



Sovellusten ohjelmointi ja käytettävyys Oppimispäiväkirja

Jaakko Kuusisto

Sisällysluettelo

[1 Viikkoharjoitukset 4](#_Toc184902258)

[1.1 Android -ympäristön asennus ja Hello World 4](#_Toc184902259)

[1.2 Jetpack Compose -tutustuminen 4](#_Toc184902260)

[1.3 Kotlin essentials – osa 1 6](#_Toc184902261)

[2 Viikkoharjoitukset 9](#_Toc184902262)

[2.1 Valuuttamuuntimen käyttöliittymä 9](#_Toc184902263)

[2.2 Sääsovelluksen käyttöliittymä 9](#_Toc184902264)

[2.3 Scaffold ja sen käyttö käyttöliittymässä 10](#_Toc184902265)

[2.4 Kotlin -harjoituksia 11](#_Toc184902266)

[3 Viikkoharjoitukset 12](#_Toc184902267)

[3.1 Lokalisointi 12](#_Toc184902268)

[3.2 Teemat 12](#_Toc184902269)

[3.3 Sovelluksen tila ja toiminnallisuus 14](#_Toc184902270)

[4 Viikkoharjoitukset 4 Navigointi, Androidin Intent -mekanismi 15](#_Toc184902271)

[4.1 Harjoitus 1: Navigointi Jetpack Composessa 15](#_Toc184902272)

[4.2 Harjoitus 2: Bottom Tabs -navigointi 15](#_Toc184902273)

[4.3 Tehtävä 3: Androidin Intent mekanismi 16](#_Toc184902274)

[5 Viikkoharjoitukset 5 Listat ja data. Sovellusprototyypin luominen Figmalla 18](#_Toc184902275)

[5.1 Dataluokat ja listojen toteuttaminen 18](#_Toc184902276)

[5.2 Ravintolan oma detaljinäkymä 20](#_Toc184902277)

[5.3 Sovellusprototyypin luominen Figmalla 21](#_Toc184902278)

[6 Viikkoharjoitukset 6 Kotlin -dataluokat, datan hakeminen palvelimelta, HTTP/JSON ja Retrofit -kirjasto 24](#_Toc184902279)

[6.1 Tehtävä 1: REST-toiminnallisuuden toteuttaminen Android-sovelluksissa 24](#_Toc184902280)

[6.1.1 Miten REST-toiminnallisuus voidaan toteuttaa Kotlin-API:lla 24](#_Toc184902281)

[6.1.2 Milloin Volley-kirjasto on sopiva vaihtoehto REST-pyyntöjen toteutukseen? 24](#_Toc184902282)

[6.1.3 Miksi Retrofit on suosituin kirjasto Android Jetpack Composessa REST-toiminnallisuuden toteutukseen Androidissa? 25](#_Toc184902283)

[6.2 Tehtävä 2: JSON-tiedon konvertointi Kotlin data-luokiksi 25](#_Toc184902284)

[6.3 Tehtävä 3: Tehtävälista-sovellus ja tietojen haku palvelimelta 27](#_Toc184902285)

[6.4 Tehtävä 4: Omavalintainen REST-pohjainen datan näyttösovellus 30](#_Toc184902286)

[7 Viikkoharjoitukset 6 - ViewModel - Persistent Data 33](#_Toc184902287)

[7.1 Tehtävä 1: Yksinkertainen ViewModel 33](#_Toc184902288)

[7.2 Tehtävä 2: Ajastin (Sekuntikello) ViewModelissa 33](#_Toc184902289)

[8 Harjoitustyö sääsovellus 35](#_Toc184902290)

[8.1 Tekoälyn generoima analyysi toteutuksesta: 35](#_Toc184902291)

[8.2 Oma analyysi 36](#_Toc184902292)

