KUNTOSALIHARJOITUSTEN YLLÄPITOSOVELLUS ANDROID-MOBIILILAITTEILLE

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Visamäki, Tietojenkäsittely

Kevät, 2020

Jani Koivula

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittely
Visamäki, Hämeenlinna

Tekijä Jani Koivula **Vuosi** 2020

Työn nimi Kuntosaliharjoitusten ylläpitosovellus Android-mobiililaitteille

Työn ohjaaja Tommi Lahti

TIIVISTELMÄ

Tässä opinnäytetyössä tehdään ensimmäinen versio harjoituslistasovelluksesta, jolla voidaan ylläpitää helposti omia kuntosaliharjoituksia. Sovellus tehdään Android-mobiililaitteille. Sovellukselle tehdään myös käyttöliittymäsuunnitelma, joka havainnollistaa sen lopullista ulkonäköä ja toimintaa. Työssä perehdytään juuri Android-ohjelmoinnin ja tietokantojen lisäksi hyvään käyttöliittymäsuunnitteluun ja erityisesti suunnitteluun Android-laitteille, seuraten Androidin omia ohjenuoria.

Ohjelmointikielenä projektissa toimii Java ja ohjelmointi suoritetaan Android Studio -ohjelmointiympäristössä. Tietokannat sovellukselle tehdään käyttämällä Android Room -tietokantakirjastoa. Käyttöliittymäsuunnitelma tehdään Figma-suunnittelusovelluksella.

Työn tulos on toimiva ensimmäinen versio harjoituslistasovelluksesta, joka on jo tekijällä itsellään käytössä. Käyttöliittymäsuunnitelma lopulliselle sovellukselle saatiin myös tehtyä. Suunnitelma sisältää muutamia ominaisuuksia, joita ei ensimmäiseen versioon saatu toteutettua. Myös suunnitelman lopullista visuaalista ulkonäköä ei sovellukseen vielä ohjelmoitu. Kuitenkin omat opinnäytetyölle asetetut tavoitteet saatiin toteutettua ongelmista ja viivästyksistä huolimatta.

Avainsanat Android, käyttöliittymäsuunnittelu, Figma, Android Room

Sivut 22 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Program in Business Information Technology
Visamäki, Hämeenlinna

Author Jani Koivula **Year** 2020

Subject Gym workout management application for Android mobile devices

Supervisor Tommi Lahti

ABSTRACT

In this thesis, the first version of the exercise list application is made, and it can be used to easily maintain your own exercises. The application is made for Android mobile devices. User interface plan is also made to demonstrate how the final version of the application should look and behave. In addition to the Android programming and databases, this work focuses on how to implement good interface design especially for Android devices.

The programming language for the project is Java and all the programming is done inside the Android Studio programming environment. Databases for the application are created using the Android Room database library. The user interface design is done with the Figma design tool.

As a result, a functional first version of the exercise list application is done. The application is already in use by the author himself. The user interface plan for the final application was also completed. However, the design contains features that could not be implemented in the first version. Also, the final visual appearance of the plan was not yet programmed into the application. Despite the problems and delays, the goals set for the thesis were achieved.

Keywords Android, user interface design, Figma, Android Room

Pages 22 pages

SISÄLLYS

1 JOHDANTO

Tässä toiminnallisessa opinnäytetyössä luodaan ensimmäinen toimiva versio Android-sovelluksesta, jonka tehtävä on auttaa kuntosalilla harjoitusten seuraamista ja ylläpitoa. Valmiille sovellukselle tehdään myös käyttöliittymäsuunnitelma, jotta sitä voidaan jatkokehittää opinnäytetyön jälkeen. Opinnäytetyön idea on peräisin itse työn tekijältä ja erityisesti kuntosaliharrastus ja halu oppia lisää mobiiliohjelmoinnista innoitti tekemään opinnäytetyön tästä aiheesta.

Sovelluksen tarkoitus on helpottaa kuntosalilla harjoitusten ja niihin sisältyvien liikkeiden ylläpitoa ja seurantaa, jotta kuntosalilla voidaan helposti muokata esimerkiksi painojen määrää ensi kertaa varten. Ensin sovellukseen voidaan luoda uusi harjoitus, jonka jälkeen harjoitukseen voidaan lisätä halutut liikkeet. Jokaiselle liikkeelle asetetaan sarjojen, toistojen ja painojen määrä. Kun harjoitukset liikkeineen on tallennettu laitteelle, voidaan niitä helposti muokata tai poistaa.

Työssä perehdytään Android-ohjelmointiin Java-ohjelmointikielellä ja käyttöliittymäsuunnitteluun erityisesti Android-mobiililaitteet mielessä pitäen. Työssä perehdytään myös käyttäjälähtöiseen sovellussuunnitteluun ja Figma-suunnittelusovellukseen. Ohjelmistoympäristönä toimii Android Studio ja tietokannat rakennettiin käyttäen Android Room -kehystä SQLiten yllä.

Käytännön osuus pitää sisällään tietokantojen rakentamisen sekä ensimmäisen sovellusversion käyttöliittymäohjelmoinnin. Työ painottuu pitkälti teknologioiden tutkimiseen, niiden toteuttamiseen ja käyttöliittymän suunnitteluun. Figma-suunnittelusovelluksella luodaan käyttöliittymäsuunnitelma havainnollistaen, miltä lopullinen sovellus tulisi näyttämään. Kuitenkaan tässä projektissa aika ei riitä ohjelmoimaan sovellusta lopullisen käyttöliittymäsuunnitelman mukaiseksi.

Työlle asetetut tutkimuskysymykset ovat, että

- kuinka tehdä toimiva ja käyttäjäystävällinen Android-mobiilisovellus?
- miten Android Roomia hyödynnetään sovellusta tehdessä?

2 MOBIILIKEHITYS

Nykypäivänä ihmiset eivät liiku paljoa ilman kannettavia älylaitteitaan. Mobiilisovelluksia on kaikkialla, olipa se sitten ostoslista kaupassa käydessä tai jokin isompi sovellus esimerkiksi työasioiden hoitamiseksi. Siksi myös mobiilikehitys on suuressa suosiossa, sillä käyttäjiä varmasti hyville sovelluksille riittää. (Ditkowska, n.d)

Jokaisessa laitteessa on oma käyttöjärjestelmä, joka ohjaa laitteen kaikkia toimintoja. Sovellus pitää rakentaa hieman eri tavalla jokaiselle käyttöjärjestelmälle. Suurimmat alustat ovat tällä hetkellä Android ja iOS. Android johtaa älypuhelimien markkinoilla, kun taas iOS johtaa markkinoita tablet-laitteiden puolella. (Ditkowska, n.d)

Mobiililaitteille on olemassa erityyppisiä sovelluksia, natiivi- ja hybridisovellukset, joista kumpikin täytyy tehdä eri tavalla. Kummallakin tyylillä on omat etunsa ja heikkoutensa. (Ditkowska, n.d)

Natiivi applikaatiot on rakennettu käyttäen tiettyjä teknologioita ja ohjelmointikieliä kuten esimerkiksi Java-ohjelmointikieli Androidille ja Swift-ohjelmointikieli iOS:lle. Natiivit applikaatiot ovat suunnattu vain tietyille alustoille, joka parantaa tuotteen toimivuutta juuri niillä alustoilla. Voidaan siis luoda graafisia, näyttäviä sovelluksia, jotka pyörivät laitteilla tehokkaasti. Huonoin puoli hybridiapplikaatioihin verraten on se, että sovellukset täytyy tehdä eri laitteille eri tyyleillä. (Patro, 2018)

Hybridiapplikaatiolla on pääsy kaikkiin natiivin alustan toimintoihin. Ne on rakennettu käyttäen web-teknologioita kuten HTML ja JavaScript. Hybridisovellukset toimivat myös web-näkymässä, ja niitä voidaan rakentaa mille tahansa järjestelmälle, joka on merkittävä etu. Joissain vaativimmissa sovelluksissa hybridiapplikaatiot eivät saavuta samaa tehokkuutta kuin natiivit. (Patro, 2018)

2.1 Android Studio-ohjelmointiympäristö

Vuonna 2018 Google esitteli kirjastojen, työkalujen ja ohjeiden sarjan nimeltä Android Jetpack, tarkoituksena tehdä sovellusten kehityksestä modernia ja luotettavampaa. Android Jetpack koostuu neljästä eri kategoriasta, joita ovat arkkitehtuuri, käyttöliittymä, käyttäytyminen sekä perusta (foundation). Jokainen kategoria sisältää eri komponentteja, esimerkiksi Room-tietokantakirjastot ovat osa arkkitehtuurikomponentteja. Jetpack koostuu myös erilaisista työkaluista ja kirjastoista sekä myös Android Studio on osa Android Jetpackia. Android Studio -ohjelmointiympäristö on kuitenkin alun perin julkaistu jo vuonna 2014 ja tuorein versio on Android Studio 3.5. (Smyth 2019, 1.)

