Hanna M. Zelis
hannazelis@gmail.com

November 29, 2023

# Polar Verity Mobile App Development Research

Content of this project PDF:
This pdf contains the content of the three main files I worked on during my research as a mobile app developer. Using Kotlin language in Android Studio, I developed the foundation of an app that will be used in research to collect data on baseball pitchers, which will then be processed and analyzed in R Studio to allow for coaches to generate a recovery and rest plan for pitchers to help in reducing shoulder injury. In the future, we hope to be able to apply this research to other overhead motion sports. This app is tested using a Motorola phone.

Link to video of the app working: App Results - YouTube

Content of the pdf:

Hanna M. Zelis
hannazelis@gmail.com

November 29, 2023

# Polar Verity Mobile App Development Research

## Results

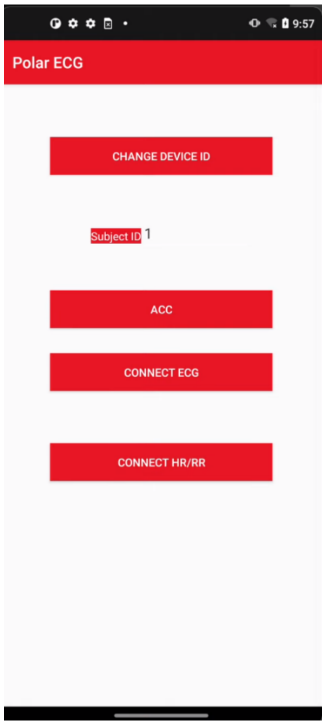


Figure 1

When the app is opened on the phone, this is the opening page. The "1" is not automatically typed in when the app is opened. However, the 1 is there to show that the user can type in a subject ID number, which will become the name of the file that is dynamically created with that session's data. To connect via Bluetooth to the phone and start running the app, click on "ACC" button.

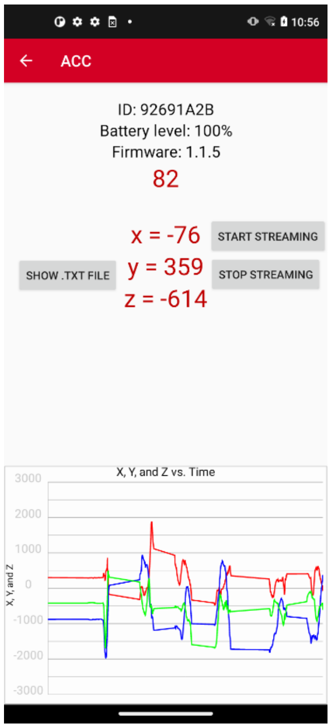
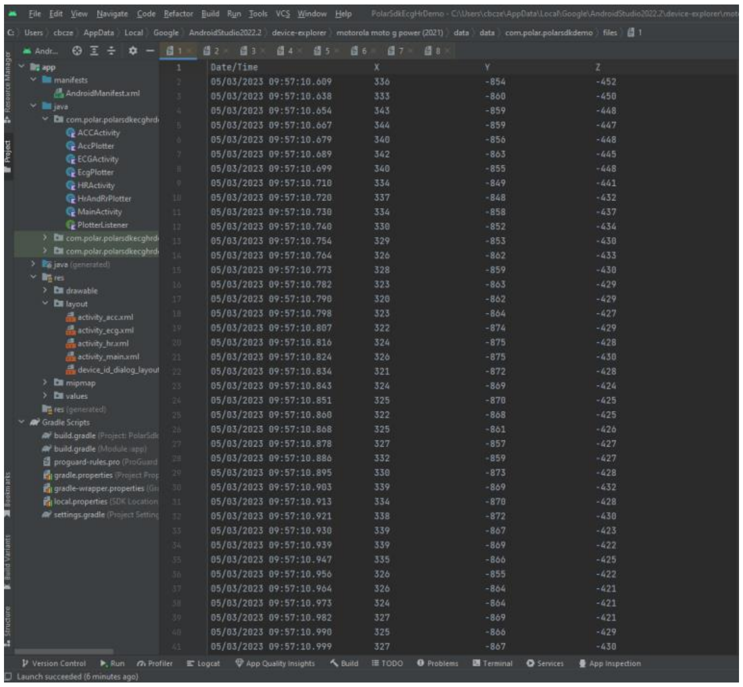


