# ADVANCED MOBILE APPLICATION DEVELOPMENT

## Report

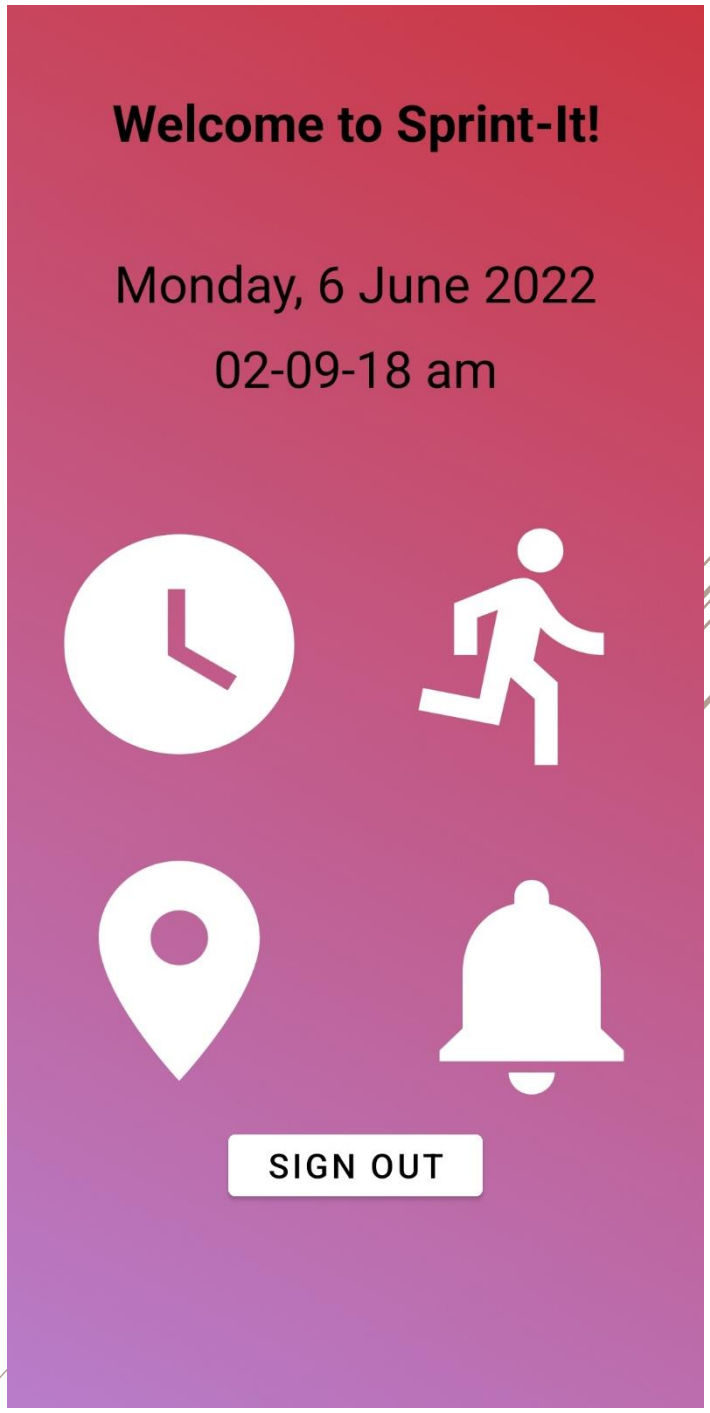


Fig.1 My App

Adam Watson (Student)
18811629

# Contents

# Abstract

With permission from my module tutor Khuong Nguyen, my developed application is different to my proposed application from Assignment 1.

# Introduction

## Project Summary

My developed application is called “Sprint-It”, it is a fitness app directed towards runners and joggers. The app acts as a hub for a user to access different features that can make their fitness session easier and safer. The app utilises various sensors, graphical interfaces and external library usage to deliver an app the user can: create and login to an account using database entry, set timers using a chronometer, count their total steps using sensor types of STEP_COUNTER and STEP_DETECTOR, set notification reminders using a notification manager, track the users current location using location services, in conjunction with google map services. It also uses various graphical features to ensure a smooth experience, such as using “slidr” library to enable swipe functionality and using navigation tools and fragments to avoid simply relying on buttons to navigate to different activities.

