**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

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 also highlights all the resources which were used to develop the application successfully.

**Chapter 1: Introduction**

* 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 is 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 justify if they were spending their time wisely, also make adjustment to their routine. If the users had ongoing projects, then he could see if they were spending enough time on it.

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 tracking 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, I mentioned there I will be using Charts.kt library for implementing charts functionality but in my 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 documentation is available for it.

* 1. 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. 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 lots 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. |
| User data needs to be stored on the cloud database 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. |
| 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 manage and update their details |
| 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. |

* 1. 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 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.
  1. work performed

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

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

* 1. Assumptions

It was assumed that the project can be quite challenging since I had not experience in android development and Kotlin programming language, nor was I 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 my progress.

PDD

**Chapter 2: Outputs Summary**

**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 Javacript (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 I felt it will also broaden my job opportunities having skills in an extra programming language. Several online resources or books will be used to learn and 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 in my 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, I have decided to proceed with Firestore as my backend and database.

The Firebase website has complete guide and documentation with video explanation on how to get started and understand the data structure and queries, I will be using those recourses to implement my database.

3.4 Charts

In order to implement the view chart functionality, I will use open-source library available for android. Some of the libraries that were found during my 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 I am familiar is 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 my project since the application needs to be completed I short time and lots of changes are expected during each iteration as I am building the application while I am 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 mine but many of them are very poorly made, overloaded with too many confusing functionalities or very impractical, which is why I am planning to make my 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 introduces and the result of these process will be further be explained in 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 4 iterations, then each of the 4 iterations goes through a 4 flexible stages: planning, designing, developing, and testing. It was also easier to make amendment withing 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**

The work plan was created in very early stage of the project and there had been some changes in requirements and functionalities though out the project development, it resulted in few changes the to time scale and iteration orders. The updated Gantt chart can be found underneath the original one on **APPENDIX.**

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.**

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 import 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.**

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. I 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.**

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, I was simple 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.**

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 from end GUI designing and backend codding 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 could pause the stopwatch when they were not working on their activity and could resume. Once they had finished their activity, they could click the stop button and it would prompt the users to select on what activity the time was spent on (which will later be stored in database). User could 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 was 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

A notification is a message that Android displays outside your app's UI to provide the user with reminders or other timely information from your app. 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 could end activity and save the progress on the database. When the users end the activity, a pop up (dialog) window would 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 is 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 would 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 contained 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. The final test checklist table was created which is available in the **APPENDIX**.

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.

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 4 iterations, each of them containing planning, designing, developing, and testing. The project was developed successfully in 5 phases:

|  |  |
| --- | --- |
| Phases | Work done |
| Phase 1 | Preparing and designing the Application. Two weeks was used to learn the Kotlin programming language and android SDK |
| Phase 2 | It involved implementing the 1st iteration of the application which was the stopwatch. It was planned, designed, implemented, and tested once all the functionalities were achieved |
| Phase 3 | 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. |
| Phase 4 | 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. |
| Phase 5 | This phase involved iteration 4 which was 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.

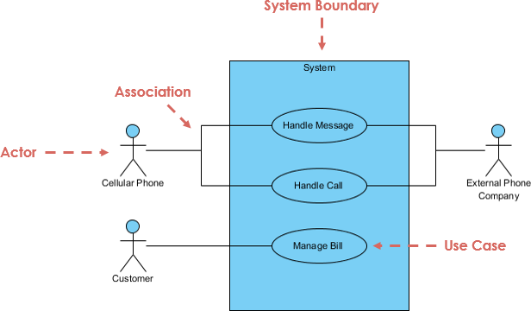
5.3 Design

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.

A use case diagram is consisting of:



* Boundary of System: it is defined as the entire system with all the functionalities.
* Use case: the functions that the system provides.
* Actor: someone that interacts with the use cases (system functions).
* Communication link: the participation of an actor in a use case.
* Use case Relationship: indicated the relationships between the use cases, include is used to show that a use case needs to include functionality of another use case and extend is used to show that a use case might or can use the functionality of another use case.