Figure 2

Once the data streaming starts, there will appear this screen, where the big number at the top, "82", is the current heart rate of the subject wearing the Polar Verity device. The x, y, and z data plus the graph display the accelerometer data in real-time at 1 data point per second. Behind the scenes, there are roughly 50-75 data points collected every second. Finally, there are three buttons. The first two on the righthand side start and stop the streaming. The third button on the lefthand side will display the data that is within the subject's subjectID file.



```
Date/Time                 X       Y       Z
05/03/2023 09:57:10.609   336     -854    -452
05/03/2023 09:57:10.638   333     -860    -450
05/03/2023 09:57:10.654   343     -859    -448
05/03/2023 09:57:10.667   344     -859    -447
05/03/2023 09:57:10.679   340     -856    -448
05/03/2023 09:57:10.689   342     -863    -445
05/03/2023 09:57:10.699   340     -855    -448
05/03/2023 09:57:10.710   334     -849    -441
05/03/2023 09:57:10.720   337     -848    -432
05/03/2023 09:57:10.730   334     -858    -437
05/03/2023 09:57:10.740   330     -852    -434
05/03/2023 09:57:10.754   329     -853    -430
05/03/2023 09:57:10.764   326     -862    -433
05/03/2023 09:57:10.773   328     -859    -430
05/03/2023 09:57:10.782   323     -863    -429
05/03/2023 09:57:10.790   320     -862    -429
05/03/2023 09:57:10.798   323     -864    -427
05/03/2023 09:57:10.807   322     -874    -429
05/03/2023 09:57:10.816   324     -875    -428
05/03/2023 09:57:10.824   326     -875    -430
05/03/2023 09:57:10.834   321     -872    -428
05/03/2023 09:57:10.843   324     -869    -424
05/03/2023 09:57:10.851   325     -870    -425
05/03/2023 09:57:10.860   322     -868    -425
05/03/2023 09:57:10.868   325     -861    -426
05/03/2023 09:57:10.878   327     -857    -427
05/03/2023 09:57:10.886   332     -859    -427
05/03/2023 09:57:10.895   330     -873    -428
05/03/2023 09:57:10.903   339     -869    -432
05/03/2023 09:57:10.913   334     -870    -428
05/03/2023 09:57:10.921   338     -872    -430
05/03/2023 09:57:10.930   339     -867    -423
05/03/2023 09:57:10.939   339     -869    -422
05/03/2023 09:57:10.947   335     -866    -425
05/03/2023 09:57:10.956   326     -855    -422
05/03/2023 09:57:10.964   326     -864    -421
05/03/2023 09:57:10.973   324     -864    -421
05/03/2023 09:57:10.982   327     -869    -421
05/03/2023 09:57:10.990   325     -866    -429
05/03/2023 09:57:10.999   327     -867    -430
```

Figure 3

Within Android Studio, the subject ID files can be opened up to see the data. Each subjectID file contains columns, from left to right, of the date/time, x, y, and z data. The format was specifically coded this way to ensure easy processing of data when transferred into R Studio.

