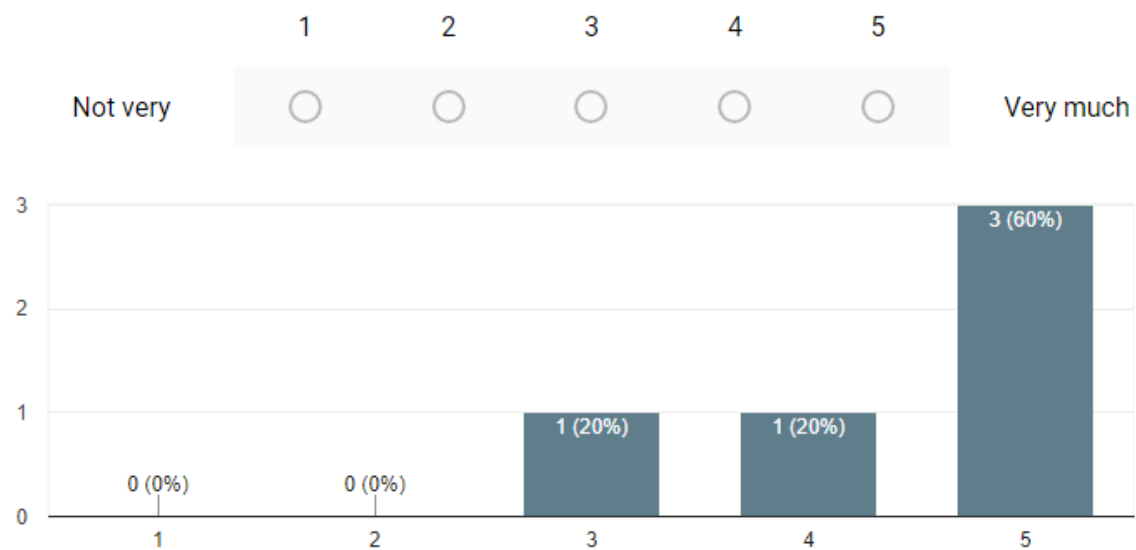
How satisfied were you with how the application performed/functioned? *



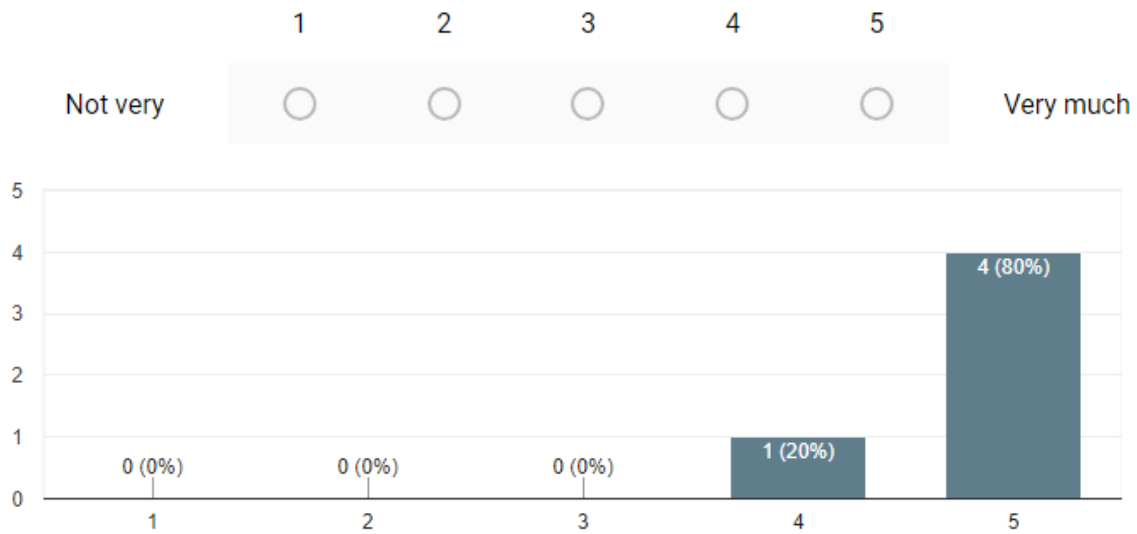
Did you find the application visually pleasing? *



Did you find the application user friendly? i.e. Easy to understand/use *



Did you find the application responsive/speedy? *



What feature of the application did you find most appropriate/rewarding? *

Short answer text

- No need to refresh pages, instant updates
- Notifications we're pretty much instant
- User account, full recovery access
- Accuracy of location
- Profile settings were nice to turn alerts on or off

What feature of the application did you find needs more work/not necessary? *

Short answer text

- Extra info page is a bit plain
- Patient profile doesn't seem necessary right now
- Emergency contact page maybe if multiple people had access to same account
- Making zones too hard
- Don't find two maps necessary, put it on one screen

If you had to introduce a new feature what would it be? *

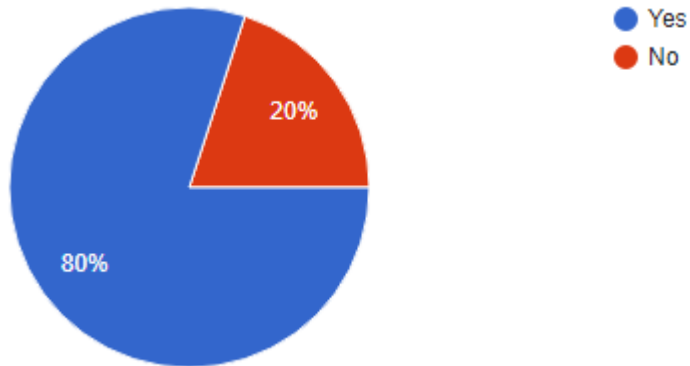
Short answer text

View camera from watch, see what patients see
Send voice messages
Maps direction
Exact timing for zones to be active, so time/day/week/month/year
Ability to link multiple watches to same account, or have alias accounts so a carer would have access to information but only you could update the personal info

Would you be interested in supporting/using an application like this if the opportunity arose? *

Yes

No



Any additional comments regarding the application as a whole.

