



# Android-kirjasto fyysisen aktiviteetin reaaliaikaiseen tunnistamiseen

Jussi Kanto

OPINNÄYTETYÖ  
Syyskuu 2024

Tietojenkäsittely  
Ohjelmistotuotanto

# TIIVISTELMÄ

Tampereen ammattikorkeakoulu  
Tietojenkäsittely  
Ohjelmistotuotanto

KANTO, JUSSI

Android-kirjasto fyysisen aktiviteetin reaaliaikaiseen tunnistamiseen

Opinnäytetyö 63 sivua  
Syyskuu 2024

---

Opinnäytetyössä kehitettiin Kotlin-ohjelmointikielellä Android-sovellus, joka hyödyntää laitteen kiihtyvyysanturia käyttäjän suorittamien aktiviteettien tunnistamiseen. Lisäksi luotiin sovelluksessa käytetyistä menetelmistä Android-kirjasto, joka julkaistiin GitHubilla ja JitPackilla Android-kehittäjän käytettäväksi.

Sovelluksessa tallennetaan käyttäjän suorittamia aktiviteetteja tietokantaan tietyllä ajanjaksolla. Tallentamiseen käytetään kiihtyvyysanturin tarjoamaa dataa. Sovellus pyrkii tunnistamaan käyttäjän tekemän aktiviteetin vertaamalla reaaliaikaista kiihtyvyysdataa tietokannassa oleviin tallennettuihin datoihin. Sovelluksessa käytetyistä komponenteista eriytettiin oma Android-kirjasto. Kirjaston dokumentaatio tuotettiin KDoc:n ja Dokkan avulla.

Tuloksena syntyi sovellus, joka tunnistaa melko hyvin kiihtyvyyden muutoksia sisältävää liikuntaa sekä dokumentoitu ja testattu kirjasto, jonka voi helposti integroida Android-projekteihin. Tulokset tarjoavat tietoa tuleville sovelluskehitysprojekteille, jotka hyödyntävät kiihtyvyysanturin dataa käyttäjän aktiviteettien seuraamiseen ja tunnistamiseen. Jatkokehityksessä voidaan harkita monimutkaisempien tunnistusmenetelmien hyödyntämistä, jotta tunnistusprosessissa saavutettaisiin vakaampia tuloksia.

---

Asiasanat: android-sovellus, android-kirjasto, kotlin, kiihtyvyysanturi, jitpack

## **ABSTRACT**

Tampereen ammattikorkeakoulu  
Tampere University of Applied Sciences  
Degree Programme in Business Information Systems  
Software Production

KANTO, JUSSI  
Android Library for Real-time Physical Activity Recognition

Bachelor's thesis 63 pages  
September 2024

---

The thesis focuses on the development of an Android application using the Kotlin programming language, which utilises the device's accelerometer to recognise user activities. Additionally, a library containing the methods used in the application was created and published on GitHub and JitPack for Android developers to utilise.

The application stores user activities in a database over a specified time, using the data provided by the accelerometer sensor. It attempts to recognise the user's activity by comparing real-time accelerometer data with the stored data in the database. The components used in the application were separated into their own Android library. The library's documentation was generated using KDoc and Dokka.

The result is an application that quite accurately recognises activities involving changes in acceleration, along with a documented and tested Android library that can be easily integrated into Android projects. The findings provide insights for future app development projects that utilise accelerometer data for tracking and recognising user activities. Further development may involve using more complex recognition methods to achieve more stable results.

---

Key words: android application, android library, kotlin, accelerometer, jitpack

## SISÄLLYS

1	JOHDANTO .....	7
2	TEKNOLOGIAT JA TYÖKALUT .....	8
2.1	Android.....	8
2.1.1	Android-sovelluskehitys.....	8
2.1.2	Kotlin-ohjelmointikieli .....	9
2.2	Anturit Androidissa .....	9
2.2.1	Kiihtyvyysanturi.....	10
2.3	SQL ja SQLite .....	11
2.4	Android-kirjastot .....	11
2.5	Rakentaminen ja hallinta: Gradle, Maven ja JitPack .....	12
3	SOVELLUKSEN TOTEUTUS .....	13
3.1	Modulaarinen suunnittelu ja kehitys .....	13
3.2	Projektin alustus.....	13
3.3	Sensor API.....	13
3.4	Signaalin suodatus.....	16
3.5	Kiihtyvyyssdan visualisointi .....	17
3.6	Tietokannan suunnittelu ja toteutus.....	20
3.6.1	Tietokannan rakenne.....	20
3.6.2	Tiedon tallennus ja haku.....	23
3.7	Treenidatan kerääminen .....	24
3.7.1	Dialogi .....	24
3.7.2	Valitun aktiviteetin näytteenotto .....	28
3.8	Reaaliaikaisen datan vertailu testidataan .....	31
3.8.1	Esikäsittely .....	31
3.8.2	Tunnistaminen .....	33
3.9	Tulokset .....	36
4	KIRJASTO .....	42
4.1	Kirjaston alustus.....	42
4.1.1	Luokkien ja funktioiden lopulliset eriyttämiset.....	42
4.2	Testaus .....	43
4.3	Kirjaston dokumentaatio.....	46
4.3.1	KDoc.....	46
4.3.2	Dokka .....	47
4.3.3	GitHub Pages .....	48
4.3.4	Readme.....	50
4.4	Kirjaston julkaiseminen .....	51

4.4.1 Lokaali testijulkaisu.....	52
4.4.2 Julkaisu GitHubiin ja jakelu JitPackilla.....	54
4.5 Kirjaston integrointi projektissa.....	58
5 POHDINTA .....	60
LÄHTEET.....	62

## ERITYISSANASTO

Activity	Android-sovelluksen käyttöliittymää muokkaava näkymä
Aktiviteetti	Käyttäjän suorittama aktiviteetti, kuten kävely, juoksu, pyöräily jne.
Artefakti	Valmiiksi rakennettu ohjelmistokomponentti
Avoin lähdekoodi	Sallii ohjelmiston vapaan käytön, jakamisen ja muokkaamisen
Domain	Osa URL-osoitetta, joka identifioi verkkosivuston nimen internetissä
JUnit	Yksikkötestauskehys Java-ohjelmointikielelle
JVM	Ohjelmisto, joka suorittaa sille käännettyjä Java-ohjelmia tavukoodina eri käyttöjärjestelmissä
Olio	Luokan ilmentymä
Relaatiotietokanta	Tietokantamalli, jossa tiedot tallennetaan taulukoihin, joidenka välillä luodaan yhteyksiä avaimien avulla
Repositorio	Tallennuspaikka versionhallintajärjestelmässä, jossa säilytetään projektin koodia ja sen muutoksia
Skripti	Ajon aikana tulkattava, usein lyhyt, ohjelma

# 1 JOHDANTO

Nyky-yhteiskunnassa mobiililaitteet ovat keskeisessä roolissa arkipäiväisten toimintojen helpottamisessa. Ne ovat vallankumouksellisesti muuttaneet tapamme kommunikoida, oppia ja viihtyä. Älypuhelimien kiihtyvyysanturit ovat yksi avainteknologioista tässä murroksessa, erityisesti liikunta- ja terveyssovelluksissa. Niiden hyödyntäminen käyttäjän liikkeen seurannassa ja tunnistamisessa tarjoaa lukuisia mahdollisuuksia mobiilisovelluskehityksessä.

Tämän opinnäytetyön tavoitteena on kehittää Android-sovellus Kotlin-ohjelmointikielellä. Sovellus pyrkii tunnistamaan käyttäjän fyysisesti suorittamia aktiviteetteja hyödyntäen laitteen kiihtyvyysanturia. Kun sovellus on saatu valmiiksi, siitä eriytetään kaikki tarvittavat osat omaan kirjastoprojektiin. Testattu ja dokumentoitu kirjasto julkaistaan GitHubissa, josta JitPackilla sitä jaetaan kehittäjille globaalisti.

Opinnäytetyön tarkoituksena on tarjota käyttäjille väline, joka ei ainoastaan tallenna heidän tekemiä aktiviteettejaan, vaan myös tunnistaa ne kiihtyvyysdatan avulla vertailemalla olemassa olevaa dataa reaaliaikaiseen dataan. Samalla työ tarjoaa kehittäjille valmiita työkaluja, jotka helpottavat kiihtyvyysdatan käsittelyä ja sen integroimista Android-projekteihin.

Opinnäytetyön tulokset tulevat tarjoamaan tietoa ja pohjaa tuleville sovelluskehitysprojekteille, jotka hyödyntävät kiihtyvyysanturin dataa aktiviteettien tunnistamiseen.

## **2 TEKNOLOGIAT JA TYÖKALUT**

Tässä osiossa käydään läpi projektissa käytettyjen teknologioiden ja työkalujen taustaa.

### **2.1 Android**

Android on Googllelle kuuluva mobiilikäyttöjärjestelmä, joka on kehitetty erityisesti älypuhelimille ja tableteille. Se perustuu Linux-ytimeen ja on avoimen lähdekoodin järjestelmä, mikä tarkoittaa, että sen koodi on julkisesti saatavilla ja kehittäjät voivat muokata sitä (Tutorialspoint. n.d). Tämä tekee siitä monipuolisen ja räätälöitävän.

Vaikka sekä Android että Applen iOS ovat vahvoja toimijoita älypuhelinmarkkinoilla, tämä työ toteutetaan Android-alustalle.

#### **2.1.1 Android-sovelluskehitys**

Android-sovellusten tekemiseen käytetään pääasiassa Android Studio -kehitysympäristöä (IDE). Se sisältää Android SDK:n (Software Development Kit), joka on kehityspaketti, tarjoten monipuoliset työkalut ja resurssit sovelluskehittäjille kaikissa kehitysvaiheissa. Sovellukset, jotka on rakennettu näillä työkaluilla, tunnetaan natiiveina sovelluksina, mikä tarkoittaa, että ne on kehitetty erityisesti Android-alustalle. Tämä avaa paremmat mahdollisuudet käyttää laitteen sisäisiä resursseja ja paremman suorituskyvyn verrattuna monialustaisiin ratkaisuihin, jotka ovat suunniteltu toimimaan useilla eri käyttöjärjestelmillä.

Ohjelmointikielet, joilla Android-sovellukset yleisimmin kirjoitetaan, ovat Java ja Kotlin. Lisäksi joissakin tapauksissa käytetään C++:aa erityistarkoituksiin, kuten suorituskyvyn optimointiin ja matalan tason järjestelmäresursseihin liittyvään ohjelmointiin (Yuudai 2023).

Android-sovellusten käyttöliittymäpuolta toteutetaan perinteisesti XML-kielellä (Extension Markup Language), joka määrittelee sovelluksen näkymän ja



ulkoasun eri komponentit, kuten painikkeet, tekstit ja asetellut erillisten XML-tiedostojen avulla. On kuitenkin tärkeää huomata, että nykyään Google suosittelee Jetpack Composea Android-sovellusten käyttöliittymien rakentamiseen. Tämä uusi tapa mahdollistaa käyttöliittymien luomisen Kotlin-ohjelmointikielellä, tarjoten modernimpia työkaluja, jotka helpottavat ja nopeuttavat kehitystä vähemmällä koodilla ja tehokkaammilla työkaluilla (Why adopt Compose 2024).

### **2.1.2 Kotlin-ohjelmointikieli**

Kotlin on moderni ohjelmointikieli, joka toimii JVM (Java Virtual Machine) -alustalla. Google on suositellut sitä käytettäväksi Android-sovelluskehityksessä vuodesta 2019 lähtien. Se tarjoaa kehittäjille sekä oliopohjaisen että funktionaalisen ohjelmoinnin mahdollisuuden. Sen syntaksi on selkeää ja tiivistä, joka tarjoaa kehittäjälle enemmän toiminnallisuutta vähemmällä koodilla. Helppolukuinen syntaksi helpottaa ohjelmoijaa ymmärtämään koodia paremmin verrattuna melko vaikea lukuiseen Javaan. Kotlin on helppo integroida Java-projekteihin, ja sillä on käytössään koko Javan ekosysteemi ja kirjastot omien kirjastojensa lisäksi. (Guru Technolabs 2023)

Kotlin syntyi tarpeesta tarjota nykyaikaisempi vaihtoehto Javalle. Sen tavoitteena oli luoda ohjelmointikieli, joka olisi selkeämpi, tehokkaampi ja turvallisempi kehittäjille. Yhteensopivuus Javan kanssa oli olennainen osa Kotlinin suunnittelua.

## **2.2 Anturit Androidissa**

Android-laitteissa on yleensä sisäänrakennettuja antureita, jotka mittaavat esimerkiksi liikettä, asentoa, ympäristön olosuhteita ja muutoksia jne. Näistä saatua dataa voidaan sitten monipuolisesti hyödyntää omissa projekteissa.

Osa antureista on rakennettu laitteeseen fyysisesti, ja osa on ohjelmistopohjaisia. Fyysiset anturit mittavat suoraan tiettyjä ympäristöominaisuuksia, kuten liikettä, magneettikentän voimakkuutta tai kulmanmuutoksia. Ohjelmistopohjaiset anturit eivät ole fyysisiä, mutta ne käyttäytyvät kuin sellaiset. Ne saavat tietonsa fyysisiltä

antureilta ja niitä kutsutaan joskus virtuaalisiksi antureiksi. (Sensors Overview 2024)

### 2.2.1 Kiihtyvyysanturi

Kiihtyvyysanturi on laite, joka havainnoi laitteeseen kohdistuvaa kiihtyvyyttä, eli muutosta nopeudessa tietyllä ajanjaksolla. Nämä anturit voivat havaita kiihtyvyyden eri suunnissa, kuten x, y ja z -akseleilla.

Fysiikan periaatteiden mukaan voima (F) voidaan laskea jakamalla massa (m) kiihtyvyydellä (a). Tämä periaate voidaan ilmaista kaavalla  $a = F/m$ . Kiihtyvyysanturi mittaa voimia, jotka kohdistuvat laitteeseen ( $F_s$ ), ja tämä johtaa kuvan 2.1 kaavaan

$$A_D = -\left(\frac{1}{mass}\right) \sum F_s$$

KUVA 2.1. Laitteen kiihtyvyys (Motion sensors 2023).

jossa  $A_D$  on laitteen kiihtyvyys ja  $\sum F_s$  kaikkien voimien summa (x, y, z -akselit).

Kuitenkaan tässä kaavassa ei huomioida maan painovoimaa ( $9,81 \text{ m/s}^2$ ). Siksi korjattu kiihtyvyys, joka ottaa huomioon painovoiman vaikutuksen, voidaan laskea kuvan 2.2 kaavalla.

$$A_D = -g - \left(\frac{1}{mass}\right) \sum F_s$$

KUVA 2.2. Laitteen kiihtyvyys painovoimalla (Motion sensors 2023).

jossa g on maan painovoima.

Paikallaan ollessaan kiihtyvyysanturin kiihtyvyys on juurikin se noin  $9.81 \text{ m/s}^2$  ja vapaassa pudotuksessa  $0 \text{ m/s}^2$ . Painovoiman vaikutuksen eliminointi edellyttää

signaalinkäsittelyä, joka tapahtuu kiihtyvyysdatan suodattamisen avulla. (Motion sensors 2023)

### **2.3 SQL ja SQLite**

Tietokannoissa käytettävä SQL, eli Structured Query Language, on standardoitu kyselykieli, jota käytetään tietojen tallentamiseen ja käsittelyyn relaatiotietokannoissa. Se toimii tapana tallentaa, päivittää, poistaa ja hakea tietoja tietokannasta. SQL on suosittu ja laajalti käytetty kyselykieli erilaisissa sovelluksissa. Se on helppo oppia ja integroituu hyvin eri ohjelmointikieliin. (Amazon Web Services n.d)

SQLite on avoimen lähdekoodin, sisäänrakennettu kevyt tietokantamoottori, joka toimii ilman erillistä palvelinta. Se soveltuu erityisen hyvin älypuhelimien ja muihin resurssirajoitettuihin ympäristöihin, kun käsiteltävät datamäärät pysyvät maltillisina. SQLiteä voidaan käyttää ilmaiseksi kaupallisiin tai yksityisiin tarkoituksiin, ja se on yksi maailman eniten käytetyistä tietokannoista. SQLitellä on maine erittäin luotettavana tietokantamoottorina, jota on tavoite kehittää jopa vuoteen 2050 saakka. (SQLite 2023)

### **2.4 Android-kirjastot**

Android-sovellusten kehittäjät hyödyntävät laajalti valmiiksi kehitettyjä Android-kirjastoja nopeuttaakseen ja tehostaakseen omien sovellustensa kehitystä. Näitä kirjastoja löytyy laajasti erilaisiin käyttötarkoituksiin, tarjoten valmiita ratkaisuja moniin yleisiin ongelmiin ja tehtäviin. JitPack on yksi suosittu palvelu, jonka avulla kehittäjät voivat integroida ulkoisia kirjastoja omiin projekteihinsa helposti.