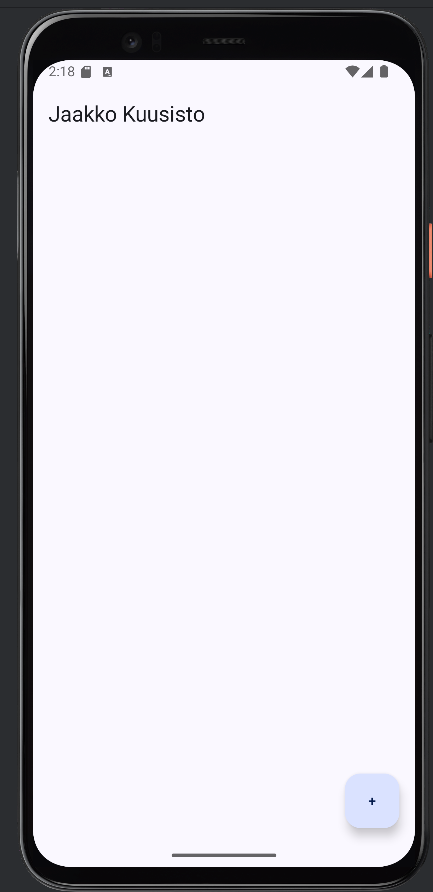
[9 Käytetyt lähteet 38](#_Toc184902293)

# Viikkoharjoitukset

## Android -ympäristön asennus ja Hello World

Android Studio asentui lataamalla Installer Wizard ja perusasetuksilla. Jossain kohtaa Installer jumittui ladatessaan palvelimelta jotain tiedostoa. Nopealla googletamisella tämä vaikutti olevan 10+ vuotta vanha ongelma, mutta ratkesi lataamalla manuaalisesti klikkaamalla tiettyä Android Studion tiedostoa.

 Projektin luonti onnistui hyvin käyttämällä Empty Activity -pohjaa. Projektia luodessa ja ohjelmaa kääntäessä Windows Defender kysyi hyväksyntää useaan otteeseen, mutta antaessa oikeudet, ei Studion käytössä ollut ongelmia.



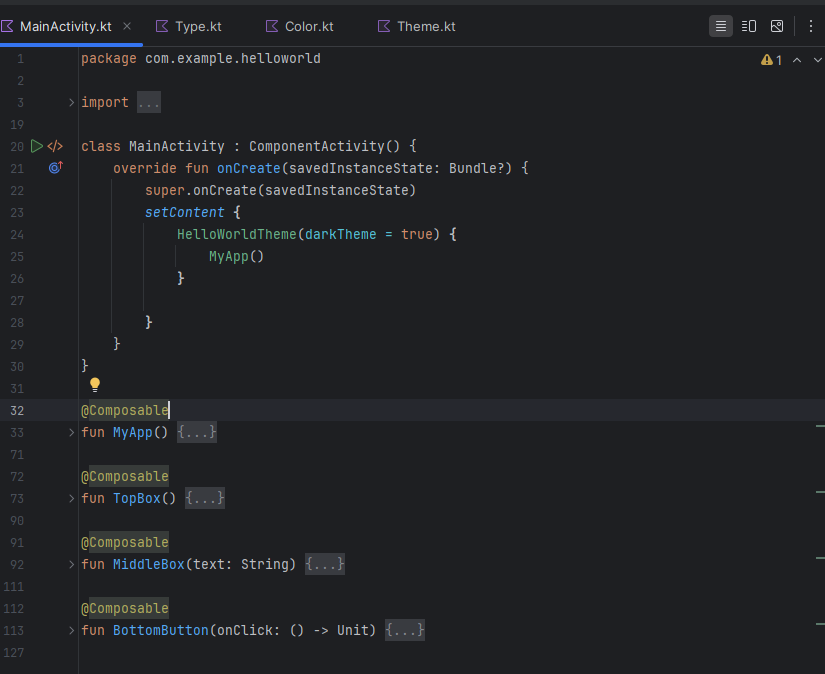
HelloWorld-projekti onnistui muokkaamalla esimerkkikoodia ja lisäämällä Scaffold()-funktioon TopBar() ja siihen teksti.

## Jetpack Compose -tutustuminen

Annotaatiot ovat Kotlinissa/Javassa tapa liittää metadataa koodin osiin, kuten luokkiin tai funktioihin jne.