Short answer text

Enjoyed how easy it was to understand

Colour scheme could be better, could do with logo

Personally think it's a little too invasive on the individual

Needs more information on the extra page

Whole application seems a little bland in terms of view, could be prettier

2.8 Limitations

Throughout the development cycle of this project I've come across some substantial limitations that I did not originally foresee or prepare for. These limitations came about due to the smartwatch I was using, a Samsung Gear S run on Tizen OS version 2.2.1.4.

At the beginning of this project I read up on some documentation about this watch to get an understanding of its features and if I could accomplish the goals for this application with it. Initially I found all the features listed that I wanted to use, but down the line this was not that case.

Unfortunately for me most of the features I needed access to we're only supported on Tizen version 2.3+, this meant I had no access to the heart rate monitor and other sensors. While I did have access to the core functionality of the application which was being able to access the GPS sensor, it was extremely disappointing to find out some of my additional features would be hindered.

I was limited to building a Tizen Web Application which is support lower than version 2.3, and it is considered legacy code. Tizen Studio does not even support versions 2.x anymore and I had to manually downgrade when creating my application.

To overcome these limitations that I now faced I had to resort to mocking the heart rate data using Mockaroo's API. While not ideal or realistic it still demonstrated how my application would function and how the data would be accessed on the Android application.

The documentation and support for Tizen is also quite poor and the developer community is quite small. If I had known what I know now at the start of this process I would not have went down the route of building a Tizen application, I would most likely have chosen Android Wear.

3 Conclusions

I can honestly say that throughout this entire experience, I have learned and progressed many of my skills along the way and learned some extra. Although I was familiar with Android starting this project, I had never developed an application of this scale before. I had never even used a smartwatch before, let alone develop an application to run on it. I was completely new to Firebase and integrating cloud technology into my project, this also brought the challenge of Cloud Functions, which I had to learn Node.js to be able to implement these.

As I specialised in the area of Internet of Things, how devices communicate with each other has always been interesting to me. It was with this project that I got to experience the full picture of M2M communication through use of cloud technology. It has given me much better insight into various technologies I would have never heard of, and it has given me the ability to achieve goals that I did not know how to achieve before with previous projects.

I've learned how to debug my code better and overcome any simple errors I may have faced. I've learned that trial and error isn't a bad thing and I should be open to implementing multiple solutions so that I can achieve the right one. I've learned better styles of code implementation and how my codebase should communicate internally.

While all projects come with ups and down, I've overcome any limitations I faced, and I feel more ups have been achieved here. I'm proud to present the application that I have developed in the time I had, even if there is so much more that could be added, the same can be said for any software project, the capabilities are endless.

I can only look at this as being a success, whether I encountered complete success or didn't manage to meet all my goals, I can now look back and analyse everything I could have done better or differently or keep the same. I can bring forward the knowledge I have now to any future projects I develop whether it be personally or professionally.

4 Further development or research

I believe if I had the opportunity and time to carry out further development and research and the available resources that I could accomplish the features discussed below.

- **Commercialisation**
Over time I believe this application could potentially be commercialised with the inclusion of extra additions and features to the system. Some such features would include a user being able to link multiple smartwatches to their account and have separate information for each watch. This would be particularly useful to larger organisations like nursing homes or hospitals, where they would pay a premium fee to avail of this service.
- **Artificial Intelligence**
The system could also incorporate some AI, in which it would learn the sleep patterns and typical heart rate of the smartwatch user, over time it could potentially sense when something is unusual and not quite right. If it were the case the AI was wrong then the user could give simple feedback to it, whether it was correct or not in order for it to learn.
- **Emergency Services**
It could also incorporate a service where the emergency services are automatically alerted and sent to the location of the smartwatch at the time of the alert, in the case that movement suddenly halts and there's a big drop in heart rate, some sort of indication that the user has encountered a life-threatening situation.

5 References

- Developer.android.com. (2017). *Android Developers*. [online] Available at: <https://developer.android.com/index.html> [Accessed 14 Nov. 2017].
- Developer.tizen.org. (2017). *Tizen Developers | An open source, standards-based software platform for multiple device categories*. [online] Available at: <https://developer.tizen.org/> [Accessed 28 Nov. 2017].
- Doughty, K. and Dunk, B. (2009). Safe walking technologies for people with mild to moderate cognitive impairments. *Journal of Assistive Technologies*, [online] 3(2), pp.54-59. Available at: <http://www.emeraldinsight.com/doi/abs/10.1108/17549450200900016> [Accessed 22 Nov. 2017].
- Firebase. (2017). *Documentation | Firebase*. [online] Available at: <https://firebase.google.com/docs/> [Accessed 14 Nov. 2017].
- Raval, K. (2017). *Geofencing, Location Tracking, and BLE: When and where to use for location-based solution*. [online] Medium. Available at: <https://medium.com/@ketanraval/geofencing-gps-and-beacons-when-and-where-to-use-for-location-based-solution-e80899f3fdea> [Accessed 22 Nov. 2017].

