

**City, University of London**

**BSc (Hons) Computer Science**  
**IN3007 Individual Project**  
academic year: 2020-2021  
**Project Report**

‘Tractivity’ An Android Application to keep track of the time  
spent on activities, projects, or tasks.

Proposed by:  
Tanvir H Ahamed  
Tanvir.Ahamed@city.ac.uk

Project Supervisor: Alex Ter-Sarkisov

## CONTENTS

### Contents

ABSTRACT.....	5
Chapter 1: Introduction .....	6
1.1 Problem Solved .....	6
1.2 Primary Objective.....	6
1.3 Sub-objectives .....	7
1.4 Project Beneficiaries .....	7
1.5 Work Performed .....	8
1.6 Assumptions.....	8
Chapter 2: Outputs Summary .....	9
2.1 Android Application .....	9
2.2 Android studio guide.....	9
2.3 Design Document.....	9
2.4 UI Screen Design .....	9
2.5 Testing documents.....	9
Chapter 3: Literature Review .....	10
3.1 Android development Tools .....	10
3.2 Programming language .....	10
3.3 Databas .....	10
3.4 Chart.....	11
3.5 Software development methodology.....	11
3.6 Similar Product.....	11
Chapter 4: Method.....	12
4.1 Development Methodology.....	12
4.2 Work Plan.....	12
4.3 Analysis .....	12
4.4 Design.....	12
4.4.1 Use Case Diagram .....	13
4.4.2 Requirement table .....	13
4.4.3 Wireframe Diagram .....	13
4.5 Implementation .....	13
4.5.1 Development Environment.....	13

4.5.2 Programming Language .....	14
4.5.3 Third Party Tools .....	14
4.5.4 Iteration 1.....	14
4.5.2 Iteration 2.....	15
4.5.3 Iteration 3.....	16
4.6 Testing.....	17
Chapter 5: Result.....	18
5.1 Methodology and Work plan .....	18
5.2 Requirement Analysis .....	18
5.3 Design.....	19
5.3.1 Use case Diagram.....	19
5.3.2 Requirement table .....	19
5.3.3 GUI - Wireframe .....	19
5.3.4 App Navigation structure Diagram .....	20
5.4 Implementation .....	20
5.4.1 Development Environment and Programming Language.....	20
5.4.2 XML .....	20
5.4.3 Version control.....	21
5.4.4 Iteration 1.....	21
5.4.5 Iteration 2.....	25
5.4.6 Iteration 3.....	30
5.4.6.7 Deleted activity .....	32
5.5 Evaluation- Testing.....	33
Chapter 6: Conclusion and Discussion .....	34
Primary Objective: .....	34
Sub-objective 1 .....	34
Sub-objective 2 .....	34
Sub-objective 3: .....	34
Sub-objective 4: .....	34
Sub-objective 5: .....	34
Sub-objective 6: .....	34
Sub-objective7: .....	34
Sub-objective 8: .....	35
Sub-objective 9: .....	35
Resources used: .....	35
Knowledge Gained .....	35

Future work.....	35
Conclusion.....	36
Chapter 7: Glossary .....	37
Chapter 8: References.....	38
Appendix A: PDD .....	41
Appendix B: Diagrams and requirements .....	51
Appendix c: Android Studio setup to run the Application. ....	54
Appendix D: GUI of the application .....	56
<b>Appendix E: Implementation Document</b> .....	65
Appendix H: Code count .....	85

## ABSTRACT

Keeping track of productivity is very important part of a successful life. Sometimes we overload ourselves with so many activities that we lose track of the time and lose focus on the tasks that are more important, or sometimes we get lost on procrastinating on unnecessary things. So, the aim of this project was to develop an Android Application to help users to track the time spent on activities, tasks, or ongoing projects. The main goals were to record the time spent on the activities and store it on the database and then present the progresses to the users in chart format.

The following report goes through the methods, with explanation of implementation of each functionality and highlights all the resources which were used to develop the application successfully.

## Chapter 1: Introduction

### 1.1 Problem Solved

Measuring productive time and being able to analyse it can help to improve productiveness and self-discipline. People often struggle to produce quality productive work as procrastination is almost everyone's nature, sometimes we sit on our worktable for the whole day and still produce very little work. The following android application, called Tractivity, that was developed can help the users in this situation. The users could track their daily activities and keep track of the certain projects or works they are working on. The amount of time spent on different activities would be stored and then at the end of the day or week or month the user could visualize the time spent on activities as a pie chart, bar chart or progress bar. Users could use that information and justify if they were spending their time wisely, also make adjustment to their routine.

There were many timer and time management apps available in the android application market (Play Store), but they did not provide easy solution. Some apps did not have charting functionality, and some were too overloaded with confusing irrelevant functionality which could give user hard time to figure it out. My aim was to keep the application simple where user could achieve the functionalities efficiently.

The main components used to develop the application:

Android Studio	It is the Official integrated development environment for Google's android operating System
Kotlin programming language	Even though Java could have been used for android development, but I chose to code my program in Kotlin programming language as it is currently the official language for android, and I wanted to take the chance to learn a new language.
XML	Extensible Markup Language, it was used design the GUI
Google Firebase	For the backend cloud database and login functionality.
MPAndroidChart	Charting library used to create the understandable and meaningful charts.

There were slight changes from the Project Definition Document, author mentioned there he will be using Charts.kt library for implementing charts functionality but in the actual application MPAndroidChart was used instead, the reason behind is MPAAndroidChart is considered the best easy to use and powerful open-source library that does not require any payment or subscription. And many online support and documentations are available for it.

### 1.2 Primary Objective

To develop an android application that will allow users to track amount of time spent on different activities, tasks or projects and help to increase productivity.

### 1.3 Sub-objectives

Objectives	Testable
Build An android Application that can perform the intended functionalities.	User can easily install the application on their device and run it.
User friendly user interface	User must not be bombarded with loads of text; it shall be easy to use and not confusing.
User account/login	Users can create accounts or login to their existing account.
Easy navigation between different functionality	User can navigate to different functionalities through easy-to-understand navigation bar.
A stopwatch that user can start, pause, and stop;	when on an activity user can start the stopwatch and once stopped the application will ask where the time was spent and store it.
Users' data needs to be stored on the cloud database so they can access the data from any device.	Users shall be able to login to their account from any device and access the data. Also be able to record activities from any device.
Add activities, projects, or tasks	User can create and add new activities, project/tasks they are working on or needs to work on. Time spent on those tasks will be recorded and user can visualize the progress.
View charts	User can visualize the activity records in a meaningful chart format. User can analyse it and focus on increasing productivity.
User profile	Users can view their details and change password.

Slightly different from the PDD as author eliminated the notification objective as it was considered optional functionality, author plans to do that after the main objectives are met.

### 1.4 Project Beneficiaries

- **Any individual looking to increase productivity:** Anyone who have any tasks work they want to focus on can use this app to increase productivity or simply can record every task the user does and reflect and adjust their habit.
- **Students:** Students will be highly benefited, and they can keep track of the time they spend on studies. For example, students have many Modules/subjects that they

need to focus on so they can simply list all the modules in the application and every time they spend time on those modules it will be recorder.

- **Employers and Employees:** Employers can ask employees to use the app when they are working on any project tasks so they can monitor their productivity.

### 1.5 Work Performed

The project was split into different parts each focusing on individual elements of the application development, these parts include:

Work	Description
<b>Project plan</b>	The PDD, work plan and risk management
<b>Literature review</b>	The literature that was used to help to develop the application
<b>Application requirements and design</b>	The use cases and requirement of the project
<b>Stopwatch implementation</b>	A way to let user record the time spent on activities
<b>Cloud database implementation</b>	Cloud database platform to allow users to store their application data
<b>View chart implementation</b>	Retrieve stored data from the database and present in charts format
<b>Profile implementation</b>	Way to view, update and reset user information
<b>Project Report</b>	This describes the full project structure, implementations, methods, results, and conclusions with supporting documents
<b>Video demo</b>	The project demonstration video was created to showcase the final application product.

### 1.6 Assumptions

It was assumed that the project can be quite challenging since author had no experience in android development or Kotlin programming language, nor was he experienced with Google Firbase. So, it was assumed that extra effort was needed to learn about these subjects first before diving into the application development. It was also assumed that there would be enough recourses and study material available online that could be easily accessible to aid the process.



## Chapter 2: Outputs Summary

This chapter explains the output of the project including their overview description, output, recipient, and link.

### 2.1 Android Application

<b>Description</b>	<b>A successful android application built through Android Studio IDE that can be run in any android devices.</b>
<b>Output</b>	The product source code is consisting of 1384 Kotlin and 10844 XML codes, out of which 2679 codes (excluding the comments) were written by the author.
<b>Recipient</b>	Project beneficiaries, Reader
<b>Link</b>	Onedrive: <a href="https://1drv.ms/u/s!AjRdDyoNsYGxnnOEzHvr80-Ycei-?e=aa6hUw">https://1drv.ms/u/s!AjRdDyoNsYGxnnOEzHvr80-Ycei-?e=aa6hUw</a>
<b>Appendix</b>	Appendix E, G

### 2.2 Android studio guide

<b>Description</b>	<b>A document providing the setup guide for running the application on android studio</b>
<b>Output</b>	Setup guide
<b>Recipient</b>	Reader
<b>Link</b>	Readme.txt
<b>Appendix</b>	Appendix C

### 2.3 Design Document

<b>Description</b>	<b>Illustrates the prototypes and diagrams of the application done before implementation of the project.</b>
<b>Output</b>	Diagrams and design plans
<b>Recipient</b>	Reader
<b>Appendix</b>	B

### 2.4 UI Screen Design

<b>Description</b>	<b>Contains all the final look of them application GUI that can help the reader understand the work implemented</b>
<b>Output</b>	Images of UI screens
<b>Recipient</b>	Reader
<b>Appendix</b>	D

### 2.5 Testing documents

<b>Description</b>	<b>A document containing a test table that was followed to carry functional testing.</b>
<b>Output</b>	Test table
<b>Recipient</b>	Author, Reader
<b>Appendix</b>	F

## Chapter 3: Literature Review

To develop the intended android application, the knowledge and research into Android SDK, programming language, database, methodology, design, and available tool is required. This section references various literature topics that is used to develop the application. Since it is a productivity application, prior to starting the project, several research about productivity and time management were carried out to understand the problem better.

### 3.1 Android development Tools

There are many tools and approach to developing a mobile application, it mostly depends of requirement and personal preferences. Different approach such as native, cross-Platform, Hybrid, etc can be used, each with different pros and cons (Velvetech 2019). Native development approach is considered the best approach as it is highly reliable, secure, and responsive.

Upon researching different tools (Top 20 Tools for Android Development, 2018), the top pick was Android Studio or IntelliJ IDEA. Having several years of experience in IntelliJ IDEA, it seemed to be the better tools at first, but further researching about android studio it was found that it is currently the best tool for android Application development as it is official integrated environment for android with ease of editing, debugging and testing codes (DEMCHENKO, 2020).

### 3.2 Programming language

There are plenty of options for developing native android apps: java, kotlin, c, c# and Javascript (Sims, 2019). As Android studio was the tool chosen, the language optionality was narrowed to Java or Kotlin. Having previous experience in Java, it seemed using java might be the easier approach but researching further, it was discovered that Kotlin is currently the official language for android app development and it is not very different from java (Gill, 2020). Kotlin is instead an improved version of java for android development, and it uses all the libraries of Java. So, in conclusion Kotlin is chosen as the programming language and author felt it will also broaden his job opportunities having skills in an extra programming language. Several online resources or books will be used to learn the language and implement to build the app. Some of the best platform to learn Kotlin programming language are Udemy, YouTube and Kotlin documentation itself.

### 3.3 Database

When it comes to choosing a database, there are many choices available depending on the requirements (IT Info-Tech, 2019). As the application aims to let users to access their account from different devices, a cloud-based database platform would be the best solution. One of the best platforms available is Google Firebase developed by Google and backed By Google Cloud. The two option that are available is Realtime Database and Firestore. Firestore is the Firebase's newest database for mobile app development and Realtime database is firebase's original database. One of the key differences between this two is data model, Firestore uses document organized in collections structure where Realtime uses JSON Tree structure (Firebase, 2021). Since Google itself recommends using Firestore over Realtime as its data structure is very power full and capable of handling complicated query, author have decided to proceed with Firestore as database and Firebase Auth for backend authentication.

The Firebase website has complete guide and documentation with video explanation on how to get started and understand the data structure and queries, author will be using those recourses during implementation process.

### 3.4 Chart

In order to implement the view chart functionality, author will use open-source library available for android. Some of the libraries that were found during research were Chart.kt, MPAndroidChart, Google Chart, AnyChartsAndroid. MPAndroidChart was recommended by many as it is powerful and easy to use and many online tutorials are available for free.

### 3.5 Software development methodology

To develop a successful project, correct development methodology is crucial. Two of the methodologies that author is familiar with are waterfall and agile. Waterfall is a linear management approach; it requires to planning and designing the whole project prior to building (coding) the product. Waterfall methodology can be lengthy, and it does not handle changes well, where in the Agile methodology the project can be divided into iterations and planning, designing, developing, and testing can be performed to each iteration (Bowes, 2014). Agile approach allows project to be created swiftly. Considering both approaches, Agile methodology was more appropriate for the project since the application needs to be completed in short time and lots of changes are expected during each iteration as author is building the application while learning.

### 3.6 Similar Product

Mobile application market is very broad, currently there are around 2.87 million apps in Google play as stated in Statista website which indicated there exist an application for almost any idea. Upon further research it was found that there are many other applications like author's but many of them are very poorly made, overloaded with too many confusing functionalities or very impractical, which is why author planning to make this application to fill these gaps with useful functionalities that are really needed and that can help user with productivity.

## Chapter 4: Method

This chapter provides a detailed and objective report of the work undertaken. Each subsection provides further detail to the processes introduced and the result of these processes will be further explained in the result section (Chapter 5).

### 4.1 Development Methodology

As mentioned earlier, the development methodology that was used to develop the application was Agile methodology. There were several types of agile methodology that were available such as Kanban, Scrum, Extreme Programming, Crystal, etc (Jigsaw Academy, 2021), since the project was required to be completed within a very short time, Scrum Agile methodology was the best approach. The whole project lifecycle was briefly planned and divided into 3 iterations, then each of the 3 iterations goes through a 4 flexible stages: planning, designing, developing, and testing. It was also easier to make amendment within an iteration without effecting the whole lifecycle of the projects.

### 4.2 Work Plan

Planning is very important initial part of a project in technology industry as stated by TechRepublic. It is needed to identify the desired goals, mitigate risks, avoid missed deadlines and deliver the project successfully with agreed results (TechRepublic, 2021).

Hence before starting the project, a suitable work plan was created by gathering and understanding the project scopes and the lifecycle in which to carry out the iterations. This workplan includes all the steps to achieve the major milestones and functionalities of the Application, it was carefully planned in consideration of non-project commitments such as job work, coursework of other modules, job interviews, etc. The workplan was described on Project Definition Document and presented in a Gantt Chart. **Appendix A**

The work plan was created in very early stage of the project and there had been some changes in requirements and functionalities throughout the project development, it resulted in few changes to the time scale and iteration orders which is discussed in **chapter 5**.

### 4.3 Analysis

The requirements were analysed by gathering the information of functionalities necessary for the application to achieve the proposed objectives. As stated in Chapter 3, Literature Review, different online research, website, forum, and blogs were read to process the System requirements.

Also, research was carried out on existing similar products available. Understanding their functionality, rating, user reviews, gave an idea to what requirements to add and to avoid. It also gave an insight of what the market was providing and what was missing. Author also discussed different requirements with colleagues, friends, and family to identify what the preferred requirements by the users would be and improve them if needed.

### 4.4 Design

The design stage includes various plan and diagrams of a high-level view of the application. This was done with tools such as Visual Paradigm, Invision, and Word. These diagrams are available in **Appendix B**.

#### 4.4.1 Use Case Diagram

A Use Case Diagram contains the primary software requirements for new software programs and how the user will interact with the software. Once all the requirements were gathered and identified, a use case diagram was created mapping the functional requirements that the users will interact with. The diagram was important as it helped the author to visualize the software functionalities during the development without the need of reading lines of words. Since throughout the project development there had been changes in the functionalities, it was very easy to make those changes in the Use Case diagram. The final Use Case diagram is available in **Appendix B1**.

#### 4.4.2 Requirement table

Usually, a Use Case Specification is created following the Use Case Diagram, but author decided to create a requirement table instead. Since agile development was used to build the application and writing use case specification can be lengthy process, author did not want to spend time writing the whole specifications before starting the application development as the requirements could change any moment during development. It was more efficient to only write a table containing all the functional requirements. Also, Author believed that writing the whole software specifications in advance voids the SCRUM Agile methodology, it defeats the purpose of dividing the whole application into smaller iterations consisting planning, designing, developing, and testing. Use Case Specification is more suitable for Water fall Methodology since the whole system design is created and finalized before passing it to the developing team. The Requirement Table can be found in **Appendix B2**.

#### 4.4.3 Wireframe Diagram

A wireframe is a set of diagrams consisting simple lines and shapes that represents the main structure of an application's GUI (Graphical user interface). It was created in the early development process, a draft of GUI for each screen of the application was created with the online tool called Invision. These diagrams were not the final design of the GUI, it was simply the initial sketch to build the actual GUI, it gave the Author ideas to where to implement the proposed features and functionalities. Wireframes are not same as UX design Tools as they do not include the full details, instead they act as a blueprint of design. It is simple which makes it easier to plan, review and update. It was considered far more effective as the author was the sole developer of the application. These diagrams are in **Appendix B3**.

### 4.5 Implementation

This section details the works undertaken to fulfil the implementation process. Since this application is developed for android platform, different methods, tools, dependencies will be stated that helped to complete the implementation.

#### 4.5.1 Development Environment

As Mentioned in the Literature Review, after considering the pros and cons, Android Studio which is the Google's official integrated environment for android was used to develop the application. Android studio comes with all the necessary tools needed to build android applications. Both the front end GUI designing and backend coding was performed through this developing environment. It can handle all the dependencies with ease.

## 4.5.2 Programming Language

### 4.5.2.1 Kotlin

Different programming languages can be used to develop an android application, but the primary ones are Java and Kotlin. Currently Kotlin is considered the official language for Android application. It is a general-purpose programming language similar to Java, and can fully interoperate with java (Heller, 2021). Kotlin programming language is more concise than Java code which allowed the Author to write a smaller number of codes and still achieve all the functionalities. It was used to build the main logic of the application. Author had no prior knowledge of Kotlin but since it is similar to java and author was experienced with java, he wanted to take the challenge and chance to learn Kotlin while developing the project. Two weeks were allocated to learn the Kotlin Programming language.

### 4.5.2.2 XML

XML, Extensible Markup Language, is another crucial language needed to build android applications on Android Studio. It is used to build the Graphical User Interface of the application. All the application layout, buttons, widgets were done utilizing this language. Even though Android Studio provides a tool to build GUI without the need of coding with XML, Author dedicated some time to learn it and use it to hard code the GUI design as it was considered more flexible and open to complex design options.