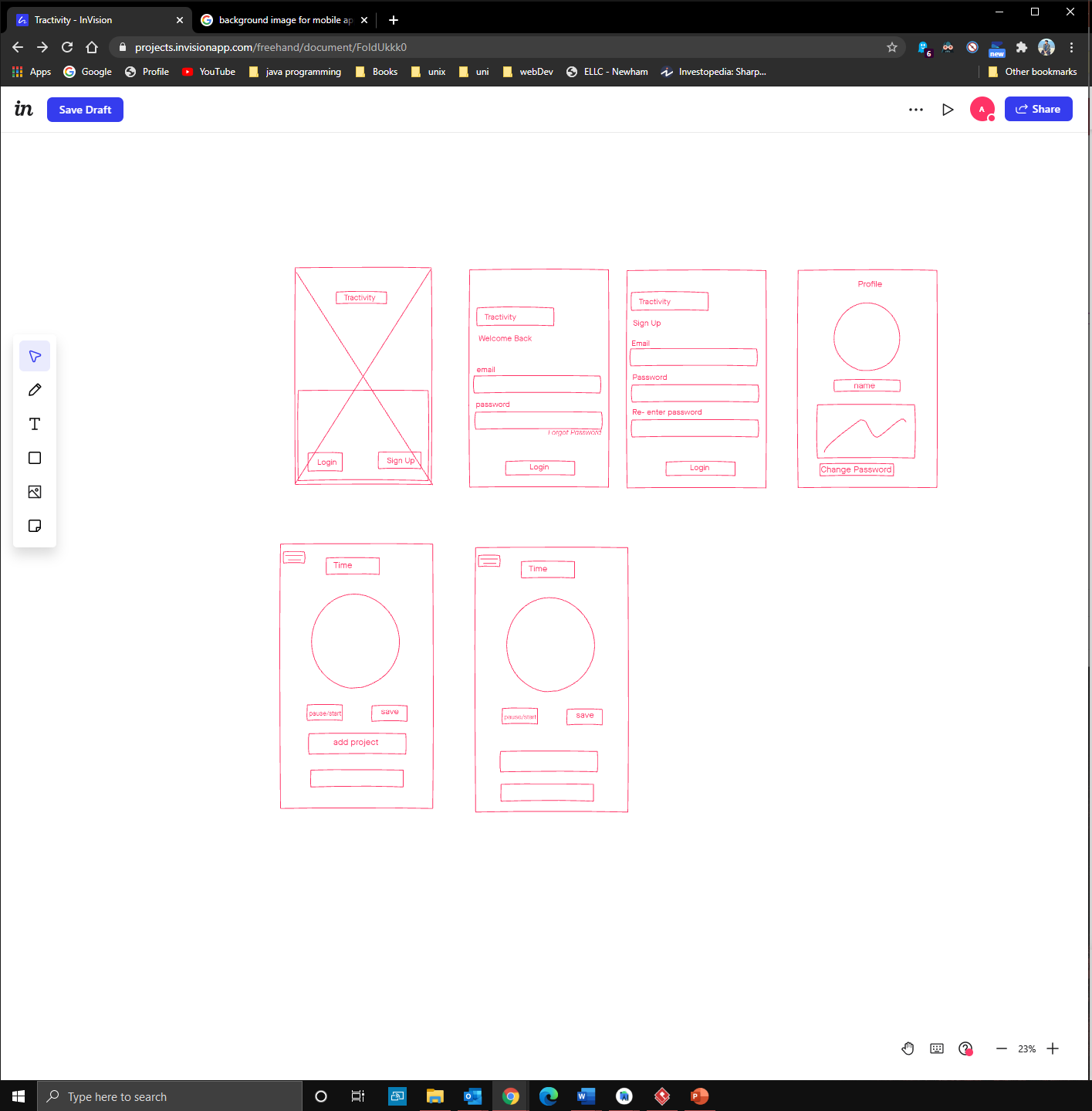
Throughout the project lifecycle 6 different versions of the use case diagrams were created. In total there are 30 use cases and 2 actors (user as primary actor and Firebase as secondary) in the final version of the use case diagram which is available in **appendix .**