@Preview on annotaatio, jota käytetään Jetpack Compose -kehityksessä esikatselun mahdollistamiseen. Tällöin IDE:llä voidaan esikatsella komponenttia ilman, että se ajetaan emulaattorissa.

@Composable on annotaatio, joka merkitsee funktiota, joka luo käyttöliittymäelementtejä Jetpack Composessa. @Composable voidaan kutsua vain toisesta @Composable-funktiosta ja näitä yhdistelemällä luodaan käyttöliittymä.

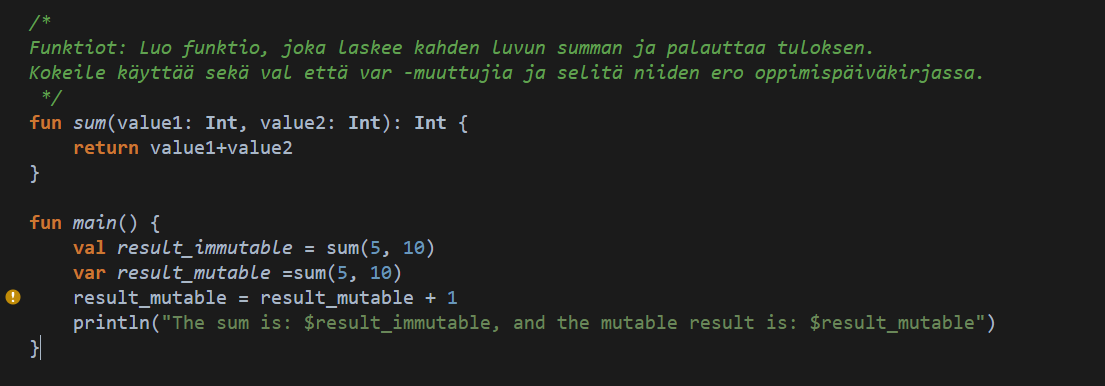


A screen shot of a cell phone

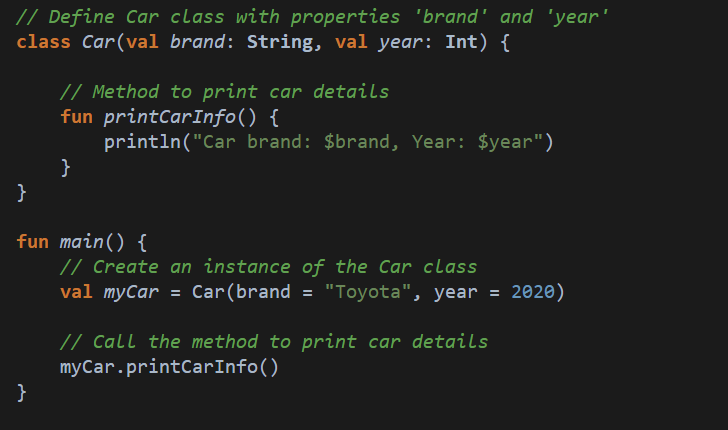
Description automatically generated

Kuvissa Android-sovellus, jossa komponenteille alifunktiot ja noudattaa Jetpack Composen väri- ja typografiateemoja.

## Kotlin essentials – osa 1



Var tarkoittaa muuttujaa ja val tarkoittaa muuttumatonta arvoa.

Null safety tarkoittaa, että Kotlinissa pyritään estämään null-viittausvirheet, jotka voivat aiheuttaa ohjelman kaatumisen. Tämä tehdään siten, että tyypit oletuksena eivät voi sisältää null-arvoa. Jos muuttuja voi olla null, sen tyyppi määritellään erikseen käyttämällä kysymysmerkkiä (?).

A computer screen shot of a program

Description automatically generated

Elvis-operaattori on lyhyt tapa käsitellä mahdollisesti null-arvoja. Se tarkistaa, onko muuttuja null, ja jos on, palauttaa sille määritetyn oletusarvon. Jos muuttuja ei ole null, se palauttaa sen arvon.

