OULUN AMMATTIKORKEAKOULU

Anna Inkala

ANDROID-SOVELLUKSEN TOTEUTUS HIIHTOURHEILUN TYÖKALUKSI

ANDROID-SOVELLUKSEN TOTEUTUS HIIHTOURHEILUN TYÖKALUKSI

Anna Inkala
Opinnäytetyö
Kevät 2018
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma

Tekijä: Anna Inkala
Opinnäytetyön nimi: Android-sovelluksen toteutus hiihtourheilun työkaluksi
Työn ohjaaja: Veikko Tapaninen
Työn valmistumislukukausi ja -vuosi: kevät 2018
Sivumäärä: 52

Työn aiheena oli Exiopsin SKIIOT-laitteelle toteutetun Android-mobiilisovelluksen käyttöliittymän ja Bluetooth Low Energy -pohjaiseen tiedonsiirtoon sekä saadun datan tallennukseen liittyvien toiminnallisuuksien teknisten ratkaisujen suunnittelu ja toteutus. Tavoitteena oli tehdä versiot sekä Android- että Android Wear -alustoille, erityisesti Polar M600 -älykellolle. Tästä johtuen työn toteutuksessa on pyritty huomioimaan molempien alustojen tarpeet ja toisaalta minimoimaan eroavaisuus koodissa.

Työn toteutus tapahtui kahdessa jaksossa, joista ensimmäinen painottui älykelloalustalle, ja myöhempi kehitystyö puolestaan painottui lähinnä puhelin ja tablettialustalle. Sovellus sisältää myös ominaisuuksia, kuten matkan ja nopeuden mittaaminen, jotka vaikuttivat sovelluksen käyttöliittymän suunnitteluun ja toteutukseen, mutta jotka muuten jäävät tämän tarkastelun ulkopuolelle.

Työn tuloksena on saatu Android- ja Android Wear -alustoilla toimiva sovellus SKIIOTille, joka sisältää Bluetooth Low Energy -pohjaisen yhteyden luonnin ja hallinnoinnin, laitteesta saatavan sensoridatan käsittelyn sekä tallennuksen sekä käyttöliittymän yhteyden luontiin ja hallinnointiin, tallennuksen hallinnointiin sekä reaaliaikaisen sensoridatan esitykseen.

Asiasanat: Android, mobiilisovellus, Bluetooth Low Energy, Java, Internet of Things

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Software development

Author(s): Anna Inkala
Title of thesis: The implementation of an Android application for skiing analyzer
Supervisor(s): Veikko Tapaninen
Term and year when the thesis was submitted: Spring 2018
Number of pages: 52

The aim of this thesis was to design and implement a UI and Bluetooth low Energy connectivity related features for SKIIOT skiing analyzer.

The implementation of the application took place in two periods. In the first period the watch application was priority and in the second period the project was focused mainly on the phone platform. The application has features such as measuring distance and speed that is not in the scope of this thesis.

The result was an application that runs on both Android and Android Wear platforms. It includes creating and managing Bluetooth Low Energy connection between application and device, handling of the acquired data, and UI to display that data. It also includes recording functionality with database and a special UI to acquire and display previously recorded data.

Keywords: Android, mobile application, Bluetooth Low Energy, Java, Internet of Things

SISÄLLYS

LYHENTEET

ADB (Android Debug Bridge) Komentorivityökalu, jonka avulla voidaan kommunikoida laitteen kanssa.

APK (Android application package) Android-sovellusten pakkaustiedosto, joka on asennettavissa laitteeseen.

BLE (Bluetooth Low Energy) Vähän virtaa vaativille toiminnoille suunniteltu Bluetoothin kevyempi versio.

CSV (Comma-separated values) Tiedostomuoto, johon tallennetaan ja siirretään taulukkomuotoista dataa.

GAP (Generic Acces Profile) Bluetoothin laiteyhteyden määrittävä profiili.

GATT (Generic Attribute Profile) Bluetooth Low Energyyn kuuluva profiili, joka sisältää tiedonsiirtoon liittyviä määrityksiä laitteiden välillä.

IoT (Internet of Things) Verkosto fyysisistä laitteista, joihin on sulautettuina muun muassa ohjelmistoja ja antureita.

SDK (Software Development Kit) Tyypillisesti joukko sovelluskehityksen työkaluja, jotka mahdollistavat sovelluksen tekemisen muun muassa tietylle alustalla.

SKIIOT Exiopsin kehittämä ja patentoima laite hiihtourheilun työkaluksi.

UI (User Interface) Käyttöliittymä.

USB (Universal Serial Bus) Sarjamuotoinen väylä jonka kautta laitteet voivat kommunikoida keskenään.

UUID (Universally unique identifier) Tietokonejärjestelmissä 128-bittinen osoite yksilöimään tietoa.

XML (Extensible Markup Language) Tekstimuotoinen rakenteellinen kuvauskieli, jolla voidaan jäsentää laajoja tietomassoja.

1 JOHDANTO

SKIIOT on oululaisen Exiopsin patentoima ja toteuttama tuote hiihtourheilun työkaluksi. SKIIOT on Bluetooth Low Energy -laite, jota pidetään nilkassa hiihtosuorituksen aikana, jolloin sen sensorit mittaavat ympäristöstä muun muassa lämpötilaa, ilmankosteutta ja ilmanpainetta sekä hiihtäjän liikettä. SKIIOTilla on Couch4Pron tuottama ohjelmisto ja pilvipalvelu ammattilaisvalmennukseen. (1.)

Exiopsilla oli kuitenkin tarvetta omalle Android-sovellukselle ja erityisesti Polar M600 -kellolle räätälöidylle sovellukselle. Syksyllä 2016 osallistuin Saranen Consultingin Java-HealthTech-koulutukseen, jonka työharjoitteluosuuden suoritin Exiops Oy:ssä. Harjoitteluni käsitti mobiilisovelluksen teknisten ratkaisujen sekä käyttöliittymän suunnittelun ja toteutuksen Exiopsin SKIIOT-laitteelle. Harjoittelujakson jälkeen olen työllistynyt samaisen projektin pariin, jonka aikana toteutin sovelluksen tässä työssä kuvatut ominaisuudet ja toiminnallisuudet. Lähtötilanteessa ei ollut vielä tarkkaa määrittelyä sovellukselle, vaan se tarkentui toteutuksen aikana. Vaikka alun pitäen sovelluksen ensisijainen alusta oli Polar M600 -urheilukello, on painopiste myöhemmässä kehityksessä ollut puhelimella ja tabletilla. Polar M600:n lisäksi sovellus toimii periaatteessa myös muissa Android Wear 1.0 -alustalla varustetuissa älykelloissa, joskaan sen käyttöliittymän suunnittelussa ei ole vielä otettu huomioon mahdollista pyöreää näyttöä.

Tämä työ on rajattu koskemaan ainoastaan mobiilisovelluksen käyttöliittymää, SKIIOT-laitteen yhteyden hallintaa sekä saadun datan käsittelyyn, esitykseen ja tallennukseen liittyviä ratkaisuja ja toteutuksia. Sovelluksen sisältämät ominaisuudet, kuten syke sekä matka ja nopeus, jäävät tämän työn ulkopuolelle.

2 TAVOITTEET

Työn tarkoituksena oli suunnitella ja toteuttaa SKIIOT-laitteen mobiilisovelluksen käyttöliittymä sekä Bluetooth Low Energy -pohjaiseen yhteyteen liittyvät tekniset ratkaisut ja saadun datan tallennus ja esitys.

Työn alkuvaiheessa SKIIOT-laitteelle oli jo olemassa Basic4Androidilla toteutettu sovellus (kuva 1).

KUVA 1. Basic4Android-sovellus SKIIOT-laitteelle

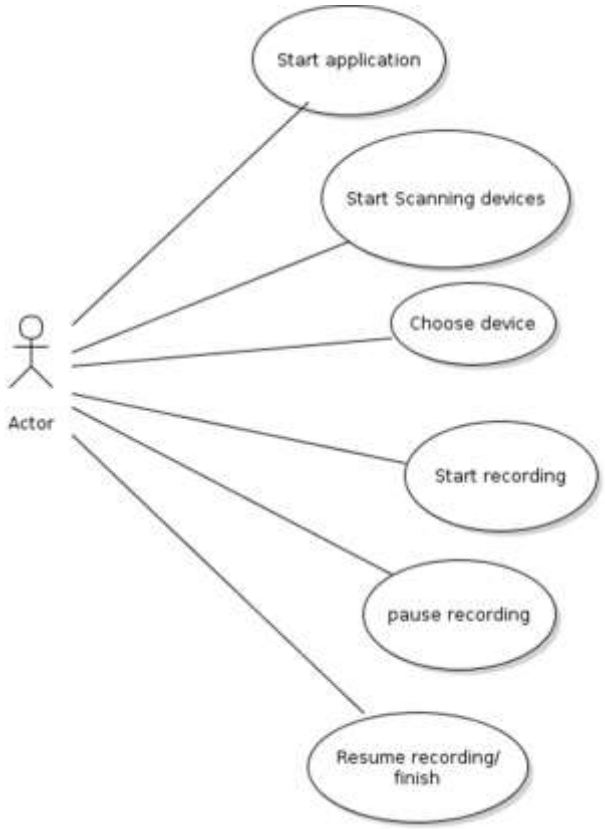
Koska nyt oli tarve kehittää sovellus sekä Polar M600 -urheilukellolle (kuva 2) että puhelimelle, oli tarkoituksenmukaista toteuttaa mobiilisovellus Java-kielisenä, jolloin kehitys olisi mahdollisimman yhdenmukaista molemmilla alustoilla.

KUVA 2. Polar M600 -urheilukello (2)

Käyttötilanteessa laite on monoon asennettuna, jolloin se mittaa hiihdon aikana muun muassa ympäristön lämpötilaa ja kosteutta sekä hiihtäjän liikettä (kuva 3).

KUVA 3. SKIIOT hiihtokenkään asennettuna

Mobiilisovelluksen avulla käyttäjän tulisikin voida valita yhdistettävä laite, yhdistämisen jälkeen aloittaa, lopettaa tai keskeyttää tallennus sen lähettämästä datasta, sekä seurata suorituksen kulkua käyttöliittymästä (kuva 4). Sovelluksen täytyy myös jatkaa tallennusta, vaikka käyttäjä poistuisi sovelluksesta.



KUVA 4. Sovelluksen käyttötapauskaavio

Mobiilisovellukselta vaadittavat ominaisuudet ovat seuraavat:

- BLE-yhteyden muodostaminen SKIIOT-laitteen sekä mobiililaitteen välille
- laitteen sensoreista saatavan datan prosessointi ja saatavien arvojen - kuten lämpötila - reaaliaikaisen keskiarvon laskeminen
- laitteesta saatavan datan tallennus tiedostoiksi, tallennus tietokantaan sekä mahdollisuus tallennettujen tiedostojen selaamiseen sekä tiedostojen sisältämien tietojen graafinen esitys
- käyttöliittymä, joka sisältää käyttäjän valitsemista sensoreista saatavien arvojen reaaliaikaisen esityksen, keskiarvonäkymän sekä tallennuksen ja yhteyden hallinnoinnin
- asetusvalikko haluttuine toiminnallisuuksineen.

Edellä mainittujen ominaisuuksien lisäksi sovelluksen tulisi sisältää matkan ja nopeuden mittaamisen sekä kellossa sykkeen. Nämä jäävät kuitenkin tämän työn tarkastelun ulkopuolelle.

3 KÄYTETYT TEKNOLOGIAT

3.1 Bluetooth Low Energy -tiedonsiirtostandardi

Bluetooth (myös Bluetooth Classic) on lyhyen matkan tiedonsiirtoon suunniteltu langattoman teknologian standardi, joka käyttää tiedonsiirtoon 2,4 GHz:n radiotaajuutta. Bluetooth Low Energy (BLE) on Bluetoothin kevyempi versio ja suunniteltu vähän virtaa vaativille toiminnoille. Bluetooth Low Energy on osa Bluetooth 4.0 -spesifikaatiota. Internet of Things (IoT) eli teollinen internet on fyysisten laitteiden verkosto, jonka laitteisiin on sulautettuina muun muassa ohjelmistoja ja antureita, joiden avulla ne pystyvät muodostamaan yhteyden ja vaihtamaan dataa. Bluetooth Low Energy -ratkaisut ovat kasvattaneet IoT-laitteiden toiminnallisuutta, ja kasvava määrä valmistajia onkin alkanut integroida Bluetooth-teknologiaa IoTiin. (3; 4; 5.)

BLE hyödyntää Bluetooth Classicin tapaan Adaptive Frequency Hopping -käytäntöä, joka lähettää dataa yli 40:llä kanavalla ja tukee 125 kbit/s – 2 Mb/s datanopeuksia. Vähäinen virrankulutus saadaan aikaan sillä, että BLE pysyy lepotilassa, paitsi kun yhteys on aloitettu ja yhteyden keston on keskimäärin muutamia millisekunteja. (5.)

Bluetooth Low Energyn ohjelmointipino sisältää GAP- ja GATT-profiilit. GAP-kerros on vastuussa yhteyteen liittyvästä toiminnallisuudesta, ja GATTia puolestaan käytetään kahden jo yhdistyneen laitteen välisen tiedonsiirron määrittämiseen. (6.)