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 .**

5.3.3 GUI - Wireframe

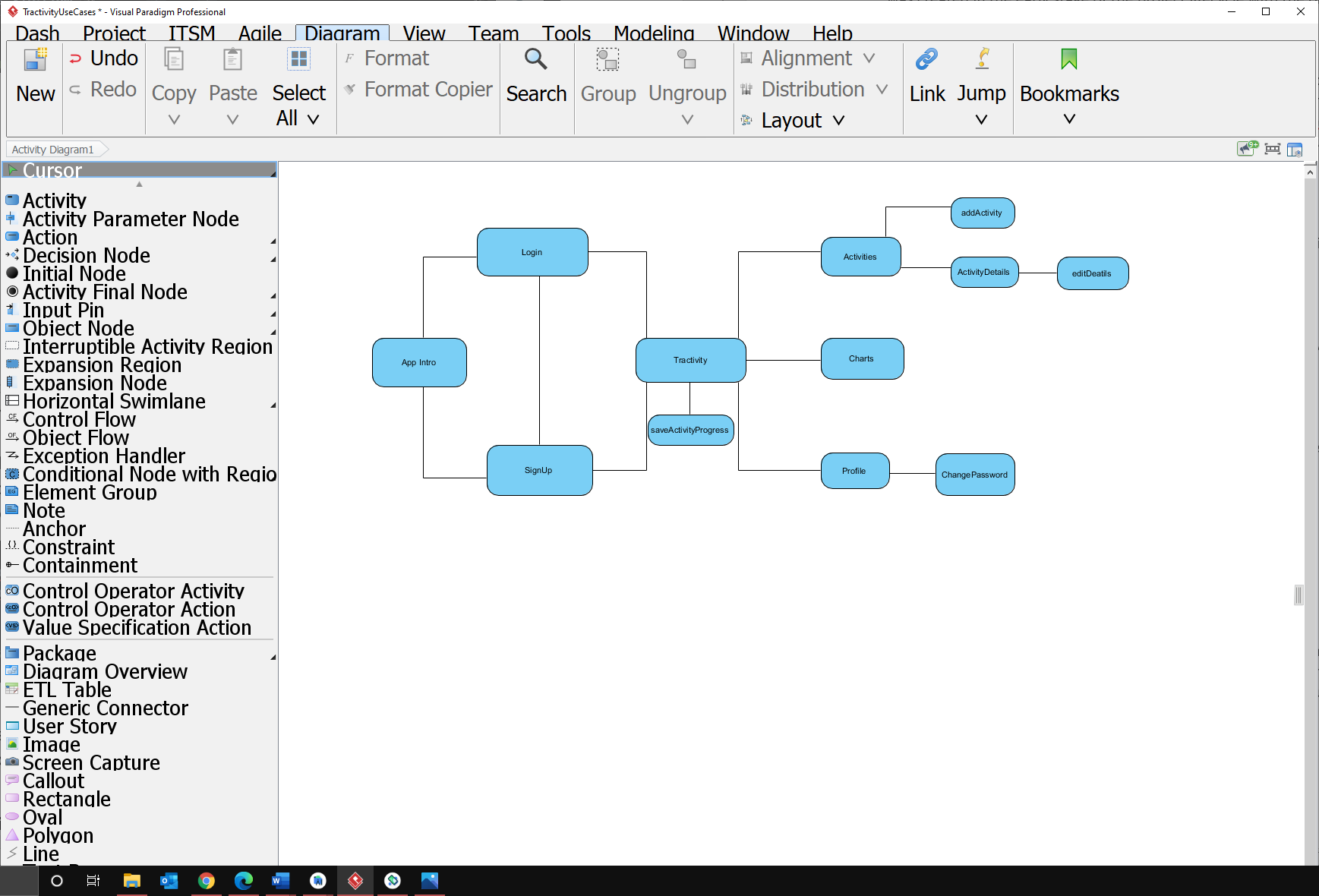


To design the User Interface of the application, a draft of the GUI design was created in the early stage of the project lifecycle with the online tool called Invision. Wireframe diagrams are made with simple lines and shape presenting the struct of the GUI, an example of Sign-up screen:

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.

