RAZVOJ MOBILNIH APLIKACIJA:

LV 4: Senzori i notifikacije

1. Senzori

Većina Android uređaja sadrži ugrađene senzore koji mjere pokrete, orijentaciju i brojne uvjete okoline. Ovi senzori daju veliku količinu sirovih podataka sa velikom preciznošću. Možemo ih koristiti prilikom razvoja aplikacija. Najčešće se koriste kod npr. Igranja igrica kada daju podatke iz korisnikovih pokreta poput trešnje, okretanja ili mahanja. Aplikacije koje daju podatke o vremenu koriste temperaturne senzore i senzore vlage. Aplikacije za putovanje koriste senzor magnetskog polja i akcelerometar.

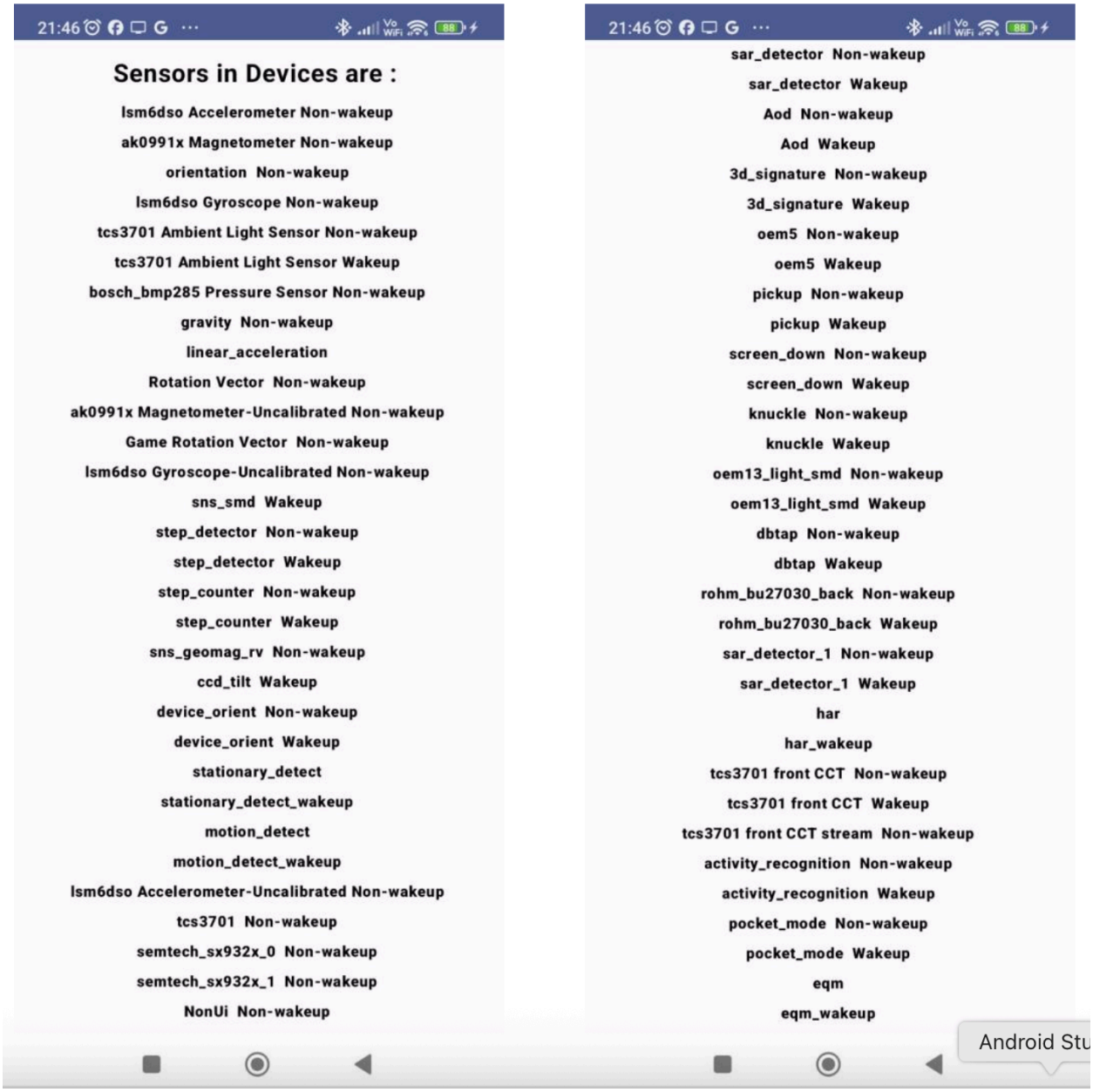
Android platforma razlikuje 3 vrste senzora:

