OSNOVE RAZVOJA WEB I MOBILNIH APLIKACIJA

LV 6: Aktivnosti i fragmenti

Osijek, 2022.

1. PRIPREMA

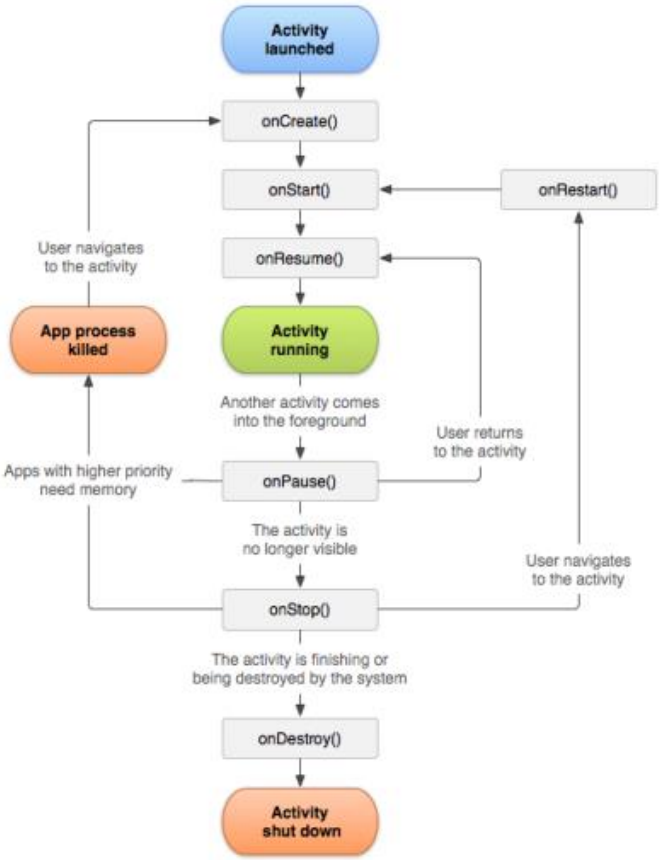
U ovoj je vježbi prikazano korištenje aktivnosti (*activity*) i fragmenata. Vježba opisuje korištenje aktivnosti i prebacivanje iz jedne aktivnosti u drugu putem *Intenta* i korištenja koncepta fragmenata koji omogućuje fleksibilniji *layout* korištenjem modularnih komponenti kao i izmjenu sučelja tijekom izvođenja aplikacije.

1.1. Aktivnosti (*Activity*)

Activity je klasa koja predstavlja jedan zaslon aplikacije. Prvi *Activity* u vježbi je stvoren prilikom stvaranja projekta, a pri tome je automatski postavljen kao početni zaslon koji se prikazuje pri pokretanju aplikacije. Za razliku od klasičnih desktop aplikacija, Android aplikacije nemaju *main* metodu koja služi kao ulazna točka programa, već se životni ciklus aplikacije, odnosno pojedinih aktivnosti oslanja na određene metode s povratnim rezultatom (engl. Callback). Ove se metode pozivaju primjerice kod stvaranja *Activitya* ili kod njegova gašenja, a metode životnog ciklusa su: *onCreate*, *onStart*, *onResume*, *onPause*, *onStop*, *onRestart* i *onDestroy*. Životni je ciklus *Activitya* prikazan slikom 1.1. Budući da je sučelje odvojeno od koda, nužno ga je „napuhati“, odnosno nužno je iz XML opisa kreirati objekte koji predstavljaju elemente korisničkog sučelja. Ovo obavlja metoda *setContentView* kojoj se kao argument daje resurs koji predstavlja sučelje za *Activity*.

