RAZVOJ MOBILNIH APLIKACIJA:

LV 3: Spremanje podataka

1. Stvaranje prve Andorid aplikacije

U ovoj vježbi bit će obrađeno spremanje podataka u Android aplikacijama. Na Android platformi trajna pohrana podataka može biti implementirana korištenjem različitih metoda, ovisno o potrebama aplikacije.

Za jednostavne podatke, poput korisničkih postavki ili manjih informacija, koristi se **DataStore**, kojim se omogućuje jednostavno spremanje i dohvaćanje podataka. Za veće količine strukturiranih podataka kojima se može upravljati, primjenjuje se **SQLite**, ugrađena baza podataka namijenjena složenijim aplikacijama. Kao druga mogućnost, mogu biti korištene obične **datoteke** na uređaju za pohranu podataka, što se može pokazati korisnim u specifičnim situacijama.

Glavni naglasak ove vježbe bit će stavljen na udaljenu pohranu podataka. Podaci će biti spremani na vanjskom poslužitelju umjesto na samom uređaju. U tu svrhu može biti razvijena vlastita pozadina (backend) ili mogu biti iskorištene gotove usluge poput **Firebasea**, kojim se olakšava rad s udaljenim podacima.

1.1. DataStore

U sklopu Android Jetpack Composea, DataStore je predstavljen kao rješenje za pohranu podataka kojim se omogućuje spremanje parova ključ-vrijednost. Za asinkronu, dosljednu i transakcijsku pohranu podataka, DataStoreom se koriste Kotlin korutine i Flow. Ovaj alat je dizajniran za male i jednostavne skupove podataka, te se može smatrati modernom zamjenom za SharedPreferences.

1.2. Room baza podataka

Room je predstavljen kao dio Android Jetpacka i koristi se za pohranu strukturiranih podataka na uređaju. Ova baza podataka temelji se na SQLite-u, ali je dizajnirana kako bi se olakšalo upravljanje podacima uz manje koda. Podaci se mogu spremati, dohvaćati i ažurirati na jednostavan način uz pomoć anotacija, dok se osigurava sigurnost i provjera tipova tijekom kompajliranja. Room je idealan za aplikacije kojima je potrebna lokalna pohrana većih količina podataka, poput lista ili tablica.

