ANDROID PROGRAMIRANJE

Vežba 2: Razvoj *WeatherAlert* aplikacije

(radi se dve nedelje)

Teorijske napomene

Za ovu vežbu koristimo **Layoute** (o kojima je bilo reči i na prethodnoj vežbi, ali sada je priložen detaljniji opis), kao i dve nove komponente – **Servise i Broadcast Receivere.** Pažljivo ispratite teorijske napomena a onda proučite i sve neophodne zahteve za razvoj tražene aplikacije.

Layouti u Androidu

Layouti su osnova svakog korisničkog interfejsa u Android aplikacijama. Oni definišu strukturu korisničkog interfejsa i način na koji će elementi biti prikazani na ekranu. Android nudi nekoliko različitih vrsta layouta koje možemo koristiti za kreiranje fleksibilnog i prilagodljivog dizajna.

LinearLayout

U LinearLayoutu svi elementi su postavljeni u jednom pravcu – horizontalno ili vertikalno. Svi elementi se nižu jedan ispod drugog (ili pored) redosledom kojim su definisani. Koristi se kada želite da jednostavno postavite elemente u red ili kolonu.

Primer:

```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Naslov"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Klikni"/>
</LinearLayout>
```

RelativeLayout

U RelativeLayoutu elementi se postavljaju relativno jedni prema drugima ili prema roditeljskom layoutu. Ovo omogućava veću fleksibilnost u odnosu na LinearLayout, jer elementi mogu biti pozicionirani, na primer, ispod ili pored drugog elementa. Koristi se za malo složeniji raspored elemenata kada želite da elementi budu postavljeni u odnosu na druge elemente ili ivice ekrana.

Primer:

```
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Naslov"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/title"
        android:text="Klikni"/>
</RelativeLayout>
```

GridLayout

GridLayout je grafički raspored koji organizuje elemente u mrežu redova i kolona. Omogućava postavljanje elemenata unutar ćelija, što ga čini pogodnim za kreiranje složenih i simetričnih interfejsa. Možete precizno definisati broj redova i kolona, a elementi mogu zauzimati više od jedne ćelije koristeći **layout_rowSpan** ili **layout_columnSpan**.

Primer:

```
<GridLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:rowCount="2"
    android:columnCount="2"
    android:padding="16dp">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Naslov"
```

```
        android:layout_row="0"
        android:layout_column="0"
        android:layout_gravity="center"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Klikni"
        android:layout_row="0"
        android:layout_column="1"
        android:layout_gravity="center"/>
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_launcher_foreground"
        android:layout_row="1"
        android:layout_column="0"
        android:layout_gravity="center"/>
    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="Unesi tekst"
        android:layout_row="1"
        android:layout_column="1"
        android:layout_gravity="center"/>
</GridLayout>
```

StackLayout

StackLayout organizuje elemente tako da su složeni jedan iznad drugog, što podseća na slojeve. Ovaj raspored je koristan kada želite da prikažete više sadržaja na istom mestu ili kada želite da elementi budu raspoređeni u vertikalnom ili horizontalnom nizu, pri čemu svaki sledeći element delimično prekriva prethodni. Na primer, StackLayout se može koristiti za prikaz informacija koje se preklapaju ili za animacije koje se smenjuju.

Primer:

```
<StackLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Prvi element"
        android:layout_gravity="center"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Drugi element"
        android:layout_gravity="center"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Treći element"
        android:layout_gravity="center"/>
</StackLayout>
```

ConstraintLayout

ConstraintLayout omogućava precizno pozicioniranje elemenata koristeći takozvane restrikcije (constraints). Svaki element se može vezati za druge elemente ili ivice layouta pomoću vertikalnih i horizontalnih ograničenja.

Primer:

```
<ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Naslov"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Klikni"
        app:layout_constraintTop_toBottomOf="@+id/title"
        app:layout_constraintStart_toStartOf="parent"/>
</ConstraintLayout>
```

Kada koristiti koji layout?