## 4.5.3 Third Party Tools

Besides the main software platforms and tools, there had been use of other third-party tools to successfully complete the project:

- Microsoft Word: A word processor used to write the project deliverables, diary, plans, etc.
- Microsoft Excel: Spreadsheet software mainly was used to create the Gantt Charts and project tracker.
- GitHub: used for storing all the project files in an online repository and a solution for version control of the application.
- Zoom: video communication service used to communicate with the project consultant and module leaders

## 4.5.4 Iteration 1

The first iteration involved implementing the home screen of the application which is the stopwatch to record the time spent on activities. Even though the initial screen of the application would be the login/sign up page, author decided to start the first iteration with home screen so he could familiarize with the front-end development of android. This iteration mainly consisted of the user interface, screen layouts, functional buttons, animations.

### 4.5.4.1 initial setup

The initial setup of the application was to install the Android Studio and follow the setup wizard which generated the main files such as an activity.java file, AndroidManifest.XML, empty layout file and build.gradle files.

### 4.5.1.2 implementing the stopwatch

This was the initial screen after the user logs in to their account. Here the user would click the start button once they had started their activity, and the app would start counting every second like a stopwatch. The user can pause the stopwatch when they are not working on their activity and can resume. Once they finishes their activity, they can click the stop button and it will

prompt the users to select on what activity the time is spent on (which will later be stored in database). User can also just reset the stopwatch without storing anything on the database.

The stopwatch functionality was implemented by built in Chronometer class (Android Developers,2021). It is a subclass of TextView and it displays the count in textView, the built in methods were applied to start and stop the stopwatch. It displays the timer values in the form of MM: SS or H:MM: SS.

#### *4.5.1.3 Implementing the rotating Clock Animation*

When the user starts the activity a rotating clock animation is presented. It was implemented by android View Animation system (tweened animation handled by android.view.animation package), it can perform of series of simple transformations on the content of view object such as rotating, moving, resizing, shrinking etc (Android Developers, 2021).

#### *4.5.1.4 Implementing the Notification*

When the user starts an activity, the app sends a notification, this was done with the help of NotificationCompat APIs from the android support library (Android Developers,2021).

#### *4.5.1.5 Implementing save activity Dialog*

Users can end activity and save the progress on the database. When the users end the activity, a pop up (dialog) window will be displayed asking to enter activity name or select name of existing activities. The dialog was implemented by following various online tutorial and android developers' documentations. As stated on the Android Developers documentation, a dialog is a small window that prompts the user to make a decision or enter additional information. A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed (Android Developers, 2021). The subclass AlertDialog of the class Dialog was used with a custom layout to implement this custom dialog. At this stage, the dialog GUI was just implemented but the storing data on the database functionality was implemented on later iteration.

### *4.5.2 Iteration 2*

Iteration 2 involves mostly designing and implementing the back-end functionalities such as user account and database. In this phase author focused on learning about the cloud database and how Firebase can be used to manage the backend service. This phase includes the implementation of signing up or signing in the users, storing user and activity data on the database

#### *4.5.2.1 Firebase*

As mentioned in the chapter 3 literature review, Firebase was the optimal platform for this application for backend management and cloud database. Firebase is a Backend-as-a-Service (Baas) which provides all the necessary tools develop quality apps(Educative,2021). The documentation and video tutorials available on Google Firebase website were used to support the development.

#### *4.5.2.2 implementing User login and Authentication*

Once the application is installed, users will be given the option to login to their account and create one. Since the data would be stored online, a user account and authentication functionality were very important. And for security purpose the CIA traid security model was needed to be integrated in the application. CIA stands for Confidentiality, Integrity and

Availability, it is the core underpinning of information security needed to be implemented in any secure system and software. The firebase Authentication was used to provide this security, each user would have their own login and password which would be used to provide access to their data stored on the database.

'Get Started with Firebase Authentication on Android' documentation available on Firebase website was read and used to integrate Authentication, it provided all the information needed to declare the Gradle dependencies, sign up new users, sign in existing users and access the user information and log out users (Firebase Authentication, 2021).

#### *4.5.2.3 Implementing Cloud data storage – Firestore Database*

Google firebase provides two database service, Realtime Database and Firestore. FireStore is the Firebase' Newest database, it was built on success of Realtime Database with new and more initiative data model (Firebase, 2021). It was used to store the user's data and activity progress. As stated in the PDD authors initial plan was to first store the data on a SQL database which would be in the device for the offline support then transfer it to some kind of online platform but later it was found that Firestore supports offline storage which cancelled the need of SQL database. The data in firestore is stored in documents in a collections structure which was quite different from the databases that author was experienced with, so some time was spent understanding the concept. Several online tutorials and articles were used to learn the Firestore fundamentals such as storing data, querying data, updating data.

#### *4.5.2.4 App Navigation*

A navigation drawer was implemented to provide easy navigation between the different app screens, such as "Tractivity", "Activities", 'Charts", "Profile" and "Log out". This helped with the objective of providing an easy to understand User Interface. The documentation " Update UI components with NavigationUI" available on Goggle developers was followed along with some online article and YouTube video called " SLIDABLE MENU WITH NAVIGATION DRAWER - Android Fundamentals" to develop the navigation menu bar.

### *4.5.3 Iteration 3*

Iteration 3 mainly involved implementing user profile, retrieving and viewing activity data and presenting meaningful charts to the users about the time spent on different activities.

#### *4.5.3.1 Implementing user profile*

The profile screen displays the user information retrieved from the database and allows the user to update information or change password. Profile screen contains user information such as profile name, email, productivity score (an optional functionality), mobile number and address. User could also upload profile image. A custom circular image view found of github called CircleImageView was used for the user profile photo.

#### *4.5.3.2 Implementing activity lists*

All the user's activity, tasks or project would be displayed on the activity list screen. It was implemented by querying the activity data from the Firestore and recycle view was used to display them in dynamic list. Googles documentations about recycle view, online tutorial on Codepath, YouTube and Firestore querying tutorial were used to achieve this objective. Upon clicking an activity list user would be taken activity details screen where the activity details such as started date, due date, activity name, and history of work would be displayed. User could also delete the activity completed.



#### 4.5.3.3 Implementing View Charts

This screen would display the time spent on different activities in a pie chart and bar chart to help the users to visualize the time spent of different activities. Users could later reflect on this visual representation of data. To achieve this objective, the opensource library called MPAndroidChart was used. It is a free Android chart view/ graph view library that can be used to implement bar, pie, radar, bubble, candlestick charts. It also supports scaling, dragging and animation.

#### 4.6 Testing

Testing was crucial stage of the application development to ensure that the objectives and proposed features are achieved correctly without any bugs in the application. As Scrum agile methodology was used, testing was performed at the end every iteration and a final testing was done after the full development of the application.

## Chapter 5: Result

This result chapter present all the result produced during the project development, which includes the outputs from each stage of the project's lifecycle; analysis, design, implementation, testing and evaluation. It will clearly explain all the methods that were taken along with their results.

Drive link: <https://1drv.ms/u/s!AjRdDyoNsYGxnnOEzHvr80-Ycei-?e=aa6hUw>

### 5.1 Methodology and Work plan

As discussed previously, the SCRUM Agile methodology was used to manage the project development. The project lifecycle was broken into 3 iterations, each of them containing planning, designing, developing, and testing. The project was developed successfully in 5 phases:

Phases	Work done
<b>Phase 1</b>	Preparing and designing the Application. Two weeks was used to learn the Kotlin programming language and android SDK
<b>Phase 2</b>	It involved implementing the 1 <sup>st</sup> iteration of the application which was the stopwatch. It was planned, designed, implemented, and tested once all the functionalities were achieved
<b>Phase 3</b>	It involved learning about Google Firebase to implement the backend functionalities. The knowledge was the used to implement the iteration 2 (Firestore database and user authentication), it was tested once done.
<b>Phase 4</b>	This this phase the iteration 3 was implemented which involved accessing and retrieving the users' data from the database and implement the activity list and chart functionality.
<b>Phase 5</b>	This phase involved finishing up or adding any remaining objective of the application and carrying out a finial testing. Also, most of the project report was written at this phase.

Note there was a slight change to the work plan from the one suggested in the PDD. The iteration 2 and 3 ( phase 3 and 4) were swapped because author wanted to implement the database on the device locally then transfer it to online with Firestore to achieve offline storage functionality, but later it was found that Firestore itself provides offline storage which defeated the purpose of having a local SQL database on the device.

### 5.2 Requirement Analysis

Different website and articles were read to gather the system requirement and user requirement. Literature review section details the research carried out to gather the necessary tools for the development of the application.

As mentioned in section 4.3 the user or application requirements were gathered by carrying out research on different existing applications and identifying the needs of deferent users. The top activity tracker application listed on the internet were looked into, then author downloaded them from Google play to understand their functionality and explore was what missing. Also, users' reviews were read to identify what functionalities were appreciated and hated, it gave an insight what the market was offering and what was missing. Negative reviews and missing functionalities helped the author to gather the key requirements. Names of the existing applications that were researched were: 'Focus To-Do', 'Timesheet', 'aTimeLogger' 'Rabit' 'Tasks'. Once all the requirements were gathered, they were discussed with the potential users

to identify what functionalities they prefer, it helped the author to distinguish the functional and non-functional requirements and design the application in that way.

### 5.3 Design

Four main software designs were carried out by the author; Use case Diagram, Requirement table, Wireframe and App Navigation Structure. Author focused on creating only the important designs to dedicate more time on development phase instead of overloading with unnecessary design material. This section explains each design materials and result in depth.

#### 5.3.1 Use case Diagram

The author created the use case diagrams, with the requirement gathered, to design the user's interaction with the applications. It shows a high-level overview of the relationships between different use cases, actors, and the system. It helped the author to visualize the functionalities that were needed to implement in the application, it was also easier to edit and update throughout the project lifecycle. Visual Paradigm was used to create the Use case diagrams since the licence was provided by the university for free.

Throughout the project lifecycle 6 different versions of the use case diagrams were created. In total there are 30 use cases and 2 actors (user as primary actor and Firebase as secondary) in the final version of the use case diagram which is available in **appendix B1**.

#### 5.3.2 Requirement table

As discussed in the **section 4.4.2** that the requirement table was created instead of use case specification, the reason being that the author thinks writing the whole specification of a project that can be changed and updated throughout the lifecycle is inefficient. Since the author is the sole designer and developer of the project, use case specification was not as necessity. Requirement table contains all the System (Application) functionalities in written format. There were 32 requirements in the table which can be found in the **appendix B2**.

#### 5.3.3 GUI - Wireframe

To design the User Interface of the application, a draft of the GUI design was created in the early stage of the project lifecycle with the online tool called Invision. Wireframe diagrams are made with simple lines and shape presenting the struct of the GUI, an example of Sign-up screen:

Although it did not represent the final design of the GUI, it acted as a blueprint of the design which author used during the implementation phase to create the actual GUI with XML. Website called 'Behance' was used to research different GUI designs of mobile applications. **APPENDIX** contains the full Wireframe diagrams.

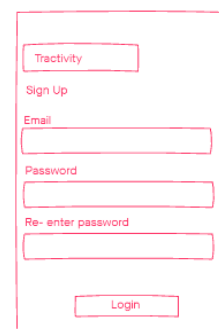


Figure 1: Wireframe Diagram

### 5.3.4 App Navigation structure Diagram

A navigation structure was designed to ensure that the application provides an easy-to-use navigation between the screen:

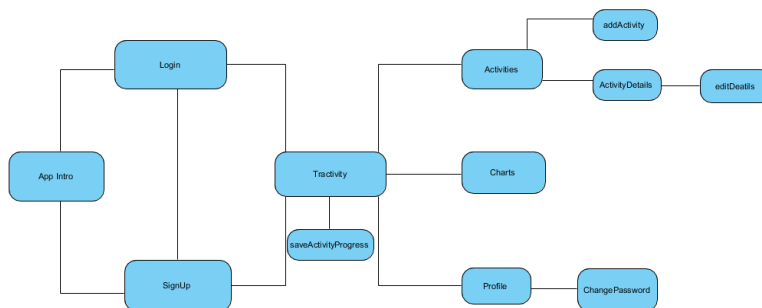


Figure 2 Navigation Diagram

## 5.4 Implementation

The following section details how the project was setup and developed with the result produces at each iterations or phases.

**Note** each screen/page of an android application is referred as activity, please be careful not to confuse it with 'recording activity' that author implies.

### 5.4.1 Development Environment and Programming Language

As mentioned previously in the literature review and method section, the Android studio was the chosen IDE for this project. The initial setup was done using the setup wizard that asks for the project type, name, storage location, minimum supported SDK version and a templet. Once completed, a base project was created with the standard Kotlin activity class and the activity layout xml file. Kotlin programming language was used to implement the application functionalities and XML language was used for User Interface. The Gradle file generated in the IDE installs all the dependencies necessary. An Android emulator was installed to run and test the Application.

### 5.4.2 XML

In android, XML (Extensible Markup Language) is used to design the application layouts. The Android User Interface is defined using the hierarchy of View and View Group objects. A View usually draws something the user can see and interact with, and View Group is the main container that organizes child views and defines the layout structure. These child views are the other widgets which are used to make the different parts of UI. One View Group can have another View Group as a child element as shown in the figure given below:

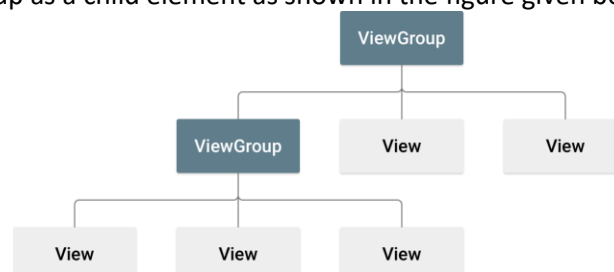


Figure 3 ViewGroup Hierarchy (developers,2021)

Several XML files were used to create the application interface. All the UI codes were hard coded by the author:

- Layout xml files: these files contained the actual Graphical User Interface of the application. It is located in the res/layout folder.
- Manifest xml files: all the application components such as theme, services, receivers and permissions are defined in the manifest file. It is located in the manifest folder called AndroidManifest.xml
- Style xml files: this file was used to define the different the style and look of the application. The custom theme of the application was defined in this file.
- Colour xml files: this file was used to define the main GUI colours.
- Drawable xml files: this file was used to provide various graphics to the elements or views of the application. For example, if a button was needed to be customised with different shapes and background colours this file was used to define those properties. All the images and icons are first pasted in the drawable file and referenced from the layout files.
- Anim xml files: these files contain the animation properties of the application.
- Menu xml files: the navigation structure was defined in this file.

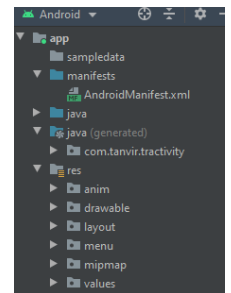


Figure 4: XML File Folders

#### 5.4.3 Version control

There are many version control system providers, but author was more familiar with github since it was used for other projects too. Hence, GitHub was used as version control system, once a major functionality was completed successfully implemented it was committed and pushed to GitHub repository. It acted as a backup in case of any technical issue or if any files were corrupted author could simply revert to previous version.

#### 5.4.4 Iteration 1

Iteration 1 contained implementing the stopwatch functionality that users can start when they work on an activity, task or project. This is the home screen that the user would interact after login in or signing up. In this section author will discuss how this objective was implemented and what result was produced. As stated in **section 4.5** this iteration being the first iteration it just focuses the front-end development of the application.

#### 5.4.4.1 Implementing the Stopwatch

The User interface of this screen was coded in the activity\_stopwatch.xml file available in the layout folder. This screen contained a rotating stopwatch with buttons to start, pause, stop and reset the stopwatch.

This stopwatch was the way to record the time spent on an activity or task. It was implemented by built in Chronometer class, which is a subclass of TextView Class and it displays the count time in textView (A user interface element that displays text to the user), it has built is methods to start and stop the stopwatch. It displays the timer values in the form of MM: SS or H:MM: SS.

The kotlin codes of the stopwatch functionality was implemented in the TractivityMain class in the file called TractivityMain.kt. Once the start button was clicked a onClickListener was triggered which builds a notification panel and calls the startAndPauseActivity() function, this function starts or pauses the stopwatch according to its state.



Figure 5: Tractivity-Home Screen

#### 5.4.4.2 implementing Start and Pause Button

If the stopwatch was in stopped (or never started) state the function 'startStopwatch' would be called which would send a notification to show that Tractivity app is running :

```
NotificationManagerCompat.from(this).notify(NOTIFICATION_ID,buildNotification)
```

Then the rotating clock arrow animation is loaded and started:

```
iArrow.startAnimation(AnimationUtils.loadAnimation(this,R.anim.rotating_arrow))
```

The chronometer base is set so it starts counting from current time displaying time passed is seconds. The chronometer is started by calling its built-in start method, and stopwatch status is set to running. When the stopwatch is in running status, the start button text is changed to "pause" which user can press to pause the stopwatch. Code snippet:

```
var pauseTime : Long = 0
var running : Boolean = false
private fun startStopWatch(){
    c_chronometer.base = SystemClock.elapsedRealtime() + pauseTime
    c_chronometer.start()
    running = true
    btn_start.text = "Pause"
}
```

If the pause button (start button changed to pause) is pressed the stopwatch is paused, the animation is stopped. But the chronometer never stops after it has started once which means stop method does not actually stop the stopwatch, it just stops displaying in the text view and the chronometer still runs in the background. So, in order to implement pause (and resume) functionality of the stopwatch, a variable called 'pauseTime' is used to store the time stopwatch was paused for. The button text is changed to "resume". Code snippet:

```
private fun pauseStopWatch(){
    iArrow.clearAnimation()
    pauseTime = c_chronometer.base - SystemClock.elapsedRealtime()
    c_chronometer.stop()
    running = false
}
```

```

btn_start.text= "Resume"
}

```

To resume the stopwatch, resume button (pause button changed to resume) was needed to be pressed which would call the 'startStopWatch' function which re sets the base value of the chronometer to re-count from the current time and the paused time was added to the base.

#### 5.4.4.3 Implementing - The reset button and Confirmation Dialog

If the users wish to cancel the time being recorded of an activity and does not want to save it then they can simply reset the stopwatch by clicking the reset button. It acts similarly as the start/pause button. First, the click event on this button would only work if the stopwatch was started. Then it would pause the stopwatch and call a confirmation dialog that would warn the consequences of resetting the stopwatch.

The confirmation dialog was an Alert Dialog created with AlertDialog class which is a subclass of Dialog Class. It displays a tittle, a massage and dissision buttons.

The cancel button would simply return to the stopwatch in paused state. If the reset is clicked the 'resetStopWatch' function will be called that resets the whole stopwatch and the buttons names. The dialog section of Android Developers documentation was followed to implement this alert dialog (Dialogs | Android Developers, 2021), code Snippet is available in the in the **appendix E5, E6**.

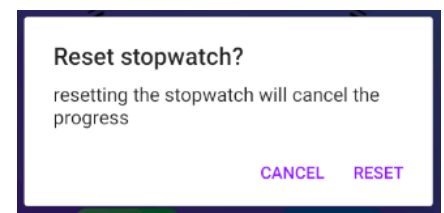


Figure 6: Reset Dialog

#### 5.4.4.4 Implementing- Rotating clock animation

The rotation clock is an animation implemented to visualize when the stopwatch is running. A clock arrow png was taken from online free resources. First it was implemented as image view in the layout xml file. The Tweened animation system was used from animation package to implement the rotation of the arrow. The animation acts according to the status of the stopwatch. The animation properties were implemented in res/anim/rotating\_arrow.xml file, code is available in the **appendix E9**, each rotation represent a second passed. Then the animation is loaded by AnimationUtils class and started with 'startAnnimation' method:

```

iArrow.startAnimation(AnimationUtils.loadAnimation(this,R.anim.rotating_arrow)
)

```

And stopped by 'clearAnimation' method.