Android Studio on ohjelmointiympäristö siinä missä muutkin. Sinne kirjoitetaan, muokataan ja tallennetaan projekteja. Android Studio tarjoaa pääsyn käyttämään Android ohjelmistokehityspakettia (Software Development Kit, SDK). Android Studiossa voidaan luoda erilaisia virtuaalilaitteita tai liittää fyysinen laite tietokoneeseen, ja käyttää sitä kätevästi ohjelmien testauksessa, kunhan laite asetetaan ensin kehittäjätilaan. (Mullis, 2017)

Google on tehnyt paljon töitä sen eteen, että saisi Android Studiosta niin tehokkaan ja vuorovaikutteisen kuin mahdollista. Virheellisten koodien korjausehdotukset tulevat välittömästi näkyviin ja Android Studio antaa vinkkejä ja ehdotuksia kuhunkin ongelmaan, jotka osoittautuvat silloin tällöin erittäin tarpeellisiksi. Kirjoittaessa Android Studio antaa myös ehdotuksia yleisimmistä komennoista, joka nopeuttaa huomattavasti työn tekoa. (Mullis, 2017)

Android Studio toimii hyvin paljon samoilla periaatteilla kuin monet muut ohjelmistoympäristöt. Kuitenkin siinä on itse Android kehitykseen paljon työkaluja ja valmiita ominaisuuksia, joista on hyötyä monissa tilanteissa. (Mullis, 2017)

2.2 **Java-ohjelmointikieli ja Android**

Sun Micro Systems julkaisi Java-ohjelmointikielen vuonna 1995, jonka jälkeen se siirtyi Oraclen omistukseen. Se on siitä asti ollut yksi suosituimmista ohjelmointikielistä monille alustoille ja ohjelmistoille ja vaikka se onkin jo kauan ehtinyt vaikuttaa ohjelmointipiireissä, vanhentunut se ei kuitenkaan ole. (Chebbi, 2019)

Android-alustan käyttöönoton jälkeen vuodesta 2008 Java on toiminut oletuskielenä kaikelle Android-ohjelmoinnille. Kuitenkin vuonna 2017 Google alkoi tukea myös Kotlin-ohjelmointikieltä ja vuodesta 2019 Google suosittelee ensisijaisesti sen käyttöä Android-kehityksessä. (Chebbi, 2019)

Java on kielenä suhteellisen helppo oppia. Syntaksi on pitkälti englanninkielistä ja sitä on helppo ymmärtää. Java tarjoaa rikkaan ohjelmointirajapinnan, tietokantayhteydet, työkalut ja paljon muuta, mitä hyvältä ohjelmointikieleltä voidaan odottaa. (Nordeen, 2018)

3 TIETOKANTA

Tietokannat sisältävät digitaalista tietoa. Tietoihin voidaan viitata ja tietoja voidaan etsiä, vertailla, muunnella ja manipuloida nopeasti sekä vähäisillä käsittelykuluilla. Tietokannat rakennetaan ja niitä ylläpidetään tietokannan ohjelmointikieltä käyttäen, joista yleisin on SQL-tietokantaohjelmointi- tai kyselykieli (Structured Query Languange). (Computer Hope, 2019)

3.1 SQLite

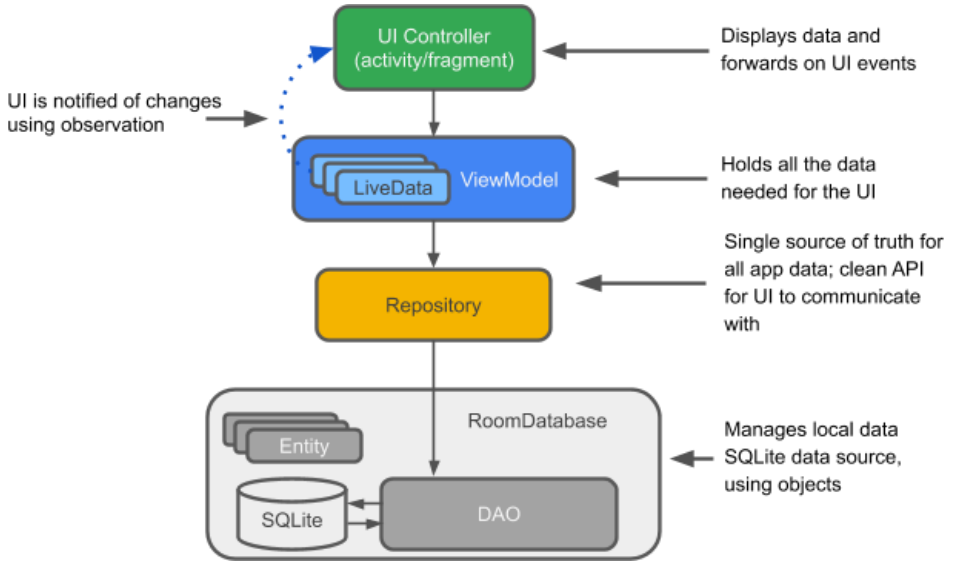
SQLite on prosessin sisäinen kirjasto, joka toteuttaa itsenäisen, palvelittoman SQL-tietokantamoottorin ja se on yksi maailman käytetyimmistä tietokannoista, ja sitä on käytetty lukuisissa projekteissa. SQLiten koodi on tekijänoikeudetonta, joten sitä voidaan käyttää vapaasti kaupalliseen tai yksityiseen tarkoitukseen. SQLite ei sisällä erillisiä palvelinprosesseja, toisin kuin useimmat SQL-tietokannat, vaan se lukee ja kirjoittaa tietoa suoraan tavallisiin levytiedostoihin. (SQLite, n.d)

Koodikantaa tukee kansainvälinen kehittäjätiimi, joka työskentelee kokopäiväisesti SQLiten parissa, edistäen sen suorituskykyä, säilyttäen yhteensopivuuden taaksepäin ja tarjoten myös ammatillista tukea kaikille sitä tarvitseville. SQLite testataan huolellisesti ennen jokaista uutta julkaisua, ja sillä onkin erityisen hyvä maine luotettavuudestaan. SQLite sai alkunsa vuonna 2000 ja kaikki suunnittelupäätökset tehdään ajatuksella, että sen kehitystä jatkettaisiin ainakin vuoteen 2050. (SQLite, n.d)

3.2 Android Room

Room on osa Android-arkkitehtuurikomponentteja, ja se tarjoaa abstraktiokerroksen SQLiten ylle. Se vähentää SQL-vakiokoodien kirjoitusta, tarkistaa SQL-kyselyt käännöshetkellä, tarjoaa sujuvan yhteyden tietokantaan ja samalla hyödyntää täyden SQLiten tehon. Room toimii myös yhdessä LiveDatan kanssa. LiveData on tietojen haltijaluokka, joka ilmoittaa tarkkailijalle, kun tiedot muuttuvat. Tämä tarkoittaa, että tietokanta päivittyy käyttöliittymälle reaaliajassa. (Codelabs, 2020)

Room-arkkitehtuuri koostuu arkkitehtuurikomponenteista. Näitä komponentteja ovat käyttöliittymäohjain (UI Controller), ViewModel, Repository, Room-tietokanta, Data Access Object (DAO) ja entiteetit. Alla olevassa kuvassa (Kuva 1) on havainnollistettu komponentit ja niiden hierarkia, sekä kuvan alla kerrottu kaikista hieman tarkemmin. (Codelabs, 2020)



Kuva 1. Room-komponentit

Entiteetti toteutetaan asettamalla normaalille luokalle eteen huomautus @Entity. Entiteettiluokat ovat yleensä malliluokkia, jotka eivät itsessään sisällä minkäänlaista logiikkaa. Entiteettiluokka vastaa tietokantataulua ja luokan kentät vastaavat taulukon sarakkeita. (Muntenescu, 2017)