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:



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.

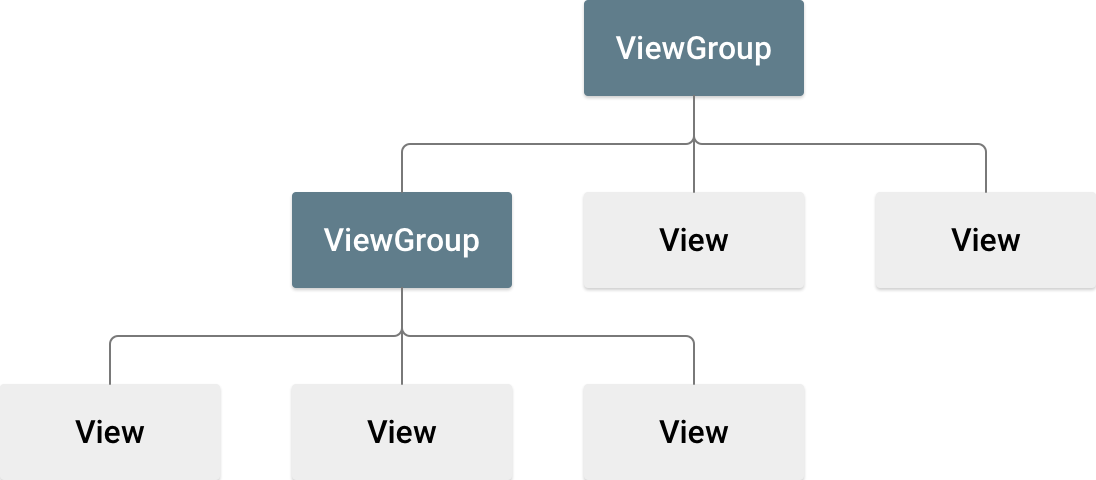
5.4.1 Development Environment and Programming Language

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.