#### 5.4.4.5 Implementing the Notification

This notification is sent every time the Tractivity app starts recording time. The android Developers documentation and YouTube tutorials were followed to implement the notification functionality (Android Developers,2021). Implementing notification was straight forward before, but now after Android 8.0 there are few steps required to implement it. First a notification channel must be created by setting its importance and characteristics. The notification must be registered with the system by passing an instance of NotificationChannel to createNotificationChannel(), code snippet in pasted in the **appendix E10** .

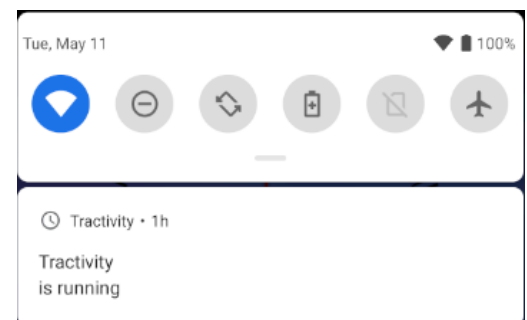


Figure 7: Notification

Then the content and properties of the notification panel was set with NotificationCompat.Builder class which required the notification CHANNEL\_ID as constructor, **Appendix E11.**

Once the notification is taped outside the app, the user would be taken to the app and to do so content intent is defined with a PendingIntent object and it is passed to NotificationCompatBuilder class with 'setContentIntent' method, the following code snippet shows how:

```
val notificationIntent = Intent(this, TractivityMain::class.java)
notificationIntent.setAction(Intent.ACTION_MAIN)
val pendingIntent : PendingIntent =
PendingIntent.getActivity(this,0,notificationIntent,PendingIntent.FLAG_UPDATE_
CURRENT)
```

#### 5.4.4.6 Implementing- Stop Button and Save Activity Dialog

When the users finish an activity, they must be able to save their progress in the database.

Once the users end an activity, they could click the stop the button of the stopwatch which will display a Custom Alert Dialog to enter the activity name or choose from the existing ones retrieved from the database. The activity name and the amount spent will later be stored in the database. Since this iteration author only focused on the front-end design, the implementation of actual data storage in the database was implemented on later iteration.

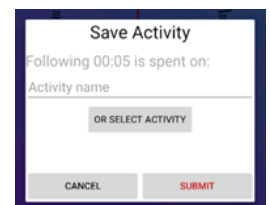


Figure 8: Save Activity Dialog

To implement the custom dialog, first the custom layout out was created (dialog\_save.xml) and was inflated with LayoutInflater object then it was passed to AlertDialog.Builder object with setView() method:

```
val saveDialogView =
LayoutInflater.from(this).inflate(R.layout.dialog_save,null)
val saveDialogbuilder = AlertDialog.Builder(this)
.setView(saveDialogView)
val saveActivityDialog = saveDialogbuilder.show()
saveActivityDialog.setCancelable(false) // prevent user to close the dialog by
clicking outside the dialog
```

On the dialog users can type a new activity name on the edit text view or clicking Select Activity it will display another Alert Dialog.

```
saveDialogView.bt_selectActivity.setOnClickListener {
    val listItems:Array<String> = activityList.toTypedArray()
    val activitySelectBuilder = AlertDialog.Builder(this)
    activitySelectBuilder.setTitle("Choose activity")
    activitySelectBuilder.setSingleChoiceItems(listItems,-1){
        dialogInterface: DialogInterface, i :Int ->
            saveDialogView.et_activityName.setText(listItems[i])
            dialogInterface.dismiss()
    }
}
```

The above snippet shows that another Alert Dialog (activitySelectBuilder) was built with setSingleChoiceItems() which will take the Array of existing activities of the users from database and display it to be selected, and the edit text will be updated with the name selected. Since the functionality of retrieving existing activities from the dataset was not implemented yet at this stage, a dummy Array List was used to test the functionality. So, this above code snippets were an earlier version of the code which were changed and updated in the later iterations,

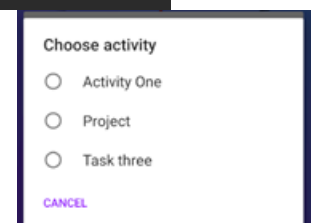


Figure 9: Select Activity Dialog



the updated code snippet is available in the **Appendix E12**. Clicking cancel would simply take back to the stopwatch where users could continue with their activity.

#### 5.4.5 Iteration 2

The backend functionality of the application was implemented in this iteration. The database and user authentication were implemented with Google's Firebase. Authors explains how the users were registered and authenticated, and how the activity progress recorded from the previous iteration were store in the database. Author discusses the methods implemented and result obtained.

##### 5.4.5.1 Firebase

As discussed in the previous (Literature review and method) sections, Firebase was used for backend servers and functionalities. Firebase is very simple and yet powerful and efficient. The author could manage the whole backed tasks from single Firebase console, it had all the backend services needed for an android application.

##### 5.4.5.1.1 Firebase- Authentication

Most apps need to know the identity of a user. Knowing a user's identity allows an app to securely save user data in the cloud database and provide access when needed. The Firebase authentication service was used to register and authenticate the users to the app. Provided documentation from the Firebase website was used to declare the dependencies in the build.gradle file:

```
// Import the BoM for the Firebase platform
implementation platform('com.google.firebase:firebase-bom:26.7.0')
// Declare the dependency for the Firebase Authentication library
implementation 'com.google.firebase:firebase-auth-ktx'
```

Using the Firebase Android BoM, the app will always use compatible versions of the Firebase Android libraries.

##### 5.4.5.1.2 Firebase- Firestore

Firestore is flexible scalable cloud databases, it was used by the author as it meets the requirements of the application. The main key features that convinced the author use Firestore was that it can keep all the users' data in sync across all the devices and offers offline support.

Cloud Firestore is a NoSQL which means there are no tables or rows, instead the data are stored in documents organized into collections. Firestore is optimized to store large collections of small documents. All documents must be stored in collections, documents can contain subcollections and nested objects, both of which can include primitive fields like strings or complex objects like lists (Cloud Firestore Data model | Firebase, 2021).

Author's Data Structure was designed where the User Collection contains all the users, each in a separate document. Then each user document will contain a collection of activity documents containing the activity details and each activity document will have a collection of records documents containing the activity progress records.

There were two ways to store data on the firestore, one was to create a HashMap of data and pass to the set method or the other way was defining a data class that can hold the data and passing the instance of that data class to the set method. The second method was preferable to the author.

Dependency was declared following the documentation:

```
// dependency for Firestore library  
implementation 'com.google.firebase:firebase-firestore-ktx'
```

The version was not need be stated it was maintained by the Firebase Android BoM

#### 5.4.5.2 App Intro

The Application intro was the first screen that users needed to interact with once they had installed the application. This screen displayed the application name with brief description and buttons to login or Sign Up with a nice background theme. The UI design was implemented in the activity\_intro.xml file. A light sliding animation was implemented to give a higher quality look and feel. The animation properties were set in the Anim folder.

The functionalities were coded in the IntroActivity.kt file, all the animation were loaded and functionalized using the AnimationUtils class. If the users pressed the login button, then they would be directed the Login Activity (screen). It was done by creating an object of Intent class with the context and the activity class (screen to be jumped to) as the constructor. An Intent is an object that provides runtime binding between separate components, such as two activities (Start another activity| Android Developers, 2021). Then the startActivity method takes user to the activity:

```
bt_login.setOnClickListener{  
    val intent = Intent(this, LoginActivity::class.java)  
    startActivity(intent)  
    finish()  
}
```

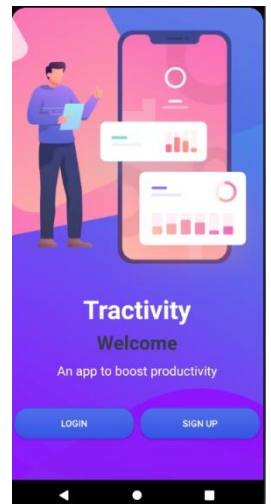


Figure 10: Intro Screen

The sign-up button takes users to the sign-up screen which was implemented the same way as the login button, code is available in **appendix E14**.

#### 5.4.5.3 implementing – user Sign Up

The primary step needed by the users to use the application is to create an account. It is done through the sign-up screen. Users is required to enter a username, email and password.

To ensure the user entered the credential correctly, a validation check was implemented as shown in the **appendix E15**. since the users are asked to re-enter password twice, a password matching function was needed to implement to check if the entered passwords matches:

```
//checks if both the passwords matches  
private fun isPasswordMatched (password:String, reTypedPassword:String)  
:Boolean{  
    return if(password == reTypedPassword){  
        true  
    }else{  
        showError("Passwords do not match, please retype")  
        false  
    }  
}
```

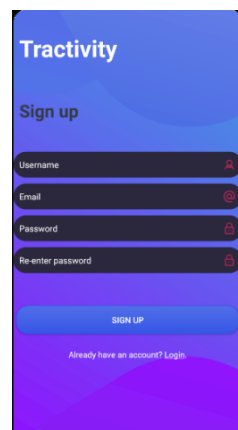


Figure 11: Sign-Up Screen

if the validation check is passed then the user account is created in the backend server by Firebase Authentication service and the user is directed to the home screen of the application (Tractivity). The code snippet in **appendix E16** shows that it was implemented with the

'createUserWithEmailAndPassword' method of FirebaseAuth class, the method takes the email and password entered by the user as the parameters.

#### 5.4.5.4 Store user Info in the Firestore

The user's info was also needed to be stored in the Firestore database as document in 'Users' collection. So, as soon a new user is added in the authentication server it had to be added in the firestore database. To implement this a User Object class had been created to hold the user info, the data inputted in the sign-up form was then used to create an instance of this Class. The method 'registerUserOnDB' was implemented in the FirestoreClass which is a class that holds all the methods to store data on Firestore. This 'registerUserOnDB' method takes the user object and create a document inside the 'Users' collection to store the user info.

```
fun registerUserOnDB (user: UserClass){  
    firestore.collection(Constants.USERS).document(getCurrentUserID())  
        .set(user, SetOptions.merge()).addOnSuccessListener {  
            Log.d("DDDBB", "Document saved")  
        }  
        .addOnFailureListener{ e->  
            Log.w("DDDBB", "Error adding document")  
        }  
}
```

#### 5.4.5.4 implementing – user login

The existing users can login to their account by entering their email address and password.

Once the users input their credentials and press login, a function called validateLogin was implemented to checks if the users had entered the details needed which displays an error Snackbar if anything was missing (code is available in **Appendix E18, E19**). Once it passed the validation check, the function called signInWithEmailAndPassword from FirebaseAuth class takes the typed user email address and password to authenticate the users. If the authentication was successful users would be redirected to the home screen if not, then a Toast displaying error would be displayed. Full source code is available in the **Appendix E20**.

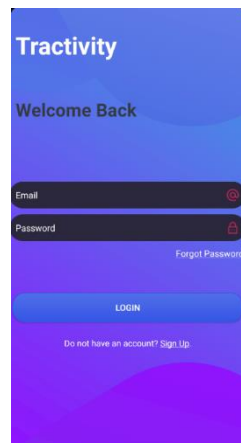


Figure 12: Login Screen

#### 5.4.5.5 Reset Password

Users can reset their password if forgotten. Clicking the 'forgot password' will direct the user to the password reset screen where the user had to enter their email address they initially used to register on the application. SendPasswordResetEmail() function from Firebase Authentication service will then send an email with instructions on resetting the password, the new password set by the user will then be updated in the backend server. The **appendix D fig4** contains the UI design and **appendix E21** contain the logic code.

#### 5.4.5.6 Save Activity Progress

Users must be able to store the activity progress recorded through the stopwatch. As discussed in the **section 5.4.4.2**, the UI and the structure were implemented in iteration 1 but in this iteration the author implemented the backend functionality of storing the recorder activity progress and activity data on the database.

As shown in the **appendix E12**, the custom dialog implemented takes the activity progress and the activity name and store it in the Firestore. As users can also select the names of the existing activities they are working, all the activity names were queried from the database and held in

an Array list which was then passed to the 'setSingleChoiceItems' function to be displayed and selected by the users.

Once the submit button is clicked, the saveDialogFunction check is the user chose an activity from the list or a new activity name was type. If the activity was one of the existing ones, then only the progress is stored as a document in the 'records' collection in the database by calling the saveRecordOnDB method implemented in the Firestore class:

```
fun saveRecordOnDB(record: ActivityRecordClass, activityName :String){
    firestore.collection(Constants.USERS)
        .document(getCurrentUserID())
        .collection(Constants.ACTIVITIES)
        .document(activityName).collection(Constants.RECORDS)
        .add(record).addOnSuccessListener {
            Log.d("DDDBB", "record saved")
        }
        .addOnFailureListener{ e ->
            Log.e("DDDBB", "record not saved")
        }
}
```

But if the activity name was typed and does not already exist, then a new document is created in the Activity collection containing the new activity data. It was done by calling the implemented saveActivityOnDB function from the FirestoreClass before calling the saveRecordOnDB function:

```
fun saveActivityOnDB(activity: ActivityClass) {
    firestore.collection(Constants.USERS)
        .document(getCurrentUserID()).collection(Constants.ACTIVITIES)
        .document(activity.name).set(activity, SetOptions.merge())
        .addOnSuccessListener {
            Log.d("DDDBB", "Activity saved")
        }
        .addOnFailureListener{ e ->
            Log.e("DDDBB", "activity not saved")
        }
}
```

#### 5.4.5.7 Parsing the record data

The chronometer view displays the data in a text format which means that the recorder time spent in an activity was a string. Since the author needed to use this data for calculation later in the application development, a parser was needed to be implemented that could take the string time format of mm:ss or hh:mm:ss and convert it to a number value. The parser split the time string at every ":" and store it in an array, and according to the string size the time in parsed into seconds and returned as long value. The implemented code is available in the **appendix E22**.

#### 5.4.5.8 custom App bar

To enhance the look of the application, the default app bar was eliminated as it looked outdated, instead a custom app bar was designed and implemented. It was designed according to the screen functionality and then implemented in the Kotlin activity class file by a function called 'setupActionBar'.

#### 5.4.5.9 App Navigation Drawer

An app navigation drawer was implemented after completing all the home screen features, to allow users easy navigation between other screens of the application. Different navigation techniques are available for android such as top app bar menu navigation, toolbar sliding menu navigation, sliding navigation drawer and bottom navigation. Out of these, sliding the navigation drawer was considered optimal approach. It is simple and easy to recognise since it is implemented by most android application, also the application is profile based so the navigation drawer can display user image and name on the navigation header. The navigation drawer is a UI panel that displays all the application screen name that users can jump to. The drawer appears when the user touches the three lined icon in the app bar or simply when the user swipes a finger from the left edge of the screen.

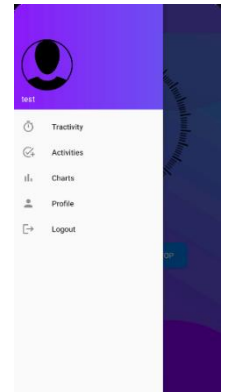


Figure 13:Navigation Drawer

First the custom navigation drawer header was designed in a xml file called nav\_header.xml with a circular image view that would hold the user's images and name and the navigation body UI was implemented in the activity\_trmai.xml inside a navigation view. The menu list was set in the drawer\_menu.xml file in the res/menu folder and referenced in the navigation view.

The functionality of taking the user to the clicked menu option was implemented by overriding the 'onNavigationItemSelectedListener' method of the build in Navigation class, code is found in the **appendix E23**.

The sliding toggle drawer was implemented using the GravityCompat class. It was passed to the icon listener of the custom Action Bar:

```
private fun toggleDrawer() {
    if (drawer_layout.isDrawerOpen(GravityCompat.START)) {
        drawer_layout.closeDrawer(GravityCompat.START)
    } else {
        drawer_layout.openDrawer(GravityCompat.START)
    }
}
```

#### 5.4.5.10 Populate Navigation header

The navigation header was updated according to the user logged in, as shown in **Appendix E24**. it was done by retrieving the user data from the firestore, a function called getCurrentUserID was defined that queries from the authentication server and return the current user logged in:

```
fun getCurrentUserID() : String {
    var currentUser = auth.currentUser
    var currentUserID = ""
    if(currentUser != null){
        currentUserID =currentUser.uid
    }
    return currentUserID
}
```

#### 5.4.5.11 Always stay logged in

To avoid unnecessary logging in every time the users launches the application, a stay logged in functionality was implemented. The logic was implemented in the IntroActivity class that checks if a user is already signed in, if he is then he will be taken straight to the main screen, if not then the user would be prompted to login. Code snippet:

```
if(currentUserID.isEmpty()){
    startActivity(Intent(this, TractivityMain::class.java))
    finish()
}
```

#### 5.4.5.12 Logout

Logout feature was implemented in the navigation drawer as shown in the **appendix E23**. Clicking the logout button the user will be logged out of the application and taken to the intro screen and all the flags will be cleared from the activity stack. The 'signOut' method of the firebase authentication library was used to log out the user from the backend server.

### 5.4.6 Iteration 3

In iteration 3 author implemented the objective of user profile, activity details screen and displaying the activity time spent in pie chart and bar to let user visualize their progress. This section focused more on retrieving the activity data from the Firestore and present them to users and also create charts to help users visualize their progress. Each time any data was retrieved it was converted to its corresponding data object class for better handling, for example when the user data was retrieved it was converted to User data class:

```
val loggedUser: UserClass? = documentSnapshot.toObject(UserClass::class.java)
```

The free open-source android charting library called MpAndroidCharts were used to implement the chart features.

#### 5.4.6.1 User profile

The user profile screen displays the user data, as mentioned earlier each user data was also stored in the Firestore as document in the 'user' collection. Each user has a unique ID, this ID is the name of document which means the user ID of the logged in user was needed to be retrieved from the authentication server and use that ID to fetch the user data document from the 'Users' collection. The ID was retrieved using the 'getCurrentUserID' method defined in the Firestore class. Then a function called 'populateProfileActivity' was implemented to populate the profile screen with the retrieved data. The code can be found in the **appendix E25** and the UI in **Appendix D15**.

#### 5.4.6.2 Change user password

Users can change their password, to implement that a custom dialog was designed that will ask user type and re-type their new password. Once the dialog is submitted, the application checks if the typed passwords are correct, if they are then it will be updated in the firebase Authentication server. Full code snippet is posted in the **appendix E26**.

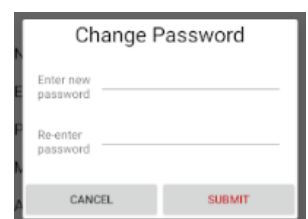


Figure 14: Change Password Dialog

#### 5.4.6.3 Display Activities in List

A screen was implemented that displays all user's activities in a list format, it was done using the RecyclerView. RecyclerView (a subclass of ViewGroup) takes the data and displays the items dynamically when they are needed. RecyclerView can display large data efficiently by not destroying the view of the items scrolled off the screen, instead those views are reused for new items.