Hanna M. Zelis
hannazelis@gmail.com

November 29, 2023

Polar Verity Mobile App Development Research

```
//Name: Hanna Zelis
//File: ACCActivity.kt
//Date: 4/21/2023
//Description: This is the accelerometer activity file from my Polar
Verity research

package com.polar.polarsdkecghrdemo

import android.content.Context
import android.content.Intent
import android.os.Bundle
import android.util.Log
import android.view.View
import android.widget.Button
import android.widget.EditText
import android.widget.TextView
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.androidplot.xy.BoundaryMode
import com.androidplot.xy.StepMode
import com.androidplot.xy.XYPlot
import com.androidplot.Plot
import com.androidplot.xy.*
import com.polar.sdk.api.PolarBleApi
import com.polar.sdk.api.PolarBleApiCallback
import com.polar.sdk.api.PolarBleApiDefaultImpl.defaultImplementation
import com.polar.sdk.api.errors.PolarInvalidArgument
import com.polar.sdk.api.model.PolarDeviceInfo
import com.polar.sdk.api.model.PolarHrData
import com.polar.sdk.api.model.PolarEcgData
import com.polar.sdk.api.model.PolarSensorSetting
import com.polar.sdk.api.PolarBleApi.PolarDeviceDataType
import com.polar.sdk.api.PolarBleApi.PolarBleSdkFeature
import com.polar.sdk.api.model.*
import io.reactivex.rxjava3.android.schedulers.AndroidSchedulers
import io.reactivex.rxjava3.disposables.Disposable
import java.text.DecimalFormat
import java.util.*
import com.androidplot.xy.XYGraphWidget
import com.polar.sdk.api.PolarBleApiDefaultImpl
import com.polar.sdk.api.PolarBleApiDefaultImpl.defaultImplementation
import java.io.File
import java.io.InputStreamReader
import java.io.OutputStreamWriter
import java.text.SimpleDateFormat
import java.time.format.DateTimeParseException
import java.util.*
```

```
//HR: import com.androidplot.xy.XYGraphWidget
//import com.polar.sdk.api.model.PolarHrData
//import java.text.DecimalFormat

//ECG: import com.polar.sdk.api.model.PolarSensorSetting
//import io.reactivex.rxjava3.android.schedulers.AndroidSchedulers
//import io.reactivex.rxjava3.disposables.Disposable



//create a stop buttton to stop recording data
//create a save button to save the data in the .txt file





class ACCActivity : AppCompatActivity(), PlotterListener {
    companion object {
        private const val TAG = "ACCActivity"
    }


    //creating variables for the api
    private lateinit var api: PolarBleApi

    //creating variables for printing out the HR and RR data
    private lateinit var textViewHR: TextView
    private lateinit var textViewRR: TextView

    //creating variables for printing out the accelerometer data for
each x, y, and z variable
    private lateinit var textViewAccX: TextView
    private lateinit var textViewAccY: TextView
    private lateinit var textViewAccZ: TextView

    //creating other variables for the components of the connected
device
    private lateinit var textViewDeviceId: TextView
    private lateinit var textViewBattery: TextView
    private lateinit var textViewFwVersion: TextView

    //creating variables for plotting
    private lateinit var plot: XYPlot
    private lateinit var accPlotter: AccPlotter
    private var accDisposable: Disposable? = null
    private var hrDisposable: Disposable? = null

    //variable for the connection of the specific Polar Device
    private lateinit var deviceId: String
    //variable for stopping and resuming the data stream
    private lateinit var stopStream: Button
    private lateinit var  resumeStream: Button

    //creating the variables for taking the data and putting it onto a
.txt file
    private lateinit var showTextFile: Button
    private lateinit var viewTextFileBox: TextView

    //variable to convert the time stamp to the date and time for the
.txt file
    private val simpleDateFormat = SimpleDateFormat("yyy-MM-dd
HH:mm:ss", Locale.ENGLISH)



    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_acc) // this is where the
intent takes places to go to the accelerometer activity

        //variable for the device's ID
        deviceId = intent.getStringExtra("id")
            ?: throw Exception("ACCActivity couldn't be created, no
deviceId given")

        //variables for HR and RR data
        textViewHR = findViewById(R.id.hr)
        textViewRR = findViewById(R.id.rr)

        //variables for accelerometer's data: x, y, and z
        textViewAccX = findViewById(R.id.acc_view_x)
        textViewAccY = findViewById(R.id.acc_view_y)
        textViewAccZ = findViewById(R.id.acc_view_z)

        //variables for the components of the connected device
        textViewDeviceId = findViewById(R.id.acc_view_deviceId)
        textViewBattery = findViewById(R.id.acc_view_battery_level)
        textViewFwVersion = findViewById(R.id.acc_view_fw_version)

        //variable for helping generate the plot
        plot = findViewById(R.id.acc_view_plot)

        //variable for stopping and resuming the data stream
        stopStream = findViewById(R.id.stopStreamButton)
        resumeStream = findViewById(R.id.resumeStreamButton)

        //creating the variables for taking the data and putting it
onto a .txt file
```

```
        showTextFile = findViewById(R.id.showTextFileButton)
        viewTextFileBox = findViewById(R.id.textFileTextView)



        //function that stops the streaming when the user clicks the
button to stop streaming
        stopStream.setOnClickListener {
            //stops the accelerometer data streaming
            accDisposable?.dispose()
            accDisposable =  null

            //stops the HR data streaming
            hrDisposable?.dispose()
            hrDisposable = null
        }


        resumeStream.setOnClickListener {
            //resumes the accelerometer data streaming
            streamACC()

            //resumes the HR data streaming
            streamHR()
        }


        //function that shows the content of the dataFile.txt file
        showTextFile.setOnClickListener {
            try {
                val fileInputStream = openFileInput("dataFile.txt")
                val inputReader = InputStreamReader(fileInputStream)
                val output = inputReader.readText()

                //Data is displayed in the TextView
                viewTextFileBox.text = output
            } catch (e: Exception) {