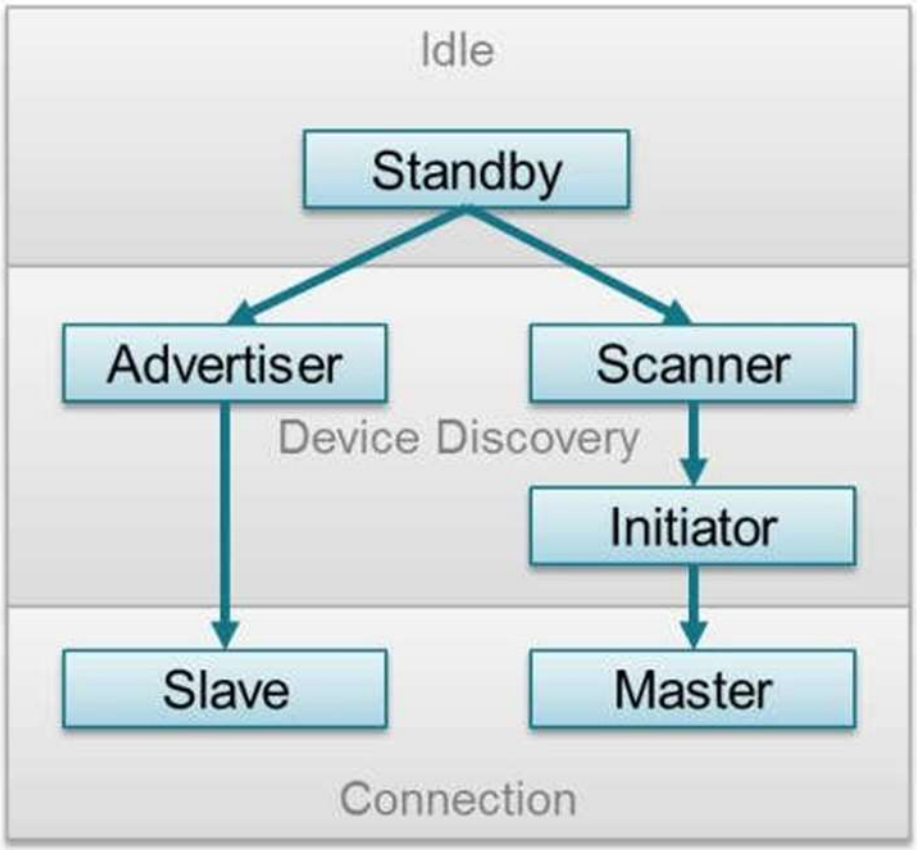
3.1.1 GAP-profiili

Generic Access Profile (GAP) vastaa Bluetoothissa yhteyden toiminnallisuudesta ja siitä, miten laitteet näkyvät ulospäin (mainostaminen) sekä miten ne vuorovaikuttavat keskenään (yhteydet). GAP määrittelee laitteille erilaisia rooleja, joista tärkeimmät ovat keskuslaite ja oheislaite. (7.)

Oheislaitteet ovat pieniä, vähän virtaa vaativia, rajoitetulla resursseilla varustettuja laitteita, jotka voivat yhdistyä keskuslaitteeseen. Oheislaitteita ovat muun muassa sykemittarit. Keskuslaitteita ovat usein puhelin tai tabletti, johon oheislaite on yhdistettynä. (6.)

Laitteella on sille määritellystä roolista riippuen seuraavassa lueteltuja tiloja (kuva 5):

- **Mainostus**. Laite mainostaa itseään erityisellä datalla antaen keskuslaitteen tietää, että se on yhdistettävä laite. Mainostus sisältää laitteen osoitteen ja voi sisältää joitain lisädataa kuten laitteen nimen.
- **Skanneri**. Kun skannaava laite (keskuslaite) saa mainostavan laitteen ilmoituksen, lähettää se pyynnön mainostavalle laitteelle. Mainostava laite vastaa skannaus- vastauksella. Tätä prosessia kutsutaan laitteen havaitsemiseksi. Skannaava laite on tietoinen mainostavasta laitteesta ja voi aloittaa laitteiden välisen yhteyden.
- **Aloittaja**. Kun yhteys aloitetaan, aloittavan laitteen täytyy tarkentaa laitteen osoite, johon yhteys on tarkoitus ottaa. Jos mainostuksella saatu osoite on tämä osoite, lähettää aloitteen tekevä laite pyynnön muodostaa yhteys (linkki) yhteysparametreilla mainostavan laitteen kanssa.
- **Isäntä/orja**. Kun yhteys on muodostettu, mainostava laite toimii orjana ja yhteyden alullepanijana toimii isäntänä. (8.)

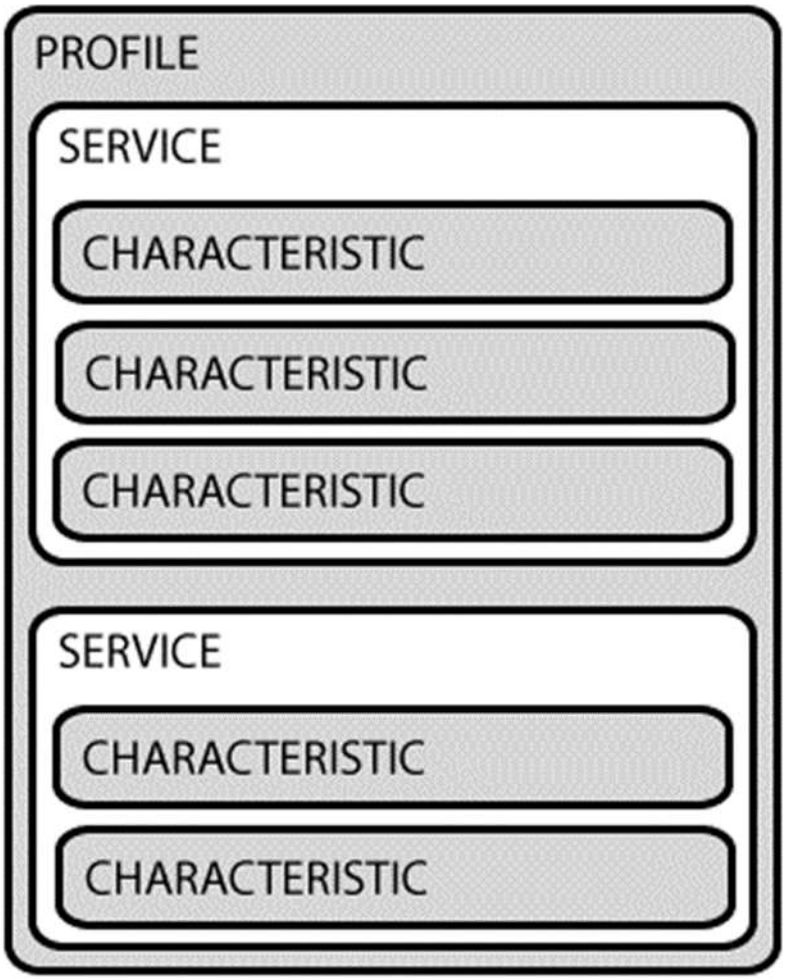


KUVA 5. GAP-tilakaavio (9)

3.1.2 GATT-profiili

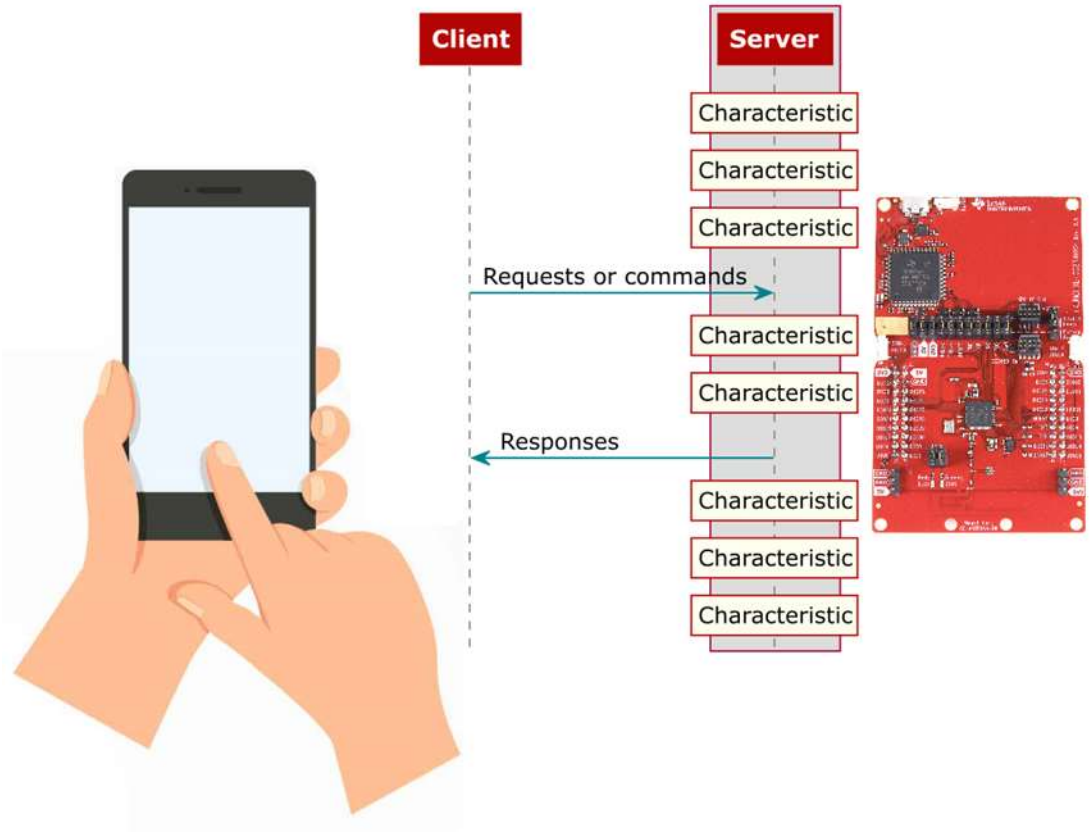
Siinä missä GAP-kerros huolehtii useimmista laiteyhteyteen liittyvistä toiminnoista, GATT-kerros antaa välineet sovelluksille jo yhdistyneiden laitteiden väliseen tiedonsiirtoon. GATT määrittää hierarkkisen rakenteen, joka käsittää seuraavat osat (kuva 6):

- **Profiili** kuvaa käyttötapausta, rooleja ja yleistä GATTiin perustuvaa toiminnallisuutta. Se on ennalta määrätty kokoelma palveluja. Profiili voi olla BluetoothSIG:n tai BLE-laitevalmistajan määrittämä. BLE-laitteella voi toiminnallisuudesta riippuen olla useampi profiili, esimerkiksi laitteella voi olla sekä sykemittari että akun kulutuksen osoittava mittari.
- **Palveluita** käytetään jakamaan dataa loogisiin osiin ja se sisältää characteristicin joita palvelulla voi olla myös useampia. Palvelulla on oma yksilöllinen 128 bitin mittainen osoite, UUID, erottamaan se muista palveluista.
- **Characteristic** on yhden arvon sisältävä tavujono. Se saattaa sisältää myös 0 – N kappaletta deskriptoreja kuvaamaan tätä arvoa.
- **Deskriptorit** ovat määriteltyjä attribuutteja kuvaamaan characteristicin arvoja. (10; 11.)



KUVA 6. GATT (12)

GATT-profiilin näkökulmasta yhdistyneet laitteet ovat joko GATT-palvelin tai GATT-asiakas. GATT-palvelin on laite, joka sisältää characteristicin tietokannan, jota GATT-asiakas voi lukea tai johon se voi kirjoittaa. GATT-asiakas on puolestaan se laite, joka lukee tai kirjoittaa GATT-palvelimelta. Asiakas myös tyypillisesti lähettää pyynnön palvelimelle. Tiedonsiirrossa data kulkee characteristicin muodossa laitteiden välillä. Characteristicit ovat varastoituneet BLE-laitteen palvelimen muistiin. (6;13.) (Kuva 7.)



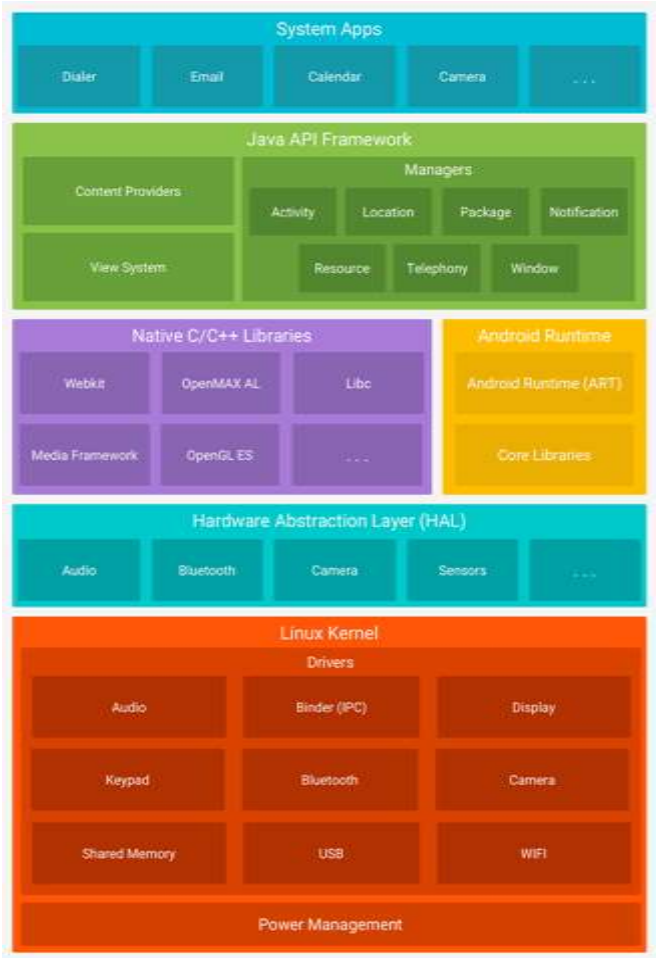
KUVA 7. Palvelin-asiakasmalli (14)

3.2 Android-käyttöjärjestelmä

Android on tämän hetken yleisin mobiilikäyttöjärjestelmä. Androidin kehityksestä vastaa Google yhdessä Open Handset Alliancen kanssa. Android on suunnattu pääasiassa kosketusnäyttöisille laitteille kuten puhelimet ja tabletit. Siitä on kehitetty puettavaan elektroniikkaan suunnattu Android Wear -alusta, jonka yleisimpiä käyttökohteita ovat älykellot. (15.)

3.2.1 Arkkitehtuuri

Vaikka Androidia kutsutaankin käyttöjärjestelmäksi, on se varsinaisesti Linux-ytimen päällä oleva ohjelmistopino, joka on kehitetty tukemaan laajaa joukkoa eri laitteita (16) (kuva 8).