Android-kirjasto kootaan usein yhteen moduuliin, joka tarjoaa valmiiksi toteutettuja apuvälineitä kehittäjille. Android-kirjastomoduuili on rakenteellisesti samankaltainen kuin sovellusmoduuili eli App-moduuili. Toisin kuin sovellusmoduuili, kirjasto ei käännä itseään APK-tiedostoksi, jota voitaisiin suorittaa laitteella. Sen sijaan se kääntyy AAR-tiedostoksi (Android Archive), joka voidaan lisätä riippuvuudeksi sovellusmoduuleihin. (Android library 2024)

## 2.5 Rakentaminen ja hallinta: Gradle, Maven ja JitPack

Gradle on tehokas avoimen lähdekoodin työkalu, joka auttaa rakentamaan ohjelmistoja. Gradle käyttää yksinkertaista ja helposti laajennettavaa rakennuskieltä, mikä tekee ohjelmistojen rakentamisesta helppoa. Gradle on yleisesti käytetty työkalu JVM (Java Virtual Machine) -pohjaisten projektien rakentamiseen. Myös Android Studioon käännösjärjestelmä perustuu siihen. (Gradle Build Tool 2023)

Myös Maven on laajalti käytetty työkalu Java-pohjaisten projektien hallintaan ja rakentamiseen. Se tarjoaa standardoidun tavan hallita projekteja objektimallin (POM) avulla. Lisäksi se helpottaa riippuvuuksien hallintaa ja automatisoi rakennusprosessin, tarjoten samalla hyödyllistä tietoa automaattisilla raporteilla. (Apache Maven 2024)

JitPack on avoin pakettirepositorio, joka tarjoaa kätevän tavan jakaa ja hyödyntää valmiiksi rakennettuja Java- ja Android-projekteja. Sen avulla kehittäjät voivat jakaa omaa koodiaan GitHubissa, minkä jälkeen JitPack rakentaa julkaisut automaattisesti, tehden valmiit käytettävät artefaktit (jar, aar) kaikkien kehittäjien saataville ympäri maailmaa. (jitpack.io n.d)

Kaikki nämä työkalut ovat yhteensopivia keskenään ja niitä käytetään tässä projektissa eri tarkoituksiin.

### **3 SOVELLUKSEN TOTEUTUS**

Tässä osiossa käydään läpi sovelluksen toteutusvaiheet.

#### **3.1 Modulaarinen suunnittelu ja kehitys**

Koska sovelluksen toiminnallisuuksista luodaan Android-kirjasto avuksi muille ohjelmistokehittäjille, on hyvä miettiä jo sovellusta suunniteltaessa ja kehittäessä, että kuinka siitä saadaan mahdollisimman kompakti ja uudelleenkäytettävä. Se ei pelkästään auta myöhemmin rakentamaan kirjastoa, vaan myös parantamaan ja selkeyttämään sovelluksen koodirakennetta.

Ylipäättään modulaarinen sovelluskehitys tarkoittaa sitä, että sovellus jaetaan eri osiin, ja nämä osat voidaan kehittää erikseen toisistaan ja kutsua tarpeen mukaan. Tämä tekee mahdolliseksi sen, että yksittäisiä osia voidaan päivittää tai vaihtaa tarvittaessa ilman suurempia hankaluuksia. Samalla testaaminen on helpompaa, koska monimutkaisia testausjärjestelmiä ei tarvita.

#### **3.2 Projektin alustus**

Työ alkoi luomalla uuden Android Studio -projektin. Nimeksi valitsin projektilleni ActivityRecognizer ja kieleksi Kotlinin. Seuraavaksi suoritettiin tarvittavat valmistelut projektia varten GitHubissa, kuten README- ja .gitignore-tiedostojen luominen. Näiden vaiheiden jälkeen kaikki oli valmista kehitystyön aloittamiseksi.

#### **3.3 Sensor API**

Ensin sovellukselle annetaan käyttöoikeus kiihtyvyysanturille lisäämällä se manifest.xml -tiedostoon. Manifest.xml on Android-sovelluksen pakollinen tiedosto, joka sisältää tärkeää tietoa sovelluksesta, kuten sen komponentit, tarvittavat luvat ja laitteistovaatimukset (App manifest overview 2023). Kun sovelluksella on oikeus käyttää kiihtyvyysanturia, voidaan se hakea käyttöön. Tässä vaiheessa luodaan uusi luokka Accelerometer (Kuva 3.1). Luokkaan luodaan kaksi tärkeää muuttujaa: sensorManager, jolla pääsee käsiksi ja

hallitsemaan laitteen sensoreita, sekä accelerometer, joka haetaan sensorManagerilta ja asetetaan kiihtyvyysanturiksi, mikäli laite sellaisen löytää.

```
class Accelerometer(context: Context) {  
  
    private val sensorManager: SensorManager = context.getSystemService(Context.SENSOR_SERVICE) as  
        SensorManager  
    private var accelerometer: Sensor? = null  
  
    init {  
        accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)  
    }  
}
```

KUVA 3.1. Accelerometerin alustus.

Koska tällä hetkellä luokka ei tee muuta kuin alustaa olion, tarvitsee meidän tuoda työkalut, jotta saamme kiihtyvyysanturilta dataa. Ensiksi toteutetaan SensorEventListener-rajapinta Accelerometeriin, jolta löytyy tarvittava onSensorChanged-funktio, joka siis nimensä mukaan kuuntelee, jos rekisteröidyssä sensorissa tapahtuu muutoksia. Kuitenkin kuuntelija on rekisteröitävä, joka onnistuu sensorManagerin registerListener-funktiota käyttämällä.

RegisterListener-funktiolle annetaan 3 parametria, SensorEventListener-kuuntelija, rekisteröitävä sensori sekä sensorin näytteenottotaajuus, eli kuinka usein dataa saapuu sensorilta. Tämän projektin käyttötarkoituksiin sopiva taajuus on SENSOR\_DELAY\_UI, joka on noin 15 lukemaa sekunnissa. Tämä taajuus valittiin, koska se tarjoaa riittävän tiheyden tunnistamiseen, samalla kuitenkin vaikuttamatta merkittävästi sovelluksen suorituskykyyn.

Kun kiihtyvyysanturin alustus ja rekisteröinti on tehty, dataa alkaa virrata onSensorChanged-funktiolle SensorEvent-parametrin values-muuttujaan FloatArrayna (liukulukutaulukkona), jossa on kiihtyvyysanturin x-, y- ja z- akselien kiihtyvyydet. Nyt tarvitsee vain saada tuo data aina muutoksen tapahtuessa activityjen käytettäväksi. Kuvassa 3.2 onSensorChanged-funktion ja rekisteröintifunktioiden toteutuksia koodissa.

```

override fun onSensorChanged(event: SensorEvent?) {

    // If true, clone it to the array and send it to the accelerometer listener
    if (event?.sensor?.type == Sensor.TYPE_ACCELEROMETER) {
        acceleration = event.values.clone()
        listener?.onAccelerationChanged(acceleration)
    }
}

// Register the accelerometer listener when called
fun register(listener: AccelerometerListener) {
    this.listener = listener
    sensorManager.registerListener( listener: this, accelerometer, SensorManager.SENSOR_DELAY_UI)
}

// Unregister the accelerometer listener when no focus
fun unregister() {
    sensorManager.unregisterListener( listener: this)
}

```

KUVA 3.2. Kuuntelijan rekisteröinti sekä onSensorChanged-funktio

Activityt siis ovat Android-sovelluksissa näkymiä, jotka mahdollistavat vuorovaikutuksen sovelluksen kanssa. Tässä työssä activityt viittaavat näihin komponentteihin. Jotta activityt voivat käyttää kiihtyvyysanturin dataa, se onnistuu käyttämällä rajapintaa.

Luodaan AccelerometerListener-rajapinta (Kuva 3.3), jolle määritellään onAccelerationChanged-funktio, joka saa parametrikseen kiihtyvyysanturin datan. Nyt activityn täytyy toteuttaa AccelerometerListener-rajapinta, mikä tarkoittaa, että activityn täytyy sisältää kaikki ne toiminnot, jotka määritellään AccelerometerListener-rajapinnassaja. Lisäksi täytyy yliajaa (override) onAccelerationChanged-funktio.

```

interface AccelerometerListener {
    fun onAccelerationChanged(acceleration: FloatArray)
}

```

KUVA 3.3. AccelerometerListener-rajapinta

OnResume-funktiossa, joka kutsuu Accelerometerin rekisteröintifunktiota, meidän täytyy antaa rekisteröintiin parametrina konteksti, eli tässä tapauksessa "this". Tämä "this" viittaa siis nykyiseen Activity-olioon, joka suorittaa toiminnon. Näin saamme datan käyttöömmme myös kiihtyvyysanturiluokan ulkopuolella.

Kuvassa 3.4 näemme miten kiihtyvyysanturin rekisteröinnit sekä kiihtyvyysdatan vastaanottaminen ovat rakennettu sovelluksen activityyn.

```
class MainActivity : AppCompatActivity(), AccelerometerListener {

    private lateinit var accelerometer: Accelerometer
    private lateinit var acceleration: FloatArray

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        this.accelerometer = Accelerometer(context: this)
        accelerometer.register(listener: this)
    }

    override fun onAccelerationChanged(acceleration: FloatArray) {
        this.acceleration = acceleration
    }

    override fun onResume() {
        super.onResume()
        accelerometer.register(listener: this)
    }

    override fun onPause() {
        super.onPause()
        accelerometer.unregister()
    }
}
```

KUVA 3.4. Kiihtyvyysanturin käyttöönotto activitystä sekä kiihtyvyysanturin datan tallentaminen liukulukutaulukkoon.

### 3.4 Signaalin suodatus

Kiihtyvyysdatan suodattamiseksi hyödynsin Androidin kehittäjä sivujen dokumentaatiosta löytyvää valmista koodia, jota muokkasin omiin tarpeisiini sopivaksi (Motion Sensors 2023).

Funktio suodattaa kiihtyvyysanturin dataa alipäästö- ja korkeapäästösuotimilla, kuten esitetty kuvassa 3.5. Alipäästösuodattimen avulla erotetaan kiihtyvyysarvoista painovoima, ja tämä painovoiman arvo tallennetaan



luokkamuuttuja gravityn taulukon indekseihin. Tämän jälkeen akselien kiihtyvyysarvoista vähennetään suodatettu painovoima, jotta saadaan poistettua painovoiman vaikutus kiihtyvyyssmittauksista. Lopuksi suodatettu kiihtyvyysarvo palautetaan.

Painovoiman erottamiseen käytetty alfa-muuttuja, eli suodatuskerroin, määrittää kuinka voimakkaasti suodatin reagoi uusiin kiihtyvyysarvoihin suhteessa aiempiin arvoihin. Mitä suurempi alfa on, sitä voimakkaampi reagointi on, ja päinvastoin.

Testasin kiihtyvyysanturin dataa eri alfa-arvoilla sovelluksen myöhemmissä vaiheissa ja lopulta päätin käyttää arvoa 0.95 lopullisessa toteutuksessa. Tämä valinta perustuu siihen, että liian pieni alfa ei reagoanut riittävän nopeasti muutoksiin, kun taas sopivaksi testaamani 0.95-arvo mahdollistaa herkän reagoinnin, joka sopii tämän projektin tarpeisiin.

```
// Filter accelerometer data with low- and high-pass filters
fun filter(acceleration: FloatArray): FloatArray {
    val alpha: Float = 0.95f

    // Low-pass, isolates the force of gravity
    gravity[0] = alpha * gravity[0] + (1 - alpha) * acceleration[0]
    gravity[1] = alpha * gravity[1] + (1 - alpha) * acceleration[1]
    gravity[2] = alpha * gravity[2] + (1 - alpha) * acceleration[2]

    // High-pass, removes the gravity contribution
    acceleration[0] = acceleration[0] - gravity[0]
    acceleration[1] = acceleration[1] - gravity[1]
    acceleration[2] = acceleration[2] - gravity[2]

    return acceleration
}
```

KUVA 3.5. Kiihtyvyysanturin datan suodatinfunktio.

### 3.5 Kiihtyvyysdatan visualisointi

Nyt kun saadaan kiihtyvyysdataa, olisi hyödyllistä visualisoida se jollakin tapaa.

Kokeilin aluksi luoda graafin käyttämällä Androidin sisäisiä komponentteja. Loin uuden toteutuksen, joka perii View-luokan. Koodissa määrittelin erilaisia maalaukseen liittyviä ominaisuuksia, kuten värit ja viivapaksuudet, ja käytin Canvas-objektia graafisten elementtien piirtämiseen. Huomasin kuitenkin tämän olevan melko monimutkainen ja vaivalloinen tapa. Tämän vuoksi päätin hylätä oman toteutukseni ja valitsin GraphView-kirjaston, joka tarjoaa valmiin ratkaisun tähän ja joka vaikutti melko helppokäyttöiseltä kirjastolta.

GraphView on kirjasto Androidille, jonka avulla voi luoda ohjelmallisesti erilaisia graafeja. Se on helppo ymmärtää, integroida ja mukauttaa (jjoe64, n.d). Muita yleisessä käytössä olevia vaihtoehtoja olisivat olleet muun muassa MPAndroidChart sekä AnyChart. Koska GraphView on ulkopuolinen kirjasto, eikä siis osa Androidin omia kirjastoja, jätin sen pois kirjastototeutuksesta.

Ensin luodaan uusi GraphActivity-luokka. Kun ollaan GraphActivityssa niin onAccelerationChanged-funktio saa kiihtyvyysanturin dataa, joka välitetään suodatettuna AccelerometerGraphView-luokalle (Kuva 3.6), joka hoitaa tämän datan visuaalisen esittämisen GraphViewin avulla.

```
// Filter data and refresh the accelerationGraphView with new values
override fun onAccelerationChanged(acceleration: FloatArray) {
    accelerometer.filter(acceleration)
    accelerationGraphView.update(acceleration[0], acceleration[1], acceleration[2])
}
```

KUVA 3.6. onAccelerationChanged-funktio välittää suodatetun kiihtyvyysanturin datan GraphActivityyn.

Kun GraphActivitysta luodaan olio, niin sinne alustetaan näkymän ominaisuuksia, kuten värit ja otsikot, sekä asetetaan graafin akselien rajat. Lisäksi luodaan graafin datapisteitä edustavat sarjat (series) x, y ja z -akseleille.