To implement the RecyclerView logic, first an Adapter and view holder was implemented that defined how the data is displayed. The ViewHolder is wrapper around the View that contains the layout for an individual item in the list and the Adapter creates ViewHolder objects as needed to set the data for those views (RecyclerView | Android Developers, 2021). Three methods were overridden inside the Adapter class; onCreateViewHolder() to create new viewHolder whenever needed, onBindViewHolder() to associate a ViewHolder with data and getItemCount() to get the size of the data set. Also, a click event interface was defined that can take the user to the selected item screen. The full class code is posted in the **appendix E27**.

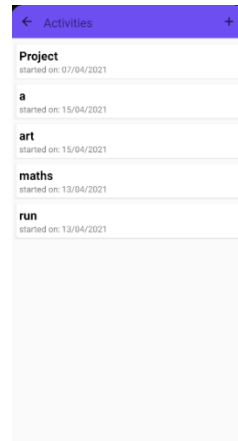


Figure 15: Activity Lists

The activity lists were then queried from the Firestore and were passed to the adapter to populate the recycle view, the items were managed using a Linear layout Manager to list the items in linear list format. The code snippet of populating the recycle view is posted in **Appendix E28** and the code snippet of querying and displaying is available in **appendix E29**.

#### 5.4.6.4 Create/add new Activity manually

The top app bar was designed with a '+' icon that users can click to create new activity manually without progress record. Clicking the icon, the user will be directed to the 'AddActivity' where user can enter the activity name and details, upon submitting it will be stored in the firestore. The UI design is available in the **appendix D fig11**, and the class code in the **appendix E30**.

#### 5.4.6.5 Activity details with all records

Clicking an activity name from the activity list will take the user to the activity details screen, where the full detail of the activity is displayed with calculated total time spent and the full record history in a recycle view.

The following code snippet shows that the onItemClick method in onItemClickListener interface that author implemented in the Activity list adapter was overridden to send the activity name that was clicked by the user to the activity details screen:

```
override fun onItemClick(position: Int, item: ActivityClass) {  
    val intent = Intent(this@ActivitiesActivity,  
        DetailsActivity::class.java)  
    intent.putExtra(Constants.ACTIVITYNAME, item.name)
```

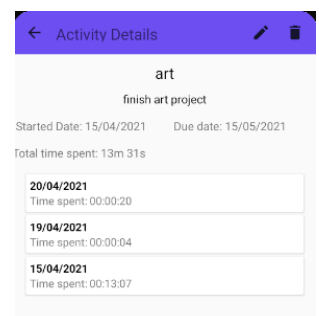


Figure 16: Activity Details Screen



```
startActivity(intent)
}
```

The code snippet in the **appendix E31 and E32** show that activity name received was used to query from the firestore and retrieve the activity details to populate the screen. Then a recycle view was implemented in a similar step used earlier (defining an Adapter and populating with data). This recycleView displays the record history queried from the firestore in a list format.

To implement total time spent, first all the progress data was retrieved from the database and were added. Since the calculated value was in seconds and in Long primitive data time, author had to implement a function called 'formateProgress' to format the second value into hours minute and seconds, and convert it to string format of hh mm ss. The code snippet of this functionality is available in the **appendix E33 and E34**.

#### 5.4.6.6 Edit activity

Users can change the description and due date of an activity by clicking the edit icon. Users can also write a small note or reminder in the description section so giving an option to change it can improve usability. This was implemented in a function called 'displayActivityEditDialog', it displays a custom alert dialog and with the description and due date in an edit text view retrieved from the Firestore. Users can modify it and once submitted the activity document on the Firestore will be updated with the new data using the 'update' method of Firestore library. The full code snippet is available in the **appendix E35**.

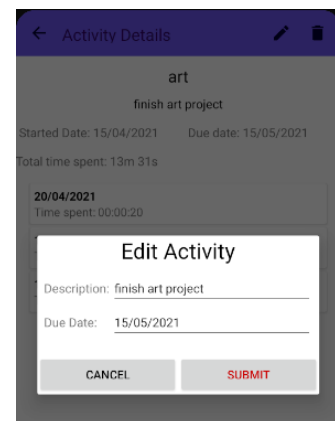


Figure 17 Edit Activity Dialog

#### 5.4.6.7 Deleted activity

Once an activity, task or project is over, user can delete them by clicking the bin icon on the app bar. To implement that a function called 'deleteActivity' was defined that takes the activity name and deletes the activity document from the firestore using 'delete' method of firestore library. But before deleting, users are alerted with a confirmation dialog to confirm the action. The Code is available in **Appendix E36**.

#### 5.4.6.8 Charts

This was the final feature of the application where the user will be presented with a pie charts and bar chart to visualize the activity progress. It was implemented using the open-source charting library called MPAndroidCharts available in the GitHub. Users can view different activity progress and reflect upon them.

The dependencies were declared in the Gradle file:

```
implementation 'com.github.PhilJay:MPAndroidChart:v3.1.0'
```

Author followed sever online tutorial to understand the library, but no tutorial was found that specifically works with the firestore database, so author had to combine the knowledge that was learnt previously to implement the intended functionalities.

#### 5.4.6.9 Pie chart

To implement the Pie chart. First a Piechart view imported from MPAndroidCharts library was defined inside a cardView in activity\_charts.xml file. This view was responsible of

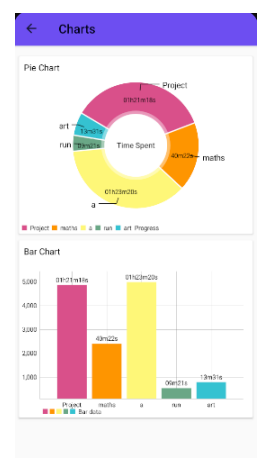


Figure 18: Charts Screen



displaying the pie chart. The logic was implemented in the 'ChartsActivity.kt' file. The function called 'showPieChart' was defined that queries all the activity documents and for each activity document the record histories were retrieved and stored in an array list of type 'PieEntry', each element containing the total time spent and the activity name. Then this array list was passed to another implemented function called 'createPieChart' that takes the data and creates the pie charts in the pieChart view. The characteristics of the pie chart was also defined in the 'createPieChart' function. The code snippet is available in the **appendix E37** .

#### *5.4.6.10 Bar chart*

UI design was done the same way as Piechart, by defining the BarChart view inside a card view in the xml file, the data was queried similar way as for pie chart. But the difference was in bar chart features and data structure. Inside the 'showBarChart' two array list ('activityBarEntries' and 'activityBarLabels') were used, one of BarEntry and one of String. Total time spent in an activity with an index value was stored as BarEntry inside 'activityBarEntries', and the corresponding activity names were stored in the 'activityBarLabels'. Then these two array lists were passed to a function called 'createBarChart' where author defined the features of the bar charts and how the data will be displayed in the bar chart view. The code snippet is available in **appendix E38**.

#### *5.4.6.11 Chart value formatter*

To change the charts values from second to a time format, author implemented a class called 'ProgressValueFormatter' which was inherited from 'ValueFormatter' class. Inside this class the 'getFormatValue' method was overridden with the required format as shown in the **appendix E39**.

### **5.5 Evaluation- Testing**

The testing was done by the author. As discussed previously, a time was dedicated end of each iteration to test if the functionalities worked as intended. Any major bugs were fixed before moving to next iteration. Then a final testing was carried out at the end of the full development to test if all the objectives work as planned. The final test table is available in the **appendix E40** .

## Chapter 6: Conclusion and Discussion

### Primary Objective:

The primary objective was “to develop an android application that will allow users to track amount of time spent on different activities, tasks or projects and help to increase productivity” which was successfully implemented, users can track their time spent, visualise, and analyse it through charts to increase productivity.

### Sub-objective 1

“Build An android Application that can perform the intended functionalities” was successfully met by developing the application that can be installed and run in any android device. The Android Studio IDE was used to achieve this. The publication on Google Play is yet to happened since author wants to include some high-level feature before that.

### Sub-objective 2

“User friendly user interface”. Author researched different application designs through ‘Behance’ website and similar applications to successfully plan and design a user interface that user can interact with without getting confused. The design codes were coded in the XML files.

### Sub-objective 3:

“Easy navigation between different functionality”, an easy-to-understand navigation drawer was successfully implemented that can let the users to navigate to different applications screens.

### Sub-objective 4:

“A stopwatch that user can start, pause, and stop”. The stop watched was successfully implemented according to its functionalities, it allows users to record the time spent on the activities so it can later be store in the database. Users can pause, resume, reset and stop the stopwatch depending on their activity.

### Sub-objective 5:

“User account/login”. User account creation and authentication was successfully implemented with Firebase Authentication service. This was one of the primary step users needed to use the application.

### Sub-objective 6:

“Users’ data needs to be stored on the cloud database so they can access the data from any device”. Firestore was used to successfully achieve this back-end objective by storing the users’ data in ‘documents in collection’ structure.

### Sub-objective7:

“Add activities, projects or tasks” was successfully implemented, users can manually add/create new activities with a description and due date in the activity screen of the application. The entered details are then stored in the firestore and can be used by other app functionalities.

### Sub-objective 8:

“User profile”. It successfully implemented in a ‘profile’ screen in the application where the user information is retrieved from the firestore and displayed, user can change their password from here which can be updated in the firebase authentication server.

### Sub-objective 9:

“View charts”, this was the final feature implemented successfully which fetches the activity progresses from the firestore database and creates Pie and bar charts displaying different time spent in different activities. This helps the users to visualise the data and compare them to improve their productivity.

### Resources used:

Several learning resources were used to learn about the application development and to meet the planned objectives, some of the main resources used by the author were:

- Udemy: it is an online course provider; Author used this platform to learn about Kotlin programming and android app development.
- Android Developers: it is the official site for android app developers that provides all the documentations and easy tutorials both in java and kotlin programming language.
- YouTube: there were many free resources available in the YouTube that author used to implement android features. Some of the channels followed were: “Coding in Flow”, “tutorialsEU”, “freeCodeCamp.org”, “Telusko”, “Sarathi Technology”, etc. Although most of the tutorials were in java the author used his kotlin knowledge to implement them in kotlin.
- Firebase Documentation: This contained the tutorials with explanation and documentations of the firebase services.
- Stack overflow: a knowledge sharing community forum where users can ask software related questions. This was very helpful to find solutions of similar questions or bugs encountered by the author.

### Knowledge Gained

Before completing the project, the author had almost no knowledge about Android development, java programming language was the only relevant knowledge he had. Being able to successfully develop an android application with thousands of lines of codes was considered very challenging. On top of that the author took the challenge of developing the application in Kotlin programming language. Author had spent many hours studying and researching Kotlin programming language and android app development features and firebase in order to successfully complete the project. Upon completing the project, the author has gained huge improvement in programming and project development skills. He is very comfortable with developing any android application. He is now also experienced with the firebase platform and NoSQL Firestore database.

### Future work

Although the main objectives of the application were achieved, the author wants to implement some extra high level and optional features such as progress sharing, controlling the app from notification or lock screen, a group activity section where different users can work on a single activity/project, a leader board among the users’ group showing who did most work. Author

also wants to improve the charting feature where the users can view work done in days, weeks, or months. Author plans to implement these features during the summer and publish the application on Google Play once completed.

## Conclusion

In conclusion, the author had an incredible experience and challenging moments to learn and build a project of this size from scratch. All the stages from the initial plan to gathering requirement, designing, implementing, testing, and reporting have taught something new. Author admits that this project development process was very important part of his degree, it helped him to realise his potential as a computer science student. Even though many challenges were faced trying to maintain the work life with university and many sacrifices were needed to make, author is very content to be able to create a final product of his own.

## Chapter 7: Glossary

This table contains the meaning of the specialised terms used in the report.

Term	Meaning
<b>SDK</b>	Software development kit, collection of software development tools.
<b>API</b>	Application Programming Interface, collection of method that provide access to functionalities of a software.
<b>XML</b>	Extensible Markup Languag, used to create UI design.
<b>Wireframe</b>	A way to design UI at the structural level.
<b>UI/GUI</b>	User Interface, the means by which user will interact graphically with the application.
<b>ViewGroup</b>	The base class for UI layouts and views containers.
<b>CardView</b>	An API used for showing information inside cards in an elevated look above their containing view group.
<b>RecyclerView</b>	A subclass of view group that handles large data and view as needed.
<b>Front end</b>	Software development layering terminology – Client facing layer of the application.
<b>Backend</b>	Software development layering terminology –Server side logic and infrastructure.
<b>Gradle</b>	System files that handle the Android build tasks by compiling the application for installation in a device.

## Chapter 8: References

AltexSoft. 2018. Top 20 Tools for Android Development. [online] Available at: <https://www.altexsoft.com/blog/engineering/top-20-tools-for-android-development/>> [Accessed 2 February 2021].

2019. [online] Available at: <https://www.velvetechnology.com/blog/5-key-mobile-development-approaches/>>[Accessed 2 February 2021].

DEMCHENKO, M., 2020. The Best Tools for Android Software Development. [online] NCube. Available at: < <https://ncube.com/blog/the-best-tools-for-android-software-development> > [Accessed 2 February 2021].

Sims, G., 2019. I want to develop Android apps — What languages should I learn?. [online] Android Authority. Available at: <<https://www.androidauthority.com/develop-android-apps-languages-learn-391008/>> [Accessed 2 February 2021].

Gill, N., 2020. Kotlin vs Java Comparison - Which One is Better? (2020). [online] XenonStack. Available at: < <https://www.xenonstack.com/blog/kotlin-android/#:~:text=Any%20chunk%20of%20code%20written,run%20in%20a%20Kotlin%20project.>> > [Accessed 2 February 2021].

IT Info-Tech. 2019. Difference Between FireBase, MySQL And SQLite Database - IT Info-Tech. [online] Available at: < <http://www.itinfotech.in/database/difference-between-firebase-mysql-and-sqlite-database/#:~:text=FireBase%20It's%20a%20cloud%20service,stored%2Fprocessed%20in%20a%20cloud.&text=SQLite%20is%20local%20database%20on,suitable%20for%20real%20time%20application> > [Accessed 2 February 2021].

Bowes, J., 2014. Agile vs Waterfall - Comparing project management methods. [online] Manifesto. Available at: < <https://manifesto.co.uk/agile-vs-waterfall-comparing-project-management-methodologies/> > [Accessed 2 February 2021].

Firebase. 2021. Choose a Database: Cloud Firestore or Realtime Database | Firebase. [online] Available at: < <https://firebase.google.com/docs/database/rtdb-vs-firestore> > [Accessed 3 February 2021].

Jigsaw Academy. 2021. 5 Important Types Of Agile Methodology (2021). [online] Available at: < <https://www.jigsawacademy.com/blogs/product-management/types-of-agile-methodology/> > [Accessed 15 March 2021].

TechRepublic. 2021. *Why planning is the most critical step in project management*. [online] Available at: < <https://www.techrepublic.com/article/why-planning-is-the-most-critical-step-in-project-management/#:~:text=Project%20planning%20plays%20an%20essential,agreed%20product%2C%20service%20or%20result>. > [Accessed 15 March 2021].

Heller, M., 2021. *What is Kotlin? The Java alternative explained*. [online] InfoWorld. Available at: < <https://www.infoworld.com/article/3224868/what-is-kotlin-the-java-alternative-explained.html> > [Accessed 17 March 2021].

Android Developers, 2021. Chronometer. [Online] Available at: <https://developer.android.com/reference/android/widget/Chronometer> [Accessed 2021]

Android Developers, 2021. View Animation. [Online]  
Available at: <https://developer.android.com/guide/topics/graphics/view-animation>  
[Accessed 2021]

Android Developers, 2021. Create A notification. [Online]  
Available at: <https://developer.android.com/training/notify-user/build-notification>  
[Accessed 2021]

Android Developers, 2021. Dialogs. [Online]  
Available at: <https://developer.android.com/guide/topics/ui/dialogs#kotlin>  
[Accessed 2021]

Educative: Interactive Courses for Software Developers. 2021. What is Firebase?. [online] Available at: <https://www.educative.io/edpresso/what-is-firebase> [Accessed 20 March 2021].

Firebase. 2021. *Firebase Authentication*. [online] Available at: < <https://firebase.google.com/docs/auth> > [Accessed 1 April 2021].

Firebase. 2021. Choose a Database: Cloud Firestore or Realtime Database | Firebase. [online]  
Available at: <https://firebase.google.com/docs/database/rtdb-vs-firestore>  
[Accessed 1 April 2021].

Abhiandroid.com. 2021. *XML in Android: Basics And Different XML Files Used In Android | Abhi Android*. [online] Available at: < [https://abhiandroid.com/ui/xml#:~:text=Strings%20xml%20File\(strings, the%20reusability%20of%20the%20code](https://abhiandroid.com/ui/xml#:~:text=Strings%20xml%20File(strings, the%20reusability%20of%20the%20code) > [Accessed 10 April 2021].

Android Developers. 2021. *Dialogs* | *Android Developers*. [online] Available at: < <https://developer.android.com/guide/topics/ui/dialogs> > [Accessed 11 April 2021].

Android Developers. 2021. Start another activity | *Android Developers*. [online] Available at: < <https://developer.android.com/training/basics/firstapp/starting-activity> > [Accessed 12 April 2021].

Firebase. 2021. *Cloud Firestore Data model* | *Firebase*. [online] Available at: < <https://firebase.google.com/docs/firestore/data-model> > [Accessed 20 April 2021].

Android Developers. 2021. *Create dynamic lists with RecyclerView* | *Android Developers*. [online] Available at: < <https://developer.android.com/guide/topics/ui/layout/recyclerview> > [Accessed 28 April 2021].



# **City, University of London**

## **BSc (Hons) Computer Science IN3007 Individual Project**

academic year: 2020-2021

### **Project Definition Document**

‘Tractivity’ An Android Application to keep track of the time spent on activities, projects, or tasks.

Proposed by:  
Tanvir H Ahamed  
Tanvir.Ahamed@city.ac.uk

Project Supervisor: Alex Ter-Sarkisov

## Problem to be solved

Time management is very crucial part of our lifestyle in order to balance work life and personal life. Sometimes we overload ourselves with so many activities that we lose track of the time and loose focus on the tasks that are more important, or sometimes we get lost on procrastinating on unnecessary things. So, I have decided to create an application to that can help to keep track of the time spent on different activities. The user can track his/her daily activities and keep track of the certain projects or works he/she is working on. The amount of time spent on different activities will be stored and then at the end of the day or week or month the user can visualize the time spent on activities as a pie chart, bar chart or progress bar. User can use this information justify if he/she is spending his/her time wisely, also make adjustment to his/her routine. If the user has ongoing projects, then he can see if he/she is spending enough time on it.

There are many timer and time management apps available in the android application market (Play Store), but they do not provide easy solution. Some apps don not have tracking functionality and some are too overloaded with confusing irrelevant functionality which can give user hard time to figure it out. My aim is to keep the application simple where user can achieve the functionalities with less interaction as possible.

To develop my android application, I will use android studio which is the Official integrated development environment for Google's android operating System. Even though Java can be used for android development I chose to code my program in Kotlin programming language as it is currently the official language for android, and I want to take the chance to learn a new language. I also plan to several application programming interfaces (APIs) and built-in libraries to build my Application.

API/Libraries	How it will be used
java.util.Timer	To apply the timer functionalities
Google Firebase	For the backend database and login functionality.
Charts.kt	Charting API to create the understandable and meaningful charts.