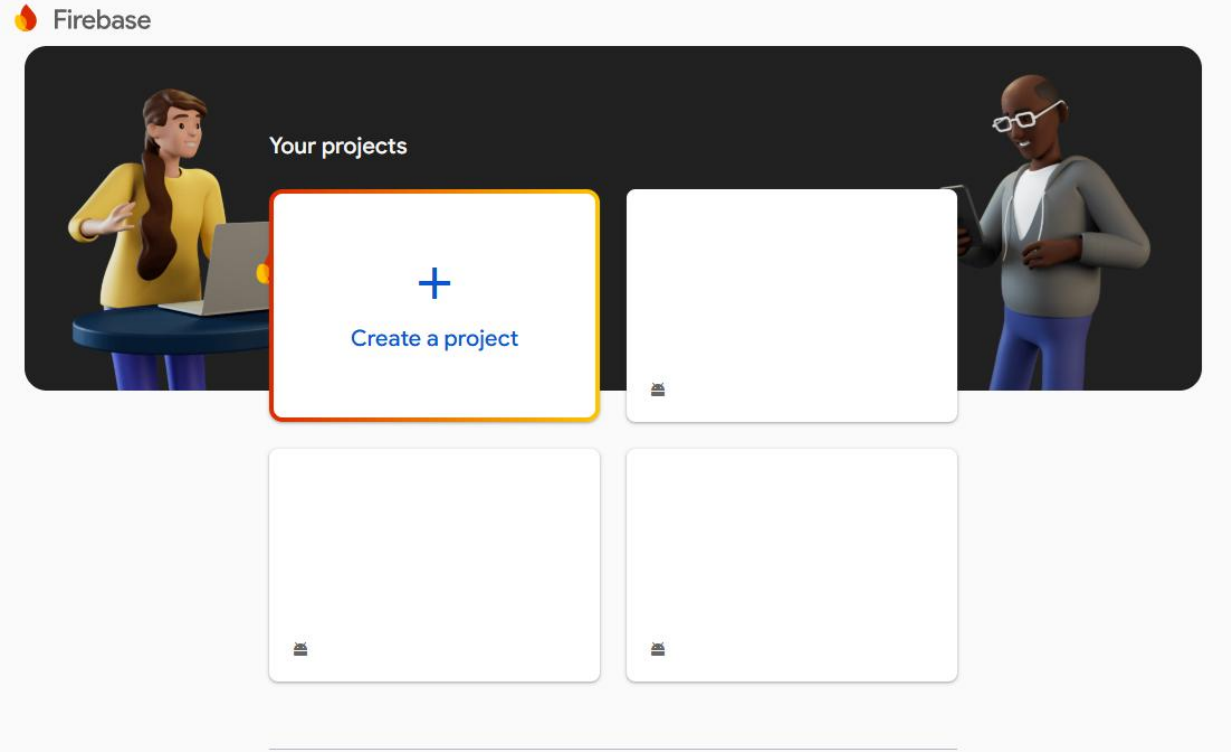
1.3. FireBase

Firebase je razvijen kao platforma od strane Googlea i koristi se za jednostavnu implementaciju udaljene pohrane podataka i funkcionalnosti u mobilnim aplikacijama. Podaci se mogu spremati i sinkronizirati u stvarnom vremenu putem Cloud Firestorea ili Realtime Databasea, ovisno o potrebama aplikacije. Osim pohrane, Firebaseom se nude usluge poput autentifikacije korisnika, analitike i push notifikacija. Ova platforma je dizajnirana kako bi se olakšao razvoj aplikacija bez potrebe za vlastitim poslužiteljem, što je čini idealnom za brzo postavljanje i testiranje projekata.

2. Rad na vježbi

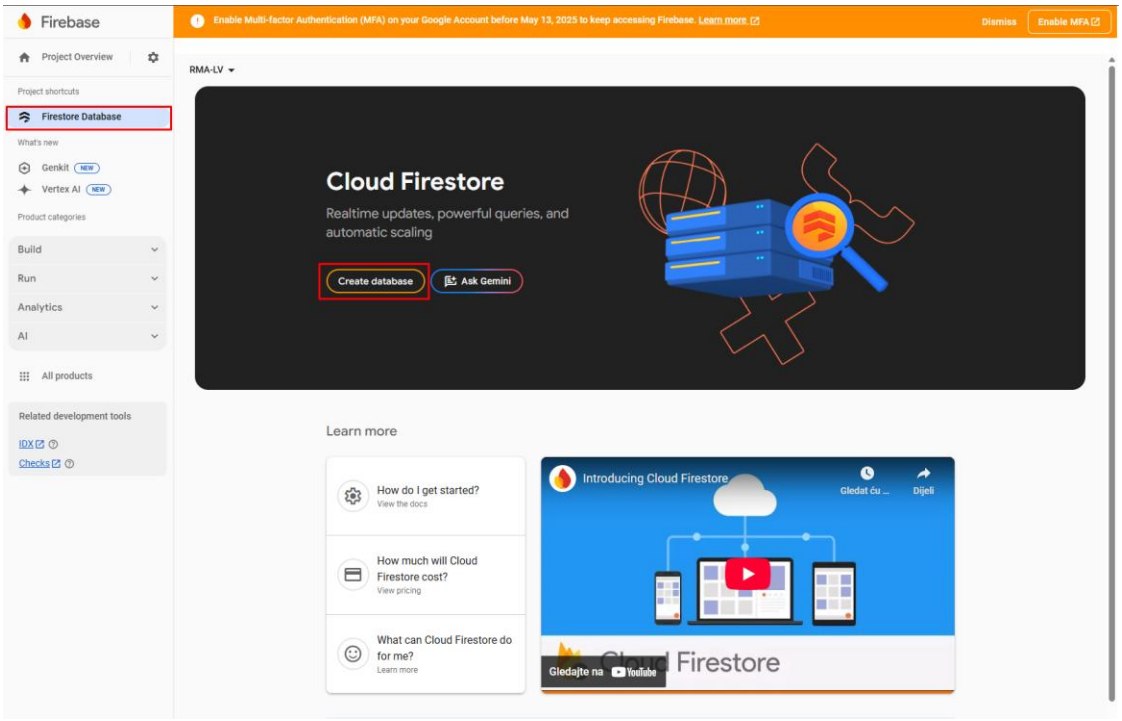
Aplikacija s prethodnih vježbi će poslužiti kao početna točka za trenutnu vježbu. U prošloj vježbi su se podatci unosili „ručno“ unutar composable funkcije. Na ovoj vježbi će biti prikazano kako dohvaćati i slati podatke na udaljenu bazu podataka.