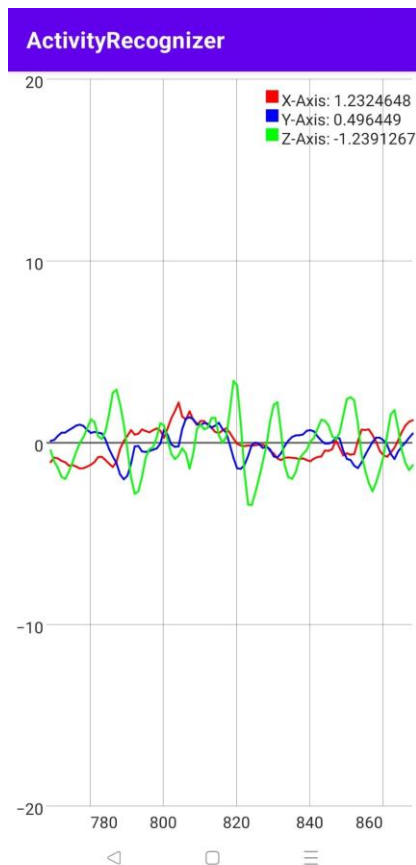
Kuvassa 3.7 nähdään update-funktio, jota kutsutaan aina uusilla kiihtyvyysanturin arvoilla, kun sitä saadaan. Arvot lisätään datapisteiksi graafiin, jolloin se päivittyy. GraphView huolehtii automaattisesti kaiken niin, että näytöllä näytetään aina viimeisimmät datapisteet. Arvot myös päivitetään oman akselinsa nimen viereen

graafin ylänurkassa. Jotta graafi pysyisi suorituskykyisenä, datapisteiden sarjat poistetaan ja lisätään uudelleen joka sadas päivityskerta.

```
fun update(x: Float, y: Float, z: Float) {  
  
    // Add the datapoints to the series  
    seriesX.appendData(DataPoint(lastX.toDouble(), x.toDouble()), scrollToEnd: true, maxDataPoints: 100)  
    seriesY.appendData(DataPoint(lastX.toDouble(), y.toDouble()), scrollToEnd: true, maxDataPoints: 100)  
    seriesZ.appendData(DataPoint(lastX.toDouble(), z.toDouble()), scrollToEnd: true, maxDataPoints: 100)  
    lastX++  
  
    seriesX.title = "X-Axis: $x"  
    seriesY.title = "Y-Axis: $y"  
    seriesZ.title = "Z-Axis: $z"  
  
    // Sets the minimum X-value to be 100 before the maximum X-value to ensure that the  
    // graph shows the most recent data.  
    val maxX = seriesX.highestValueX.coerceAtMost(seriesY.highestValueX)  
        .coerceAtMost(seriesZ.highestValueX)  
    val minX = maxX - 100.0  
    viewport.setMinX(minX)  
    viewport.setMaxX(maxX)  
  
    // Removes all datapoint series  
    if (lastX % 100 == 0) {  
        graphView.removeAllSeries()  
        graphView.addSeries(seriesX)  
        graphView.addSeries(seriesY)  
        graphView.addSeries(seriesZ)  
    }  
}
```

KUVA 3.7. Graafin päivitys.

Kuvassa 3.8 nähdään kuinka GraphView sekä kiihtyvyysanturin x, y ja z -akselien muutokset näkyvät käyttäjälle.



KUVA 3.8. X, y ja z -akselien reaaliaikaiset muutokset GraphView-graafissa kävellessä puhelin kädessä.

### 3.6 Tietokannan suunnittelu ja toteutus

Tietokannan suunnittelu ja toteutus ovat keskeinen osa sovelluksen kehitysprosessia. Tässä osiossa esitellään sovelluksen käyttämää tietokantaratkaisua.

#### 3.6.1 Tietokannan rakenne

Tietokantaan liittyvät toiminnot on toteutettu TrainingDbHelper-luokassa, joka periytyy SQLiteOpenHelper-luokasta. Tämä luokka vastaa tietokannan luomisesta ja päivittämisestä (onCreate, onUpgrade). Tarvittaessa tästä löytyy myös funktio kaikkien taulujen poistamiseen. Tietokannan nimi on Training.db ja sen versiota seurataan muuttujalla DATABASE\_VERSION, jotka ovat companion objectin sisällä (Kuva 3.9). Companion object mahdollistaa pääsyn luokan muuttujiin ja funktioihin ilman olion luomista (Object expressions and... n.d). Tämä on käytännössä sama kuin Javan Static.

```

class TrainingDbHelper(context: Context) : SQLiteOpenHelper(context, DATABASE_NAME, factory: null, DATABASE_VERSION) {

    override fun onCreate(db: SQLiteDatabase) {
        db.execSQL(TrainingContract.ActivityEntry.SQL_CREATE_TABLE)
        db.execSQL(TrainingContract.TrainingDataEntry.SQL_CREATE_TABLE)
    }

    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
        db.execSQL(TrainingContract.TrainingDataEntry.SQL_DROP_TABLE)
        db.execSQL(TrainingContract.ActivityEntry.SQL_DROP_TABLE)
        onCreate(db)
    }

    // Be careful with this function
    fun deleteAllTables() {
        val db = writableDatabase
        db.execSQL(TrainingContract.TrainingDataEntry.SQL_DROP_TABLE)
        db.execSQL(TrainingContract.ActivityEntry.SQL_DROP_TABLE)
        onCreate(db)
        db.close()
    }

    companion object {
        const val DATABASE_VERSION = 2
        const val DATABASE_NAME = "Training.db"
    }
}

```

KUVA 3.9. TrainindDbHelper-luokka.

Tietokanta sisältää kaksi taulua: training\_data ja activity, jotka molemmat luodaan tietokannan luomisen yhteydessä. Tauluun training\_data tulee jokaisen tallennettavan näytteen datat ja activity-tiluun aktiviteetin nimi ja näytteiden lukumäärä. Kummallakin taululla on omat rakenteensa, jotka on määritelty TrainingContract-objektissa. Kuvassa 3.10 nähdään TrainingContract objekti, josta näkee molempien taulujen määritellyt rakenteet ja niihin liittyvät SQL-kyselyt.

```

object TrainingContract {
  object TrainingDataEntry {
    const val TABLE_NAME = "training_data"
    const val COLUMN_NAME_X_AXIS = "x_axis"
    const val COLUMN_NAME_Y_AXIS = "y_axis"
    const val COLUMN_NAME_Z_AXIS = "z_axis"
    const val COLUMN_NAME_TIMESTAMP = "timestamp"
    const val COLUMN_NAME_ACTIVITY_ID = "activity_id"
    const val COLUMN_NAME_TOTAL_ACCELERATION = "total_acceleration"

    const val SQL_CREATE_TABLE = "CREATE TABLE ${TrainingDataEntry.TABLE_NAME} (" +
      "${BaseColumns._ID} INTEGER PRIMARY KEY, " +
      "${TrainingDataEntry.COLUMN_NAME_TIMESTAMP} LONG DEFAULT 0, " +
      "${TrainingDataEntry.COLUMN_NAME_X_AXIS} REAL, " +
      "${TrainingDataEntry.COLUMN_NAME_Y_AXIS} REAL, " +
      "${TrainingDataEntry.COLUMN_NAME_Z_AXIS} REAL, " +
      "${TrainingDataEntry.COLUMN_NAME_TOTAL_ACCELERATION} REAL DEFAULT 0, " +
      "${TrainingDataEntry.COLUMN_NAME_ACTIVITY_ID} REAL, " +
      "FOREIGN KEY(${TrainingDataEntry.COLUMN_NAME_ACTIVITY_ID}) REFERENCES " +
      "${ActivityEntry.TABLE_NAME}(${BaseColumns._ID}))"

    const val SQL_DROP_TABLE = "DROP TABLE IF EXISTS ${TrainingDataEntry.TABLE_NAME}"
  }

  object ActivityEntry {
    const val TABLE_NAME = "activity"
    const val COLUMN_NAME_ACTIVITY = "activity_name"
    const val COLUMN_NAME_SAMPLES = "samples"
    const val COLUMN_NAME_AVERAGE_SPEED = "average_speed"

    const val SQL_CREATE_TABLE = "CREATE TABLE ${ActivityEntry.TABLE_NAME} (" +
      "${BaseColumns._ID} INTEGER PRIMARY KEY, " +
      "${ActivityEntry.COLUMN_NAME_ACTIVITY} TEXT, " +
      "${ActivityEntry.COLUMN_NAME_SAMPLES} INTEGER DEFAULT 0)"

    const val SQL_DROP_TABLE = "DROP TABLE IF EXISTS ${ActivityEntry.TABLE_NAME}"
  }
}

```

KUVA 3.10. TrainingContract-objekti.

Seuraavassa selitettynä training\_data -taulun (Taulukko 3.1) sekä activity-taulun (Taulukko 3.2) sarakkeet:

TAULUKKO 3.1. Taulu training\_data.

Sarake	Tietotyyppi	Selitys
_id	Kokonaisluku	Uniikki tunniste
timestamp	Kokonaisluku (pitkä)	Ajanhetki, jolloin data kerätty
x_axis	Liukuluku	Kiihtyvyyden x-akselin arvo
y_axis	Liukuluku	Kiihtyvyyden y-akselin arvo
z_axis	Liukuluku	Kiihtyvyyden z-akselin arvo
total_acceleration	Liukuluku	Kokonaiskiihtyvyys ( $\sqrt{x^2+y^2+z^2}$ )
activity_id	Kokonaisluku	Viittaus activity-tiluun vierasavaimella

TAULUKKO 3.2. Taulu activity.

Sarake	Tietotyyppi	Selitys
_id	Kokonaisluku	Uniikki tunniste
activity_name	Teksti	Aktiviteetin nimi
samples	Kokonaisluku	Näytteiden lukumäärä

Vierasavain tarkoittaa viittausta toisen taulun pääavaimen arvoon. Se mahdollistaa tietojen yhdistämisen ja suhteiden muodostamisen eri taulujen välillä tietokannassa. Molemmissa id-kentissä käytetty BaseColumns on Androidin rajapinta, joka sisältää tietokannan perussarakkeiden nimet, kuten \_ID. Monet Androidin toiminnot odottavat automaattisesti generoitua \_ID:tä, joten sen käyttö on suositeltavaa. (Save data using... 2024)

### 3.6.2 Tiedon tallennus ja haku

Tiedon tallennusta ja hakemista varten luotiin ActivityDao-luokka (Data Access Object), joka toimii välittäjänä sovelluksen ja tietokannan kanssa (Kuva 3.11). ActivityDaon luotiin runsaasti toimintoja, joilla voi esimerkiksi tallentaa, hakea sekä poistaa aktiviteetteja ja näytteitä. Projektissa kutsutaankin juuri tämän ActivityDao-luokan funktioita vaadituilla parametreilla.

```

fun saveData(accelerations: MutableList<FloatArray>, activityId: Long) {

    lateinit var values: ContentValues
    val db = dbHelper.writableDatabase

    for (acc in accelerations) {
        val x = acc[0].toDouble()
        val y = acc[1].toDouble()
        val z = acc[2].toDouble()

        val totalAcc = sqrt(x*x + y*y + z*z)

        values = ContentValues().apply { this: ContentValues
            put(trainingDataEntry.COLUMN_NAME_TIMESTAMP, System.currentTimeMillis())
            put(trainingDataEntry.COLUMN_NAME_X_AXIS, acc[0])
            put(trainingDataEntry.COLUMN_NAME_Y_AXIS, acc[1])
            put(trainingDataEntry.COLUMN_NAME_Z_AXIS, acc[2])
            put(trainingDataEntry.COLUMN_NAME_TOTAL_ACCELERATION, totalAcc)
            put(trainingDataEntry.COLUMN_NAME_ACTIVITY_ID, activityId)
        }
        if (values.size() > 0) {
            db.insert(trainingDataEntry.TABLE_NAME, nullColumnHack: null, values)
        }
    }
    db.close()
}

```

KUVA 3.11. Esimerkki kiihtyvyyssanturin datan tallentamisesta ja vuorovaikutuksesta tietokantaan ActivityDao-luokassa.

### 3.7 Treenidatan kerääminen

Nyt kun tietokanta on olemassa, niin voidaan alkaa keräämään sinne käyttäjän suorittamien aktiviteettien dataa. Päätin, että kerättävää dataa kutsutaan treenidataksi. Ensiksi luotiin uusi TrainActivity, jossa dataa olisi tarkoitus kerätä. Kun MainActivity:sta siirrytään TrainActivityyn, niin silloin aukeaa dialogivalikko, joka hoitaa tarvittavat kysymykset käyttäjältä.

#### 3.7.1 Dialogi

Jotta käyttäjältä saataisiin kaikki tarvittavat tiedot datan keräykseen, käytettiin tähän tarkoitukseen AlertDialogia. AlertDialog on pieni ikkuna, joka näyttää viestin ja voi sisältää eri toimintopainikkeita. Se on modaalinen, eli se ei peitä koko näyttöä, ja yleensä se näkyy muiden näkymien päällä. AlertDialog on



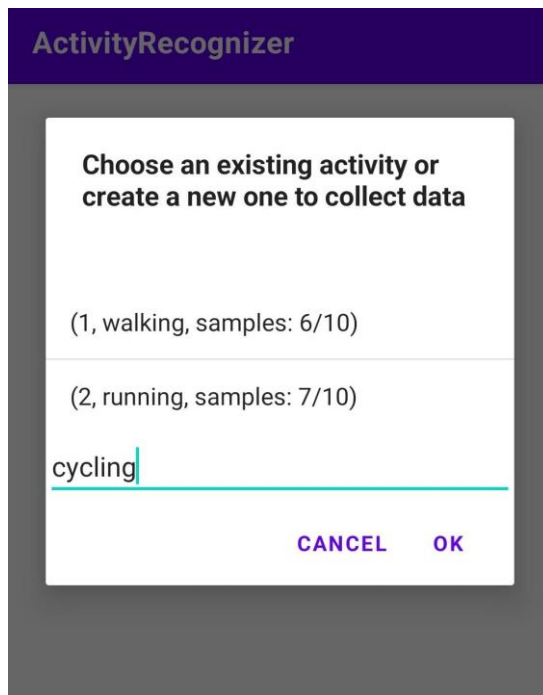
yleinen käyttöliittymäelementti, ja monet sovellukset käyttävät sitä viestien näyttämiseen. (Waldo 2021)

Dialogeissa käytetään lambda-funktioita, jotka lyhyesti sanottuna ovat nimettämiä funktioita, joita esimerkiksi voidaan välittää toisille funktiolle tai tallentaa muuttujiin. Ensimmäinen dialogi kysyy, haluaako käyttäjä tallentaa treenidataa (Kuva 3.12). Kun käyttäjä painaa "Yes" tai "No", kutsutaan vastaavasti onPositiveButtonClick- tai onNegativeButtonClick-funktiota. Tällä tavoin toteutetaan toiminnallisuus, joka on määritelty TrainActivityssa, jolloin sieltä pääsee suoraan muokkaamaan käyttöliittymää.

```
class DialogManager(private val context: Context) {  
  
    private val builder = AlertDialog.Builder(context)  
    private val activityDao = ActivityDao(TrainingDbHelper(context))  
    private val updatedActivities = mutableListOf<Triple<Long, String, String>>()  
    private lateinit var adapter: ArrayAdapter<Triple<Long, String, String>>  
  
    // Open the first dialog when the activity starts  
    fun showActivityDialog(title: String, message: String, onPositiveButtonClick: () -> Unit, onNegativeButtonClick: () -> Unit) {  
  
        //startActivity(intent)  
        builder.setTitle(title).setMessage(message)  
        builder.setPositiveButton(text: "Yes") { dialog, which ->  
            onPositiveButtonClick()  
            dialog.dismiss()  
        }  
        builder.setNegativeButton(text: "No") { dialog, which ->  
            onNegativeButtonClick()  
            dialog.dismiss()  
        }  
  
        val dialog = builder.create()  
        dialog.show()  
    }  
}
```

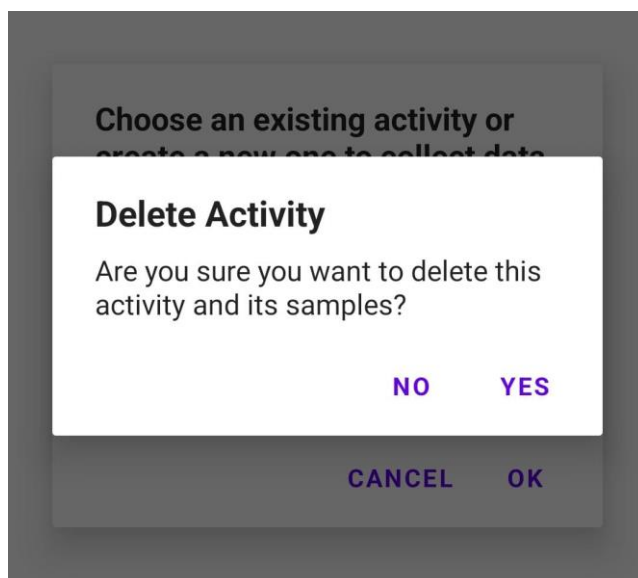
KUVA 3.12. Esimerkki ensimmäisen dialogin funktiosta ja DialogManager-luokasta.