DAO-luokka (Data Access Object) määrittelee metodit ja kyselyt, joilla tietokantaa käytetään. Android Room sisältää joitain valmiita SQL-tietokantakyselyjä, esimerkiksi @Insert ja @Update, joihin ei erikseen tarvitse SQL-koodia kirjoittaa. Kuitenkin jos halutaan luoda omia tietokantakomentoja, voidaan lisätä metodin eteen @Query, jonka jälkeen sulkeiden sisälle kirjoitetaan SQL-kysely. DAOn on oltava abstrakti luokka tai Interface, jonka eteen lisätään huomautus @Dao. (Muntenescu, 2017)

Room-tietokanta on abstrakti luokka, jonka eteen lisätään huomautus @Database. Luokalle täytyy ilmoittaa myös käytettävät entiteetit sekä DAOt. Tietokantaluokka toimii pääyhteytenä sovelluksen ja relaatiotiedon välillä. (Muntenescu, 2017)

Repository eli niin sanottu varastoluokka luodaan avustamaan useiden tietolähteiden hallinnassa. Room tietokannan lisäksi Repository voi hallita myös etätietolähteitä, esimerkiksi verkkopalvelinta. ViewModel toimii välikätenä tietokantojen ja käyttöliittymän välillä piilottaen tietokannan toteutuksen ja toiminnan käyttöliittymältä. (Codelabs, 2020)

4 KÄYTTÖLIITTYMÄ

Käyttöliittymä on käyttöjärjestelmän, ohjelman tai laitteen osa, jolla käyttäjä syöttää sekä vastaanottaa tietoa. On olemassa merkkipohjaisia ja graafisia käyttöliittymiä. Merkkipohjaisessa näkyvillä on yleensä vain tekstiä ja komentorivi, jonka kanssa käyttäjä on vuorovaikutuksessa. Usein käytössä on kuitenkin graafinen käyttöliittymä, jossa käyttö perustuu graafisten objektien ja toimintojen käyttöön. (Helsingin yliopisto, n.d)

4.1 Käyttöliittymäsuunnittelu

Käyttöliittymäsuunnittelu tarkoittaa nimensä mukaisesti suunnitelman tekoa esimerkiksi verkkosivustoille ja erilaisille sovelluksille. Käyttökokemus ja vuorovaikutus tulevien käyttäjien ja tuotteen välillä on ensisijaisen tärkeä ottaa huomioon suunnitelmaa tehdessä ja siinä koko käyttöliittymäsuunnittelun idea piileekin. (The Design Blitz, n.d.)

Päätavoite käyttöliittymäsuunnittelussa on tehdä käyttäjän vuorovaikutuksesta yksinkertaista ja tehokasta. Käyttäjäkeskeiset suunnittelumallit ovatkin parhaita käyttöliittymäsuunnitelmia. Mikäli käyttöliittymä toimii ilman mitään häiriötekijöitä, se voidaan silloin luokitella hyväksi käyttöliittymäksi. (The Design Blitz, n.d.)

Hyvän käyttöliittymän suunnittelu vaatii taitoa ja ymmärrystä käyttäjien tarpeita kohtaan. On olemassa monia prosesseja ja sääntöjä, joita käyttöliittymän suunnittelussa on noudatettava. (The Design Blitz, n.d.)

Jokaisen hyvän käyttöliittymän tulisi olla selkeä, sillä se estää käyttäjiä tekemästä turhia virheitä ja tekee tärkeistä tiedoista helposti näkyviä ja saatavia. Hyvä käyttöliittymä on myös tiivis, joka ei kuitenkaan tarkoita, etteikö käyttöliittymä sisältäisi kaikkea tarvittavaa informaatiota. Sisällössä pitäisi kiinnittää huomiota, mitkä asiat kannattaa pitää lyhyenä, ja mitkä taas vaativat enemmän avaamista. (The Design Blitz, n.d.)

Käyttöliittymän on oltava johdonmukainen, jotta käyttäjät saadaan nopeasti omaksumaan sovelluksen eri käyttötapoja, esimerkiksi miten mikäkin painike, kuvake tai muu käyttöliittymäelementti toimii ja mihin ne on sijoitettu. Käyttöliittymään tuotetaan myös helppolukuista tekstiä ilman monimutkaisia sanoja ja muita vaikeasti ymmärrettäviä ilmaisuja. (The Design Blitz, n.d.)

Responsiivisuus on ominaista hyvälle käyttöliittymälle. Käyttöliittymä antaa käyttäjälle palautetta ja pitää käyttäjän ajan tasalla, mitä tapahtuu milloinkin. Latausajat tulisi myös pyrkiä pitämään mahdollisimman lyhyinä. (The Design Blitz, n.d.)

Etenkin mobiilisovellusta käyttäessä virhepainalluksien todennäköisyys on suuri. Tämän vuoksi pitäisi käyttäjällä olla mahdollisuus peruuttaa esimerkiksi jonkin tiedon poistaminen. Käyttäjän ei pitäisi joutua tilanteeseen, jossa ei tiedä mitä pitäisi tehdä. Tämän takia käyttöliittymän pitää olla anteeksiantavainen. (The Design Blitz, n.d.)

Käyttöliittymän on tunnuttava tutulta, joka ei kuitenkaan tarkoita, että kaikkien sovelluksien pitäisi olla identtisiä. Tällä tarkoitetaan esimerkiksi elementtien sijoittelua paikkoihin, jossa käyttäjä niiden olettaisi loogisesti olevan. (The Design Blitz, n.d.)

Tehokas käyttöliittymä tekee sen mitä sen on tarkoitettu tekevän ja mitä käyttäjä haluaa sillä saavuttaa. Tehokkuus tarkoittaa myös mahdollisimman helppoa käyttöönottoa ja käyttämistä. (The Design Blitz, n.d.)

Houkuttelevuutta ei saa suunnittelussa unohtaa. Vaikka helppokäyttöisyys ja tehokkuus varmasti ovat lopussa tärkeimpiä ominaisuuksia, ei voida poissulkea visuaalisuutta. Moni käyttäjä todennäköisesti valitsee kahdesta toiminnoiltaan täysin samasta vaihtoehdosta sen, joka on visuaalisesti näyttävämpi. (The Design Blitz, n.d.)

4.2 **Käyttöliittymäsuunnittelu Androidille**

Android on tällä hetkellä maailman käytetyin mobiilialusta, joten sovelluksiin kannattaa panostaa jo suunnitteluvaiheessa. Parhaat sovellukset Android-kaupassa ovatkin rakennettu noudattaen alustan asettamaa yleistä ohjenuoraa. Google on kehittänyt Material Design -muotokielen, josta löytyy ohjenuorat Android-sovellusten suunnittelun tueksi. (Babich, 2018)

Jotkut suunnittelijat uskovat, että tehdäkseen uuden näyttävän sovelluksen, pitäisi jokainen toiminto ja vuorovaikutuksellisuus suunnitella itse alusta asti. Jokainen komponentti, joka ei seuraa Android-muotokieltä pakottaa käyttäjän päättelemään sen toiminnan itse, koska sen ulkonäkö ei ole entuudestaan täysin tuttu. Lisäksi omien toimintojen rakentaminen vie aikaa projektista ja yleensä siinä samalla myös vaatii enemmän rahaa. Valmiiden komponenttien käyttäminen vähentää huomattavasti työmäärää ja helpottaa sovelluksen tekemistä. (Babich, 2018)

Animaatiot sovelluksessa auttavat käyttäjiä ymmärtämään sivujen väliset yhteydet. Animaatiot myös ilmaisevat käyttöliittymän tarkoituksen ja tekevät siitä dynaamisemman. Ohjeita myös animaatioihin löytyy Material Designista. (Babich, 2018)

Laitteita on saatavilla eri kokoisia ja eri tarkkuuksia omaavilla näytöillä. Tämän vuoksi sovellukset on optimoitava sopivaksi monille eri laitteille. Androidissa onkin käytössä pikseliyksikkö, tiheydestä riippumaton pikseli, joka skaalautuu hyvin eri pikselitiheyden omaaviin näyttöihin. Vaikka onkin

tehty tällaisia helpottavia ominaisuuksia, ei silti ole yksikertaista tehdä sovellusta, joka toimisi hyvin joka laitteella. (Babich, 2018)

