Tampereen ammattikorkeakoulu

Androidin imperatiivisen ja deklaratiivisen käyttöliittymäsuunnittelun perusteet

Matias Yliluoma

OPINNÄYTETYÖ
Joulukuu 2023

Tietojenkäsittelyn tutkinto-ohjelma
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn tutkinto-ohjelma
Ohjelmistotuotanto

Yliluoma Matias:
Androidin imperatiivisen ja deklaratiivisen käyttöliittymäsuunnittelun perusteet

Opinnäytetyö 34 sivua, joista liitteitä 0 sivua
Joulukuu 2023

Opinnäytetyön tarkoituksena oli vertailla Androidin imperatiivista lähestymistapaa, toisin sanoen XML-pohjaista käyttöliittymän rakentamista uuteen, deklaratiiviseen tapaan eli Jetpack Composeen. Tässä työssä esitellään yleisiä käyttöliittymän rakentamiseen liittyviä asioita, kuten elementtien ja komponenttien rakentamista. Tässä työssä sivutaan myös XML:ään upottamisen Composella tehtyjä elementtejä sekä XML-näkymien upottamisen Compose-sovellukseen. Composen upottaminen XML:ään myös sivuaa tapaa kirjoittaa XML-pohjainen sovellus uudelleen Composella.

Androidin käyttöliittymän suunnitteluun tarkoitetut työkalut jakautuvat kahteen kategoriaan; imperatiivinen ja deklaratiivinen lähestymistapa. Imperatiivinen tapa on ollut olemassa koko Android-käyttöjärjestelmän ajan. Ongelmat käyttöliittymien suhteen alkoivat, kun sovellusten ja laitevalmistajien määrät kasvoivat voimakkaasti. Tämä tarkoitti sitä, että samojen sovellusten piti sopia erilaisille laitteille. Suurimmaksi ongelmaksi kehkeytyivät laitteiden näyttöjen kokoerot. Ratkaisu ongelmaan oli Googlen kehittämä Jetpack Compose, joka noudattaa deklaratiivista lähestymistapaa käyttöliittymien suunnittelussa.

Opinnäytetyön tuloksena havaittiin, että deklaratiivinen lähestymistapa on huomattavasti nopeampi ja yksinkertaisempi, kun luodaan käyttöliittymää ja rakennetaan elementtejä. Elementtien ulkoasun muokkaaminen on myös paljon nopeampaa sekä yksinkertaisempaa verrattuna imperatiiviseen lähestymistapaan.

Asiasanat: xml, jetpack compose, imperatiivinen, deklaratiivinen, käyttöliittymä

ABSTRACT

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

Matias Yliluoma:
The Basics of Android Imperative and Declarative UI Design

Bachelor's 34 pages, appendices 0 pages
December 2023

The purpose of this thesis was to compare the imperative approach and the declarative approach in the Android operating system. This thesis focuses on the basics of both approaches such as building elements. It also covers embedding Compose elements into XML, as well as incorporating XML views into a Compose application. The integration of Compose into XML-based applications is discussed, touching upon the approach of rewriting an XML-based application using Compose.

The UI-toolkits of Android are divided into two separate categories. The first one is the imperative approach and the second one is the declarative approach. Imperative approach was the first one to be used on Android development. Problems begun to appear when the number of platforms and devices started to increase. This meant that all applications had to be adapted on all devices. The biggest challenge in this was the differences on screen sizes between the devices and manufacturers. One of the solutions was Jetpack Compose which was designed by Google. Jetpack Compose follows the declarative approach.

As the result, declarative approach is much faster while creating user interface and building elements. Editing the layouts of the elements is also faster than with imperative approach.

Key words: xml, jetpack compose, imperative, declarative, user interface

SISÄLLYS

1 JOHDANTO 7
2 NATIIVI MOBIILIKEHITYS 8
2.1 Android kehitys yleisesti 8
2.1.1 Elämänkaari 8
2.1.2 Elämänkaaren metodit 8
2.2 XML-pohjaisen UI:n kirjoittaminen 10
2.2.1 Layout container 11
2.2.2 Komponentit XML:ssä 13
2.2.3 Komponentin ulkoinen muokkaus XML:llä 16
3 JETPACK COMPOSE 19
3.1 Compose funktiot 20
3.1.1 Compose UI:n päivitys recompositionilla 20
3.1.2 Compose funktion UI sekä toiminnallisuus 21
3.1.3 Box, Row, Column ja ConstraintLayout 21
3.1.4 Perus UI-elementit Composessa ja vertailua XML:n tapaan 22
3.2 Sivuvaikutusten käsittely 27
3.2.1 SideEffect 27
3.2.2 LaunchedEffect 27
3.2.3 DisposableEffect 28
4 INTEROPERAATIBILITEETTI COMPOSEEN JA XML:ÄÄN 29
4.1 ComposeView XML:ään ja XML-näkymä Composeen 29
4.1.1 Interoperaabiliteetin toteutus 29
4.1.2 Interoperaabiliteetin tulos 30
4.2 Migraatiotaktiikka 31
4.2.1 Migraation toteutus 31
4.2.2 Migraation lopputulos 31
5 POHDINTA 33
LÄHTEET 35

LYHENTEET JA TERMIT

XML	Extensible Markup Language
UI	user interface/käyttöliittymä
HTML	Hyper Text Markup Language
iOS	iPhone Operating System
DP/DIP	Density-independent pixels
API	Application Programming Interface/sovellusohjelmointirajapinta
ID	Resource identity/resurssin identiteetti

1 JOHDANTO

Kaksi suosituinta älypuhelin alustaa on Applen iOS alusta sekä Android puhelimet. Tämä opinnäytetyö kattaa Android sovellusten käyttöliittymän kaksi työkalua, millä niitä nykypäivänä toteutetaan. (Counterpoint, 2023)

Aihe on valittu yleisen mielenkiinnon vuoksi mobiilikehittämistä kohtaan, sekä vähäisen kokemuksen käyttöliittymien suunnittelujen takia. Tässä opinnäytetyössä käydään läpi XML-näkymä pohjainen käyttöliittymän rakentamisen perusteet sekä Jetpack Composella rakennetun käyttöliittymän perusteet.

Tarkoituksena on ottaa esiin molempien työkalujen perusteet sekä tehdä vertailua työkalujen välillä nopeuden, tehokkuuden ja yksinkertaisuuden kannalta. Tarkoituksena on myös ottaa esiin molempien ratkaisujen vahvuuksia ja heikkouksia.

Mielestäni käyttöliittymien suunnittelua pitäisi olla enemmän opetuksessa, niin kielessä kun kielessä. Tästä syystä halusin käydä läpi kaksi toisestaan poikkeavaa työkalua, jolla suunnitellaan täysin samaa ratkaisua. Künnethin (2022) mukaan hyvin rakennettu käyttöliittymä vankan tiedon pohjalta vaikuttaa isosti sovellusten toimivuuteen ja esimerkiksi laitteen akun syöntiin.

2 NATIIVI MOBIILIKEHITYS

Android ja iOS mobiilikehitys ovat natiivia mobiilikehitystä. Natiivi mobiilikehitys tarkoittaa sovellusten tekemistä juuri kyseisille laitteille. Tämä tarkoittaa sitä, että Androidille tehty sovellus itsessään ei toimi Applen iOS alustalla ja toisinpäin. Tämä opinnäytetyö keskittyy Androidin natiiviin mobiilikehityksen käyttöliittymän rakentamiseen. (Koffer 2023)

2.1 Android kehitys yleisesti

Android sovelluksia tehdään pääsäännöllisesti Javalla tai Kotlinilla. Android sovelluksia on mahdollista myös ohjelmoida C#, C++ ja Python ohjelmointikielillä erilaisia käyttötarkoituksia varten. Nykyisesti suositumpi ja modernimpi vaihtoehto on tähän Kotlin. Monet vanhemmat sovellukset on kirjoitettu Javalla ja niitä kehitetään edelleen Javalla. Kotlin on kieli, joka toimii Java-virtuaalikoneessa. Kotlinin edut ovat lyhyempi ja selkeämpi syntaksi, mikä tekee monien mielestä kielestä ketterämmän. (Codemotion, 2023)

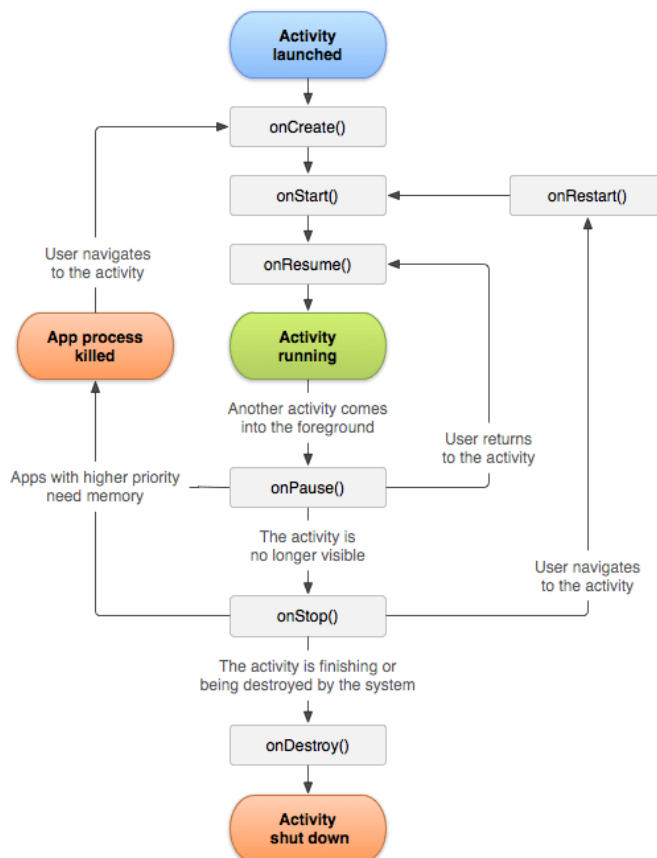
2.1.1 Elämänkaari

Android kehityksessä elämänkaari on olennainen asia Android-sovellusta kehitettäessä. Tämän ymmärtäminen auttaa suuresti välttämään sovelluksen kaatumisia sekä muunlaisia mahdollisia bugeja. Tämä myös mahdollistaa, että voi turvallisesti vaihtaa eri sovellukseen kesken tietyn toiminnon ilman, että menetämme tätä tehtyä työtä. (Gitau 2021)

2.1.2 Elämänkaaren metodit

Elämänkaarella on seuraavat metodit: onCreate, onStart, onResume, onPause, onStop, onRestart ja onDestroy. OnCreate metodia kutsutaan sovellusta käynnistäessä ja myös mitä tahansa activitya käynnistäessä tai vaihdessa.