- **LinearLayout:** Kada je raspored jednostavan i želite da svi elementi budu u jednoj liniji (horizontalno ili vertikalno). Na primer, ako imate niz dugmadi koja želite postaviti jedno ispod drugog ili pored drugog, LinearLayout je idealan izbor.
- **RelativeLayout:** Kada želite više fleksibilnosti i preciznije pozicioniranje elemenata u odnosu na druge. Na primer, kada vam je potrebno da jedan element bude ispod, iznad, levo ili desno od drugog elementa ili na određenoj udaljenosti od ivica ekrana.
- **GridLayout:** Kada želite organizovati elemente u obliku mreže sa redovima i kolonama. GridLayout je idealan kada imate simetričan raspored ili ako želite da elementi budu postavljeni u ćelije, na primer, raspored dugmadi u aplikaciji kalkulatora ili raspoređivanje slika u mreži.
- **StackLayout:** Kada želite da elementi budu složeni jedan iznad drugog. StackLayout je koristan kada želite prikazati više informacija na istom mestu ili kada želite da elemente postavite u vertikalni ili horizontalni niz, gde svaki element "preklapa" prethodni.
- **ConstraintLayout:** Za malo složeniji raspored i optimizovano prilagođavanje različitim veličinama ekrana. ConstraintLayout pruža napredne opcije pozicioniranja elemenata u odnosu na druge elemente, margine i vodiče, uz veću efikasnost i fleksibilnost, posebno na uređajima sa različitim veličinama ekrana.

Servisi (Services) u Androidu

Servisi su komponente Androida koje omogućuju dugotrajno izvršavanje zadataka u pozadini, čak i kada korisnik nema interakcija direktno sa aplikacijom. Servisi ne sadrže korisnički interfejs i često se koriste za zadatke poput preuzimanja podataka sa servera, slanje obaveštenja ili obrade muzike. Shodno tome, servisi omogućavaju korisnicima da uživaju u funkcionalnosti aplikacije čak i kada su druge aplikacije u fokusu ili kada je ekran isključen.

Tipovi servisa:

- **Foreground Service**: Pokreće se kada aplikacija obavlja zadatak koji je korisniku vidljiv i zahteva aktivnu pažnju, kao što je npr. reprodukcija muzike. Mora prikazivati obaveštenje (notification) dok radi.
- **Background Service**: Ovi servisi obavljaju zadatke koji ne zahtevaju korisničku pažnju (npr. sinkronizacija podataka). Međutim, u novijim verzijama Androida postoje restrikcije za dugotrajne background servise.
- **Bound Service**: Servis koji omogućava komunikaciju između aplikacije i servisa. Ovaj servis se koristi kada aplikacija treba redovno da interaguje sa servisom (npr. za deljenje podataka ili izvođenje zadataka na zahtev).

Ključne metode u servisima:

- onStartCommand(): Ova metoda se poziva svaki put kada se servis pokrene. Tu se obavlja logika zadatka koji servis treba da obavi.
- onBind(): Ova metoda se koristi u **bound** servisima da poveže komponentu sa servisom.
- onDestroy(): Poziva se kada servis prestaje da radi.

Broadcast Receivers u Androidu

Broadcast receiver je bitna komponenta u Androidu koja omogućava aplikacijama da primaju sistemske ili prilagođene poruke (broadcastove). Na primer, Android može poslati poruku kada uređaj završi punjenje baterije, kada se promeni mrežna povezanost (WiFi ili mobilni podaci), ili kada korisnik završi preuzimanje neke datoteke.

Vrste broadcastova:

- **Sistemski broadcastovi**: Android šalje brojne sistemske poruke na koje aplikacije mogu da reaguju, poput promene mrežne veze, pražnjenja baterije, ili promene jezika.
- **Prilagođeni broadcastovi**: Aplikacije mogu same da šalju broadcastove kako bi obavestile druge aplikacije ili komponente o određenim događajima unutar aplikacije.