## Project Objectives

### Main objective

The project shall provide user functionality to track amount of time spent on certain activities throughout the day and also, increase productivity by recording time spent on certain works or projects.

### Sub-objectives

Objectives	Testable
Build An android Application that can perform the intended functionalities.	User can easily install the application on their device and run it.
User friendly user interface	User must not be bombarded with lost text; it shall be easy to use and not confusing.
A timer that user can start, pause, and stop;	when on an activity user can start the timer and once stopped the application will ask where the time was spent and store it.
User data needs to be stored on the cloud so he/she can access the data from any device.	User shall be able to login to his/her account from any device and access the data. Also be able to record activities from any device.
Create account/Login.	User can create accounts or login to his existing account.
Add projects or tasks	User can insert any project/tasks they are working on or needs to work on. Time spent on those tasks will be recorded and user can visualize the progress.
View charts	User can visualize the activity records in a meaningful chart format. User can analyse it and focus on productivity.
Control the timer from notification panel	User should be able to control the timer from the notification panel easily, this will

	allow the user to stay focus on their tasks and not get distracted by smartphone.
Notify user to track their time	Notification will be sent to inactive user to track their time and motivate them to focus on important tasks and help them to increase productivity.

## Project Beneficiaries

Beneficiaries that can benefit from this project and how it will benefit them:

- **Any individual looking to increase productivity:** Anyone who have any tasks work they want to focus on can use this app to increase productivity or simply can record every task the user does and reflet and adjust their habit.
- **Students:** Students will be highly benefited, and they can keep track of the time they spend on studies. For example, students have many Modules/subjects that they need to focus on so he/she can simply list all the modules in the application and every time he/she spends time on those modules it will be recorder.
- **Employers and Employees:** Employers can ask employees to use the app when they are working on any project tasks so they can monitor their productivity.

## Work Plan

As I am very new to android development and Kotlin programming language with a tight schedule, I have carefully planned my work plan by breaking my project into smaller iterations. First, I will implement a very basic functionality and gradually increase features. I will be using Agile software development methodology where I will plan and test each iteration as I go. I will follow and learn from online resources and courses while I build my application.

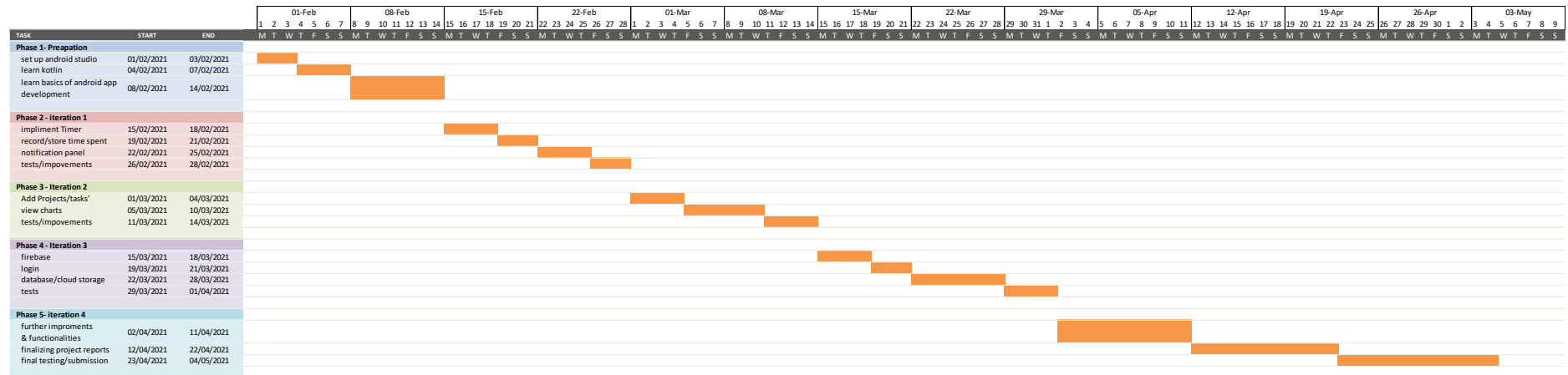
**Phase 1- Preparation:** I have assigned 2 weeks to learn the basics of android development before I dive into my project by enrolling to some online courses that will teach me about android SDK and Kotlin programming language.

**Phase 2- 1<sup>st</sup> iteration:** here i will plan and implement the foundation of the application, a timer with its functionalities (start, pause and save) and the data will be saved on the app. I will build the notification panel and user can use it to control the application too. I will end this iteration by testing whether the buttons respond correctly, if the data are stored properly and the notification panel is responsive.

**Phase 3 - 2<sup>nd</sup> iteration:** in this iteration I will implement the 'Add project functionality' where user will be able to add the projects they are working on or will be working, here the user will be able to see how much time they are spending on them and visualize the progress. I will also implement the chart functionality where user can see all the saved records in various charts format. I will end this phase by testing if the features work accurately.

**Phase 4- 3<sup>rd</sup> iteration:** Here I will take my application online; the data will be stored in the cloud and the users will have logins. This iteration involves backend development which can be tricky sometimes as I will be using firebase, I have devoted some time to learn about it before I start.

**Phase 5: finalization:** By this time, I should be done with the main features of my application. So here I will further improve it by making the GUI more interactive and will also add more helpful features. Also, as I mentioned in project risk, I might miss some deadlines so I will use this time to catch up. I will carry my final tests and finalize my reports before I submit it on 4<sup>th</sup> may. This phase is during the revision weeks I need to be careful not to clash with other modules.



## Project Risk

Any project can contain risk of failure during the development, the best approach is to identify the risks before it happens and have a plan to mitigate them. Below I have listed all the risks that my project can encounter and mitigation/prevention technique to prevent damages. I will score each risk from 1 to 5, 1 very low and 5 very high.

Risks	Risk score	Mitigation/prevention
No experience in android development.	4	There are many online resources to learn android development that can be accessed easily. I have experience in building other software system, those experience can be useful. Also, I have dedicated some time to learn about android development before starting the project.
No experience in Kotlin Programming language	3	Kotlin can be learnt easily as I am experienced with Java programming language which is very similar to Kotlin and uses same libraries.
As I am a part time worker in Tesco, it poses a risk of not meeting the work plan deadlines.	5	it is very likely to happens as I must focus on other university module as well. To mitigate this, I have extended test durations which can be used to complete some missed tasks.
Never used firebase before	2	Have dedicated some extra time to learn about android Firebase.
Not familiar with Charts.kt API to create charts	1	Documentation is easily available online. Many other charting libraries and APIs are available online
Login security	1	Security fundamental module from last term will be useful to mitigate this.
Submission of another module project.	4	Since the deadline of the project submission is 4 <sup>th</sup> may and I will have another module project to submit on 3 <sup>rd</sup> may I will have to be very careful with my time management. To prevent this clash I

		will strictly follow my work plan Gantt chart.
--	--	--

## Ethical Checklist

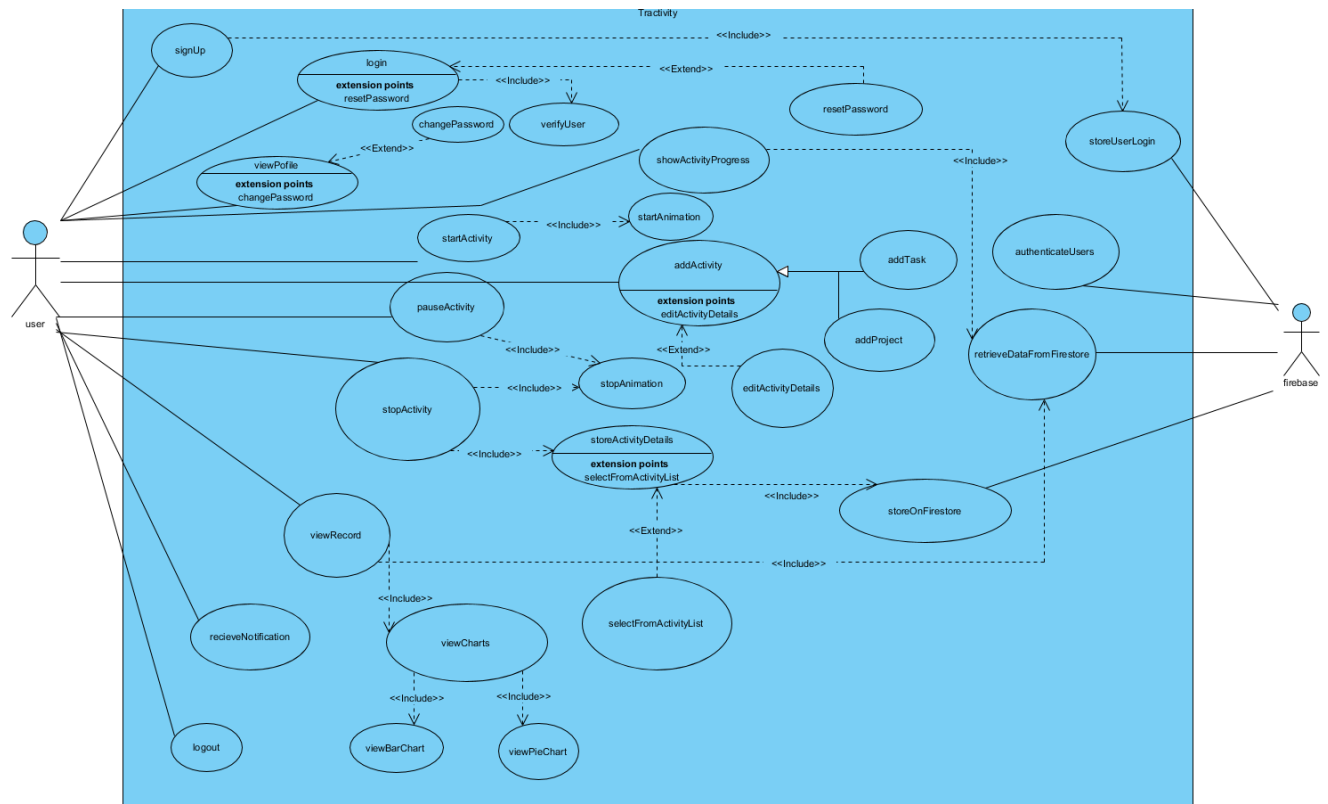
<b>A.1 If you answer YES to any of the questions in this block, you must apply to an appropriate external ethics committee for approval and log this approval as an External Application through Research Ethics Online - <a href="https://ethics.city.ac.uk/">https://ethics.city.ac.uk/</a></b>		Delete as appropriate
1.1	Does your research require approval from the National Research Ethics Service (NRES)? <i>e.g. because you are recruiting current NHS patients or staff?</i> If you are unsure try - <a href="https://www.hra.nhs.uk/approvals-amendments/what-approvals-do-i-need/">https://www.hra.nhs.uk/approvals-amendments/what-approvals-do-i-need/</a>	NO
1.2	Will you recruit participants who fall under the auspices of the Mental Capacity Act? <i>Such research needs to be approved by an external ethics committee such as NRES or the Social Care Research Ethics Committee - <a href="http://www.scie.org.uk/research/ethics-committee/">http://www.scie.org.uk/research/ethics-committee/</a></i>	NO
1.3	Will you recruit any participants who are currently under the auspices of the Criminal Justice System, for example, but not limited to, people on remand, prisoners and those on probation? <i>Such research needs to be authorised by the ethics approval system of the National Offender Management Service.</i>	NO
<b>A.2 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee, you must apply for approval from the Senate Research Ethics Committee (SREC) through Research Ethics Online - <a href="https://ethics.city.ac.uk/">https://ethics.city.ac.uk/</a></b>		Delete as appropriate
2.1	Does your research involve participants who are unable to give informed consent? <i>For example, but not limited to, people who may have a degree of learning disability or mental health problem, that means they are unable to make an informed decision on their own behalf.</i>	NO
2.2	Is there a risk that your research might lead to disclosures from participants concerning their involvement in illegal activities?	NO
2.3	Is there a risk that obscene and or illegal material may need to be accessed for your research study (including online content and other material)?	NO



2.4	Does your project involve participants disclosing information about special category or sensitive subjects? <i>For example, but not limited to: racial or ethnic origin; political opinions; religious beliefs; trade union membership; physical or mental health; sexual life; criminal offences and proceedings</i>	<b>NO</b>
2.5	Does your research involve you travelling to another country outside of the UK, where the Foreign & Commonwealth Office has issued a travel warning that affects the area in which you will study? <i>Please check the latest guidance from the FCO - <a href="http://www.fco.gov.uk/en/">http://www.fco.gov.uk/en/</a></i>	<b>NO</b>
2.6	Does your research involve invasive or intrusive procedures? <i>These may include, but are not limited to, electrical stimulation, heat, cold or bruising.</i>	<b>NO</b>
2.7	Does your research involve animals?	<b>NO</b>
2.8	Does your research involve the administration of drugs, placebos or other substances to study participants?	<b>NO</b>
<b>A.3 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee or the SREC, you must apply for approval from the Computer Science Research Ethics Committee (CSREC) through Research Ethics Online - <a href="https://ethics.city.ac.uk/">https://ethics.city.ac.uk/</a></b> <b>Depending on the level of risk associated with your application, it may be referred to the Senate Research Ethics Committee.</b>		<i>Delete as appropriate</i>
3.1	Does your research involve participants who are under the age of 18?	<b>NO</b>
3.2	Does your research involve adults who are vulnerable because of their social, psychological or medical circumstances (vulnerable adults)? <i>This includes adults with cognitive and / or learning disabilities, adults with physical disabilities and older people.</i>	<b>NO</b>
3.3	Are participants recruited because they are staff or students of City, University of London? <i>For example, students studying on a particular course or module.</i> <i>If yes, then approval is also required from the Head of Department or Programme Director.</i>	<b>NO</b>
3.4	Does your research involve intentional deception of participants?	<b>NO</b>
3.5	Does your research involve participants taking part without their informed consent?	<b>NO</b>
3.5	Is the risk posed to participants greater than that in normal working life?	<b>NO</b>
3.7	Is the risk posed to you, the researcher(s), greater than that in normal working life?	<b>NO</b>

<p><b>A.4 If you answer YES to the following question and your answers to all other questions in sections A1, A2 and A3 are NO, then your project is deemed to be of MINIMAL RISK.</b></p> <p><b>If this is the case, then you can apply for approval through your supervisor under PROPORTIONATE REVIEW. You do so by completing PART B of this form.</b></p> <p><b>If you have answered NO to all questions on this form, then your project does not require ethical approval. You should submit and retain this form as evidence of this.</b></p>		<i>Delete as appropriate</i>
4	<p>Does your project involve human participants or their identifiable personal data?</p> <p><i>For example, as interviewees, respondents to a survey or participants in testing.</i></p>	<b>NO</b>

## 1 UseCase Diagram

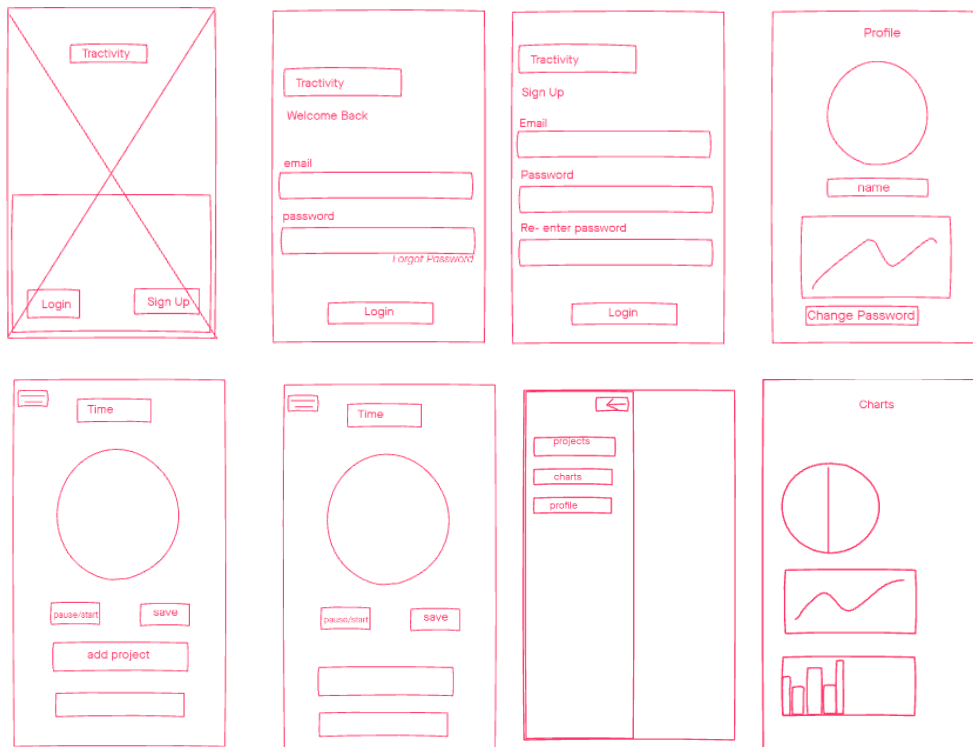


## 2 Requirement Table:

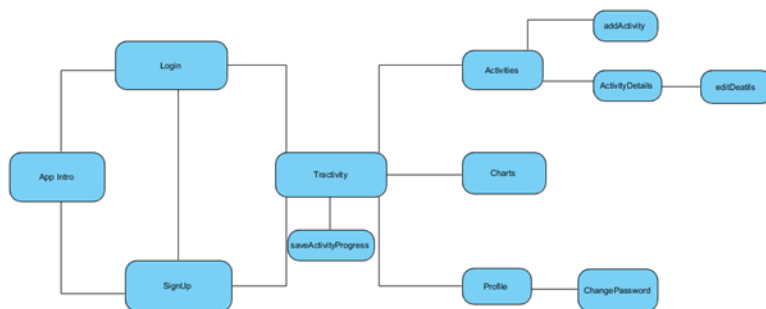
No.	Description
1.0	Users must be given option to create account or login to existing account.
2.0	Users must be able to create new account.
2.1	Users must type username, email, and password to create the account.
3.0	Users must be able to login to existing account.
3.1	Users must type valid email and password to login.
3.2	Users must be authenticated before authorising access.
3.3	Users must be able to reset password if forgotten.
4.0	Users must be presented with the main screen which is the activity stopwatch.
4.1	Users must be able to start the stopwatch when an activity/task is started.
4.2	Users must be able to visualise the stopwatch running with nice clock animation.

<b>4.3</b>	Users must be able to pause the stopwatch while not working on the activity/task.
<b>4.4</b>	Users must be able to stop stopwatch once the activity/task is finished or done.
<b>4.5</b>	Users must be able to reset the stopwatch.
<b>4.6</b>	Users must be informed with a notification when the stopwatch is running in the background.
<b>5.0</b>	Users must be able to store the progress of the activity/task on the database.
<b>5.1</b>	Users must be able to create and store new activity to the database.
<b>5.2</b>	Users must be able to select existing activity/task to store new progress made.
<b>6.0</b>	User must be presented with easy to understand and simple GUI.
<b>6.1</b>	User must be able to navigate to different options through navigation bar.
<b>7.0</b>	Users must have option to view all the activities/tasks worked on in a list format.
<b>7.1</b>	Users must be presented with all the details and history of every activity/tasks.
<b>7.2</b>	Users must be able to edit the details of the activities/ tasks.
<b>7.3</b>	Users can create new activity/task from the activity details page.
<b>7.4</b>	Users must be able to delete an existing activity/ task once completed or no longer needed.
<b>8.0</b>	User must be able to navigate to chart page.
<b>8.1</b>	User must be presented with the pie charts and bar charts of the time spent on the activities/ tasks.
<b>8.2</b>	Users must be able to understands the charts easily.
<b>9.0</b>	Users must have a profile page where users' details will be displayed.
<b>9.1</b>	Users can change or update their details.
<b>9.2</b>	Users must be able to change their password.
<b>10.0</b>	Users must be able to log out of their account.