OnCreate on kaikissa Android sovelluksissa ilman poikkeuksia, ilman sitä sovellus ei käynnisty ja se ei luo mitään näkymään. Activitya kutsutaan näkymäksi, jos esimerkiksi vaihdat Whatsappissa yhteistietoluettelosta viestin kirjoittamiseen yhteyshenkilölle, tässä kohtaa vaihdoit näkymää. Toisin sanoen, käynnistämällä sovelluksen elämänkaari alkaa. Kuviossa 1 kuvastettuna elämänkaaren funktioiden kierto. (Gitau 2021)



KUVIO 1. Elämänkaari. (Gitau 2021)

On tärkeää ymmärtää koko elämänkaari, vaikka sovellusta kirjoittaessa ei välttämättä tarvita muita metodeja, kun onCreate. Opiskellessa Android sovellusten tekemistä usein kirjoitetaan kaikki elämänkaaren metodit ylös ja kirjoitetaan niihin logiteksti. ”Logituksella” pystytään näkemään, milloin näitä metodeja kutsutaan missäkin sovelluksen elämänkaaren vaiheessa. Tämä tuo laajemman ymmärryksen sovelluksen käyttäytymisestä ja mahdollisista bugeista tai sovelluksen kaatumisesta. (Gitau 2021)

OnStart metodi kutsutaan aina kun sovelluksessa vaihtuu näkymä ja tämä voi tapahtua monta kertaa sovelluksen elämänkaaren aikana. OnStop metodia

kutsutaan myös, kun näkymä vaihtuu. Silloin tiettyjä määriteltyjä resursseja vapautetaan. OnStartilla voidaan taas alustaa uudestaan vapautettuja resursseja. (Gitau 2021)

OnResume metodi seuraa OnStart metodia. OnResume metodi kutsutaan, kun käyttäjä palaa takaisin näkymään, joka oli aiemmin piilotettu tai oli taustalla. Jos sovellus on taustalla, niin silloin elämänkaaren mukaan on kutsuttu onPause metodia tai onStop. OnPause metodissa myös vapautetaan resursseja laitteen muistin hallinnan vuoksi, milloin onResume myös voi taas alustaa ne uudelleen. OnStop metodin kutsun aikana sovelluksen melkein kaikki toiminta on tuhottu, vaikka itse sovellus on vielä päällä. OnStop metodista voi vielä siirtyä onRestart metodiin, joka sanansa mukaan käynnistää sovelluksen tai alustaa sen alusta, ilman että sovelluksen elämänkaaren aikana sitä ei ole vielä tuhottu. OnDestroy metodia kutsutaan, kun sovellus halutaan sammuttaa. (Gitau 2021)

2.2 XML-pohjaisen UI:n kirjoittaminen

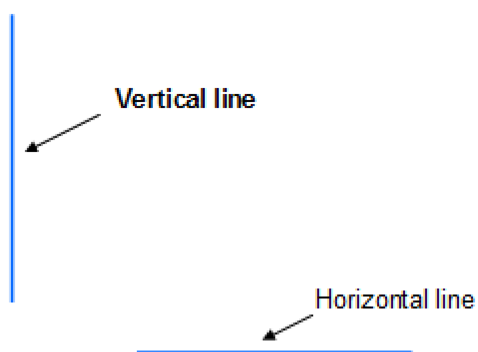
Aloitetaan ensiksi ymmärtäminen laajemmin perinteistä XML-pohjaista koodausta. XML tulee sanoista Extensible Markup Language, joka kehitettiin alkujaan päihittämään HTML-ohjelmointikielen puutteet. XML on HTML:n tapaan merkintäpohjainen kieli. (Sivakumar 2016)

Ennen Composen tuloa XML-pohjainen oli tapa, jolla Android käyttöliittymän kehitystä tehtiin. Siinä missä Composea markkinoidaan kätevänä, aikaa säästävänä tapana, XML on edelleen monimutkaisimpien ratkaisujen parempi vaihtoehto. (Jackson 2014, 2.)

2.2.1 Layout container

Layout container, tai suomeksi käännettynä ”asettelusäiliö”, on XML-pohjaisen käyttöliittymän tekemisen peruslähtökohta. Näiden säiliöiden sisälle rakennamme sen näkymän, määrittelemme kaikki erilaiset napit, tekstikentät, värit ja vastaavat. Toisin sanoen, asettelusäiliö määritellään aivan ensimmäiseksi, kun lähdemme tekemään käyttöliittymää. XML-pohjaisessa tekemisessä käytetään schema-nimistä Google Androidin tietovarastoa (repository). Tietovarastosta haetaan Androidin omia XML-elementtejä ja ominaisuuksia. (Jackson 2014, 2.)

Opinnäytetyössä käytetään esimerkkinä Linearlayout-säiliötä. Linearlayout on kaikista mahdollisista säiliöistä yksinkertaisin. Se on julkinen luokka, joka laajentaa ViewGroup-luokkaa. Linearlayout antaa nimensä mukaan asetella kaikki elementit vertikaalisesti tai horisontaalisti, kuviossa 2 suunnat kuvastettuna. Tämä suunta on pakko määritellä Linearlayouttia käytättäessä ja sen määritelmänä toimii ”android:orientation” -attribuutti. (Jackson 2014, 12.)



KUVIO 2. Vertikaali ja horisontaali. (Math Open Reference)

Seuraavat pakolliset attribuutit elementtejä määritellessä ovat layout_width ja layout_height. Näille on olemassa 3 erilaista vakioarvoa. Arvot ovat match_parent, wrap_content ja fill_content. Fill_content toimii ainoastaan vanhoissa sovelluksissa ja Android käyttöjärjestelmissä, tämä korvattiin

match_parent arvolla. Tämän arvon korvaus tapahtui API Level 8:ssa. Nämä arvot voidaan korvata dp-mittayksiköllä. (Jackson 2014, 2)

Match_parentilla voi täyttää koko ruudun, vaikka yhdellä painonapilla, kun annat tämän arvon korkeudelle ja leveydelle. Wrap_content taas kutistuu tai laajenee tämän sisällön mukaan ja on hyödyllinen halutessasi komponentin sopeutumaan dynaamisesti. (Jackson 2014, 2)

Muita Androidissa käytettäviä asettelusäiliöitä ovat constraint layout,
absolute layout, frame layout, relative layout ja table layout. Kaikki ovat tärkeitä työkaluja tehdessä toimivaa käyttöliittymää. Yhden säiliön sisälle voi laittaa toisen säiliön. Nämä muut säiliöt ovat yleisesti enemmän joustavia, kun linear layout.

Constraint layout on säiliö, joka antaa mahdollisuuden asetella ja muuttaa komponenttien kokoa joustavasti. Tämän säiliön tarkoituksena on parantaa sovellusten suoriutumista, eli vähentää säiliön sisällä käytettäviä toisia säiliötä, mikä tekee sovelluksesta raskaamman. Tällaisella tapaa pystytään helpommin ylläpitämään sovelluksen käyttöliittymää. Sisäisten säiliöiden väheneminen tekee myös XML:stä helpommin luettavaa. Tällä säiliöllä on myös olemassa kahvat, joilla pystytään määrittelemään esimerkiksi komponenttien etäisyyksiä toisistaan tai näytön reunoista. (Shah & Chettri 2020)

Absolute layout on säiliö, joka antaa mahdollisuuden tarkasti määrittää jokaisen näkymäkomponentin paikan. Tämä säiliö on poistettu käytöstä sen vaikean käytön vuoksi; jokainen komponentti laitetaan x ja y -koordinaattien mukaan ja näyttöjen eri koot tekevät tästä hyvin vaikean. (Shah & Chettri 2020)

Frame layout säiliö varaa tilan näytöltä, näyttääkseen yhden näkymän. Jos lisäät näkymäkomponentteja ne menevät päällekkäin, viimeiseksi aseteltu aina ylimmäiseksi. Voit kuitenkin asettaa niitä eri paikkoihin käyttämällä näkymäkomponentien attribuutteja, esimerkiksi layout_gravity, layout_marginTop tai layout_marginBottom. Frame layout on kätevä tapa asettaa esimerkiksi taustakuva sovelluksen näkymään. (Shah & Chettri 2020)

Relative layout mahdollistaa hyvin joustavan käyttöliittymän esittelyn esimerkiksi kuvien, animaatioiden tai monien eri näkymäkomponenttien kanssa, eli lapsinäkymien kanssa. Relative layout on käytännöllisyytensä vuoksi Linear layoutin jälkeen suosituin asettelusäiliö. (Shah & Chettri 2020)