Kako radi broadcast receiver:

- **Registrovanje**: Receivere možete registrovati u AndroidManifest.xml ili dinamički unutar koda. Kada Android pošalje broadcast, svaki registrovani receiver prima obaveštenje i može reagovati na događaj.
- **Primanje obaveštenja**: Kada sistem pošalje broadcast, svi registrovani broadcast receiveri dobijaju obaveštenje. Oni se aktiviraju kada primaju odgovarajući **Intent*.**
- **Obrada obaveštenja**: Kada receiver primi obaveštenje, koristi se metoda onReceive() za obradu podataka iz Intent objekta.

Ključne metode u broadcast receiveru:

- **onReceive():** Ovo je glavna metoda koja se poziva kada receiver primi broadcast. Unutar ove metode se obrađuje poruka i izvode akcije kao odgovor na događaj.

***Intenti** su objekti koji se koriste za pokretanje raznih aktivnosti, usluga ili za slanje broadcast obaveštenja u Android aplikacijama. Oni su suštinski način komunikacije između različitih komponenti aplikacije ili između različitih aplikacija.

Vrste Intenta:

1. **Implicitni Intenti**: Ovi inteti se koriste kada ne znate tačno koju aktivnost želite da pokrenete, već samo opisujete akciju koju želite da izvršite. Na primer, ako želite da otvorite web stranicu, možete koristiti implicitni intent koji specificira URL, a sistem će odabrati aplikaciju koja može otvoriti taj URL.

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("http://www.example.com"));
startActivity(intent);
```

2. **Eksplicitni Intenti**: Ovi inteti se koriste kada tačno znate koju aktivnost želite da pokrenete unutar aplikacije. U eksplicitnom intentu navodite punu klasu ciljne aktivnosti.

