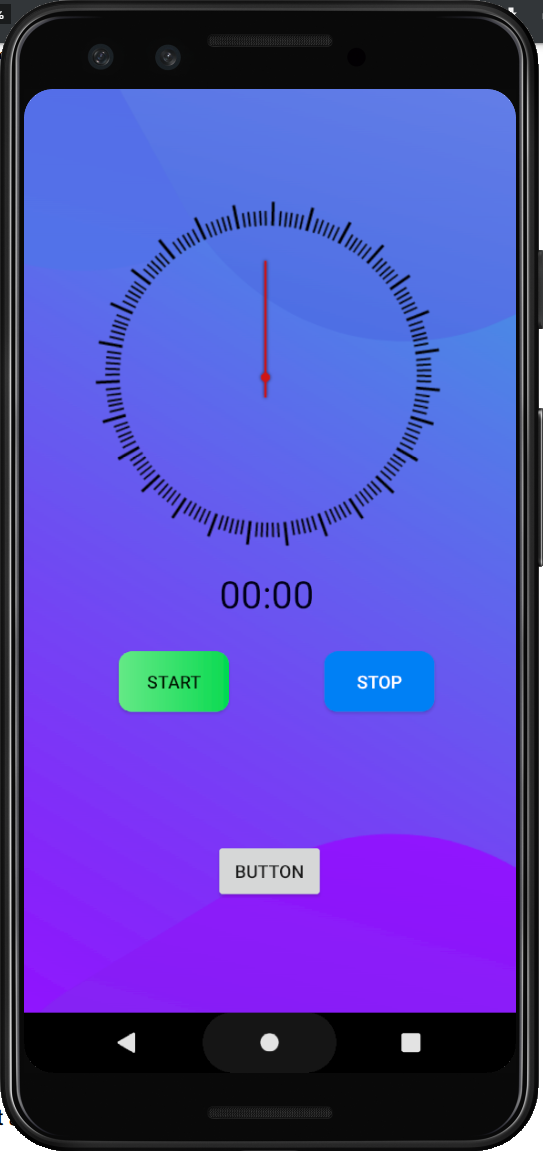
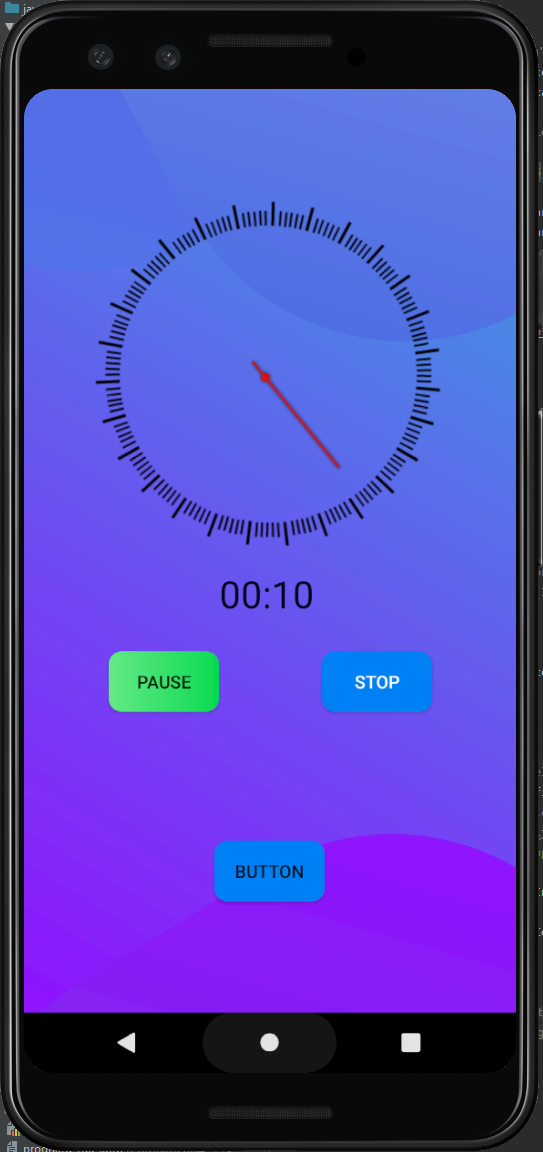
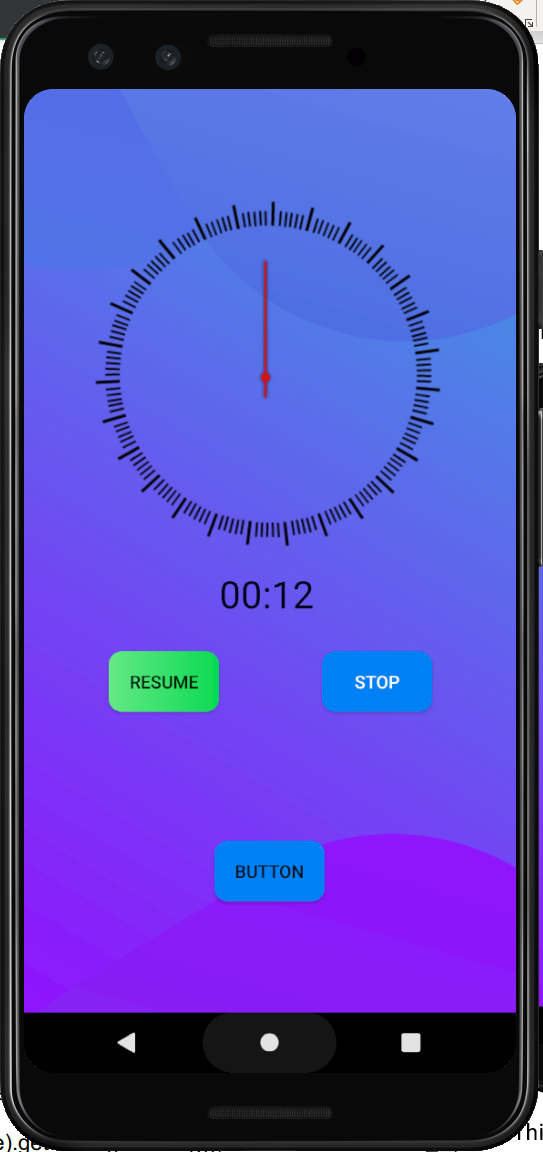
**Iteration 1**