6 Appendix

6.1 Project Proposal

Objectives

The main objective of my project is to create a fully functional mobile tracking and monitoring application through use of a smartwatch. The target audience are those who have children, elderly relations or loved ones which need to be monitored daily for safety reasons.

The application will consist of a Samsung Gear S smartwatch and a Samsung Galaxy S6 smartphone which will be used for the displaying of crucial information. The wearer of the smartwatch will be the individual that will be monitored and tracked, so that myself as a user can keep up to date with their location and certain health vitals.

The smartwatch comes with Tizen OS installed on it, an operating system that I've not come across or used before which could be a challenge, applications written on Tizen are developed in C++ and HTML5.

The mobile application will be developed in Java using Android Studio which I've experience with in past projects.

I will be using AWS to host the backend of the application, this is where the smartwatch will transmit data to, and the mobile pull information from. I may use Python to develop the backend which I've some experience with from work placement.

Background

One of the reasons I decided to choose this project is because I'm specialising in the Internet of Things stream. Technology being able to talk to each other from wherever they are has always been interesting to me so therefore I've gone the direction of wearables and cloud.

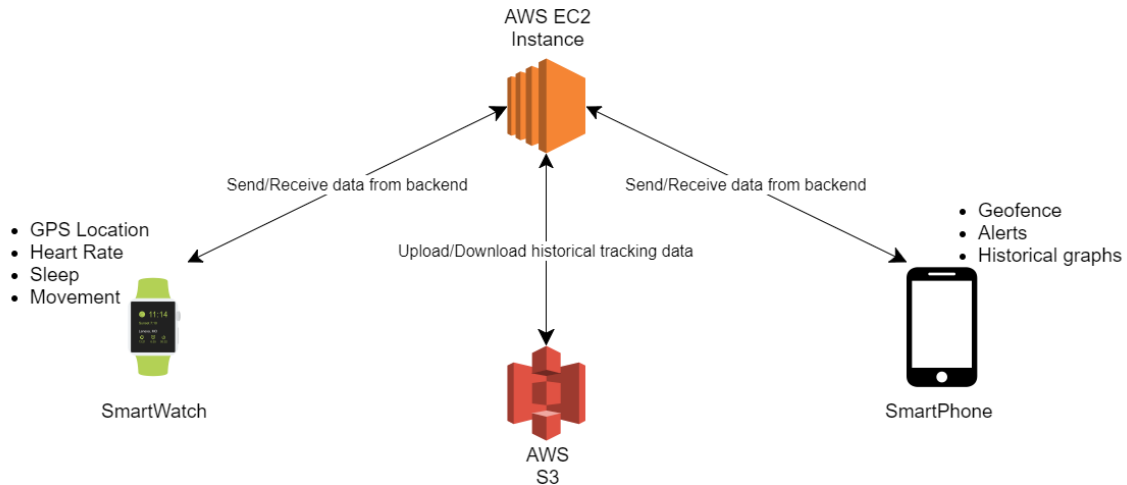
Another reason I had come up with this idea in the first place is from speaking to a friend of mine. His grandad suffers from Alzheimer's disease quite severely, often he doesn't remember his wife or my friends name. He has had cases of disappearing and bursts of rage. When I told my friend about my idea, he thought it could would great if done properly, his family would benefit hugely from something like this.

My application will be used for monitoring human activity, while the smartwatch is being worn it will transmit information to the cloud (AWS) which will then be taken down and displayed on the phone. Information that will be monitored will be heart rate and sleep statistics as well as real-time geolocation. The real-time geolocation feature will be used for geofencing, allowing the user to set a geofence location through the phone which will be communicated to the watch, if the wearer of the watch moves outside of this geofence then the user will receive an alert on the phone. This feature is geared towards elderly people which may have dementia or are prone to wandering away from home which could put them in danger. It may also be geared towards children.

I would also like to include a panic button feature; the Samsung Gear S has one main button on it and depending on how many times it's pressed in succession then it performs an action. An action I would like to add being a panic alert, if the wearer has fallen or is distressed and in need of help. Activation of the panic alert will send an alert

to the users mobile, and in the application, they will be able to see where the individual is located that needs assistance. I may use Twilio for this feature, as it may be quite a severe situation if the panic alert is set off. Twilio allows for automated texts/calls to be put through to a specified number, in this case it would be the user wanting to monitor the wearer of the watch.

Technical Approach



I will start off with some research on some of the technologies I will be using and what they are capable as, such as Tizen OS as I've never used it before. I need to look at the capabilities of the Samsung Gear S as I've never worked with smartwatches before, and to learn how it can communicate with a smartphone.

I'll build a basic UI for the Android application for displaying initial information received from the smartwatch.

I will start creating the backend functionality which will be hosted on AWS, this is what both the smartwatch and android application will connect to and communicate to each other through. I am thinking I will develop this in Python.

Developing the backend will take the longest, but I will break it down into small chunks, first starting with getting the smartwatch to send information up and the Android application reading that information.

I will have to implement a Register/Login system, and to allow users to pair with a smartwatch so I know exactly which smartwatch I'd like to communicate with.

Once I have both devices communicating with each other I will then start working on processing the exact information I would like to send and receive.

I'll move onto developing the most crucial part of my application which is the geofencing feature. It will allow the user to draw on a map free hand or enter specific coordinates. I will need to process this and then be able to tell the smartwatch that if it moves outside of this location that it should send an alert.