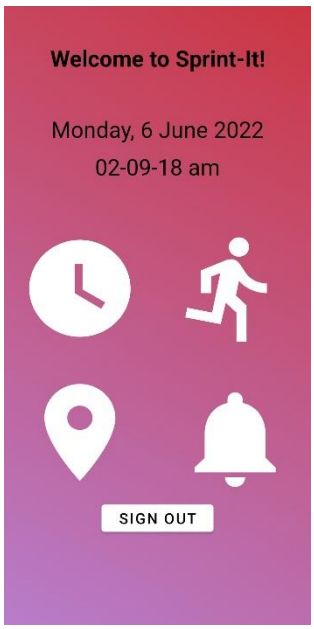


Fig.2 Home Screen


Fig.3 Step Activity

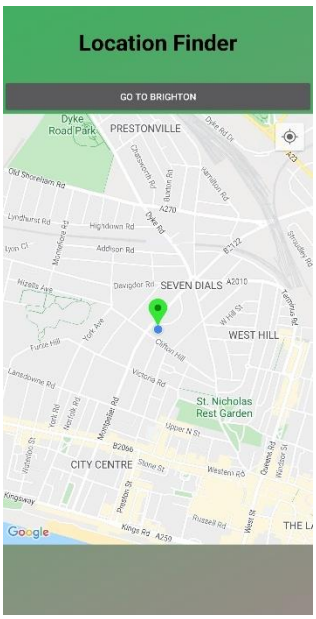


Fig.4 Map Activity

# Research

## Java vs Kotlin

Android Studio gives developers the option to write code with Java or Kotlin. Java was the official language used by Android, as it makes it easier for applications to communicate with the operating system and directly use hardware, however, it was replaced by Kotlin in 2019. Java is object-oriented, which is the programming paradigm I prefer for development; it has 4 main benefits, Encapsulation - which allows you determine the scope of each piece of data, Abstraction by using classes and the ability to generalise object types, Inheritance to eliminate redundant code and Polymorphism that allows multiple objects to be created within one class.

I chose to use Java for the project due to my familiarity with it from previous projects, but Kotlin was also considered, due to it being an improved version of Java. It is faster to compile and scripts are smaller due to it being less verbose, resulting in less bugs. However, Kotlin has a significantly smaller developer community so research resources and assistance would be limited. Due to my planned usage of third-party libraries and implementing features, I believe went above my initial Java knowledge that I would need online help with and that Java was the preferred language for this project.

## Sensors

Android devices usually include a lot of different sensors that developers can make use of. Light levels can be detected to automatically adjust brightness settings, a gyroscope sensor can be used to record gestures as inputs and proximity sensors can be used to automatically disable touch operations, such as when a user has the phone up to their ear taking a call.

***Environmental sensors*** include temperature, humidity, and ambient light levels, I utilised the light sensor within this project. Android measures light in 'lux' units which measures the intensity of light, this is usually detected using the device's front or back camera. When using any sensor, ensuring measures are put in place for the absence of the sensor are essential to not hinder the user's experience. Disabling the feature until the user grants the app permission is an important initial step, as this enables the app to check if the sensor is available and grants access to the phone's available sensors.

***Motion sensors*** allow developers to use the device's gravity, linear acceleration, and rotation for example to manipulate actions and values within their apps. Most android devices include an accelerometer, a gyroscope or other hardware-based sensors due to the reliance software-based sensors has on them being included. The android permission 'CAMERA is required to allow the light value to be detected.

My app utilises a motion sensor that detects device movement through a step counter. This sensor returns the number of steps taken by a user since their last device reboot. The value returns as a float, as it only reset back to 0 on a system reboot. The value is not updated on every step but rather on every second, so this sensor does not give the most accurate reading. To utilise this sensor, the sensor must not be unregistered to allow the sensor to work even when the app is in the background. The android permission 'ACTIVITY_RECOGNITION' is required to access these sensor values. The other approach to counting steps using a motion sensor is a step detector. This sensor triggers an event on every step taken. This sensor detects when a foot hits the ground as it records a high variation in acceleration.