Table layout -säiliöllä tehdään näkymäkomponentti rivejä ja jonoja. Jokainen jono voi sisältää nolla tai enemmän solua. Jokainen solu voi sisältää erilaisia näkymäkomponentteja, esimerkiksi tekstinäkymän, kuvan tai painonapin. Tämä malli on hyödyllinen, kun käyttöliittymästä suunnitellaan taulukkomaista rakennetta. Table Layout attribuutit myös antavat mahdollisuuden määrittää rivi- ja sarakemarginaalit sekä painorajoitukset. Painorajoitukset määrittää paljonko solu vie tilaa vaakasuunnassa. (Shah & Chettri 2020)

2.2.2 Komponentit XML:ssä

XML on tag-pohjainen kieli. Tämä tekee XML:stä suhteellisen helposti luettavan kielen, mutta myös työlään kirjoittaa. Kaikki komponentit vaativat edellisen otsikon alla mainitut attribuutit, sen lisäksi attribuuttien valjastaminen helpommin sovellukseen näkyväksi suositellaan käyttämään identifier tapaa, eli android:id -attribuuttia. Jokaisen id:n täytyy olla uniikki ja se voidaan nimetä vaikka ”painonappi1” tai miksikä vain. Selkeyden vuoksi on hyvä noudattaa ohjelmoinnin hyviä toimintatapoja eli nimetä ne selkeästi ja kuvaavaksi, mitä kukin komponentti myös tekee. Komponentteja kutsutaan Androidin terminologiassa widgeteiksi, mutta tässä työssä käytetään komponentti sanaa. (Jackson 2014. 2.)

Komponentteja on erilasia, kuten TextView, EditText, Button, ImageView, ImageButton, Checkbox, RadioButton, RadioGroup, ListView ja AutoCompleteTextView. (Shah & Chettri 2020)

TextView on pelkän tekstin esittelyä varten sovelluksen näytöllä. Tekstin kokoa, väriä, sijaintia ja haluaako tekstin esimerkiksi näkymän keskelle, pystyy muokkaamaan arttribuuteilla. (Shah & Chettri 2020)

EditText näkymä on kirjoitusnäkymä, eli tässä näkymässä on tarkoitus kirjoittaa. Näkymällä on erilaisia attribuutteja, millä pystyy edellisten tapaa muuttamaan sijaintia ja kokoa ja niin edelleen, mutta nämä omat attribuutit ovat esimerkiksi muuttaa kenttään kirjoitettavat asiat numeroiksi, tai salasanan kaltaisena ”******” merkistönä. (Shah & Chettri 2020)

ImageView näkymällä pystyy näyttämään kuvan sovelluksessa. Kuva otetaan käyttöön laittamalla se Android projektin drawable -kansioon, sekä määrittelemällä Imagen attribuuttiin ”android:src”, mikä tässä viittaa sanaan lähde. Arvona toimii ”@drawable/kuvan_nimi”. (Shah & Chettri 2020)

ImageButton näkymä on nappi, joka näkyy kuvana. Kuva haetaan samalla tapaa kuin ImageView:ssä. (Shah & Chettri 2020)

Checkbox näkymä on myös nappi, joka on perinteinen, valitse haluamasi tai tarvittava määrä esimerkiksi poistoa varten listalta. Valintalaatikossa voi aina painaa uudestaan, jos ei halua tätä valita. (Shah & Chettri 2020)

RadioButton näkymä on taas yksittäistä valintaa varten monista eri vaihtoehdoista, esimerkiksi tehdessä profiilia valitaan yksi sukupuoli ja niin edelleen. Tämän uniikki attribuutti on ”android:checked”. (Shah & Chettri 2020)

RadioGroup näkymä on samanlainen, kun RadioButton näkymä. Eroavaisuutena on automaattinen ”uncheck” jos valitset toisen valinnan, jotka kuuluvat samaan valintaryhmään. (Shah & Chettri 2020)

ListView on vieritettävä lista, eikä tätä vieritys toiminnallisuutta tarvitse erikseen määrittää. Tämä näkymä käyttää adapteri luokkia. ListView ottaa sisällön erilaisista tietoa sisältävästä asiasta esimerkiksi tietokannasta. (Shah & Chettri 2020)

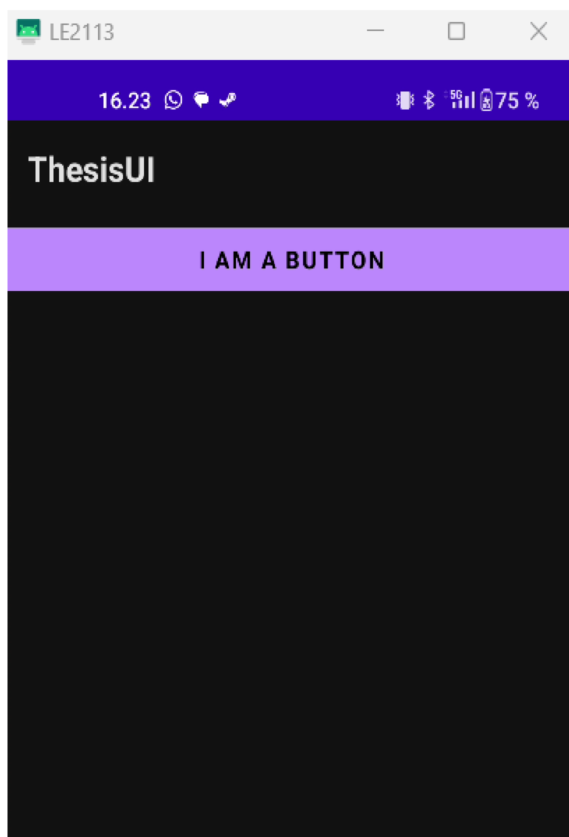
AutoCompleteTextView on EditText näkymän alaluokka. Tämä on siis tekstikenttä, joka automaattisesti ehdottaa kirjoituksen perusteella erilaisia mahdollisia sanavaihtoehtoja. Vaihtoehdot tulevat pudotusvalintana. (Shah & Chettri 2020)

Kuvassa 1 on luotu Button näkymä LinearLayout säiliöön, jonka tarkoituksena on hoitaa ”click handling” eli mitä tapahtuu, kun painonappia on painettu. Edellä mainittu ImageButton:in tarkoitus on tehdä sama asia. (Shah & Chettri 2020)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">


    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="I am a button" />

</LinearLayout>
```

KUVA 1. activity_main.xml

Kuvan 2 painonappi on luotu vain määrittelemällä XML-tiedostoon tarpeelliset tiedot. Text-attribuutilla saa määriteltyä tekstin, jos haluaa painonapin sisältävän tekstin. Tässä kohtaa ei olisi tarvetta määrittää painonapille edes id:tä, koska tätä tarvitaan vasta silloin kun, lähdemme toteuttamaan painonapille jotain toimintaa, mitä tapahtuu, kun nappia painetaan. Ei siis vielä ole tarvetta kirjoittaa koodia Kotlin kielellä. Nappi piirtyy sovellukseen joka tapauksessa, koska MainActivity.kt -tiedoston MainActivity -luokassa on aiemmin mainittu onCreate funktio, joka sisältää setContentView funktion, missä setContentView kutsuu activity_main.xml tiedostoa. Tässä kyseisessä tiedostossa on määritelty tämä nappi, sekä Linearlayout.



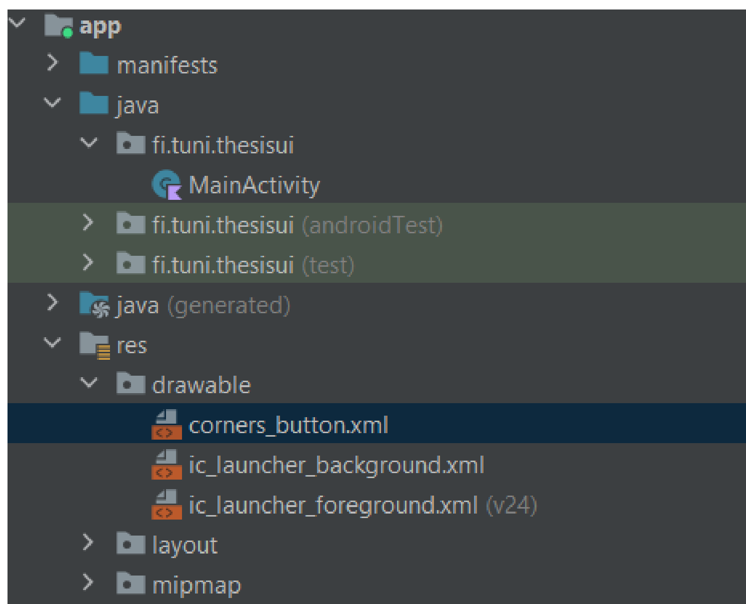
KUVA 2. Näyttökuva laitteen näkymästä.

2.2.3 Komponentin ulkoinen muokkaus XML:llä

XML-pohjaisella tavalla komponenttien ulkoinen muokkaus toimii erillisen xml-tiedoston kautta, joka työnnetään Android projektissa olevaan drawable -kansioon. Tähän erilliseen tiedostoon on tarkoitus määrittää millaiseksi haluamme napin käyttäen erilaisia attribuutteja ja arvoja. (Chugh 2022)

Drawable on resurssi, joka on pääasiallinen konsepti ruudulle piirrettävälle grafiikalle. Käytännössä tätä tekniikkaa käytetään XML:ssä, kun halutaan viilata yksittäisiä elementtejä paremman näköiseksi pelkkien palikoiden tai suorakulmien sijaan. Asettelusäiliössä sijaitsevaan komponenttiin liitetään tämä uusi XML-tiedosto arvoineen haluttuun komponenttiin komponentin, ”android:background” attribuuttiin arvoksi, eli XML-tiedoston nimi on tämän arvo. (Codepath)

Kuvassa 3 drawable kansioon on luotu uusi XML-tiedosto, komponentin ulkonäköä pystytään muokkaamaan. Muokataan painonapin kulmat pyöreämmäksi.



KUVA 3. Kuva projektijuuresta.

Kuvassa 4 annetaan shape ominaisuus corners_button tiedostoon ja määrittelemme corners merkinnän, joka ottaa ”android:radius” atttribuutin. Attribuutti ottaa vastaan numeraalisen arvon dp-mittayksiköllä. Muita yleisiä tunnisteita mitä shape merkintä ottaa on gradient, stroke ja solid. Näillä määritellään värejä sekä spesifejä kokoja komponentin ulkoasulle esimerkiksi ulkoreunat, jos niiden värejä haluaa moniulkoistaa. (Codepath; Chugh 2022)

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <corners android:radius="25dp"/>
</shape>
```