```
Intent intent = new Intent(this, SecondActivity.class);
startActivity(intent);
```

Ključne metode Intenta

- **setAction()**: Postavlja akciju koju intent treba da izvrši, kao što su Intent.ACTION_VIEW ili Intent.ACTION_SEND.
- **setData()**: Postavlja podatke koji se prenose zajedno sa intentom, kao što su URI adrese.
- **putExtra()**: Dodaje sve dodatne informacije u intent uz pomoć ključ-vrednost para. Ove informacije mogu biti različitih tipova (stringovi, integeri, itd).

Opis aplikacije

U sklopu ove laboratorijske vežbe, kreiraćete Android aplikaciju koja koristi Broadcast Receivere i Servise kako bi korisnike redovno obaveštavala o bitnim vremenskim promenama. Aplikacija će slati obaveštenja kada se dogode određeni vremenski uslovi.

Koraci:

1. **Postavljanje projekta**
    - Otvorite Android Studio i kreirajte novi projekat.
    - Izaberite "Empty Activity" i imenujte svoj projekat ("WeatherAlertApp").
    - Postavite minimalnu verziju Androida (preporučeno API 21).
2. **Dodavanje potrebnih biblioteka i dozvola**

    U build.gradle dodajte zavisnosti za pristup internetu i obaveštenja:

    ```
    implementation 'com.google.android.gms:play-services-location:18.0.0'
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.android.support:design:28.0.0'
    ```

    U AndroidManifest fajlu dodajte i sledeće permisije, da biste omogućili aplikaciji pristup internetu i preuzimanje podataka sa javnog **OpenWeatherMap** API-ja.

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

    Navedena permisija **INTERNET** omogućava našoj aplikaciji da se poveže na internet, dok **ACCESS_NETWORK_STATE** omogućava proveru statusa mrežne konekcije (da li je uređaj povezan na WiFi ili mobilnu mrežu).

    Takođe, najverovatnije će nam biti potrebna i biblioteka za HTTP pozive, kao što je Retrofit. I nju ćemo dodati u build.gradle.

    ```
    implementation 'com.squareup.retrofit2:retrofit:2.9.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
    ```

    Retrofit je popularna Android biblioteka koja olakšava rad sa **REST API** komponentama, omogućavajući jednostavno slanje HTTP zahteva i primanje odgovora. Korisnicima omogućava da konvertuju JSON podatke u Java objekte i pruža elegantan način za organizovanje API poziva kroz definisanje interfejsa i korišćenje anotacija.

3. **Podešavanje OpenWeatherMap API ključa**

   Da biste koristili OpenWeatherMap API, neophodan je API ključ, koji dobijate nakon registracije na njihovom sajtu (https://openweathermap.org/appid).
   Ovaj ključ ćete koristiti za autentifikaciju u svakom API pozivu. Preporučuje se da ključ čuvate u fajlu koji nije vidljiv u aplikaciji, ali za svrhu ove vežbe možete ga direktno staviti u kod. (na primer, `private static final String API_KEY = "YOUR_API_KEY";`). Detaljnije instrukcije za rad sa ovim API-jem su date u okviru koraka 5.
4. **Kreiranje korisničkog interfejsa (UI)**
   Otvorite **activity_main.xml** i dodajte sledeće grafičke elemente:
   - **TextView** za prikaz trenutnih vremenskih informacija (grad, temperatura, prateći vremenski uslovi).
   - **Button** za pokretanje servisa (npr. "Pokreni obaveštenja").
   - **Button** za zaustavljanje servisa (npr. "Zaustavi obaveštenja").
   - **CheckBox** za odabir obaveštenja (npr. "Vreme svitanja", "Vreme zalaska sunca", "Nagla promena temperature", "Kiša", "Jak vetar", "Sneg").

5. **Kreiranje Servisa**

   U ovom koraku kreiraćete **Servis** koji će, pre svega, biti odgovoran za uspešno prikupljanje vremenskih informacija i slanje obaveštenja korisnicima aplikacije.
   Koristićemo **Foreground Service** da bismo osigurali da servis ostane aktivan čak i kada je aplikacija u pozadini.

   a. **Kreirajte novu Java/Kotlin klasu** za servis.

      Dodati klasu **WeatherService** i proširiti je sa Service. Primer koda je dat u nastavku:

```
public class WeatherService extends Service {
    @Override
    public IBinder onBind(Intent intent) {
        // Ne koristimo interfejs za vezivanje, vraćamo null
        return null;
    }
    @Override
    public void onCreate() {
        super.onCreate();
           // Ovde možete pripremiti sve što je potrebno pre nego što servis počne
           sa radom
    }
}
```

   b. **Implementirajte onStartCommand() metodu**.

      Ova metoda će se koristiti za pokretanje servisa i prikupljanje podataka o vremenu. Možete koristiti AlarmManager za zakazivanje periodičnog prikupljanja podataka ili koristiti WorkManager za unapred zakazane zadatke. U nastavku je dat primer koda:

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    // Dobijamo AlarmManager instancu
    AlarmManager alarmManager =
            (AlarmManager) getSystemService(Context.ALARM_SERVICE);
    // Kreiramo Intent koji će pokrenuti servis kada alarm aktivira
    Intent alarmIntent = new Intent(this, WeatherService.class);
    // PendingIntent koji će AlarmManager koristiti da pokrene servis
    PendingIntent pendingIntent =
            PendingIntent.getService(this, 0, alarmIntent,
            PendingIntent.FLAG_UPDATE_CURRENT);
    // Zakazujemo ponavljanje svakih 30 minuta
    long interval = AlarmManager.INTERVAL_HALF_HOUR;
    long triggerAt = SystemClock.elapsedRealtime() + interval;
    alarmManager.setInexactRepeating(AlarmManager.ELAPSED_REALTIME_WAKEUP,
    triggerAt, interval, pendingIntent);
    // Vraćamo START_STICKY da bi se servis ponovo pokrenuo ako ga sistem ugasi
    return START_STICKY;
}
```

c. **Kreiranje koda za prikupljanje vremenskih podataka**.

U ovom delu, potrebno je da pozovete OpenWeatherMap API koristeći vaš API ključ i prikažete dobijene podatke. Kada pošaljete zahtev ka OpenWeatherMap API-ju, na primer za trenutne vremenske podatke, dobićete JSON podatke sa informacijama o vremenskim uslovima, temperaturi, vetru itd.

Primer API URL za preuzimanje trenutnih podataka o vremenu:

```
https://api.openweathermap.org/data/2.5/weather?q={CITY_NAME}&appid={YOUR_API_KEY}&units=metric
```

Sada sledi kreiranje **Retrofit** interfejsa za OpenWeatherMap koji će definisati poziv ka OpenWeatherMap API-ju. On, na primer, može izgledati kao što je dato u nastavku:

```
public interface OpenWeatherMapService {
    @GET("data/2.5/weather")
    Call<WeatherResponse> getCurrentWeather(
        @Query("q") String cityName,
        @Query("appid") String apiKey,
        @Query("units") String units // metric
    );
}
```

@GET: Ovaj anotacija označava HTTP GET metod i ukazuje na deo URL-a koji se koristi za zahteve. Ostatak URL-a se definiše u Retrofit klijentu.

@Query: Ova anotacija se koristi za dinamičke parametre koji se dodaju u URL, kao što su ime grada, API ključ i jedinice za temperaturu.

Retrofit nam omogućava da automatski konvertujemo JSON odgovor sa API-ja u Java objekte. Ovo je ključno jer većina API-ja, uključujući OpenWeatherMap API, vraća podatke u formatu **JSON** (JavaScript Object Notation). JSON je format za razmenu podataka koji je lako razumljiv, ali da bismo ga koristili u Android aplikaciji, moramo da te podatke prevedemo u Java ili Kotlin objekte. Evo jednog primera kako izgledaju JSON podaci o vremenu:

```json
{
  "main": {
    "temp": 22.5
  },
  "weather": [
    {
      "description": "clear sky"
    }
  ]
}
```

Ove podatke nije jednostavno direktno koristiti u aplikaciji, pa je potrebna dodatna implementacija. Klase koje mapiraju JSON podatke (npr WeatherResponse) kreiramo nakon što definišemo Retrofit interfejs, a pre nego što napravimo Retrofit instancu i pošaljemo zahteve. Ove klase čine strukturu podataka koju očekujemo od API-ja, a Retrofit ih koristi da bi mogao automatski da konvertuje JSON odgovor u Java objekte. Sledi primer kako može WeatherResponse da izgleda:

```java
public class WeatherResponse {
    private Main main;
    private List<Weather> weather;

    // Getteri za podatke
    public Main getMain() { return main; }
    public List<Weather> getWeather() { return weather; }

    public class Main {
        private float temp;
        public float getTemp() { return temp; }
    }

    public class Weather {
        private String description;
        public String getDescription() { return description; }
    }
}
```

Nakon opisanih koraka, treba implementirati i Retrofit instancu u okviru servisa. Ovaj korak se obično radi u metodi onCreate() ili u okviru samog servisa gde se vrši prikupljanje podataka. Jedno od mogućih programskih rešenja je dato u nastavku:

```
public class WeatherService extends Service {
    private Retrofit retrofit;
    private OpenWeatherMapService weatherApi;

    @Override
    public void onCreate() {
        super.onCreate();
        // Inicijalizacija Retrofit instance
        retrofit = new Retrofit.Builder()
            .baseUrl("https://api.openweathermap.org/")
            .addConverterFactory(GsonConverterFactory.create())
            .build();
        // Kreiranje interfejsa
        weatherApi = retrofit.create(OpenWeatherMapService.class);
    }

    // Ova metoda pokreće prikupljanje podataka
    private void getWeatherData(String cityName) {
        // Pozivanje OpenWeatherMap API-ja
       Call<WeatherResponse> call = weatherApi.getCurrentWeather(cityName,
                                    "YOUR_API_KEY", "metric");
        // Asinhrono izvršavanje poziva
        call.enqueue(new Callback<WeatherResponse>() {
            @Override
            public void onResponse(Call<WeatherResponse> call,
                                   Response<WeatherResponse> response) {
                if (response.isSuccessful()) {
                    // Prikaz dobijenih podataka
                    WeatherResponse weatherData = response.body();
                    showNotification(weatherData); // Notifikacija rezultata
                }
            }
            @Override
            public void onFailure(Call<WeatherResponse> call, Throwable t) {
                t.printStackTrace(); // Obrada greške
            }
        });
    }
}
```

d. **Prikazivanje Notification Channel-a** (za Android 8.0 i novije).

Nakon što dobijemo podatke o vremenskim prilikama, možemo ih prezentovati preko notifikacije. Za *foreground* servise, obavezno treba kreirati Notification Channel. Ovaj kod može biti dodat u onCreate() metodu servisa. Detaljnije je opisano u koraku 6.

6. **Kreiranje obaveštenja**

Za slanje obaveštenja korisniku koristimo **NotificationManager.** Notifikacije će se slati kada BroadcastReceiver detektuje odgovarajuću akciju, npr naglu promenu temperature, kišu, jak vetar, itd. Dodajte kod u **WeatherService** za slanje obaveštenja:

```
private void sendNotification(String message) {
    NotificationManager notificationManager = (NotificationManager)
    getSystemService(Context.NOTIFICATION_SERVICE);
    NotificationCompat.Builder builder =
              new NotificationCompat.Builder(this, "weatherChannel")
              .setSmallIcon(R.drawable.ic_weather)  // Dodajte vašu custom ikonu
              .setContentTitle("Weather Alert")
              .setContentText(message)
              .setPriority(NotificationCompat.PRIORITY_HIGH);
    notificationManager.notify(1, builder.build());
}
```

Naravno, prilagodite ga svojim naslovima, tekstovima i ikonicama iz resursa projekta. Takođe, potrebno je kreirati **Notification Channel** (za Android 8.0 i novije), da bi slanje obaveštenja bilo moguće. Ovaj kod se dodaje ili u MainActivity fajlu, unutar onCreate() metode ili u Service klasi, unutar onStartCommand() metode. Sledi jedan primer koda:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        NotificationChannel channel = new NotificationChannel(
                "weatherChannel",
                "Weather Alerts",
                NotificationManager.IMPORTANCE_HIGH
        );
        NotificationManager manager = getSystemService(NotificationManager.class);
        if (manager != null) {
            manager.createNotificationChannel(channel);
        }
    }
}
```

7. **Registracija servisa u AndroidManifest-u:**

   Da bi servis mogao da radi u pozadini, potrebno je da ga registrujete u AndroidManifest-u. Dodajte sledeći kod u AndroidManifest.xml:

```
<service android:name=".WeatherService"
    android:enabled="true"
    android:exported="false"/>
```

8. **Pokretanje i zaustavljanje servisa iz korisničkog interfejsa**

   U glavnoj aktivnosti možete kreirati dugmeta za pokretanje i zaustavljanje servisa koji prikuplja vremenske podatke.

   1. Pokretanje servisa: U metodi onClick dugmeta "Pokreni obaveštenja", dodajte kod za pokretanje servisa:

      ```
      Intent intent = new Intent(MainActivity.this, WeatherService.class);
      startService(intent);
      ```

   2. Zaustavljanje servisa: U metodi onClick dugmeta "Zaustavi obaveštenja", dodajte kod za zaustavljanje servisa:

      ```
      Intent intent = new Intent(MainActivity.this, WeatherService.class);
      stopService(intent);
      ```

Dodatne napomene:

- Servis (WeatherService) se koristi za periodično prikupljanje vremenskih podataka sa API-ja. Kada servis dobije podatke, on ih može procesuirati i proslediti dalje kako bi obavestio druge delove aplikacije o promenama, koristeći BroadcastReceiver. Na ovaj način, servis emituje događaj kroz Intent koji sadrži podatke o vremenskim uslovima.
- BroadcastReceiver (WeatherReceiver) sluša emitovane događaje. Kada primi informaciju o promeni vremena od servisa, koristi te podatke kako bi obavio neku akciju, kao što je ažuriranje korisničkog interfejsa ili slanje notifikacije korisniku.

9. **Kreiranje Broadcast Receiver-a**

   - Napravite novu Java/Kotlin klasu koja nasleđuje **BroadcastReceiver.** Možete je nazvati **WeatherBroadcastReceiver.**
   - Ova klasa će primati obaveštenja o vremenskim promenama i odgovarajuće aktivnosti (npr. obaveštavanje korisnika).
   - Registrujte Receiver za akcije kao što su **ACTION_SUNRISE, ACTION_SUNSET, ACTION_TEMP_UP, ACTION_TEMP_DOWN, ACTION_RAIN,** itd.

   Nakon što servis pošalje broadcast poruku, BroadcastReceiver prima poruku i može prikazati obaveštenje korisniku ili promeniti podatke u interfejsu. Sledi primer:

```
public class WeatherReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (action != null) {
            switch (action) {
                case "ACTION_RAIN":
                    showNotification(context, "Kiša počinje. Ponesite kišobran!");
                    break;
                case "ACTION_WIND":
                    showNotification(context, "Jak vetar! Budite oprezni.");
                    break;
                case "ACTION_TEMP_DOWN":
                    showNotification(context, "Zahladilo je. Obucite se toplo.");
                    break; // Dodajte i druge akcije po želji
            }
        }
    }
    private void showNotification(Context context, String message) {
       NotificationManager notificationManager = (NotificationManager)
       context.getSystemService(Context.NOTIFICATION_SERVICE);
        NotificationCompat.Builder builder = new
        NotificationCompat.Builder(context, "weatherChannel")
                .setSmallIcon(R.drawable.ic_weather)
                .setContentTitle("Weather Alert")
                .setContentText(message)
                .setPriority(NotificationCompat.PRIORITY_HIGH);
        notificationManager.notify(1, builder.build());
    }}
```

10. **Registracija Broadcast Receiver-a**

    Registrujte Receiver u AndroidManifest.xml fajlu na sledeći način:

```
<receiver android:name=".WeatherReceiver">
    <intent-filter>
        <action android:name="ACTION_SUNRISE"/>
        <action android:name="ACTION_SUNSET"/>
        <action android:name="ACTION_TEMP_DOWN"/>
        <action android:name="ACTION_RAIN"/>
        <action android:name="ACTION_WIND"/>
       // Dodajte druge akcije po želji
    </intent-filter>
</receiver>
```

Korisni resursi

Layouts:

- https://adapty.io/blog/android-layouts-and-views/
- https://www.lambdatest.com/blog/layouts-in-android/
- https://www.browserstack.com/guide/android-ui-layout

Services:

- https://developer.android.com/develop/background-work/services
- https://www.geeksforgeeks.org/services-in-android-with-example/
- https://medium.com/@dilip2882/android-services-types-examples-and-when-to-use-b3fff058862d

Broadcast Receivers:

- https://developer.android.com/reference/android/content/BroadcastReceiver
- https://www.tutorialspoint.com/android/android_broadcast_receivers.htm
- https://medium.com/@khush.panchal123/understanding-broadcast-receivers-in-android-044fbfaa1330

WeatherApp tutorijali:

- https://youtube.com/playlist?list=PLam6bY5NszYPU3Bk39HuPnnVVTaLhWIL0&si=_fl_OijzY3VH72Z3
- https://www.youtube.com/watch?app=desktop&v=GFhKfMY0L2E
- https://www.youtube.com/watch?v=q7NF-2gtfEU