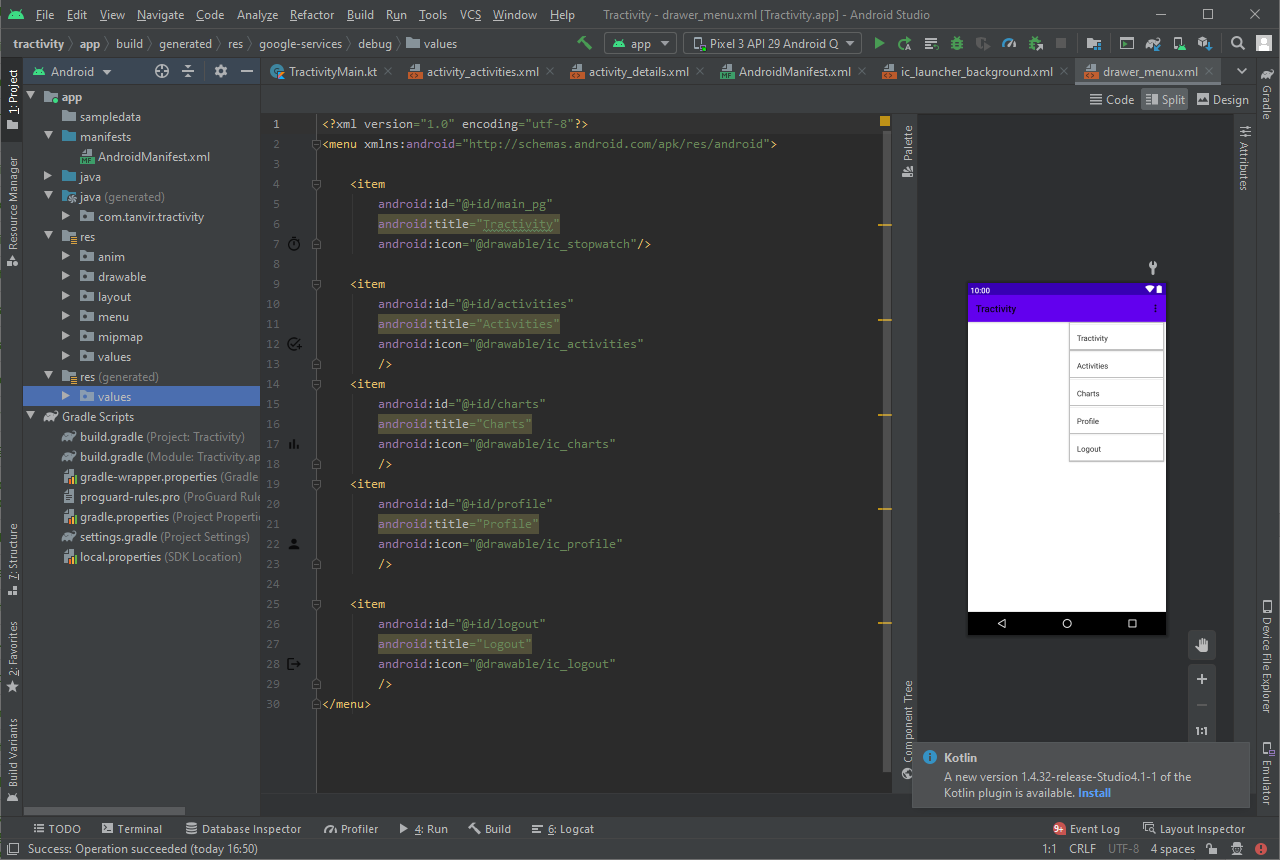
An Android emulator was installed to run the Application.

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:



(developers,2021)

Several XML files were used to create the application interface. All the UI codes were hard codded 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 is 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 varies 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.

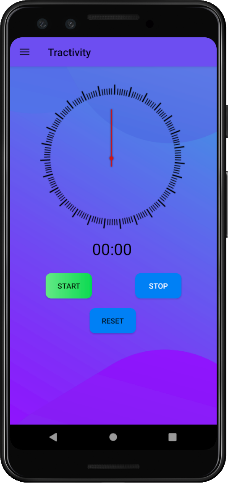
5.4.3 Version control

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 produces. 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 codded 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.

5.4.4.2 implementing start and pause stopwatch

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"  
}

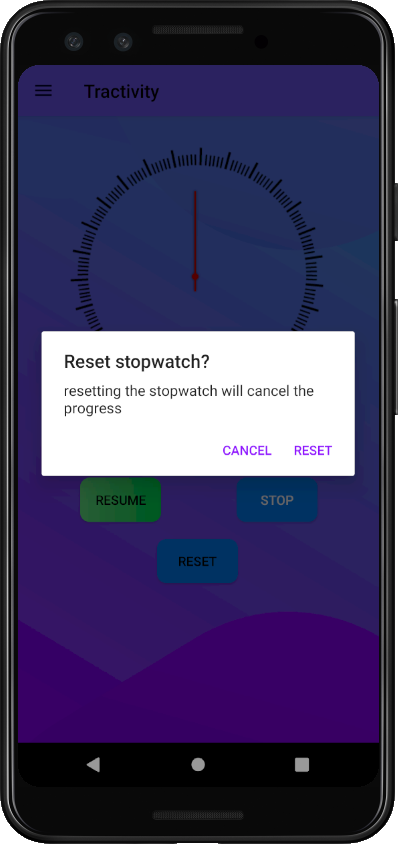
If the pause button (start button changed to pause) is pressed the stopwatch is paused, the animation is stopped. But the chronometer never stops after it has started once which means stop method does not actually stop the stopwatch, it just stops displaying in the text view and the chronometer still 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 wished to cancel the time being recorded of an activity and did not want to save it then they could 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 .**