1.2. Fragmenti

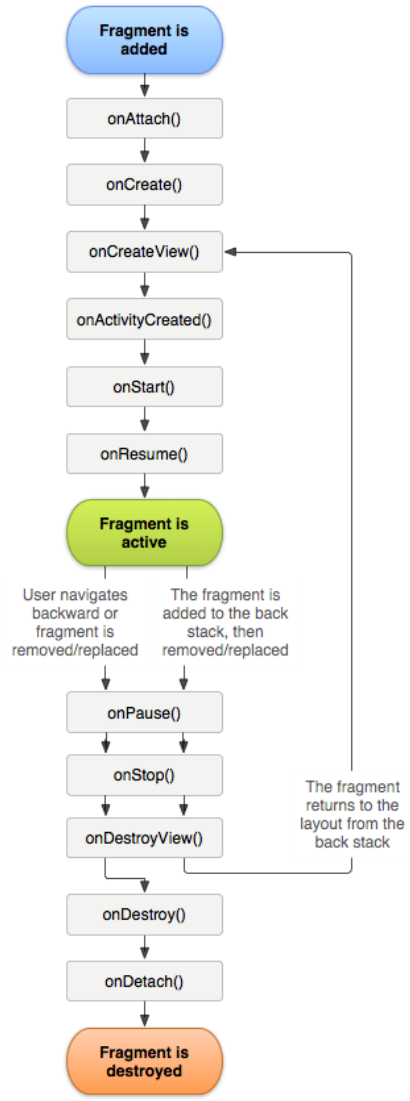
Fragment predstavlja Android komponentu koja sadrži nekakvo ponašanje ili dio korisničkog sučelja koje se nalazi unutar jedne aktivnosti. Oni omogućavaju modularan dizajn aktivnosti jer se više *Fragmenata* može kombinirati unutar jedne aktivnosti. *Fragmenti* imaju vlastiti životni ciklus koji je usko vezan uz aktivnost u kojoj se trenutno nalaze i mogu osluškivati vlastite korisničke događaje. Također ih možete dinamički dodavati ili brisati iz aktivnosti koja se trenutno izvodi na korisničkom zaslonu. Važno je napomenuti da rad s fragmentima nije jednostavan, a prilikom njihova korištenja potrebno je biti oprezan.



Slika 1.1. Prikaz životnog ciklusa Android aplikacije (developer.android.com).

1.2.1. Životni ciklus fragmenta

Kao i aktivnosti, svaki fragment ima vlastiti životni ciklus s događajima koji se okidaju kada se stanje fragmenta promjeni. Kao i kod aktivnosti možete dodati programski kod za svaku metodu s povratnim pozivom (engl. Callback) i tako programirati ponašanje vašeg fragmenta. Životni ciklus fragmenta prikazan je na slici 1.2.



Slika 1.2. Prikaz životnog ciklusa fragmenta.

1.2.2. Pravljenje fragmenta

Prije Android 9.0 imali ste dvije vrste kreiranja fragmenata prva je bila kreirati fragment koji je osiguran putem verzije platforme koju korisnik vaše aplikacije koristi. Dok drugi način (koji se sada i koristi) je putem *V4 Support Librarya* koji se u projekt uvozi kao vanjska biblioteka. Korištenjem druge metode dobijete dosljednost programskog koda i funkcionalnosti kroz više različitih uređaja i Android verzija.

1.3. Intent

Ako se Android aplikacija sastoji od više aktivnosti potrebno je navigirati od jedne do druge aktivnosti. Mehanizam koji omogućuje tu navigaciju naziva se Intent. Korištenjem odgovarajućeg Intenta moguće je pokrenuti bilo koju aktivnost. On je zapravo objekt koji enkapsulira zahtjev, odnosno skup informacija, čiji su osnovni elementi akcija (opća radnja koju je potrebno izvršiti) i podaci nad kojima se akcija izvršava. Razlikujemo implicitne i eksplicitne *Intente*. Eksplicitni definiraju ciljanu komponentu u samom *Intentu*, dok implicitni traže od sustava da procjeni komponente temeljeno na podacima koje *Intent* sadrži. Osim promjene *Activitya*, *Intenti* omogućuju i komunikaciju među drugim komponentama Android aplikacije.

1.3.1. Eksplicitni Intent

Eksplicitni *Intent* definira konkretnu komponentu koju sustav treba pozvati korištenjem klase kao identifikator i uobičajeno se koristi za komponente unutar aplikacije. Za pokretanje određene aktivnosti potrebno je pozvati *startActivity* metodu kojoj se kao argument prosljeđuje *Intent* objekt. Primjer korištenja eksplicitnog *Intenta* je prikazano na slici 1.3.

```
val intent = Intent(this, SecondActivity::class.java)
startActivity(intent)
```

Slika 1.3. Prikaz korištenja eksplicitnog Intenta.