## Libraries

**Slidr** (2015) – Slidr is a third-party library developed by the 'r0adkll' team on GitHub. Its primary functionality is to allow developers to add swipe interactivity within their app and to add attractive animations when a user switches between activities. An alternative considered for this app was **Lottie,** a library that parses Adobe After Effects animations exported as JSON and renders them natively, but due to Adobe After Effects being rather

expensive to design my animations, Slidr's easy implementation was utilised instead. Slidr is implemented into activities using Slidr.attach(this);.

**Google API Client** – This client is a necessary object to access Google APIs provided by the Google Play services library, in my projects case, I needed this client to access Google location services in order to allow the user to view their current location, whilst using a Google Maps activity. These location services were necessary to access the location service's 'FusedLocationAPI', this API combines different input signals and location sources to provide exact location information for my Location Finder activity.

**Firebase SDK** – Firebase is a library that covers services such as authentication, databases, and storage. Firebase is a cloud-based service that would allow me to store user's login information during the sign-in/sign-up activity to grant the user access to the app. It could be used within the app to store all user information, if in-depth profile creation was created.

**SQLite** – SQLite is an SQL database that stores data to a text file on a device, it comes built in within Android as database implementation (android.sqlite.db). This is a good alternative to Firebase if I wanted to store user preferences locally on their device for example, such as remembering if they have light or dark mode selected.

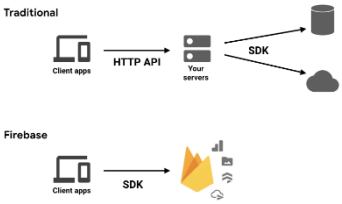


Fig.5 Firebase Model

# Implementation

## Introduction

'Sprint-It' is a fitness app that includes multiple empty activities and a Google Maps activity, it is navigated using a mixture of buttons, swipe gestures using Slidr library and a bottom navigation bar utilising fragments. My app allows the user to set a notification, such as workout reminder to the notification bar, which allows the user to see their exact location and also allows the user to track their step count and the time taken. The background design was created using a gradient list with 5 different animated gradients alternating between one another.

## Design & Features

### Sign-In

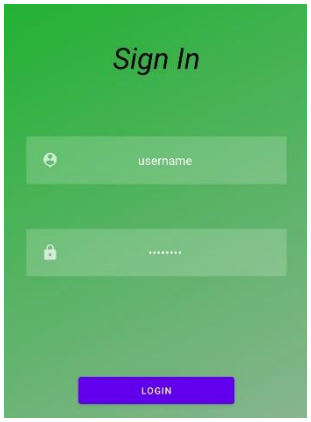


Fig.6 Sign-In Page

My Sign-In page is the first screen the user is taken to after the splash screen and granting storage permissions. The user is greeted by a basic form layout with a username and password. Due to complications with the Firebase library and API levels, the create account activity was removed due to the onComplete method containing the account authorisation. It was not returning task.isSuccessful() as true, meaning the user's new information was not being saved within the Firebase database. Due to this a basic login screen was used where the login parameters were set within Java as:

```
if (username1.getText().toString().equals("username")
&& password1.getText().toString().equals("password"))
```

## Homepage

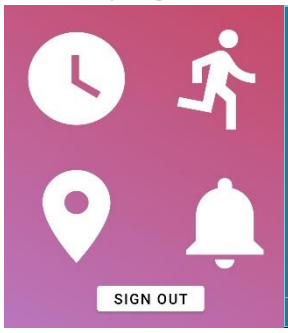


Fig.7a Homepage

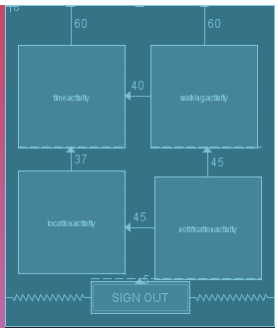


Fig.7b HomePage-FrameView

My homepage is designed using a RelativeLayout as it combines a text section with a cardview-based button grid. The elements are aligned according to their surroundings and their margin values. The button grid is created within a LinearLayout, as the buttons needed to be symmetrical for a pleasing UI and for easy navigation between activities. The user can also press the sign out button to be brought back to the sign-in screen. The buttons are ImageButtons, as this allows me to set a vector asset as the button instead of texts or a coloured button. Due to this being the main screen of the app, each feature must work as intended and I believe I achieved this. Each button directs to different activity using an onClickListener to call the activity function when pressed. This page contains two other basic features, which are to show the user the live date/time and allow them to sign out.