- Senzori pokreta
- Senzori okoline
- Senzori pozicije

Kako bismo što bolje koristili senzore i njihove podatke u razvoju Android aplikacija, moramo najprije doznati koje senzore uređaj sadrži a potom i ustvrditi mogućnosti pojedinog senzora. Senzori mogu biti hardverski ili softverski.

Više o senzorima pogledati na poveznici: [Obavezno]

Github repozitorij – kako dobiti listu senzora [Obavezno]



Slika 1. Popis dostupnih senzora na mobilnom uređaju

2. Notifikacije

Notifikacije u Androidu su obavijesti koje se prikazuju na zaslonu vašeg uređaja kako bi vam obavijestile o novim događajima ili aktivnostima u aplikaciji. Ove obavijesti mogu uključivati informacije o novoj poruci, propuštenom pozivu, novoj e-pošti, obavijestima iz društvenih mreža i drugim aplikacijama. Notifikacije u Androidu se mogu prikazati u različitim oblicima, poput ikona u traci s obavijestima, obavijesti na zaključanom zaslonu, padajućih obavijesti ili obavijesti koje se pojavljuju na vrhu zaslona. Ovisno o vrsti obavijesti, korisnik ih može primati od aplikacije, sustava, mreže ili drugih izvora. Uglavnom se koriste u svrhu obavještavanja kada ne koristimo aplikaciju.

3. Rad na vježbi

Cilj današnje vježbe je izmjeriti broj koraka putem senzora uređaja. Broj koraka ćemo prikazati na zasebnom zaslonu.

3.1. Rad na više zaslona

Da bismo uveli drugi zaslon u aplikaciju, krenuti ćemo sa implementacijomu Navigation Grapha. Potrebno je dodati implementaciju u build.gradle.kts.

```
implementation("androidx.navigation:navigation-compose:2.7.5")
```

Programski kod 1. *Implementacija Navigation Grapha*

Zatim je potrebno dodati :

```
import androidx.navigation.NavController

import androidx.navigation.compose.rememberNavController

import androidx.navigation.compose.NavHost

import androidx.navigation.compose.composable
```

Programski kod 2. *Implementacija Navigation Grapha*

vrijednosti navController u MainActivity. Ispod linije setContent { upisati programski kod 3.

```
val navController = rememberNavController()
```

Programski kod 3. *Deklaracija navControllera*

U setContentu mijenjamo UserPreview() sa programskim kodom 4.

```
NavHost(navController = navController, startDestination = "main_screen",
modifier = Modifier.padding(innerPadding)
) { composable("main_screen") {
MainScreen(navController = navController)
}
composable("step_counter") {
StepCounter(navController = navController)
}
 }
```

Programski kod 4. *Postavljanje NavHosta*

Ovime smo našem projektu rekli da mu je početni ekran naziva „main_screen“ a sa njega se može prijeći na ekran step_counter.

Sada je potrebno definirati ova dva ekrana. Prvi ekran je cijeli UserPreview() sa dodanim gumbom za navigaciju te će nam ta nova composable funkcija izgledati ovako:

```
@Composable
fun MainScreen(navController: NavController, modifier: Modifier = Modifier) {
    Box( modifier = modifier
        .fillMaxSize()){
        UserPreview(
        heightCm = 191,
        weightKg = 100,
        modifier = Modifier.fillMaxSize()
    )
        Button(
            onClick = { navController.navigate("step_counter") },
            modifier = Modifier
                .align(Alignment.BottomEnd)
                .padding(16.dp)
        ) {

            Text("Idi na brojač koraka")
        }}
}
```