Prvi korak u kreiranju jest otvoriti Firebase Console putem poveznice. Otvoriti će se prozor prikazan na slici 1. te je potrebno odabrati **Create a project** i unijeti ime projekta i onemogućiti google analitiku, jer neće biti potrebna za izradu ove vježbe.



Slika 1. *Kreiranje novog firebase projekta*

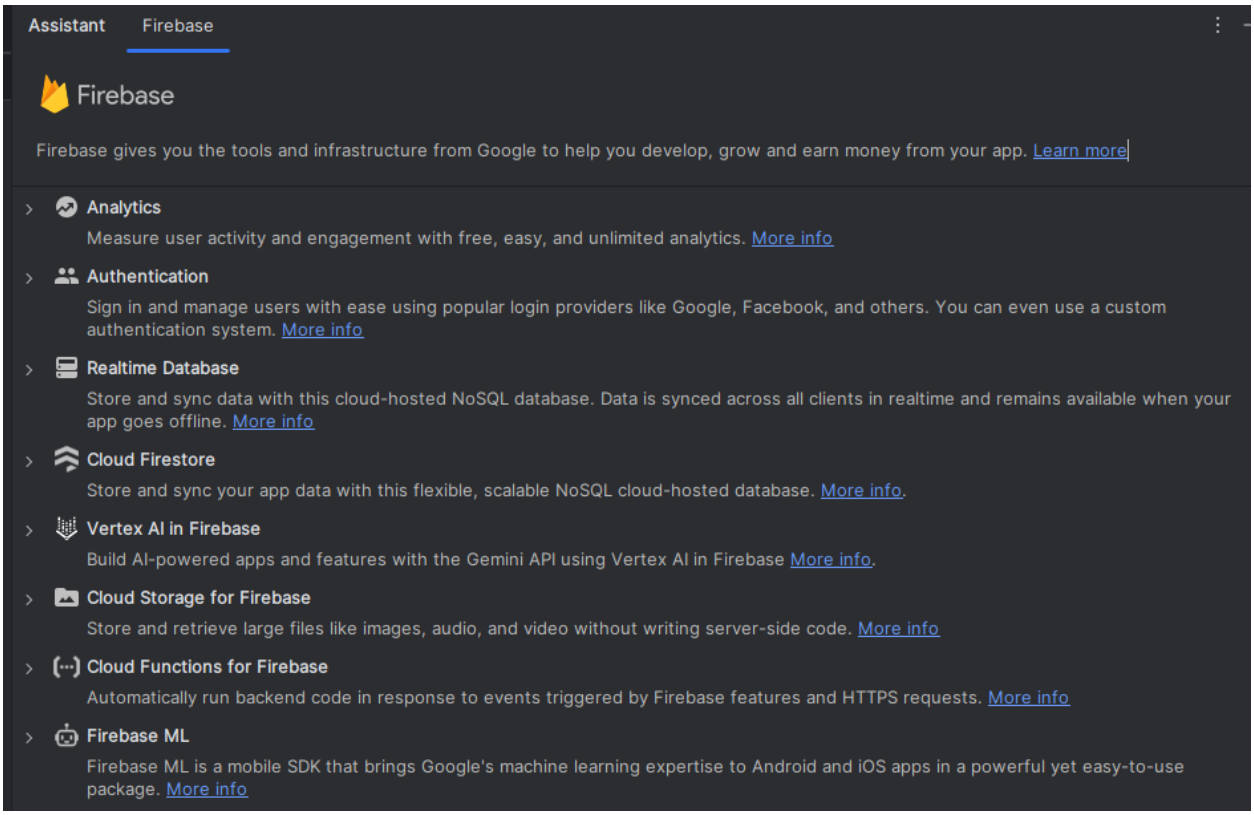
Nakon kreiranja projekta, potrebno je odabrati **Firestore Database** i **Create database**, prikazano na slici 2.



Slika 2. *kreiranje firesotre baze podataka*

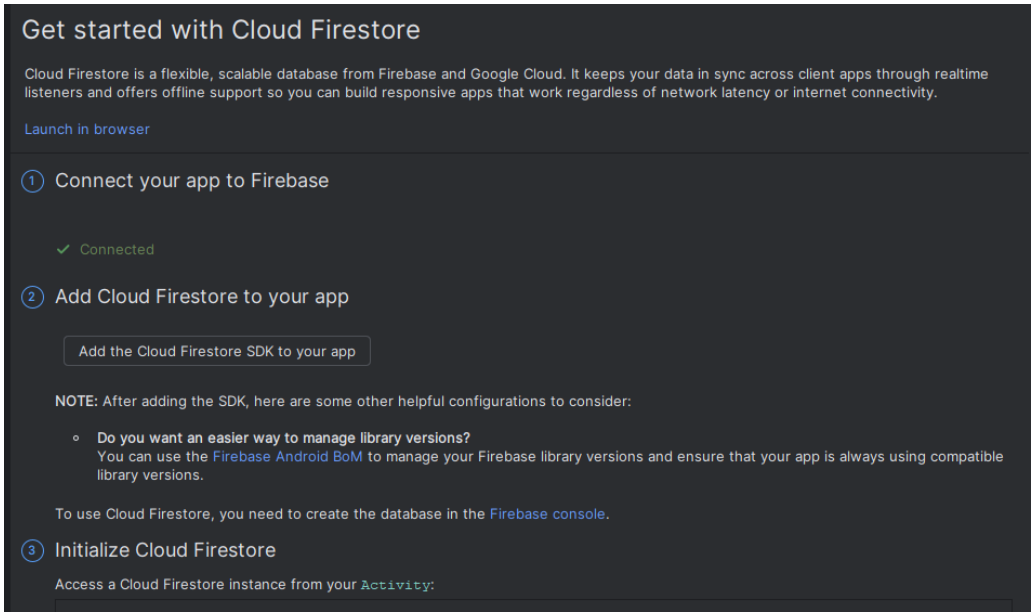
U konfiguracijskom izborniku, za lokaciju poslužitelja je potrebno postaviti **Frankfurt** i odabrati **test mode**. U test modu, baza se postavlja s pravilima koja omogućuju svima (autentificiranim i neautentificiranim korisnicima) čitanje i pisanje podataka, ali ta pravila imaju rok trajanja od 30 dana. Nakon tog razdoblja, pristup bazi bit će onemogućen ako se pravila ne promijene.

Nakon kreiranja baze podataka potrebno ju je povezati sa projektom unutar android studija. U Android studiju odabrati Tools → FireBase. S desne strane ekrana pojaviti će se izbornih prikazan na slici 3.