## Time Activity/ Fragments

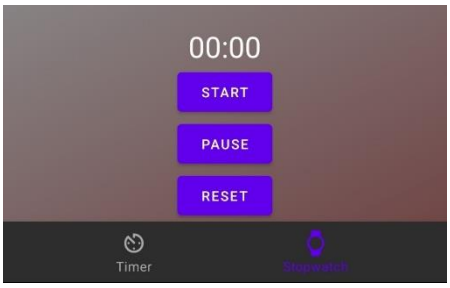


Fig.8 Bottom Navigation Bar

The time activity was my attempt at adding some advanced navigation within the app. This activity simply holds a BottomNavigationView that displays and enables two buttons at the bottom of the screen, which in turn allows the user to switch between two fragments within the same activity. These fragments use a FragmentManager and replace each other when the user chooses which to display. Issues arose however, my chronometer and stopwatch features work as intended in an activity due to the direct link of an XML layout file, but fragments do not recognise findViewByID the same as an activity meaning I could not enable the layout and functions relating to my layout file. Tried solutions included trying to inflate the Fragment's view and call the view it returns but this worked for elements within the inflated view. Creating a onViewCreated() override method where both onCreate() and onCreateView() are initialised, this caused my chronometer and timer to be visible but the functionality remains broken and crashes the app. The intended features would allow the user to time their run and assist in creating an alarm-based timer.

## Notification Activity

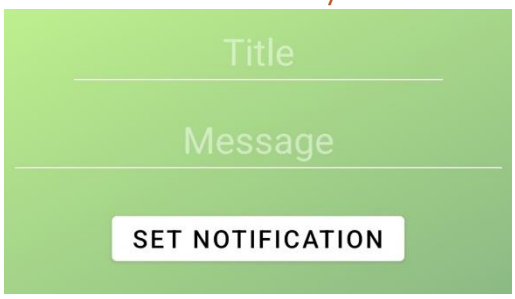


Fig.9 Notification Set

This is a simple activity to allow the user to push a notification of their choosing to their notification bar. A NotificationManager is used here to create a notification channel, IMPORTANCE_HIGH within this manager java classes ensures the notification can use sound, vibration and is pushed to the user's pull-down menu. Using NotificationCompat.Builder allowed me to have access to customisation to the notification such as setting an icon, what category the notification is and sets its priority.

## Step Activity

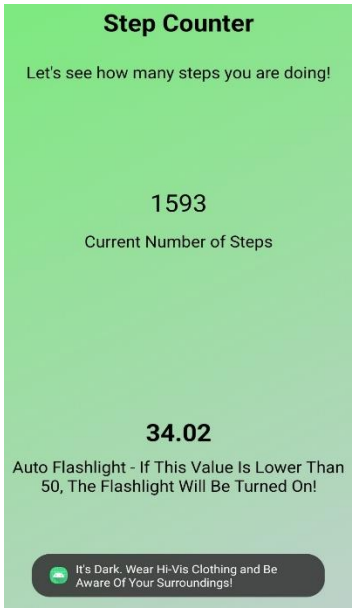


Fig.10 Step Counter Activity

This activity is the most important for the project. The user can see how many steps they have done, this has been created using a step counter and a safety feature has been included that takes a light value using a light sensor and adjusts the user's flashlight according to the value. To use the step counter, the app needs permission ACCESS to ACTIVITY_RECOGNITION to access the user's physical activity. Permission must be granted to use this feature otherwise the text view will not show a number. The app first needs to check if permissions have been granted, if they haven't, request it:

```
requestPermissions(new
String[]{Manifest.permission.ACTIVITY_RECOGNITION}
```

A SensorManager is created to get SENSOR_SERVICE, this then enabled me to call Sensor.TYPE_STEP_COUNTER. When the sensor detects a change (onSensorChanged), the integer is adjusted using the stepCounterTotal textView.