KUVA 8. Androidin ohjelmistopino (17)

Pinon pohjimmaisena on Linux-ydin, jonka päällä on laitteiston abstraktiotaso (HAL). Laitteiston abstraktiotaso tarjoaa yhdenmukaiset rajapinnat, joiden avulla korkeamman tason Java ohjelmistorajapinnoilla on pääsy laitteistoon. HAL käsittää useita kirjastomoduuleja, joista kukin toteuttaa erityyppisen laitteistokomponentin, kuten kamera tai Bluetooth, rajapinnan. Laitteiston abstraktiotason päällä on C-kielisiä ohjelmistorajapintoja muun muassa SQLite-tietokannan hallinnointiin. Käytännössä sovelluskehittäjä pääsee käyttämään näitä kirjastoja Java-pohjaisten kirjastojen ohjelmointirajapinnan kautta tai suoraan Java Native Intefacin (JNI kautta). Android-kirjasto puolestaan sisältää avoimen lähdekoodin Apache Harmony -projektin Java kirjastoja sovelluskehittäjien käyttöön koska Android ei tue JDK:n mukana tulevia Javan standardi kirjastoja. (18.)

Seuraava kerros on sovelluskehyskerros (Java-ohjelmointirajapinta), jonka päällä sovellukset ajetaan, ja joka sisältää Java-yhteensopivia kirjastoja. Sen kautta joukko Android-käyttöjärjestelmän ominaisuuksia on käytettävissä Java-kielisenä. Sovelluskehittäjillä on

pääsy näihin palveluihin, joita he voivat käyttää omissa sovelluksissaan. Androidin Bluetooth-sovelluskehys tarjoaa pääsyn Bluetooth APIen kautta Bluetoothin toiminnallisuuteen. Bluetooth Low Energy on osa Androidia versiosta 4.3 alkaen ja tarjoaa APIn laitteiden löytämiseen, palvelujen pyytämiseen ja datan vaihtoon laitteiden välillä. (19.)

Pinon päällimmäisenä on sovelluskerros, joka sisältää sekä järjestelmäsovellukset että käyttäjän itsensä asentamat niin sanotut kolmannen osapuolen sovellukset (18).

3.2.2 Android Wear -alusta

Wear OS, joka on aiemmin tunnettu nimellä Android Wear, on Android-käyttöjärjestelmän versio, joka on suunnattu nimenomaan puettaville laitteille ja älykelloille (20). Android Wear APIt on saatavilla Android Support Librarystä ja Google Play Servicestä. Kirjastoja käytettäessä puhelimet, joiden Android versio on 4.3 tai suurempi, voivat kommunikoida wearable-laitteiden kanssa. Kommunikointi saadaan aikaan yhdistämällä laitteet, johon tarvitaan Google Playstä saatava Android Wear Smartwatch-sovellus, jonka avulla voidaan puhelimen kautta asentaa sovelluksia Google Play:stä.wearable-laitteeseen. Android Wear 1.0:ssa on tuki muun muassa Bluetooth Low Energy -yhteydelle sekä Wi-Fi- että LTE-yhteyksille. Android Wear 2.0 julkaistiin helmikuussa 2017, jonka myötä kelloista on tullut entistä riippumattomampia ja muun muassa sovellusten asennus onnistuu suoraan kellosta Android Wearin omasta Google Play -sovelluskaupasta. Android Wear 2.0:ssa on myös parempi tuki yhä yleistyvälle pyöreälle kellon näytölle. (21; 22.)

3.2.3 Android-sovellukset

Android-sovellus on Android-käyttöjärjestelmässä ajettava ohjelmisto. Tyypillisesti sovellus on tehty puhelimelle tai tabletille. Sovellusten virallinen kehitysympäristö on Android Studio. (23.)

Android-sovellus koostuu neljäntyyppisestä komponentista:

- aktiviteetti
- palvelu
- broadcast receiver (lähetyksen vastaanottaja)
- content provider (Sisällön tarjoaja).

Sovelluksilla on pääsy ainoastaan niihin komponentteihin, mitä se kulloinkin tarvitsee, jolloin sovellus ei pääse käyttämään systeemin niitä osia, mihin sillä ei ole oikeuksia. Sovellusten oikeudet on määritelty manifestissa. (24.)

Manifesti

Jokaisella sovelluksella täytyy olla AndroidManifest.xml-tiedosto, joka sijaitsee Android Studio -projektin lähdekoodikansioiden juuressa. Manifesti kuvailee olennaisimmat tiedot sovelluksesta Androidin koontityökaluille, käyttöjärjestelmälle ja tarvittaessa Google Playlle. (25.)

APK-paketit

Jakelua varten Android-sovellukset pakataan apk-päätteisiksi tiedostoiksi. APK-paketit ovat JAR-tiedostoihin pohjautuvia zip-tiedostoja, jotka sisältävät kaiken, mikä kuuluu sovellukseen. Laitteet, joissa on Android-käyttöjärjestelmä, käyttävät APK-paketteja sovelluksen asentamiseen Sovellusten yleisin ja virallinen jakelukanava on Google Play -sovelluskauppa. (26.)

Aktiviteetti

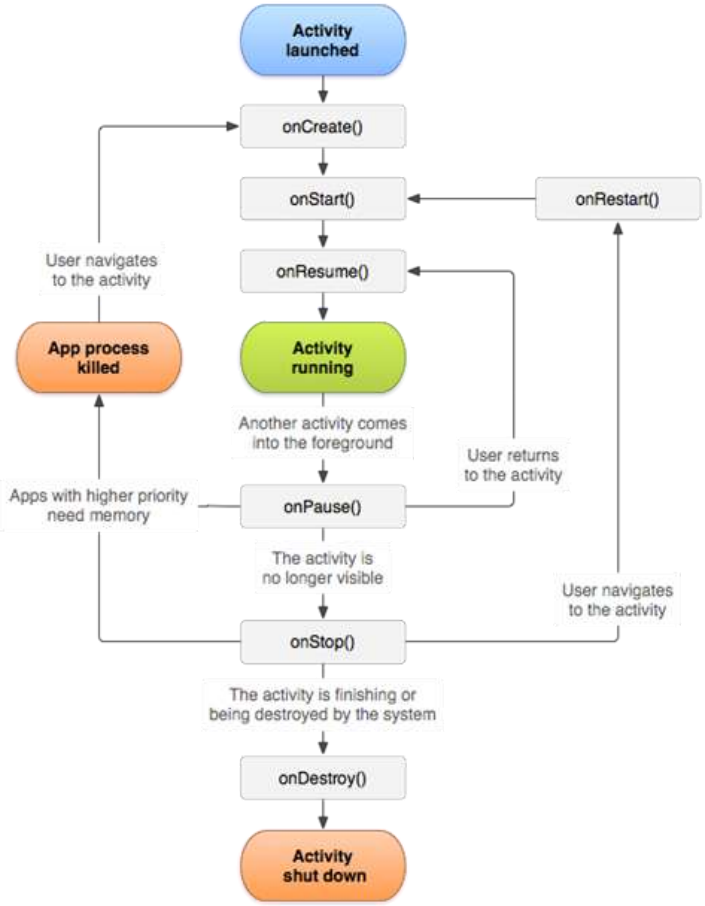
Käyttöliittymän pääasiallinen komponentti on Activity-luokka, joka vastaa ikkunoinnista sekä on käyttäjälle näkyvä toiminnallinen kokonaisuus, aktiviteetti. Activity-luokkia voi sovelluksella olla useita ja ne täytyy määritellä manifestissa. Manifestissa myös määritetään, mikä aktiviteetti käynnistyy sovelluksen alussa. Usein tämä on niin kutsuttu pääaktiviteetti, joka on sovelluksen oletusnäkymä. (27.)

WearableActivity

WearableActivity on perusaktiviteettiluokka wearable-sovelluksille ja se tarjoaa yhteensopivuuden Ambient Mode -tuelle. (28.)

Aktiviteetin elinkaari ja palvelut

Aktiviteetilla on elinkaari, joka pitää sisällään erilaisia tiloja, jotka vaihtelevat käyttäjän liikkuessa sovellusten välillä (kuva 9). Aktiviteetti saa tiedon muuttuneesta tilasta Activity-luokan valmiiksi määriteltyjen callbackien avulla.



KUVA 9. Aktiviteetin elinkaari (29)

Aktiviteetin elinkaarenmetodeja ovat seuraavat:

- **onCreate(Bundle)**, jossa muun muassa asetetaan setContetnView(Int)-metodilla aktiviteetin layout. Layout annetaan parametrina XML-tiedoston kokonaislukuarvoisena resurssina. findViewById(Int)-metodilla, joka saa puolestaan parametrinaan View'n kokonaislukuarvoisen resurssin, voidaan puolestaan etsiä tästä layoutista sen sisältämät aktiviteetin näyttöliittymäkomponentit, jolloin niitä voidaan käsitellä ohjelmallisesti
- **onPause()**, jossa huolehditaan toiminnoista, jotka täytyy tehdä, kun käyttäjä lähtee aktiviteetista.
- **onResume()** metodia kutsutaan, kun käyttäjä alkaa vuorovaikuttamaan aktiviteetin kanssa. onResume-metodia kutsutaan aina, jos onPause()-metodia on kutsuttu.

- **onDestroy()** on viimeinen metodikutsu ennen aktiviteetin tuhoutumista. Tämä voi tapahtua, mikäli kutsutaan lopetus-metodia tai mikäli systeemi tarvitsee tilaa ja tuhoaa aktiviteetin. (30.)

Mikäli sovellukselle on tärkeää suorittaa toimintoja jos sovellus ei ole aktiivisena, täytyy toiminnot toteuttaa aktiviteetin elinkaaren ulkopuolella. Tähän voidaan käyttää palveluja (Services). Palvelu on Android-järjestelmän komponentti, joka ei sisällä käyttöliittymää ja jonka avulla voidaan suorittaa pitkään kestäviä operaatiota taustalla. Sovellus voi aloittaa palvelun, joka jatkaa suoritustaan taustalla vaikka käyttäjä vaihtaisi toiseen sovellukseen. (31.)

Palveluja on kolmea eri tyyppiä: Foreground-palvelu suorittaa joitakin käyttäjälle näkyviä operaatioita, Background-palvelu suorittaa operaatioita, jotka eivät näy suoraan käyttäjälle. Kolmas palvelutyyppi on Bound eli sidottu. Palvelu on sidottu, kun sovelluksen komponentti sidotaan siihen kutsumalla bindService-toimintoa. Sidottu palvelu tarjoaa asiakas-palvelinrajapinnan, joka mahdollistaa komponenttien vuorovaikutuksen palvelun kanssa, lähettää pyyntöjä, ja ottaa vastaan vastauksia. Sidottua palvelua suoritetaan ainoastaan niin kauan, kuin jokin sovelluksen komponentti on sidottuna siihen ja kun sidos puretaan (unbind) tuhoutuu palvelu. (31.)

Palvelu luodaan luomalla Service-luokan aliluokka tai käyttämällä joitain sen valmiista aliluokista. Aliluokassa täytyy ylikirjoitaa joitakin callback-metodeja jotka pitävät huolen palvelun elinkaaren pääkohdista ja tarjoavat mekanismin joka mahdollistaa komponentin sitomisen palveluun. Näistä tärkeimpiä ovat:

- **onStartCommand().** Kun aktiviteetista (tai muusta komponentista) kutsutaan onStartService()-metodia, aloittaa se onStartCommand()-metodin. Tätä metodia kutsutaan, kun halutaan ajaa palvelua taustalla siihen asti, kunnes se lopetetaan kutsumalla stopSelf() tai yleisemmin stopService()-metodia.
- **onBind()**-metodi aktivoituu, kun aktiviteetista (tai muusta komponentista) kutsutaan bindService()-metodia. Tämän metodin toteutuksessa täytyy tarjota rajapinta, jolla voidaan kommunikoida palvelun kanssa. Rajapinnan kautta palvelu palauttaa iBinder-olion, jonka kautta kommunikointi toteutuu. onBind()-metodi täytyy aina toteuttaa palvelussa. Mikäli binding-toimintoa ei haluta käyttää, palauttaa onBind() silloin tyhjän.

- **onCreate()**-metodia kutsutaan kerran, kun palvelu on ensimmäisen kerran luotu suorittamaan tarpeelliset toimenpiteet ja ennen onStartCommand() ja onBind()-metodien kutsuja. Mikäli palvelu on jo aloitettu, metodia ei kutsuta.
- **onDestroy()**-metodi käynnistyy, kun palvelua ei enää käytetä ja se tuhotaan. Tässä metodissa pitäisi huolehtia palvelun käyttämien resurssien vapauttamisesta. (31.)

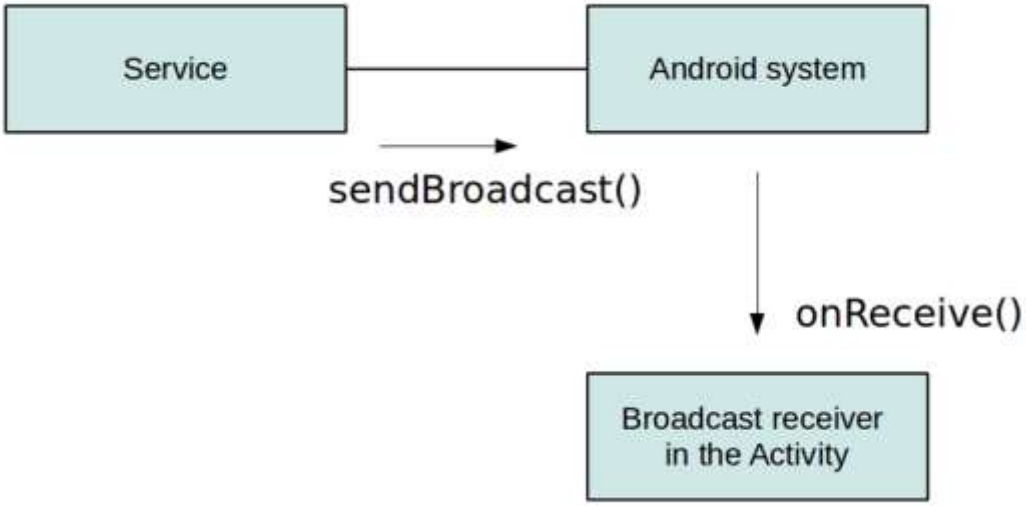
Fragmentit

Aktiviteetti voidaan jakaa itsenäisiin osiin, fragmentteihin. Tämä on hyödyllinen ominaisuus, jolla voidaan lisätä koodin selkeyttä. Fragmenteilla on oma elinkaarensa, joka on osin riippuvainen aktiviteeteista, johon ne kuuluvat. Mikäli aktiviteetti tuhoutuu, tuhoutuvat myös siihen kuuluvat fragmentit, eikä pysäytetyssä aktiviteetissa voida aloittaa fragmenttia. Fragmenttien hallinnointi aktiviteetissa tapahtuu FragmentManagerin kautta. FragmentManager saadaan Activity.getFragmentManager() tai Fragment.getFragmentManager() -toimintojen avulla. Fragmenttien asettaminen tai vaihto aktiviteetin sisällä tapahtuu FragmentManagerin beginTransaction()-metodista saatavan FragmentTransaction-olion replace()-metodin avulla, joka saa parametrinaan fragmentin säiliönä toimivan näyttökomponentin resurssin ja vaihdettavan tai asetettavan fragmentin instanssin. Eräitä fragmentin elinkaareen liittyviä metodeja ovat

- **onCreateView()** palauttaa fragmentin layouttiin liittyvän View-hiearkkian
- **onViewCreated()** kutsutaan, kun onCreateView() on suoritettu. (32.)