Myös laitteiston tehoissa ilmenee huomattavia eroja. Nämä asiat myös täytyy pitää mielessä sovellusta tehdessä. Hyvän sovelluksen tulisi toimia vanhoilla laitteilla sekä uusimmilla malleilla. Mahdollisuus on julkaista sovellus esimerkiksi vain tietystä ohjelmistoversiosta uudemmille laitteille, mutta silloin täytyy pitää mielessä, että ei voida saavuttaa kaikkia mahdollisia käyttäjiä. (Babich, 2018)

4.3 **Figma-suunnittelusovellus**

Figma on selainpohjainen käyttöliittymäsuunnittelutyökalu, joka tarjoaa erinomaiset ja helppokäyttöiset ominaisuudet ja toiminnot niin aloittelijalle kuin ammattilaiselle. Figmasta on myös ladattavissa työpöytäversiot Windowsille ja Mac OS:lle. Nämä ovat kuitenkin toiminnoiltaan samanlaiset kuin selainversiot. (Bracey, 2018)

Figmaan voi ladata erilaisia suunnittelupohjia, esimerkiksi Android UI-kitin. Nämä lisäävät Figmaan valmiita komponentteja ja elementtejä, jotka tekevät suunnittelusta helpompaa ja takaavat näyttävän sekä ohjesääntöjä seuraavan lopputuloksen. (Bracey, 2018)

Figmassa on mahdollista tehdä sovelluksesta alustava prototyyppi. Eri näkymien välille voidaan luoda yhteyksiä, ja näin simuloida kuinka lopullinen sovellus tulisi toimimaan. (Bracey, 2018)

Yksi Figman suurimmista ja kehutuimmista vahvuuksista on reaaliaikainen yhteistyö verkossa. Se mahdollistaa työskentelyn saumattomasti samassa projektissa tiimiläisten kanssa. Työt tallentuvat automaattisesti verkkoon ja päivittyvät heti kaikille näkyviksi. Figmassa on laajat yhteistyöominaisuudet ja toiminnot, esimerkiksi kommenttien lisäys suoraan kanvaasille muiden nähtäväksi. (Bracey, 2018)

5 SOVELLUKSEN KÄYTTÖLIITTYMÄ

Idea sovellukselle on peräisin itse työn tekijältä. Tarvittiin yksinkertainen sovellus, johon käyttäjä voi suunnitella ja kirjoittaa muistiin oman harjoituslistan kuntosalilla käyntiä varten. Sovelluksen pääidea on, että sinne ensiksi voidaan lisätä harjoitus ja yhteen harjoitukseen lisättäisiin kaikki siihen tarvittavat liikkeet. Jokaiselle liikkeelle asetetaan sarjat, toistot ja painojen määrä. Liikkeitä voidaan helposti muokata, ja erityisesti painon muokkaus on tärkeää, sillä sitä yleensä lisätään onnistuneen suorituksen jälkeen seuraavaa harjoituskertaa varten.

5.1 Ideointi ja suunnittelu

Ensimmäisessä sovellusversiossa käyttöliittymä on melko yksikertainen ja listarakenteinen. Näkymien väliin on lisätty liukumisanimaatiot, jotka parantavat myös käyttäjän havainnointia ja ymmärrystä sovelluksen toiminnasta. Myös Androidille ominaisia leijuvia painikkeita (Floating Action Button) on käytetty sovelluksessa uuden harjoituksen ja liikkeiden luonnissa, jotka jo itsestään kertovat käyttäjälle, mitä niitä painaessa tapahtuu. Käyttöliittymä on tarkoitus pitää mahdollisimman yksinkertaisena myös lopullisessa versiossa, että tietoja on helppo muunnella nopeasti ja tehokkaasti.

Sovelluksen lopullisesta käyttöliittymästä oli ideoinnin alusta asti olemassa tietty näkemys. Yksinkertaiset listanäkymät olisivat aluksi riittäviä, kunhan sovellus muuten toimisi niin hyvin, että sitä voisi alkaa käyttämään kuntosalilla. Kun sovelluksen ensimmäinen versio olisi valmis, voisi käyttöliittymään panostaa enemmän. Tämän vuoksi sovellukselle tehdään käyttöliittymäsuunnitelma, joka havainnollistaa sovelluksen lopullista ulkonäköä. Opinnäytetyö-prosessin aikana ei kuitenkaan sovellusta saatu ohjelmoitua täysin lopullisen suunnitelman mukaiseksi.

5.2 Käyttöliittymäsuunnitelman teko Figmalla

Käyttöliittymäsuunnittelun alussa nähtiin parhaaksi asentaa Figma-suunnittelusovelluksesta työpöytäversio, mutta se kuitenkin toimii aivan samalla tavalla kuin verkkosivustolla oleva. Halusimme aluksi Figmaan valmiita Android-elementtejä, joita voisimme käyttää helposti oman sovelluksen suunnittelussa, joten latasimme valmiina olevan Android UI -kit -sarjan. Erilaisia sarjoja löytyy monia, mutta tämä oli myös Figman omilla sivuilla ladattavissa. UI-kit ladataan ja se siirtyy automaattisesti Figma-sovellukseen heti salasanan ja käyttäjänimen syöttämisen jälkeen. UI-kit ilmestyy sovellukseen uutena projektina, ja sieltä voidaan suoraan kopioida ja siirtää erilaisia elementtejä omaan projektiin.

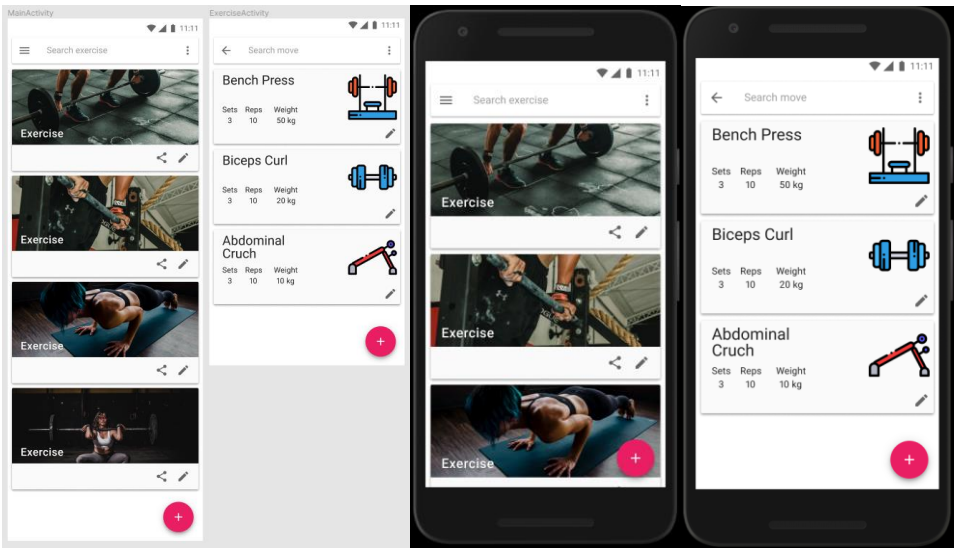
Figman aloitussivulla näkyvissä ovat kaikki projektit. Projekti voidaan avata tuplaklikkaamalla sitä. Kun tyhjä projekti on auki, yläreunan

työkaluvalikosta valitaan ensin risuaitaa muistuttavasta kuvakkeesta sovellukselle kehykset, joka tässä projektissa on Android. Nyt kehyksen päälle voi kopioida UI-kitistä erilaisia elementtejä. Kopioitujen elementtien pitäisi olla valmiiksi oikeassa koossa, etenkin kun kehykseksi on valittu Android.

Sanat aktiviteetti ja dialog ovat Android-kehityksessä käytettäviä nimityksiä eri näkymistä. Figma-ohjelmassa näitä sanoja ei ole, kuten esimerkiksi Android dialog vastaa Figmassa overlayta. Tässä osiossa on käytetty hieman molempia ilmaisuja.