                e.printStackTrace()
            }
        }


        //api section where different features are enabled
        api = PolarBleApiDefaultImpl.defaultImplementation(
            applicationContext,
            setOf(
                PolarBleSdkFeature.FEATURE_POLAR_ONLINE_STREAMING,
                PolarBleSdkFeature.FEATURE_BATTERY_INFO,
                PolarBleSdkFeature.FEATURE_DEVICE_INFO
            )
        )

        //api section where different features are enabled
        api.setApiLogger { str: String -> Log.d("SDK", str) }

        api.setApiCallback(object : PolarBleApiCallback() {
            override fun blePowerStateChanged(powered: Boolean) {
                Log.d(TAG, "BluetoothStateChanged $powered")
            }

            override fun deviceConnected(polarDeviceInfo:
PolarDeviceInfo) {
                Log.d(TAG, "Device connected
${polarDeviceInfo.deviceId}")
                Toast.makeText(applicationContext, R.string.connected,
Toast.LENGTH_SHORT).show()
            }

            override fun deviceConnecting(polarDeviceInfo:
PolarDeviceInfo) {
                Log.d(TAG, "Device connecting
${polarDeviceInfo.deviceId}")
            }

            override fun deviceDisconnected(polarDeviceInfo:
PolarDeviceInfo) {
                Log.d(TAG, "Device disconnected
${polarDeviceInfo.deviceId}")
            }

            override fun bleSdkFeatureReady(
                identifier: String,
                feature: PolarBleApi.PolarBleSdkFeature
            ) {
                Log.d(TAG, "feature ready $feature")

                when (feature) {
                    PolarBleSdkFeature.FEATURE_POLAR_ONLINE_STREAMING
-> {
                        streamHR();
                        streamACC();
                        //streamECG();
                    }

                    else -> {}
                }
            }

            override fun hrFeatureReady(identifier: String) {
                //deprecated (updated github - 04/15/2023)
                //Log.d(TAG, "HR Feature ready $identifier")
            }

            override fun disInformationReceived(identifier: String,
uuid: UUID, value: String) {
                if (uuid == UUID.fromString("00002a28-0000-1000-8000-
00805f9b34fb")) {
                    val msg = "Firmware: " + value.trim { it <= ' ' }
                    Log.d(TAG, "Firmware: " + identifier + " " +
value.trim { it <= ' ' })
                    textViewFwVersion.append(msg.trimIndent())
                }
            }

            override fun batteryLevelReceived(identifier: String,
level: Int) {
                Log.d(TAG, "Battery level $identifier $level%")
                val batteryLevelText = "Battery level: $level%"
                textViewBattery.append(batteryLevelText)
            }

            /*override fun hrNotificationReceived(identifier: String,
data: PolarHrData) {
                //deprecated (updated github - 04/15/2023)
                //Log.d(TAG, "HR " + data.hr)
            }
             */

            override fun polarFtpFeatureReady(identifier: String) {
                //deprecated (updated github - 04/15/2023)
                //Log.d(TAG, "Polar FTP ready $identifier")
            }

        }) //ends onCreate and setApiCallback


        try {
            api.connectToDevice(deviceId)
        } catch (a: PolarInvalidArgument) {
            a.printStackTrace()
        }

        val deviceIdText = "ID: $deviceId"
        textViewDeviceId.text = deviceIdText


        accPlotter = AccPlotter()
        accPlotter.setListener(this)

        //I think I have to add a time series for each formatter
        //plot.addSeries(accPlotter.xTimeSeries,
accPlotter.xFormatter)
        plot.addSeries(accPlotter.xSeries, accPlotter.xFormatter)

        //plot.addSeries(accPlotter.yTimeSeries,
accPlotter.yFormatter)
        plot.addSeries(accPlotter.ySeries, accPlotter.yFormatter)

        //plot.addSeries(accPlotter.zTimeSeries,
accPlotter.zFormatter)
        plot.addSeries(accPlotter.zSeries, accPlotter.zFormatter)

        plot.setRangeBoundaries(-3000, 3000, BoundaryMode.FIXED)
        plot.setDomainBoundaries(0, 360000, BoundaryMode.AUTO)

        //Left labels will increment by 250.00
        plot.setRangeStep(StepMode.INCREMENT_BY_VAL, 500.00)
        plot.setDomainStep(StepMode.INCREMENT_BY_VAL, 60000.0)

        with(plot.domainTitle.text) {

        }

        plot.legend.isVisible

        //Make left labels into an integer (no decimal places)
        plot.graph.getLineLabelStyle(XYGraphWidget.Edge.LEFT).format =
DecimalFormat("#")

        //These don't seem to have an effect
        plot.linesPerRangeLabel = 2

    }

    public override fun onDestroy() {
        super.onDestroy()
        api.shutDown()
    }


    //function that streams the accelerometer data (used both the
PolarAdkEcgHrDemo app's streamECG() function in
    //     in the ECGActivity.kt file and the code in the
AndroidBleSdkTestApp's accButton.setOnClicklistener)
    fun streamACC() {
        val fileOutputStream = openFileOutput("dataFile.txt",
Context.MODE_PRIVATE)
        val outputWriter = OutputStreamWriter(fileOutputStream)
        val isDisposed = accDisposable?.isDisposed ?: true
        if (isDisposed) {
            //toggleButtonDown(accButton, R.string.stop_acc_stream)
//don't believe I need this line from the function from
AndroidBleSdkTestApp bc I am not clicking a start accelerometer data
extraction button in this file's .xml file
            //accDisposable = api.requestStreamSettings(deviceId,
sensorSetting.maxSettings()) //DeviceStreamingFeature.ACC exists in
line 260 of the PolarBleApi.DeviceStreamingFeature.html file in the
github
            accDisposable = api.requestStreamSettings(deviceId,
PolarBleApi.PolarDeviceDataType.ACC)
                .toFlowable() //unsure if I need this or not
                .flatMap { sensorSetting: PolarSensorSetting ->
                    api.startAccStreaming(
                        deviceId,
                        sensorSetting.maxSettings()
                    )
                }
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(
                    { polarAccelerometerData: PolarAccelerometerData -
>
                        Log.d(TAG, "accelerometer update")