Programski kod 5. MainScreen ekran

StepScreen ekran će izgledati ovako:

```
fun StepCounter(navController: NavController, modifier: Modifier = Modifier) {
    Box(
        modifier = modifier
            .fillMaxSize()
            .background(Color.White)
    ) {
        Image(
            painter = painterResource(id = R.drawable.fitness),
            contentDescription = "Pozadinska slika",
            contentScale = ContentScale.Crop,
            alpha = 0.1f,
            modifier = Modifier.fillMaxSize()
        )
            Text("Ekran za brojač koraka")
            Spacer(modifier = Modifier.height(16.dp))
            Button(
                onClick = { navController.navigate("main_screen") },
                modifier = Modifier
                    .align(Alignment.BottomEnd)
                    .padding(16.dp)
            ) {

            Text("Idi na glavni ekran")
            }
        }

}}
```

Programski kod 6. StepScreen ekran

3.2. Dodavanje senzora

Senzore je relativno lagano dodati u projekt. Prvo dajemo dopuštenje u manifestu programskim kodom 7.

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

Programski kod 7. Uvođenje dopuštenja u projekt

Potom je potrebno u projekt uvesti pakete:

```
import android.hardware.Sensor
 import android.hardware.SensorEvent
import android.hardware.SensorEventListener
 import android.hardware.SensorManager
```

Programski kod 8. *Importi potrebni za korištenje senzora*

```
val sensorManager =
(LocalContext.current.getSystemService(Context.SENSOR_SERVICE) as
SensorManager)
```

Programski kod 9. Deklaracija Sensor Managera

Ova linija koda dobavlja instancu SensorManager iz sustava usluga. SensorManager je klasa koja pruža pristup senzorima uređaja. LocalContext.current se koristi za dobavljanje trenutnog konteksta aplikacije, a zatim se poziva getSystemService() metoda na njemu s argumentom Context.SENSOR_SERVICE kako bi se dobio SensorManager. Nakon toga, rezultat se casta u SensorManager tip kako bi se dobio pristup metodama i funkcijama koje pruža ta klasa.

Nakon ovog koraka odabiremo senzor koji ćemo koristiti. Zadatak je izmjeriti broj koraka pa ćemo sukladno tome koristiti senzor Step Counter.

```
val stepCounter = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER)
```

Programski kod 10. Dodavanje brojača koraka

Nakon što je instanca SensorManager-a dohvaćena i step counter senzor identificiran, sljedeći korak u radu sa senzorima jest kreiranje sučelja SensorEventListener. Ovo sučelje igra ključnu ulogu jer definira kako će aplikacija reagirati na podatke koje senzor šalje. Ono omogućuje praćenje detektiranih koraka u stvarnom vremenu, što je temelj za daljnju obradu i prikaz u korisničkom sučelju.

SensorEventListener je sučelje koje zahtijeva implementaciju dviju metoda: onSensorChanged() i onAccuracyChanged(). Ove metode omogućuju aplikaciji da odgovori na nove podatke senzora i promjene u njihovoj točnosti.

onSensorChanged(event: SensorEvent?)

Ova metoda se poziva svaki put kada step counter senzor detektira novi broj koraka. Parametar event tipa SensorEvent sadrži informacije o događaju, uključujući:

- event.values: Polje tipa FloatArray koje sadrži ukupan broj koraka od posljednjeg resetiranja uređaja (npr. nakon restartanja).
- event.sensor: Referenca na senzor koji je generirao događaj.
- event.accuracy: Trenutna točnost mjerenja.

Za Sensor.TYPE_STEP_COUNTER, event.values[0] predstavlja ukupan broj koraka od posljednjeg pokretanja uređaja.

onAccuracyChanged(sensor: Sensor?, accuracy: Int)

Ova metoda se poziva kada se promijeni točnost senzora (npr. zbog kalibracije ili vanjskih uvjeta). Parametri su:

- sensor: Senzor čija se točnost promijenila (može biti null ako nije dostupan).
- accuracy: Nova razina točnosti, koja može biti:
  - SensorManager.SENSOR_STATUS_UNRELIABLE
  - SensorManager.SENSOR_STATUS_ACCURACY_LOW
  - SensorManager.SENSOR_STATUS_ACCURACY_MEDIUM
  - SensorManager.SENSOR_STATUS_ACCURACY_HIGH

Ova metoda nije uvijek potrebna za osnovne primjene i može ostati prazna ako točnost nije kritična za funkcionalnost aplikacije, što će biti slučaj u ovoj vježbi.