### 3 Wireframe diagrams:



### 4 Application Navigation:



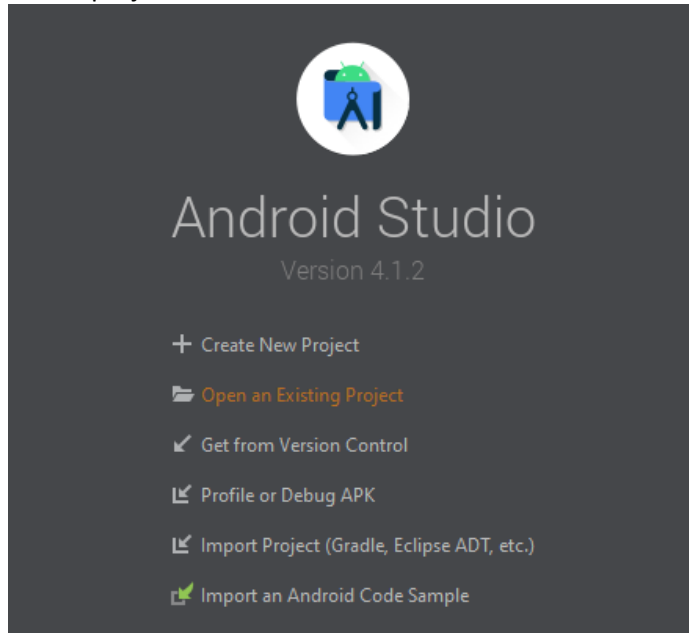
## Appendix C: Android Studio setup to run the Application.

The project application (Tractivity) can be downloaded from the OneDrive:

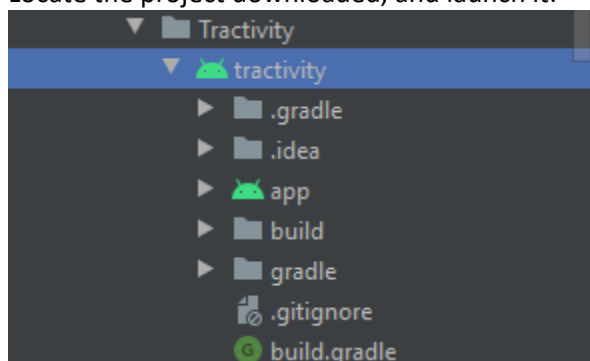
<https://1drv.ms/u/s!AjRdDyoNsYGxnnOEzHvr80-Ycei-?e=aa6hUw>

The following instruction is required to follow to compile and run the project application:

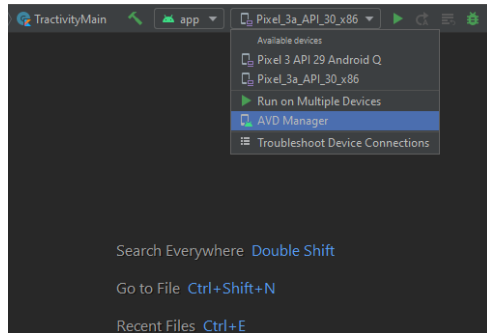
1. Install Android Studio from the following URL: <https://developer.android.com/studio>
2. Once installed, launch Android Studio and select the option to “Open an existing Android Studio project”:



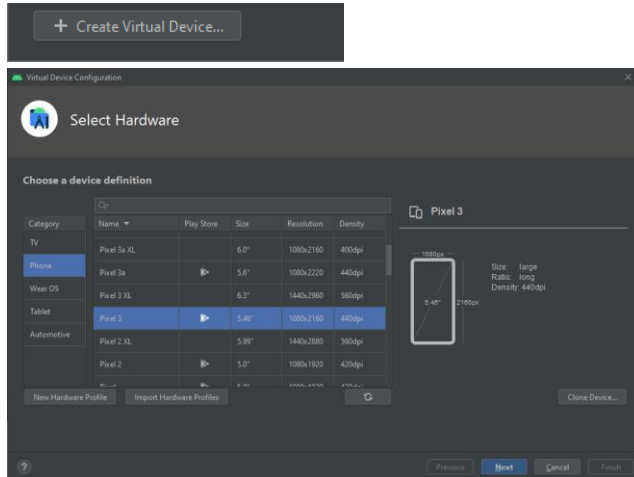
3. Locate the project downloaded, and launch it:



4. Once the project is open, an Android Emulator needs to be installed to run the application:
  - If an emulator is already installed, then proceed to step 5
  - In the Toolbar, select the ADV Manager:



- Click 'Create Virtual Device' and choose a virtual machine from the 'Phone' category. Author recommends to choose Pixel 3 or Pixel3a as its lightweight.



- Select the Emulator on the tool bar and click run (Green play button). It might take few moments as the Gradle file will build the application and run on the emulator.

To test the application login with:

Test User email: [Test1@gmail.com](mailto:Test1@gmail.com)