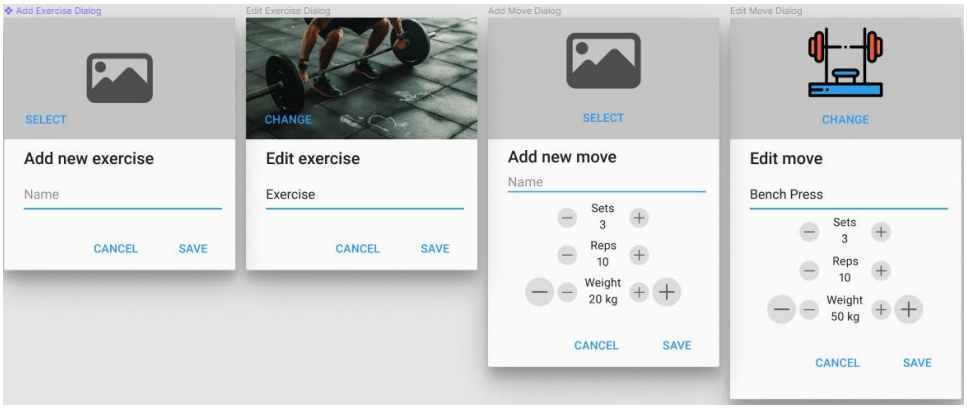
Jokaiselle sovelluksen uudelle sivulle eli aktiviteetille on luotava oma kehyksensä. Tässä projektissa haluttiin olevan kaksi erilaista aktiviteettia. Alla olevassa kuvassa (Kuva 2) vasemmalla näkyy aktiviteettien suunnittelunäkymä. Figmassa suunnitelma on mahdollista näyttää myös esitelmätilassa, ja tätä havainnollistavat oikealla olevat kuvat.

Kuvan suunnittelunäkymässä ensimmäinen aktiviteetti venyy alaspäin pitkulaisena, sillä se näyttää kaikki elementit mitä se pitää sisällään, mutta esittäjänäkymässä aktiviteetti skaalautuu näytölle ja sitä voidaan vierittää alaspäin. Elementit kuten tilapalkki, hakupalkki ja leijuva lisäys-painike on lukittu näyttöön, eli ne pysyvät paikallaan myös näyttöä vierittäessä esittäjätilassa.



Kuva 2. Figma-suunnittelunäkymä ja esitysnäkymä

Aktiviteettien lisäksi sovelluksessa on neljä Dialog-ikkunaa (Kuva 3). Dialog on kuin ponnahdusikkuna, joka aukeaa suoraan aktiviteetin päälle. Sovelluksessa näitä ovat harjoituksien ja liikkeiden lisäys -ja muokkaus dialogit. Dialogit aukeavat lisäys- tai muokkauspainikkeita painamalla, joita ovat leijuvat lisäyspainikkeet sekä kynän kuvat sovelluksessa.



Kuva 3. Dialog-ikkunat

Figma-suunnittelusovelluksessa on mahdollista tehdä sovelluksesta prototyyppi sen toiminnan demonstroimista varten. Aktivoidaan esimerkiksi ensimmäisen aktiviteetin lisäyspainike ja siirrytään Figmassa oikealla olevaan prototyyppivalikkoon. Interaktio-asetuksien alla voidaan määrittää, mitä tapahtuu, kun painiketta painetaan eli valitaan On Tap-asetus. Figmassa on muitakin interaktioita kuin pelkkä painallus, mutta tässä projektissa käytetään vain sitä, sillä vain painaessa elementtejä toiminnot tapahtuvat. Painikkeesta tulisi avautua uuden harjoituksen luonti Dialog. Joten OnTap-valinnan alta valitaan kohta Open Overlay ja sen perään valitaan Add Exercise Dialog, joka on siis dialogin nimi projektissa. Näin overlay aukeaa aktiviteetin päälle, eikä uuteen ikkunaan. Jos halutaan avata kokonaan uusi näkymä eli aktiviteetti, Open Overlayn tilalle asetetaan Navigate To, ja sen perään asetetaan haluttu näkymä tai sivu.

Edellä mainittuja interaktioita tehtiin sovelluksessa kaikkien eri näkymien välille ja toteutettiin myös niistä palaaminen edelliselle näkymälle. Myös animaatiot siirtymien välille voidaan lisätä prototyyppiasetusten alta, jossa juuri määriteltiin sivujen väliset interaktiot. Näin sovellusta voidaan testata ja siitä voidaan kerätä palautetta jo ennen itse ohjelmointiosuutta, joka tekee juuri Figman kaltaisesta sovelluksesta erittäin käytännöllisen ja tarpeellisen.

Ajan riittämättömyyden vuoksi ei opinnäytetyön aikana ollut mahdollista ohjelmoida sovelluksen ensimmäistä versiota täysin tämän käyttöliittymäsuunnitelman mukaiseksi. Puuttumaan jäävät muun muassa kuvat, jotka luovat sovellukselle tyylikkäämmän ulkonäön, jotkin animaatiot ja mahdollisuus jakaa harjoituksia muille käyttäjille. Myös päävalikko ja ylhäällä oleva hakupalkki jää sovelluksen ensimmäisestä versiosta puuttumaan. Seuraavissa kappaleissa kerrotaan sovelluksen ohjelmoinnissa, jossa myös on nähtävissä, että sovelluksen ulkonäkö ei ole aivan yhtä siisti kuin suunnitelmassa.

6 MOBIILISOVELLUKSEN TOTEUTUS

Koska opinnäytetyön tekijällä ei ollut aiempaa kokemusta Android ohjelmoinnista sekä siihen liittyvistä erilaisista ominaisuuksista ja työkaluista, tätä työtä ei ollut täysin mahdollista tehdä vaihe vaiheelta minkään valmiin suunnitelman mukaan. Sovellusta siis tehtiin sitä mukaa, kun tiedot ja taidot kasvoivat.

Työn tekoa organisoitiin ja ylläpidettiin tavoitteet mielessä pitäen ja tulevat päivät suunniteltiin valmiiksi, jotta olisi helppo tarttua työhön, kun on tiedossa mitä pitää tehdä. Tarkoitus oli, että ei tehtäisi oman mielen mukaan mitä vaihetta haluaa ja milloin haluaa, vaan että työn teko olisi suunniteltua ja järjestyksellistä.

Jotkut asiat ohjelmoinnissa osoittautuivat erittäin hankaliksi ja niihin kului aikaa paljon enemmän kuin olisi saanut kulua, esimerkiksi tietokantojen ja käyttöliittymän vuorovaikutus oli vaikea saada toimimaan oikein. Kuitenkin sovellus saatiin kehitettyä suunniteltuun tilaan opinnäytetyöjakson aikana.

Kuvat koodeista ovat tarkoituksella tähän dokumenttiin yritetty pitää mahdollisimman lyhyinä, joten esimerkiksi import-tuonnit eivät niissä näy. Android Studio kuitenkin hoitaa tuonnit automaattisesti koodia kirjoittaessa. Tietokannan luonti on työssä selitetty hieman tarkemmin kuin muut ohjelmointiosuudet, sillä tietokannat olivat osa työstä mihin erityisesti haluttiin paneutua.

6.1 Tietokantojen rakentaminen

Kun sovelluksen idea oli tiedossa, yksi ensimmäisistä työvaiheista sen ohjelmoinnissa oli löytää tapa, kuinka tiedot saataisiin tallennettua laitteelle. Android tietokantaratkaisuja etsiessä löytyi SQLiten lisäksi Android Room-kirjasto, joka tarjoaa abstraktin kerroksen SQLiten päälle taaten sujuvan tietokannan käytön ja vähentäen pakollisten vakiotekstien kirjoittamista.

Room:n toteuttamista harjoiteltiin seuraamalla opetusmateriaaleja ja lukemalla androidin dokumentaatioita. Tietokannat ja niiden toteuttaminen sekä ymmärtäminen ovat olleet aina haastavia, joten ROOM:n harjoittelemisessa asia ei ollut yhtään erilainen.

Harjoituslistasovelluksessa tietokanta täytyy rakentaa yhden suhde moneen -periaatteella. Sovelluksessa siis ensin luodaan harjoitus. Jokainen luotu harjoitus voi sisältää monia eri liikkeitä.

Room-tietokantaa käyttääkseen se on ensin implementoitava build.gradle-tiedostoon. Gradle-tiedostoja on olemassa kaksi erilaista, ja ne ovat Module:app ja Project. Tämän tyyppiset implementoinnit on lisättävä app-

moduuliin. Alla olevassa kuvassa (Kuva 4) näkyy, kuinka Room lisätään build.gradle tiedostoon dependenssien eli niin sanottujen riippuvuuksien alle.

```
dependencies {

    def room_version = "2.2.3"

    implementation "androidx.room:room-runtime:$room_version"
    annotationProcessor "androidx.room:room-compiler:$room_version"

}
```

Kuva 4. Room-riippuvuudet

6.1.1 Entiteetti-luokat

Dependenssien lisäyksen jälkeen tietokannan teko aloitettiin luomalla entiteettiluokat harjoituksille (Kuva 5). Jokainen luotu entiteetti vastaa tietokantataulua. Entiteettiluokan teko aloitetaan luomalla normaali Java-luokka, mutta sen jälkeen lisätään luokan eteen huomautus @Entity. Huomautuksen perässä on ilmoitettu myös tableName, joka määrittää tietokantataulun nimen. Jos nimeä ei erikseen anneta, taulun nimi on sama kuin luokalla.