Painamalla kyllä, käyttäjälle ilmestyy uusi dialogi, jossa annetaan käyttäjälle mahdollisuus joko luoda uusi aktiviteetti, tai käyttää jo luotua aktiviteettia (Kuva 3.13).



KUVA 3.13. Aktiviteetin luonti/valintaruutu.

Nämä tiedot haetaan sekä tallennetaan tietokannasta käyttämällä ActivityDao-luokan apufunktioita. Tässä näytetään myös kerättyjen näytteiden kokonaismäärä ja suositeltava vähimmäismäärä (x/10). Jos olemassa olevaa aktiviteettia painaa pohjassa, niin aukeaa uusi dialogi-ikkuna, joka kysyy, haluaako käyttäjä poistaa aktiviteetin ja sen kaikki näytteet tietokannasta (Kuva 3.14).



KUVA 3.14. Aktiviteetin ja näytteiden poistaminen.

Kuva 3.15 näyttää kuinka koko dialogiprosessi on yhden funktion alla TrainActivityssa ja kuinka lambda-funktiot ohjaavat toimintoja. Tämä funktio siis hallinnoi dialogeja käyttäjän valitessa, haluaako hän aloittaa datan keräämisen vai ei.

```
private fun activityChoiseDialogHandler() {
    val dialogManager = DialogManager(context: this)
    dialogManager.showActivityDialog(
        title: "Create Training Data",
        message: "Would you like to start taking the training data?",
    ) {
        dialogManager.showNameDialog(
            title: "Choose an existing activity or create a new one to collect data",
        ) { activityName ->
            if (activityName != null && activityName.isNotEmpty()) {
                activityId = activityDao.saveActivityAndGetId(activityName)
                showTempActivity.text = "$activityName"
                startBtn.visibility = View.VISIBLE
            }
        },
        { activityId, activityName ->
            if (activityId != null && activityName != null) {
                this.activityName = activityName
                this.activityId = activityId
                showTempActivity.text = "${activityName}"
                startBtn.visibility = View.VISIBLE
            } else {
                this.activityName = null
                this.activityId = null
            }
        }, {
            val intent = Intent(packageContext: this, MainActivity::class.java)
            startActivity(intent)
            finish()
        }
    )
}, {
    val intent = Intent(packageContext: this, MainActivity::class.java)
    startActivity(intent)
    finish()
}
```

KUVA 3.15. Koko dialogiprosessi TrainActivitysta, jossa käyttäjä valitsee mistä aktiviteetista lähtee ottamaan näytettä.

### 3.7.2 Valitun aktiviteetin näytteenotto

Nyt kun käyttäjä on luonut tai valinnut aktiviteetin, on aika kerätä näyte. Sovellus näyttää nyt aktiviteetin nimen, johon uusi näyte tullaan lisäämään, sekä napin, josta näytteenotto alkaa (Kuva 3.16).

walking



KUVA 3.16. Näytteenoton aloitusruutu.

Kun käyttäjä painaa start-nappia, käynnistyy 10 sekunnin aika, jolloin dataa kerätään taulukkoon. Tämän teen uudessa Threadissa eli säikeessä. Säikeet ovat erillisiä kevyitä suorituspolkuja, joita voidaan suorittaa samanaikaisesti muiden säikeiden kanssa. Käyttöjärjestelmä huolehtii säikeiden luomisesta ja hallinnasta, mikä mahdollistaa niiden yhteistyön ja tehokkaan toiminnan yhden ohjelman sisällä. (Geeks for Geeks 2023)

Kun säie on luotu, kasvatetaan muuttujan  $x$  arvoa yhdellä aina 1000 millisekunnin (1 sekunti) välein kunnes  $x$  on 10. Pääsäie (main thread) vastaa käyttöliittymän muutoksista, joten emme voi luomastamme säikeestä suoraan muokata käyttöliittymää. Siksi käyttöliittymän muutokset suoritetaankin `runOnUiThread`-funktion sisällä, joka suorittaa koodin pääsäikeessä (Kuva 3.17).

```

fun startClicked() {
    startBtn.visibility = View.GONE
    showTempActivity.textSize = 17f
    Thread {
        var x = 0
        accelerometer.register( listener: this)
        while (x < 10) {
            runOnUiThread() {
                showTimer.text = (x + 1).toString()
            }
            Thread.sleep( millis: 1000)
            x++
        }
        accelerometer.unregister()
        runOnUiThread() {
            showTimer.textSize = 50F
            showTimer.text = "Completed"
            saveDataDialogHandler()
        }
    }.start()
}

```

KUVA 3.17. Ajastimen ja säikeen toiminnot.

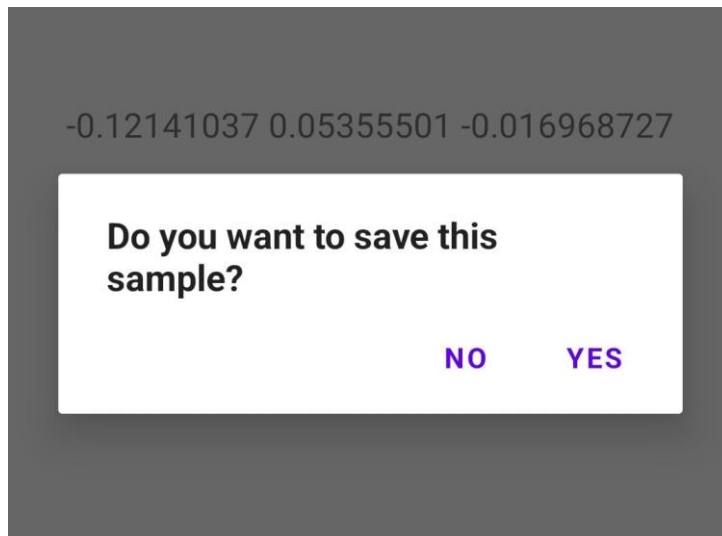
Tässä siis näytämme käyttäjälle kiihtyvyysanturin x, y ja z -akselien reaaliaikaiset arvot sekä ajastimen ajan, joka käy siis yhdestä kymmeneen (Kuva 3.18).

-0.20048442 0.28401637 -0.4729166

9

KUVA 3.18. Kiihtyvyysanturin sekuntikello ja x, y ja z -akselit.

Kun kymmenen sekuntia on kulunut, aukeaa jälleen uusi dialogi, jossa kysytään käyttäjältä, haluaako hän tallentaa datan, eli onko näytteenotto onnistunut (Kuva 3.19).



KUVA 3.19. Dialogi tallentamisesta.

Painaessaan "Yes", sovellus tallentaa juuri kerätyt datat tietokantaan. Lopuksi käyttäjälle avautuu näkymä, jossa ilmoitetaan näytteenoton olevan valmis, näytteiden lukumäärä, sekä nappi, josta voi halutessaan ottaa uuden näytteen samalle aktiviteetille (Kuva 3.20).

-0.12141037 0.05355501 -0.016968727

# Completed

Total samples 9/10

RETAKE?

KUVA 3.20. Näyte otettu ja tallennettu.

### **3.8 Reaaliaikaisen datan vertailu testidataan**

Nyt kun tietokannasta löytyy eri aktiviteetteja ja jokaiselle on otettu tarpeeksi näytteitä, voidaan alkaa vertaamaan niitä reaaliaikaiseen dataan ja pyrkimään löytää käyttäjän parhaillaan tekemä aktiviteetti.

Olen toteuttanut projektissa hyvin yksinkertaisen tavan. Tunnistukseen olisi paljon erilaisia vaihtoehtoja mm. koneoppiminen ja neuroverkot, jotka antaisivat paljon parempia ja luotettavampia tuloksia. Toteutin kuitenkin tämän yksinkertaisen tavan, koska parempien vaihtoehtojen opiskelu ja näiden monimutkaisten ratkaisujen integroiminen projektiin olisivat vaatineet merkittävästi enemmän aikaa, ja halusin pitää projektin työmäärältään sekä ajallisesti siedettävänä.

#### **3.8.1 Esikäsittely**

Ensiksi luodaan uusi RecognizeActivity, joka toteuttaa AccelerometerListener-rajapinnan. Loin myös luokan nimeltä DataCompare (Kuva 3.21), jossa siis reaaliaikaista dataa ja tietokannan dataa vertaamalla pyritään tunnistamaan käyttäjän tekemä aktiviteetti. Jotta suorituskyky olisi taattu aina RecognizeActivityn käynnistyessä, on suotavaa suorittaa esikäsittely vertailtavalle datalle tietokannassa. onCreate-funktioon luotiin kutsu DataCompare-luokan preprocessing-funktioon, joka siis esikäsittelee tarvittavat datat DataComparen luokkamuuttujiin, jotta vertaaminen reaaliaikaiseen dataan olisi mahdollisimman nopeaa.

```

class DataCompare(private val context: Context) {

    private val activityDao = ActivityDao(TrainingDbHelper(context))
    // id, name and total acceleration average for every activity
    private val activityAverageTotalAccList = mutableListOf<Triple<Long, String, Double>>()
    private var thresholdAcceleration = 0.0
    private val stillThreshold = 0.1

    fun preprocessing() {

        val activities = activityDao.getActivitiesList()

        for (activity in activities) {
            val activityId = activity.first
            val activityName = activity.second

            val samples = activityDao.getAllTrainingDataForActivity(activityId)

            if (samples.isNotEmpty()) {

                val x = samples.map { it.x_axis.toDouble() }
                val y = samples.map { it.y_axis.toDouble() }
                val z = samples.map { it.z_axis.toDouble() }

                val averageTotalAcc = AccelerationUtils.calculateAverageTotalAcceleration(x,y,z)
                activityAverageTotalAccList.add(Triple(activityId,activityName, averageTotalAcc))
            }

            // Creates thresholds
            if (activityAverageTotalAccList.isNotEmpty()) {
                val maxAverage = activityAverageTotalAccList.maxByOrNull { it.third }!!.third
                val minAverage = activityAverageTotalAccList.minByOrNull { it.third }!!.third

                thresholdAcceleration = (maxAverage - minAverage) / activityAverageTotalAccList.size
            }
        }
    }
}

```

KUVA 3.21. DataCompare-luokan preprocessing-funktio, joka suoritetaan aina kun RecognizeActivity käynnistyy.

Kuvassa 3.21 nähdään, kuinka activityDao:ta apuna käyttäen (vain sovelluksessa) ensin haetaan kaikki aktiviteetit tietokannasta, jonka jälkeen jokaisen aktiviteetin jokaisen näytteen. Tämän jälkeen tietokannasta saadut x, y ja z -akselien kiihtyvyydatat lähetetään listoina funktiolle, joka laskee ja palauttaa näiden kokonaiskiihtyvyydet. Kun kokonaiskiihtyvyyden keskiarvot on lisätty jokaisesta aktiviteetista activityAverageTotalAccList-listaan, etsitään sen suurin (maxAverage) ja pienin (minAverage) kiihtyvyyden keskiarvo kolmannen komponentin perusteella (.third). Tämä kolmas komponentti on siis kokonaiskiihtyvyyden keskiarvo, kun ensimmäinen on id ja toinen on aktiviteetin nimi.



Määritellään threshold (kynnys) luokkamuuttuja. Se toimii rajana sille, milloin aktiviteetin tunnistus siirtyy seuraavaan aktiviteettiin, eli kun kokonaiskiihtyvyyden keskiarvo ylittää tämän rajan ylös- tai alaspäin. Laskin threshold-arvon vähentämällä suurimman kokonaiskiihtyvyyden keskiarvon pienimmällä ja jakamalla sen kaikkien aktiviteettien lukumäärällä.

### **3.8.2 Tunnistaminen**

Kiihtyvyysanturi rekisteröidään samantien kun activity aukeaa, jolloin onAccelerationChanged-funktioon virtaa kiihtyvyysdataa. Ensiksi kiihtyvyysdata suodatetaan. Tämän jälkeen lisätään suodatettu kiihtyvyysdata dynaamiseen liukulukutaulukoiden listaan. Kun listassa on 40 kpl kiihtyvyysanturin dataa, map-toiminto käy läpi currentData-listan ja tekee jokaisesta juuri kerätystä akselin kiihtyvyysdatasta Double-listan. Nyt jokainen x, y ja z sisältää 40 kpl kiihtyvyysdataa, jotka laitetaan eteenpäin funktiolle, joka laskee ja palauttaa näiden kokonaiskiihtyvyyksien keskiarvon. Koska nyt siis lasketaan tuon 40 kappaleen kiihtyvyyssatojen keskiarvo ja sitä verrataan tietokannan datoihin, niin tämä on samalla tunnistuksen päivitysnopeus. Ottamalla isomman määrän dataa, heilahtelut tasoittuvat ja tulos on varmempi, mutta liikaa ottamalla kerralla tunnistus hidastuu. Kuvassa 3.22 toteutus.

```

override fun onAccelerationChanged(acceleration: FloatArray) {

    val accData = accelerometer.filter(acceleration)

    currentData.add(
        floatArrayOf(accData[0], accData[1], accData[2])
    )

    if (currentData.size == 40) {
        val x = currentData.map { it[0].toDouble() }
        val y = currentData.map { it[1].toDouble() }
        val z = currentData.map { it[2].toDouble() }

        val avrgTotalAccReal = AccelerationUtils.calculateAverageTotalAcceleration(x,y,z)

        comparison.compareDataAverages(avrgTotalAccReal) { activityName, activityTotAcc ->

            val activityInfo = if (activityName != null && activityName != "still") {
                "You are $activityName\n(${String.format("%.3f", activityTotAcc)})\n"
            } else if (activityName == "still") {
                "still\n"
            } else {
                ""
            }

            val formattedReal = "%.3f".format(avrgTotalAccReal)
            currentActivity!!.text = "$activityInfo\nReal time total acceleration\n$formattedReal"
        }
        currentData.clear()
    }
}

```

KUVA 3.22. Funktio onAccelerationChanged RecognizeActivityssä.

Kuten kuvista näkyy, tunnistaminen tapahtuu compareDataAverages-funktion avulla, joka saa parametrikseen reaaliaikaisesti kerättyjen 40 kappaleen kokonaiskiihtyvyyksien keskiarvon sekä lambda-funktion. Kuvassa 3.23 compareDataAverages kokonaisuudessaan. Luokalle on määriteltä muuttuja nimeltään stillThreshold, joka on kynnyksiarvo silloin, kun käyttäjä on paikoillaan. Näin sopivaksi arvoksi 0.1. Heti ensimmäisenä tarkastetaan, onko käyttäjän funktiolle tuoma kokonaiskiihtyvyyksien keskiarvo pienempi kuin tämä still-kynnyksiarvo. Jos keskiarvo on pienempi, kutsutaan onActivityRecognized lambda-funktiota argumenteilla "still" ja "null". Kuvassa 3.21 nähtiin, että onAccelerationChanged-funktiossa compareDataAveragesin lambda-funktion palautusarvona tuleva aktiviteetin nimi muokkaa activityn tekstikenttää, jos sellainen löytyy. Eli nyt kun käyttäjä on paikoillaan, activityn tekstikenttään ilmestyy "still" ilman kokonaiskiihtyvyyksien keskiarvoa.