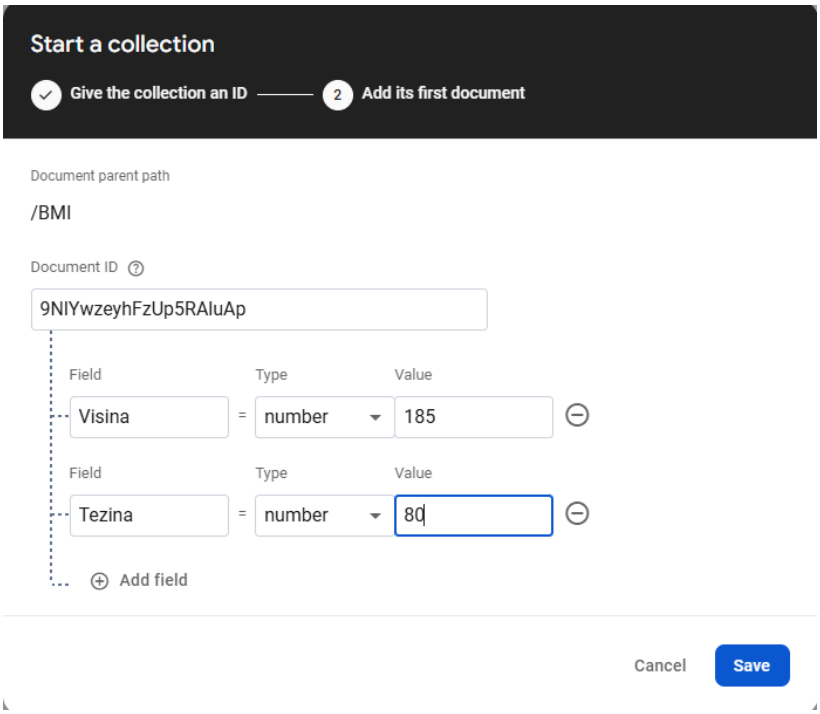
Slika 3. *Firebase izbornik*

U ovom izborniku potrebno je odabrati **Cloud Firestore**, nakon čega je potrebno odabrati **Get started with Cloud Firesotre**. U izborniku koji se nakon toga otvori prvo kliknuti na **Connect to Firebase.** Nakon povezivanja andorid projekta sa firebase bazom podataka, potrebno je odabrati **Add the cloud firestore SDK to your app** i **Accept changes** kako bi omogućili Android studiju da automatski ažurira vaše Gradle datoteke. U ovome koraku će čarobnjak automatski postaviti sve potrebne biblioteke i pripremiti vaš projekt za korištenje Cloud Firestorea. Cloud Firestore je noSQL baza podataka i svoje podatke sprema u kolekcije i dokumente.



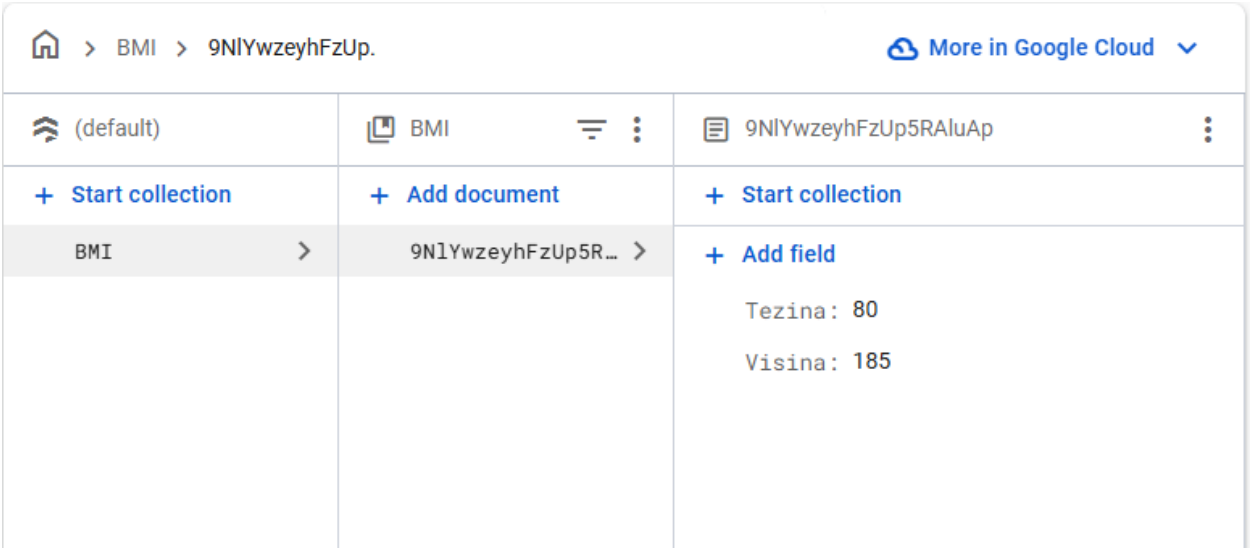
Slika 4. *Povezivanje Android studija s Firebasom*

