**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()  
 saveActivityDialog.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 snakbar

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) {  
 gbloggedUser = 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.setOnclickListener(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)  
 }  
 }  
 }  
}