**5.4.4.4** Implementing- Rotating clock animation:

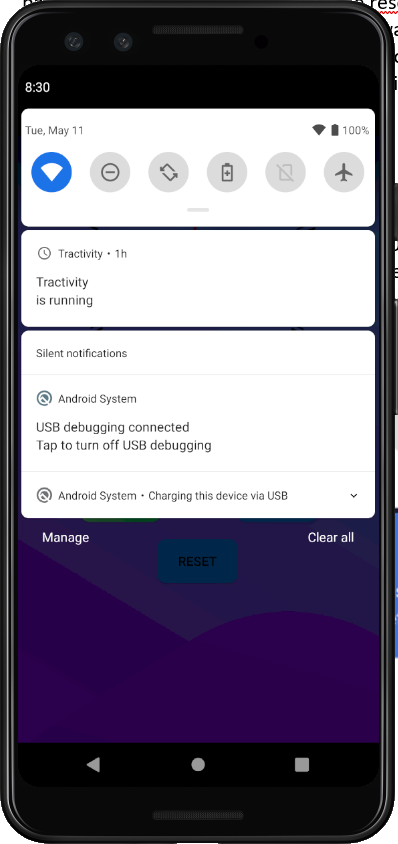
The rotation clock is an animation implemented to visualize when the stopwatch is running. A clock arrow png was taken from online free resources. First it was implemented as image view in the layout xml file. The Tweened animation system was used from animation package to implement the rotation of the 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,** 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** .

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

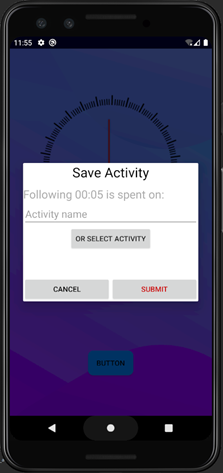
val buildNotification = NotificationCompat.Builder(this, CHANNEL\_ID)  
 .setSmallIcon(R.drawable.*ic\_stat\_name*)  
 .setContentTitle("Tractivity")  
 .setContentText("is running")  
 .setNotificationSilent()  
 .setContentIntent(pendingIntent)  
 .setPriority(NotificationCompat.*PRIORITY\_DEFAULT*).build()

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.

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)

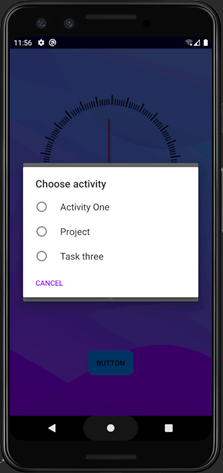
.setView(saveDialogView)

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 created 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 would 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 was changed and updated in the later iterations, the updated code snippet is available in the **Appendix** . Clicking cancel would simply take back to the stopwatch where users could continue with their activity.

5.4.4.7 Testing

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.

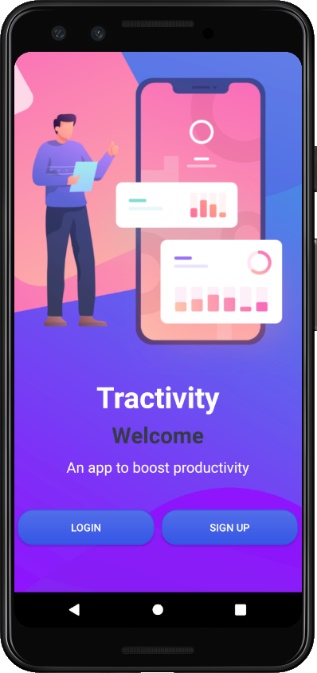
There were two ways to store data on the firestore, one was to create a HashMap of data and pass to the set method or the other way was defining a data class that can hold the data and passing the instance of that data class to the set method. The second method was preferable to the author.