Password: qweqwe

## Appendix D: GUI of the application

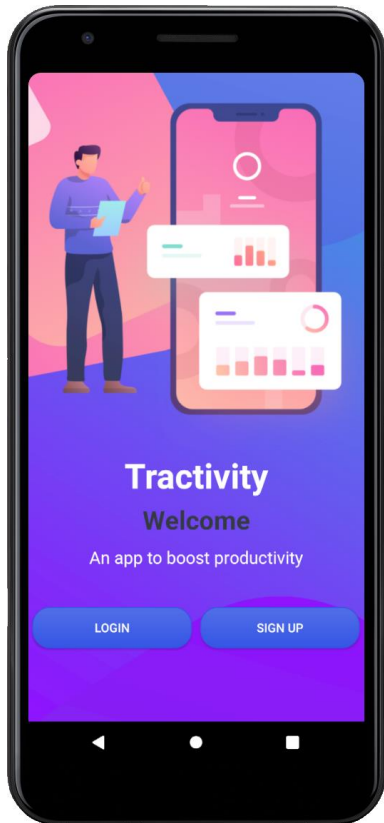


Figure 19 Intro Screen

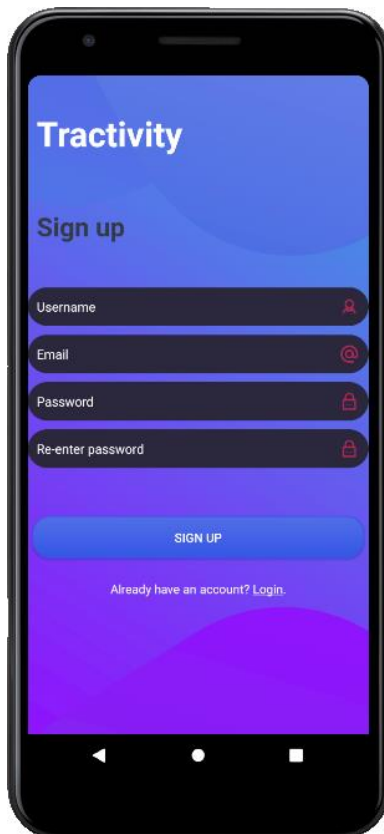


Figure 20 Sign Up Screen



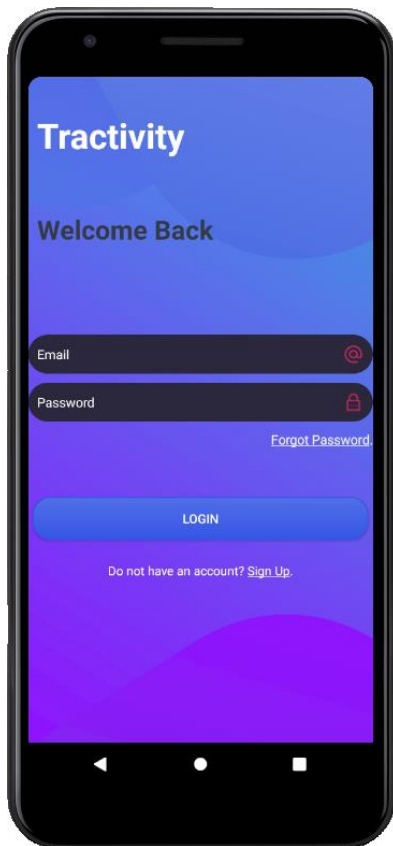


Figure 21 Login Screen

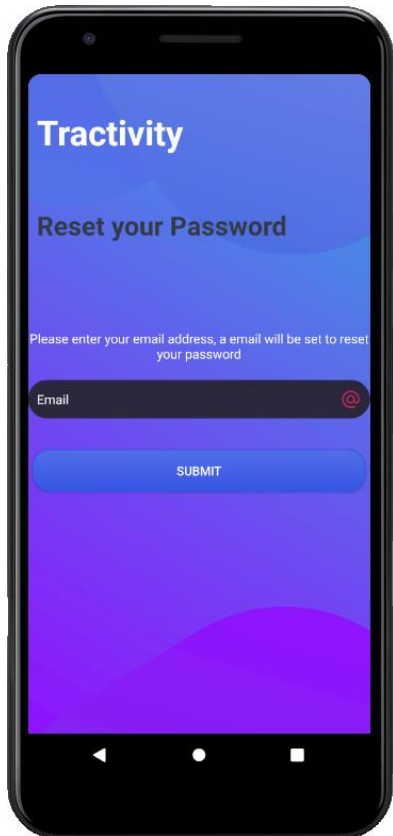


Figure 22 Password Reset Screen



Figure 23 Tractivity-Home Screen

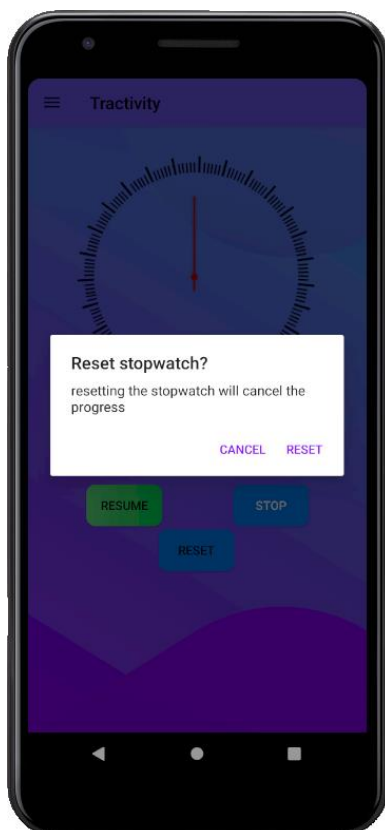


Figure 24 Stopwatch Reset Dialog

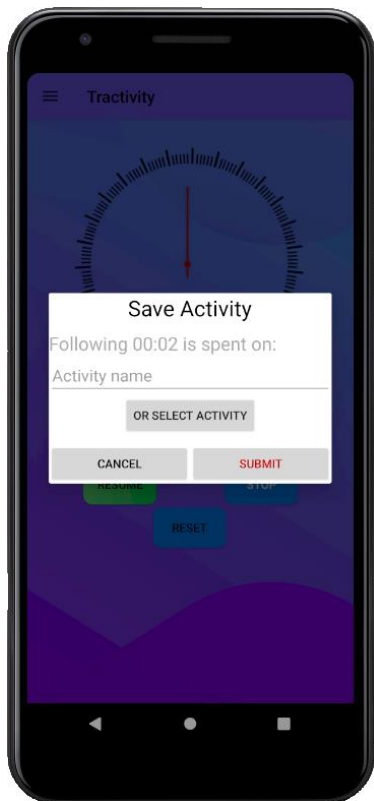


Figure 25 Save Activity Dialog

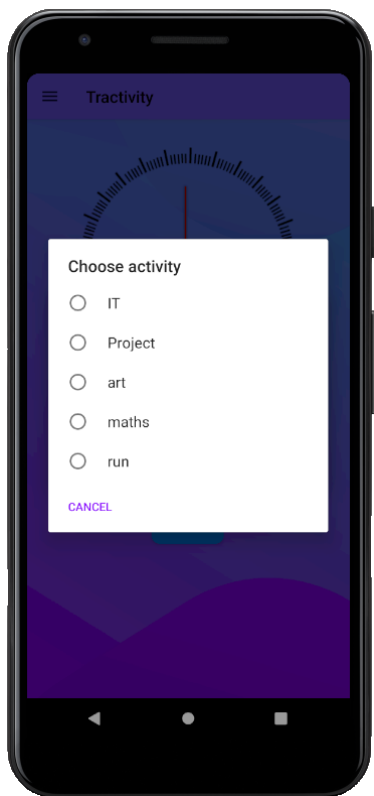


Figure 26 Activity Select Dialog

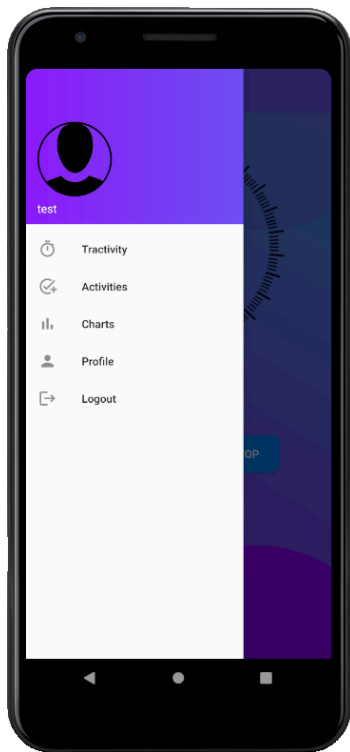


Figure 27 Navigation Drawer

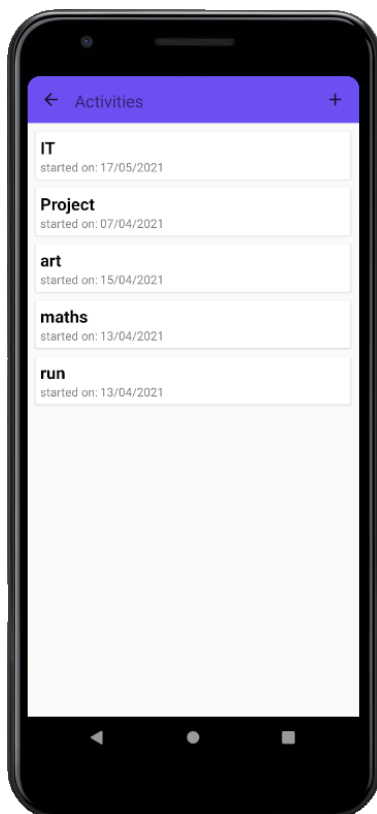


Figure 28 Activity List Screen

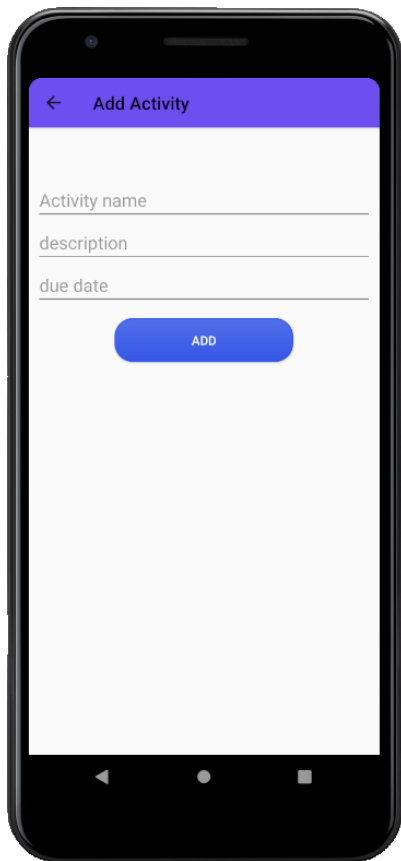


Figure 29 Create Activity

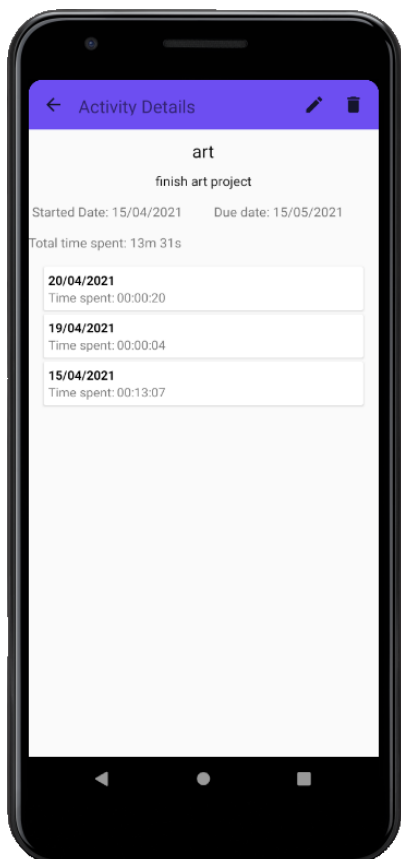


Figure 30 Activity Details

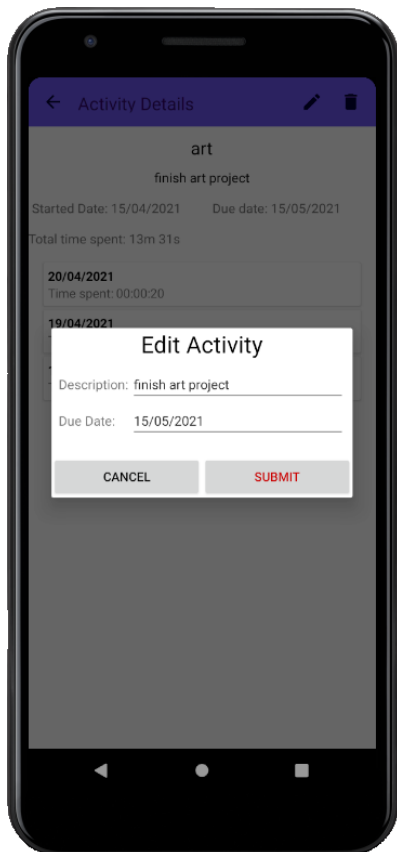


Figure 31 Edit Activity Dialog

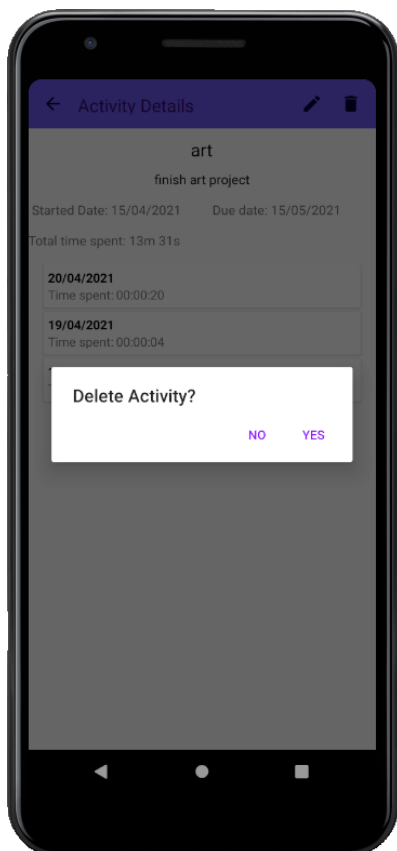


Figure 32 Delete Activity

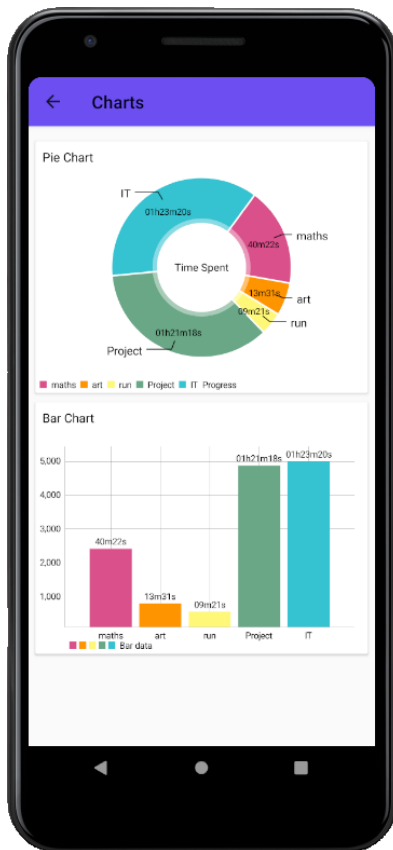


Figure 33 Charts Screen

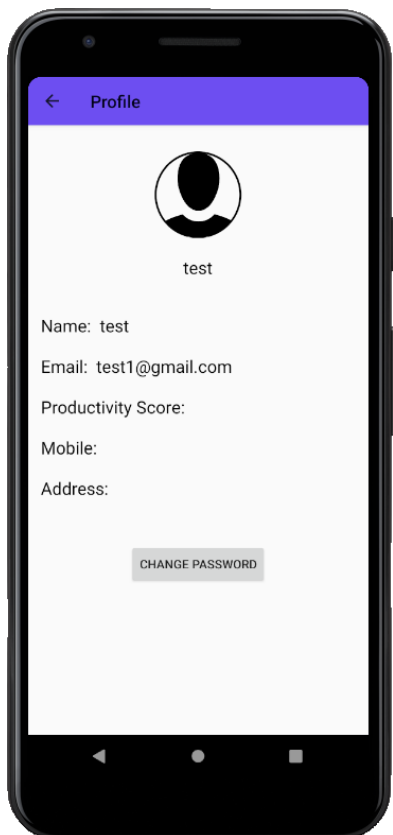


Figure 34 Profile Screen

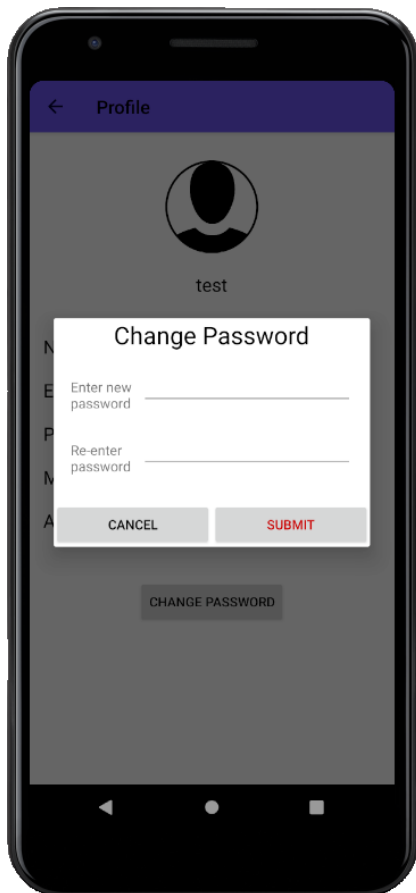


Figure 35 Change Password Dialog



## Appendix E: Implementation Document

Following document contains the source code of the Tractility application.

### 1. Click event on Start Button

```
btn_start.setOnClickListener {  
    isActivityStarted = true  
    NotificationManagerCompat.from(this).notify(NOTIFICATION_ID,buildNotification)  
    startAndPauseActivity()  
}
```

### 2. start and pause Activity

```
private fun startAndPauseActivity(){  
    if (!running) {  
        startStopWatch()  
    } else {  
        pauseStopWatch()  
    }  
}
```

### 3. Start stop watch

```
// START THE STOPWATCH  
private fun startStopWatch(){  
    iArrow.startAnimation(AnimationUtils.loadAnimation(this,  
R.anim.rotating_arrow))  
    c_chronometer.base = SystemClock.elapsedRealtime() + pauseTime  
    c_chronometer.start()  
    running = true  
    btn_start.text = "Pause"  
}
```

### 4. Pause the stopwatch

```
// PAUSE THE STOPWATCH  
private fun pauseStopWatch(){  
    iArrow.clearAnimation()  
    pauseTime = c_chronometer.base - SystemClock.elapsedRealtime()  
    c_chronometer.stop()  
    running = false  
    btn_start.text= "Resume"  
}
```

### 5. Click event on Reset button:

```
btn_reset.setOnClickListener {  
    if(isActivityStarted){  
        pauseStopWatch()  
        confirmationDialog()  
    }  
}
```

## 6. Reset stopwatch

```
private fun resetStopwatch(){  
    c_chronometer.base= SystemClock.elapsedRealtime()  
    iArrow.clearAnimation()  
    pauseTime = 0  
    running=false  
    btn_start.text="Start"  
}
```

## 7. Stop button of Stopwatch:

```
btn_stop.setOnClickListener{  
    if( isActivityStarted){  
        NotificationManagerCompat.from(this).cancel(NOTIFICATION_ID)  
        progress = c_chronometer.text.toString()  
        pauseStopWatch()  
        saveDialogFunction()  
    }else{  
        Toast.makeText(this, "Activity not started", Toast.LENGTH_LONG).show()  
    }  
}
```

## 8. confirmation dialog

```
private fun confirmationDialog(){  
    val confBuilder = AlertDialog.Builder(this)  
    confBuilder.setTitle("Reset stopwatch?")  
    confBuilder.setMessage("resetting the stopwatch will cancel the progress")  
    confBuilder.setPositiveButton("RESET") { dialog: DialogInterface, i: Int ->  
        resetStopwatch()  
        isActivityStarted=false  
        dialog.cancel()  
    }  
    confBuilder.setNegativeButton("Cancel") { dialog: DialogInterface, i: Int ->  
        dialog.cancel()  
    }  
    confBuilder.show()  
}
```

## 9. Rotating clock arrow:

```
<?xml version="1.0" encoding="utf-8"?>  
<set xmlns:android="http://schemas.android.com/apk/res/android"  
    android:shareInterpolator="@android:interpolator/linear"  
>  
    <rotate  
        xmlns:android="http://schemas.android.com/apk/res/android"  
        android:fromDegrees="0"  
        android:toDegrees="360"  
        android:pivotX="50%"  
        android:pivotY="50%"  
        android:repeatCount="infinite"  
        android:duration="1000"
```

```

    />
</set>

```

## 10. creating the notification channel

```

//creating a notification channel
private fun createNotificationChannel() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {

        val importance = NotificationManager.IMPORTANCE_DEFAULT
        val channel1 = NotificationChannel(CHANNEL_ID, CHANNEL_NAME,
importance).apply {
            //behaviour of the notification
            setSound(null,null)
        }
        // Register the channel with the system
        val notificationManager: NotificationManager =
            getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
        notificationManager.createNotificationChannel(channel1)
    }
}

```

## 11. Build Notification panel

```

// to build the notification panel
val buildNotification = NotificationCompat.Builder(this, CHANNEL_ID)
    .setSmallIcon(R.drawable.ic_stat_name)
    .setContentTitle("Tractivity")
    .setContentText("is running")
    .setNotificationSilent()
    .setContentIntent(pendingIntent)
    .setPriority(NotificationCompat.PRIORITY_DEFAULT).build()

```

## 12. Save Activity Dialog function

```

//SAVE ACTIVITY CUSTOM DIALOG
@SuppressLint("SetTextI18n")
private fun saveDialogFunction() {

    val saveDialogView =
LayoutInflater.from(this).inflate(R.layout.dialog_save,null)
    saveDialogView.tv_description.text= "Following $progress is spent on:"
    val saveDialogbuilder = AlertDialog.Builder(this)
        .setView(saveDialogView)
    val saveActivityDialog = saveDialogbuilder.show()
    saveActivityDialog.setCancelable(false) // prevent user to close the dialog by
clicking outside the dialog
    saveDialogView.bt_selectActivity.setOnClickListener {
        // ACTIVITY DIALOG populated with the activity lists from the DB
        firestore.collection(Constants.USERS).document(FireStoreClass().getCurrentUserID()
        )
            .collection(Constants.ACTIVITIES)
            .get()
            .addOnSuccessListener { activities ->

```

```

        Log.d("DB", "All documents received")
        val activityNameList = ArrayList<String>()
        for (activity in activities) {
            val activityClass: ActivityClass? =
                activity.toObject(ActivityClass::class.java)
            if (activityClass != null){
                activityNameList.add(activityClass.name)
            }
        }

        val items:Array<String> = activityNameList.toTypedArray()
        val selectActivityBuilder = AlertDialog.Builder(this)
        selectActivityBuilder.setTitle("Choose activity")
        selectActivityBuilder.setSingleChoiceItems(items,-1){
            dialogInterface: DialogInterface, i :Int ->

            saveDialogView.et_activityName.setText(items[i])
            isActivitySelected = true
            dialogInterface.dismiss()

        }
        selectActivityBuilder.setNeutralButton("cancel"){
            dialog:DialogInterface, which->
            dialog.cancel()
        }
        val activitySelectDialog = selectActivityBuilder.create()
        activitySelectDialog.show()
    }
    .addOnFailureListener { exception ->
        Log.d("DB", "Error getting documents: ", exception)
    }
}

saveDialogView.bt_submit.setOnClickListener{
    activityName = saveDialogView.et_activityName.text.toString()
    if (TextUtils.isEmpty(activityName)){
        Toast.makeText(applicationContext, "Enter or select Activity Name",
            Toast.LENGTH_SHORT).show()
    } else {
        val activity = ActivityClass(activityName)
        val record = ActivityRecordClass(parseProgress(progress!!))
        if(isActivitySelected){
            FirestoreClass().saveRecordOnDB(record,activity.name)
        } else{
            FirestoreClass().saveActivityOnDB(activity)
            FirestoreClass().saveRecordOnDB(record,activity.name)
        }
        isActivitySelected = false
        isActivitySelected = false
        resetStopwatch()
        saveActivityDialog.dismiss()
    }
}

saveDialogView.bt_cancel.setOnClickListener{
    Toast.makeText(applicationContext, "Activity not saved",
        Toast.LENGTH_SHORT).show()
}

```

```

        saveAlertDialog.dismiss()
    }
}

```

13. Login button on intro activity;

```

bt_login.setOnClickListener{
    val intent = Intent(this, LoginActivity::class.java)
    startActivity(intent)
    finish()
}

```

14. SignUp button on Intro activity:

```

bt_signUp.setOnClickListener{
    val intent = Intent(this, SignUpActivity::class.java)
    startActivity(intent)
    finish()
}

```

15. validate SignUp form:

```

private fun validateSignUp (name:String , email: String ,
                           password: String, reTypedPassword : String) :Boolean{
    when {
        TextUtils.isEmpty(name) ->{
            showError("Please enter your name")
            return false
        }
        TextUtils.isEmpty(email) ->{
            showError("Please enter your email address")
            return false
        }
        TextUtils.isEmpty(password) ->{
            showError("Please enter a password")
            return false
        }
        TextUtils.isEmpty(reTypedPassword) ->{
            showError("Please re-Type your password")
            return false
        }
        !isPasswordMatched(password, reTypedPassword) ->{
            return false
        }
        else ->{
            return true
        }
    }
}

```

#### 16. create User account:

```
private fun createUser(){
    val name: String = et_name.text.toString().trim{ it <= ' ' }
    val email: String = et_email.text.toString().trim{ it <= ' ' }
    val password : String = et_password.text.toString()
    val reTypedPassword : String = et_reTypePassword.text.toString()

    if(validateSignUp(name,email,password,reTypedPassword)){
        auth.createUserWithEmailAndPassword(email,password)
            .addOnCompleteListener(this){
                task ->
                if (task.isSuccessful) {
                    val user = UserClass(Firebase.auth.currentUser!!.uid,name,email)
                    FirestoreClass().registerUserOnDB(user)
                    Toast.makeText(this,
                        "Welcome $name to Tractivity",Toast.LENGTH_SHORT).show()
                    val intent = Intent(this, TractivityMain::class.java)
                    startActivity(intent)

                    finish()
                }else {

                    Toast.makeText(this,task.exception!!
                        .message.toString(),
                        Toast.LENGTH_LONG).show()

                }
            }
    }
}
```

#### 17. Store user on firestore

```
fun registerUserOnDB (user: UserClass){
    firestore.collection(Constants.USERS).document(getCurrentUserID())
        .set(user, SetOptions.merge()).addOnSuccessListener {
            Log.d("DDDBB", "Document saved")

        }
        .addOnFailureListener{ e->
            Log.w("DDDBB", "Error adding document")

        }
}
```

#### 18. validate login form:

```
private fun validateLogin ( email: String ,
    password: String) :Boolean{
    when {
        TextUtils.isEmpty(email) ->{
            showError("Please enter your email address")
            return false
        }
        TextUtils.isEmpty(password) ->{
            showError("Please enter your password")
            return false
        }
    }
}
```

```

        else ->{
            return true
        }
    }
}

```

## 19. error snackbar

```

fun showError(errorMessage:String){
    val snackBar=
    Snackbar.make(findViewById(android.R.id.content),errorMessage,Snackbar.LENGTH_LONG
    )
    snackBar.show()
}

```

## 20. implementing user Login:

```

private fun loginUser (){
    val email:String = et_email_login.text.toString().trim{ it <= ' ' }
    val password:String = et_password_login.text.toString()
    if(validateLogin(email,password)){
        auth.signInWithEmailAndPassword(email, password)
        .addOnCompleteListener(this) { task ->
            if (task.isSuccessful) {
                Log.d("Login", "LoginInWithEmail:success")
                val user = auth.currentUser
                Toast.makeText(this,
                    "Welcome back to Tractivity",Toast.LENGTH_SHORT).show()
                val intent = Intent(this, TractivityMain::class.java)
                startActivity(intent)
                finish()

            } else {
                // If sign in fails, display a error message to the user.
                Log.w("Login", "LogInWithEmail:failure", task.exception)
                Toast.makeText(baseContext, "Authentication failed, incorrect
                    email or password was typed",Toast.LENGTH_SHORT).show()
            }
        }
    }
}

```

## 21. reset Password:

```

private lateinit var auth: FirebaseAuth

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_password_reset)
    window.decorView.systemUiVisibility = View.SYSTEM_UI_FLAG_FULLSCREEN

    btn_resetPass.setOnClickListener{
        val email:String = et_email_login.text.toString().trim{ it <= ' ' }
        if (email.isEmpty()){
            Toast.makeText(this, "Please enter your email",

```

```

        Toast.LENGTH_SHORT).show()
    } else{
        auth.sendPasswordResetEmail(email).addOnCompleteListener{
            task ->
            if(task.isSuccessful){
                Toast.makeText(this, "Email sent, check your inbox",
                    Toast.LENGTH_SHORT).show()
                finish()
            }
        }
    }
}
}
}
}
}

```

22. parse string time to long second:

```

fun parseProgress (progress:String) : Long{
    var progressInSeconds :Long = 0
    val strArray : Array<String> = progress.split(":").toTypedArray()
    if(strArray.size<3){
        progressInSeconds += strArray[0].toLong()*60
        progressInSeconds += strArray[1].toLong()
        return progressInSeconds
    }
    progressInSeconds += strArray[0].toLong()*3600
    progressInSeconds += strArray[1].toLong()*60
    progressInSeconds += strArray[2].toLong()
    return progressInSeconds
}

```

23. Navigation Drawer:

```

override fun onNavigationItemSelected(item: MenuItem): Boolean {
    when (item.itemId){
        R.id.main_pg -> {
            drawer_layout.closeDrawer(GravityCompat.START)
            return true
        }
        R.id.activities ->{
            val intent = Intent(this, ActivitiesActivity::class.java)
            startActivity(intent)
        }
        R.id.charts ->{
            val intent = Intent(this, ChartsActivity::class.java)
            startActivity(intent)
        }
        R.id.profile ->{
            val intent = Intent(this, UserProfileActivity::class.java)
            startActivity(intent)
        }
        R.id.logout ->{
            FirebaseAuth.getInstance().signOut()
            val intent = Intent(this, IntroActivity::class.java)
            intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP or Intent.FLAG_ACTIVITY_NEW_TASK)
            startActivity(intent)
            finish()
        }
    }
    drawer_layout.closeDrawer(GravityCompat.START)
    return true
}

```



#### 24. Populate username on navigation:

```
fun populateNavUsername(){
    FirebaseFirestore.getInstance().collection(Constants.USERS)
        .document(FireStoreClass().getCurrentUserID()).get()
        .addOnSuccessListener {documentSnapshot ->
            val user = documentSnapshot.toObject<UserClass>()!!
            val navHeader = navigation.getHeaderView(0)
            val navUsername = navHeader.findViewById<TextView>(R.id.tv_username)
            navUsername.text = user.name
        }
}
```

#### 25. populate the Profile screen:

```
fun populateProfileActivity () {
    firestore.collection(Constants.USERS)
        .document(getCurrentUserID()).get()
        .addOnSuccessListener {documentSnapshot ->
            Log.d(ContentValues.TAG, "Document received")
            val loggedUser: UserClass? = documentSnapshot
                .toObject(UserClass::class.java)
            if (loggedUser != null) {
                gloggedUser = loggedUser
                tv_username_profile.text = loggedUser.name
                tv_profile_name.text=loggedUser.name
                tv_profile_email.text = loggedUser.email
                tv_score.text = ""
            }
        }
        .addOnFailureListener{
            exception ->
            Log.d(ContentValues.TAG, "get failed with ", exception)
        }
}
```

#### 26. Change User Password:

```
private fun changePass(){
    val changePassDialogView =
    LayoutInflater.from(this).inflate(R.layout.dialog_change_pass,null)
    val changePassDialogBuilder =
    AlertDialog.Builder(this).setView(changePassDialogView)
    val changePassDialog = changePassDialogBuilder.show()
    changePassDialogView.bt_submit.setOnClickListener{
        val password = changePassDialogView.et_pass.text.toString()
        val reTypedPassword = changePassDialogView.et_rePass.text.toString()
        if(isPasswordMatched(password,reTypedPassword)){
            FirebaseAuth.getInstance().currentUser!!.updatePassword(password).addOnCompleteListener{
                task ->
                if(task.isSuccessful){
                    Toast.makeText(this, "Password changed Successfully",
                        Toast.LENGTH_SHORT).show()
                    changePassDialog.dismiss()
                }
            }
        }
    }
}
```

```

        }else{
Toast.makeText(this,task.exception!!.message.toString(),Toast.LENGTH_LONG).show()
        }
    }
    }else{
        Toast.makeText(this, "Passwords do not match",
            Toast.LENGTH_SHORT).show()
    }
}
changePassDialog.bt_cancel.setOnClickListener{
    changePassDialog.cancel()
}
}
//checks if both the passwords matches
private fun isPasswordMatched (password:String, reTypedPassword:String) :Boolean{
    return password == reTypedPassword
}
}