**Method:**

**Implementing the stopwatch**

This is the initial page after the user logs in to their account. Here the user will click the start button when they start their activity, and the app will start counting every second like a stopwatch. The user can pause the timer when they are not working on their activity and resume. Once they finish their activity they can click the stop button and it will prompt the users to select on what activity the time was spent on (which will later be stored in 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, 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.

**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).

**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 is done with the help of NotificationCompat APIs from the android support library (Android Developers).

**Implementing stop and save activity:**

User can end activity and save the progress on the Firestore database. When the users end the activity, a pop up (dialog) window is 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.

**Result:**

**Implementing Start and pause activity:**

Users click the start button when they start their activity. Once the start button is pressed a notification is sent 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

if (!running) {  
 iArrow.startAnimation(AnimationUtils.loadAnimation(this,R.anim.*rotating\_arrow*))  
 c\_chronometer.*base* = SystemClock.elapsedRealtime() + pauseTime  
 c\_chronometer.start()  
 running = true  
 btn\_start.*text* = "Pause"

when the pause button (start button changed to pause) is pressed the stopwatch is paused, the animation is stopped. 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:

} else {  
 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) is pressed which will re set the base value of the chronometer to re-count from the current time and the pause time is added to the base.

**Rotating clock arrow animation:**

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 in used from animation package to implement the rotation of the arrow. The animation properties were implemented in res/anim/rotating\_arrow.xml file:

<?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>

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.

**Implementing the Notification**

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() :