Sada možemo postaviti i sensorListener kako je prikazano kodom 11.

```
val sensorListener = object : SensorEventListener {
    override fun onSensorChanged(event: SensorEvent) {
        if (event.sensor.type == Sensor.TYPE_STEP_COUNTER) {
            val brojKoraka = event.values[0]
            // Obrada broja koraka, npr. prikaz u sučelju
        }
    }

    override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {
        // Ova metoda može ostati prazna
    }
}
```

Programski kod 11. sensorListener za step counter

Registracija i odjava slušača (Listener) ključni su za upravljanje podacima senzora u Android aplikaciji, jer osiguravaju da aplikacija prima podatke samo kada je to potrebno i da se resursi oslobađaju kada se senzor više ne koristi. Registracija sensorListenera vrši se pozivom metode registerListener() na instanci SensorManager-a. Ova metoda povezuje SensorEventListener sa senzorom i započinje slanje podataka prema definiranoj brzini osvježavanja. Programski kod 4. prikazuje primjer registracije za stepCounter.

```
sensorManager.registerListener(sensorListener, stepCounter,
SensorManager.SENSOR_DELAY_NORMAL)
```

Programski kod 12. sensorListener za step counter

Odjava slušača jednako je važna kao i registracija, jer osigurava da senzor prestane slati podatke kada više nije potreban, čime se štede resursi uređaja poput baterije i procesorske snage. Primjer odjave je prikazan programskim kodom 13.

```
sensorManager.unregisterListener(sensorListener)
```

Programski kod 13. Odjava Listenera

U ovoj vežbi za registraciju i odjavu Listenera korist će se DisposableEffect Blok. DisposableEffect je specijalizirani efekt u Jetpack Compose-u koji se koristi za upravljanje resursima ili radnjama koje imaju jasno definiran početak i kraj, odnosno zahtijevaju "čišćenje" nakon što prestanu biti potrebni. DisposableEffect omogućuje izvođenje koda pri ulasku composable-a u kompoziciju (ekvivalent "startu") i oslobađanje resursa pri njegovom izlasku iz kompozicije (ekvivalent "stopu").

U kontekstu rada sa senzorima, DisposableEffect je idealan za registraciju i odjavu slušača (SensorEventListener).

Registracija: Kod unutar DisposableEffect bloka izvršava se kada composable postane dio korisničkog sučelja, što je trenutak kad želimo započeti praćenje podataka senzora (slično onStart u tradicionalnim aktivnostima).

Odjava: onDispose blok unutar DisposableEffect-a pokreće se kada composable napusti kompoziciju (npr. kada korisnik ode na drugi zaslon), što je trenutak kad želimo prestati pratiti senzor i osloboditi resurse (slično onStop). Ovo osigurava da senzor ne ostane aktivan u pozadini, čime se sprječava nepotrebna potrošnja baterije i curenje resursa, što je ključno za optimizaciju performansi aplikacije. Primjer korištenja DisposableEffect-a prikazan je programskim kodom 14.

```
DisposableEffect(Unit) {
    val stepSensor = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER)
    if (stepSensor != null) {
        sensorManager.registerListener(stepListener, stepSensor,
SensorManager.SENSOR_DELAY_UI)
        Log.d("step_counter", "Step Counter senzor je registriran.")
    } else {
        Log.d("step_counter", "Step Counter senzor nije dostupan.")
    }

    onDispose {
        sensorManager.unregisterListener(stepListener)
        Log.d("step_counter", "Step Counter senzor je odjavljen.")
    }
}
```

Programski kod 14. DisposableEffect

3.3. Notifikacije

Kao i za korištenje senzora, potrebno je dodati dozvolu za notifikacije u AndroidManifest.xml:

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

Programski kod 15. Dozvola za notifikacije

Te sljedeću ovisnost u build.gradle.kts

```
implementation ("androidx.core:core:1.12.0")
```

Programski kod 16. Ovisnost za build.gradle.kts

NotificationManager je klasa koja omogućuje aplikacijama da komuniciraju s korisnikom putem vizualnih obavijesti koje se prikazuju u statusnoj traci uređaja. Korištenjem NotificationManager klase, aplikacija može generirati, prikazati, ažurirati ili ukloniti notifikacije, čime se korisniku prenose važne informacije, obavijesti ili podsjetnici bez potrebe za njegovom trenutnom interakcijom s aplikacijom. Kreiranje NotificationManager je prikazano programskim kodom 17.

```
val notificationManager = context.getSystemService(Context.NOTIFICATION_SERVICE) as
NotificationManager
```

Programski kod 17. Notification manager

NotificationCompat.Builder je klasa za izradu i konfiguriranje notifikacija. Ova klasa dolazi iz Android Jetpack biblioteke i omogućava jednostavno stvaranje notifikacija koje će raditi na svim verzijama Android operativnog sustava. NotificationCompat.Builder nudi širok spektar funkcija za prilagodbu izgleda i ponašanja notifikacija. Primjer korištenja prikazan je programskim kodom 18.

```
val notification = NotificationCompat.Builder(context, "channelId")
.setSmallIcon(R.drawable.ic_dialog_info) // Mala ikona .setContentTitle("Naslov") // Naslov notifikacije
.setContentText("Ovo je tekst notifikacije.") // Tekst notifikacije
.setPriority(NotificationCompat.PRIORITY_HIGH) // Postavljanje prioriteta
.build()
notificationManager.notify(1, notification) // Slanje notifikacije
```

Programski kod 18. NotificationCompat.Builder

Kanal notifikacija je komponenta Android operativnog sustava koja omogućava aplikacijama da organiziraju i upravljaju notifikacijama na temelju njihovih vrsta, važnosti i postavki koje korisnik odabere. Uveden je u Android 8.0 (API level 26) i postao je obavezan za sve aplikacije koje šalju notifikacije. Kanali notifikacija omogućuju korisnicima veću kontrolu nad načinom na koji primaju obavijesti. Kroz kanale, korisnici mogu birati hoće li primati obavijesti s određenih kategorija, hoće li one biti zvučne, vibrirajuće, tihe ili uopće isključene. Kanal notifikacije mora biti kreiran prije nego što se notifikacija može poslati. Ovo uključuje određivanje naziva kanala, njegove važnosti (prioriteta) i drugih karakteristika. Svaka notifikacija može pripadati samo jednom kanalu, a korisnik može promijeniti postavke tog kanala prema svojim željama. Programskim kodom 19. prikazan je primjer kreiranja kanala notifikacija.

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        val channelId = "steps_channel"
        val channelName = "Steps Notifications"
        val channel = NotificationChannel(channelId, channelName,
        NotificationManager.IMPORTANCE_DEFAULT)
        notificationManager.createNotificationChannel(channel)
}
```

Programski kod 19. Kanal notifikacija

Uvjet provjerava je li trenutna verzija Android operativnog sustava na uređaju veća ili jednaka od verzije 8.0 (Android O). Naime, kanali notifikacija su uvedeni u Android 8.0, pa je za uređaje sa starijim verzijama sustava potrebno koristiti alternativne metode za slanje notifikacija. U slučaju da je uređaj na Androidu 8.0 ili novijem, slijedi kreiranje kanala za notifikacije. U ovom procesu prvo se definira jedinstveni identifikator kanala (channelId), koji služi za prepoznavanje i razlikovanje različitih kanala unutar aplikacije. U našem primjeru, channelId je postavljen na "steps_channel".

Zatim se postavlja naziv kanala (channelName), te kreiranje instance kanala obavlja se pomoću klase NotificationChannel, kojoj se prosljeđuju ID kanala, naziv kanala te prioritet obavijesti. U ovom primjeru, prioritet kanala je postavljen na IMPORTANCE_DEFAULT, što znači da će notifikacije s ovog kanala imati srednji prioritet i neće biti previše upadljive, ali će i dalje biti prikazane korisnicima. Nakon što je kanal stvoren, on se registrira u sustavu putem NotificationManager-a pomoću metode createNotificationChannel(channel), čime postaje aktivan i spreman za slanje notifikacija korisnicima. 3. Dozvola za praćenje aktivnosti i notifikacije Da bi aplikacija ispravno funkcionirala potrebno je zatražiti odgovarajuće dozvole od korisnika. Naime, Android sustav, radi zaštite privatnosti i sigurnosti korisnika, automatski ne dopušta aplikacijama pristup određenim osjetljivim podacima i funkcijama, kao što su praćenje aktivnosti

korisnika ili slanje notifikacija. Stoga, funkciju prikazanu programskim kodom 10. potrebno je dodati u MainActivity klasu, te ju pozvati prije setContent bloka, kako bi se prilikom pokretanja aplikacije zatražila dozvola od korisnika za praćenje aktivnosti i slanje notifikacija.