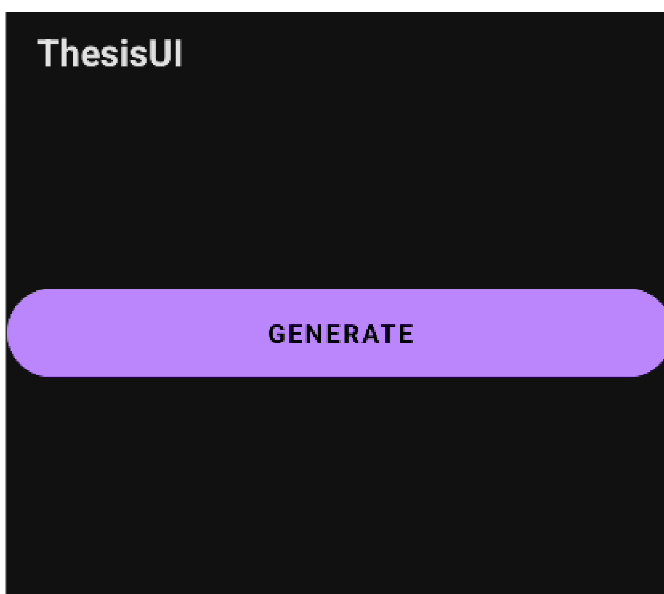
KUVA 4. Corners_button.xml painonapin muokkaus.

Viimeisenä annamme kuvassa 5 ”androind:background” attribuutille arvon “corners_button”, mikä on uuden XML-tiedoston nimi.

```
<Button
    android:id="@+id/button1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Generate"
    android:layout_gravity="center"
    android:layout_marginTop="100dip"
    android:background="@drawable/corners_button"
    />
```

KUVA 5. Kuva activity_main.xml tiedostosta.

Näemme kuvassa 6 painonapin reunojen pyöristyneen halutulla tavalla.



KUVA 6. Kuva lopputuloksesta käyttöliittymästä.

3 JETPACK COMPOSE

Android laitteiden määrän kasvamassa erittäin suureksi eri valmistajien mallien vuoksi sekä uusien käyttöjärjestelmä versioiden vuoksi ja sovellusten mennessä monimutkaisimmiksi, sovellukset täyttyivät ”boilerplate” koodilla, eli pohjakoodilla, joka on rutiinin omaista koodin toistoa ilman merkittäviä muutoksia sovellukseen. Tämä pohjakoodi alkoi näkymään normaaleissa, perustoiminnoissa esimerkiksi listojen vierityksissä. Tätä tehtiin esimerkiksi XML-pohjaisella UI-tekniikalla. (Künneth 2022. 1.)

Nämä ongelmat eivät olleet ainoastaan Androidin ongelmia. Muissa alustoissa ja käyttöjärjestelmissä UI-työkalut seurasivat myös imperatiivista lähestymistapaa, mikä tekee sovellusten kasvaessa isoiksi ja monimutkaisiksi hyvin jäykäksi tavaksi. Imperatiivisessa suunnittelussa käytännössä pitää muokata koko komponentti puu ja niiden attribuutit, kun lähdetään vaihtamaan käyttöliittymän ulkomuotoa. Ensimmäisenä deklaratiivisen suunnittelun otti käyttöön React, mikä on web kehys. Apple seurasi ja otti käyttöön deklaratiivisen suunnittelun toteuttamalla SwiftUI-tekniikan. (Künneth 2022. 1.)

Jetpack Compose on Googlen deklaratiivinen käyttöliittymä kehys Android kehitykseen, joka yksinkertaistaa huomattavasti käyttöliittymän tekemistä. Vaikka Android kehitystä voidaan tehdä myös Javalla, Composea voidaan tuottaa ainoastaan Kotlinilla. (Künneth 2022. 1.)

Compose työkalu on kohtalaisen uusi Androidin käyttöliittymän tekemiseen tarkoitettu työkalu, jonka ensimmäinen versio julkaistiin 28. heinäkuuta 2021. Androidin UI-työkaluna on käytetty XML-pohjaa. Compose enemmänkin yhdistää nämä tyylit ja sitä pystyy kirjoittamaan suoraan Kotlin koodiin. Tämän vuoksi Compose tukee enemmän pohjataitojamme. (Zhang 2022)

3.1 Compose funktiot

Imperatiivisessa menetelmässä käytetään luokkia, kuten View -luokkaa sekä ViewGroup -luokkaa, mikä toisin tarkoittaa sitä, että komponentit perivät View ja ViewGroup luokan kaikki muutkin ominaisuudet, vaikka niitä ei tarvittaisi. Tällä tapaa ei siis ole mahdollista yhdistää komponentteja erikoistuneemmaksi käyttöliittymäelementiksi. (Künneth 2022. 2.)

Composessa luokkien käytön sijaan käytetään komposiitiofunktioita.
Käyttääkseen Composea funktioissa, täytyy kirjoittaa ”@Composeable” annotaatio funktion yläpuolelle. Tämä antaa tiedon Compose-compilerille, että funktio muuttaa dataa käyttöliittymäelementeiksi. Näissä funktioissa pystytään suorilta luomaan näkymään komponentit, ilman tarvetta kirjoittaa XML-tiedostoon. (Künneth 2022. 2.)

Aikaisemmin mainitsin setContentView-funktiosta, joka sai parametriksi viitteen activity_main.xml tiedostoon. Composessa tämä on korvattu setContent-funktiolla, joka ei tarvitse viitettä mihinkään. SetContent funktion sisällä kutsumme Compose -funktioita. Tämä siis esittää tai piirtää ja toimii muuten samalla ajatusmallilla, kun setContentView. (Shah 2022)

3.1.1 Compose UI:n päivitys recompositionilla

Compose käyttöliittymän näkymän päivittäminen tapahtuu recompositionilla. Se toimii automaattisesti, kun composeable funktio, joka on liitoksissa käyttöliittymään, saa tiedon muutoksesta. Muutokset voivat olla tiloja, jotka vaihtuvat aikanaan tai esimerkiksi käyttäjän kirjoitettu teksti. Tilat voidaan määrittää mutableStateOf -funktiolla. Käyttäessään tätä tilan määritystä, tämä tila täytyy muistaa käyttäen remember -funktiota. Remember -funktio auttaa tilan säilyttämisessä, eikä se esimerkiksi nollaannu käyttöliittymän päivittyessä recompositionilla. (Künneth 2022. 5.)

3.1.2 Compose funktion UI sekä toiminnallisuus

Koska Compose-funktiot kirjoitetaan suoraan Kotlin tiedostoon voidaan funktiossa suorittaa suoraan näkymään luodun painonapin alle myös toiminnallisuus. Tämä eroaa XML-pohjaisesta, missä napin ulkoasu ja näkyvyys luotiin XML-tiedostossa ja tehdä erikseen viittaus Kotlinilla tähän XML-tiedostoon sen uniikilla ID:llä, mikä määriteltiin erillisellä attribuutilla.

Kun luodaan toiminnallisuutta uudelleen käännettävään funktioon, pitää olla varovainen sivuvaikutusten suhteen. Vaikka painonappi on yksinkertaista luoda Compose tai composeable-funktioissa, napin varsinainen toiminnallisuus kannattaa luoda aina erillisessä funktiossa, sekä hyödyntää Jetpack Compose -kirjaston sivuvaikutusfunktioita. (Android for Developers; Künneth 2022. 9.; Morteza 2023)

3.1.3 Box, Row, Column ja ConstraintLayout

Box-funktio asettaa haluttuja UI-elementtejä päällekkäin. Samalla tapaa kuin XML:n FrameLayout:ssa, sillä varataan itselleen tilaa, jolloin voidaan laittaa näitä elementtejä päällekkäin. Viimeisenä määritelty elementti tulee olemaan päällimmäisenä. (Künneth 2022. 4.)

Row-funktiolla saat elementit näkymään vertikaalisti ja Column-funktiolla horisontaalisti. Nämä kaksi muistuttavat LinearLayout:a XML:stä, missä pystyit valitsemaan joko horisontaalisen tai vertikaalisen suunnan. Samalla tapaa kuin XML:ssä, Composella voi myös tehdä sisäkkäisiä UI-funktioita määritelläkseen esimerkiksi elementeille eri suuntia, miten asettuvat käyttöliittymään. (Künneth 2022. 4.)