                        for (data in polarAccelerometerData.samples) {
                            Log.d(
                                TAG,
                                "ACC:    x: ${data.x} y: ${data.y} z:
${data.z} timeStamp: ${data.timeStamp}"
                            )
                            //attempting to add controls for
SAMPLE_RATE and RANGE in the Polar Sensor Settings
                            val settings:
MutableMap<PolarSensorSetting.SettingType, Int> =
                                mutableMapOf()

//settings[PolarSensorSetting.SettingType.SAMPLE_RATE] = 208

//settings[PolarSensorSetting.SettingType.RANGE] = 16
                            textViewAccX.text = "x = " +
data.x.toString()
                            textViewAccY.text = "y = " +
data.y.toString()
                            textViewAccZ.text = "z = " +
data.z.toString()
                            accPlotter.addValues(data)


                            //this code below is going to create a
text file and write string into it, else it opens the file and writes
the string, and also saves the information to the .txt file
```

```
                            if (data.timeStamp.toString().isNotEmpty()
&& data.x.toString().isNotEmpty() &&
                                data.y.toString().isNotEmpty() &&
data.z.toString().isNotEmpty()) {
                                try {
                                    //outputWriter.write("Time Stamp:
" + data.timeStamp. + "\n")
                                    outputWriter.write("Time: " +
getDateString(data.timeStamp) + "\n")
                                    outputWriter.write("X: " +
data.x.toString() + "\n")
                                    outputWriter.write("Y: " +
data.y.toString() + "\n")
                                    outputWriter.write("Z: " +
data.z.toString() + "\n")
                                    outputWriter.write("\n")
                                    //for (data in
polarAccelerometerData.samples){
                                    //
outputWriter.write(data.timeStamp.toString() + " " + data.x.toString()
+ " " +
                                    //
data.y.toString() + " " + data.z.toString() + "\n")
                                    //}

                                } catch (e: Exception) {
                                    e.printStackTrace()
                                }

                            } else {
                                Toast.makeText(applicationContext, "No
input?", Toast.LENGTH_SHORT)
                                    .show()
                            } //end of reading data into .txt file

                        //See if I can stop the stream if they click
the "Stop Stream" button
                        //  which will also finish the streaming

                        //See if I can put an if condition if the save
button is clicked to ensure the stream
                        // has stopped running before opening up .txt
file

                        } // end of reading in data

                    //on click of the show text file button


                    },
                    { error: Throwable ->
                        Log.e(TAG, "Accelerometer stream failed
$error")
                        accDisposable = null
                    },
                    {
                        Log.d(TAG, "Accelerometer stream complete")

                        outputWriter.close()
                        Toast.makeText(
                            baseContext,
                            "File saved successfully!",
                            Toast.LENGTH_SHORT
                        ).show()
                    }
                )
        } else {
            //NOTE stops streaming if it is "running
            accDisposable?.dispose()
            accDisposable = null
            //didn't include line from AndroidBleSdkTestApp
accelerometer button function that says "toggleButtonUp(accButton,
R.string.start_acc_stream)
        }
    }

    private fun getDateString(time: Long) : String =
simpleDateFormat.format(time * 1000L)
    private fun getDateString(time: Int) : String =
simpleDateFormat.format(time * 1000L)


    //function that streams the HR data (used both the
PolarAdkEcgHrDemo app's streamHR() function in
    //     in the ECGActivity.kt file and the code in the
AndroidBleSdkTestApp's hrButton.setOnClicklistener
    fun streamHR(){
        val isDisposed = hrDisposable?.isDisposed ?: true
        if (isDisposed) {
            hrDisposable = api.startHrStreaming(deviceId)
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(
                    { hrData: PolarHrData ->
                        for (sample in hrData.samples) {
                            Log.d(TAG, "HR " + sample.hr)
                            if (sample.rrsMs.isNotEmpty()) {
                                val rrText =
"(${sample.rrsMs.joinToString(separator = "ms, ")}ms)"
                                textViewRR.text = rrText
                            }
                            textViewHR.text = sample.hr.toString()
                            //plot.addValues(sample)

                        }
                    },
                    { error: Throwable ->
                        Log.e(TAG, "HR stream failed. Reason $error")
                        hrDisposable = null
                    },
                    { Log.d(TAG, "HR stream complete")}
                )
        } else {
            //NOTE stops streaming if it is "running"
            hrDisposable?.dispose()
            hrDisposable = null
        }
    }


    override fun update() {
        runOnUiThread { plot.redraw() }
    }

    //private fun onClickConnectEcg(view: View) {
    //    streamACC().cancel
    //}


/*
    ***my attempt to make a function for inputting data into a .txt
file but as of now, 04/21/2023,
        I am deciding to just put my code into the actual streamACC()
function***

    fun dataToTxt () {
        if (data.x.toString().isNotEmpty()){
            try{
                val fileOutputStream = openFileOutput("dataFile.txt",
Context.MODE_PRIVATE)
                val outputWriter =
OutputStreamWriter(fileOutputStream)
                outputWriter.write(data.x.toString())
            }
        }
    }
 */
```