For the flashlight functionality, the app needs to ask the user for permissions to the CAMERA_SERVICE as this contains the light sensor value I need for the conditions in the onSensorChanged() method. When the lightValue is less than 50, the flashlight is turned on and a toast appears, when the lightValue is more than 50, the flashlight is turned off and a toast appears.

## Maps Activity

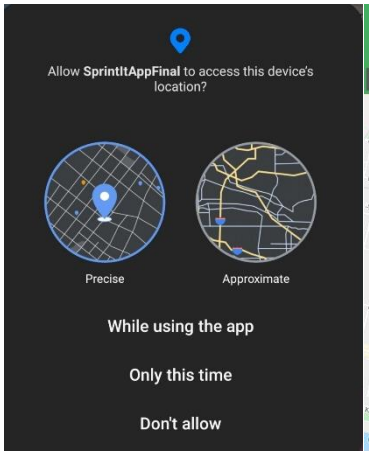


Fig.11a Location Permission

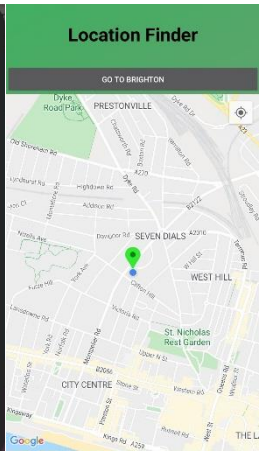


Fig.11b Current Location

This activity implements a Google Maps fragment that allows the user to click the button in the right top corner to find their current location on the map, this is useful if the user is lost. This combines the use of the Google API Client and Google location services. The user must grant permission to both ACCESS_FINE_LOCATION and ACCESS_COARSE_LOCATION for the location function to work. The onConnected() method requests the location of the user by requesting updates using the FusedLocationAPI. The onLocationChanged() method is the visual representation of the user on the map, it uses LatLng to calculate the location and places a marker for the user to see.

# Project Management

## Planning

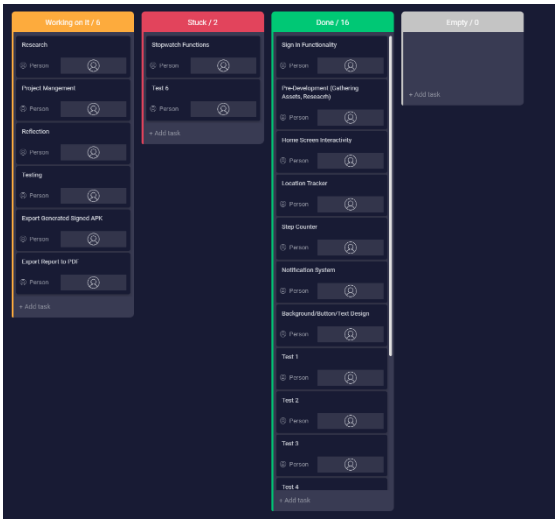


Fig.12 My Kanban Plan Board

The project was developed using the Agile methodology with a Kanban board. The agile approach of Scrum was used by implementing ‘Sprints’ as I was able to split the development features into short periods, these sprints lasted 1 or 2 weeks depending on the complexity of the feature. The ability to visually see what features I had and hadn’t done helped ensure the goals of the project were being met on a weekly/bi-weekly basis.