Jos if-ehto ei toteudu, listasta lähdetään etsimään filter-funktiolla sellaisia aktiviteetteja, joidenka kokonaiskiihtyvyyksien keskiarvo (third) on annettujen

ehto sisällä. Tämä tarkistus perustuu threshold-muuttujaan, joka on määritelty aiemmassa esikäsittelyvaiheessa (ks. luku 3.8.1). Threshold siis asettaa rajan sallitulle erolle aktiviteettien ja reaaliaikaisen kokonaiskiihtyvyyden keskiarvojen välillä. Kun ehdot täyttävät aktiviteetit tai aktiviteetti löytyvät, palauttaa filter-funktio uuden listan löydettyillä aktiviteeteilla ja tallentaa sen candidates-listamuuttujaan.

Seuraavassa vaiheessa tarkastetaan, että jos ehdon täyttäviä aktiviteetteja on yksi, lähetetään sen nimi ja kokonaiskiihtyvyyden keskiarvo onActivityRecognize-funktiolla eteenpäin activitylle. Jos ehdon täyttäviä aktiviteetteja on useampia, tarkastetaan, että mikä niistä on lähimpänä reaaliaikaista keskiarvoa. Tämän jälkeen hyödynnetään Kotlinin let-funktiota. Let-funktiota käytetään usein suorittamaan koodilohko, joka ei sisällä null-arvoja ja joka voidaan ketjuttaa muihin toimintoihin. (Scope functions 2023). Eli jos closestActivityn arvo ei ole null, funktio suoritetaan ja sen arvo saadaan it-parametriin. Nyt kutsutaan onActivityRecognized-funktiota currentActivityn toisella ja kolmannella komponentilla, jotka ovat siis aktiviteetin nimi (it.second) ja kokonaiskiihtyvyyksien keskiarvo (it.third), ja päivitetään ne RecognizeActivityn tekstikenttään.

```
fun compareDataAverages(realTimeTotAvrgAcc: Double, onActivityRecognized: (String?, Double?) -> Unit) {  
    // if staying still  
    if (realTimeTotAvrgAcc < stillThreshold) {  
        onActivityRecognized("still", null)  
        return  
    }  
  
    val candidates = activityAverageTotalAccList.filter { it: Triple<Long, String, Double>  
        it.third - thresholdAcceleration <= realTimeTotAvrgAcc && realTimeTotAvrgAcc <= it.third +  
            thresholdAcceleration  
    }  
  
    when {  
        candidates.size == 1 -> onActivityRecognized(candidates.first().second,  
            candidates.first().third)  
        candidates.size > 1 -> {  
            val closestActivity = candidates.minByOrNull { Math.abs(it.third - realTimeTotAvrgAcc) }  
            closestActivity?.let { it: Triple<Long, String, Double>  
                onActivityRecognized(it.second, it.third)  
            }  
        }  
        else -> onActivityRecognized(null, null)  
    }  
}
```

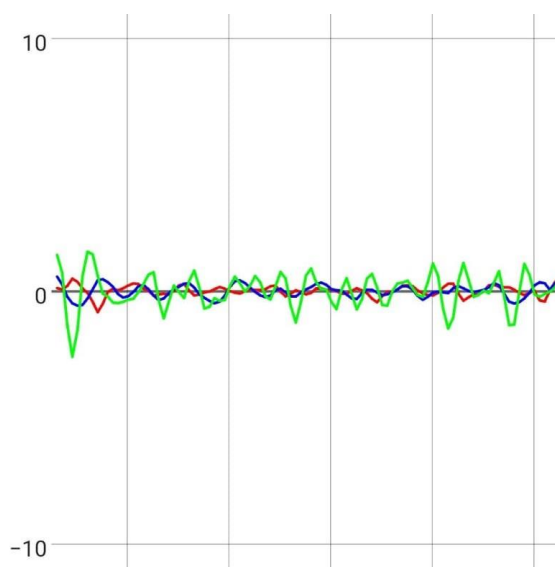
KUVA 3.23. Funktio compareDataAverages.

Kun tunnistaminen on suoritettu, currentActivity-lista tyhjennetään ja aloitetaan datan keruu siihen uudestaan.

### 3.9 Tulokset

Kokonaiskiihtyvyyksien keskiarvojen käyttö vertailussa oli muuten toimivaa, mutta koska se perustui vain kiihtyvyyksiin, ei esimerkiksi pyörällä ajo ollut kovin tunnistettavissa, koska "tasaisessa" liikkeessä ei juurikaan ole kiihtyvyyttä. Tässä tapauksessa kuitenkin kiihtyvyyksiä riitti jo senkin puolesta, että testit on otettu talvella epätasaisella jäisellä tiellä.

Mutta normaalisti autolla tai pyörällä ajaessa tasaisessa liikkeessä, kokonaiskiihtyvyyteen vaikuttavat vain pienet muutokset ympäristössä esimerkiksi juuri tien epätasaisuus, renkaat, käytössä olevan kulkupelin yksilöllisiin eroihin liittyvät asiat jne. joten on hyvin vaikea satunnaisien muuttujien takia arvioida aktiviteettia. Nämä kuitenkin taitavat olla ne ainoat keinot, jolla tasaisen liikkeen aktiviteetit voidaankin tunnistaa. Syvemmät tavat voisivat varmasti edes jollakin tapaa tätäkin pystyä ennustamaan ja laskea todennäköisyyksiin perustuvia laskuja, mutta niihin ei nyt oteta enempää kantaa tässä työssä. Esimerkiksi autolla ajaminen melko tasaisella 50 km/h nopeudella näyttäytyy suurin piirtein tällaiselta (Kuva 3.24):



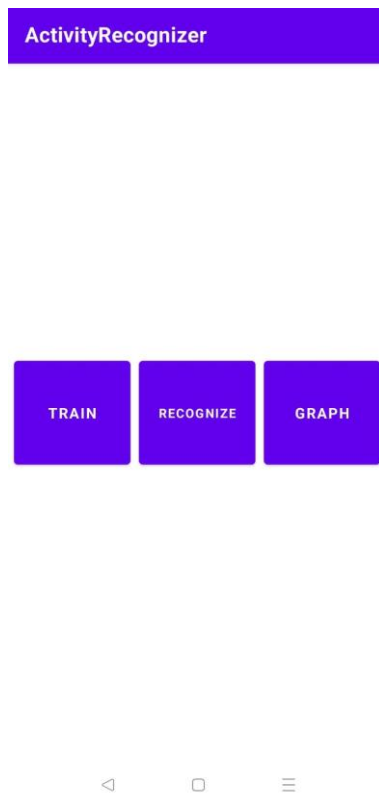
KUVA 3.24. X, y ja z -akselit autolla ajaessa.

Kuten huomaa, siitä on hyvin vaikea pelkkää kiihtyvyyssanturia ja kokonaiskiihtyvyyksien keskiarvoja käyttäen tietää, että käyttäjä ajaa nyt autoa. Siihen tarvittaisiin jo lisäksi muitakin mittareita tai parempia tapoja kuin kokonaiskiihtyvyyden keskiarvo.

Tulokset kuitenkin kävelyn ja juoksemisen osalta antoi hyvät tulokset. Tässäkin piti huomioida se, että puhelin ei voinut olla mitatessa ihan missä tahansa, koska esimerkiksi jos puhelin olisi kädessä kävellessä tai juostessa ”normaalisti”, eli heiluisi edes takaisin sivulla, olisivat kiihtyvyydetkin erilaiset. Ja tuloksiin myös vaikuttaa yksilökohtaiset erot, koska jokainen kävelee ja juoksee eri tahtia ja eri tavalla. Näin ollen mittaus soveltuu vain saman henkilön mittauksiin, ei useamman. Tosin siihen tarkoitukseen, eli yhdelle henkilölle personoituun tulokseen, varmasti suurimmalta osin tällaisissa sovelluksissa pyritäänkin.

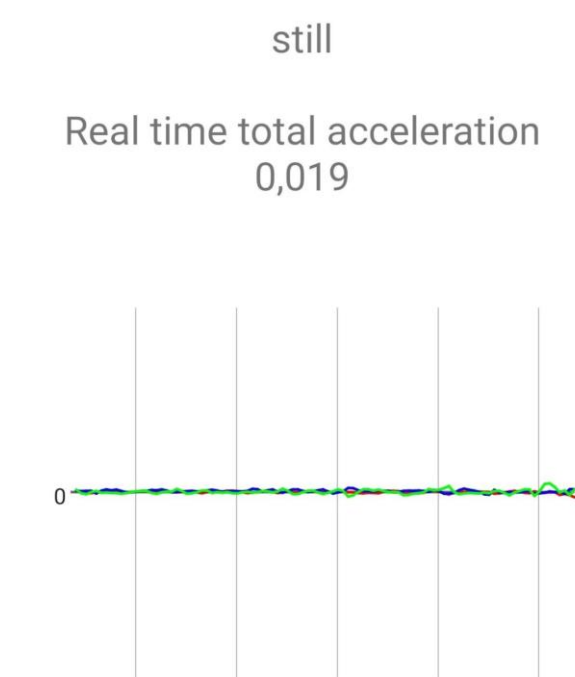
Puhelinta tulisi siis pitää samanlailla aina dataa tallentaessa ja vertailtaessa. Halutessaan voisi tietysti tallentaa aktiviteetit erikseen myös niin, että niissä mainitaan missä sitä on pidetty, mutta nyt työssä keskitytään perusasioihin ja tallennetaan jokainen näyte ja tehdään vertailu niin, että puhelin on rinnan edessä, jotta liialliset kiihtyvyyden vaihtelut minimoitaisiin ja saataisiin arvot suurin piirtein kropan kiihtyvyyden vaihteluista. Ja tämä myös siksi, että näen tuloksetkin heti.

Kuvassa 3.25 nähdään MainActivity, josta jokaisesta on napit tässä luvussa tutuiksi tulleisiin activityihin. Nyt kun on otettu aktiviteeteista kävely, juoksu ja pyöräily 10 kpl näytteitä, voidaan siirtyä Recognize-nappia painamalla tunnistus-activityyn.



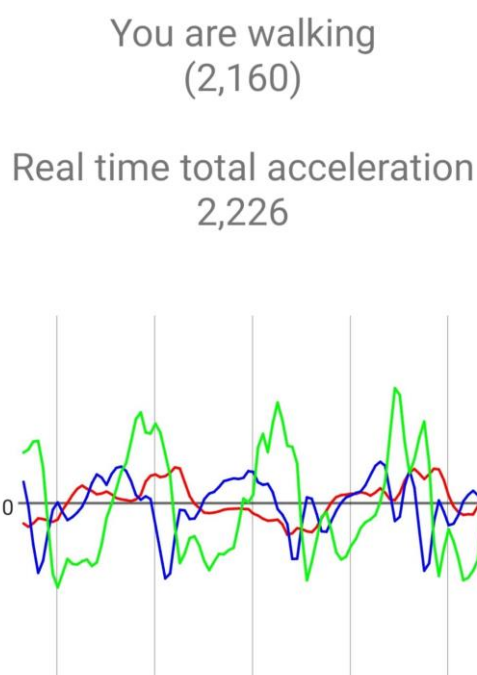
KUVA 3.25. MainActivity.

Kun ollaan paikallaan antaa sovellus "still" niin kuin pitääkin (kuva 3.26).



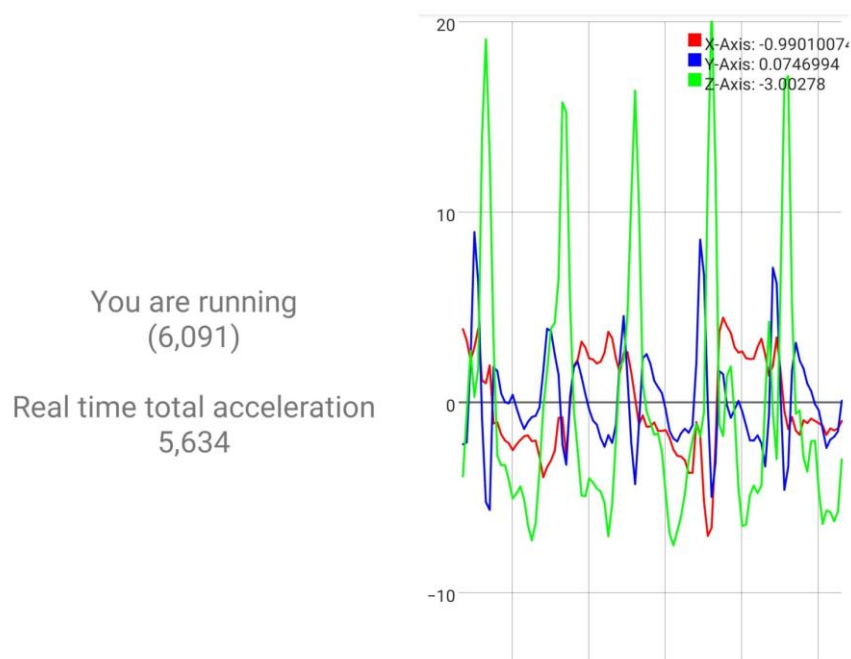
KUVA 3.26. Paikallaan seistessä.

Nyt testataan kävelyä. Kuten kuvasta 3.27 nähdään, sovellus tunnistaa hyvin kävelyn eikä tulos vaihdellut testaamisen aikana. Kun sovellus tunnistaa aktiviteetin, se myös ilmoittaa sen tietokantaan tallennetuista datoista kokonaiskiihtyvyyksien keskiarvon ja reaaliaikaisen kokonaiskiihtyvyyksien keskiarvon, joka siis aina lasketaan 40 kpl kiihtyvyydatasta, joten reaaliaikaisuus on tässä tilanteessa tuo 40 datan viive.



KUVA 3.27. Kävellessä.

Juostessa (Kuva 3.28) tunnistus oli myös tasaisesti oikea, eikä se vaihdellut.



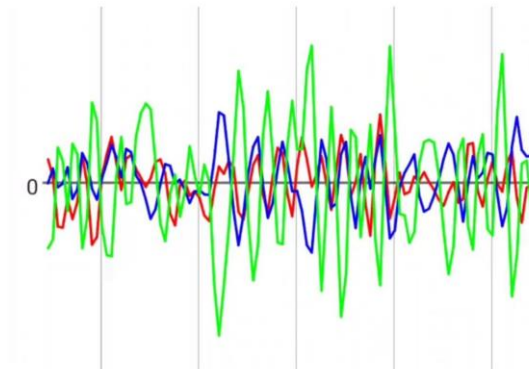
KUVA 3.28. Juostessa.

Nyt kun testataan pyöräilyä, graafista huomataan (Kuva 3.29), että kiihtyvyyssarvot vaihtelevat melko paljon, suurimmalta osin aiemmin mainitsemastani tien epätasaisuudesta johtuen, joten tuloskin vaihtelee. Pääasiassa sovellus kyllä antoi oikean tuloksen, mutta välillä se saattoi hyppiä hetkellisesti myös kävelyn puolelle. Kun katsotaan tietokannan kokonaiskiihtyvyyksien keskiarvoja, niin pyöräilyllä se on 2,657 ja kävelyllä 2,160. Nämä arvot ovat sen verran lähellä toisiaan, että ne voivat mennä melko helposti sekaisin. Kesällä tasaisella asfaltilla keskiarvo olisi tietysti paljon lähempänä nollaa mitä se nyt talvella on.



You are cycling  
(2,657)

Real time total acceleration  
2,618



KUVA 3.29. Pyöräillessä.

En itse pidä tätä tapaa kovin luotettavana aktiviteetin tunnistamiseen. Toki jos reaaliaikaista dataa kerättäisiin isompi määrä listalle ja pidemmän aikaa, saataisiin varmempi tulos, mutta silloin ei voitaisi puhua enää kovin reaaliaikaisesta tunnistamisesta. Tulokset kuitenkin antavat hyvän kokonais kuvan siitä, miten tunnistamista tulisi parantaa, ja myös osoittavat sen, että pelkästään kiihtyvyysanturia käyttäen ja ilman monimutkaisempia tunnistustoteutuksia, käyttäjän tekemän aktiviteetin tunnistaminen, varsinkin jos se on suurimmalta osin tasaista, ei ole kovin helppoa.

## 4 KIRJASTO

Tässä osioissa käydään läpi kirjaston luomisen, testaamisen ja dokumentoinnin vaiheet ja lopuksi sen julkaisu GitHubiin sekä jakaminen JitPackilla.