A screen shot of a computer

Description automatically generated

Laajennusfunktiot mahdollistavat olemassa olevien luokkien laajentamisen lisäämällä niihin uusia metodeja, muokkaamatta itse luokan koodia.

A screen shot of a computer

Description automatically generated

# Viikkoharjoitukset

## Valuuttamuuntimen käyttöliittymä

Tein UI:n jossa pudotusvalikot valuutoille, laatikko tulostukselle, summalle tekstikenttä ja nappi. Käyttöliittymä modulisoitu jakamalla se composable funktioihin, jotka halutessa voidaan jakaa eri tiedostoihin. Näin pienessä projektissa, en nähnyt tarvetta tähän.

Lähdekoodi: https://github.com/MikeDanton/5G00EV05-3004\_Sovellusten\_ohjelmointi\_ja\_kaytettavyys/tree/main/week2/CurrencyConverter  
A screenshot of a phone

Description automatically generated

## Sääsovelluksen käyttöliittymä

Toteutettiin UI tehtävänmukaiselle sovellukselle jossa erilaisia komponentteja ja omina resursseina kaksi vektorigrafiikkatiedostoa konvertoituna androidin käyttämään tiedostomuotoon.  
Lähdekoodi: https://github.com/MikeDanton/5G00EV05-3004\_Sovellusten\_ohjelmointi\_ja\_kaytettavyys/tree/main/week2/WeatherApp

A screen shot of a phone

Description automatically generated

## Scaffold ja sen käyttö käyttöliittymässä

Tämä tehtävä oli yksinkertainen, koska olen käyttänyt Scaffoldia jo kaikissa aikaisemmikin projekteissa.

Lähdekoodi: https://github.com/MikeDanton/5G00EV05-3004\_Sovellusten\_ohjelmointi\_ja\_kaytettavyys/tree/main/week2/Scaffold  
A screen shot of a phone

Description automatically generated

## Kotlin -harjoituksia

A screenshot of a computer program

Description automatically generated

IntelliJ IDEA:lla demottu ListOf() ja MutableListOf(). Kuten nimi kuvailee, ensimmäinen on muuttumaton, sen arvoja ei voi muuttaa tai lisätä siihen. Muuttuvaan listaan voidaan näin tehdä.

Lähdekoodi: <https://github.com/MikeDanton/5G00EV05-3004_Sovellusten_ohjelmointi_ja_kaytettavyys/tree/main/week2/kotlinHarjoituksia/src/main/kotlin>

When -avainsana on kotlinin vastine switch-case:lle, mutta tukee myös erilaisia tietotyyppejä ja sen takia joustava. ChatGPT kertoo suorituskyvyn vastaavan C++ switch-case:a perustietotyypeillä, joten se on myös tehokas.

# Viikkoharjoitukset

## Lokalisointi

Lokalisointi on hyvä hoitaa resurssitiedostoissa koodin sijaan useista eri syistä. Tavallisiin ohjelmointiperiaatteisiin kuuluu koodin tekeminen modulaariseksi, loogisten ja tekstikomponenttien erottaminen, aivan samalla kuin funktioiden erottaminen yhden tehtävän perusteella. Tämä myös johtaa helpompaan koodin ylläpitoon ja myös ohjelmointia osaamaton henkilö voi tehdä käännöksiä helposti. A screen shot of a phone

Description automatically generatedA black and white sign with a star

Description automatically generated

Kuvissa tehtävän sovellus oletus- (englanti tässä) ja suomenkieliasetuksilla. Kieli ja teema-asetuksia voi vaihtaa emulaattorin pikavalikosta.

## Teemat

Teemoilla on samat hyödyt kuin lokalisoinnilla. Modulaarisuus, loogisten ja ulkoasukomponenttien erottaminen. Koodin ylläpito jne. Teemoilla määritellään eri komponenttien värejä ja tekstin fonttiasetuksia.

A screen shot of a cell phone

Description automatically generatedA close-up of a certificate

Description automatically generated

A screen shot of a computer

Description automatically generated

Kuvissa eri kaksi moodia, satunnaisesti valitusta theme builder tiedostosta ja custom- väriteemoja. Sovelluksen alanurkan napista väriteema vaihtuu.

## Sovelluksen tila ja toiminnallisuus

A screenshot of a phone

Description automatically generatedA screenshot of a phone

Description automatically generated

Kuvissa polttoainekustannuslaskin kahdella eri lokalisaatiolla ja dark/light modella.

Käyttöliittymän tila määrittää, miltä sovelluksen käyttöliittymä näyttää ja miten se käyttäytyy tietyllä hetkellä. Esimerkiksi valittu teema tai mitkä elementit ovat näkyvillä.

Remember on Jetpack Composen funktio, jonka avulla voidaan säilyttää muuttujan arvo komposition avulla. Tätä voidaan käyttää tilojen säilyttämiseen.

RememberSaveable on samankaltainen, mutta säilyttää arvon silloinkin, kun sovellus on taustalla.

# Viikkoharjoitukset 4 Navigointi, Androidin Intent -mekanismi

## Harjoitus 1: Navigointi Jetpack Composessa

A screenshot of a phone

Description automatically generatedA screenshot of a phone

Description automatically generated

Tässä demosovelluksessa käytettiin Androidin NavHost -Composable funktiota ja NavController -luokkaa toteutuksessa siirtyä eri näkymiin. NavController vastaa navigoinnista ja Composable -funktio logiikasta.

## Harjoitus 2: Bottom Tabs -navigointi

A screenshot of a phone

Description automatically generated

Tehtävässä käytin NavigationBaria tehtävänannossa mainitun BottomNavigation -komponentin sijaan. <https://m2.material.io/components/bottom-navigation>, ymmärrykseni mukaan BottomNavigation on vanhentunut, material 2 ominaisuus ja material 3 mukainen tapa on NavigationBar. En saanut material 2 ominaisuuksia importattua, muuttamatta erilaisia Gradle ja build-asetuksia ja oletin, että se ei ollut tehtävänannon mukaista.

## Tehtävä 3: Androidin Intent mekanismi

Common intents -mekanismi Androidissa tarkoittaa järjestemän tarjoamia vakioituja toimintoja, joita sovellukset voivat käyttää helpottamaan vuorovaikutusta muiden sovellusten ja palveluiden kanssa. Esimerkiksi avata selain tai soittaa puhelu.

A screen shot of a cell phone

Description automatically generated

Kuvassa aloitusnäkymä ja Common Intents -mekanismin käyttöä. Android emulaattoreista usein puuttuu hälytyssovellus, joten if-else lausekkeella järjestelmä ilmoittaa puuttuvasta hälytys-sovelluksesta.

# Viikkoharjoitukset 5 Listat ja data. Sovellusprototyypin luominen Figmalla

## Dataluokat ja listojen toteuttaminen

Kotlinin data class (dataluokka) on erikoistyyppinen luokka, joka on suunniteltu sisältämään ja käsittelemään dataa. Kotlinin dataluokat automatisoivat joidenkin metodien, kuten equals(), hashCode(), toString(), ja copy(), jotka Javassa täytyy kirjoittaa käsin. Tämän takia ne ovat tehokkaita käyttää.

Column ja LazyColumn ero on, että Column piirtää kaikki lapsielementit, riippumatta mahtuvatko ne ruudulle. LazyColum piirtää elementtejä vasta kun ne näkyvät ruudulla. Tämä säästää muistia ja suorituskykyä.

A screenshot of a phone

Description automatically generatedA screen shot of a computer code

Description automatically generated

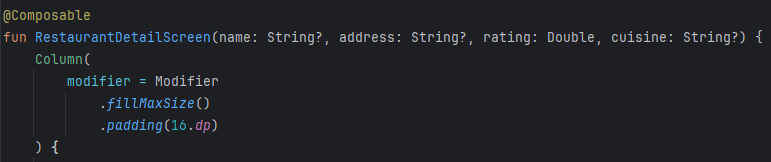
A computer screen shot of a program code

Description automatically generated

Kyseiseen Ohjelmaan totetutettiin hakutoiminto käyttäen Data Class -luokkaa hyväksi, sijoitettiin data LazyColumniin ja piirrettiin.

## Ravintolan oma detaljinäkymä

Seuraavaksi toteutettiin ohjelmalle detailScreen, joka listaa ravintolan tarkemmat tiedot. Tänne navigointiin käyttäen aiemmilta viikolta tuttuva navControlleria.



A screen shot of a computer

Description automatically generated

## Sovellusprototyypin luominen Figmalla

Päätin tehdä idle-game sovelluksen, joka loppupeleissä ei välttämättä ollut paras valinta, koska pelit vaativat kustomoituja assetteja ja tarkan asettelun hahmottaminen voi olla haastavaa. Hahmotelin viisi näkymää ensin paperilla, joista yksi esimerkkinä. Sen jälkeen yritin tehdä tehtävää Simple designila, mutta en löytänyt haluamiani UI-komponentteja joten tein loput material UI 3. Siirsin käsin piirtämäni hahmotelmat figmaan ja asettelin niiden päälle UI-elementtejä.

https://www.figma.com/design/DFibqjvpeJGvUABfaQWFcG/IdleGame?node-id=1-1964&t=WJ8uDmd3zn0rdFj6-1

A screenshot of a game

Description automatically generatedA drawing of a basket with a couple of papers

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

A white board with black text and symbols

Description automatically generated

# Viikkoharjoitukset 6 Kotlin -dataluokat, datan hakeminen palvelimelta, HTTP/JSON ja Retrofit -kirjasto

## Tehtävä 1: REST-toiminnallisuuden toteuttaminen Android-sovelluksissa

### Miten REST-toiminnallisuus voidaan toteuttaa Kotlin-API:lla

**HttpURLConnection**: Tämä on osa Java-kirjastoa, ja se on saatavilla Android-sovelluksissa ilman lisäkirjastoja. HttpURLConnectionilla voidaan tehdä suoria HTTP-pyyntöjä, mutta sen käyttö on hieman monimutkaista verrattuna muihin vaihtoehtoihin, sillä se ei tarjoa automaattista JSON-käsittelyä tai välimuistia.

**OkHttpClient**: Tämä on erillinen kirjasto, joka tekee verkkopyynnöistä huomattavasti yksinkertaisempia verrattuna HttpURLConnectioniin. OkHttpClient tarjoaa suoraan erilaisia toimintoja, kuten välimuistin hallinnan, automaattiset uudelleenohjaukset ja JSON-integraation yhdessä muiden kirjastojen, kuten Gsonin, kanssa.

Jos suoria yhteyksiä tehdään pääsäikeessä, tämä voi johtaa sovelluksen suorituskyvyn heikkenemiseen. Verkkopyynnöt ovat hitaita ja niiden suorittaminen pääsäikeessä voi johtaa ANR (Application Not Responding) -virheeseen, mikä vaikuttaa negatiivisesti käyttäjäkokemukseen. Siksi kaikki verkkopyynnöt tulisi aina suorittaa erillisellä taustasäikeellä tai asynkronisesti esimerkiksi Kotlin Coroutines - tai AsyncTask-luokalla.

### Milloin Volley-kirjasto on sopiva vaihtoehto REST-pyyntöjen toteutukseen?

**Volley** on Googlen kehittämä kirjasto, joka on optimoitu REST-pyyntöjen suorittamiseen Android-sovelluksissa. Sitä on hyvä käyttää, kun sovelluksen pitää käsitellä useita peräkkäisiä verkkopyyntöjä tai kun sovelluksessa on tarve välimuistille, sillä Volley tarjoaa sisäänrakennetun välimuistimekanismin ja JSON-tuen.

* **Milloin käyttää Volleyta**: Volley on erityisen hyödyllinen yksinkertaisiin, pieniin pyyntöihin, jotka edellyttävät välimuistia ja puskurointia, kuten kuvanlataukseen ja yksinkertaisiin JSON-tietopyyntöihin.
* **Keskeiset ominaisuudet**: Volleyn keskeisiä ominaisuuksia ovat sen sisäänrakennettu **välimuisti** ja **JSON**-tuki. Välimuistin avulla samoja pyyntöjä ei tarvitse toistaa verkossa, mikä parantaa sovelluksen suorituskykyä ja vähentää verkkoliikennettä. Lisäksi Volley käsittelee automaattisesti JSON-objekteja, mikä helpottaa JSON-pohjaisten REST-rajapintojen kanssa työskentelyä.

### Miksi Retrofit on suosituin kirjasto Android Jetpack Composessa REST-toiminnallisuuden toteutukseen Androidissa?

Retrofit on usein parempi vaihtoehto kuin Volley, erityisesti laajoissa projekteissa, joissa on tarvetta käsitellä useita erilaisia rajapintoja tai kompleksisia JSON-rakenteita. Lisäksi Retrofitin kyky toimia Kotlin Coroutines -tuen kanssa tekee siitä ihanteellisen Jetpack Compose -ympäristössä, jossa asynkronisuus on keskeisessä roolissa.

## Tehtävä 2: JSON-tiedon konvertointi Kotlin data-luokiksi

Data-luokat luovat automaattisesti toimintoja, kuten toString(), equals(), ja hashCode(), ja mahdollistavat kopioinnin copy()-funktiolla. Tämä tekee niistä erittäin sopivia tilanteisiin, joissa tarvitaan pelkästään datan säilyttämistä, kuten JSON-muotoisten tietorakenteiden mallintamiseen. Kotlinissa tämä on myös yksinkertaisempaa kuin Javassa, koska Kotlinin data-luokat ovat vähäkoodisia ja suoraviivaisia verrattuna Javan luokkiin.

A screen shot of a computer program

Description automatically generatedA screen shot of a computer code

Description automatically generated

Tehtiin jokaisesta objektista dataluokka. Koska viimeisessä User-tapauksessa oli sisäkkäisiä rakenteita, tuli rakenteeseen useampi dataluokka.

## Tehtävä 3: Tehtävälista-sovellus ja tietojen haku palvelimelta

A screen shot of a computer program

Description automatically generated

Ennen koodausta suurin haaste on laittaa gradlen build kuntoon retrofitin ja corutiinien osalta.

Aluksi importataan retrofit-kirjasto ja luodaan singleton-objekti yhteyttä varten:

A computer screen shot of a program code

Description automatically generated

Retrofit hoitaa Interfacen http-pyyntöjä varten

A computer screen shot of a program

Description automatically generated

Periytetään TodoViewModel-luokka, joka noutaa datan tietoliikennemoduulia käyttäen ja säilyttää sen.

A computer screen shot of a program code

Description automatically generated

Siistitään data esitettävään muotoon, esimerkiksi lisäämällä Checkbox suoritetuille todo:ille.

A screen shot of a computer program

Description automatically generated

Ja lopullinen sovellus näyttää:

A screenshot of a computer

Description automatically generated

Tehtävässä käytetty todo-dataluokka:A screenshot of a computer program

Description automatically generated

## Tehtävä 4: Omavalintainen REST-pohjainen datan näyttösovellus

Käytettiin samaa mallia kuin edellisessä tehtävässä eli retrofit. Android manifestissa annetaan oikeudet käyttää internet yhteyttä, joka unohtui edellisessä tehtävässä mainita.

A screen shot of a computer program

Description automatically generated  
Itse UI on simppeli, käytetään taas viewModel-pohjaluokkaa, jolla noudetaan data.

Network-moduuli ja sen interface openWeather apiin ovat seuraavat:  
A computer screen shot of a program code

Description automatically generatedA screen shot of a computer screen

Description automatically generated

Ja lopuksi kuva itse sovelluksesta: A screen shot of a screen

Description automatically generated

# Viikkoharjoitukset 6 - ViewModel - Persistent Data

## Tehtävä 1: Yksinkertainen ViewModel

A screen shot of a computer program

Description automatically generated

Tehdäään CounterViewModel-luokka periyttämällä VievModelista.