I'll need to develop some sort of graphing system for average heart rate and sleep patterns over a period, most likely a week. Once I have starting data, over time I can alert when readings may seem to be too high, too low, or very sporadic.

I will have to develop a notification system for the Android application, so that if an alert comes through from the smartwatch that it reaches the user immediately.

Special resources required

- Smartwatch
Samsung Gear S SM-R750 (Tizen version 2.2.1.4)
Possibly capabilities include sensors and monitors such as gyro, accelerometer, compass, ambient light, barometer, uv light, heart rate, sleep, movement and gps.
Required to transmit location and person data to phone and read geofence area from phone via AWS.
- Android Smartphone
Samsung Galaxy S5 (Android version 6.0.1)
Required to read data from watch and send geofence area to watch via AWS.
- AWS Resources
EC2 required for hosting backend application and S3 required for storing historical data for graphing purposes (weekly/monthly).
Handles communication between both devices.

Technical Details

Android Studio

I will be using Android Studio for my mobile application development part of my project. This is will mainly be written in Java at the time of me writing this. For the most part my Android application will just be a frontend for pulling in information and displaying, it won't contain that much functionality, except for user registration/login and google maps for setting geofences.

Tizen Studio

I will be using Tizen Studio for my smartwatch development; the Samsung Gear S comes installed with Tizen. It will be my first time developing in this Operating System. It is written mainly in HTML5 and C++ along with some JavaScript. I will be developing an application to take readings from the user and send them to the cloud, I'll also be reading the geofence information from the cloud, so I can alert.

Amazon Web Services

I will be using AWS for EC2 instances for hosting the backend and possibly S3 Buckets for storing historical data. I have been able to setup a Student Developer account, which grants me with \$100 worth of free credit to use, I've also signed up to the GitHub Student Developer pack which gives me an extra \$50 credit. For the most part I will be using the free tier resources but just in case I go over that amount or need something extra then I have those credits as backup.

IntelliJ IDEA

I plan to use the IntelliJ IDEA for all my Python development as I've previous experience with it and I've got access to the ultimate version, which supports a lot more and contains a lot of useful plugins to enhance the development experience.

Evaluation

Self/Unit Testing

I will be evaluating my application thoroughly during the development process. As I require the smartwatch to send/receive information to the cloud, so I can process it, I will be acting as both the user and the wearer, so as I'm wearing the watch I will be able to retrieve real time information to work with, in this way I can thoroughly debug and develop my application efficiently.

I will also be writing unit tests along the way for my classes and methods to ensure I don't accidentally break parts of my code during change and updates.

System Integration Testing

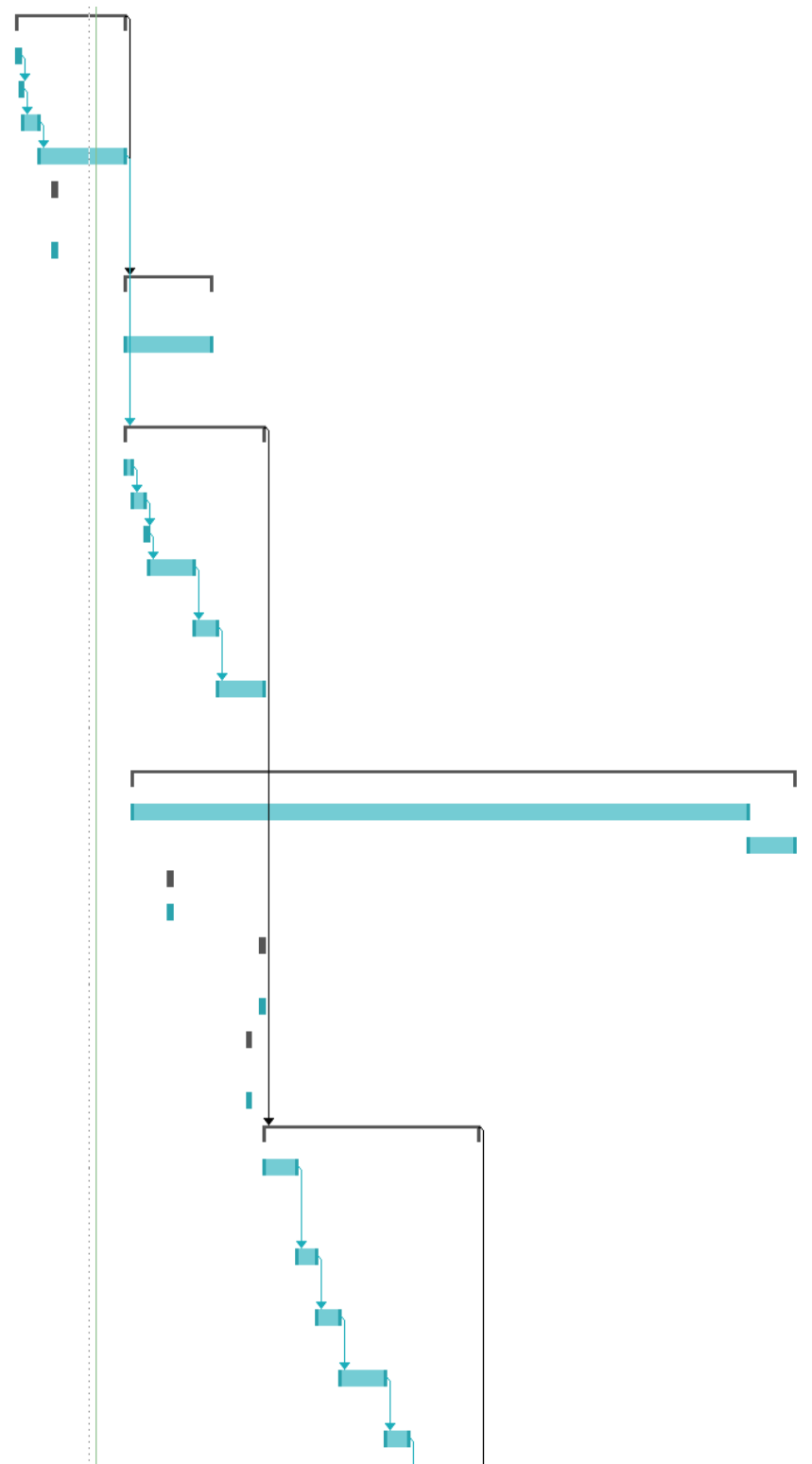
I will be incorporating system integration tests to ensure each component of my application can successfully carry out its specified tasks, and that there are no failures or lost information when transmitting data via the backend on AWS.