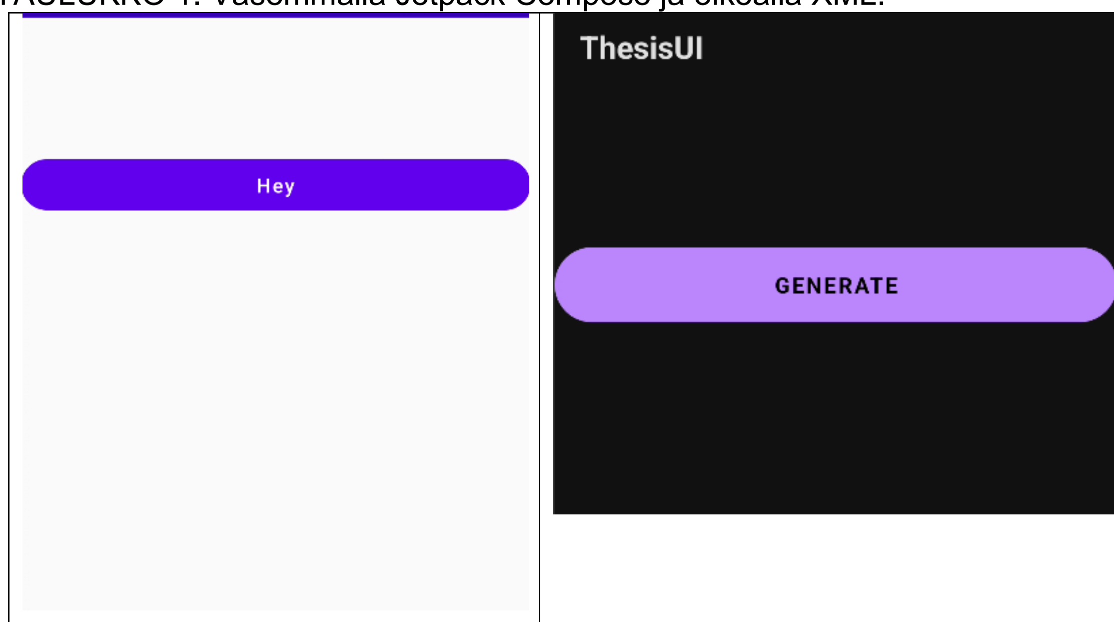
ConstraintLayout-funktiolla pystyt vähentämään näitä UI-funktioiden käyttämistä sisäkkäin. Käyttääksesi tätä funktiota, build.gradle -tiedostoon on pakko lisätä riippuvaisuus ”implementation ”androidx.constraintlayout:constraintlayout-compose:1.0.0-rc02”. Samalla käsityksellä, mitä oli XML-pohjaisessa

ConstraintLayout:ssa, myös Composessa käytetyssä funktiossa on olemassa ankkurit, mitä kautta tällä funktiolla käyttöliittymän elementtien ja komponenttien asettelu tapahtuu. Composessa linkitys tapahtuu ankkureilla, mutta elementit tarvitsevat referenssin voidakseen linkittyä. Tämä linkitys tapahtuu createRefs-funktiolla. (Künneth 2022. 4.)

3.1.4 Perus UI-elementit Composessa ja vertailua XML:n tapaan

Jetpack Composessa rakennuspalikoiden luonti ja muokkaus verrattuna XML-pohjaiseen on yksinkertaisempaa. Kaiken pystyy tekemään suoraan Kotlin tiedostoon, myös palikoiden ulkoiset muokkaukset modifier luokalla. Modifier luokkaa käyttäen on mahdollista tehdä hyvin pitkälle samat muokkaukset kuin XML:ssä ja yksinkertaisemmin. Taulukossa 1 on Composella ja XML:llä luotu täysin samat painonapit. (Künneth 2022. 1.; Künneth 2022. 3.)

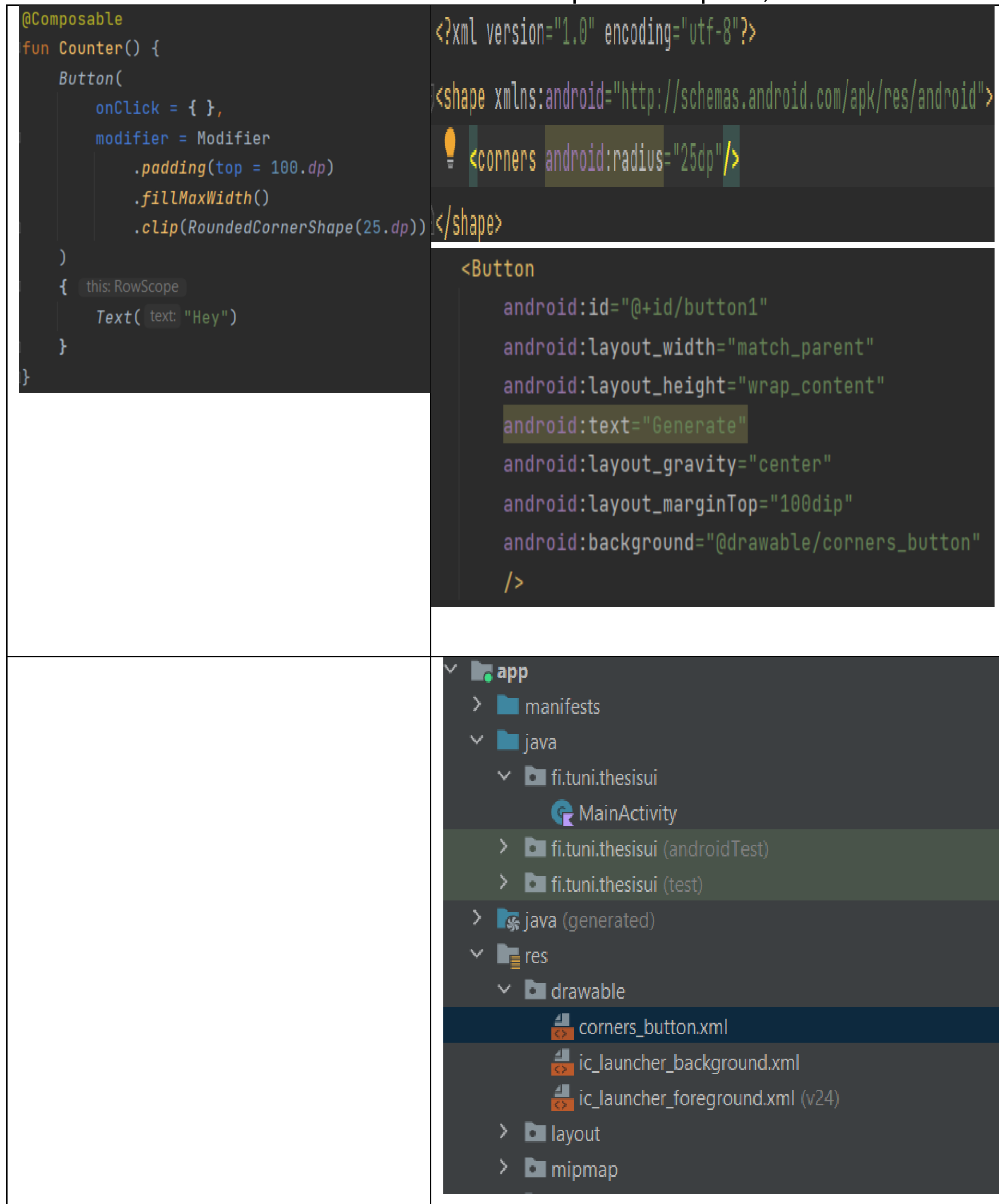
TAULUKKO 1. Vasemmalla Jetpack Compose ja oikealla XML.



Taulukossa 2 näemme napin luonnin eroavaisuudet Composella ja XML:llä. XML:n viimeinen sarake näyttää corners_button.xml tiedoston luomisen drawable-kansioon. Rakennuspalikan luonti ja muokkaaminen Composessa vaati yhteensä 13 riviä koodia. Tähän on mukaan laskettu funktion luonti ja onClick tapahtumankäsittelijä, joka määrittää sen mitä painonapilla tehdään. XML vaati 13 riviä koodia myös. XML:ssä jouduit tekemään uuden tiedoston eri kansioon, antamaan viitteen background attribuutille tästä tiedostosta. XML:n

tapauksessa ei ole tapahtumankäsittelijää painonapille luotu ollenkaan, se luodaan erikseen Kotlin tiedostossa.