Broadcast ja broadcast receiver

Android-sovellukset voivat lähettää ja vastaanottaa broadcast-lähetyksiä. Lähetykset voivat olla järjestelmältä sovellukselle tai sovellusten välisiä. Lähetykset ovat myös tapa viestittää sovelluksen sisällä esimerkiksi palvelimelta aktiviteetille (kuva 10). (33.)



KUVA 10. Broadcast receiver (34)

Lähetyksen vastaanottamiseen on omistettu BroadcastReceiver-luokka, jonka instanssi voidaan rekisteröidä vastaanottamaan lähetyksiä joko dynaamisesti Context.registerReceiver()-metodin avulla tai staattisesti manifestissa. Dynaamisesti rekisteröitäessä sitä kutsutaan esimerkiksi aktiviteetin tai fragmentin onResume()-metodissa, jolloin lähetysten lopettamiseksi täytyy onPause()-metodissa kutsua unregisterReceiver(BroadcastReceiver)-metodia. (35; 33.)

Käyttöliittymän rakennuspalikat View ja Layout

View-luokka on käyttöliittymän suorakaiteenmuotoinen peruselementti, joka vastaa näytöllä komponenttien piirtämisestä sekä tapahtuman käsittelystä (36). Layoutit puolestaan määrittävät Androidissa käyttöliittymän rakenteen sisältäen hierarkkisen rakenteen näkymistä (View) sekä toisista layouteista. Koodin selkeyttämiseksi layoutit voidaan määritellä xml-tiedostoihin, jolloin toiminnallisuus voidaan pitää koodissa erillään näytön määrittävästä layoutista. (37.)

Sovellusta käännettäessä layoutin xml-tiedostot käännetään View-resursseiksi. Kun aktiviteetin layout halutaan ladata, annetaan Activity-luokan onCreate()-metodissa setContentView(int)-metodille parametrina viittaus aktiviteetin layoutin resurssiin. Fragmenttien layout puolestaan ladataan fragmentin onCreateView()-metodissa View'n palauttavalla

LayoutInflaterin inflate()-metodilla, joka myös saa parametrinaan viittauksen fragmentin layoutin resurssin. (27; 32.)

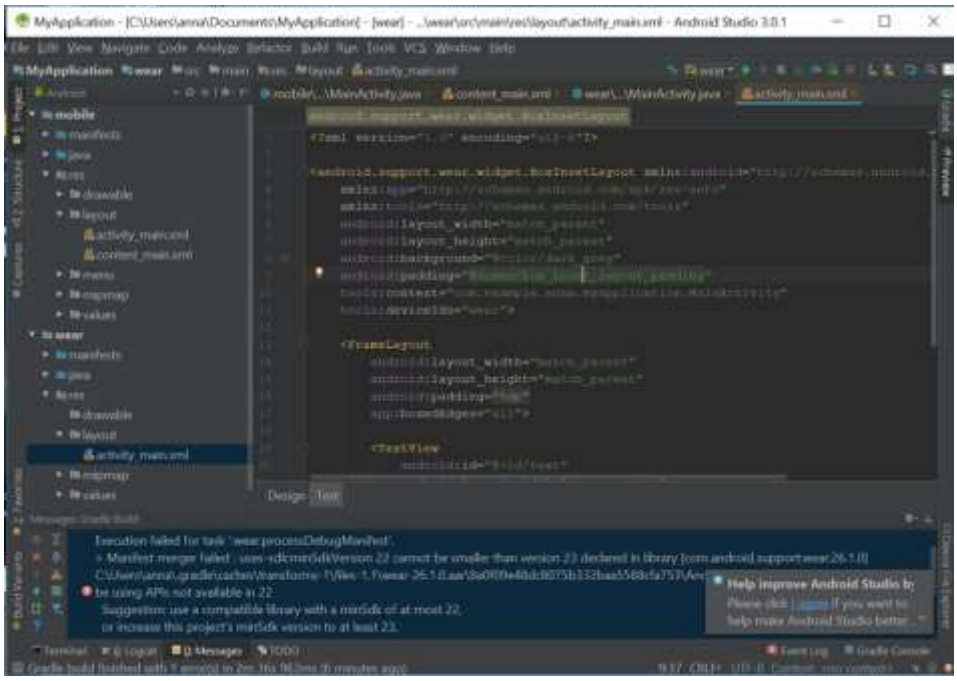
3.2.4 Sovelluskehitys Androidissa

Android SDK sisältää Android-sovelluksen tekemiseen tarvittavat välineet, muun muassa Platform tools, Build tools, SDK tools, The Android Debug Bridge(ADB) ja Android Emulator. Näistä tärkein on SDK tools, joka varsinaisesti luo mobiililaitteeseen asennettavan APK:n Java-lähdekoodista. (38.)

ADB:n (Android Debug Bridge) avulla voidaan siirtää kehitettävä sovellus esimerkiksi USB:n kautta laitteeseen. Tämä vaatii, että laitteessa on USB-virheenkorjaus päällekytkettynä asetuksista. (39.)

Android-sovellusten virallinen kehitysympäristö on Android Studio, joka toimii IntelliJ:n koodieditorin ja kehitystyökalujen päällä. Kun uusi projekti luodaan, Android Studio luo tarvittavan rakenteen tiedostoille ja tekee ne näkyväksi projekti-ikkunassa (kuva 11). (23; 40).

Android Studio käyttää gradle-koontityökalua projektien hallintaan ja lähdekoodin kääntämiseen. Build.gradle-skriptitiedosto määrittelee projektin ja sen tehtävät ja siinä voidaan myös määritellä sovelluksen riippuvuudet sekä mahdolliset tarvittavat ulkopuoliset kirjastot. (41;42.)



KUVA 11. Android Studio

Moduuli

Moduuli on kokoelma lähdetiedostoja ja asetuksia, jotka mahdollistavat projektin jakamisen erillisiin toiminnallisiin osiin. Projektissa voi olla yksi tai useampi moduuli ja moduuli voi olla riippuvuussuhteessa toiseen moduuliin. Jokainen moduuli voi olla itsenäisesti käännetty ja testattu. Lisämoduuleja tarvitaan, kun halutaan luoda koodikirjastoja tai jos halutaan luoda eri koodit ja resurssit eri laitetyypeille, kuten puhelimille ja puettaville laitteille, mutta kuitenkin pitää kaikki tiedostot samassa projektissa ja jakaa koodia. Android Studiossa moduulien käyttö helpottaa uusien laitteiden, kuten Android Wear tai Android TV, lisäämistä projektiin. Android Studio lisää automaattisesti moduuliin tarvittavat hakemistot ja build.gradle-tiedoston (kuva 12). Luotaessa uutta projektia Create New Module -ikkunassa Android Studio tarjoaa seuraavat moduulit:

- Phone & Tablet (app)-moduuli
- Android Wear (wear)-moduuli
- Android TV moduuli
- Glass-moduuli.

Jokainen näistä tarjoaa olennaiset tiedostot ja joitain koodimallia, jotka ovat sopivia vastaavan sovelluksen tai laitteen tyypille. (43.)

Android Wear 1.0 -sovellusta luotaessa projektiin luodaan sekä wear- että mobile-moduulit, jolloin mobile-moduuliin voidaan sijoittaa asennuksen yhteydessä tarvittava koodi wear-moduulin sisältäessä varsinaisen koodin. Mobile-moduuliin voidaan myös sijoittaa mahdollinen wearable-laitteen kanssa vuorovaikuttava toiminnallisuus. Lisäksi build.gradle-tiedostoon täytyy lisätä Wear UI Library- ja Support Library -riippuvuudet (kuva 12). (44.)

```
dependencies {
   compile 'com.google.android.gms:play-services-wearable:10.0.1'
   compile 'com.android.support:support-compat:25.1.0'
   wearApp project(':wearable')
```

KUVA 12. Wear-sovelluksen riippuvuudet (44)

Kirjastomoduuli

Android-kirjastomoduuli on rakenteellisesti samankaltainen kuin app- tai wear-moduuli, mutta se ei käänny APK-tiedostoksi vaan arkistotiedostoksi eikä siten ole itsenäisenä asennettavissa laitteelle. Android-kirjastoja on kahdenlaisia: Android Library, joka voi sisältää kaiken tyyppisiä tiedostoja, jotka kuuluvat projektiin. Android Libraryn käännösprosessi tuottaa Android Archive (AAR)-tiedoston, jonka voi tuoda riippuvuutena app-moduuleihin. Toinen on Java Library: Tämän tyyppinen kirjasto voi sisältää ainoastaan Java-lähdetiedostoja. Se kääntyy Java Archive (JAR) -tiedostoksi, jonka voi lisätä riippuvuutena app-moduuleihin tai muihin Java projekteihin. (43.)

4 SKIIOT- JA POLAR M600 -LAITTEET URHEILUN SEURANTAAN

4.1 SKIIOT

SKIIOT on GATT-spesifikaation mukainen BLE-laite. Kun laite laitetaan päälle, alkaa se mainostamaan itseään, jolloin keskuslaitteena toimiva mobiililaite voi havaita sen ja GA-Pin mukainen yhteydenotto voidaan aloittaa. SKIIOT-laitteesta saadaan tällöin nimi ja osoite. Yhteyden muodostuttua SKIIOT-laite toimii orjana mobiililaitteen toimiessa isäntänä.

GATT-profiilin näkökulmasta SKIIOT-laite toimii palvelimena, joka sisältää sensorien antamien ympäristö- ja liikedatan lukuarvot sisältävistä characteristeceista koostuvan tietokannan. Tästä tietokannasta se lähettää dataa 20 Hz:n ja 1 Hz:n nopeuksilla asiakaslaitteelle. Eri sensorien arvot on määritelty tiettyihin charactersticin tavuihin, jolloin yhteen lähetykseen saadaan useampi sensoriarvo. Lähetettäessä kahta eri taajuutta saadaan lähetyksiin vielä useamman sensorin arvo tarvitsematta kasvattaa lähetettävän characteristicin pituutta.

SKIIOT-laitteen sensorit ovat

- barometri
- Infrapunalla mitattavan lumen lämpötila
- kiihtyvyysmittari
- gyroskooppi
- magnetometri.

Näiden lisäksi on vielä valinnaisena lisäominaisuutena kosteus- ja lämpötilamoduuli. (1.) (Taulukko 1.)

TALUKKO 1 SKIIOT-laitteen sensorit

Sensori	**Arvot**	**Nopeus/Hz**
Barometri	Ilmanpaine (suhteellinen korkeus) ja lämpötila	1
Lumen lämpötila (infrapuna)	Lämpötila	20
Ilman kosteus	Ilman kosteus ja lämpötila	1
Ulkoinen lämpötila	Lämpötila	1
Gyroskooppi	X,y ja z koordinaatit	20
Magnetometri	X,y ja z koordinaatit	20
Kiihtyvyysanturi	X,y, ja z koordinaatit	20

SKIIOTin akku ladataan microUSB:n kautta ja se kestää 35 tuntia. SKIIOTissa on myös 512 MB:n muisti sekä Wi-Fi- ja BLE-tuki (1).

4.2 Polar M600

Polar M600 on Polarin kehittämä Android Wear -alustainen urheilukello. Kellon ominaisuuksiin kuuluu muun muassa 4GB:n sisäinen tallennustila ja 512 MB RAM, 1,3-tuumainen näyttö, jonka tarkkuus on 240 x 240 pikseliä sekä 500 mAh:n akku. Kellon sisältää optisen sykemittarin, kiihtyvyysmittarin, gyroskoopin ja ympäristön valotunnistimen. Näyttö on kosketusnäyttö. Kellossa on Wi-Fi 802.11 b/g/n sekä Bluetooth 4.2 eli myös Bluetooth Low Energy -tuki. Kellon käyttöjärjestelmä on täysin Android Wear -yhteensopiva. Sovelluskehitys-ympäristö on Android Studio sekä ohjelmointikieli kieli Java. APK pakettien asennus kelloon tapahtuu joko USB-kaapelin tai Bluetoothin kautta, jolloin liitettyyn puhelimeen tarvitaan asentaa Android Wear App -sovellus. Virheenkorjaus tapahtuu ADB rajapinnan kautta. (45.)