End User Testing

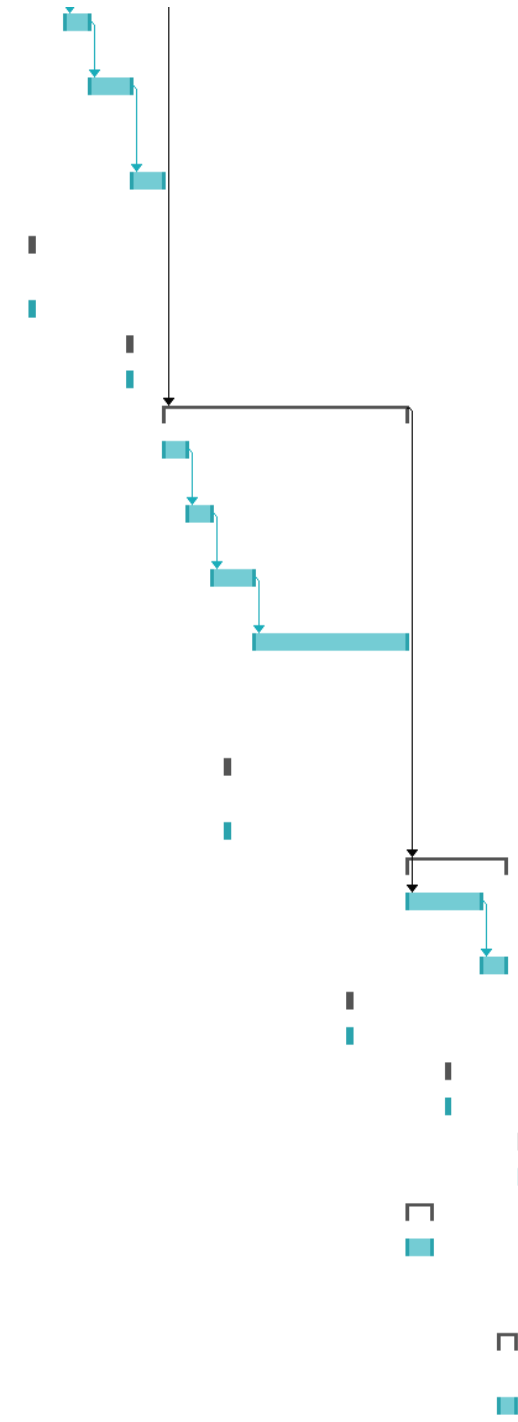
Coming towards the end of development when I have a stable application, I will also get a family member to wear the watch, just to ensure differentiation in data, and to randomise test readings.

6.2 Project Plan

📁	Project Idea & Proposal	33 days	Mon 25/09/17	Fri 27/10/17
🚀	Brainstorm Idea	1 day	Mon 25/09/17	Mon 25/09/17
🚀	Prepare for Pitch	1 day	Tue 26/09/17	Tue 26/09/17
🚀	Research	5 days	Wed 27/09/17	Sun 01/10/17
🚀	Project Proposal	26 days	Mon 02/10/17	Fri 27/10/17
📁	Reflective Journal September	1 day	Fri 06/10/17	Fri 06/10/17
🚀	Upload Journal	1 day	Fri 06/10/17	Fri 06/10/17
📁	Requirements Specification	26 days	Sat 28/10/17	Wed 22/11/17
🚀	Requirement Specification Document Upload	26 days	Sat 28/10/17	Wed 22/11/17
📁	Phase One Development	42 days	Sat 28/10/17	Fri 08/12/17
🚀	Create Basic GUI	2 days	Sat 28/10/17	Sun 29/10/17
🚀	User Login/Registration	4 days	Mon 30/10/17	Thu 02/11/17
🚀	Pair Watch	1 day	Fri 03/11/17	Fri 03/11/17
🚀	Start Backend Development	14 days	Sat 04/11/17	Fri 17/11/17
🚀	Start hosting backend on AWS	7 days	Sat 18/11/17	Fri 24/11/17
🚀	Communication between Watch and Phone via AWS	14 days	Sat 25/11/17	Fri 08/12/17
📁	Testing	201 days	Mon 30/10/17	Fri 18/05/18
🚀	Self/Unit Testing	187 days	Mon 30/10/17	Fri 04/05/18
🚀	Peer Testing	14 days	Sat 05/05/18	Fri 18/05/18
📁	Reflective Journal October	1 day	Fri 10/11/17	Fri 10/11/17
🚀	Upload Journal	1 day	Fri 10/11/17	Fri 10/11/17
📁	Reflective Journal November	1 day	Fri 08/12/17	Fri 08/12/17
🚀	Upload Journal	1 day	Fri 08/12/17	Fri 08/12/17
📁	Mid Point Presentation Prototype	1 day	Mon 04/12/17	Mon 04/12/17
🚀	Application Prototype	1 day	Mon 04/12/17	Mon 04/12/17
📁	Phase Two Development	65 days	Sat 09/12/17	Sun 11/02/18
🚀	Change/Implement Feedback from Mid Point Presentation	10 days	Sat 09/12/17	Mon 18/12/17
🚀	Start Realtime GPS reading	6 days	Tue 19/12/17	Sun 24/12/17
🚀	Set Geofence using coordinates	7 days	Mon 25/12/17	Sun 31/12/17
🚀	Set Geofence by drawing on map	14 days	Mon 01/01/18	Sun 14/01/18
🚀	Add Alerts for leaving geofence	7 days	Mon 15/01/18	Sun 21/01/18