1.3.2. Implicitni Intent

Implicitni Intent ne definira konkretnu komponentu koju sustav treba pozvati, već opisuje komponentu kakva je potrebna za obradu nekog zadatka nad određenom vrstom podataka i prepušta odluku sustavu ako postoji više komponenti koje mogu obaviti određene vrste posla. Primjer je pokretanje PDF datoteke na mobilnom uređaju koji ima instaliranih više PDF preglednika. Odabir komponente koja je zadužena za obradu određene akcije zahtijevane *Intentom* izvršava se korištenjem *Intent filtera*. *Intent filteri* za svaku komponentu definiraju se u manifest datoteci. Ako se sustavu pošalje *Intent*, on provodi određivanje komponente koja odgovara pri čemu koristi podatke iz *Intenta*. Ovim načinom određuje se jedino koja komponenta će obraditi implicitni *Intent*, dok se za eksplicitni ona određuje u objektu *Intenta*. Ako postoji više

komponenti koje mogu odraditi posao, onda sustav nudi izbor između. U tu svrhu dodaje se *Intent* filter u *Manifest* datoteci. Tri ključna podatka su: akcija (npr. ACTION_SEND koja omogućuje dijeljenje sadržaja), podaci koji su podržani (npr. „text/plain“) i kategorija (npr. DEFAULT koja uključuje sve zahtjeve poslane zahtjeve gdje kategorija nije eksplicitno zadana). Primjer korištenja implicitnog *Intenta* prikazano je na slici 1.4.

Slika 1.4. Prikaz korištenja implicitnog *Intenta*.

```
val sendIntent = Intent().apply {
    action = Intent.ACTION_SEND
    putExtra(Intent.EXTRA_TEXT, textMessage)
    type = "text/plain"
}

try {
    startActivity(sendIntent)
}
catch (e: ActivityNotFoundException) {
}
```

1.3.3. Prosljeđivanje podataka između aktivnosti

Osim navigacije između aktivnosti i drugih komponenata sustava, *Intentom* je moguće proslijediti podatke drugim komponentama sustava. S ciljem postizanja prosljeđivanja podataka koristi se *putExtra* metoda. Podaci koji se dodaju definiraju se kao parovi ključ/vrijednost gdje je ključ uvijek objekt *String* klase, a vrijednost može biti jedan o ugrađenih tipova podataka, ali i objekti tipa *String* ili *Bundle*. Ako se želi slati objekte vlastitih klasa, tada je potrebno implementirati jedno od sučelja jednog od sučelja *Serializable* ili *Parcelable*. Ako klasa implementira prvo sučelje, tada nije dužna implementirati nikakve dodatne metode već ju je automatski moguće serijalizirati uporabom refleksije. S obzirom da se rabi refleksija, riječ je o relativno sporoj serijalizaciji što može postati usko grlo unutar aplikacije. Definirano je stoga sučelje *Parcelable* koje u početku traži više truda jer je programer sam odgovoran za serijalizaciju, no pri tome se ostvaruju značajna ubrzanja. Ipak, slanje velikih količina podataka bilo bi dobro izbjeći te pribjeći drugačijem načinu razmjene. Primjerice, umjesto slanja čitavih objekata pohraniti objekt u bazu podataka te poslati

samo identifikator i slično. Primjer korištenja prosljeđivanja podataka putem *Intenta* je prikazan na slici 1.5.

```
val displayIntent = Intent(this, DataDisplayActivity::class.java)
displayIntent.putExtra("KEY_MESSAGE", "Hello World")
displayIntent.putExtra("KEY_SIZE", 128)
startActivity(displayIntent)
```

Slika 1.5. Prikaz korištenja prosljeđivanja podataka putem Intenta.

Kada je riječ o prosljeđivanju podataka između dva *Fragmenta* često se koristi *Bundle*, a podaci se definiraju kao parovi ključ/vrijednost. Prilikom zamjene *Fragmenata* pomoću *replace* metode potrebno je deklarirati klasu novog fragmenta i nakon toga je potrebno postaviti argumente. Kao argument *Fragmenta* se stavlja *Bundle*, a prije no što se *Fragment* prikaže na korisničkom zaslonu programski kod može preuzeti sadržaj argumenata. Prosljeđivanje podataka između dva *Fragmenta* prikazano je u tablici 1.