# Gantt Chart

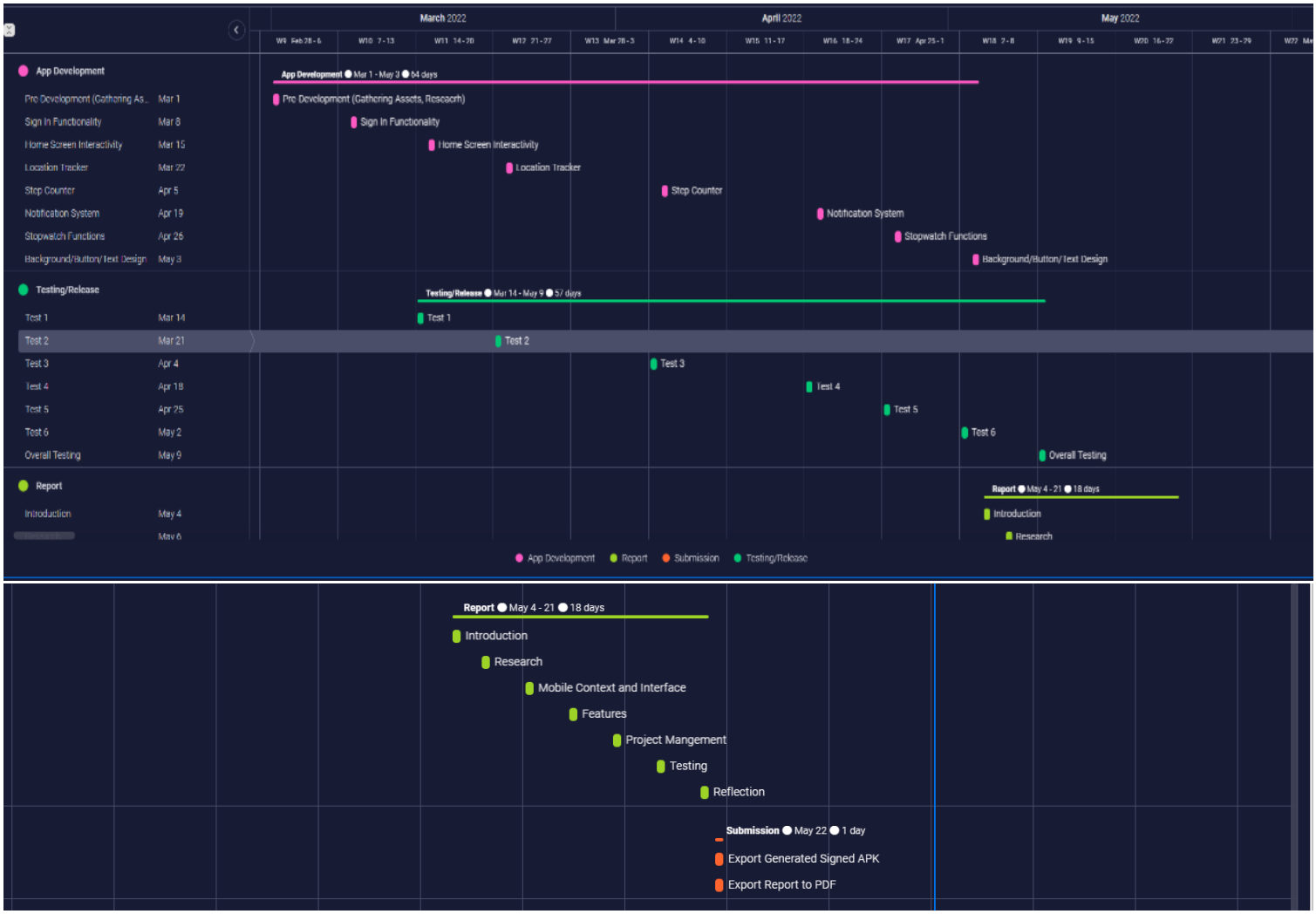





Fig.13 Gannt Chart Plan Near Project End

This Gantt chart provided me with an accurate plan to follow throughout the development process. I did not have to stary from this plan due to the flexibility between the light and heavier tasks. Having the tests before the end of each sprint made the testing process easier as it eliminated most of the app breaking bugs.

## Testing


Fig.14 Testing

For my app, a feature-driven testing approach was used. At the end of each sprint a test was done to ensure all expected features were working as intended before moving onto the next feature sprint. I made use of have a real android device to constantly test features using the debug service Gradle offers, I fixed many crashing bugs through this method as I was able to pinpoint which activity/class was causing the issue. Due to Android Studio being difficult to develop with when it comes to fixing errors, I decided to run quick deployment tests when a slight change was made. Deploying the app to my device allowed me to immediately test the feature to ensure it was ready.

# Reflection

## Strengths

I believe the strongest aspect from this project was the correct usage of sensors due to the technical challenge they provided. This project was more technically demanding than anything I had done before in terms of a mobile context. I learned how to implement sensors by first asking users to grant permission, once granted I was able to take detected values and produce useful features to help a user with their running routine. I am particularly proud of the Map Activity due to having no prior knowledge on using Google APIs. I believe I created a well organised and intuitive user interface using animations, gestures, icons and using layout files in an appropriate way to ensure the best user experience.