🚀	Add Alerts for leaving geofence	7 days	Mon 15/01/18	Sun 21/01/18
🚀	Add multiple level zones to geofence (green/yellow/red)	12 days	Mon 22/01/18	Fri 02/02/18
🚀	Customise alerts per zone colour	9 days	Sat 03/02/18	Sun 11/02/18
📅	🔪 Reflective Journal December	1 day	Fri 05/01/18	Fri 05/01/18
🚀	Upload Journal	1 day	Fri 05/01/18	Fri 05/01/18
📅	🔪 Reflective Journal January	1 day	Fri 02/02/18	Fri 02/02/18
🚀	Upload Journal	1 day	Fri 02/02/18	Fri 02/02/18
📅	🔪 Phase Three Development	70 days	Mon 12/02/18	Sun 22/04/18
🚀	Add graph's for historical data	7 days	Mon 12/02/18	Sun 18/02/18
🚀	Implement Panic button for watch	7 days	Mon 19/02/18	Sun 25/02/18
🚀	Introduce Twilio for panic alert	12 days	Mon 26/02/18	Fri 09/03/18
🚀	Extra time for finishing development or implementing any extra/new ideas	44 days	Sat 10/03/18	Sun 22/04/18
📅	🔪 Reflective Journal February	1 day	Fri 02/03/18	Fri 02/03/18
🚀	Upload Journal	1 day	Fri 02/03/18	Fri 02/03/18
📅	🔪 Bugs/Glitches	28 days	Mon 23/04/18	Sun 20/05/18
🚀	Fix any bugs/glitches or broken code	21 days	Mon 23/04/18	Sun 13/05/18
🚀	Cleanup code	7 days	Mon 14/05/18	Sun 20/05/18
📅	🔪 Reflective Journal March	1 day	Fri 06/04/18	Fri 06/04/18
🚀	Upload Journal	1 day	Fri 06/04/18	Fri 06/04/18
📅	🔪 Reflective Journal April	1 day	Fri 04/05/18	Fri 04/05/18
🚀	Upload Journal	1 day	Fri 04/05/18	Fri 04/05/18
📅	🔪 Final Project Upload	1 day	Fri 25/05/18	Fri 25/05/18
🚀	Upload project code	1 day	Fri 25/05/18	Fri 25/05/18
📅	🔪 Project Showcase	7 days	Mon 23/04/18	Sun 29/04/18
🚀	Prepare showcase materials/working application	7 days	Mon 23/04/18	Sun 29/04/18
📅	🔪 Final Presentation Preperation	5 days	Sat 19/05/18	Wed 23/05/18
🚀	Prepare slides/pitch	5 days	Sat 19/05/18	Wed 23/05/18



6.3 Monthly Journals

6.3.1 September

Student name: Evan Masterson

Programme: BSHCIOT4

Month: September 2017

My Achievements

September 20th

I started giving my project idea some thought. I have a list of technologies that I'd like to use such as AWS, Android, and a Raspberry Pi. My initial idea is some sort of live score application for GAA, I've done some research and there's nothing like that out there. The current Official GAA application only has results and fixtures of games, there's no real-time aspect.

September 27th

As I'm specialising in IOT, I contacted Dominic Carr about setting up a meeting to discuss some of my project idea's, so I could get a better grasp of what I had in mind. As I would like to implement the use of a Raspberry Pi, I felt that he would be of great help.

September 28th

I met with Dominic today and sat down to discuss my project idea. I told him I'd like to develop a frontend android application, with the backend on the Raspberry Pi, and to host the Raspberry Pi on AWS, so I can connect to it through AWS instead of directly. He gave me some good insight into what I plan to achieve, he helped me flesh out my idea and develop the complexities that I should introduce into it. I didn't think of how I was going to get my information for the live scores, I don't believe the GAA have an API for this, as they don't provide live scores in the first place. Dominic mentioned that it might be a good idea to crawl reliable sports websites for my information, and by comparing them to each other, determine if the score is reliable and correct.

October 2nd

Today we had our project pitch, my project was approved but due to technicalities of the GAA season not starting until February and no live data available, I've decided to change my idea. I've decided to develop an application that targets the elderly. They will wear a smart watch that will track their vitals and location, if you are a carer or family member you will be able to open an

android application which will be linked to the watch. You will be notified of any out of the ordinary circumstances such as heart rate dropping too low or being able to setup a geofence, so you will be alerted if the elder person was wandered too far away from home.

October 5th

I've acquired a Samsung Galaxy S5 and a Samsung Gear S from the college to use for development purposes.