QuestionFragment.kt

```
val sureFragment = QuestionSureFragment()
val bundle = Bundle()
bundle.putString("BUTTON", radioButton.text.toString())
sureFragment.arguments = bundle

val fragmentTransaction: FragmentTransaction? =
activity?.supportFragmentManager?.beginTransaction()
fragmentTransaction?.replace(R.id.mainFragment, sureFragment)
fragmentTransaction?.commit()
```

QuestionSureFragment.kt

```
sureText.text = arguments?.getString("BUTTON").toString()
```

Tablica 1. Prikaz prosljeđivanja podataka između dva Fragmenta.

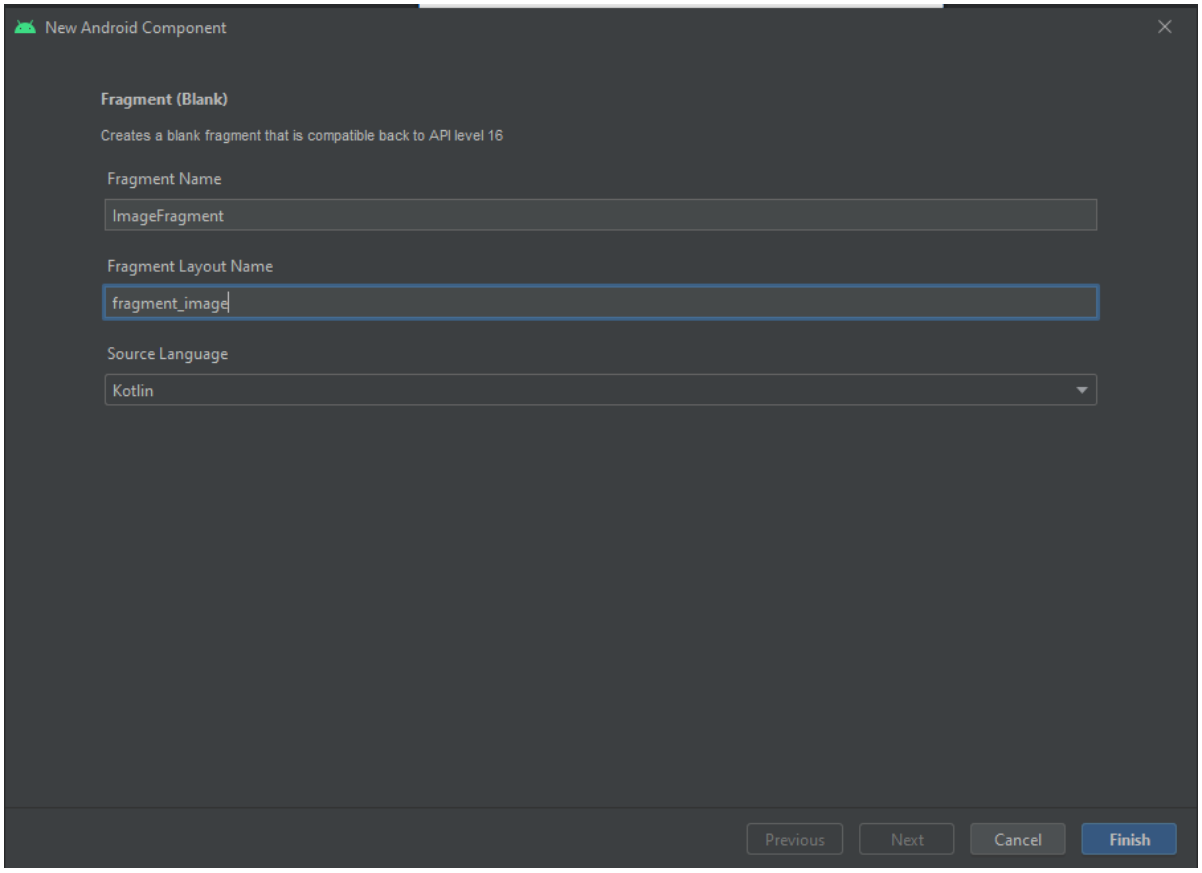
2. RAD NA VJEŽBI

Iz teorije i koraka prikazanih u pripremi ove laboratorijske vježbe potrebno je stvoriti fragmente pomoću čarobnjaka za stvaranje fragmenata.

2.1.1. Stvaranje fragmenta u Android Studiju

U ovome primjeru bi trebalo realizirati dva fragmenta i dva gumba. Prvi gumb treba omogućiti zamjenu pozicija dvaju fragmenata, dok drugi gumb treba omogućiti prelazak iz *MainActivitya* u neki drugi *Activity*.

Stvaranje u Android Studiju je vrlo jednostavno i moguće ga je stvoriti putem čarobnjaka za stvaranje novih fragmenata gdje Android Studio automatski generira i izgled i programski kod. Za stvaranje novog fragmenta potrebno je otići: File → New → Fragment→ Fragment (Blank). U čarobnjaku je potrebno upisati naziv fragmenta i ostalo ostaviti kako je predefinirano. Primjer ispunjenog čarobnjaka za stvaranje fragmenata prikazan je na slici 2.1. Prvi *Fragment* će sadržavati kontrole za prikaz upisanog teksta unutar *TextEdit* polja.



Slika 2.1. Prikaz ispunjenog čarobnjaka za stvaranje fragmenata.

Nakon što je za završeno stvaranje fragmenta potrebno je obrisati većinu generiranog programskog koda i ostaviti samo dio programskog koda koji je prikazan na slici 2.2.

```
class TextEditFragment : Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(R.layout.fragment_text_edit, container,
false)
    }
}
```

Slika 2.2. Prikaz obrisanog programskog unutar fragmenta.

Isti postupak je potrebno ponoviti za drugi fragment koji je nazvan *ImageFragment* jer će sadržavati samo sliku. Nakon stvorenih fragmenata potrebno je napraviti dizajn izgleda (engl. Layouta). Programski kod za izglede fragmenata je priložen u tablici 1.

fragment_text_edit.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".TextEditFragment">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            android:id="@+id/textView"
            android:layout_width="0dp"
            android:layout_height="128dp"
            android:layout_marginStart="64dp"
            android:layout_marginTop="8dp"
            android:layout_marginEnd="64dp"
            android:text="@string/app_name"

app:layout_constraintBottom_toTopOf="@+id/editTextTextPersonName"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="0.498"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <EditText
            android:id="@+id/editTextTextPersonName"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginStart="64dp"
            android:layout_marginTop="16dp"
            android:layout_marginEnd="64dp"
            android:ems="10"
```

```
            android:inputType="textPersonName"
            android:minHeight="48dp"
            android:text="@string/input_text"
            app:layout_constraintBottom_toTopOf="@+id/button"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/textView" />

        <Button
            android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="24dp"
            android:text="@string/save_button"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toBottomOf="@+id/editTextTextPersonName" />
    </androidx.constraintlayout.widget.ConstraintLayout>
</FrameLayout>
```

fragment_image.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ImageFragment">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:srcCompat="@drawable/ic_launcher_background" />

</FrameLayout>
```

Tablica 1. Prikaz programskih kodova za izgled fragmenata.