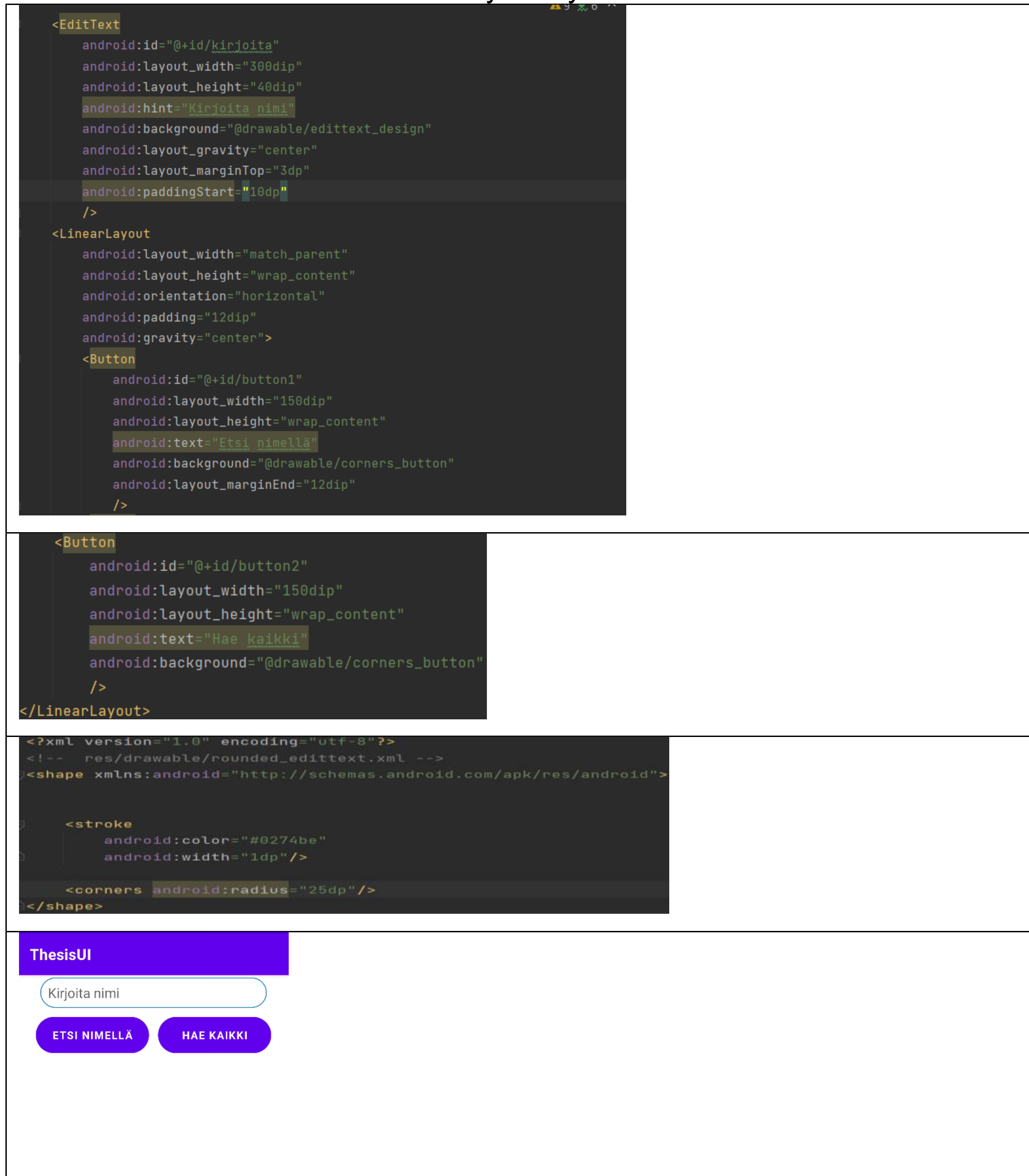
TAULUKKO 2. Vasemmassa sarakkeessa Jetpack Compose, oikeassa XML.



Taulukossa 3 on minimaalinen perus käyttöliittymä XML:llä rakennettuna. Toteutuksessa hyödynnetään sisäkkäistä asettelusäiliön käyttöä. XML:ssä on omat attribuuttinsa ja tässä toteutuksessa on käytetty layout_gravity ja gravity attribuutteja asetellakseen elementit keskelle. EditText:n attribuutti paddingStart antaa käyttäjän määrittää, mistä kohtaa tekstikenttää kirjoitus alkaa dp-mittayksiköllä. Tekstikentän ulkonäkö kustomoitiin tekemällä edittext_design.xml -tiedosto ja asettamalla tämä tekstikentän arvoksi background attribuuttiin. Stroke merkinnän sisällä on määritetty tekstikentän rajan leveydeksi 1 ja väriksi #0274b, että pyöristetyt kulmat nähdään. Koska tekstikentän paikanpitäjä viittaa nimen

antamiseen, tähän kenttään voi kirjoittaa ainoastaan kirjaimia ja se on määritelty inputType attribuutilla.

TAULUKKO 3. XML:llä suunniteltu käyttöliittymä



```
<EditText
    android:id="@+id/kirjoita"
    android:layout_width="300dip"
    android:layout_height="40dip"
    android:hint="Kirjoita nimi"
    android:background="@drawable/edittext_design"
    android:layout_gravity="center"
    android:layout_marginTop="3dp"
    android:paddingStart="10dp"
    />
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:padding="12dip"
    android:gravity="center">
    <Button
        android:id="@+id/button1"
        android:layout_width="150dip"
        android:layout_height="wrap_content"
        android:text="Etsi nimellä"
        android:background="@drawable/corners_button"
        android:layout_marginEnd="12dip"
        />
```

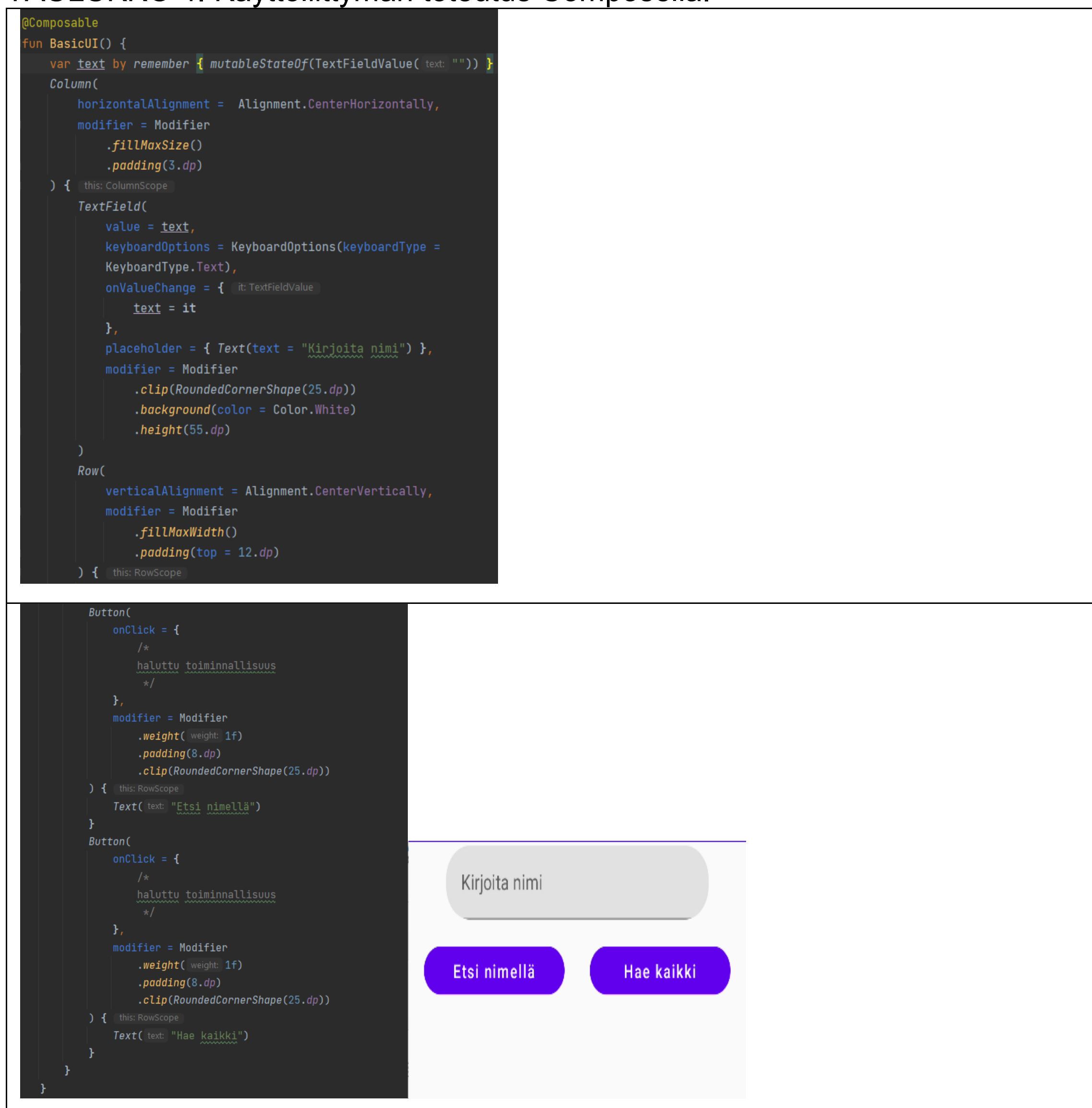
```
    <Button
        android:id="@+id/button2"
        android:layout_width="150dip"
        android:layout_height="wrap_content"
        android:text="Hae kaikki"
        android:background="@drawable/corners_button"
        />
</LinearLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>
<!-- res/drawable/rounded_edittext.xml -->
<shape xmlns:android="http://schemas.android.com/apk/res/android">

    <stroke
        android:color="#0274be"
        android:width="1dp"/>

    <corners android:radius="25dp"/>
</shape>
```


Composen vastaavassa toteutuksessa on hyödynnetty myös sisäkkäistä asettelua Column ja Row funktioita hyödyntäen, jotka ajavat tässä tapauksessa saman asian, kun XML-toteutuksen sisäkkäiset LinearLayout:it. Compose toteutuksessa on käytetty horizontalAlignment ja verticalAlignment attribuutteja Columnin ja Row:n sisällä, jotta ne saataisiin aseteltua käyttöliittymän keskelle. Composessa on käytetty keyboardOptions attribuuttia. Tämä saa arvoksi KeyboardOptions funktion. Funktio ottaa vastaan attribuutin keyboardType ja saa

arvoksi KeyboardType.Text, joka pakottaa arvon olevan kirjaimia. Muita pakollisia Composen tekstikentän attribuutteja ovat value ja onValueChange. Nämä attribuutit ovat tekstikentän syöte arvoja. Koska tekstikentän arvo muuttuu aikanaan, on hyvä tallentaa tämä mutableStateOf tilamuuttujaan. Taulukossa 4 näemme Composella tuotetun koodin ja lopputuloksen.