6.3.2 October

Student name: Evan Masterson

Programme: BSHCIOT4

Month: October 2017

My Achievements

October 11th

I met with Dominic again to run through my new idea. He seemed happy about the idea to target the elder and those with disabilities. He especially liked the geofencing aspect of the application, allowing users to set up a geofence for those wearing the watch, and to alert when the wearer has wandered outside of the specified area.

October 12th

Today we received our supervisors, I've got Dominic Carr, which I'm very happy about. I will schedule another meeting with him once I've started making some ground on my project. Currently I'm just working through documentation.

October 14th

As I plan to use Amazon Web Services for my project, I've set up an AWS account, I've also signed up to AWS Educate, which grants me with \$100 worth of credit. I claimed the GitHub student developer pack also which grants me an additional \$50 worth of credit on AWS.

October 18th

Today I had my first official supervisor meeting with Dominic, we went through my project proposal draft and he pointed out some small things I should change. I also just started my Gantt chart for my project plan.

October 19th

I've now finished my project proposal along with a fully fleshed out Gantt chart. I will be uploading this today or over the weekend in case I need to tweak anything else.

October 30th

I haven't made any progress yet as I've been working on other module deliverables which are due shortly, but today I have paired my smartwatch and phone together and setup Tizen Studio to get ready for development.

Next Month

I plan to get my requirement specifications document done along with beginning development, I need to start managing my time better as until now I've been focusing solely on other closer deliverables, I need to spread my workload out better. I would like to have a decent working prototype ready for the midpoint presentation, so I can't be wasting anymore time.

6.3.3 November

Student name: Evan Masterson

Programme: BSHCIOT4

Month: November 2017

My Achievements

November 7th

Today I've just read the requirement specification document fully to understand what is required before I start diving into it. I've also been looking at some past technical report documents to get a better grasp of what's to be expected. I then completed the Introduction to the document.

November 11th

I started working on my GUI design for the requirement specification document using a program called Mockplus. I've now fully completed the GUI section of the document.

November 14th

I've finally begun development, I've created my GitHub repository and have started pushing to it. So far, I've created all my blank activities that I will be using, and I've started my user registration/login functionality using Firebase.

November 15th

I've completed the entire user registration/login and password reset functionality in conjunction with Firebase. It's my first time using Firebase, but I've found it easy and straight forward to get to grips with.

November 16th

I started work on my use case diagram and defining each of my use cases for the requirement specification document.

November 20th

I've now completed all my use cases and the functional requirements section of the document, ready to move onto completing the non-functional requirements and the system architecture. I've also sent a draft of my document to Dominic to review before our meeting on Wednesday.

November 21st

Today I finished the non-functional requirements and the system architecture parts of my document, I also proof read it and tidied up any mistakes or tweaked any information I thought necessary.

November 22nd

I've met with Dominic today to go through my requirements specification document draft, I've just got some minor changes to make regarding my use cases and other slight tweaks in between.

November 24th

I've amended any changes I had to in my document and have uploaded it today. Now it's time I start preparing my mid-point presentation slides and application prototype.

November 27th

I've begun work on the final technical report document which is due on the 30th for the mid-point presentation marks. I won't have it fully finished as it's a working document, but I will have most of it done.

Next Month

Most of my projects will be submitted by the 2nd week in December, over Christmas and New Year I will be focusing a lot on development and the technical document now that my requirement specification document is complete, and I've received some feedback, and I won't have any other deadlines to concentrate on until my exams in January.

6.3.4 January

Student name: Evan Masterson

Programme: BSHCIOT4

Month: January 2018

My Achievements

January 16th

I've been looking into Cloud Functions which are used to trigger certain events based on a certain condition specified. Firebase has recently supported them, so I will be using Cloud Functions to detect changes in specified database fields like location and trigger a function to check whether the user's location is inside or out of the specified geofence zones. These functions will be tailored specifically for sending various notifications/text alerts.

January 19th

I've had some issues regarding my smartwatch in pushing data to the cloud independently. I've got it working when connected to Wi-Fi and Bluetooth via a smartphone. When I disconnect from the smartphone but am still connected to the Wi-Fi, it can access the internet but will not push to Firebase.

January 21st

I've overcome my issue with the smartwatch not working independently, it supports a micro sim, so I have tested with my own sim card containing data and it can now push data to Firebase on its own wherever it is. I will have to get another sim card with data on it, so I can continuously test it without having to swap the sim in and out of my phone.

January 25th

I've moved onto trying to develop my Android application to allow users define geofence zones. Google only natively support geofences with a radius, but not free hand drawn geofences, this is my current issue. I've investigated polygons, but this way the user would have to continuously click on a map and join the dots, which isn't very user friendly.

January 27th

I haven't gotten as much done as I would have like this month, as I didn't have as much time over Christmas as I thought would have had. The exams also took up roughly 2 weeks of my time.

Next Month

I need to figure out how to allow users to draw a free hand outline on a google map and be able to define it as a geofence zone. I think I might be able to do it by tracking the users finger motion on the device but haven't thought about how I will be able to match that to map coordinates. Once I have my geofence issues solved, I can then start developing my Cloud Functions to perform actions when a user's location changes.

I've also thought about possibly moving to AWS IoT to use MQTT to push my location data to, AWS also support a variation of Cloud Functions although I don't know a lot about it as of yet.

6.3.5 February

Student name: Evan Masterson

Programme: BSHCIOT4

Month: February 2018

My Achievements

February 12th

Today I completed my showcase profile which will be going into the final showcase booklet, I've also gotten it signed off by careers and my photo taken.