Potrebno je u Firebase konzoli napraviti kolekciju naziva **BMI**. Document ID odredite klikom na Auto-ID. Zatim napravite polje Visina u koju upišite vrijednost te Težina kojoj možete opcionalno dodijeliti neku vrijednost. Ovaj postupak je prikazan slikom 5.



Slika 5. Nova Firestore kolekcija

U kreiranoj bazi podataka sada možemo vidjeti unos prikazan slikom 6.



Slika 6. *Početno stanje Firestore baze podataka*

Sada je potrebno vratiti se u Android Studio, gdje će se nastaviti s radom na integraciji projekta. Potrebno je pronaći datoteka **AndroidManifest.xml**. Ova datoteka smatra se glavnom datotekom koja se koristi za opisivanje bitnih informacija o Android aplikaciji. Te informacije koriste se od strane alata za izradu, Android operativnog sustava i Google Play trgovine kako bi se razumjelo što aplikacija predstavlja, kako funkcionira i koji resursi su joj potrebni. Više o ovoj datoteci možete pronaći ovdje.

Kako bi naša aplikacija mogla komunicirati sa Firebaseom, potrebno joj je dati dopuštenje pristupa internetu. Na slici 7 prikazan je AndroidManifest.xml u kojem je iznad application elementa dodan:

```
<uses-permission android:name="android.permission.INTERNET" />
```

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
```

Slika 7. *Dopuštenje pristupa internetu kroz AndroidManifest.xml*

Također u **build.gradle.kts (Module: app)** je potrebno dodati:

```
implementation("com.google.firebase:firebase-firestore:24.9.1")

implementation("androidx.activity:activity-ktx:1.8.0")

implementation("androidx.fragment:fragment-ktx:1.8.6")

implementation("io.coil-kt:coil-compose:2.5.0")
```

Slika 8. *Promjena build.gradle.kts datoteke*

Nakon promjene ove datoteke, obavezno odabbrati Sync now kako je prikazano na slici 8.

Varatimo se sada u MainActivity.kt u kojem ćemo definirati randnje na aplikaciji i bazi podataka. Prvo, u composable funkciju UserPreview potrebno je deklarirati vrijednost *db* koja će držati referencu na bazu podataka i ona omogućuje komunikaciju s Firestore-om. Možemo je koristiti za spremanje ili dohvaćanje podataka.

```
val db = FirebaseFirestore.getInstance()
```

Programski kod 1. *Deklaracija db vrijednosti*

Nadalje, trenutno težina i visina (weight i height) dolaze kao fikse vrijednosti iz parametara. Želimo da se one mogu mijenjati, stoga je potrebno kreirati varijable koje se mogu ažurirati. Također nam treba varijabla za novu težinu koju korisnik unese. Za to koristimo *remember* i *mutableStateOf* jer Compose prati promjene u ovim varijablama i automatski osvježava ekran kad se promijene. Ispod val db potrebno je dodati programski kod 2. U row elemente za ispis korisnikove visine i težine potrebno je umjesto *weight* i *height* postaviti *currentWeight* i *currentHeight.*