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)  
 }  
}

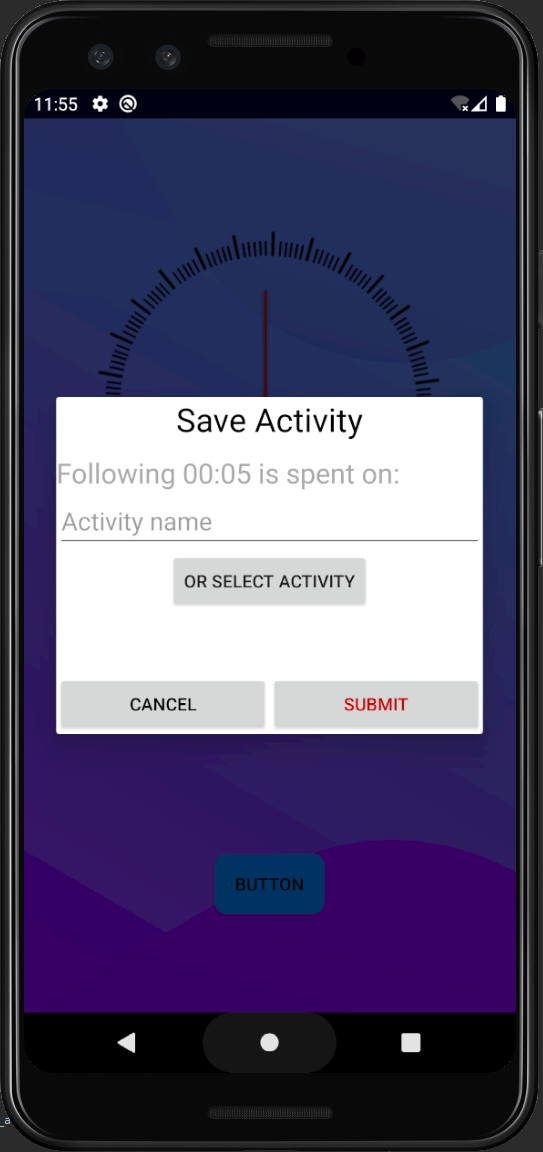
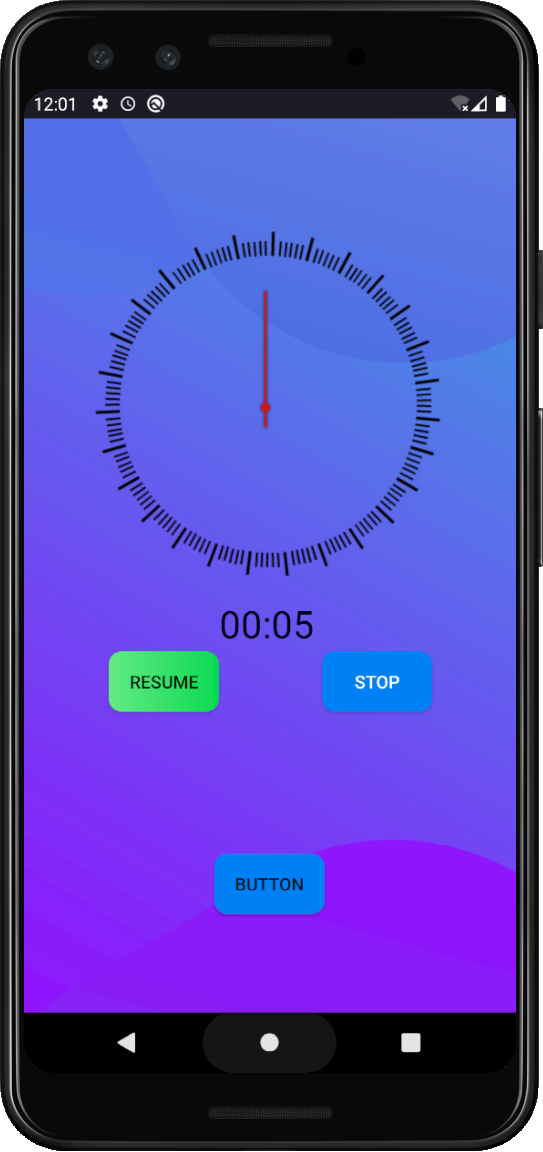
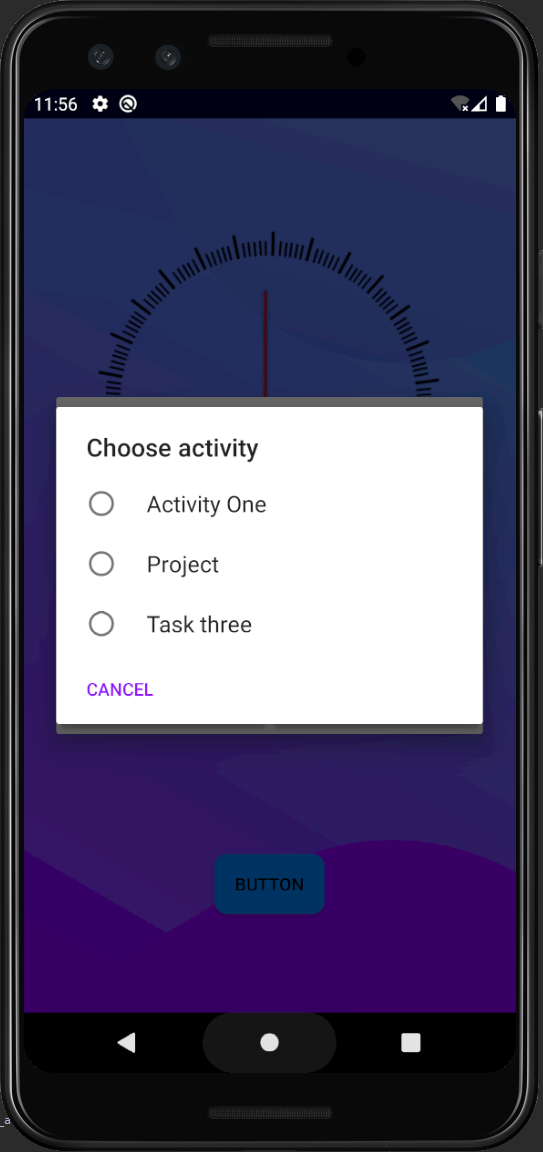
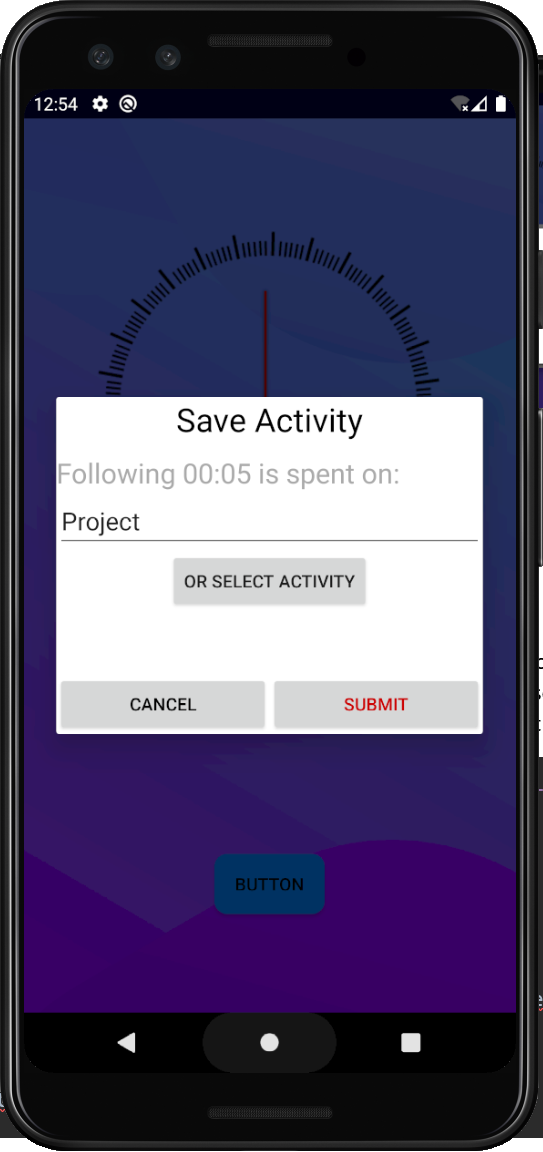
Following code snippet shows that the notification content was then set by NotificationCompat.Builder class which requires the notification CHANNEL\_ID as constructor:

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*)

**Implementing stop and save activity:**

Once the user ends an activity he can click the stop the button which will stop the stopwatch and alert dialog will be displayed to enter the activity name or choose from the existing ones. The activity name and the amount spent will later be stored in the database (will be implemented in later iteration). On submission the stopwatch in reset. Code snippet:

btn\_stop.setOnClickListener**{** NotificationManagerCompat.from(this).cancel(NOTIFICATION\_ID)  
 progress = c\_chronometer.*text*.toString()  
 c\_chronometer.stop()  
 saveDialogFunction()  
 resetStopwatch()  
**}**

private fun resetStopwatch(){  
 c\_chronometer.*base*= SystemClock.elapsedRealtime()  
 iArrow.clearAnimation()  
 pauseTime = 0  
 running=false  
 btn\_start.*text*="Start"  
}

**Implementing Pop Up Custom Alert 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 user 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 object is 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 will be updated with the name selected.

Reference:

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]

Resources used: