RAZVOJ MOBILNIH APLIKACIJA:

LV 4: MVVM i rad s više zaslona

1. Uvod

Android projekt koji će služiti kao polazišna točka za ovu vježbu možete preuzeti s sljedeće poveznice: https://drive.google.com/file/d/1Iw6PcSKfU4WBrHGtjCnCpLYaKzsxV8bE/view?usp=sharing. Preuzetu ZIP datoteku potrebno je raspakirati, a zatim u Android Studio uvesti projekt putem opcije File -> Open i odabrati raspakiranu mapu projekta. Nakon otvaranja, pričekajte da se projekt sinkronizira s Gradle datotekama kako bi bio spreman za rad.

Nakon otvaranja projekta, potrebno je povezati ga s vašom Firestore bazom podataka koju ste kreirali u prethodnoj vježbi. Da biste to učinili, slijedite sljedeće korake:

1. **Promijenite ID dokumenta u kodu**

U kodu projekta pronađite dio gdje se referencira ID dokumenta za Firestore i zamijenite ga s ID-em dokumenta iz vaše baze podataka, kao što je prikazano na slici 2.

```
Button(
    onClick = {
        // Update local values
        if (newWeight > 0) currentWeight = newWeight
        if (newHeight > 0) currentHeight = newHeight

        val docRef = db.collection("BMI").document("9NlYwzeyhFzUp5RAluAp")
        docRef.update(
            mapOf(
                "Tezina" to newWeight,
                "Visina" to newHeight
            )
        )
    },
    modifier = Modifier
        .padding(top = 16.dp)
) {
    Text("Spremi promjene")
```

Slika 2. *Promjena ID dokumenta*

2. **Povežite projekt s Firestore bazom putem Android Studio alata**:

Idite na izbornik Tools -> Firebase.

U Firebase prozoru koji se otvori odaberite opciju Cloud Firestore.

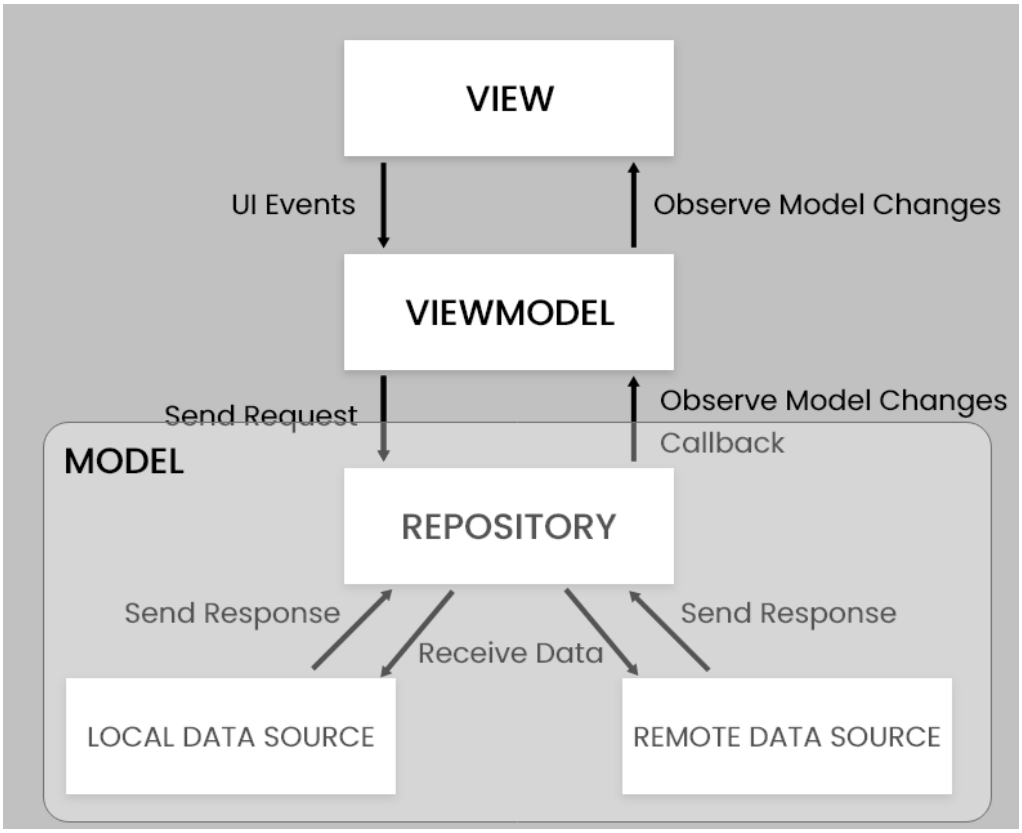
Kliknite na Get started with Firestore.

Slijedite upute za povezivanje i odaberite vašu Firestore bazu podataka koju ste kreirali u prethodnoj vježbi.

Nakon ovih koraka, projekt će biti povezan s vašom Firestore bazom i spreman za daljnji rad.

2. MVVM

MVVM (Model-View-ViewModel) je arhitektonski uzorak koji se koristi u razvoju softvera, uključujući Android aplikacije, kako bi se olakšala organizacija koda, omogućilo jednostavno testiranje i osigurala bolja modularnost. Na slici 1. prikazan je dijagram MVVM arhitekture.



Slika 2. *MVVM arhitektura*

MVVM dijeli aplikaciju na tri ključne komponente:

Model:

Model predstavlja podatke i poslovnu logiku aplikacije. To uključuje strukture podataka (npr. objekt koji sadrži informacije o korisniku, poput visine i težine) te logiku za rad s podacima, poput spremanja u bazu podataka ili dohvaćanja podataka s poslužitelja. Model

je potpuno neovisan o korisničkom sučelju i ne zna ništa o View-u ili ViewModelu, što omogućava njegovu ponovnu upotrebu u različitim dijelovima aplikacije.

View:

View je odgovoran za prikaz korisničkog sučelja (UI) i interakciju s korisnikom. U Android aplikacijama, View može biti implementiran pomoću XML layouta ili Jetpack Compose-a. View promatra podatke koje mu pruža ViewModel i ažurira se automatski kada se ti podaci promijene (npr. koristeći reaktivne mehanizme poput LiveData ili StateFlow). View također šalje korisničke akcije (npr. klik na gumb) ViewModelu.

ViewModel:

ViewModel djeluje kao posrednik između Modela i View-a. On upravlja stanjem UI-a i sadrži poslovnu logiku koja se odnosi na prikaz podataka. ViewModel priprema podatke za View (npr. formatira ih ili izračunava potrebne vrijednosti) i obrađuje korisničke akcije, poput spremanja novih podataka u bazu. U Androidu, ViewModel je dio Jetpack biblioteke i osigurava da podaci prežive promjene konfiguracije, poput rotacije ekrana, jer je neovisan o životnom ciklusu View-a.

Repozitorij

Iako repozitorij nije ključna komponenta MVVM-a, često se koristi kao dodatni sloj za upravljanje podacima. Repozitorij djeluje kao posrednik između ViewModela i izvora podataka, skrivajući detalje implementacije (npr. rad s bazom podataka ili mrežnim pozivima). Na dijagramu je prikazano da repozitorij prima zahtjeve od ViewModela ("Send Request") i komunicira s lokalnim ("Local Data Source") i udaljenim ("Remote Data Source") izvorima podataka. Repozitorij šalje odgovore i prima podatke od izvora, a zatim prosljeđuje promjene ViewModelu putem povratnih poziva.

3. Rad s više zaslona

U modernim Android aplikacijama često je potrebno omogućiti navigaciju između više zaslona kako bi se korisnicima pružilo fluidno i intuitivno iskustvo. U Jetpack Composeu, navigacija između zaslona implementira se pomoću **Jetpack Navigation Compose** biblioteke, koja je prilagođena za rad s deklarativnim pristupom Composea. Ova biblioteka omogućava definiranje navigacijskih ruta, upravljanje povratnim stogom (back stack) i prijenos podataka između zaslona, uz minimalnu količinu koda.Rad s više zaslona u Jetpack Composeu temelji se na konceptu

navigacijskog grafa, gdje svaki zaslon (Composable) predstavlja odredište (destination), a navigacija između zaslona definira se putem ruta.

Ključni Koncepti Navigacije u Jetpack Composeu

NavController:

NavController je središnji objekt koji upravlja navigacijom. Koristi se za prelazak na nova odredišta, povratak na prethodni zaslon i upravljanje povratnim stogom. U Composeu, NavController se stvara pomoću rememberNavController(), što osigurava da se isti primjerak zadrži tijekom životnog ciklusa aplikacije.

NavHost:

NavHost je Composable koji definira navigacijski graf i povezuje rute s određenim zaslonima (Composable funkcijama). Unutar NavHost-a definiraju se sva odredišta (zasloni) i njihove rute. Na primjer, možemo imati rutu "bmi" koja vodi na zaslon za unos BMI-a i rutu "history" koja vodi na zaslon s poviješću mjerenja.

Rute i Odredišta:

Svaki zaslon ima jedinstvenu rutu (string identifikator, npr. "bmi" ili "history"). Rute se koriste za navigaciju do određenog zaslona. Odredišta se definiraju unutar NavHost-a pomoću composable funkcije, gdje svaka composable lambda definira sadržaj zaslona.

Prijenos Podataka Između Zaslona:

Podaci se mogu slati između zaslona putem argumenata u ruti. Na primjer, ruta može izgledati kao "details/{id}", gdje je {id} parametar koji se prosljeđuje ciljanom zaslonu. Alternativno, podaci se mogu dijeliti putem zajedničkog ViewModela, što je često bolji pristup za složenije aplikacije jer osigurava konzistentnost stanja.

Povratni Stog (Back Stack):

Jetpack Navigation automatski upravlja povratnim stogom. Kada korisnik navigira na novi zaslon, stari zaslon se stavlja na stog, a pritiskom na gumb za povratak (ili pozivom popBackStack()) vraća se na prethodni zaslon. Možeš manipulirati stogom, npr. ukloniti određene zaslone ili navigirati na korijenski zaslon.

4. Rad na vježbi

Prvi korak u implementaciji MVVM arhitekture je kreiranje modela podataka koji će služiti kao temelj aplikacije. U tu svrhu potrebno je kreirati novu Kotlin datoteku i nazvati je Measurement.kt. Unutar te datoteke definiramo data klasu koja će sadržavati ključne atribute potrebne za rad aplikacije. Programski kod 1. Prikazuje kreiranje data klase Measurement.

```
data class Measurement(
    val height: Float,
    val weight: Float
)
```

Programski kod 1. *Kreiranje data klase Measurement*

Nakon što je model podataka kreiran, sljedeći korak je implementacija repozitorija koji će služiti kao posrednik između ViewModela i izvora podataka. U tu svrhu potrebno je stvoriti novu Kotlin datoteku pod nazivom MeasurementRepository.kt. Unutar te datoteke definirati klasu MeasurementRepository koja će upravljati komunikacijom s izvorom, omogućujući spremanje podataka na strukturiran i modularan način. MeasurementRepository klasa je prikazana programskim kodom 2.

```
class MeasurementRepository {
    private val db = FirebaseFirestore.getInstance()
    private val docRef =
db.collection("BMI").document("9NlYwzeyhFzUp5RAluAp")

    fun saveMeasurement(measurement: Measurement) {
        val data = mapOf(
            "Visina" to measurement.height,
            "Tezina" to measurement.weight
        )
        docRef.update(data)
    }
}
```

Programski kod 2. *Kreiranje klase MeasurementRepository*

Sljedeće na redu jest kreiranje ViewModela. Za to je potrebno kreirati novi .kt file pod nazivom BmiViewModel i unutar njega implementirati BmiViewModel klasu prema programskom kodu 3.

```
class BmiViewModel : ViewModel() {
    private val _height = mutableStateOf(1.91f)
    private val _weight = mutableStateOf(60f)
    private val _bmi = mutableStateOf("0.00")
    val height: State<Float> = _height
    val weight: State<Float> = _weight
    val bmi: State<String> = _bmi

    private val repository = MeasurementRepository()

    fun updateMeasurements(newHeight: Float?, newWeight: Float?) {
        if (newWeight != null && newWeight >= 0) {
            _weight.value = newWeight
        }
        if (newHeight != null && newHeight >= 0) {
            _height.value = newHeight
        }
        if (_height.value > 0) {
            _bmi.value = String.format("%.2f", _weight.value / (_height.value
* _height.value))
        }
        repository.saveMeasurement(
            Measurement(
                height = _height.value,
                weight = _weight.value,
            )
        )
    }
}
```

Programski kod 3. *Kreiranje klase BmiViewModel*

Klasa BmiViewModel omogućuje ažuriranje vrijednosti za visinu i težinu u bazi podataka putem funkcije updateMeasurements i korištenjem repository objekta, čime se osigurava da su podaci sigurno pohranjeni i dostupni za buduće korištenje. Ova funkcija prima nove vrijednosti visine i težine, provjerava njihovu valjanost (npr. jesu li veće ili jednake nuli), ažurira trenutno stanje tih vrijednosti u ViewModelu i zatim ih prosljeđuje repozitoriju. Repozitorij, koji je instanca klase MeasurementRepository, brine se o komunikaciji s Firebase bazom podataka, gdje se podaci spremaju u određeni dokument unutar kolekcije "BMI". Na taj način, BmiViewModel ne samo da upravlja podacima za prikaz na ekranu, već i osigurava da se korisnički unosi trajno pohrane u bazi, omogućujući aplikaciji da zadrži konzistentnost podataka čak i nakon ponovnog pokretanja.

Nakon implementacije modela i viewModela, sljedeći korak u MVVM arhitekturi jest urediti View, odnosno prilagoditi prikaz podataka korisniku. View je dio aplikacije koji definira kako će podaci biti prikazani na ekranu i kako korisnik može s njima komunicirati. U ovom slučaju, to ćemo postići putem prilagodbi funkcije userPreview što će biti zadatak za samostalni rad.

4.1. Navigacija

Da bi se omogućilo korištenje navigacije unutar Composea potrebno je dodati sljedeću liniju koda u gradle i sinkronizirati projekt:

```
implementation "androidx.navigation:navigation-compose:2.7.7"
```

Za implementaciju drugog zaslona aplikacije koji prikazuje broj koraka, prvo ćemo kreirati novu datoteku **StepsScreen.kt** koja će sadržavati kod za zaslon na kojem će se prikazivati broj koraka. Programski kod 4. Prikazuje StepsScreen funkciju.

```
@Composable
fun StepsScreen() {
    var steps = 0
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement =
androidx.compose.foundation.layout.Arrangement.Center
    ) {

        Text(text = "Broj koraka:")
        Text(text = "${steps}")
    }
}
```

Programski kod 4. *StepsScreen funkcija*

Kako bi se omogućila navigacija između početnog zaslona i zaslona za korake potrebno je unutar glavne aktivnosti (*MainAcitvity*) definirati NavHost koji upravlja navigacijom između zaslona, što je prikazano programskim kodom 5.

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            LV4Theme {
                val navController = rememberNavController()
                NavHost(navController = navController, startDestination = "
HomeScreen") {
                    composable("HomeScreen") {
                        HomeScreen(navController = navController)
                    }
                    composable("stepsScreen") {
                        StepsScreen(navController = navController)
                    }

            }
        }
    }
}
```

Programski kod 5. *Definiranje NavHost-a*

Te u funkcije HomeScreen, BackgroundImage, UserPreview i StepsScreen dodati parametar navController koji će biti objekt klase NavController.

Kako bi se omogućila navigacija s početnog zaslona na zaslon za prikaz broja koraka, potrebno je dodati gumb koji će, nakon što se pritisne, preusmjeriti korisnika na zaslon za korake. Ovaj proces je prikazan u programskom kodu 6.

```
Button(
    onClick = { navController.navigate("stepsScreen") },
    modifier = Modifier
        .padding(bottom = 16.dp, end = 16.dp)
        .wrapContentSize()
) {
    Text("Pogledaj broj koraka")
}
```

Programski kod 6. *Navigacijski gumb*

5. Zadatci za samostalni rad

Potrebno je dodati gumb na zaslonu za prikaz koraka koji omogućuje povratak na početni zaslon aplikacije. Prilagoditi *UserPreview* funkciju kako bi bila u skladu s MVVM (Model-View-ViewModel) arhitekturom. Trenutna verzija funkcije koristi lokalne varijable za upravljanje stanjem visine, težine i BMI-ja te direktno komunicira s Firebase bazom podataka, što nije u skladu s MVVM principima. Umjesto toga, potrebno je koristiti postojeći BmiViewModel za upravljanje stanjem i spremanje podataka, a View (korisničko sučelje) treba samo prikazivati podatke i prosljeđivati korisničke akcije ViewModelu.