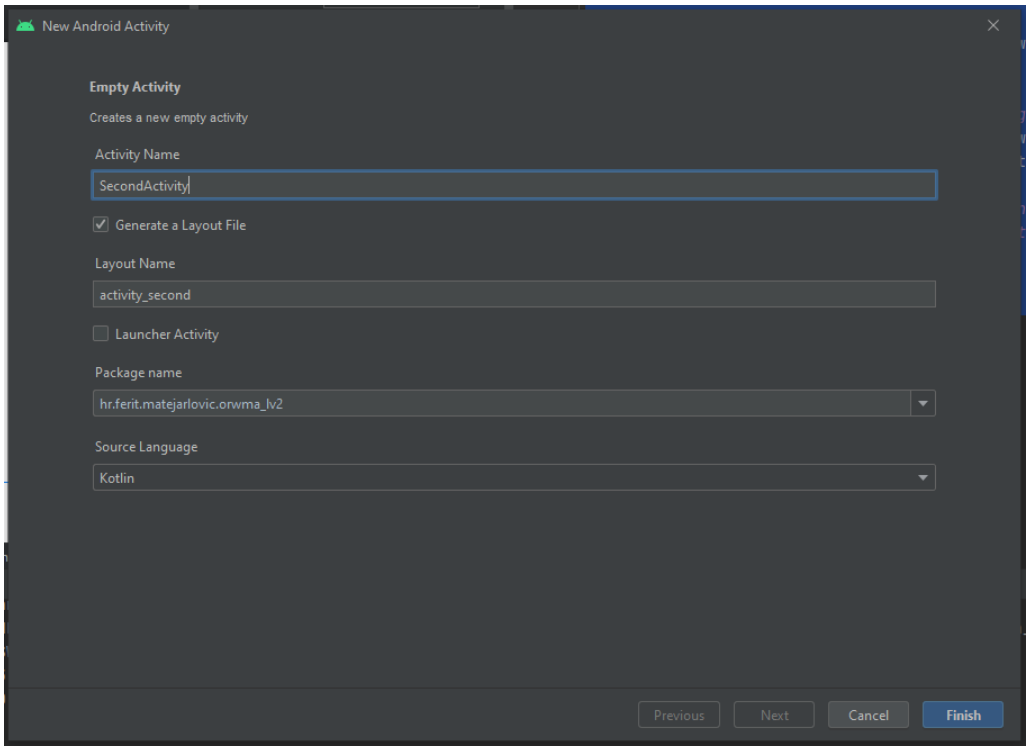
Sljedeće je potrebno isprogramirati funkcionalnosti drugog fragmenta pod nazivom *TextEditFragment* gdje bi se sadržaj *TextLabela* trebao ažurirati sadržajem iz *EditTexta* kada korisnik klikne gumb. Programski kod rješenja tog fragmenta se nalazi u slici 2.3.

```
class TextEditFragment : Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val view = inflater.inflate(R.layout.fragment_text_edit, container,
false)
        val textView = view.findViewById<TextView>(R.id.textView)
        val editText = view.findViewById<EditText>(R.id.editText)

        view.findViewById<Button>(R.id.saveButton).setOnClickListener {
            textView.text = editText.editableText
        }
        return view
    }
}
```

Slika 2.3. Prikaz rješenja primjera gdje će se sadržaj *TextViewa* ažurirati sadržajem *EditTexta* pritiskom na gumb.

Sljedeći korak je omogućiti promjenu aktivnosti iz MainActivity u SecondActivity. Postupak stvaranja aktivnosti je sličan kao i za stvaranje fragmenata. Za pokretanje čarobnjaka za stvaranje aktivnosti potrebno je otići na: File → New → Activity → Empty Activity. Čarobnjaka za stvaranje aktivnosti je prikazan na slici 2.4.



Slika 2.4. Prikaz čarobnjaka za stvaranje nove aktivnosti.

Izgled druge aktivnosti će sadržavati samo *Button* koji će na klik vraćati natrag na *MainActivity*. Programski kod za rješenje drugog *Activitya* je prikazan u tablici 2. U pripremi je objašnjen *Intent*, a putem njega će se pokretati nove aktivnosti. Potrebno je *Intentu* predati trenutnu aktivnost i JAVA klasu druge aktivnosti koju želite pokrenuti. A putem *startActivity* metode se pokreće aktivnost zadana unutar *Intenta*.

activity_second.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SecondActivity">

    <Button
        android:id="@+id/backButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/back_button"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

SecondActivity.kt

```
class SecondActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_second)

        findViewById<Button>(R.id.backButton).setOnClickListener {
            val intent = Intent(this, MainActivity::class.java)
            startActivity(intent)
        }
    }
}
```

Tablica 2. Prikaz rješenja za izgled i programski dio druge aktivnosti.

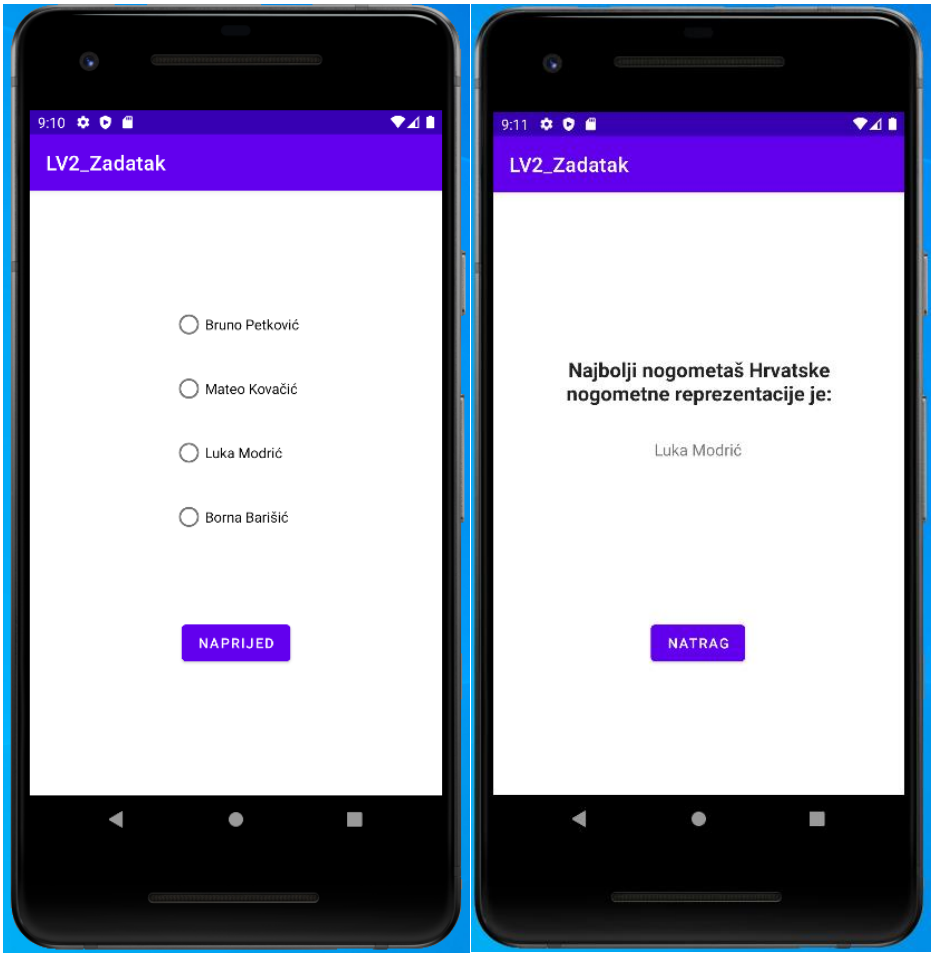
Unutar *MainActivity* datoteke je potrebno napraviti male izmjene i dodati *Intent* i *startActivity* kada korisnik klikne gumb za promjenu aktivnosti. Izmjene programskog koda su prikazane na slici 2.5.

```
findViewById<Button>(R.id.switchActivity).setOnClickListener {
    val intent = Intent(this, SecondActivity::class.java)
    startActivity(intent)
}
```

Slika 2.5. Prikaz izmjene programskog koda unutar MainActivity.kt.

3. ZADACI ZA SAMOSTALAN RAD

1. Potrebno je napraviti Android aplikaciju koja će sadržavati jednu aktivnost i dva fragmenta. Prvi fragment treba sadržavati: 1x *RadioGroup* (unutar njega 4x *RadioButton* elementa) i 1x *Button*. Sljedeći fragment treba sadržavati: 2x *TextViewa* i 1x *Button*. Kada korisnik na prvom Fragmentu odabere jedan *RadioButton* i klikne *Button* za prelazak u drugi *Fragment*, tada se sadržaj kliknutog *RadioButtona* treba prenijeti na drugi *Fragment* putem *Bundlea*. Na drugom Fragmentu je potrebno u *TextView* prikazati sadržaj prenesenog podatka i omogućiti povratak na prošli *Fragment* putem *Buttona*. Izgledi oba *Fragmenta* prikazani su na slici 3.1.



Slika 3.1. Prikaz izgleda Fragmenata za rješenje zadatka 1.