hannazelis@gmail.com
November 29, 2023

Polar Verity Mobile App Development Research

```
//Name: Hanna Zelis
//File: AccPlotter.kt
//Date: 4/21/2023
//Description: This is the accelerometer plotter file from my Polar
Verity research


package com.polar.polarsdkecghrdemo

import android.graphics.Color
import android.graphics.Paint
import com.androidplot.xy.LineAndPointFormatter
import com.androidplot.xy.SimpleXYSeries
import com.androidplot.xy.XYSeriesFormatter
import com.polar.sdk.api.model.PolarAccelerometerData
import java.util.*


//import com.androidplot.xy.AdvancedLineAndPointRenderer

class AccPlotter {
    companion object {
        private const val TAG = "AccPlotter"
        //may need new number values to determine the distance of
plotting for accelerometer data
        private const val NVALS = 300 //15 = 15 seconds; 300 = 5 min
        private const val RR_SCALE = .1
    }



    private var listener: PlotterListener? = null

    //val timeFormatter: XYSeriesFormatter<*>
    val xFormatter: XYSeriesFormatter<*>
    val yFormatter: XYSeriesFormatter<*>
    val zFormatter: XYSeriesFormatter<*>

    //val xTimeSeries: SimpleXYSeries
    //val yTimeSeries: SimpleXYSeries
    //val zTimeSeries: SimpleXYSeries
    val xSeries: SimpleXYSeries
    val ySeries: SimpleXYSeries
    val zSeries: SimpleXYSeries

    private val xTimeAccVals = MutableList(NVALS) { 0.0 }
    private val yTimeAccVals = MutableList(NVALS) { 0.0 }
    private val zTimeAccVals = MutableList(NVALS) { 0.0 }
    private val xAccVals = MutableList(NVALS) { 0.0 }
    private val yAccVals = MutableList(NVALS) { 0.0 }
    private val zAccVals = MutableList(NVALS) { 0.0 }

    //val hrFormatter: XYSeriesFormatter<*>
    //val rrFormatter: XYSeriesFormatter<*>

    //val hrSeries: SimpleXYSeries
    //val rrXYSeries: SimpleXYSeries

    //private val xHrVals = MutableList(NVALS) { 0.0 }
    //private val yHrVals = MutableList(NVALS) { 0.0 }
    //private val xRrVals = MutableList(NVALS) { 0.0 }
    //private val yRrVals = MutableList(NVALS) { 0.0 }



    init {
        //variables to help calculate the start, end, and delta time
(may need new math for acceleterometer data)+
        val now = Date()
        val endTime = now.time.toDouble()
        val startTime = endTime - NVALS * 1000
        val delta = (endTime - startTime) / (NVALS - 1)

        //specify initial values to keep it from auto sizing
        for (i in 0 until NVALS) {
            xTimeAccVals[i] = startTime + i * delta
            xAccVals[i] = -0.1
            yTimeAccVals[i] = startTime + i * delta
            yAccVals[i] = -1.0
            zTimeAccVals[i] = startTime + i * delta
            zAccVals[i] = -0.2

            //xHrVals[i] = startTime + i * delta
            //yHrVals[i] = 60.0
            //xRrVals[i] = startTime + i * delta
            //yRrVals[i] = 100.0
        }

        //plotting timeStamp vs. x-values with a red line
        xFormatter = LineAndPointFormatter(Color.RED, null, null,
null)
        //xFormatter.isLegendIconEnabled()
        xSeries = SimpleXYSeries(xTimeAccVals, xAccVals, "x")

        //plotting timeStamp vs. y-values with a blue line
        yFormatter = LineAndPointFormatter(Color.BLUE, null, null,
null)
        //yFormatter.isLegendIconEnabled()
        ySeries = SimpleXYSeries(yTimeAccVals, yAccVals, "y")
        //plotting timeStamp vs. z-values with a green line
        zFormatter = LineAndPointFormatter(Color.GREEN, null, null,
null)
        //zFormatter.isLegendIconEnabled()
        zSeries = SimpleXYSeries(zTimeAccVals, zAccVals, "z")


    }


    /**
     * Implements a strip chart by moving series data backwards and
adding
     * new data at the end.
     *
     * @param polarAccelerometerData The Accelerometer data that came
in.
     */



    fun addValues(polarAccelerometerData:
PolarAccelerometerData.PolarAccelerometerDataSample) {
        val now = Date()
        val time = now.time
        for (i in 0 until NVALS - 1) {
            xTimeAccVals[i] = xTimeAccVals[i + 1]
            yTimeAccVals[i] = yTimeAccVals[i + 1]
            zTimeAccVals[i] = zTimeAccVals[i + 1]
            xAccVals[i] = xAccVals[i + 1]
            yAccVals[i] = yAccVals[i + 1]
            zAccVals[i] = zAccVals[i + 1]

            xSeries.setXY(xTimeAccVals[i], xAccVals[i], i)
            ySeries.setXY(yTimeAccVals[i], yAccVals[i], i)
            zSeries.setXY(zTimeAccVals[i], zAccVals[i], i)

            //hrSeries.setXY(xHrVals[i], yHrVals[i], i)

            //xHrVals[i] = xHrVals[i + 1]
            //yHrVals[i] = yHrVals[i + 1]
            //xRrVals[i] = xRrVals[i + 1]
            //yRrVals[i] = yRrVals[i + 1]

            //hrSeries.setXY(xHrVals[NVALS - 1], yHrVals[NVALS - 1],
NVALS - 1)

        }
        xTimeAccVals[NVALS - 1] = time.toDouble()
        xAccVals[NVALS - 1] = polarAccelerometerData.x.toDouble()
        xSeries.setXY(xTimeAccVals[NVALS - 1], xAccVals[NVALS - 1],
NVALS - 1)

        yTimeAccVals[NVALS - 1] = time.toDouble()
        yAccVals[NVALS - 1] = polarAccelerometerData.y.toDouble()
        ySeries.setXY(yTimeAccVals[NVALS - 1], yAccVals[NVALS - 1],
NVALS - 1)

        zTimeAccVals[NVALS - 1] = time.toDouble()
        zAccVals[NVALS - 1] = polarAccelerometerData.z.toDouble()
        zSeries.setXY(zTimeAccVals[NVALS - 1], zAccVals[NVALS - 1],
NVALS - 1)

        //below here begins the impolementation of RR data, which I
will not focus on right now
        //     as we don't know at what time the RR intervals start

        listener?.update()
    }



    fun setListener(listener: PlotterListener?) {
        this.listener = listener
    }

}
```