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



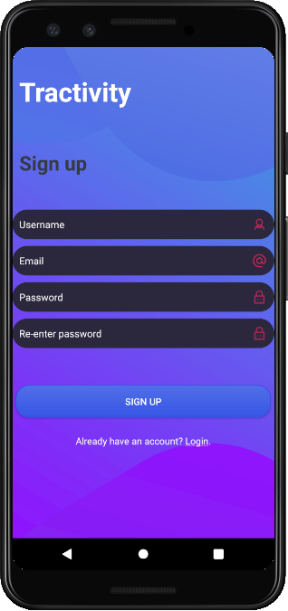
The Application intro was the first screen that users needed to interact with once they had installed the application. This screen displayed the application name with brief description and buttons to login or Sign Up with a nice background theme. The UI design was implemented in the activity\_intro.xml file **APPENDIX** . 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()  
**}**

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**.

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 .** 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:

private fun isPasswordMatched (password:String, reTypedPassword:String) :Boolean{  
 return if(password.*equals*(reTypedPassword,false)){  
 true  
 }else{  
 showError("Password does not match, please retype")  
 false  
 }

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** 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.

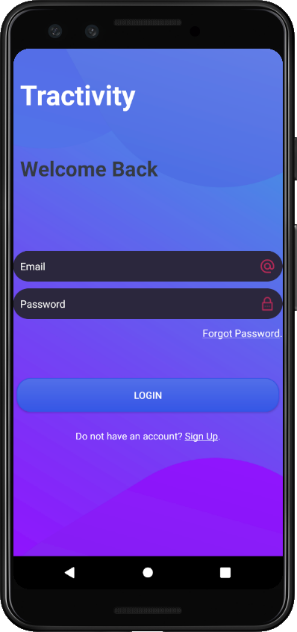
5.4.5.4 Store user Info in the Firestore

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 resisterUserOnDB 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 could login to their account by entering their email address and password. ~~The UI layouts and screen elements were designed in the activity\_login.xml file and the coding on LoginActivity.kt file.~~

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**) . 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** .

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 instructors on resetting the password, the new password set by the user will then be updated in the backend server. The **appendix** contains the UI design and **appendix** contain the logic code.

Save Activity Progress

Users must be able to store the activity progress recorded through the stopwatch. As discussed in the **section**  , the UI and the structure was implemented in iteration 1 but tin 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**  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")  
 **}**}

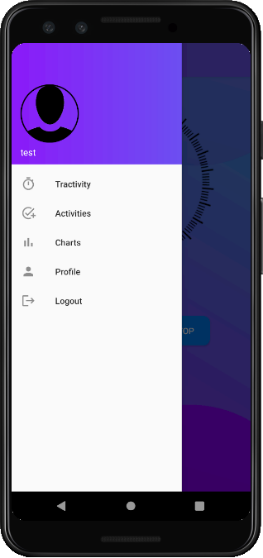
Parsing the record data

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 .**

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 setupActioBar.

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.

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 onNavigationItemSeleceted method of the build in Navigation class, code is found in the **appendix .** setNavigationItemSelectedListener

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*)  
 }  
}

Populate Navigation header

The navigation header was updated according to the user logged in, as shown **in Appendix** . 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  
}

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 it, if he is then he will be taken straight to the main screen, if not then the user would be prompted to login.

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 data from the firestore and processing it to present understandable charts. Each time any data was retrieved it was converted to its corresponding data object class for better handing, 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.

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 user was needed to se retrieved from the authentication server and use that ID to find and retrieve the user data document from the ‘Users’ collection. The ID was retrieved using the getCurrentUserID method defined in the firetore class. Then a function called populateProfileActivity was implemented to populated the profile screen with the retrieved data. The code can be found in the **appendix**  and the UI in **Appendix .**