A screenshot of a computer

Description automatically generated

## Tehtävä 2: Ajastin (Sekuntikello) ViewModelissa

A screen shot of a computer program

Description automatically generated

Määritellään \_seconds MutableStateFlowna ja paljastetaan se read-only Stateflowna seconds -muuttujalle. Init-blokissa coroutine kasvattaa laskurin arvoa joka sekunti.

A screen shot of a computer program

Description automatically generated

Ui päivittyy automaattisesti käyttämällä collectAsState() tarkkailemalla.

# Harjoitustyö sääsovellus

## Tekoälyn generoima analyysi toteutuksesta:

* **REST API -integraatio**
* Sovellus käyttää OpenWeatherMap APIa (WeatherApiService) hakemaan säätietoja kaupungin tai sijainnin koordinaattien perusteella​​.
* WeatherRepository hoitaa verkkoviestinnän ja abstraktoi API-kutsut sovelluksen käyttöön​.

**Tila:** Toteutettu.

* **2. Navigointi ja sivut**
* Sovelluksessa on kaksi pääsivua:
  + WeatherScreen, joka näyttää säätiedot​.
  + SettingsScreen, jossa hallitaan käyttäjän asetuksia ja näkymän sisältöä​.
* Navigointi toteutetaan NavHostilla, joka sijaitsee MainActivity-tiedostossa​.

**Tila:** Toteutettu.

* **3. Material UI**
* Sovelluksessa käytetään Material UI -komponentteja, kuten Scaffold, Button ja Checkbox, käyttöliittymän rakenteessa​​.

**Tila:** Toteutettu.

* **4. Common Intentit**
* Sovellus hyödyntää sijaintilupia ja karttanäkymää, jossa käyttäjä voi valita sijainnin​.

**Tila:** Toteutettu. (Tämä tosin ei toimi halutulla tavalla, luulin että google mapsiin olisi ollut jokin API jolla sovellus ja Gmaps voisivat keskustella. Tällä hetkellä sovellus vain avaa karttapalvelun, eikä liity sääsovellukseen mitenkään. Tarjolla olisi ollut muita sovelluksia, jolla olisi voinut tarkastella säätä kartalla, mutta en jaksanut kirjoittaa koodia uudestaan.)

* **5. Resurssit**
* Sovelluksen merkkijonot on siirretty resurssitiedostoihin (strings.xml) lokalisaation mahdollistamiseksi​​.

**Tila:** Toteutettu.

* **6. Lokalisaatio**
* Sovellus tukee kahta kieltä: suomea ja englantia. Lokalisaatio on toteutettu täydellisesti kahden strings.xml-tiedoston avulla​​.

**Tila:** Toteutettu.

* **7. Pysyvä tiedon tallennus**
* Sovellus käyttää DataStorea käyttäjän asetusten ja viimeksi tunnetun sijainnin tallentamiseen​.

**Tila:** Toteutettu.

* **8. Valinnainen laiterajapinta**
* Sijaintipohjainen säähaku on toteutettu LocationHelperilla ja Google Play Services -integraatiolla​.

**Tila:** Toteutettu (valinnainen vaatimus).

## Oma analyysi

Harjoitustyönä toteutin sääsovelluksen, jossa on kaksi näkymää. Päänäkymä, joka näyttää säätiedot ja navigointi asetuksiin ja Google mapsiin. Alla on kuvankaappaukset suomenkielisestä lokalisaatiosta (Nappien lokalisaatio on unohtunut, kuten kuvasta näkyy). Asetuksista voidaan muuttaa muutamaa asetusta, jotka muuttavat päänäkymää.

Oli mukavaa tehdä hieman laajempi ohjelma, ainaisen uuden projektiasetusten säätämisen sijaan. Projektista opin ainakin Javan com-paketeista ja kansiorakenteesta, joka oli melko toimivaa näin C++ taustaiselle. Pakettien lisääminen kesken projektin oli yksinkertaista ja helppoa. 1,5kk työn ja raportin välillä hieman heikentää yksityiskohtia kaikesta tehdystä.

A screenshot of a phone

Description automatically generatedA screenshot of a phone

Description automatically generated

# Käytetyt lähteet

https://chatgpt.com  
https://kotlinlang.org/docs/home.html