TAULUKKO 4. Käyttöliittymän toteutus Composella.



Composen toteutuksessa pakollisia attribuutteja on runsaasti vähemmän verrattuna XML-pohjaiseen toteutukseen. Composella kirjoittaessa tarvitsi huomattavasti vähemmän koodia saadakseen halutun lopputuloksen. XML-toteutuksessa ilman startPadding attribuuttia tekstin kirjoituksen aloitus olisi alkanut päällekkäin tekstikentän reunojen kanssa, tätä ei tarvinnut huomioida Composessa.

3.2 Sivuvaikutusten käsittely

Compose-funktioiden täytyy lähtökohtaisesti olla sivuvaikutuksilta vapaita. Tästä syystä hyvät toimintamallit toiminnallisuuksia tehtäessä on näiden toteutuksien toteuttaminen erillisissä funktioissa, kun Compose-funktioissa. Nämä voivat aiheuttaa odottamatonta käyttäytymistä sovellukselta. Ratkaisuna asiaan on Compose-kirjaston sivuvaikutusfunktiot, SideEffect, LaunchedEffect ja DisposableEffect. Näiden kolmen funktion tarkoituksena on auttaa ohjelmoijaa pitämään käyttöliittymän päivittäminen erillään sovelluksen sisäisestä toiminnasta eli dataan liittyvään toimintaan, toisin sanoen sivuvaikutuksista. Näiden tarkoitus ja suurimmat hyödyt on sovelluksen suorituskyvyn parantaminen, koodin pitäminen selkeämpänä ja helpottaa virheiden jäljitystä. (Android for Developers; Morteza 2023)

3.2.1 SideEffect

SideEffect on Compose-funktio, jonka sisällä voidaan kutsua tai määrittää sivuvaikutus toimintoja. SideEffect on kätevimmillään virheiden jäljityksessä. SideEffect-funktiota voi käyttää vain, jos sen isäntäfunktio voidaan uudelleen kääntää (recomposed). Isäntäfunktiolla täytyy siis olla määritelty erillinen tila, mikä tekee Compose-funktiosta uudelleenkäännettävän. Sisäkkäisissä Compose-funktioissa SideEffect lähtee käyntiin aina sisimmässä Compose-funktiossa, vaikka ulkoinen funktio kääntyy uudelleen myös. (Morteza 2023)

3.2.2 LaunchedEffect

LaunchedEffect on osa ”suspended functions” -kategoriaa. Tämä tarkoittaa funktion suorittavan mahdollisesti aikaa vievää toimintaa. Tämä kuuluu myös sivuvaikutuksiin kuuluviin toimintoihin. Compose-funktioissa jos tehdään esimerkiksi API-kutsua ja tapahtuu uudelleen kääntäminen, tämä API-kutsu voi tapahtua uudelleen ja uudelleen, jos LaunchedEffect toimintoa ei ole olemassa ja tätä ei ole määritelty tapahtumaan LaunchedEffect:n sisällä. LaunchedEffect ottaa parametriksi vähintäänkin yhden avain argumentin, joka on tila, joka muuttuu ajan myötä. LaunchedEffect, eli sivuvaikutus, tulee tapahtumaan sen korutiinialueen (coroutineScope) sisällä, missä voi suorittaa aikaa vievän funktion tai ei aikaa vievän. (Kumarin 2023)

3.2.3 DisposableEffect

DisposableEffect ja LaunchedEffect ovat kaksi Jetpack Compose -kirjaston tarjoamaa funktiota, jotka käyttävät tila-avainta. DisposableEffect käynnistää toiminnon onDispose-lohkossa, kun komponentti poistetaan näkymästä tai kun tila-avain muuttuu. Tämä tarjoaa mahdollisuuden suorittaa siivoustoimia ja vapauttaa resursseja. DisposableEffect on hyödyllinen muistinhallinnan tehostamiseen ja suorittaa toiminnon, kun siihen liittyvä komponentti ei ole enää tarpeellinen. (Morteza 2023)

4 INTEROPERAATIBILITEETTI COMPOSEEN JA XML:ÄÄN

Interoperaabiliteetti viittaa tässä kontekstissa XML:n integroimista Jetpack Composeen sekä toisinpäin. XML:llä tehtyjä kustomoituja näkymiä saa upotettua Compose-sovellukseen, sekä XML-pohjaiseen näkymäsovellukseen tehtyä Composenäkymiä. (Ajmal 2021a, 2021b)

4.1 ComposeView XML:ään ja XML-näkymä Composeen

Jetpack Compose on tuottanut ComposeView säiliön, jonka voi lisätä XML-pohjaiseen käyttöliittymä-malliin. Tämä näkymä vaatii samalla tapaa attribuutit, kun XML:ssä näkymät yleisesti. Kun lähdetään tekemään yksinkertaisia composenäkymiä, ei välttämättä ole tarvetta tehdä silloin funktiosta ”composeablea”. Tällaisessa tapauksessa on mahdollista funktion sisälle luomaan näitä Jetpack Composen UI-elementtejä. (Ajmal 2021)

Niin kuin XML-pohjaisessa, tässäkin tapauksessa tapahtuu ComposeView:n lisäys joko uniikilla ID:llä findViewById-funktiolla, tai sitten ”data binding” eli datan sitomisella. (Ajmal 2021)

XML:llä tehty, esimerkiksi monimutkaisempi näkymä on mahdollista lisätä Compose-sovellukseen. Tämä on myös hyödyllinen tapa, jos tarvitset jotain käyttöliittymä elementtiä mitä ei vielä ole olemassa Composessa. Tämä onnistuu esimerkiksi AndroidViewBinding:n kautta. (Künneth 2022. 9.)

4.1.1 Interoperaabiliteetin toteutus

Taulukossa 5 on interoperaabiliteetin toteutus Composenäkymän tuominen XML-pohjaiseen sovellukseen. Ensiksi pitää gradle.build tiedostoon laittaa tarvittavat riippuvaisuudet sekä erilaiset gradle:n määritykset. Toteutusta tehdessä Android Studio ilmoitti Kotlin version olevan myös liian vanha, tämän päivittäminen tapahtui määrittämällä Kotlin plugin 1.7.20 versioon projektijuuren toiseen gradle.build tiedostoon. (KmDev 2023)

TAULUKKO 5. Interopeerabiliteetin toteutus.

```kotlin
        val composeView = findViewById<ComposeView>(R.id.compose_view)
        composeView.setContent {
            InteroperaabiliteettiNakyma()
        }
        //val myButton : Button = findViewById(R.id.button1)
    }
}

@Composable
fun InteroperaabiliteettiNakyma() {

    Button(
        onClick = {
            /*TODO*/
        },
    modifier = Modifier
        .width(250.dp)
        .clip(RoundedCornerShape(25.dp))
    ) { this: RowScope
        Text( text: "I am a compose button")
    }
}
```

```groovy
    // Compose
    implementation 'androidx.compose.ui:ui'
    implementation 'androidx.compose.material:material'
    implementation 'androidx.compose.runtime:runtime'
    implementation 'androidx.activity:activity-compose:1.8.1'
    implementation platform('androidx.compose:compose-bom:2022.10.00')
    implementation 'androidx.compose.ui:ui-graphics'
    implementation 'androidx.compose.ui:ui-tooling-preview'
}
```

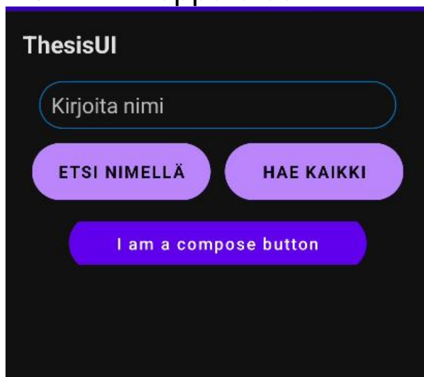
```xml
    <androidx.compose.ui.platform.ComposeView
        android:id="@+id/compose_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"/>
</LinearLayout>
```

```groovy
    buildFeatures {
        compose true
    }
    composeOptions {
        kotlinCompilerExtensionVersion '1.3.2'
    }
```

4.1.2 Interoperaabiliteetin tulos

Kuvassa 7 on käyttöliittymästä kuva, missä ”I am a compose button” on taulukko 5:n lopputulos. Painonappien värit ovat eri väriä, koska molemmat napit on luotu XML:n ja Composen vakioväreillä ja molemmilla tekniikoilla on omat vakiovärinsä. Compose hakee värinsä MaterialTheme kirjaston kautta. (Android for Developers)

KUVA 7. Lopputulos



4.2 Migraatiotaktiikka

Migraatiotaktiikalla kutsutaan tilannetta, kun halutaan korvata XML-pohjainen käyttöliittymä hiljalleen Compose-muotoon. Tämä tapahtuu korvaamalla XML:n käyttöliittymä elementtejä hiljalleen Composella, askeleittain. Jetpack Compose suunniteltiin alusta lähtien näkymä interoperaabilitetiksi tarkoituksena pystyä muuntamaan näkymä pohjainen sovellus hiljalleen Composeksi. (Android for Developers)