### 4.1 Kirjaston alustus

Android-kirjastoprojektilleni valitsin MIT-lisenssin, koska se tarjoaa laajat vapaudet käyttää, muokata ja jakaa ohjelmistoa. Lisäksi se on myös yleisin ohjelmistolisenssi (Murat Karagözgil 2023). Koska käytän pääasiassa avoimen lähdekoodin Android-kirjastoja omassa projektissani, MIT-lisenssi on luonnollinen valinta, sillä se yhdistää avoimen lähdekoodin periaatteet ja tarjoaa joustavuutta käytön suhteen. Toinen vaihtoehto olisi ollut Apache-lisenssi, joka on hyvin samankaltainen kuin MIT-lisenssi.

Android studiossa loin uuden projektin, mutta tällä kertaa en tarvitse mitään valmiina tarjottavista activity-templateista, koska kirjastohan ei tarvitse minkään näköistä käyttöliittymää. Vaikka valitsee kohdan "no activity", Android Studio lisää kuitenkin automaattisesti sovellukseen sovellusmoduulin, mutta se ei ole tarpeellinen kirjastolle. Koska kyseessä on kirjasto, voidaan sovellusmoduuli poistaa ja lisätä uusi Android-kirjastomoduuli.

#### 4.1.1 Luokkien ja funktioiden lopulliset eriyttämiset

Itse sovelluksessa laitoin kaikki toisiinsa liittyvät tiedostot omiin paketteihinsa. Tämä kirjasto on kuitenkin sen verran pieni, johon ei tule esimerkiksi activityihin liittyviä luokkia, niin kaikki kiihtyvyysanturiin liittyvät ja laskemiseen tarkoitetut luokat ovat kirjaston juuressa. Tietokantaan liittyvät luokat on kuitenkin laitettu database-paketin alle. Tämä on perusteltua, koska tietokantatoiminnallisuus muodostaa erillisen osan kirjastosta, jonka komponentteja käyttäjä voi joko hyödyntää, tai olla hyödyntämättä (pois lukien TrainingData), joten on järkevää pitää ne erillään omassa paketissaan.

Joitakin hienosäätöjä oli tarpeen tehdä, koska sovellusta rakennettaessa hain muutamissa kohdassa dataa suoraan tietokannasta sellaisissa luokissa, jotka oli

tarkoitus viedä myös kirjastoon. Jotta kirjaston modulaarisuus ja käytettävyys säilyisi hyvänä, oli tarvittavat datat tuotava luokan funktioille parametreina, jolloin ne eivät olisi riippuvaisia tietokannasta ja käyttäjä voisi tehdä oman näköisensä toteutuksen.

## **4.2 Testaus**

Jotta kirjasto toimisi odotetulla tavalla, on sitä hyvä testata. Tähän kirjastoprojektiin käytin yksikkötestejä JUnit:lla, jotka testaavat yksittäisiä komponentteja eri syötteillä ja odottavat tiettyä tulosta. JUnit on yleinen yksikkötestikehys Java-sovelluksille, ja se tarjoaa mekanismin yksikkötestien kirjoittamiseen ja suorittamiseen. Se näyttelee suurta roolia test-driven development -menetelmässä (TDD), jonka ideana on ensin testaus, sitten koodaus. (Tutorialspoint n.d) Yksikkötestejä kirjoittaessa on hyvä miettiä, mitkä ovat mahdollisia ongelmakohtia ja rakentaa niistä erilaisia testitapauksia. Kuvassa 4.1 on esimerkki AccelerationDataBufferTest-testiluokasta.

```

class AccelerationDataBufferTest {

    private lateinit var dataBuffer: AccelerationDataBuffer

    @Before
    fun setUp() {
        // Initialize buffer size as 3
        dataBuffer = AccelerationDataBuffer(bufferSize = 3)
    }

    @Test
    fun testAddData() {
        dataBuffer.addData(Triple( first: 1.0, second: 2.0, third: 3.0), time: 1000L)
        assertEquals( expected: 1, dataBuffer.getData().size)
    }

    @Test
    fun testGetDataEmptyBuffer() {
        assertEquals( expected: 0, dataBuffer.getData().size)
    }

    @Test
    fun testGetDataNonEmptyBuffer() {
        dataBuffer.addData(Triple( first: 1.0, second: 2.0, third: 3.0), time: 1000L)
        dataBuffer.addData(Triple( first: 4.0, second: 5.0, third: 6.0), time: 2000L)
        assertEquals( expected: 2, dataBuffer.getData().size)
    }

    @Test
    fun testAddDataWithBufferLimit() {
        dataBuffer.addData(Triple( first: 1.0, second: 2.0, third: 3.0), time: 1000L)
        dataBuffer.addData(Triple( first: 4.0, second: 5.0, third: 6.0), time: 2000L)
        dataBuffer.addData(Triple( first: 7.0, second: 8.0, third: 9.0), time: 3000L)
        dataBuffer.addData(Triple( first: 10.0, second: 11.0, third: 12.0), time: 4000L)

        assertEquals( expected: 3, dataBuffer.getData().size)
    }

    @Test

```

KUVA 4.1. Esimerkki yksikkötestauksesta.

@Before tarkoittaa sitä, että tämä funktio suoritetaan aina ennen jokaisen testitapauksen suorittamista, tässä tapauksessa siis alustaa bufferin koolla 3. @Before on tärkeä suorittaa, jotta jokainen testitapaus on lähtötilassa niitä suorittaessa.

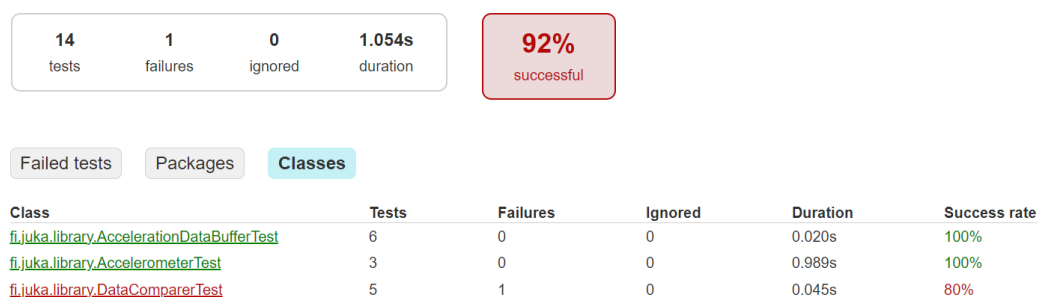
@Test on itse testitapaus. Kuten kuvasta 4.1 huomaa, testitapauksia on aina runsaasti ja niitä on hyvä ollakin. Testitapauksissa on lopussa assertEquals, jossa ensimmäinen argumentti on odotettu paluuarvo, ja toinen todellinen

paluuarvo. Jos nämä kaksi eivät ole samat, niin testi epäonnistuu. Kuitenkin on huomioitava, että assertEquals ei toki ole ainoa JUnit:n tapa testata totuusarvoja, vaan lukuisia muitakin on olemassa esimerkiksi assertEquals, assertNotNull jne. Runsas tarjonta antaa testien kirjoittajalle monipuolisemmat työkalut hyvien ja selkeiden testitapausten rakentamiseen.

Testitapaukset suoritetaan komentoriviltä joko windowsilla komennolla gradlew.bat test, tai linuxilla komennolla ./gradlew test. Jos kaikki sujuu hyvin, niin komentoriville ilmestyy teksti "build successful", mutta jos testit eivät mene läpi, antaa komentorivi ilmoituksen siitä. Komennolle voi myös antaa erilaisia lisäparametreja, jotta testaaja saa tarkempaa tietoa testeistä suoraan komentoriviltä.

Testien tuloksia voi kuitenkin yksinkertaisemmin selvittää kirjoittamalla selaimeen testien tiedostopolku ja lisätä index.html seuraavalla tavalla: file:/// {tiedostopolku} /index.html. Kuvassa 4.2 esimerkki kirjaston testien juuritiedostosta, josta mahdollisuus siirtyä tutkimaan epäonnistuneita testejä tarkemmin sellaisen sattuessa.

#### Test Summary



KUVA 4.2. Testien juuritiedosto selainäkymässä.

Kuvassa 4.3 nähdään tämän menetelmän kätevyys, kun tutkitaan missä on mennyt pieleen. Tällä kertaa assertEqualsiin on annettu tarkoituksenmukaisesti väärä odotusarvo.

## Class fi.juka.library.DataComparerTest

all > fi.juka.library > DataComparerTest

5	1	0	0.045s	80% successful
tests	failures	ignored	duration	

Failed tests

Tests

### testCompareDataAveragesForRecognizedActivity

```
org.junit.ComparisonFailure: expected:<[Walk]ing> but was:<[Runn]ing>  
    at org.junit.Assert.assertEquals(Assert.java:117)  
    at org.junit.Assert.assertEquals(Assert.java:146)  
    at fi.juka.library.DataComparerTest.testCompareDataAveragesForRecognizedActivity(DataComparerTest.kt:89)
```

KUVA 4.3. Esimerkki pieleen menneestä testitapauksesta.

## 4.3 Kirjaston dokumentaatio

Dokumentaatio on erittäin tärkeä osa kirjastoa, jotta käyttäjä osaa käyttää sitä oikein. Tässä kirjastoprojektissa voidaan sanoa olevan kolme tärkeää osaa liittyen dokumentaatioon: KDoc-syntaksi, Dokka, sekä README.md-tiedosto. Nämä yhdessä auttavat muodostamaan kattavan dokumentaation, joka auttaa käyttäjiä hyödyntämään kirjastoa tehokkaasti ja oikein.

### 4.3.1 KDoc

Virallinen tapa dokumentoida Kotlin-koodia on käyttää KDoc-syntaksia. Se laajentaa Javalla käytettävää JavaDoc-syntaksia tukemalla Kotlinin erityisrakenteita. JavaDocin tapaan, KDoc-kommentit alkavat `/**` ja päättyvät `*/`, ja jokaisen rivin alussa voi olla asteriski ilman, että sitä käsitellään kommentin sisältönä. Jokainen lohkon tunniste alkaa uudelta riviltä ja alkaa merkillä `@`.

KDocin avulla voi tuottaa selkeää ja jäsenneltyä dokumentaatiota Kotlin-projekteissa. Se mahdollistaa kattavan tiedon sisällyttämisen koodin toiminnasta, kuten parametrien, paluuarvojen ja poikkeusten kuvauksia. KDoc auttaa ylläpitämään hyvin dokumentoitua ja helposti ymmärrettävää koodikantaa, mikä edistää tehokasta yhteistyötä kehittäjien välillä ja helpottaa uusien kehittäjien perehtymistä projektiin. (Baeldung 2023)

Kuvassa 4.4 esimerkki KDoc:n käytöstä tässä kirjastoprojektissa:

```

/**
 * Saves a new activity to the database and returns its ID.
 *
 * @param activityName The name of the activity to be saved.
 * @return The ID of the newly inserted activity.
 */
fun saveActivityAndGetId(activityName: String): Long {
    val db = dbHelper.writableDatabase
    val formattedActivityName = activityName.lowercase().replace("\\s".toRegex(), replacement: "_")

    val values = ContentValues().apply {
        put(activityEntry.COLUMN_NAME_ACTIVITY, formattedActivityName)
    }
    val id = db.insert(activityEntry.TABLE_NAME, nullColumnHack: null, values)

    db.close()

    return id
}

```

KUVA 4.4. KDoc-syntaksia.

### 4.3.2 Dokka

Dokka on Kotlinin virallinen API-dokumentaatiotyökalu, joka ymmärtää Kotlinin KDoc- ja Javan Javadoc-kommentit. Dokka mahdollistaa automaattisen dokumentaation generoinnin ja tarjoaa tuen useille eri formaateille, mukaan lukien Dokkan oma HTML-formaatti sekä Javan Javadoc HTML. (Dokka Introduction 2024)

Jotta Dokkaa voidaan hyödyntää, täytyy Dokkan plugin (lisäosa) lisätä projektiin. Tämä tapahtuu lisäämällä Dokka pluginin kirjastomoduulin build.gradle-tiedostoon, kuten kuvassa 4.5 näytetään.

```

plugins {
    id 'com.android.library'
    id 'kotlin-android'
    id 'maven-publish'
    id 'org.jetbrains.dokka' version '1.9.10'
}

```

KUVA 4.5. Dokka plugin lisäys projektiin.

Nyt kun suoritetaan komentoriviltä komento `gradlew.bat dokkaHtml` (Linuxilla `./gradlew dokkaHtml`), Dokka generoi HTML-dokumentaation

KDoc/JavaDoc-kommenteista hakemistoon `.build/dokka/html`.

Hakemistosta löytyy nyt `index.html`-tiedosto, joka tarjoaa pääsyn tähän luotuun dokumentaatioon selaimesta.

Huomionarvoista liittyen projektin dokumentaatioon on, että Dokkan dokumentaatiossa todetaan seuraavaa:

If you want to publish your library to a repository, you may need to provide a `javadoc.jar` file that contains API reference documentation of your library.

For example, if you want to publish to Maven Central, you must supply a `javadoc.jar` alongside your project. However, not all repositories have that rule. (Dokka Introduction 2024)

Eli jos olisin julkaissut kirjaston Maven Centraliin, minun olisi täytynyt generoida ja toimittaa julkaisun mukana myös `javadoc.jar`-tiedosto. Mutta koska julkaisisin kirjaston GitHubissa ja rakentaisin sen JitPackilla, niin tähän se ei mitään ilmeisemmin päde. JitPack tunnistaa JavaDocin/KDocin suoraan koodista ja generoi dokumentaation itse repositoriosta. Tämän ansiosta erillistä `javadoc.jar`-tiedostoa ei välttämättä tarvitse. Tein tämän kuitenkin varmuuden vuoksi ja se onnistui seuraavalla kuvan 4.6 skriptillä:

```
tasks.register('dokkaJavadocJar', Jar.class) { Jar it ->
    dependsOn(dokkaJavadoc)
    from(dokkaJavadoc)
    archiveClassifier.set("javadoc")
}
```

KUVA 4.6. JavaDocin jar-tiedoston generointi.

### 4.3.3 GitHub Pages

Nyt kun HTML-muotoinen dokumentti on generoitu Dokkalla ja kopioitu GitHubiin `./docs`-kansioon, voimme luoda oman sivuston sille käyttämällä GitHub Pagesia. GitHub Pages on palvelu, joka mahdollistaa staattisten verkkosivujen julkaisemisen suoraan GitHubin repositoriosta. Se ottaa HTML-, CSS- ja




JavaScript-tiedostot suoraan GitHubista ja mahdollistaa niiden julkaisemisen verkkosivustona. (GitHub Docs 2024)

GitHub Pages konfiguroidaan GitHub-repositorion asetuksista kohdasta Pages. Siellä määritellään, mistä haarasta ja mistä hakemistosta sivusto luodaan. GitHub Pages seuraa tämän valitun haaran ja hakemiston committeja ja automaattisesti generoi niistä verkkosivuston. Rakennusprosessia voi seurata Actions-välilehdellä. Kun prosessi on valmis, GitHub Pages generoi osoitteen, josta verkkosivusto on saatavilla (Kuva 4.7).

## GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

Your site is live at <https://jk634.github.io/physical-activity-recognition-util/>  
Last deployed by  jk634 1 minute ago

 Visit site

...