```
var currentWeight by remember { mutableStateOf(weight) }
var currentHeight by remember { mutableStateOf(height) }

var newWeight by remember { mutableStateOf(0f) }
var newHeight by remember { mutableStateOf(0f) }
```

Programski kod 2. *Deklaracija db vrijednosti*

Sada ćemo dodati mogućnost da korisnik unese svoju težinu pomoću *TextField*-a. *TextField* je Compose funkcija koja stvara polje za unos teksta, gdje korisnik može upisati podatke, u ovom slučaju težinu. Međutim, *TextField* radi samo s tekstom, odnosno tipom podataka *String*, a ne izravno s brojevima poput težine koja je broj (tip *Float*). Zato nam treba posebna varijabla koja će pratiti tekst koji korisnik upisuje. Tu varijablu ćemo nazvati *weightInput* jer će sadržavati unos težine kao tekst. Deklaracija varijable *weightInput* je prikazana programskim kodom 3.

```
        var weightInput by remember { mutableStateOf("") }
```

Programski kod 3. *Deklaracija weightInput varijable*

Ispod zadnjeg *Text* elementa u *Column-u* dodajem *TextField* prema programskom kodu 4.

```
TextField(
    value = weightInput,
    onValueChange = { input -> weightInput = input

        val newValue = input.toFloatOrNull()

        if (newValue!= null && newValue >= 0) {
            newWeight = newValue
        }
    },
    label = { Text("Unesi novu tezinu (kg)") },
    modifier = Modifier
        .padding(top = 16.dp)
        .width(200.dp)
)
```

Programski kod 4. *TextField element*

Parametar *value* određuje koji tekst će biti prikazan u *TextField*-u. On predstavlja trenutnu vrijednost polja, odnosno ono što korisnik vidi dok upisuje ili što je već uneseno. Parametru

onValueChange predaje se lambda funkcija koja se aktivira pri svakoj promjeni unosa. Unutar nje, *weightInput* se ažurira s novim tekstom (*input*), a zatim se taj tekst pretvara u broj metodom *toFloatOrNull*(), spremajući rezultat u *newValue*. Ako je *newValue* valjan broj i veći ili jednak nuli, ažurira se *newWeight*.

Nakon što smo omogućili unos nove težine putem *TextField*-a, sljedeći korak je dodavanje gumba (*Button*) koji će ažurirati lokalnu vrijednost težine te je spremiti u bazu podataka. Dodavanje gumba prikazano je programskim kodom 5.

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

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

Programski kod 5. *Button element*

Gumb se definira pomoću komponente *Button* u Jetpack Composeu. Parametar *onClick* sadrži logiku koja se izvršava kada korisnik pritisne gumb. Prvo, provjeravamo jeli unesena nova vrijednosti za težinu (*newWeight*) veća od nule. Ako je, ažuriramo lokalnu varijablu *currentWeight* s tom vrijednosti, što osvježava prikaz na ekranu.

Zatim, kreiramo vrijednost *docRef* koja predstavlja referencu na određeni dokument u kolekciji "BMI". Metoda *update* ažurira polje "Tezina" u tom dokumentu s novom vrijednosti newWeight, koristeći mapu (*mapOf*) koja stvara par ključ-vrijednost za definiranje podataka.

Ukoliko ste dobro popratili sve korake, aplikacija bi trebala izgledati kao na slici 9.

Slika 9. *Izgled dovršene aplikacije*

3. Zadatak za samostalni rad

1. Napraviti *TextField* za unos visine, te ažurirati vrijednost korisnikove visine na zaslonu (za minimalni broj bodova)
2. Ažurirati vrijednost visine u Firestore bazi, te na temelju nove visine i težine izračunati te prikazati novi BMI.

Nakon odrađenih zadataka, aplikacija bi trebala izgledati kao na slici 10.

Slika 10. *Izgled dovršene aplikacije*