Harjoitus-entiteetti sisältää kaksi saraketta, id ja nimi. Entiteetti tarvitsee aina pääavaimen, joka on tässä tapauksessa id ja se on määritetty automaattisesti kasvavaksi. Nimen alla näkyy myös luokan rakentaja sekä get - ja set-metodit.

```
@Entity(tableName = "exercise_table")
public class Exercise {

    @PrimaryKey(autoGenerate = true)
    private int id;

    private String exercise_name;

    public Exercise(String exercise_name) {
        this.exercise_name = exercise_name;
    }

    public void setId(int id) {
        this.id = id;
    }
    public int getId() {
        return id;
    }
    public String getExercise_name() {
        return exercise_name;
    }
```

Kuva 5. Harjoitus-entiteetti

Liike-entiteetti (Kuva 6) on pääpiirteiltään samanlainen kuin edellä oleva harjoitusentiteetti. Kuitenkin on muistettava näiden taulujen yhden suhde moneen -yhteys, joka on hoidettava liike-entiteettiä tehtäessä.

Alla olevassa kuvassa @Entity huomautuksen jälkeen on ilmoitettu jälleen nimi kyseiselle taululle. Tämän jälkeen on ilmoitettu viiteavain foreignKey, joka haetaan harjoitusentiteetistä. ParentColumn vastaa harjoitusentiteetin id:tä. ChildColumn taas luodaan liike-entiteettiin ja se viittaa parentColumn id:seen. Lisäksi onDelete = CASCADE tarkoittaa, että jos harjoitus poistuu, kaikki sen alla olevat liikkeet poistuvat.

Liikkeelle ominaiset sarakkeet lisätään normaalisti, joita tässä tapauksessa ovat nimi, sarjat, toistot sekä painojen määrä. Myös liikkeellä on automaattisesti kasvava id, joka toimii myös pääavaimena. Get -ja set-metodit ovat kuvasta piilotettu tilan säästämiseksi.

```
@Entity(tableName = "move_table",
        foreignKeys = @ForeignKey(entity = Exercise.class,
        parentColumns = "id",
        childColumns = "exerciseId",
        onDelete = CASCADE))
public class Move {

    @PrimaryKey(autoGenerate = true)
    private int id;

    private String move_name;
    private int set_count;
    private int rep_count;
    private float weight_count_kg;

    private int exerciseId;

    public Move(String move_name, int set_count, int rep_count,
                        float weight_count_kg, int exerciseId) {
        this.move_name = move_name;
        this.set_count = set_count;
        this.rep_count = rep_count;
        this.weight_count_kg = weight_count_kg;
        this.exerciseId = exerciseId;
    }
    //Getters and Setters here
}
```

Kuva 6. Liike-entiteetti

6.1.2 Data Access Object

Entiteettien jälkeen luodaan DAO-luokat (Data Access Object), joissa määritellään SQL-kyselyt (Kuva 7). Lisätään alkuun huomautus @Dao. Harjoituksessa on viisi eri metodia, lisäys, päivitys, poisto, kaikkien harjoitusten poisto ja kaikkien harjoitusten haku. Room:n ansioista kaikkiin metodeihin ei tarvitse kirjoittaa SQL-lauseketta, vaan pelkkä huomautus riittää. Kuitenkin Room:ssa ei ole valmiina huomautusta kaikille metodeille, joten osa vaatii perinteisen SQL-lausekkeen kirjoituksen, joten käytetään ensin huomautusta @Query ja sen perään syötetään haluttu SQL-lauseke.

```
@Dao
public interface ExerciseDao {

    @Insert
    void insert(Exercise exercise);

    @Update
    void update(Exercise exercise);

    @Delete
    void delete(Exercise exercise);

    @Query("DELETE FROM exercise_table")
    void deleteAllExercises();

    @Query("SELECT * FROM exercise_table")
    LiveData<List<Exercise>> getAllExercises();
}
```

Kuva 7. Harjoitus DAO

Liike-DAO on pääpiirteissään samanlainen kuin harjoitus-DAO. Liikkeitä voidaan lisätä, päivittää, poistaa ja hakea kaikki liikkeet. Liikkeessä ero harjoitukseen on, että pitää voida hakea liikkeet tietyn harjoituksen alla. Tähän käytetään alla näkyvää kyselyä.

```
@Query("SELECT * FROM move_table WHERE exerciseId=:exerciseId")
LiveData<List<Move>> findMovesForExercise(int exerciseId);
```

Kuva 8. Hae liikkeet harjoitukselle

DAO-kuvissa on näkyvissä LiveData-objekteja, joita käyttämällä käyttöliittymä päivittyy dynaamisesti tietokannassa tapahtuvien muutosten jälkeen. Jos esimerkiksi poistat tiedon, se poistuu myös käyttäjältä näkyvistä.

6.1.3 ROOM-tietokanta, Repository ja ViewModel

Entiteettien ja Data Access Objectien luontien jälkeen tehdään itse tietokantaluokka (Kuva 9). Luokka on oltava abstrakti. Lisätään alkuun huomautus @Database, jonka perään lisätään edellisissä kohdissa tehdyt entiteetit ja asetetaan myös versionumero.

Harjoituslistasovellukselle suoritettiin myös esitäyttö, eli tietokantaan lisätään valmiiksi tietoa ensimmäistä asennusta varten. Sovellukseen lisättiin kolme erilaista harjoitusta ja jokaisessa niissä on muutama liike. Esitäytön koodista ei liitetä erillistä kuvaa dokumenttiin. Alla olevassa kuvassa kommentoitu rivi .addCallback(roomCallback) liittyy myös esitäyttöön, joten sitä ei vaadita, jos esitäyttöä tietokannalle ei tehdä.

```
@Database(entities = {Exercise.class, Move.class}, version = 1)
public abstract class ExerciseDatabase extends RoomDatabase {

    private static ExerciseDatabase instance;

    public abstract ExerciseDao exerciseDao();
    public abstract MoveDao moveDao();

    public static synchronized ExerciseDatabase getInstance(Context context){
        if (instance == null){
            instance = Room.databaseBuilder(context.getApplicationContext(),
                    ExerciseDatabase.class, "exercise_database")
                    .fallbackToDestructiveMigration()
                    //.addCallback(roomCallback)
                    .build();
        }
        return instance;
    }
```

Kuva 9. Harjoitus tietokanta

Tietokannan jälkeen sovellukseen luotiin Repository -ja ViewModel-luokat. Erityisesti Repositoryn ja ViewModelin toiminta oli työn alussa vaikea ymmärtää. Repository-luokkaa ei olisi pakollista tehdä, mutta sitä pidetään kuitenkin hyvänä toimintatapana ja tässä projektissa se toteutettiin. Repositoryn kautta voidaan hallita juuri luotua Room-tietokantaa, mutta lisäksi myös etätietolähteitä, joita ei tässä projektissa kuitenkaan käytetty. ViewModel piilottaa käyttöliittymältä tietokantojen toiminnan ja toteutuksen, joka helpottaa tietokantojen käyttöönottoa ja hallintaa käyttöliittymän kautta sekä lisää suorituskykyä sovellukseen.