5 ANDROID-SOVELLUKSEN TOTEUTUS SKIIOT-LAITTEELLE

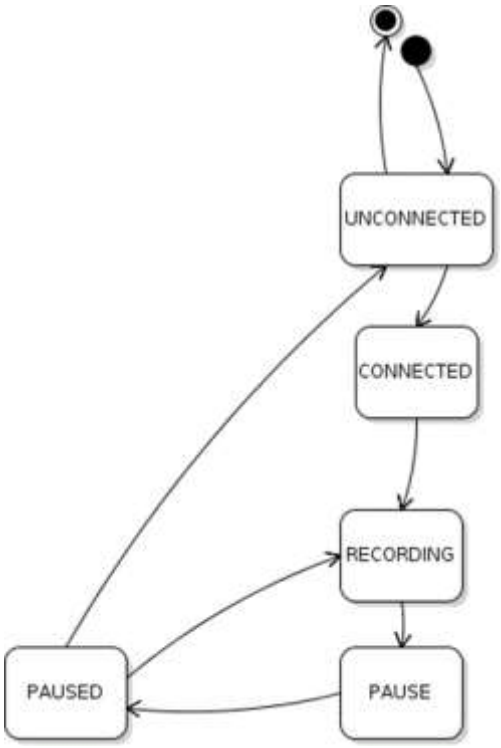
Toteutuksen kehitysympäristönä käytettiin Android Studiota. SKIIOT-laitteesta käytössä olivat versiot 1.98, 2.85 ja 2.8C. Testilaitteina olivat Samsung S5- ja Samsung S6 puhelimet sekä Polar M600 -älykello, jonka käyttöjärjestelmänä oli Android Wear 1.0.

Alkutoimenpiteenä lisättiin SKIIOT-laitteelle jo olemassa olevan Basic4Android-sovelluksen git-projektiin proto-kansio, joka tulisi sisältämään uuden sovelluksen tarvitsemat lähdekoodit ja Android Studio -projektit. BLE-yhteyden vaatimat tiedot sekä matemaattiset menetelmät saadun datan käsittelyä varten saatiin myös Basic4Android-sovelluksen lähdekoodista. Sovelluksen Android Wear -projektin testauta varten yhdistettiin (pair) Polar M600 testilaitteena toimivan puhelimen kanssa. Tämä edellytti Android Wear -sovelluksen asentamisen testipuhelimeen. Polar M600 -kellossa suoritettiin tarvittavat toimet BLE-asetusten suhteen sekä USB-virheenkorjauksen päällekytkemiseksi, jonka kautta kehitettävä sovellus voidaan asentaa kelloon.

5.1 Sovelluksen rakenne ja toiminnallisuus

Sovelluksen perustaksi valittiin ainoastaan yksi aktiviteetti. Se sisältää sekä fragmenttien ja palvelujen hallinnoinnin sekä laiteyhteyden muodostamiseen ja hallinnointiin liittyvät toiminnot. Sovelluksen muu toiminnallisuus jaettiin neljään fragmenttiin. GaugeFragment ja GraphFragment sisältävät sensorien arvojen esityksen, IconFragment sisältää ikonit GPS:n sekä akun tilalle sekä asetusvalikon ja kalenteri-ikonin ja lopuksi ConnectFragment sisältää käyttäjän aktivoimien toimintojen hallinnoinnin laiteyhteyden ja tallennuksen osalta. Fragmentit toimivat näin sekä toiminnallisuuden osiin jakavina rakenteina että käyttöliittymän näyttöelementtien säiliöinä. Sovelluksen laiteyhteys sekä tallennukseen liittyvät toiminnot toteutettiin palvelujen avulla, jolloin toiminnot eivät ole riippuvaisia aktiviteetin elinkaaresta.

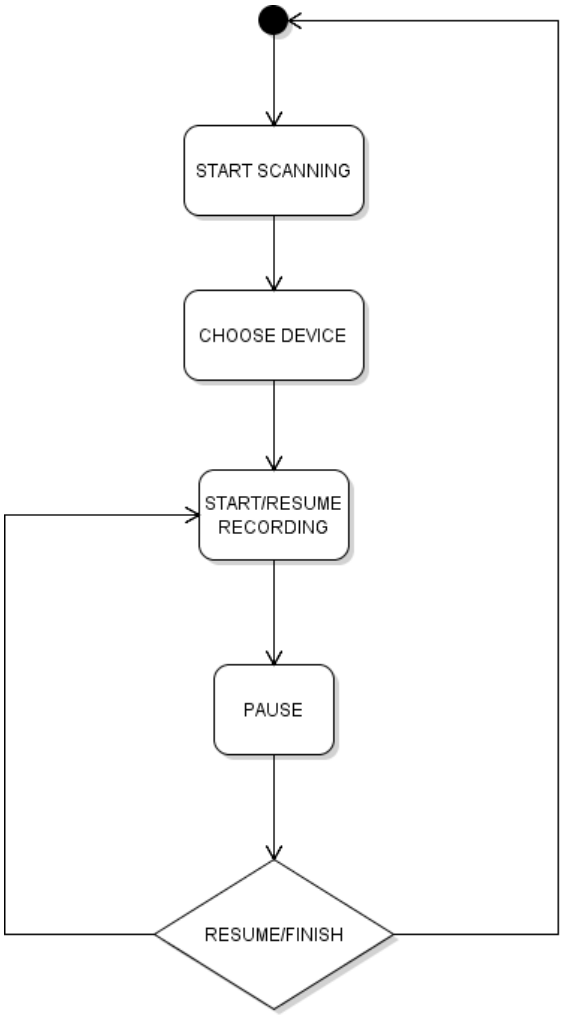
Eri toiminnallisuuksien ja palveluiden yhteensovittamiseksi täytyy sovelluksen pystyä pitämään kirjaa sekä yhteyden- että palveluiden tilasta. Tämä toteutettiin määrittämällä Utils-luokassa kokonaislukuvakiot UNCONNECTED, CONNECTED, RECORDING, PAUSE ja PAUSED (kuva 13).



KUVA 13. Sovelluksen tilakaavio

Tilat voivat muuttua callbackien kautta (BLE-yhteys) tai käyttäjän toimesta (tallennuksen aloitus ja keskeytys). Toiminnallisuuden käyttöliittymänä toimii kontrollipainike, jonka avulla käyttäjä hallinnoin laiteyhteyttä ja tallennusta. Poikkeuksena tähän on tallennuksen pysäytystila, josta kontrollipainikkeen painalluksesta avautuu valikko tallennuksen lopettamiseen tai jatkamiseen.

Siinä missä edellä mainitut viisi tilaa määrittävät sovelluksen kulloisetkin sisäiset toiminnot, voidaan sovelluksen käyttäjärajapinnan toiminnot jakaa myös viiteen osaan: ”START SCANNING”, ”CHOOSE DEVICE”, ”START RECORDING” sekä ”RESUME/FINISH”. (kuva 14)



KUVA 14. Aktiviteettikaavio

Koska laiteyhteys ja tallennus on toteutettu palveluna, voi käyttäjä poistua sovelluksesta tallennuksen siitä keskeytymättä. Käyttäjän lopettaessa tallennuksen tallennetaan tiedostot tietokantaan. Mikäli halutaan keskeyttää tallennus ja tuhota jo luodut tiedostot, täytyy sovelluksesta poistua asetusvalikon lopetus-toiminnon kautta, jolloin palvelut lopetetaan ja sovellus palautuu UNCONNECTED-tilaan.

5.2 Käyttöliittymä

Graafisen käyttöliittymän haaste oli esittää eri sensorien arvot sekä muu tarvittava tieto havainnollisesti sekä tarjota käyttäjälle selkeä tapa hallinnoida toimintoja. UI on suunniteltu ja toteutettu ensin Polar M600:lle. Saatu ratkaisu sovitettiin puhelimen ja tabletin näytöille, jotka itsessään pitivät sisällään laajan kirjon eri näyttökokoja.

5.2.1 Käyttöliittymän suunnittelun lähtökohtia

Ensimmäiseksi toteutettu käyttöliittymä (kuva 15) sisälsi listamaisen näkymän halutuista arvoista.

KUVA 15. Listamainen käyttöliittymä

Ratkaisussa oli lista sensoriarvoista, asetusnäkymä josta valitaan listaan haluttujen sensorien arvot sekä keskiarvonäkymä. Käyttöliittymä oli ensimmäinen kellolle toteutettu sovelluksen osa, ja sen toteutus käsitti myös joihinkin Android Wear -alustalle ominaisiin ratkaisuihin tutustumista. Näihin kuului muun muassa GridViewPager, joka on Android Wear -alustan layout, joka sallii sekä pysty- että sivusuuntaisen liikkumisen näyttöjen välillä (46). Nämä ratkaisut jäivät kuitenkin pois myöhemmässä vaiheessa, jotta kellon ja puhelimen koodi saatiin pidettyä yhdenmukaisempana

Listamainen käyttöliittymä ei kuitenkaan hyödyntänyt tarpeeksi kellolle ominaista näyttöä. Koska tavoitteena oli myös saada kaikki olennainen tieto yhteen näkymään, vaati tämä myös komponenttien uudenlaista suunnittelua ja sijoittelua mielekkääksi kokonaisuudeksi kellon näytöllä. Komponenttien valinnassa oli myös tavoitteena löytää esitys, joka olisi kuvaava kyseiselle sensorille ja sen arvolle ja toisaalta komponentin sijoittelu näytöllä heijastelisi kyseisen tiedon tärkeyttä käyttäjän kannalta. Värien lisäämisellä komponenteihin oli tarkoitus havainnollistaa visuaalisesti suorituksen tilaa siten, että värit vaihtuisivat kyseisen komponentin senhetkisen arvon mukaan. Lisäksi komponenttien värien yhteisvaikutus kertoisi käyttäjälle suorituksen senhetkisen tilan ja käyttöliittymä ikään kuin eläisi suorituksen mukana. Näytön suunnittelussa oli myös pyrkimys pitää koodi ja graafinen ilme mahdollisimman yhdenmukaisena kellosta tabletteihin. Tämä tarkoitti laajaa skaalautuvuutta. Tämä myös merkitsee, että kellon käyttöliittymän kohdalla ei ole paljoa hyödynnetty Android Wearin ominaisuuksia.

Käyttöliittymän perusidea komponenttien värien kautta käyttäjälle välittyvästä kokonaistilasta riippuu minimi ja maksimiarvojen sekä ihannearvojen valinnasta.

5.2.2 Käyttöliittymän toteutus

Koska käyttöliittymä pohjautui alkujaan kellon näytölle, ja pyrkimys oli pitää koodi mahdollisemman yhtenäisenä, käyttöliittymä rakennettiin kellon päänäytön ympärille. Kellon päänäyttö sisälsi mittarinäkymän, ikonit sekä kontrollipainikkeen. Mittarinäkymä ja ikonit toteutettiin FrameLayoutina, jolloin voidaan asetella päällekkäin View-olioita tai toisia layoutteja (47). Näiden layouttien muodostama looginen näyttökomponentti on molemmilla alustoilla sama ja sen koko määräytyy näytön leveyden mukaan. Pääasiallinen ero alustojen käyttöliittymässä on muiden näyttökomponentit sisältävien fragmenttien sijoittelu. Kontrollipainikkeen sisältävä fragmentti on sijoitettuna molemmilla alustoilla näytön alalaitaan, kellossa päänäytön päällimmäisenä komponenttina.

Puhelimen ja tabletin käyttöliittymän pohjalla on pääaktiviteetin layout, joka sisältää mittarinäkymän, graafin ja kontrollipainikkeen sisältävien fragmenttien layoutit allekkain. Tämä toteutettiin LinearLaytilla, joka asettelee siihen kuuluvat View’t ja layoutit riviin joko horisontaalisesti tai vertikaalisesti. (48).

Kellossa käytettiin aktiviteetin layouttina android.support.wear.widget.BoxInsetLayoutia, joka on näytön mallin huomioon ottava layout (49). Kontrollipainikkeen sisältävää fragmenttia lukuunottamatta fragmenttien layoutit täyttävät koko näytön ja ovat toteutettuina layoutit päällekkäin asettelevalla FrameLayoutilla. Kunkin senhetkisen näkymän muodostavien fragmenttien välillä käyttäjä liikkuu ViewGroup.dispatchTouchEvent(MotionEvent event)-metodilla toteutetulla pyyhkäisytoiminnolla. Toiminto aktivoi FragmentTransaction.replace()-metodin, joka suorittaa vaihdon fragmenttien välillä.

Datan esitys käyttöliittymässä

Mittarinäkymän mittarin sisältävä GaugeFragment-luokka on kellolle ja puhelimelle yhteinen siinä missä graafin sisältävä GraphFragment on tällä hetkellä toteutettu ainoastaan puhelimelle. GaugeFragment sisältää näyttökomponenttinaan GaugeView’n ja GraphFragmentin päänäyttökomponentti on toteutettu GraphView-kirjaston avulla. GraphView-kirjasto on avoimen lähdekoodin kirjasto Androidille kaavioiden piirtämiseen, jonka varsinaisena näyttökomponenttina toimii GraphView-luokka (50). GaugeView on GraphView’n tavoin kustomoitu View-luokan aliluokka. Mukautettu View’n aliluokka oli

tarkoituksenmukainen valinnaksi skaalautuvuutensa vuoksi, jolloin sama koodi käy kaikkiin näyttötyyppeihin. GaugeView (kuva 16) koostuu päämittarista, sekä sen sisälle sijoitetuista komponenteista, jotka on tässä rajattu keltaisella. GaugeView'n komponentit toteutettiin piirtämällä ne View'n onDraw()-metodissa.

.

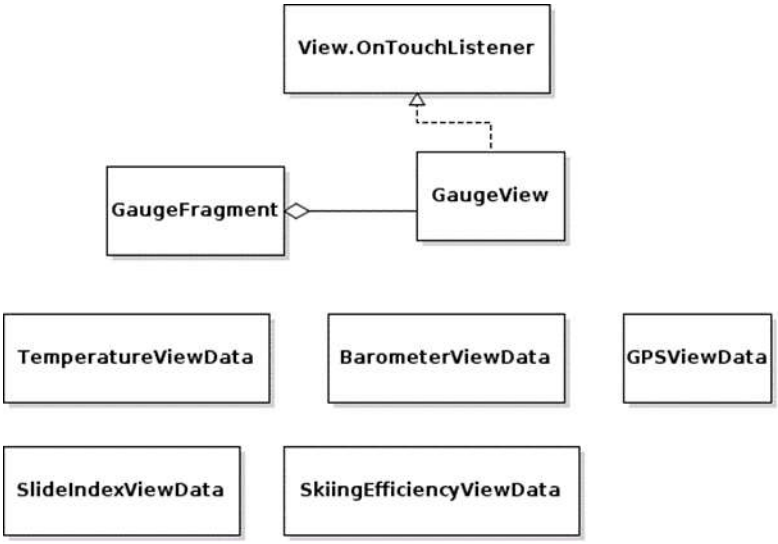
KUVA 16. GaugeView

Koska SKIIOTin tärkeimpiä ominaisuuksia on suoritusta kuvaavat luisto- ja suoritusindeksit, oli ne mielekästä sijoittaa päämittarin viisareiksi. Loput sensoreista saatavat arvot sekä matka ja nopeus ja kellossa syke, sijoitettiin kolmeen kuvassa keltaisella rajattuun ryhmään, joita kutakin esittää näytöllä yksi looginen komponentti kutakin ryhmää kohti.