hannazelis@gmail.com
November 29, 2023

Polar Verity Mobile App Development Research

```
//Name: Hanna Zelis
//File: MainActivity.kt
//Date: 4/17/2023
//Description: This is the main activity file from my Polar Verity
research


package com.polar.polarsdkecghrdemo

import android.Manifest
import android.app.Activity
import android.bluetooth.BluetoothAdapter
import android.bluetooth.BluetoothManager
import android.content.DialogInterface
import android.content.Intent
import android.content.SharedPreferences
import android.content.pm.PackageManager
import android.os.Build
import android.os.Bundle
import android.text.InputType
import android.util.Log
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Button
import android.widget.EditText
import android.widget.Toast
import androidx.activity.result.ActivityResult
import androidx.activity.result.contract.ActivityResultContracts
import androidx.appcompat.app.AlertDialog
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {
    companion object {
        private const val TAG = "Polar_MainActivity"
        private const val SHARED_PREFS_KEY = "polar_device_id"
        private const val PERMISSION_REQUEST_CODE = 1
    }

    private lateinit var sharedPreferences: SharedPreferences
    private val bluetoothOnActivityResultLauncher =
registerForActivityResult(ActivityResultContracts.StartActivityForResu
lt()) { result: ActivityResult ->
        if (result.resultCode != Activity.RESULT_OK) {
            Log.w(TAG, "Bluetooth off")
        }
    }
    private var deviceId: String? = null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        sharedPreferences = getPreferences(MODE_PRIVATE)
        deviceId = sharedPreferences.getString(SHARED_PREFS_KEY, "")

        val setIdButton: Button = findViewById(R.id.buttonSetID)
        val ecgConnectButton: Button =
findViewById(R.id.buttonConnectEcg)
        val hrConnectButton: Button =
findViewById(R.id.buttonConnectHr)
        val accConnectButton: Button =
findViewById(R.id.buttonConnectAcc)
        checkBT()

        setIdButton.setOnClickListener { onClickChangeID(it) }
        ecgConnectButton.setOnClickListener { onClickConnectEcg(it) }
        hrConnectButton.setOnClickListener { onClickConnectHr(it) }
        accConnectButton.setOnClickListener { onClickConnectAcc(it)}
    }

    private fun onClickConnectEcg(view: View) {
        checkBT()
        if (deviceId == null || deviceId == "") {
            deviceId = sharedPreferences.getString(SHARED_PREFS_KEY,
"")
            showDialog(view)
        } else {
            showToast(getString(R.string.connecting) + " " + deviceId)
            val intent = Intent(this, ECGActivity::class.java)
            intent.putExtra("id", deviceId)
            startActivity(intent)
        }
    }

    private fun onClickConnectHr(view: View) {
        checkBT()
        if (deviceId == null || deviceId == "") {
            deviceId = sharedPreferences.getString(SHARED_PREFS_KEY,
"")
            showDialog(view)
        } else {
            showToast(getString(R.string.connecting) + " " + deviceId)
            val intent = Intent(this, HRActivity::class.java)
            intent.putExtra("id", deviceId)
            startActivity(intent)
        }
    }

    private fun onClickConnectAcc(view: View) {
        checkBT()
        if (deviceId == null || deviceId == "") {
            deviceId = sharedPreferences.getString(SHARED_PREFS_KEY,
"")
            showDialog(view)
        } else {
            showToast(getString(R.string.connecting) + " " + deviceId)
            val intent = Intent(this, ACCActivity::class.java)
            intent.putExtra("id", deviceId)
            startActivity(intent)
        }
    }

    private fun onClickChangeID(view: View) {
        showDialog(view)
    }

    private fun showDialog(view: View) {
        val dialog = AlertDialog.Builder(this, R.style.PolarTheme)
        dialog.setTitle("Enter your Polar device's ID")
        val viewInflated =
LayoutInflater.from(applicationContext).inflate(R.layout.device_id_dia
log_layout, view.rootView as ViewGroup, false)
        val input = viewInflated.findViewById<EditText>(R.id.input)
        if (deviceId?.isNotEmpty() == true) input.setText(deviceId)
        input.inputType = InputType.TYPE_CLASS_TEXT
        dialog.setView(viewInflated)
        dialog.setPositiveButton("OK") { _: DialogInterface?, _: Int -
>
            deviceId = input.text.toString().uppercase()
            val editor = sharedPreferences.edit()
            editor.putString(SHARED_PREFS_KEY, deviceId)
            editor.apply()
        }
        dialog.setNegativeButton("Cancel") { dialogInterface:
DialogInterface, _: Int -> dialogInterface.cancel() }
        dialog.show()
    }

    private fun checkBT() {
        val btManager =
applicationContext.getSystemService(BLUETOOTH_SERVICE) as
BluetoothManager
        val bluetoothAdapter: BluetoothAdapter? = btManager.adapter
        if (bluetoothAdapter == null) {
            showToast("Device doesn't support Bluetooth")
            return
        }

        if (!bluetoothAdapter.isEnabled) {
            val enableBtIntent =
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE)
            bluetoothOnActivityResultLauncher.launch(enableBtIntent)
        }

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {

requestPermissions(arrayOf(Manifest.permission.BLUETOOTH_SCAN,
Manifest.permission.BLUETOOTH_CONNECT), PERMISSION_REQUEST_CODE)
            } else {

requestPermissions(arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
PERMISSION_REQUEST_CODE)
            }
        } else {

requestPermissions(arrayOf(Manifest.permission.ACCESS_COARSE_LOCATION)
, PERMISSION_REQUEST_CODE)
        }
    }

    override fun onRequestPermissionsResult(requestCode: Int,
permissions: Array<String>, grantResults: IntArray) {
        super.onRequestPermissionsResult(requestCode, permissions,
grantResults)
        if (requestCode == PERMISSION_REQUEST_CODE) {
            for (index in 0..grantResults.lastIndex) {
                if (grantResults[index] ==
PackageManager.PERMISSION_DENIED) {
                    Log.w(TAG, "Needed permissions are missing")
                    showToast("Needed permissions are missing")
                    return
                }
            }
            Log.d(TAG, "Needed permissions are granted")
        }
    }

    private fun showToast(message: String) {
        val toast = Toast.makeText(applicationContext, message,
Toast.LENGTH_LONG)
        toast.show()
    }
}
```