## Build and deployment

### Source

Deploy from a branch ▾

### Branch

Your GitHub Pages site is currently being built from the `/docs` folder in the `main` branch. [Learn more about configuring the publishing source for your site.](#)

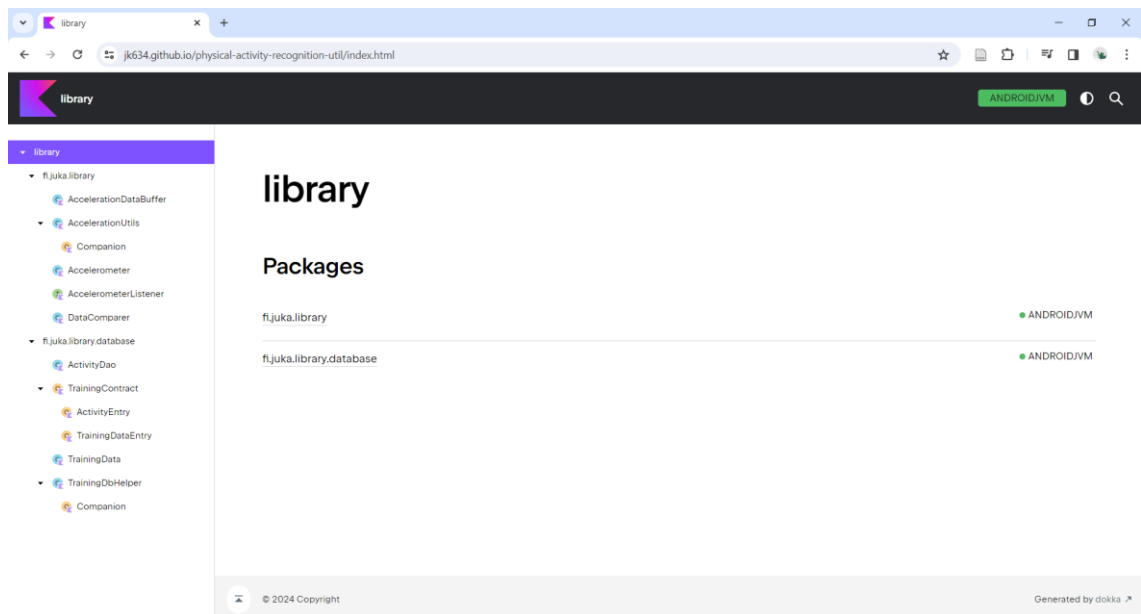
 main ▾

 /docs ▾

Save

KUVA 4.7. Verkkosoite dokumentaatioon kohdassa "Pages" repositorion asetuksissa.



Nyt kun kaikki on valmista, näemme kirjaston dokumentaation menemällä tähän osoitteeseen (Kuva 4.8).





KUVA 4.8. Kirjaston dokumentaatio verkkosivulla.

#### 4.3.4 Readme

README.md-tiedosto on olennainen osa projektien dokumentaatiota. Se tarjoaa yleensä yleiskuvan projektin toiminnasta, asennusohjeet ja muuta tärkeää tietoa käyttäjille. Yleensä se sijaitsee GitHub-repositorion juuressa ja on näkyvillä suoraan repositorion etusivulla. Tiedostopääte .md viittaa Markdown-muotoilukieleen, joka tarjoaa yksinkertaisen ja helpon tavan kirjoittaa ja muotoilla dokumentaatiota. (GitHub Docs 2024). Tässä kuvassa 4.9 osa kirjastoprojektin Readme.md-tiedostoa:

 README  License

## Activity Recognition Library

This library provides Android developers with tools to recognize different physical activities using the accelerometer sensor data.

The library includes features such as:

- **Activity Recognition:** Recognizes user's physical activities in real-time.
- **Utilization of Accelerometer:** Utilize the smartphone's accelerometer for accurate activity detection.
- **Flexible Integration:** Easily integrate into Android projects.
- **Clear Documentation:** Provides comprehensive documentation.

## Documentation

Documentation is available at <https://jk634.github.io/physical-activity-recognition-util/>.

## Installation

Make sure to include maven `url 'https://jitpack.io'` in your root build.gradle file, to ensure the library can be fetched from JitPack repository:

```
allprojects {
    repositories {
        ...
        maven { url 'https://jitpack.io' }
    }
}
```

Add the following dependency to your module-level build.gradle file to integrate the library into your project:

```
implementation 'com.github.jk634:juka:physical-activity-recognition-util:1.1.0'
```

## Usage Examples

### Accelerometer

KUVA 4.9. Osa kirjastoprojektin README.md-tiedostoa.

## 4.4 Kirjaston julkaiseminen

Jotta voimme julkaista kirjaston, täytyy tehdä tiettyjä konfiguraatioita. Liikkeelle on kuitenkin hyvä lähteä lisäämällä kirjastomodulin build.gradle-tiedostoon maven-publish -lisäosa, koska se mahdollistaa kirjaston julkaisun Maven-repositorioon tai muihin vastaaviin jakelukanaviin (Kuva 4.10).

```

plugins {
    id 'com.android.library'
    id 'kotlin-android'
    id 'maven-publish'
}

```

KUVA 4.10. Tarvittava maven-publish plugin.

#### 4.4.1 Lokaali testijulkaisu

Ennen kuin kirjasto julkaistaan, on hyvä testata lokaalisti sen toimivuus. Tähän tarkoitukseen lisätään tarvittava skripti kirjastomodulin build.gradle-tiedostoon, jotta kirjaston käyttöä voidaan testata paikallisesti (Kuva 4.11).

```

// Only for local publishing
afterEvaluate {
    android.libraryVariants.each { NonExistentClass variant ->
        publishing.publications.create(variant.name, MavenPublication) { MavenPublication it ->
            from components.findByName(variant.name)

            groupId='fi.juka'
            artifactId='physical-activity-recognition-util'
            version='1.0.3'
        }
    }
}

```

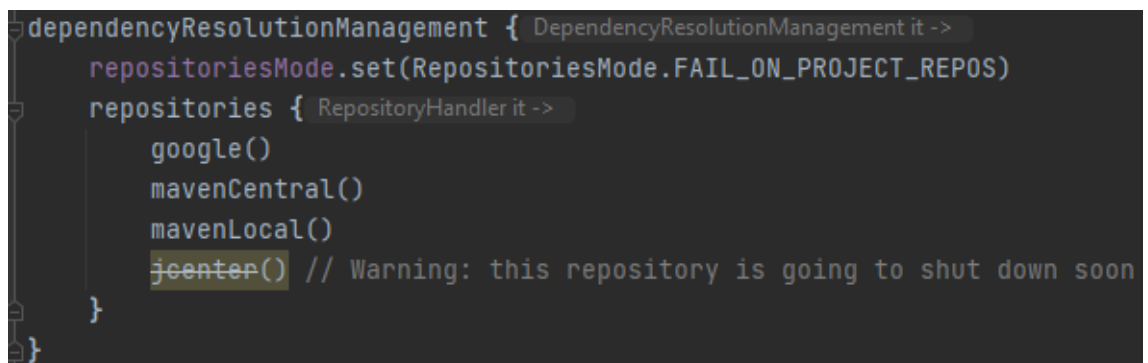
KUVA 4.11. Lokaaliin julkaisuun käytettävä skripti.

Tämä skripti suoritetaan, kun kaikki projektiin liittyvät arvot ja konfiguraatiot on arvioitu ja valmiina käytettäviksi. Tärkeä osa tässä koodissa on groupId, artifactId sekä version. Kun näiden arvot yhdistetään ja erotellaan kaksoispisteillä, saadaan Maven-koordinaatti. Eli groupId:artifactId:version ovat pakolliset kentät, jotka toimivat projektin osoitteena ja aikaleimana. GroupId määrittelee organisaation tai projektin, johon kirjasto kuuluu. Se on yleensä käänteinen domain-nimi. ArtifactId on projektin nimi ja lopuksi tulee versionumero. Nämä elementit auttavat erottamaan projektin muista ja kertovat Mavenille, mitä versiota projektista haluamme käyttää. (Apache Maven, 2024)

Kun publishToMavenLocal-skripti suoritetaan, se julkaisee kirjaston paikallisesti Mavenin local repositoryyn, joka sijaitsee windowsilla polussa

`{user}/.m2/repository/{maven-koordinaatti}`. Tämä mahdollistaa kirjastojen jakamisen ja käytön eri projekteissa samalla tietokoneella.

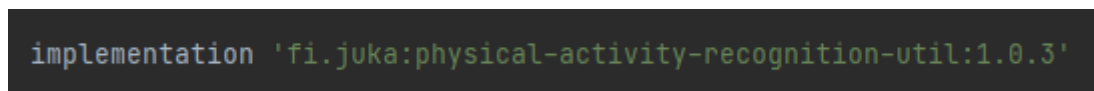
Nyt kirjastoa on mahdollista käyttää toisessa paikallisessa projektissa, mutta ensin on varmistettava, että Mavenin paikallinen repository on mukana. Tämä tehdään lisäämällä `mavenLocal()` `settings.gradle`-tiedostoon, jossa määritellään eri repositoriot (Kuva 4.12). Tämä mahdollistaa paikallisesti tallennettujen kirjastojen käytön ja jakamisen muiden projektien kanssa.



```
dependencyResolutionManagement { DependencyResolutionManagement it ->
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories { RepositoryHandler it ->
        google()
        mavenCentral()
        mavenLocal()
        jcenter() // Warning: this repository is going to shut down soon
    }
}
```

KUVA 4.12. Repositorioiden määrittely `settings.gradle`-tiedostossa.

Nyt kirjasto voidaan tuoda projektiin paikallisesti. Sovelluksen `build.gradle`en lisätään uusi riippuvuus, joka siis on kaksoispistein yhdistetty `groupId`, `artifactId` ja `version` (Kuva 4.13).



```
implementation 'fi.juka:physical-activity-recognition-util:1.0.3'
```

KUVA 4.13. Kirjaston sisällyttäminen projektiin.

Kuten kuvasta 4.14 huomataan, nyt kirjaston luokat ovat käytettävissä tässä toisessa paikallisessa projektissa.

```
import fi.juka.library.Accelerometer

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        Accelerometer( context: this)
    }
}

fi.juka.library.Accelerometer
public constructor Accelerometer(
    context: Context
)

Gradle: fi.juka:physical-activity-recognition-util:1.0.3@aar:
```

KUVA 4.14. Paikallisesti julkaistun kirjasto toimii nyt paikallisessa projektissa.

#### 4.4.2 Julkaisu GitHubiin ja jakelu JitPackilla

Kun kirjasto halutaan julkaista GitHubiin ja jakaa JitPackilla, tarvitsee luoda yksi uusi tiedosto projektin juureen, jitpack.yml (Kuva 4.15). Tänne lisätään, mitä JDK:ta (Java Development Kit) käytetään projektin rakentamiseen. JDK on kehityspaketti, joka sisältää työkalut Java-sovellusten kehittämiseen. Oletusarvoisesti JitPack käyttää OpenJDK Java 8, mutta me muutamme sen tässä OpenJDK Java 11:ksi.

```
jdk:
- openjdk11
```

Kuva 4.15. Jitpack.yml.

Tämä kuvassa 4.16 esitetty tarvittava julkaisuskripti lisätään kirjastomodulin build.gradle-tiedostoon.

```

afterEvaluate{
    publishing{
        publications{ PublicationContainer it ->
            release(MavenPublication){
                from components.release

                groupId='com.github.jk634'
                artifactId = 'physical-activity-recognition-util'
                version = '1.1.0'
            }
        }
    }
}

```

Kuva 4.16. Kirjaston julkaisuskripti.

Tämä skripti muistuttaa lähestulkoon paikallista julkaisua. Tässä merkittävin ero on groupId:ssa, joka nyt koostuu "com.github" -alkuliitteestä sekä GitHub-käyttäjätunnuksesta. ArtifactId on GitHub-repositorion nimi. Lisäksi versionumero on päivitetty vastaamaan uutta julkaisua (kts. seuraava kappale).

Kun tarvittavat toimenpiteet on tehty julkaisemista varten ja kaikki on työnnetty GitHubiin, on aika tehdä julkaisu (release). Repositorion etusivulta oikeasta laidasta löytyy kohta, jossa lukee "Releases". Jos tehdään ensimmäistä julkaisua, siellä lukee "Create a new release". Kun siitä painetaan, avautuu uusi sivu (Kuva 4.17). Tällä sivulla määritellään tagi, johon voit esimerkiksi laittaa ensimmäisen julkaisun numeron, kuten 1.0.0. Tämän jälkeen täytetään otsikko (title) ja kuvaus (description), ja lopuksi painetaan "Publish release". Näin julkaisu on tehty.

Releases

Tags

1.0.0

Target: main

Previous tag: auto

Generate release notes

Excellent! This tag will be created from the target when you publish this release.

Initial Release

Write

Preview

H B I

This release marks the initial version of the project. It includes these Kotlin classes:

library/

- AccelerationDataBuffer
- AccelerationUtils
- Accelerometer
- AccelerometerListener (interface)
- DataComparer

library.database/

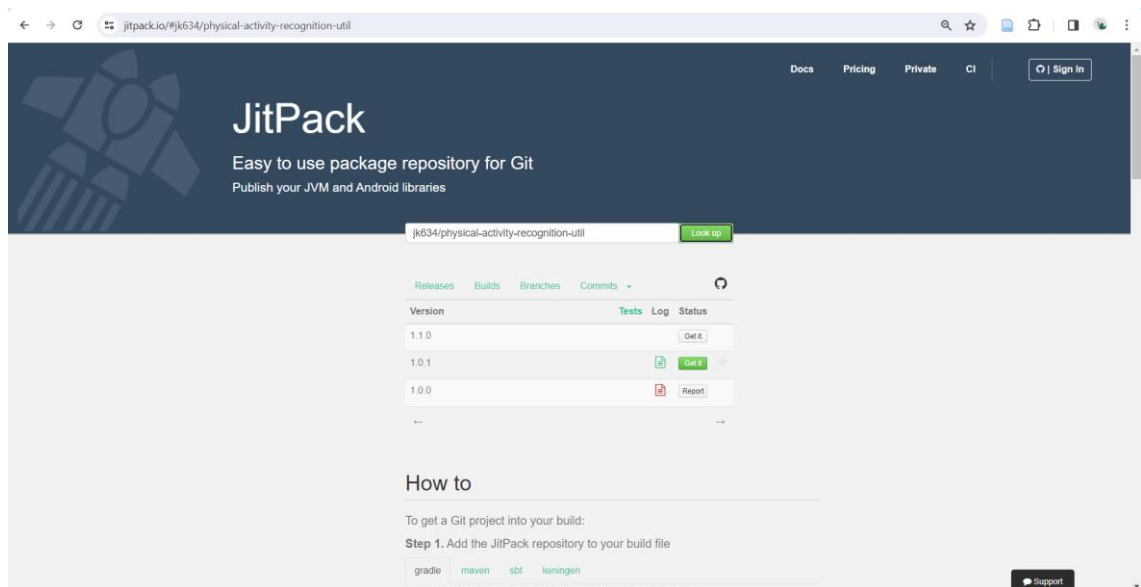
- ActivityDao
- TrainingContract (object)
- TrainingData
- TrainingDbHelper

Markdown is supported Paste, drop, or click to add files

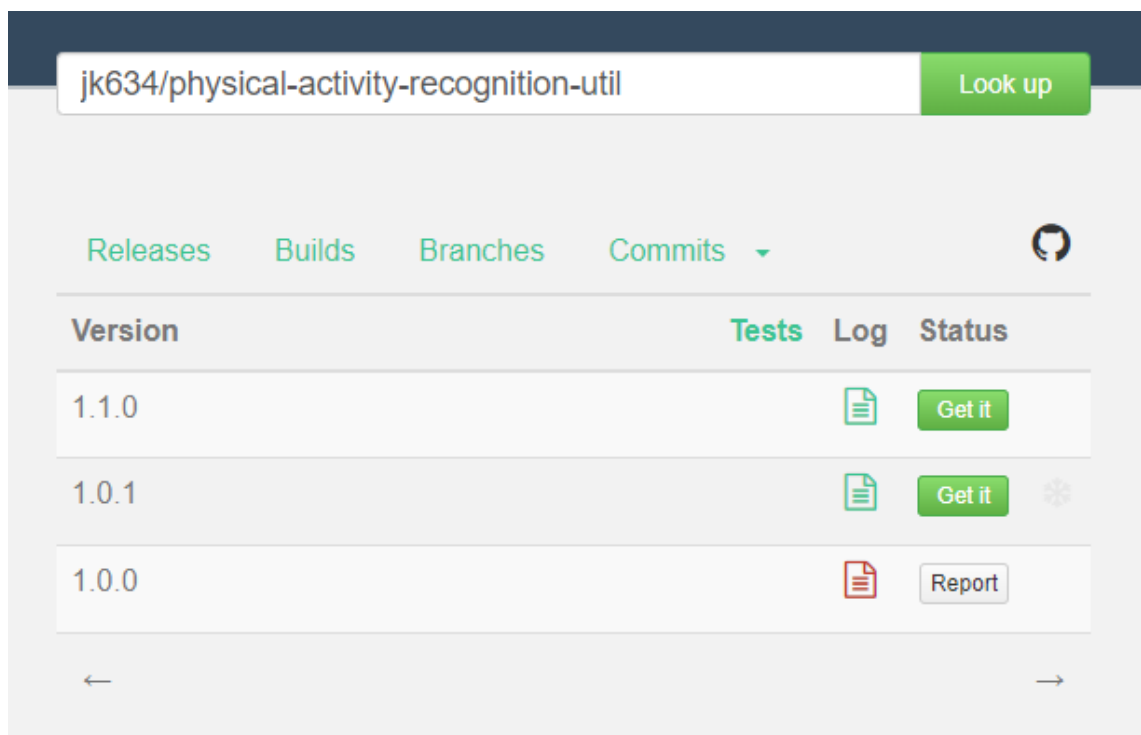
KUVA 4.17. Ensimmäisen version julkaisusivu GitHubissa.