Ryhmien idea on, että käyttäjä voi pitkällä painalluksella vaihtaa ryhmän sisällä sensoria, jolloin ei tarvita asetusnäkymää tai valikkoja, mutta käyttäjällä on kuitenkin pääsy kaikkiin sensoreista saataviin arvoihin. Myös luisto- ja suoritusindeksit voidaan nähdä omana ryhmänään. Koska viisarikomponentteja voidaan lisätä tarpeen mukaan, ei tämän ryhmän sisällä ole tarvetta komponenttien vaihtamiselle. Ryhmät toteutettiin jakamalla näyttö osiin ja luomalla metodit joilla valitaan ryhmä painalluksen sijainnin perusteella.

Komponentteja vastaavan ryhmän sensorien arvojen käsittelyä ja tallennusta varten luotiin kutakin ryhmää vastaavat luokat: BarometerViewData, DistanceViewData, SkiingEfficiencyViewData, SlideIndexViewData sekä TempViewData (kuva 17). Luokat pitävät muun muassa huolen arvojen muuntamisesta näytön vaatimaan muotoon sekä arvojen näyttämisestä reaaliaikaisena tai keskiarvotilassa. Komponentteja vastaa ryhmäänsä vastaavassa luokassa määritetyt kokonaislukuarvot, joiden avulla luokat myös pitävät kirjaa aktiivisista komponenteistaan. Komponentti on ryhmänsä aktiivinen komponentti, kun

se on valittuna ryhmänsä käyttöliittymässä näkyväksi näyttökomponentiksi. Komponentit saavat arvonsa ainoastaan luokkien kautta.



KUVA 17. Sensoridatan esitykseen omistetut luokat

Komponenttien valinta, vaihtaminen ja päivitys

GraphFragmentin sensoria voidaan vaihtaa valitsemalla GaugeView'ssa ryhmien aktiivisten komponenttien väliltä. Jotta GraphFragment saa tiedon valinnasta toteutettiin SelectedSensor-rajapinta joka sisältää ryhmiä vastaavat metodit jotka saavat parametrinaan valitun komponentin lukuarvon. Komponentin valinta toteutettiin GaugeView'n View-luokan painalluksen tunnistavalla onTouch(View v, MotionEvent event)-metodilla, jonka aktivoituessa testataan painalluksen sijainti. Sijainnin mukaan saadaan valittu ryhmä ja siitä komponentti, jolloin suoritetaan vastaava rajapinnan callback parametrinaan valitun komponentin lukuarvo. Rajapinnalla on toteutus GraphFragmentissa. jossa päivitetään näyttökomponentit parametrina saadun komponentin mukaisesti.

GaugeView'n onTouch()-metodissa toteutettiin myös aktiivisen komponentin vaihto valitun ryhmän sisällä. Painalluksen tapahtuessa testataan sen kesto, ja pitkän painalluksen tapauksessa vaihdetaan aktiivista komponenttia valitun ryhmän luokassa tätä varten toteutetun metodin avulla. Koska käyttöliittymän komponenttien tila ja arvot tulevat ainoastaan ryhmiä vastaavista luokista, ei muualle tarvitse tehdä muutoksia.

Komponenttien arvojen ja värien päivitys toteutettiin broadcast receiverin avulla. Kun BLEService-palvelu lähettää broadcastin GaugeFragmentin broadcast receiveriin, talletetaan uudet arvot näyttökomponenttiryhmiä vastaaviin luokkiin. Tällöin broadcast receiverissä suoritetaan myös GaugeView'n invalidate()-metodi, jolloin GaugeView päivittää komponenttinsa luokista. GraphFragment puolestaan päivittää komponenttinsa Handlerin onPostDelayed()-metodissa luokkiin tallennetuilla nyt päivittyneillä arvoilla ja niiden mukaisilla väreillä SKIIOTin nopeimman lähetysnopeuden mukaisesti. Tämä siksi, että kiinteä nopeus näyttää reaaliaikaisen kuvaajan tasaisemmin kuin broadcast receiveristä päivitettynä.

Komponenttien värien toteutus

Käyttöliittymäkomponenttien värit toteutettiin muuntamalla komponenttikohtaiset sensoriarvot 0 – 255 välillä oleviksi Red-, Green- ja Blue -kokonaislukuarvoiksi valitun skaalan mukaisesti, joista komponentin väri sitten saadaan Color.argb(alpha, Red,Green,Blue)-metodilla. Muunnoksessa voidaan käyttää vakioita halutun skaalan aikaansaamiseksi. Muunnos suoritetaan Uilts-luokkaan toteutetussa getCurrentColor()-metodissa, jonka palauttama arvo talletetaan kyseisestä komponenttiryhmästä vastaavaan luokkaan.

Ikonien sijoittelu käyttöliittymässä

GPS:n ja akun tilalle, asetusvalikolle sekä kalenteri-ikonille luotiin IconFragment-luokka. Fragmentin layout määritettiin GaugeFragmentin kokoiseksi ja läpinäkyväksi ja ikonit sijoitettiin layoutin nurkkiin.

IconFragmentin asetusvalikko toteutettiin Androidin ContextMenuna. ContextMenu on käyttäjän painalluksesta avautuva kelluva valikko (51). Asetusvalikko sisältää muun muassa tiedot yhdistetystä SKIIOT-laitteesta sekä valinnan, jonka kautta sovellus voidaan keskeyttää, jolloin tallennustoiminto keskeytyy eikä tietokantaan tallennusta tapahdu.

Tallennuksen- ja laiteyhteyden hallinnoinnin toteutus käyttöliittymässä

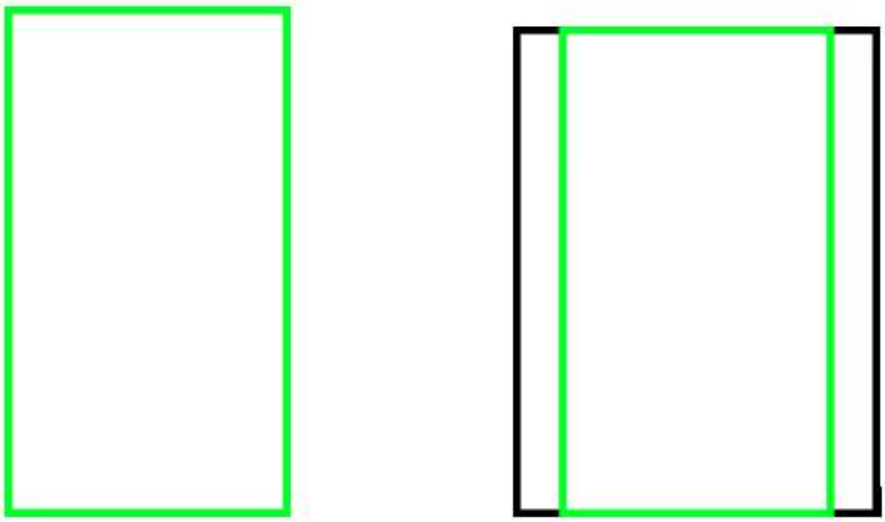
Laiteyhteyden ja tallennuksen hallinnointiin omistetun ConnectFragment-luokkaan sijoitettiin Button-tyyppinen kontrollipainike huolehtimaan käyttäjärajapinnasta ja toimimaan käyttöliittymässä visuaalisena indikaattorina tallennuksen ja yhteyden tilasta. Painikkeen teksti ilmaisee sovelluksen kulloisenkin tilan: ”CONNECT”, kun laite ei ole yhdistetty,

”START”, kun yhteys on olemassa ja ollaan valmiina tallennukseen sekä ”DISCONNECTED”, kun olemassaoleva yhteys on katkaistu tai menetetty. Tallennuksen aikana painike toimii sekuntikellon näyttönä, ja ”PAUSE”-tilassa se näyttää pysäytyksen ajankohdan välkkyvänä, jolloin käyttäjä tietää tilan muuttuneen keskiarvotilaksi. Puhelimen ja kellon layouteissa painike sijoittuu näytön alalaitaan.

Sekuntikellon näyttö toteutettiin omana palvelunaan, joka aloitetaan ConnectFragment-luokassa tallennuksen alkaessa. Ajanottoon liittyvät metodit on määritetty Stopwatch-luokassa, ja StopwatchService-luokka huolehtii ajanoton jatkumisesta myös sovelluksen ollessa taustalla. kontrollipainikkeeseen päivittyy tallennuksen senhetkinen kesto sekunnin sadasosan tarkkuudella Tämä toteutettiin bindService()-toiminnolla, joka päivittää Stopwatch-luokasta saatavan ajan kontrollipainikkeeseen.

Puhelimen käyttöliittymän skaalautuvuus

Puhelimen käyttöliittymä on pystymuodossa ja sen layout noudattaa 16:9 kuvasuhdetta. Laitteille, joilla on eri kuvasuhde, käyttöliittymä skaalautuu säilyttäen layoutin alkuperäisen kuvasuhteen (kuva 18).



KUVA 18. Eri näyttöjen kuvasuhteet: 16:9 ja 4:3

Skaalautuvuus toteutettiin siten, että kukin käyttöliittymän fragmentti huolehtii omien komponenttiensa skaalautuvuudesta. Fragmenttien layoutit puolestaan skaalautuvat päätivitettiin toteutetuissa callbackeissä, joita kutsutaan kunkin fragmentin onViewCreated()-

metodista, jolloin taataan se, että fragmentin layout on valmis. Tällöin LayoutParams.setLayoutParams()-metodin avulla skaalataan fragmenttien alla olevat layoutit kunkin näytön kokoon sopiviksi.

Kellossa näytön malli voi vaihdella suorakaiteen ja ympyränmuotoisen välillä. Sovellus tukee tällä hetkellä ainoastaan suorakaiteen mukaisia näyttöjä.

5.3 Bluetooth Low Enery -yhteyden muodostaminen SKIIOT-laitteen ja sovelluksen välille

Koska on tärkeää, että tallennus jatkuu myös sovelluksen ollessa taustalla, BLE-yhteys ja tallennustoiminnot toteutettiin palvelun avulla, jolloin yhteys ja tallennus ovat riippumattomia sovelluksen aktiviteetin elinkaaresta. Tätä varten luotiin Service-luokasta peritty BLEService-luokka. BLEService-luokka, kuten myös muu BLE-yhteyteen liittyvä toiminnallisuus toteutettiin alkuun pitkälti android-BluettoothLeGatt-esimerkkiprojektin pohjalta (52).

5.3.1 Alkutoimenpiteet

Jotta sovellus voisi saada käyttöönsä Bluetooth-ominaisuudet kuten yhteyden pyytäminen, yhteyden hyväksyminen ja tiedon lähettäminen, täytyi Android-projektin manifestiin lisätä BLUETOOTH-permissio. Jos sovelluksen halutaan aloittavan laitteiden etsiminen tai manipuloivan Bluetoothin asetuksia, täytyy myös lisätä BLUETOOT_ADMIN-permissio. (53.)

rmissioiden tarkistus tapahtuu pääaktiviteetin onCreate()-metodissa. Permissiot kysytään ajoaikana yhden kerran, jonka jälkeen ne talletetaan preferensseihin. Mikäli permissiot ovat kunnossa, voidaan tarkistaa laitteen Bluetooth-tuki. Tämä tapahtuu luomalla BluetoothAdapter-olio joka saadaan getSystemService()-metodin palauttamasta BluetoothManagerin instanssista. Koko järjestelmässä on ainoastaan yksi BluetoothAdapter, ja sovellus voi vuorovaikuttaa sen kanssa tämän BluetoothAdpaterista luodun olion kanssa. BluetoothAdapteria tarvitaan kaikkeen Bluetooth-toimintaan Androidissa ja se edustaa laitteen omaa BluetoothAdapteria. (54.)

5.3.2 Yhteyden muodostaminen

Kun SKIIOT-laite laitetaan päälle, aloittaa se itsensä mainostamisen. Mainostavien laitteiden havaitsemiseksi toteutettiin pääaktiviteetissa oleva mukautettu startLeScan()-metodi, jossa suoritetaan loopissa ennalta määrätyn ajan BluetoothAdapterin startLeScan()-metodia joka skannaa lähistöllä olevia mainostavia laitteita. Metodi on osa Androidin BLE APIa ja saa parametrinaan BluetoothAdapter.LeScanCallBackin, jolloin löydetyt laitteet saadaan kyseisestä callbackista BluetoothDevice-olioina jotka sisältävät löydetyn laitteen nimen ja osoitteen. Tällöin SKIIOT-laitteet saadaan suodattamalla nimen perusteella ja ne lisätään LeDeviceListAdapteriin. Jotta käyttäjä voi valita halutun laitteen toteutettiin ListView, johon päivittyvät adapteriin lisätyt laitteet ja josta käyttäjä valitsee painalluksella haluamansa laitteen. Valitusta laitteesta talletetaan laitteen nimi ja osoite.

Sovelluksen BLE-yhteyden muodostaminen toteutettiin tavalla, jossa aloitetaan uusi BLE-Service-palvelu joka kerta haluttaessa yhteys SKIIOT-laitteeseen. Käyttäjän valittua laite listasta aloitetaan uusi BLEService-palvelu startService()-toiminnolla. Tällöin BLEServicen onStartCommand()-metodissa suoritetaan BlluetoothManager- ja BluetoothAdapter-olioiden luominen. Pääaktiviteetissa suoritetaan myös bindService()-toiminto, jolloin toteutetaan ServiceConnection-rajapinta, jonka avulla voidaan kommunikoida palvelun kanssa. Mikäli palvelun ja aktiviteetin yhdistäminen onnistuu aktivoituu ServiceConnection-rajapinnan onServiceDiscovered()-callback.