February 21st

I've run into some issues revolving around allowing a user to create a geofence, I would like to allow them to free hand draw but it's more complicated than I would have liked, I've done some research and I think I would need to use a separate WebView over my map, I would like to try and find an alternative.

February 26th

I met with Dominic today to discuss the progress of my project so far, also to discuss my showcase profile. I brought up the issue with drawing geofences, he gave me some ideas and pointed me in a few different directions.

After some tinkering I've figured out how to draw a geofence using polygons, I've implemented it in a way so that when a user holds down a point on the map it joins them together with polylines, creates an array of the latitude and longitude of each point and then creates the polygon on save. It needs some refining and isn't perfect just yet.

Next Month

I need to look at how I'm going to achieve user alerts and what my best options are, I'm thinking of staying away from AWS.

6.3.6 March

Student name: Evan Masterson

Programme: BSHCIOT4

Month: March 2018

My Achievements

March 31st

Honestly, march has been a terrible month for the software project, I haven't worked on it since February. I've been too busy preparing for and attending job interviews, I've also gotten caught up in my other projects, which was not how I intended it to pan out. At this point I'm feeling as if I may not complete my project to the best of my ability as time is flying by and I'm running short on it.

Next Month

I need to focus on finishing the projects for my ongoing modules earlier than their respective deadlines, so I have more time at the end to complete all of the features of the software project.

6.3.7 April

Student name: Evan Masterson

Programme: BSHCIOT4

Month: April 2018

My Achievements

April 9th

I've submitted my Data Mining and Visualisation final project which lifts some workload of my shoulders, I've still two other projects to finish and upload but they are almost done as well and I'm comfortable with each of their development.

April 19th

I made some small progress on my geofences, storing each of the latitude and longitude points of a zone in an array and pushing them under my zones object in my database, with this I can now load them easily when reading from the database.

April 20th

I've started implementing my cloud functions for triggering alerts which are built using Node.js. So far, I've got the function triggering if the latitude and longitude of the watch changes.

April 22nd

I've added to my cloud functions, reading in the tokenId of the Android device which was been set. With this tokenId I can now target the specific device for push notifications. I have tested these notifications working through the Firebase console.

April 23rd

I've now completed implementing push notifications to the user's device, a notification only gets triggered when the watch's latitude and longitude point enters a defined zone. I achieved this through using a npm package for Node.js called 'point-in-polygon', it uses a ray casting algorithm, which accepts in a latitude and longitude value and a list of latitude and longitude points which make up the polygon zone.

April 27th

All my other projects are now uploading, I can now focus the rest of the time I've got on finishing my project which is due in 16 days.

April 28th

I've implemented the Twilio service into my cloud functions now, if the user has a phone number associated with their account, it will send automated text alerts as well as push notifications.

April 29th

I've made a lot of progress on the Android side of things, refactored some code and moved all the user info from the database into a separate class with an event listener as I had multiple classes needing access to the same information. I've also implemented proper email and password validation and introduced email verification and allow a user to change their email.

I started looking at extra information I need to get from the watch such as heart rate, sleep etc. I've run into complications with the watch I have not supporting updated versions of Tizen, so I cannot access the heart rate monitor and such features. This is a real problem as these are features of my application, I've got to weigh up all the options I have and have a chat with Dominic to get his opinion too.

April 30th

I've implemented allowing the user to opt into text alerts, using SMS verification with Firebase, the user receives a text with a verification code which they must enter.

Next Month

I've only got 2 weeks left to finish everything now, I need to figure out what I'm going to do regarding the issues I ran into with my watch not having access to the heart rate monitor.

6.3.8 May

Student name: Evan Masterson

Programme: BSHCIOT4

Month: May 2018

My Achievements

May 1st

With regards to the issue I had with my watch. I've decided that because I've got the core functions working, that I will mock or dummy heart rate information to demonstrate how my application would work given the proper hardware. It doesn't make sense to create a new application using Android Wear and, on an emulator, as I would be going backwards with what I've already got.

May 2nd

I've decided to use Mockaroo to mock my heart rate values, it provides an API in which you can generate data from the schema you have created. I will call this API from the watch and push the values to Firebase.

May 3rd

I've ran into another issue with the watch, I am unable to send GET requests to the Mockaroo over HTTPS, as the watch fails to acknowledge the SSL Certification. I am going to make the requests from my Cloud Functions, as they are triggered whenever the watch updates location.

May 4th

I've implemented support for different coloured zones, green/yellow/red, they now draw correctly and supported in cloud functions, green zones only trigger push notifications while yellow and red zones will trigger texts also.

May 7th

I've added in a toggle feature to allow the user to opt in/out of receiving notifications, this could be used for a time when they know the patient will be violating zones but knows they are in safe hands.

May 8th

I've change my cloud functions to only trigger when the watch latitude value changes. I've also implemented the mocking of the heart rate which I cannot do on the watch. I've implemented my heart rate graphing on Android using a library called Androidplot.

May 10th

I've implemented some unit tests in Android to demonstrate how the application should work/interact. I've had to mock all the tests due to the majority of my codebase heavily relying on Firebase and I do not want to make real calls in my tests. I've began commenting all my codebase also.

May 11th

Adjusted heart rate mock to ignore previously saved values older than a day for graphing purposes. Done the patient profile page, some more unit tests and commented some more code. I think I'm finished the coding aspect of this project, I just need to finish my document.

May 12th

I just reviewed my code and ran through a final usability test to work out any bugs or null pointers that I encountered and fixed and removed any used code.