```
private fun requestPermission() {
        val permissionsToRequest = mutableListOf()
// Dozvola za praćenje aktivnosti (koraci)
if (ContextCompat.checkSelfPermission(
this,
Manifest.permission.ACTIVITY_RECOGNITION )
 != PackageManager.PERMISSION_GRANTED ) {
 permissionsToRequest.add(Manifest.permission.ACTIVITY_RECOGNITION) }
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
        if (ContextCompat.checkSelfPermission(
        this,
        Manifest.permission.POST_NOTIFICATIONS ) !=
        PackageManager.PERMISSION_GRANTED ) {
        permissionsToRequest.add(Manifest.permission.POST_NOTIFICATIONS)
        }
         } if (permissionsToRequest.isNotEmpty()) {
        ActivityCompat.requestPermissions(this, permissionsToRequest.toTypedArray(), 1)
        } }
```

Programski kod 20. requestPermission funkcija

ZADATAK: Brojač koraka s notifikacijom i pohranom u Firestore

Proširiti prethodnu aplikaciju tako da se na posebnom ekranu stepScreen prikaže broj prijeđenih koraka. Budući da tablet nema Step Counter senzor, potrebno je koristiti **akcelerometar** kako bi se detektirali koraci. Kada korisnik napravi **više od 50 koraka**, aplikacija treba **prikazati lokalnu notifikaciju** te **pohraniti broj koraka u Firebase bazu**.

Zahtjevi

1. Ekran s prikazom broja koraka

- Kreirati novi ekran stepScreen.
- Prikazati tekstualno broj detektiranih koraka: Broj koraka: 23.

2. Korištenje akcelerometra

- Umjesto TYPE_STEP_COUNTER koristiti Sensor.TYPE_ACCELEROMETER.
- Implementirati **detekciju koraka** pomoću promjene ubrzanja (npr. $\text{sqrt}(x^2 + y^2 + z^2)$).
- Svaki put kada se zadovolji uvjet "hodanja", povećati broj koraka (stepCount++).
- Preporučeno: detektirati korak kada magnitude > 12f.

3. Lokalna notifikacija

- Kada korisnik napravi **više od 50 koraka**, aplikacija mora:
  - Prikazati **notifikaciju** s porukom (npr. "Bravo! Napravili ste više od 50 koraka!").
  - Notifikacija mora sadržavati: naslov, tekst i ikonu.

4. Firebase Firestore

- Po prelasku 50 koraka, potrebno je:
  - Pohraniti dokument u kolekciju "Koraci".
  - Dokument treba sadržavati:
    - "koraci": broj koraka
    - "timestamp": FieldValue.serverTimestamp()
- Svaki put kada korisnik prijeđe 50, 100, 150... koraka – pohraniti novi zapis.

Napomene

- Akcelerometar treba registrirati unutar `DisposableEffect`.
- Koristiti `mutableStateOf` za prikaz broja koraka.
- Notifikacije se šalju pomoću `NotificationCompat.Builder`.
- Podatke u Firestore pohranjujte samo **jednom po svakih 50 novih koraka**, a ne nakon svakog.