4.2.1 Migraation toteutus

Migraatio on XML-näkymäkomponentin korvaaminen Composenäkymällä. Tämä tarkoittaa sitä, että korvataan XML-tiedostosta esimerkiksi Button Compose-näkymällä. Taulukossa 6 on kuvitteellisesti korvattu XML-nappi Composella. Migraatio ja interoperaabiliteetti toteutuu siis samalla tapaa, migraatiossa korvataan ja interoperaabiliteetissa lisätään korvaamisen sijaan. Eroavaisuutena on, että migraatiota ei toteuteta korvaamalla Compose-näkymiä XML-näkymiksi. (Android for Developers)

TAULUKKO 6. Migraation toteutus koodissa.

```
    <androidx.compose.ui.platform.ComposeView
        android:id="@+id/compose_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"/>
</LinearLayout>
```

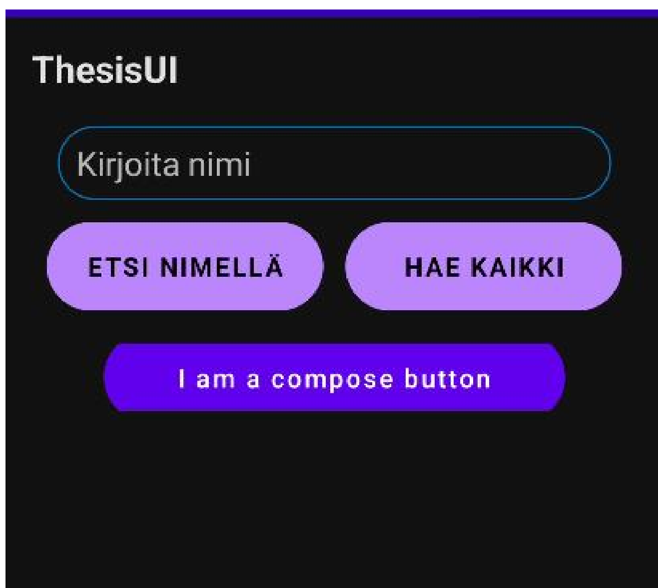
```
        val composeView = findViewById<ComposeView>(R.id.compose_view)
        composeView.setContent {
            InteroperaabiliteettiNakyma()
        }
        //val myButton : Button = findViewById(R.id.button1)
    }
}

@Composable
fun InteroperaabiliteettiNakyma() {

    Button(
        onClick = {
            /*TODO*/
        },
    modifier = Modifier
        .width(250.dp)
        .clip(RoundedCornerShape(25.dp))
    ) { this: RowScope
        Text( text: "I am a compose button")
    }
}
```

4.2.2 Migraation lopputulos

Kuvassa 8 on kuvitteellisen tuloksen lopputulema samoilla koodiasetuksilla, mitä aiemmin on aseteltu interoperaabiliteetissa. Migraation toteutukseen tarvitaan

samat riippuvuudet, mitä interoperabiliteetin toteutus vaatii. (Android for Developers)

KUVA 8. Näkymä migraation toteutuksesta.



5 POHDINTA

Tämän opinnäytetyön tavoitteena oli ymmärtää XML-pohjaisen layout-tyylin ja Jetpack Composen vahvuudet ja heikkoudet sekä verrata näitä kahta keskenään. Jetpack Compose on nopea tapa rakentaa käyttöliittymää. Isoimmat vaarat Jetpack Composen käytössä on sivuvaikutukset. Ne ovat tärkeä osa ymmärtää halutessaan tehdä hyvän ja toimivan sovelluksen. Lähtökohtaisesti kaikkien käyttöliittymä elementtien rakentaminen on paljon nopeampaa Composella, näkymäkomponenttien muokkaus on paljon helpompaa modifier:n kautta, kun lähteä tekemään drawable kansioon muutoksia ja liittää tätä layout-tiedostoon. Myös sisäkkäiset elementit ovat mahdollisesti kevyempiä, kun XML:ssä tehdä sisäkkäisiä layout ratkaisuja päästäkseen samaan tulokseen.

Halutessaan Jetpack Compose sisältää XML:n layout-kaltaiset ratkaisut omina funktioinaan. Kaiken kaikkiaan, jos lähtee suunnittelemaan uuden sovelluksen rakentamista, Jetpack Compose olisi järkevin valinta. Tätä tukee myös mahdollisuus upottaa XML-näkymiä Composeen, eli mahdollisesti voi ottaa käyttöön vanhoista projekteista monimutkaisemmat näkymät uuteen Compose-projektiin, ilman tarvetta kirjoittaa näitä uudestaan Composella. Android mobiilikehittäjänä olisi tärkeää kuitenkin osata molemmat ratkaisut. Kehittäessä vanhempaa sovellusta, on hyvä ymmärtää laajasti XML-layout ratkaisut. Pystyt tekemään uudet komponentit tai jopa korvaamaan vanhat käyttämällä Composea.

XML-näkymä pohjaisessa on omat puolensa. Kun asettelusäiliö on määritetty, on jo tiedossa, miten käyttöliittymän rakennuspalikat tulevat suurin piirtein asettumaan. On huomattavaa, että useat voivat kokea XML:n tavan selkeämmäksi, kun käyttöliittymän rakentamiseen on oma tiedostonsa, jossa käytetään myös XML:n omaa merkintä kieltä. Tämä antaa varsinkin aloittelevalle ohjelmoijalle selkeyttä. Suurin heikkous on mielestäni XML:n tuen katoavaisuus, uudet käyttöliittymä dokumentaatiot ovat Androidin osalta pelkkää Composea.

Composea voi opetella korvaamalla XML:llä tehtyjä komponentteja uudelleen Composella olemassa olevan projektin parissa, tai tehdä uusia Compose-

näkymiä XML-näkymään. Tämä mielestäni helpottaa uuden tekniikan hallitsemista pala palalta.

LÄHTEET

Ajmal, M. 2021. Custom Views in Jetpack Compose (Interoperability) – Part 1. Viitattu 3.12.2023. https://coding-with-aj.medium.com/custom-views-in-jetpack-compose-interoperability-part-1-cdac4824f68e

Ajmal, M. 2021. Add Compose in XML (Interoperability) – Part 2. Viitattu 3.12.2023. https://coding-with-aj.medium.com/add-compose-in-xml-interoperability-part-2-42975b79dbdb

Android for Developers. Migration Strategy. Viitattu 22.11.2023. https://developer.android.com/jetpack/compose/migrate/strategy

Chugh, A. 2022. Android Button Design, Custom Button, Round Button, Color. DigitalOcean. Viitattu 10.11.2023. https://www.digitalocean.com/community/tutorials/android-button-design-custom-round-color

Codepath. Drawables. Viitattu 10.11.2023. https://guides.codepath.com/android/drawables

Codemotion. 2023. Android App Development: Which Language to Choose. Viitattu. 2.12.2023. https://www.codemotion.com/magazine/frontend/mobile-dev/android-app-development-which-language-to-choose/

Counterpoint. 2023. Global Smartphone Sales Shared by Operating System. Viitattu. 2.12.2023. https://www.counterpointresearch.com/insights/global-smartphone-os-market-share/

Gitau, A. How to Implement the Android Lifecycle Callback Methods. Section. Viitattu 10.11.2023. https://www.section.io/engineering-education/understanding-and-implementing-the-android-lifecycle/

Jackson, W. 2014. Pro Android UI. E-kirja. Appress. Viitattu 15.11.2023. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/pro-android-ui/9781430249863/

Gitau, A. How to Implement the Android Lifecycle Callback Methods. Section. Viitattu 10.11.2023. https://www.section.io/engineering-education/understanding-and-implementing-the-android-lifecycle/

KmDev, 2023. Introducing Jetpack Compose into an existing project with XML. Medium. Viitattu 10.12.2023. https://medium.com/@mkcode0323/introducing-jetpack-compose-into-an-existing-project-with-xml-24c42c5be42e

Künneth, T. 2022. Android UI development. E-kirja. Birmingham-Mumbai. Packt Publishing. Viitattu 25.11.2022. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/android-ui-development/9781801812160/

Koffer, P. 2023. What is a Native Mobile App Development? mDevelopers. Viitattu 2.12.2023. https://mdevelopers.com/blog/what-is-a-native-mobile-app-development-

Kumarin, J. 2023. Side-Effects and Effect-Handlers In Jetpack Compose. Medium. Viitattu 28.11.2023. https://blog.stackademic.com/side-effects-and-effect-handlers-in-jetpack-compose-626501f7e14a

Math Open Reference. Viitattu 5.11.2023. https://www.mathopenref.com/vertical.html

Morteza 2023. Jetpack Compose Side Effects in Details. Medium. Viitattu 28.11.2023. https://medium.com/@mortitech/exploring-side-effects-in-compose-f2e8a8da946b

Shah, A. 2022. Jetpack Compose Basics – Part 1. ProAndroidDev. Viitattu 27.11.2023. https://proandroiddev.com/jetpack-compose-basics-26b62999ea9d

Shah, R & Chettri, K. 2020. Android XML Layouts – A Study of ViewGroups And Views for UI Design in Android Application. PDF-dokumentti. Viitattu 5.11.2023. https://andor.tuni.fi/discovery/fulldisplay?docid=cdi_crossref_primary_10_32628_CSEIT206249&context=PC&vid=358FIN_TAMPO:VU1&lang=fi&search_scope=My_inst_and_CI_extended_search&adaptor=Primo%20Central&tab=Everything&query=any,contains,android%20AND%20xml&offset=60

Sivakumar, P. XML Basics. Medium. Viitattu 14.11.2023. https://medium.com/@PrakhashS/xml-basics-a4ea2509f199

Zheng, C. 2022. Why we adopted jetpack compose. ProAndroidDev. Viitattu 24.10.2023. https://proandroiddev.com/why-we-adopted-jetpack-compose-b66bfd3dbde5