Laiteyhteyden muodostaminen toteutettiin onServiceDiscovered()-callbackissä, jolloin kutsutaan BLEServicen connect()-metodia. Metodissa yritetään BluetoothAdapterista luoda ensin uuden osoitteen mukainen BluetoothDevice-olio getDevice(address)-metodilla ja tämän onnistuttua yhteys vihdoin muodostetaan osoitteen mukaiseen laitteeseen BluetoothDevicen connectGatt()-metodilla, joka saa parametrinaan BluetoothGattCall-Backin.

Jotta nyt päästään käsiksi yhdistetystä laitteesta saapuvaan dataan, täytyy muodostaa yhteys BluetoothGatt-palveluun, joka määrittää tarvittavan protokollan. Tähän tarkoitukseen toteutettiin BLEServicessä Androidin BluetoothGatt-ohjelmointirajapinnan mukainen BluetoothGattCallBack, jonka connectGatt() saa parametrinaan. Tämä toteutus sisältää metodit onConnectionChange, onServiceDiscovered, onCharacteristicRead sekä onCharacteristicChanged. (19.)

OnConnectionChangea kutsutaan, kun yhteydessä on tapahtunut muutoksia. Metodi saa parametrinaan newStaten, joka on Androidin Bluetooth profiilin julkiseen APIin kuuluvaan BluetoothProfileen kuuluva kokonaislukuarvoinen vakio, joka kertoo yhteyden tilasta. Mikäli newStaten arvo on BluetoothProfile.STATE_DISCONNECTED, on yhteys katkennut tai katkaistu, jolloin sovelluksessa tehdään tarvittavat toimenpiteet. Mikäli newStaten arvo on BluetoothProfile.STATE_CONNECTED, on yhteys muodostettu laitteeseen ja voidaan suorittaa discoverService()-metodi. Tämän onnistuessa metodin onServiceDiscovered()-callbackissä luodaan BluetoothGattin getService()-metodilla BluetoothGattService ensin akulle, jolloin metodi saa parametrinaan SkiiotBoard-luokassa määritellyn UUID:n akulle, ja vastaavasti BluetoothGattService datalle, jolloin metodi saa parametrinaan SkiiotAttributes-luokassa määritellyn UUID:n sensoridatalle. (19.)

Seuraavaksi luodaan BluetoothGattCharacteristic-olio akulle, joka annetaan parametrina readCharacteristic()-metodille. readCharacteristic() on asynkroninen operaatio, joka lukee pyydetyn characteristicin laitteelta. Sen callback-metodi on onCharactersiticRead(), joka saa kutsuvan characteristicin parametrinaan (19). Metodissa asetetaan BluetoothGattin setCharacteristicNotification() saadulle akun characteristicille. Metodissa asetetaan myös pienen viiveen jälkeen setCharacteristicNotification() datan characteristicille, jolloin sen onCharacteristicChagned(BluetoothGatt gatt, BluetoothGattCharacteristic characteristic)-callbackissä saadaan uuden datan sisältävä characteristic. Uuden datan saapuessa metodissa myös kutsutaan broadcastUpdate()-apumetodia, jossa sendBroadcast()-metodilla lähetetään broadcast sitä kuuntelevalle GaugeFragmentissä olevalle broadcast receiverille, jossa käyttöliittymä päivitetään vastaamaan saatuja arvoja. Myös pääaktiviteetissa on broadcast receiver, joka kuuntelee BLE-yhteyden tilan muutoksia ja jonka kautta käyttöliittymä päivitetään vastaamaan näitä muutoksia. Eri receivereille menevät broadcastit erotetaan SkiiotBoard-luokassa määritettyjen String-tyyppisten action-arvojen avulla, jotka broadcast lähettää Intent-olion välityksellä.

5.4 Sensoridatan käsittely ja tallennus

5.4.1 Datan muuntaminen sensorikohtaisiksi lukuarvoiksi

Koska SKIIOT-laitteen datan käsittely on yhteistä sekä kellolle että puhelimelle, oli se mielekästä toteuttaa omana erillisenä kirjastomoduulinaan jonka puhelin- ja kelloprojekti jakavat. SKIIOT-laitteen uudempi malli on nimenomaisesti lasketteluun suunnattu ja siinä

datan lähetysnopeus on 100 Hz. Tämä asetti uudet vaatimukset datan käsittelylle ja talletukselle. Basic4Android-lähdekoodin mukaisesti saapuva characteristic muutetaan ensin merkkijonoksi. Merkkijonosta saadaan sitten alijonoina eri sensorien data, jotka muutetaan metodien avulla lukuarvoiksi. Tämä osoittautui liian hitaaksi ja saapuvan characteristicin data täytyi käsitellä tavuina.

Tämän seurauksena kirjastomoduuli ja sensorikohtaiset datan tallennukseen ja käsittelyyn omistetut luokat jäivät pois myöhemmästä kehityksestä.

Kirjastomoduuli

Kirjastomoduulissa toteutettiin muun muassa metodit keskiarvon laskemiseen. Moduuliin sijoitettiin myös jokaista SKIIOT-laitteen sensoria vastaavat luokat, jotka sisältävät kuhunkin sensoriin liittyvän datan käsittelyyn tarvittavat tiedot ja metodit. Koska jokaisessa luokassa oli kaikki tarvittava tieto, oli niissä paljon päällekkäistä koodia matemaattisten metodien osalta. Tämä ratkaisu tehtiin virheenkorjauksen ja testauksen helpottamiseksi. Matemaattiset metodit saatiin Basic4Android-sovelluksen lähdekoodista.

SKIIOT-laitteesta saatavan sensoridatan lisäksi myös mobiililaitteesta saatavat matka, nopeus ja kellon syke käsiteltiin kirjastossa. Tämä siksi, että datan käsittely ja saanti oli mahdollisimman yhdenmukaista sovelluksen sisällä.

Luokkiin päästiin käsiksi sovelluksen alussa luotavilla listoilla, jotka sisälsivät seuraavat ilmentymät luokista:

- device_data_list_f1 20 Hz:n nopeudella SKIIOT-laitteesta tulevalle datalle
- device_data_list_f2 1 Hz:n nopeudella SKIIOT-laitteesta tulevalle datalle
- device_data_list_distance mobiililaitteen GPS-datalle
- device_data_list_heartrate kellosta saatavalle sykkeelle.

Nopeampi datan käsittely

Kun kirjastomoduulin datan prosessointi osoittautui liian hitaaksi, siirtyi datan käsittely päämoduuliin. Kirjaston sisältämät sensoriluokat korvautuivat staattisilla muuttujilla, ja datan käsittelyyn tarvittavat metodit sijoitettiin SkiiotBoard-luokkaan, joka myös sisältää characteristicien sisältämän sensoridatan tavupaikat ja BLE-yhteyden vaatimat UUID:t. SkiiotBoardin-metodit saatiin muokkaamalla annetusta mallikoodista. Olennaisimpana muutoksena datan käsittelyssä on, että laitteesta saapuvaa characteristicia ei muunneta merkkijonoksi datan myöhempää käsittelyä varten, vaan data käsitellään suoraan tavuina. Samaisesta mallikoodista saatiin myös nopeammat metodit tallennustoiminnallisuuteen.

Juoksevan keskiarvon laskeminen

SensorDataUtils-luokassa toteutettiin jokaista sensorikohtaista muuttujaa kohti oma ilmentymänsä RunningStat-luokasta. RunnigStat-luokka sisältää metodit juoksevan keskiarvon laskemiseksi ja kunakin hetkenä saatu lukuarvoksi muunnettu sensoriarvo annetaan parametrina kyseistä sensoria vastaavan RunnignStat-olion push(double value)-metodille, joka palauttaa siihenastisten arvojen keskiarvon. Kunkin sensorin keskiarvo talletetaan omaan muuttujaansa, josta se on sovelluksen muiden osien käytössä.

Päivitykset uusille laiteversioille ja spesifikaatioille

BLE-yhteyteen liittyvät määritykset sekä metodit datan käsittelylle sijoitettiin samaan luokkaan, johon mahdollisten päivitysten tuomat muutokset saadaan rajattua ylläpidon helpottamiseksi.

5.4.2 Tallennus

Tallennustoiminnolla käyttäjä tallentaa harjoittelukerran tiedot myöhempää tarkastelua ja analysointia varten. Koska laite lähettää kahdella eri lähetysnopeudella dataa, tiedot tallentuvat kahdeksi eri CSV-tiedostoksi lähetysnopeuden mukaan. Kun tallennus lopetetaan, tallentuvat tiedostot tietokantaan, josta niitä pääsee myöhemmin tarkastelemaan. Tallennustoiminto voidaan aloittaa, kun yhteys on muodostettu SKIIOT-laitteen ja sovelluksen välille.

CSV-tiedostojen luonti ja tallennus tietokantaan

Tallennustoiminto toteutettiin luomalla kaksi CSV-tiedostoa, joihin kirjoitetaan BLEServiceen saapuvasta datasta lasketut sensoriarvot pilkulla erotettuina toisistaan ja jotka tallennuksen loputtua tallennetaan tietokantaan. CSV on yksinkertainen tiedostomuoto, johon tallennetaan ja siirretään taulukkomuotoista dataa (55).

Aloitettaessa tallennus luodaan lähetysnopeuksien mukaiset tiedostot SkiiotLoggerF1 20 Hz:n nopeudella ja SkiiotLoggerF2 1 Hz:n nopeudella saapuvalle datalle getExternalStoragePublicDirectory()-metodin palauttamaan hakemistoon. Android-laitteilla on kaksi tiedoston tallennukselle varattua aluetta: ”internal” eli sisäinen ja ”external” eli ulkoinen. Sovelluksessa tiedostot tallennetaan testaussyistä ulkoiseen tallennustilaan, jolloin manifestiin täytyi lisätä WRITE_EXTERNAL_STORAGE-permissio.

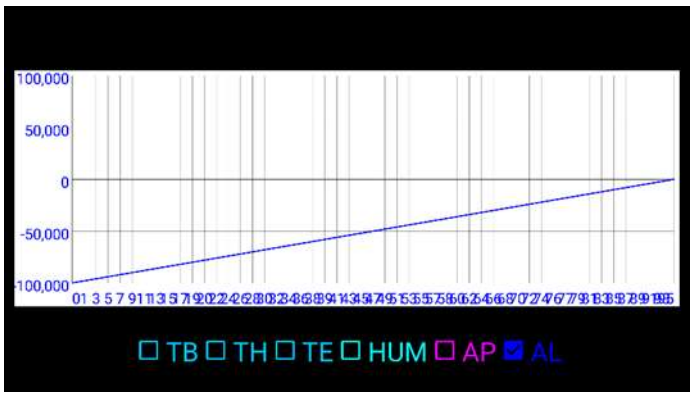
CSV-tiedostoihin kirjoittaminen toteutettiin asynkronisena taustaprosessina, johon käytettiin Androidin AsyncTask-luokkaa, joka suorittaa halutun toiminnon omassa erillisessä säikeessään (56). AsyncTask suoritetaan BLEServicen BroadcastUpdate()-metodissa aina uuden characteristicin saapuessa, jolloin datasta lasketuista sensorikohtaisista lukuarvoista muodostetaan ensin CSV-tyyppinen merkkijono, joka sisältää sensorien arvot pilkulla erotettuina. Tämä merkkijono sitten kirjoitetaan BufferedWriter()-instanssin avulla edellä luotuun characteristicin nopeuden mukaiseen tiedostoon.

Kun tallennus lopetetaan, suoritetaan tiedostojen osoitteiden ja päivämäärän kirjoittaminen tietokantaan. Tietokannan luonti ja hallinnointi toteutettiin SQLite-tietokannan avulla. SQLite on kevyt relaatiotietokanta ja kuulu Android-käyttöjärjestelmään (57). Tietokantaan kirjoittamista varten toteutettiin SkiiotFileRecord-luokka, joka pitää sisällään metodit tietokantaan luettavien tietojen tallentamiseksi. Joka tallennuskerta luodaan luokasta instanssi, johon talletetaan halutut tiedot. DatabaseOperations-luokka sisältää perustietokantaoperaatiot ja sen tietokantaan kirjoittamisesta vastaavat metodit saavat kyseisen olion parametrinaan, jolloin tiedoston osoitteet ja päivämäärä voidaan lukea tietokantaan.

Tietokannasta hakeminen toteutettiin luomalla DatabaseOperations-luokkaan metodi, jossa hakutoiminto tapahtuu annetun päivämäärän mukaan.

Tallennetun tiedon esittäminen

Tietokannassa oleviin tiedostoihin päästään käsiksi kalenterinäkymästä, johon on merkitty kulloisenkin päivän harjoittelukerrat. Kalenteri toteutettiin avoimen lähdekoodin Caldroid-kirjastolla, jota voidaan käyttää kustomoidun kalenterin toteutukseen (58). Käyttäjän valitessa päivän aktivoituu pääaktiviteetissa toteutettu Caldroidin kuuntelijarajapinnan onSelectDate(Date date, View view)-metodi, jossa käynnistetään kyseisen päivämäärän mukaisen ListViiew'na toteutetun tiedostolistauksen sisältävä aktiviteetti. Tästä käyttäjä valitsee haluamansa tiedoston, joka puolestaan käynnistää uuden aktiviteetin. Tämä aktiviteetti sisältää GraphView-näkymän, jolla sensoriarvojen graafinen esitys on toteutettu. Arvot näkyvät kuvaajina, jotka valitaan näytön alalaidassa olevista valintaruuduista. (Kuva 19)



KUVA 19. Sensoridatan graafinen esitys

6 TULOKSET

Sovelluksen avulla käyttäjä voi seurata hiihtosuoritustaan sekä tallentaa harjoittelukerran aikana SKIIOT-laitteen sensorien keräämän tiedon. Kun käyttäjä aktivoi laiteyhteyden muodostamiseen liittyvät toiminnot, näyttää sovellus lähistöllä olevat SKIIOT-laitteet, joista käyttäjä myös voi valita haluamansa. Tällöin sovellus ottaa yhteyden valittuun laitteeseen, jolloin käyttäjän on mahdollista aloittaa tallennustoiminto, jonka hallinnointiin sovellus tarjoaa käyttöliittymän. Tallennettujen tiedostojen myöhempi tarkastelu onnistuu tietokannan ja siihen kalenterin avulla toteutetun käyttöliittymän avulla. Valitun tiedoston tiedot sitten avautuvat käyttäjälle grafiikkana, jossa eri sensorien arvojen esitys on toteutettu käyttäjän valitsemina kuvaajina kaaviossa

Asetusvalikko sisältää muun muassa tiedot sovelluksen versiosta, yhdistetyn laitteen nimen ja version sekä mahdollisuuden talletustoiminnon keskeytykseen.