```

## 27. Activity list adapter:

```

class ActivityAdapter (private val list : ArrayList<ActivityClass> )
:RecyclerView.Adapter<RecyclerView.ViewHolder>(){

    private var onItemClickListener : OnItemClickListener? = null

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
RecyclerView.ViewHolder {
        return ActivitiesViewHolder(LayoutInflater.from(parent.context)
            .inflate(R.layout.item_activity,parent,false))
    }

    override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position: Int) {
        val item = list[position]
        if (holder is ActivitiesViewHolder){
            holder.itemView.tv_activityName.text=item.name
            holder.itemView.tv_date.text = item.date
            holder.itemView.setOnClickListener{
                if(onItemClickListener!=null){
                    onItemClickListener!!.onItemClick(position,item)
                }
            }
        }
    }

    override fun getItemCount(): Int {
        return list.size
    }

    class ActivitiesViewHolder(itemView:View) : RecyclerView.ViewHolder(itemView){
        val textView1 : TextView = itemView.tv_activityName
        val textView2 : TextView = itemView.tv_date
    }

    fun setOnClickListener(onClickListener: OnItemClickListener){
        this.onItemClickListener = onClickListener
    }
}

```

```

interface OnItemClickListener {
    fun onItemClick(position: Int, item: ActivityClass)
}
}

```

28. Populate the recycle View of Activity list:

```

// POPULATE RECYCLEVIEW
private fun populateActivityListRv (activityList: ArrayList<ActivityClass>){
    if (activityList.size> 0){
        rv_activitiesList.layoutManager = LinearLayoutManager(this)
        rv_activitiesList.setHasFixedSize(true)

        val adapter = ActivityAdapter(activityList)
        rv_activitiesList.adapter = adapter

        adapter.setOnItemClickListener(object: ActivityAdapter.OnItemClickListener {
            override fun onItemClick(position: Int, item: ActivityClass) {
                val intent = Intent(this@ActivitiesActivity, DetailsActivity::class.java)
                intent.putExtra(Constants.ACTIVITYNAME,item.name)
                startActivity(intent)
            }
        })
    }else{
        Toast.makeText(this, "no activity stored", Toast.LENGTH_SHORT).show()
    }
}

```

29. Query Activity list from firestore and display in recycle view:

```

private fun displayActivityListFromDB(){
    firestore.collection(Constants.USERS).document(FireStoreClass().getCurrentUserID())
        .collection(Constants.ACTIVITIES)
        .get()
        .addOnSuccessListener { results ->
            Log.d("DB", "All documents received")
            val activityList = ArrayList<ActivityClass>()
            for (result in results) {
                val activity: ActivityClass? = result.toObject(ActivityClass::class.java)
                if (activity != null) {
                    activityList.add(activity)
                }
            }
            populateActivityListRv(activityList)
        }.addOnFailureListener { exception ->
            Log.d("DB", "Error getting documents: ", exception)
        }
}

```

### 30 add Activity manually

```
bt_add.setOnClickListener{
    val activityName = et_activityName.text.toString()
    val description = et_description.text.toString()
    val dueDate = et_dueDate.text.toString()
    if (TextUtils.isEmpty(activityName)) {
        Toast.makeText(
            applicationContext,
            "Please Enter Activity Name",
            Toast.LENGTH_SHORT
        ).show()
    } else {
        val activity = ActivityClass(activityName,
            SimpleDateFormat("dd/MM/yyyy", Locale.getDefault()).format(Date()),
            description, dueDate)
        FirestoreClass().saveActivityOnDB(activity)
        val intent = Intent(this, ActivitiesActivity::class.java)
        startActivity(intent)
        finish()
    }
}
```

### 31. display the activity details:

```
private fun displayActivityDetailsFromDB () {
    firestore.collection(Constants.USERS).document(FirestoreClass().getCurrentUserID())
        .collection(Constants.ACTIVITIES).document(activityName).get()
        .addOnSuccessListener { document ->
            Log.d("DB", "document received")
            val activity : ActivityClass? = document.toObject(ActivityClass::class.java)
            populateActivityDetails(activity!!)
        }
        .addOnFailureListener { exception ->
            Log.d("DB", "Error getting the document: ", exception)
        }
}
```

### 32. populate the Activity details screen:

```
private fun populateRecordsRv (recordList: ArrayList<ActivityRecordClass>) {
    if (recordList.size > 0) {
        rv_recordList.layoutManager = LinearLayoutManager(this)
        rv_recordList.setHasFixedSize(true)

        val adapter = RecordAdapter(recordList)
        rv_recordList.adapter = adapter
    }
}
```

### 33. Query and display total time spent in the activity:

```
fun displayProgress(){
    firestore.collection(Constants.USERS).document(FireStoreClass().getCurrentUserID())
        .collection(Constants.ACTIVITIES).document(activityName).collection(Constants.RECORDS).get(
        )
        .addOnSuccessListener{ documents ->
            Log.d("DB", "collection received")
            var totalTime = 0L
            for (document in documents){
                val record : ActivityRecordClass =
                    document.toObject(ActivityRecordClass::class.java)
                totalTime += record.progress
            }
            tv_progress.text= "Total time spent: ${formatProgress(totalTime)}"
        }.addOnFailureListener { exception ->
            Log.d("DB", "Error getting the document: ", exception)
        }
}
```

### 34. Formatting the total progress in string format:

```
private fun formatProgress(progressInSec : Long) : String{
    val hours = progressInSec / 3600
    val minutes = (progressInSec % 3600) / 60
    val seconds = progressInSec % 60
    return when {
        hours >0 -> {
            String.format("%02dh %02dm %02ds", hours, minutes, seconds)
        }
        minutes>0 -> {
            String.format("%02dm %02ds",minutes, seconds)
        }
        else -> {
            String.format("%02ds", seconds)
        }
    }
}
```

### 35. edit Activity details:

```
private fun displayActivityEditDialog(){
    val editDialogView = LayoutInflater.from(this).inflate(R.layout.dialog_edit_activity,null)
    firestore.collection(Constants.USERS).document(FireStoreClass().getCurrentUserID())
        .collection(Constants.ACTIVITIES).document(activityName).get()
        .addOnSuccessListener{ document ->
            Log.d("DB", "document received")
            val activity : ActivityClass? = document.toObject(ActivityClass::class.java)
            if (activity != null) {
                editDialogView.et_description.setText(activity.description)
                editDialogView.et_dueDate.setText(activity.dueDate)
                val editDialogBuilder =AlertDialog.Builder(this).setView(editDialogView)
                val editDialog = editDialogBuilder.show()
                editDialogView.bt_submit.setOnClickListener {
                    val description = editDialogView.et_description.text.toString()
                    val dueDate = editDialogView.et_dueDate.text.toString()
                    firestore.collection(Constants.USERS)
                        .document(FireStoreClass().getCurrentUserID())
                        .collection(Constants.ACTIVITIES).document(activityName).update(
                        mapOf(
                            "description" to description,
                            "dueDate" to dueDate
                        )
                    )
                }
            }
        }
}
```

```

    )
    val intent = Intent(this, ActivitiesActivity::class.java)
    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP or Intent.FLAG_ACTIVITY_NEW_TASK)
    startActivity(intent)
    finish()
    Toast.makeText(this, "Activity details changed", Toast.LENGTH_SHORT).show()
    editDialog.dismiss()
}
editDialogView.bt_cancel.setOnClickListener {
    editDialog.cancel()
}
}
}.addOnFailureListener { exception ->
    Log.d("DB", "Error getting the document: ", exception)
}
}
}

```

### 36.Delete activity & confirmation dialog:

```

private fun deleteActivity(){
    firestore.collection(Constants.USERS).document(FireStoreClass().getCurrentUserID())
        .collection(Constants.ACTIVITIES).document(activityName).delete()
        .addOnSuccessListener {
            Log.d("DATA", "Activity has been deleted")
            Toast.makeText(this,
                "Activity has been deleted", Toast.LENGTH_SHORT).show()
        }
        .addOnFailureListener { e -> Log.w("DATA", "Error deleting document", e) }
}

private fun confirmationDialog(){
    val confBuilder = AlertDialog.Builder(this)
    confBuilder.setTitle("Delete Activity?")
    confBuilder.setPositiveButton("YES") { dialog: DialogInterface, i: Int ->
        deleteActivity()
        val intent = Intent(this, ActivitiesActivity::class.java)
        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP or Intent.FLAG_ACTIVITY_NEW_TASK)
        startActivity(intent)
        finish()
        dialog.cancel()
    }
    confBuilder.setNegativeButton("NO") { dialog: DialogInterface, i: Int ->
        dialog.cancel()
    }
    confBuilder.show()
}
}

```

### 37. implementing pie Chart:

```

private fun showPieChart(){
    FireStoreClass().fireStore.collection(Constants.USERS)
        .document(FireStoreClass().getCurrentUserID())
        .collection(Constants.ACTIVITIES)
        .get()
        .addOnSuccessListener { documents ->
            Log.d("DB", "All documents received")

            val activityPieData = ArrayList<PieEntry>()

            for (document in documents) {
                var totalProgress: Long = 0
                val activity = document.toObject(ActivityClass::class.java)
                FireStoreClass().fireStore.collection(Constants.USERS)
                    .document(FireStoreClass().getCurrentUserID())
                    .collection(Constants.ACTIVITIES).document(activity.name)
                    .collection(Constants.RECORDS)
                    .get().addOnSuccessListener { results ->

```

```

        for (result in results){

            val record : ActivityRecordClass = result.toObject(ActivityRecordClass::class.java)
            totalProgress += record.progress
        }

        if (totalProgress > 0){
            activityPieData.add(PieEntry(totalProgress.toFloat(), activity.name))
        }
        createPieChart(activityPieData)
    }.addOnFailureListener { exception ->
        Log.d("DB", "Error getting records: ", exception)
    }
}

}.addOnFailureListener { exception ->
    Log.d("DB", "Error getting activities: ", exception)
}
}

private fun createPieChart(activityData: ArrayList<PieEntry>){

    var colors :ArrayList<Int> = ArrayList<Int>()
    for (color in ColorTemplate.JOYFUL_COLORS){
        colors.add(color)
    }
    for (color in ColorTemplate.COLORFUL_COLORS){
        colors.add(color)
    }
    val pieDataSet = PieDataSet(activityData,"Progress")
    pieDataSet.valueFormatter= ProgressValueFormatter()
    pieDataSet.colors = colors
    pieDataSet.valueTextSize = 10f
    pieDataSet.valueTextColor = Color.BLACK
    pieDataSet.sliceSpace= 2f
    pieDataSet.xValuePosition = PieDataSet.ValuePosition.OUTSIDE_SLICE

    val pieData = PieData(pieDataSet)
    crt_pieChart.data = pieData
    crt_pieChart.setEntryLabelColor(Color.BLACK)
    crt_pieChart.legend.isWordWrapEnabled = true
    crt_pieChart.animateXY(50, 50)
    crt_pieChart.description.isEnabled = false
    crt_pieChart.centerText="Time Spent"
    crt_pieChart.invalidate()
}
}

```

### 38. Implementing Bar chart

```

private fun showBarChart(){
    FirestoreClass().fireStore.collection(Constants.USERS).document(FirestoreClass().getCurrentUserID())
        .collection(Constants.ACTIVITIES)
        .get()
        .addOnSuccessListener { documents ->
            Log.d("DB", "All documents received")

            val activityBarEntries = ArrayList<BarEntry>()
            val activityBarLabels = ArrayList<String>()
            var index = 0f
            for (document in documents) {
                var totalProgress: Long = 0

                val activity = document.toObject(ActivityClass::class.java)
                FirestoreClass().fireStore.collection(Constants.USERS).document(FirestoreClass().getCurrentUserID())
                    .collection(Constants.ACTIVITIES).document(activity.name).collection(
                        Constants.RECORDS)
                    .get().addOnSuccessListener { results ->

                        for (result in results){

                            val record : ActivityRecordClass = result.toObject(ActivityRecordClass::class.java)
                            totalProgress += record.progress
                        }

                        if (totalProgress > 0){

```

```

                activityBarEntries.add(BarEntry(index++,totalProgress.toFloat()))
                activityBarLabels.add(activity.name)
            }
            createBarChart(activityBarEntries,activityBarLabels)
        }.addOnFailureListener { exception ->
            Log.d("DB", "Error getting records: ", exception)
        }
    }

    }.addOnFailureListener { exception ->
        Log.d("DB", "Error getting activities: ", exception)
    }
}

private fun createBarChart(activityBarEntries: ArrayList<BarEntry>, activityBarLabels : ArrayList<String>){
    var colors :ArrayList<Int> = ArrayList<Int>()
    for (color in ColorTemplate.JOYFUL_COLORS){
        colors.add(color)
    }
    for (color in ColorTemplate.COLORFUL_COLORS){
        colors.add(color)
    }
    val barDataSet = BarDataSet(activityBarEntries,"Bar data")
    barDataSet.valueTextSize=10f
    barDataSet.valueTextColor = Color.BLACK
    barDataSet.colors=colors
    barDataSet.valueFormatter= ProgressValueFormatter()

    val barData = BarData(barDataSet)
    crt_barChart.setDrawGridBackground(false)
    crt_barChart.setFitBars(true)
    crt_barChart.xAxis.valueFormatter= IndexAxisValueFormatter(activityBarLabels)
    crt_barChart.xAxis.position = XAxis.XAxisPosition.BOTTOM
    crt_barChart.description.isEnabled= false
    crt_barChart.data = barData

    crt_barChart.axisRight.isEnabled = false
    crt_barChart.invalidate()
}
}

```

#### 40. Chart value formatter:

```

private class ProgressValueFormatter : ValueFormatter(){
    override fun getFormattedValue(value:Float):String {
        return formatProgress(value.toLong())
    }
    private fun formatProgress(progressInSec : Long) : String{
        val hours = progressInSec / 3600
        val minutes = (progressInSec % 3600) / 60
        val seconds = progressInSec % 60
        return when {
            hours >0 -> {
                String.format("%02dh%02dm%02ds", hours, minutes, seconds)
            }
            minutes>0 -> {
                String.format("%02dm%02ds",minutes, seconds)
            }
            else -> {
                String.format("%02ds", seconds)
            }
        }
    }
}
}

```



## Appendix F: Test Table

### Intro screen

<b>Test no.</b>	<b>Test Description</b>	<b>Expected Result</b>	<b>Pass/Fail</b>
1	Clicking the 'LOGIN' button	The app should direct the user to the login screen.	P
2	Clicking the 'SIGN UP' button	The app should direct the user to sign up screen.	P

### Sign Up

<b>Test no.</b>	<b>Test Description</b>	<b>Expected Result</b>	<b>Pass/Fail</b>
1	Users fill up the form and click Sign up	Check if all the details are entered correctly and check if the password typed matches. If passed, users should be directed to the home screen and accounts need to be created in the firebase authentication server.	p
2	Clicking 'Already have an account? Login'	The app should direct the users to the login screen.	p
3	Clicking sign up without filling up the form	Display error.	P

### Login

<b>Test no.</b>	<b>Test Description</b>	<b>Expected Result</b>	<b>Pass/Fail</b>
1	Users enter their correct credentials and click login	The user will be authenticated and granted access to the home screen.	P
2	Users enter incorrect credentials and click login	The user must not be granted access, and an error message must be displayed.	P
3	Users click 'Do not have an account? Sign Up'	User should be directed to the Sign-up screen.	P
4	Users click 'Forgot Password'	User shall be directed to the password reset screen.	P
5	Clicking the Login button without entering the credential	Display an error.	P

## Reset Password

<b>Test no.</b>	<b>Test Description</b>	<b>Expected Result</b>	<b>Pass/Fail</b>
1	Users enter their email of the existing account and click submit	The users shall receive an email in their email inbox with instruction to reset password.	P
2	User enters a wrong email and click submit	An error message is printed	P

## Tractivity- Home screen (stopwatch)

<b>Test no.</b>	<b>Test Description</b>	<b>Expected Result</b>	<b>Pass/Fail</b>
1	Click start button	The stopwatch and the animation should be started by displaying the time passed.	P
2	click pause button	The stopwatch and the clock animation should stop.	P
3	Click resume button	The paused stopwatch should resume, and animation starts again.	P
4	Clicking Stop button after the stopwatch has been started	The save activity dialog should be displayed.	P
5	Clicking the stop button without starting the stopwatch yet	Display an error Toast	P
6	Clicking the reset button after starting the stopwatch	Display a toast to confirm the action and act upon the user decision.	P
7	Clicking the three lined icons	The navigation drawer will slide open on the left-hand side.	P

## Save Activity Dialog

<b>Test no.</b>	<b>Test Description</b>	<b>Expected Result</b>	<b>Pass/Fail</b>
1	Clicking the 'OR SELECT ACTIVITY'	Display list of the names of activities user is working on.	P
2	Clicking 'CANCEL'	Takes user back to the stopwatch.	P

3	Clicking 'SUBMIT'	Stores the progress of the activity in the database.	P
---	-------------------	--	---

### Navigation drawer

Test no.	Test Description	Expected Result	Pass/Fail
1	Clicking 'Activities'	Takes user to the activities screen where all the activities are retrieved from the fire store and displayed in list.	P
2	Clicking 'Charts'	Takes users to the charts screen that displays the pie charts and bar charts of the user's progress of different activities.	P
3	Clicking 'Profile'	Takes user to the user profile screen.	P
4	Clicking 'logout'	Logs user out of the application.	P

### Activity screen

Test no.	Test Description	Expected Result	Pass/Fail
1	Clicking an activity from the list	Take user to the details screen of that activity.	P
2	Clicking the plus icon	User will be directed to a screen where he can manually create an activity.	P

### Activity details screen

Test no.	Test Description	Expected Result	Pass/Fail
1	Click edit icon	A dialog will be displayed to edit the project description and due date.	P
2	Clicking 'SUMBIT' on Edit dialog	The edited fields should be update in the database and activity detail screen.	P

3	Clicking cancel	The edit dialog will be dismissed.	P
4	Clicking the bin icon	A confirmation dialog will be displayed asking if user was not to delete the activity, if confirmed it will be deleted from the list and from the database.	P

### Profile

<b>Test no.</b>	<b>Test Description</b>	<b>Expected Result</b>	<b>Pass/Fail</b>
1	Clicking 'CHANGE PASSWORD'	Display a dialog to change password.	P
2	Clicking 'SUMBIT' on the dialog	Checks if the typed passwords match and changes the user password on the authentication server.	P

## Appendix G: Code count

Cloc was used to count the number of total codes:

```
github.com/AlDanial/cloc v 1.90 T=3.39 s (96.3 files/s, 10086.1 lines/s)
-----
Language           files      blank      comment      code
-----
JSON                130         0          0         18151
XML                 168       2926         14        10859
Kotlin              20         268        162        1387
Bourne Shell        1           21         22         129
Gradle              3           19          8          73
DOS Batch           1           23          2          59
Java                1           1           3           8
SQL                 1           0           0           2
ProGuard            1           3          18           0
-----
SUM:                326       3261        229       30668
-----
C:\Users\ahame\OneDrive\Documents\CUL\year3\TERM2\IndividualProject\Tractivity>
```