hannazelis@gmail.com
November 29, 2023

Polar Verity Mobile App Development Research

```
//Name: Hanna Zelis
//File: activity_acc.xml
//Date: 4/21/2023
//Description: This is the accelerometer data streaming file from my
Polar Verity research


<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/acc_view_z"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textAlignment="center"
        android:textColor="#C00000"
        android:textSize="30sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/acc_view_y"
        tools:text="100" />

    <TextView
        android:id="@+id/acc_view_y"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textAlignment="center"
        android:textColor="#C00000"
        android:textSize="30sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/acc_view_x"
        tools:ignore="MissingConstraints"
        tools:text="100" />

    <LinearLayout
        android:id="@+id/acc_view_heading"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:orientation="vertical"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">
        <TextView
            android:id="@+id/acc_view_deviceId"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textAlignment="center"
            android:textColor="@android:color/black"
            android:textSize="20sp"
            tools:text="Id 123456" />

        <TextView
            android:id="@+id/acc_view_battery_level"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textAlignment="center"
            android:textColor="@android:color/black"
            android:textSize="20sp"
            tools:text="Battery level: 80%" />

        <TextView
            android:id="@+id/acc_view_fw_version"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textAlignment="center"
            android:textColor="@android:color/black"
            android:textSize="20sp"
            tools:text="Firmware: 3.1.1" />

        <TextView
            android:id="@+id/hr"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginBottom="8dp"
            android:textAlignment="center"
            android:textColor="#C00000"
            android:textSize="30sp"
            tools:text="100" />

        <TextView
            android:id="@+id/rr"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textAlignment="center"
            android:textColor="@android:color/holo_blue_dark"
            android:textSize="16sp"
            tools:text="(1002ms, 1009ms)" />

    </LinearLayout>
    <com.androidplot.xy.XYPlot
        android:id="@+id/acc_view_plot"
        style="@style/FullScreenGraph"
        android:layout_width="fill_parent"
        android:layout_height="304dp"
        android:layout_marginTop="345dp"
        app:backgroundColor="@color/colorAccent"
        app:graphBackgroundColor="@color/colorAccent"
        app:graphMarginBottom="12dp"
        app:graphMarginLeft="30dp"
        app:graphMarginRight="5dp"
        app:graphMarginTop="20dp"
        app:gridBackgroundColor="@color/colorAccent"
        app:gridInsetLeft="25dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:lineLabels="left"
        app:rangeTitle="@string/hr_range_title"
        app:rangeTitleTextColor="@android:color/black"
        app:rangeTitleTextSize="12dp"
        app:renderMode="use_background_thread"
        app:title="@string/hr_title"
        app:titleTextColor="@android:color/black"
        app:titleTextSize="15dp" />

    <Button
        android:id="@+id/stopStreamButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="7dp"
        android:layout_marginEnd="7dp"
        android:text="Stop Streaming"
        app:layout_constraintEnd_toEndOf="@+id/acc_view_y"
        app:layout_constraintTop_toBottomOf="@+id/acc_view_x"
        tools:ignore="MissingConstraints" />

    <Button
        android:id="@+id/resumeStreamButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:text="Start Streaming"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/acc_view_heading"
        tools:ignore="MissingConstraints" />

    <Button
        android:id="@+id/showTextFileButton"
```

Hanna M. Zelis
hannazelis@gmail.com
November 29, 2023

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginStart="16dp"
        android:layout_marginTop="8dp"
        android:text="Show .txt file"
        app:layout_constraintStart_toStartOf="@+id/acc_view_y"
        app:layout_constraintTop_toBottomOf="@+id/acc_view_x"
        tools:ignore="MissingConstraints" />

    <TextView
        android:id="@+id/acc_view_x"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="1dp"
        android:textAlignment="center"
        android:textColor="#C00000"
        android:textSize="30sp"
        app:layout_constraintTop_toBottomOf="@+id/acc_view_heading"
        tools:ignore="MissingConstraints"
        tools:layout_editor_absoluteX="-2dp"
        tools:text="100" />

    <TextView
        android:id="@+id/textFileTextView"
        android:layout_width="320dp"
        android:layout_height="79dp"
        android:layout_marginTop="2dp"
        app:layout_constraintTop_toBottomOf="@+id/acc_view_z"
        tools:layout_editor_absoluteX="37dp" />

</androidx.constraintlayout.widget.ConstraintLayout>
```