Käyttöliittymä mahdollistaa reaaliaikaisen sensoriarvojen- sekä juoksevan keskiarvon seurannan, jotka on havainnollistettu graafisten komponenttien avulla. Käyttöliittymä sisältää versiot sekä puhelin- ja tabletti -alustalle että kellolle (kuva 20).

KUVA 20. SKIIOT-sovellus puhelimelle ja kellolle

7 POHDINTA

Työn tavoitteena oli käyttöliittymän sekä Bluetooth Low Energy -yhteyteen liittyvien teknisten ratkaisujen suunnittelu ja toteutus Exiopsin SKIIOT-laitteelle. Sovelluksen Bluetooth Low Energy -toiminnallisuus sisältää vaaditut ominaisuudet ja käyttöliittymä on tarkoituksenmukainen laitteen antaman datan esitykselle ja hallinnoimiselle. Sovellus sisältää myös matkan ja nopeuden mittaamisen, tallentamisen ja esityksen toteutuksen, kuten myös kellossa sykkeen. Näiden ominaisuuksien käsittely jäi kuitenkin tämän työn aihepiirin käsittelyn ulkopuolelle.

Kellon kehitys jäi ajanpuutteen vuoksi keskeneräiseksi ja se sisältää tällä hetkellä vain olennaisimmat käyttöliittymätoiminnot. Koska kellon sovellus toteutettu Android Wear 1.0 -alustalle, olisi sen päivitys Android Wear 2.0 -alustalle tarpeellinen. Tarpeellista olisi myös päivittää käyttöliittymä pyöreää näyttöä varten.

Projekti oli osin haasteellinen käyttöliittymäratkaisujen kannalta, jolloin kellolle suunniteltu käyttöliittymä tuli sovittaa puhelimien ja ennen kaikkea tablettien erikokoisille näytöille. Tabletille suunniteltua käyttöliittymää voisi jatkokehittää suuremman näytön tarjotessa erilaisia mahdollisuuksia yksityiskohtaisempaan tiedon esittämiseen.

Ongelmia tuotti myös määritykset datan sisältävän characteristicin sisältämien sensorikohtaisten arvojen paikasta. Määritykset oli annettu characteristicista muodostetulle merkkijonolle ja tämän muuttaminen suoraan charcteristicin tavujonon tavujen paikoiksi oli osin haasteellista.

Ylläpidon helpottamiseksi sovelluksessa on pyritty keskittämään muutoksen vaatimat metodit ja määritykset yhteen luokkaan, jotta SKIIOT-laitteiden uusien versioiden ja spesifikaatioiden päivitysten tuomat muutokset sovellukseen olisi kohtalaisen helppo toteuttaa.

LÄHTEET

1. SKIIOT. Saatavissa: http://skiiot.com/. Hakupäivä 18.04.2018.

2. Polar M600. Polar. Saatavissa: https://www.polar.com/sites/default/files/product2/600x600/polar-m600-front-black-600x600.png. Hakupäivä 7.5.2018.

3. Internet of Things. 2018. Wikipedia. Saatavissa: https://en.wikipedia.org/wiki/Internet_of_things. Hakupäivä 30.03.2018.

4. Rockershousen, Nathan 2016. Internet of Things and Bluetooth. Grid Connect. Saatavissa: https://gridconnect.com/blog/general/the-internet-of-things-and-bluetooth/. Hakupäivä 30.3.2018.

5. Radio Versions. 2018. Bluetooth. Saatavissa: https://www.bluetooth.com/bluetooth-technology/radio-versions. Hakupäivä 25.3.2018.

6. How GAP and GATT work. 2018. PunchThrough. Saatavissa: https://punchthrough.com/bean/docs/guides/everything-else/how-gap-and-gatt-work/. Hakupäivä 8.3.2018.

7. GAP. 2015. Adafruit. Saatavissa: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap. Hakupäivä 12.3.2018.

8. Generic Access Profile (GAP). 2016. BLE-Starck User's Guide for Bluetooth 4.2. Texas Instruments Saatavissa: http://dev.ti.com/tirex/content/simplelink_cc2640r2_sdk_1_40_00_45/docs/blestack/ble_user_guide/html/ble-stack-3.x/gap.html#gap. Hakupäivä 1.3.2018.

9. GAP State Diagram. Texas Instruments. Saatavissa: http://dev.ti.com/tirex/content/simplelink_cc2640r2_sdk_1_40_00_45/docs/blestack/ble_user_guide/html/_images/image72.jpeg. Hapupäivä 7.3.2018.

10. Bluetooth Low Energy. 2018. Android Developers. Saatavissa: https://developer.android.com/guide/topics/connectivity/bluetooth-le.html. Hakupäivä 27.6.2017.

11. GATT. 2015. Adafruit. Saatavissa: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt. Hakupäivä 1.4.2018.

12. GATT diagram. Adafruit. Saatavissa: https://cdn-learn.adafruit.com/assets/assets/000/013/828/medium800/microcontrollers_GattStructure.png?1390836057. Hakupäivä 8.3.2018.

13. Generic Attribute Profile Service (GATT Service). 2016. BLE-Starck User's Guide for Bluetooth 4.2. Texas Instruments. Saatavissa: http://dev.ti.com/tirex/content/simplelink_cc2640r2_sdk_1_40_00_45/docs/blestack/ble_user_guide/html/ble-stack-3.x/gatt.html#generic-attribute-profile-service-gatt-service. Hakupäivä 8.3.2018.

14. GATT client server overview. BLE-Starck User's Guide for Bluetooth 4.2. Texas Instruments. Saatavissa: http://dev.ti.com/tirex/content/simplelink_cc2640r2_sdk_1_40_00_45/docs/blestack/ble_user_guide/html/_images/gatt_client_server.png. Hakupäivä 8.3.2018.

15. Karch, Mariah 2017. What Is Google Android? Lifewire. Saatavissa: https://www.lifewire.com/what-is-google-android-1616887. Hakupäivä 8.3.2018.

16. Platform Architecture. 2018. Android Developers. Saatavissa: https://developer.android.com/guide/platform/. Hakupäivä 27.5.2018.

17. The Android software stack. Android Developers. Saatavissa: https://developer.android.com/guide/platform/images/android-stack_2x.png. Hakupäivä 8.3.2018.

18. Gore Amar, 2018. Learn Android Architecture. SmartBit Tutorials. Saatavissa: http://www.smartbittutorials.com/programs/learn-android-architecture/. Hakupäivä 24.03.2018.

19. BluetoothGatt. 2018. Android Developers. Saatavissa: https://developer.android.com/reference/android/bluetooth/BluetoothGatt.html. Hakupäivä 18.4.2018.

20. Android Wear. 2018. Wikipedia. Saatavissa: https://en.wikipedia.org/wiki/Android_Wear. Hakupäivä 8.3.2018.

21. Lehtiniitty, Markus 2017. Näin Android-kellot nyt muuttuvat – Android Wear 2.0 julkaistu. Mobiili.fi. Saatavissa: https://mobiili.fi/2017/02/08/android-wear-20/. Hakupäivä 23.05.2017.

22. Create and run a wearable app. 2018. Android Developers. Saatavissa: https://developer.android.com/training/wearables/apps/creating. Hakupäivä 22.05.2018.

23. Meet Android Studio. 2018. Android Developers. Saatavissa: https://developer.android.com/studio/intro/index.html. Hakupäivä 8.3.2018.

24. Application Fundamentals. 2018. Android Developers. Saatavissa: https://developer.android.com/guide/components/fundamentals.html. Hakupäivä 1.4.2018.

25. Android manifest. 2018. Android Developers. Saatavissa: https://developer.android.com/guide/topics/manifest/manifest-intro.html. Hakupäivä 18.5.2018.

26. Android application package. 2018. Android Developers. Saatavissa: https://en.wikipedia.org/wiki/Android_application_package. Hakupäivä 9.3.2018.

27. Activity.2018. Android Developers. Saatavissa: https://developer.android.com/reference/android/app/Activity.html. Hakupäivä 9.3.2018.

28. WearableActivity. 2018. Android Developers. Saatavissa: https://developer.android.com/reference/android/support/wearable/activity/WearableActivity.html. Hakupäivä 18.4.2018.

29. Activity lifecycle. LanDen Labs Saatavissa: http://landenlabs.com/android/info/activity-life-cycle/activity_lifecycle.png. Hakupäivä 8.3.2018.

30. Understand the Activity Lifecycle. 2018. Android Developers. Saatavissa: https://developer.android.com/guide/components/activities/activity-lifecycle.html#alc. Hakupäivä 18.4.2018.

31. Services. 2018. Android Developers. Saatavissa: https://developer.android.com/guide/components/services.html. Hakupäivä 9.3.2018.

32. Fragment. 2018. Android Developers. Saatavissa: https://developer.android.com/reference/android/app/Fragment.html. Hakupäivä 9.3.2018.

33. Broadcasts. 2018. Android Developers. Saatavissa: https://developer.android.com/guide/components/broadcasts.html. Hakupäivä 12.3.2018.

34. Broadcast receiver. Vogella. Saatavissa: http://www.vogella.com/tutorials/AndroidServices/img/xservice_receiver10.png.pagespeed.ic.rR3qMM5rjQ.webp. Hakupäivä 8.3.2018.

35. BroadcastReceiver. 2018. Android Developers. Saatavissa: https://developer.android.com/reference/android/content/BroadcastReceiver. Hakupäivä 11.5.2018.

36. View. 2018. Android Developers. Saatavissa: https://developer.android.com/reference/android/view/View. Hakupäivä 24.5.2018.

37. Layout. 2018. Android Developers. Saatavissa: https://developer.android.com/guide/topics/ui/declaring-layout. Hakupäivä 24.5.2018.

38. Sincki, Adam 2017. Android SDK tutorial for beginners. Android Authority. Saatavissa: https://www.androidauthority.com/android-sdk-tutorial-beginners-634376/. Hakupäivä 25.3.2018.

39. Android Debug Bridge (adb). 2018. Android Developers. Saatavissa: https://developer.android.com/studio/command-line/adb.html. Hakupäivä 25.3.2018.

40. Create a project. 2018. Android Developers. Saatavissa: https://developer.android.com/studio/projects/create-project.html. Hakupäivä 22.7.2017.

41. Build Script Basics. Gradle Build Tool. Saatavissa: https://docs.gradle.org/current/userguide/tutorial_using_tasks.html. Hakupäivä 11.5.2018.

42. Configure Your Build. 2018. Android Developers. Saatavissa: https://developer.android.com/studio/build/index.html. Hakupäivä 17.3.2018.

43. Projects Overveiw. 2018. Android Developers. Saatavissa: https://developer.android.com/studio/projects/. Hakupäivä 17.3.2018.

44. Package and distribute Wear apps. 2018. Android Developers. Saatavissa: https://developer.android.com/training/wearables/apps/packaging. Hakupäivä 7.5.2018.

45. Polar M600 for developers. Saatavissa: https://wiki.metropolia.fi/download/attachments/144081572/Polar%20M600%20for%20developers.pdf?version=1&modificationDate=1486367291000&api=v2. Hakupäivä 25.3.2018.

46. GridViewPager. 2018. Android Developers. Saatavissa: https://developer.android.com/reference/android/support/wearable/view/GridViewPager. Hakupäivä 8.5.2018.

47. FrameLayout. 2018. Android Developers. Saatavissa: https://developer.android.com/reference/android/widget/FrameLayout. Hakupäivä 9.3.2018.

48. LinearLayout. 2018. Android Developers. Saatavissa: https://developer.android.com/reference/android/widget/LinearLayout. Hakupäivä 8.5.2018.

49. BoxInsetLayout. 2018. Android Developers. Saatavissa: https://developer.android.com/reference/android/support/wear/widget/BoxInsetLayout. Hakupäivä 8.5.2018.

50. GraphView. Saatavissa: http://www.android-graphview.org/. Hakupäivä 8.5.2018.

51. Geetha, Sai 2011. Context Menu Android Tutorial. Mobile Zone. DZone. Saatavissa: https://dzone.com/articles/context-menu-android-tutorial. Hakupäivä 8.5.2018.

52. android-BluetoothLeGatt. Saatavissa: https://github.com/googlesamples/android-BluetoothLeGatt. Hakupäivä 7.5.2018.

53. Bluetooth. 2018. Android Developers. Saatavissa: https://developer.android.com/guide/topics/connectivity/bluetooth.html. Hakupäivä 18.4.2018.

54. Bluetooth Low Energy. 2018. Android Developers. Saatavissa: https://developer.android.com/guide/topics/connectivity/bluetooth-le.html#find. Hakupäivä 12.3.2018.

55. CSV-tiedoston tuominen Exceliin. 2017. Excel-ohjeet. Saatavissa: http://excelohjeet.com/csv-tiedoston-tuominen-exceliin/. Hakupäivä 25.4.2018.

56. AsyncTask. 2018. Android Developers. Saatavissa: https://developer.android.com/reference/android/os/AsyncTask.html. Hakupäivä 18.04.2018.

57. Tavada, Rami 2018. Android SQLite Database Tutorial. AndroidHive. Saatavissa: https://www.androidhive.info/2011/11/android-sqlite-database-tutorial/ Hakupäivä 26.5.2018.

58. GitHub Caldroid. Saatavissa: https://github.com/roomorama/Caldroid. Hakupäivä 7.5.2018.