6.2 **Sovelluksen ohjelmointi**

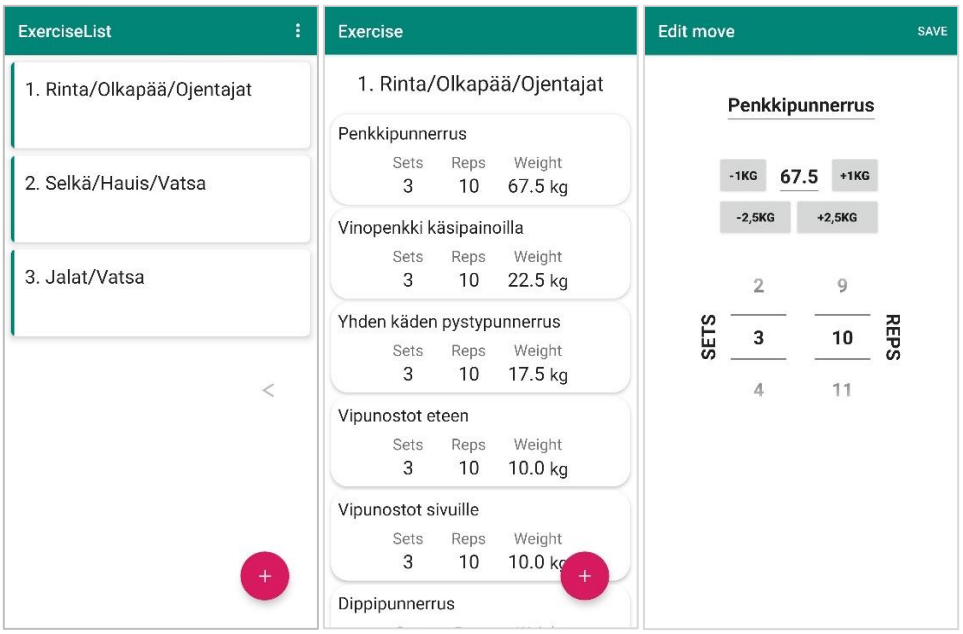
Sovelluksen ohjelmoinnin ensimmäinen osa oli saada kuntoon sen ydintoiminta, johon kuuluu välttämättömänä osana jo edellisessä kappaleessa läpi käyty tietokantojen rakentaminen. Muita välttämättömiä ominaisuuksia sovellukselle olivat kaikkien harjoituksien näkyminen listana ensinäkymässä. Ensinäkymässä on myös mahdollisuus lisätä uusi harjoitus.

Harjoitus-listaobjektia painettaessa aukeaisi uusi sivu, jossa näkyvillä olisi listana kaikki valittuun harjoitukseen kuuluvat liikkeet. Näkyvillä aina yhdessä liike-listaobjektissa olisi liikkeen nimi, sarjat, toistot sekä painojen määrä. Liike-sivulla on myös painike uuden liikkeen lisäykselle, samaan tapaan kuin harjoitusnäkymässä.

Liikeobjektia painettaessa aukeaa valitun liikkeen muokkaamissivu. Kesken harjoituksen liikkeiden painon muokkaus seuraavaa harjoituskertaa varten on kaikista yleisin muokkaus mitä sille tehdään. Painoa voidaan lisätä tai vähentää helposti. Myös liikkeen nimeä, sarjoja ja toistoja voidaan muokata.

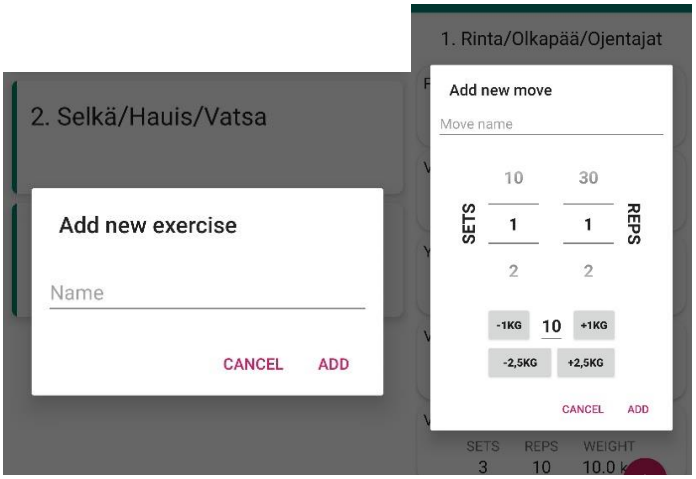
Sovellus sisältää kolme erilaista aktiviteettia (Kuva 10). Aktiviteetti on kuin näkymä, esimerkiksi etusivu on yksi aktiviteeteista, joka on tässä projektissa nimetty MainActivity. MainActivity ja ExerciseActivity näyttävät

listana kaikki luodut harjoitukset ja liikkeet. Kolmannessa EditMoveActivityssa voidaan muokata valittua liikettä.



Kuva 10. Aktiviteetit

Sovelluksessa käytetään kahta Dialog-ikkunaa (Kuva 11), NewExerciseDialog ja NewMoveDialog, jotka saadaan avattua painamalla leijuvia lisäyspainikkeita. Dialog toimii kuin ponnahdusikkuna aueten aktiviteetin päällä eikä avaa kokonaan uutta näkymää, lisäten käyttömukavuutta ja visuaalisuutta sovellukseen. Ensimmäisessä harjoituksen luonti Dialogissa lisätään uusi harjoitus, ja sille syötetään vain harjoituksen nimi. Toinen Dialog, liikkeen lisäys on pitkälti samanlainen kuin harjoituksen lisäys, mutta nimen lisäksi syötetään liikkeelle sarjat, toistot ja painojen määrä. Aktiviteeteille ja dialogeille luodaan myös omat layout-tiedostot, jossa määritellään niiden ulkonäkö ja objektien asettelut, esimerkiksi tässä sovelluksessa aktiviteetti MainActivity käyttää layout-tiedostoa nimeltä activity_main.xml.



Kuva 11. Dialog-ikkunat

Aktiviteetit kuvassa (Kuva 10) näkyvät listat on tehty käyttäen RecyclerView-näkymää. RecyclerView tarvitsee oman Adapter-luokan, joka ohjailee sen toimintoja, esimerkiksi poistaa listaobjektin näkyvistä poistamatta sitä silti tietokannasta. Niille pitää myös erikseen luoda omat layout-tiedostot ja ne määrittävät miltä yksi listaobjekti näyttää. Esimerkiksi activity_main.xml-tiedostoon on ohjelmoitu RecyclerView-näkymä, jossa määritellään listan koko ja missä kohtaa näytöllä sen halutaan olevan. Tässä työssä melkein koko näyttö on varattu listalle kummassakin aktiviteetissa MainActivity ja ExerciseActivity. Layout-tiedosto activity_main.xml :n RecyclerView komponentti saa listaobjektin ulkonäön ja sijoittelun tiedot toisesta layout-tiedostosta nimeltä exercise_item.xml. Listaobjektilla tarkoitetaan yhtä listatietoa, esimerkiksi kuvassa näkyvä harjoitus Rinta/Olkapää/Ojentajat, mutta sama exercise_item.xml-asettelu pätee kaikkiin luotuihin harjoituslistaobjekteihin.

Jokainen listaobjekti on CardView eli niin sanottu korttinäkymä. Näin saadaan listasta korttimaisen näköinen, joka lisää käyttömukavuutta sovellukselle, ja tällaista näkymää nykyään paljon sovelluksissa nähdäänkin.

Jokainen listaobjekti on vuorovaikutteinen. Sovelluksessa pyyhkäisemällä listaobjektia vasemmalle tai oikealle aktivoidaan tiedon poisto -funktio. Sovelluksen ensimmäisessä vaiheessa, kun tieto pyyhkäistiin pois, alareunaan ilmestyi ilmoituslaatikko eli Snackbar, jossa oli mahdollisuus peruuttaa tiedon poisto. Kuitenkin Snackbar poistui melko äkkiä näkyvistä ja tieto menetettiin, joka johti vahinkopoistoihin.

Sovellus testattiin muutamalla käyttäjällä ja niin siitä saatiin kerättyä kriittisimpiä virheitä ja bugeja. Palautteen ansiosta muutettiin poisto käyttäjäystävällisemmäksi. Kun tieto halutaan poistaa, voidaan se edelleen pyyhkäistä, mutta sen jälkeen näytölle aukeaa AlertDialog, jossa käyttäjä voi varmistaa tai peruuttaa poiston. Sovellus sai palautetta myös painomäärän muuttamisesta. Ensin sovelluksen liikkeen muokkauksessa ja lisäyksessä oli mahdollisuus korottaa tai laskea painoa vain 2,5 kilogramman välein, koska niin painot yleensä kuntosalilla jakautuvat. Kuitenkaan näin ei aina ole varsinkaan alle kymmenen kilon painomäärissä ja siksi sovellukseen lisättiin mahdollisuus vaihtaa painoa myös yksi kilogramma kerrallaan.