## Weaknesses

My weaknesses within this project were related to my underestimation of the difficulty of using Android Studio to develop a complicated application. Due to my lack of knowledge within utilising databases to store information, my login system was implemented at an amateur level. The crashing bug within the timer activity remains the biggest weakness in my project as this causes the app to be unfinished. Some permission functionalities could be fine-tuned to improve the user experience as for example, the user can still access the login screen even if they deny the initial storage permissions.

# Future Development

If I were to spend more time on this project, I would fix the timer activity. Due to the timer activity using a navigation bar that splits the activity into fragments it caused issues with layout. I would instead ensure the navigation bar loads the activities with fragments contained within the view, this would fix the layout issues and still allow the usage of fragment features such as animations/transitions.

The login system would be greatly improved by furthering the LoginScreen and SignUp java classes within my project. The base of a login system where the user can create an account and have their data stored is present, but more insight into compatible API levels and minimum android version is needed to fix possible linking problems.

I would advance the notification system by allowing the user to select a time for the notification to be sent to them. This would require further research into how to keep processes running even if the app has been closed (not in the background).

I would allow the user to use the Map Activity to enter an address and show them directions within the map fragment. This would require further research into using Google's location service and also how a user could enter a valid address to search.

# Evaluation

Overall, I believe my application was successful to some extent. I feel more confident is my usage of sensors, how to create an engaging and well-designed user interface and using external libraries to further my project complexity. It failed in some areas as some features did not work as intended which limited how advanced my application ended up.

# References

StackOverflow (2017). findViewById in Fragment (Available from: https://stackoverflow.com/questions/6495898/findviewbyid-in-fragment)

Code Academy (2022) Why Object-Programming? (Available from: https://www.codecademy.com/article/cpp-object-oriented-programming)

Jessica Thronsby (2019) Master Android's sensors: hardware, software and multidimensional (Available from: https://www.androidauthority.com/master-android-sensors-946024/)

Android Developers (2022) Motion Sensors (Available from: https://developer.android.com/guide/topics/sensors/sensors_motion)

Doug Stevenson (2018) What is Firebase? The complete story, abridged. (Available from: https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0)

TutorialsPoint (2022) Android - SQLite Database (Available from: https://www.tutorialspoint.com/android/android_sqlite_database.htm)

# Project Development References

## *Links to YouTube videos used for development guidance*

Coding in Flow (2018) Notifications Tutorial Part 1 - NOTIFICATION CHANNELS - Android Studio Tutorial. Available at: https://www.youtube.com/watch?v=tTbd1Mfi-Sk&ab_channel=CodinginFlow (Accessed April 2022)

Philipp Lackner (2020) Android Fundamentals. Available at: https://www.youtube.com/watch?v=-vAI7RSPxOA&t=852s&ab_channel=PhilippLackner (Accessed April 2022)

Easy Tuto (2022) BottomNavigationView with Fragments | Android Studio Tutorial | 2022. Available at: https://www.youtube.com/watch?v=OV25x3a55pk&t=373s&ab_channel=EasyTuto (Accessed April 2022)

Stevdza-San (2020) BottomNavigationView with Navigation Component - Android Studio Tutorial. Available at: https://www.youtube.com/watch?v=Chso6xrJ6aU&ab_channel=Stevdza-San (Accessed April 2022)

Coding With Tea (2020) Android Login Screen | Login Android Studio | Android Studio. Available at: https://www.youtube.com/watch?v=ayKMfVt2Sg4&ab_channel=CodingWithTea (Accessed April 2022)

AllCodingTutorials (2020) Login and Register Form using SQLite Database in Android Studio | login registration android studio. Available at: https://www.youtube.com/watch?v=8obgNNlj3Eo&ab_channel=AllCodingTutorials (Accessed April 2022)

Stevdza-San (2020) Mixing Java & Kotlin in a Single Project | Android Studio Tutorialio. Available at: https://www.youtube.com/watch?v=8AAUztC0VJ0&t=140s&ab_channel=Stevdza-San (Accessed April 2022)