Seuraavaksi kirjaudutaan JitPackiin GitHub-tunnuksilla osoitteessa [jitpack.io](https://jitpack.io). Kirjautumisen jälkeen voidaan JitPackista etsiä omia repositorioita, josta löytyy niiden kaikki julkaisut. Kuvassa 4.18 näkyy kirjaston repositorion julkaisuhistoria. Kun kohdassa 1.1.0 painetaan "Get it", JitPack aloittaa kirjaston rakentamisen eli buildin. Tämä vie hetken aikaa, mutta kun prosessi on valmis, ilmestyy statuksen viereen joko vihreä tai punainen paperi. Vihreä paperi ilmaisee, että kirjaston buildaus ja julkaisu ovat onnistuneet, kun taas punainen merkintä osoittaa, että jokin meni vikaan ja julkaisu epäonnistui. Kuten kuvasta 4.19 ilmenee, ensimmäinen julkaisu 1.0.0 ei mennyt läpi, johtuen pienestä julkaisuun liittyvästä virheestä. Toisessa 1.0.1 tämä ongelma korjattiin ja julkaisu onnistui. Nyt kolmannessa versiossa 1.1.0 on tehty isompia korjauksia, esimerkiksi tunnistamiseen, niin siksi tagi on myös suurempi.





KUVA 4.18. jitpack.io ja julkaisuhistoria.



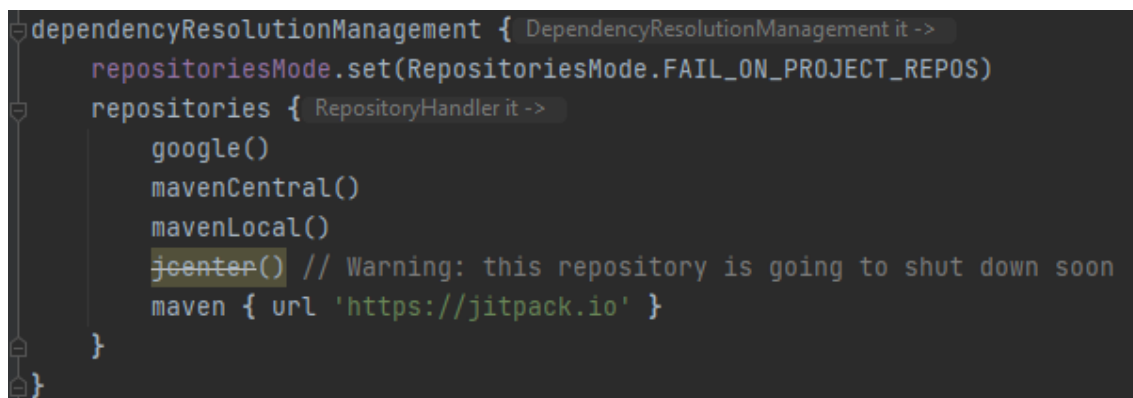
KUVA 4.19. Epäonnistunut ja onnistuneet buildit.

Kirjasto on nyt onnistuneesti julkaistu GitHubissa sekä buildattu ja jaettu JitPackilla. Tämän ansiosta muut Android-kehittäjät voivat helposti käyttää kirjastoa ja integroida sen omiin projekteihinsa.

## 4.5 Kirjaston integrointi projektissa

Jotta kirjastoa voidaan käyttää projektissa, se on tuotava projektiin käyttämällä JitPackia. Kirjaston integroiminen Android-projektiin JitPackin avulla on yksinkertainen prosessi, joka koostuu muutamasta vaiheesta.

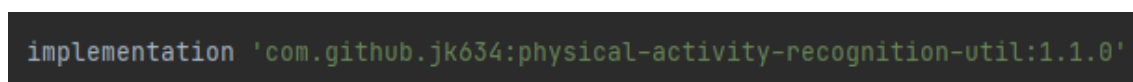
Ensiksi projektin settings.gradle-tiedostoon lisätään maven { url 'https://jitpack.io' }, kohtaan jossa määritellään käytettävät repositoriot (Kuva 4.20). Näin projekti osaa hakea kirjaston oikeasta paikasta.

A screenshot of a code editor showing the 'settings.gradle' file. The code is as follows:

```
dependencyResolutionManagement { DependencyResolutionManagement it ->
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories { RepositoryHandler it ->
        google()
        mavenCentral()
        mavenLocal()
        jcenter() // Warning: this repository is going to shut down soon
        maven { url 'https://jitpack.io' }
    }
}
```

KUVA 4.20. Lisätään jitpack.io -repositorio.

Seuraavaksi integroidaan kirjasto projektiin, jotta sitä voidaan käyttää. Kirjasto lisätään riippuvuudeksi (Kuva 4.21).

A screenshot of a code editor showing a dependency declaration in a 'build.gradle' file. The code is as follows:

```
implementation 'com.github.jk634:physical-activity-recognition-util:1.1.0'
```

KUVA 4.21. Kirjasto projektiin riippuvuudeksi.

Tässä vaiheessa on hyvä huomata, että kirjasto haetaan nyt käyttämällä "com.github"-alkuliitettä lisättynä GitHub-käyttäjätunnuksella, repositorion nimellä sekä julkaisun (release) tagilla. Kirjasto noudetaan siis JitPackin avulla, joka on sidottu näihin GitHub-tietoihin.

Kun testataan, toimiiko kirjasto testiprojektissa, niin kuten kuvasta 4.22 nähdään, kaikki toimii kuten pitääkin. Tässä huomataan myös se, että dokumentaatiokin on näkyvissä.

```
import fi.juka.library.Accelerometer


class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        Accelerometer(context: this)
    }
}
```

```
fi.juka.library.Accelerometer
public constructor Accelerometer(
    context: Context
)
```

The Accelerometer class represents an accelerometer sensor and provides methods to register and unregister listeners, as well as filter accelerometer data.

 Gradle:

com.github.jk634:physical-activity-recognition-util:1.1.0@aar

KUVA 4.22. Kirjaston toimivuus ulkopuolisessa projektissa.

## 5 POHDINTA

Tavoitteisiin nähden, projekti onnistui suurimmalta osin odotetusti ja alkuperäistä suunnitelmaa noudattaen. Poikkeuksena kuitenkin oli, ettei koneoppimismenetelmiä lähdetty tekemään, koska niiden opiskelu ja käyttäminen tässä projektissa olisi tullut viemään todella paljon enemmän aikaa mitä tämä projekti nyt vei. Muut asiat menivät kutakuinkin niin, miten suunnitelmaan oli kirjattu. Toinen pienempi asia mikä poikkesi, oli se, että en tehnyt kirjastoa sovellusprojektin aliprojektiksi, vaan tein siitä uuden aivan eri projektin. Tämä oli mielestäni hyvä ratkaisu ja on sitä edelleen, varsinkin kun on tekemässä tällaista toteutusta ensimmäistä kertaa. Tällä tavoin kirjaston repositorio ja muutenkin koko projekti saadaan pidettyä selkeänä.

Haasteita riitti projektin aikana jonkin verran, mutta mitään ylitsepääsemätöntä ei tullut kuitenkaan vastaan. Haasteet liittyivät enimmäkseen joihinkin pieniin ohjelmointipuolen ongelmiin, joihin melko nopeasti sai apua etsimällä tietoa netistä. Kuitenkin etukäteen suunniteltu ja mietitty toteutus auttoi suurimmalta osin kulkemaan oikeaa polkua pitkin.

Yhtenä huomiona se, että näin jälkikäteen mietittynä tekisin tunnistamisen eri tavalla. Dynaamista graafia seuraamalla huomasin, että todennäköisesti olisi parempikin tapa toteuttaa tunnistaminen ilman monimutkaisempia menetelmiä. Siinä en laskisi kokonaiskiihtyvyyksien keskiarvoja, vaan laskisin näytteistä kiihtyvyyksien piikkejä, eli milloin arvo on korkeimmillaan ja tallentaisin niiden ajanhetket ja kiihtyvyyssarvot. Kun seuraava piikki tulee, ja jos aktiviteetteja on monta saman kynnyksarvon sisällä, niin laskisin näiden erotuksen ja pyrkisin sillä tunnistamaan aktiviteetin. Jokaisen aktiviteetin piikkiarvojen keskiarvot sekä ajanhetkien välisten erotusten keskiarvot olisivat tallennettuna siis tietokantaan, josta näitä vertailtaisiin reaaliaikaisiin arvoihin. Uskoisin tällaisen tavan olevan paljon tehokkaampi ja tarkempi kuin toteuttamani, mutta ei sekään riittävä silti olisi. Tarvittaisiin joko edistyneempiä koneoppimis- tai neuroverkkomenetelmiä, tai vaihtoehtoisesti tulisi ottaa käyttöön lisää mittareita.

Kirjastoa olisi tietysti mahdollista vielä parantaa ja laajentaa esimerkiksi tällä toisella tunnistustavalla. Tällöinhän se vaatisi vain dokumentaatioiden ja uusien

GitHub-julkaisujen päivitystä JitPackilla. Luulen kuitenkin, että tämä kirjastoprojekti jää tähän, mutta nyt tiedän ainakin melko hyvin, kuinka tällainen kirjasto toteutetaan ja kuinka se saadaan ihmisten saataville kaikkialta maailmassa.

Loppujen lopuksi valmiiksi saatiin toimiva sovellus, joka osaa verrata käyttäjän aikaisempia tietokantaan tallennettuja kiihtyvyyssdataa reaaliaikaiseen kiihtyvyyssdataan, ja näin kertoa käyttäjälle melko hyvin mitä ollaan tekemässä, sekä aktiviteetin tunnistamiseen soveltuva kirjasto, joka on kaikille kehittäjille saatavilla JitPackin avulla.

## LÄHTEET

Amazon Web Services. n.d. What is SQL?. Verkkosivu. Viitattu 9.2.2024.  
<https://aws.amazon.com/what-is/sql/>

Android library. 2024. Android Developers. Verkkosivu. Viitattu 9.2.2024.  
<https://developer.android.com/studio/projects/android-library>

Apache Maven. 2024. What is Maven. Verkkosivu. Viitattu 17.2.2024.  
<https://maven.apache.org/what-is-maven.html>

Apache Maven. 2024. POM Reference. Verkkosivu. Viitattu 30.1.2024.  
<https://maven.apache.org/pom.html>

App manifest overview. 2024. Android Developers. Verkkosivu. Viitattu 10.2.2024.  
<https://developer.android.com/guide/topics/manifest/manifest-intro>

Baeldung. 2023. An Introduction to KDoc. Verkkosivu. Viitattu 3.2.2024.  
<https://www.baeldung.com/kotlin/kdoc>

Dokka Introduction. 2024. Kotlin. Verkkosivu. Viitattu 7.2.2024. <https://kotlin-lang.org/docs/dokka-introduction.html>

Geeks for Geeks. 2023. Thread in Operating System. Verkkosivu. Viitattu 18.12.2023. <https://www.geeksforgeeks.org/thread-in-operating-system/>

GitHub Docs. 2024. About GitHub Pages. Verkkosivu. Viitattu 4.2.2024.  
<https://docs.github.com/en/pages/getting-started-with-github-pages/about-github-pages>

GitHub Docs. 2024. Quickstart for writing on GitHub. Verkkosivu. Viitattu 7.2.2024. <https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/quickstart-for-writing-on-github>

Gradle Build Tool. 2023. Gradle User Manual. Verkkosivu. Viitattu 19.3.2024.  
<https://docs.gradle.org/current/userguide/userguide.html>

Guru Technolabs. 2023. The Rise of Kotlin: How It's Revolutionizing Android Development. Medium 10.7.2023. Viitattu 10.2.2024.  
<https://medium.com/@GuruTechnolabs/the-rise-of-kotlin-how-its-revolutionizing-android-development-931ac8c13d67>

JitPack.io. n.d. JitPack.io. Verkkosivu. Viitattu 17.2.2024.  
<https://jitpack.io/docs/#jitpackio>

jjoe64. n.d. GraphView. Verkkosivu. Viitattu 27.5.2023.  
<https://github.com/jjoe64/GraphView>

Motion sensors. 2023. Android Developers. Verkkosivu. Viitattu 24.4.2023.  
[https://developer.android.com/guide/topics/sensors/sensors\\_motion.html](https://developer.android.com/guide/topics/sensors/sensors_motion.html)

Murat Karagözgil. 2023. Software Licenses on GitHub: Which One Should You Choose?. Medium 11.10.2023. Viitattu 21.3.2024.  
<https://muratkaragozgil.medium.com/software-licenses-on-github-which-one-should-you-choose-3d4cfbb6c2f9>

Object expressions and declarations. n.d. Kotlin. Verkkosivu. Viitattu 15.12.2023. <https://kotlinlang.org/docs/object-declarations.html#semantic-difference-between-object-expressions-and-declarations>

Save data using SQLite. 2024. Android Developers. Verkkosivu. Viitattu 11.2.2024. <https://developer.android.com/training/data-storage/sqlite>

Scope Functions. 2023. Kotlin. Verkkosivu. Viitattu 18.01.2024. <https://kotlinlang.org/docs/scope-functions.html#let>

Sensors Overview. 2024. Android Developers. Verkkosivu. Viitattu 7.2.2024. [https://developer.android.com/develop/sensors-and-location/sensors/sensors\\_overview](https://developer.android.com/develop/sensors-and-location/sensors/sensors_overview)

SQLite. 2023. About SQLite. Verkkosivu. Viitattu 9.2.2023. <https://www.sqlite.org/about.html>

Tutorialspoint. n.d. JUnit – Overview. Verkkosivu. Viitattu 25.01.2024. [https://www.tutorialspoint.com/junit/junit\\_overview.htm](https://www.tutorialspoint.com/junit/junit_overview.htm)

Tutorialspoint. n.d. What is Android?. Verkkosivu. Viitattu 30.08.2024. [https://www.tutorialspoint.com/android/android\\_overview.htm](https://www.tutorialspoint.com/android/android_overview.htm)

Waldo. 2021. Kotlin AlertDialog: Explained in Detail With an Example. Verkkosivu. Viitattu 16.12.2023. <https://www.waldo.com/blog/kotlin-alertdialog>

Why adopt Compose. 2024. Android Developers. Verkkosivu. Viitattu 16.09.2024. <https://developer.android.com/develop/ui/compose/why-adopt>

Yuudai, I. 2023. Unleashing the Power of Native C++ for Android App Development. Medium 25.6.2023. Viitattu 9.2.2024. <https://medium.com/@tesla8877/unleashing-the-power-of-native-c-for-android-app-development-e2ade42fe06f>