Intent-lähetysobjektia (Kuva 12) käyttäen voidaan lähettää tietoja aktiviteettien välillä. Sovelluksen teossa ilmeni vaihe, joka kulutti paljon aikaa, ja se saatiinkin juuri ratkaistua käyttäen Intent-lähetystä. Harjoitusta painaessa aukeaa sen harjoituksen id-numeron alla olevat liikkeet uuteen aktiviteettiin. Intentiä käyttäen on lähetettävä siis tiedot valitusta harjoituksesta uudelle aktiviteetille. Alla olevassa kuvassa on MainActivity-luokan koodi, jolla lähetetään harjoituksen tiedot uuteen aktiviteettiin. Harjoitusobjektia painettaessa avataan siis ExerciseActivity oikealla id-numerolla.

```
adapter.setOnItemClickListener(new ExerciseAdapter.OnItemClickListener() {
    @Override
    public void onItemClick(Exercise exercise) {

        Intent intent = new Intent(MainActivity.this, ExerciseActivity.class);

        int exerciseId = exercise.getId();
        intent.putExtra(ExerciseActivity.EXTRA_ID, exerciseId);
        intent.putExtra(ExerciseActivity.EXTRA_TITLE,
                exercise.getExercise_name());
        startActivityForResult(intent, OPEN_EXERCISE_REQUEST);
    }
});
```

Kuva 12. Intent-lähetys

Kun tiedot on lähetetty toiseen aktiviteettiin, otetaan vastaan alla olevan kuvan mukaisesti (Kuva 13). Kuvassa ExerciseActivityn koodi, jolla otetaan vastaan pelkkä harjoituksen id-numero. Näin ollen saatiin valitun harjoituksen id tietoon ja harjoituksen avattaessa saatiin näkyviin vain sen alla olevat liikkeet.

```
final Intent intent = getIntent();
int exerciseId = intent.getExtras().getInt(EXTRA_ID);
```

Kuva 13. Intent-vastaanotto

Kun sovelluksen ensimmäinen versio oli ohjelmoitu, sitä testattiin ja otettiin samalla käyttöön myös kuntosalilla. Niin kuin työssä on ennemmin mainittu, sovellusta ei ollut aikaa ohjelmoida täysin käyttöliittymäsuunnitelman mukaiseksi. Kuitenkin ensimmäinen versio on toimiva, helposti ymmärrettävä ja melko käyttäjäystävällinen. Käyttäjä ei voi vahingossa saada sovellusta rikki tai virhepainallusten takia poistaa tietoja.

7 YHTEENVETO

Opinnäytetyön tavoitteena oli oppia lisää Android-ohjelmoinnista, tietokannoista ja käyttöliittymäsuunnittelusta. Tavoitteena oli myös saada tehtyä ensimmäinen versio kuntosalisovelluksesta sekä suunnitella käyttöliittymä lopulliselle sovellukselle.

Toimivan käyttöliittymän ja tietokannan rakennus Room-arkkitehtuurikomponentteja käyttäen Android-mobiililaitteelle osoittautui paljon vaativammaksi tehtäväksi kuin oli odotettu. Kuitenkin vaikeuteen sisältyi tietämättömyyttä, joka oli projektissa odotettavissa, sillä tekijällä ei ollut juuri näistä aiheista aikaisempaa kokemusta.

Projektin jälkeen asioita voi katsoa hieman eri tavoin. Projektissa oli muutamia hankalia ongelmatilanteita, jotka veivät paljon aikaa, esimerkiksi harjoitus id:n lähetys toiselle aktiviteetille Intent-toimintoa hyödyntäen. Tämänkin asian oppimisen jälkeen työnteko nopeutui ja kyseistä toimintoa tarvittiin projektissa useissa tilanteissa. Android ohjelmoinnissa yksinkertaisenkin asian lisäys sovellukseen tuntui aluksi vaikealta, ja ne vaativat paljonkin koodin kirjoitusta.

Room-tietokannan rakennus varsinkin näin pienessä skaalassa tuntuu projektin jälkeen mietittynä melko helpolta toteuttaa. Kuitenkin Roomin opiskelu oli aluksi haastavaa, varsinkin sen ymmärtäminen ja kuinka tehdä monen suhde yhteen -relaatio kahden eri taulun välille. Room todellakin helpottaa tietokantojen tekoa ja hallintaa. Erityisesti pakollisten SQL-koodien kirjoitus vähenee huomattavasti, joka oli yksi isoimmista Roomin eduista.

Sovellusten käyttäjäystävällisyys voidaan usein nähdä tarpeettomampana osana kuin se oikeasti on, eikä siihen välttämättä haluttaisi kuluttaa aikaa. Projektissa opittiin, että käyttäjäystävällisyys on erittäin tärkeä asia, jos sovelluksen haluaa todella menestyvän. Material Design ja yleiset käyttöliittymän ohjesäännöt sekä vaatimukset on tärkeä ottaa huomioon projektia suunnitellessa, sekä niille on varattava riittävästi aikaa.

Back end -ohjelmoinnin ja tietokantojen haastavuuden vuoksi käyttöliittymäohjelmointi jäi vähäiseksi sovelluksen ensimmäisessä versiossa vähäiseksi. Tämän vuoksi lopullista käyttöliittymää suunnitellaan Figma-sovelluksella poistaen ohjelmoinnin haasteet sen edestä, ja näin opitaan samalla enemmän käyttöliittymäsuunnittelusta.

Opinnäytetyö ja sovellus haasteista huolimatta on onnistunut suunnitelman mukaisesti. Sovelluksesta saatiin ensimmäinen versio valmiiksi ja se on jo tekijällä omassa käytössä kuntosalilla. Paljon tietoa saatiin Android-ohjelmoinnista, Android Studiosta, käyttöliittymäsuunnittelusta ja erityisesti Room-tietokannoista.

LÄHTEET

Babich, N. (2018) The Beginner's Guide to Android App Design. Haettu 30.1.2020 osoitteesta https://www.mockplus.com/blog/post/android-app-design

Bracey, K. (2018). What is Figma? Haettu 24.1.2020 osoitteesta https://webdesign.tutsplus.com/articles/what-is-figma--cms-32272

Chebbi, A. (2019) Choosing the best programming language for mobile app development. Haettu 29.1.2020 osoitteesta https://developer.ibm.com/articles/choosing-the-best-programming-language-for-mobile-app-development/

(Codelabs, 2020) Android Developer Fundamentals Course. Android fundamentals 10.1 Part A: Room, LiveData, and ViewModel. Haettu 3.3.2020 osoitteesta https://codelabs.developers.google.com/codelabs/android-training-livedata-viewmodel/index.html?index=..%2F..android-training#0

(Computer Hope, 2019) Database. Haettu 3.3.2020 osoitteesta https://www.computerhope.com/jargon/d/database.htm

Ditkowska, M. (n.d) What is Mobile Development? Haettu 30.1.2020 osoitteesta https://invotech.co/blog/what-is-mobile-development/

(Helsingin yliopisto, n.d). Käyttöjärjestelmä ja käyttöliittymä. Haettu 30.1.2020 osoitteesta https://blogs.helsinki.fi/opiskelijan-digitaidot/1-tietokoneen-kayton-perusteet/1-1-tietokoneen-toimintaperiaate/kayttojarjestelma-ja-kayttoliittyma/

Mullis, A. (2017) Android Studio tutorial for beginners. Haettu 31.1.2020 osoitteesta https://www.androidauthority.com/android-studio-tutorial-beginners-637572/

Muntenescu, F. (2017) 7 Steps To Room. Haettu 3.3.2020 osoitteesta https://medium.com/androiddevelopers/7-steps-to-room-27a5fe5f99b2

Nordeen, A. (2018) Why Java is the Best Programming Languange for Mobile Applications. Haettu 30.1.2020 osoitteesta https://irishtechnews.ie/why-java-is-the-best-programming-language-for-mobile-applications/

Patro, N. (2018) Choose the best — Native App vs Hybrid App. Haettu 30.1.2020 osoitteesta https://codeburst.io/native-app-or-hybrid-app-ca08e460df9

Smyth, N. (2019). *Android Studio 3.5 Development Essentials.* Payload Media, Inc.

(SQLite, n.d.). About SQLite. Haettu 3.3.2020 osoitteesta https://www.sqlite.org/about.html

(The Design Blitz n.d.). Characteristics of A Great User Interface Design. Haettu 28.1.2020 osoitteesta http://thedesignblitz.com/characteristics-of-a-great-user-interface-design/