

Kreiranje prve aplikacije

Cilj vježbe je da nauči studente osnovno o Android-u i Android aplikacijama. U sklopu vježbe studenti će kreirati prvu aplikaciju i upoznati se s okruženjem **Android Studio**, s podešavanjem emulatora, te sa debug-om aplikacije.

Android - uvod

Android predstavlja operativni sistem baziran na modificiranoj verziji Linux kernela i drugih open source softvera, dizajniran za pametne telefone i tablete. Android se pojavljuje 2008. godine i do danas se razvilo 10 verzija Android operativnog sistema, koje su poznate po kôdnim imenima, kao što su: Aestro, Blender, Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean, KitKat, Lollipop, Marshmallow, Nougat, Oreo, Pie, Q i R.

Android aplikacije su se ranije primarno pisale u Javi, a danas se preferira Kotlin programski jezik. Pored njih, koriste se i sljedeći jezici: C#, Python, C/C++, Lua, HTML5 + CSS + JavaScript, itd. Za razvoj Android aplikacija koristi se skup alata nazvan Android SDK (software development kit). Ovaj skup alata možete preuzeti sa web stranice: [Download Android Studio and SDK tools](#)

Android aplikacije se izvršavaju na Android operativnom sistemu koji je instaliran na uređaju (ili virtuelnoj mašini). U sklopu Android OS svaka aplikacija je drugačiji korisnik sa jedinstvenim ID brojem. Za svaku aplikaciju se postavljaju permisije, tako da sve fajlove te aplikacije može koristiti korisnik (tj. aplikacija) sa navedenim ID brojem. Za potrebe pristupanja korisničkim datotekama ili sistemskim funkcionalnostima, potrebno je zahtijevati odgovarajuće permisije od korisnika. Više o permisijama možete pročitati na sljedećem linku: [Permissions overview](#)

Android sadrži automatski build sistem **Gradle**. Sa njim je moguće buildati, pokretati i testirati aplikacije. Rezultat buildanja često je APK datoteka, koja se može prebaciti na uređaj ili emulator, te pokretati. APK fajl je softverski paket, arhiva koja sadrži Android program i sve metapodatke potrebne za njegovo pokretanje. Android studio automatski generiše sve potrebne fajlove i konfiguracije za buildanje aplikacije koju razvijate. Fajlovi koji su vezani za Gradle, a bitni su za ispravan rad i razvoj aplikacije su:

- *build.gradle* - na nivou projekta - postavke vezane za sve module na nivou projekta;
- *build.gradle* - na nivou modula, app modul - postavke vezane za dati modul, sadrži podatke o *compileSdkVersion*, *minSdkVersion* i *targetSdkVersion*. Postavka *compileSdkVersion* govori o tome koja verzija Android SDK će se koristiti prilikom buildanja aplikacije, *minSdkVersion* sadrži podatke o najstarijoj verziji koja je podržana u aplikaciji, *targetSdkVersion* podaci o verziji na kojoj ste testirali aplikaciju i za koju je aplikacija namjenjena.
- *settings.gradle*- definišemo listu svih modula i/ili biblioteka koji će biti uključeni u build procesu.

U sklopu vježbi za razvoj Android aplikacija koristit će se Android Studio IDE, posebna verzija IntelliJ IDEA. Android Studio sadrži vlastiti emulator sa raznim verzijama mobilnih telefona i Android verzija. Pored ovog emulatora, savjetuje se

korištenje vlastitih telefona za testiranje, ili [Genymotion Android Emulator](#), radi brzine i opterećenosti.

Komponente Android aplikacije

Osnovne komponente/gradivni blokovi Android aplikacije su:

- [Activity](#) – Aktivnost predstavlja jedan ekran sa svojim korisničkim interfejsom. Na primjeru aplikacije za pregled slika, jedna aktivnost može biti ekran na kojem je prikazana lista svih slika, drugi npr. lista opcija za selektovanu sliku.
- [View](#) – View predstavlja UI elemente Android aplikacije, kao što su Button, Label, Text, itd. Sve što korisnik vidi i s čim vrši interakciju predstavlja View. Više View objekata se grupiše u ViewGroup.
- [Intent](#) – Intent predstavlja abstraktnu definiciju željene akcije. Koriste se i za pozivanje komponenti, npr. omogućava pokretanje servisa, aktivnosti, prikazivanja web stranice, liste kontakata, slanje poruka itd.
- [Fragment](#) – Predstavlja dio aktivnosti. Jedna aktivnost može prikazati više od jednog fragmenta na ekranu istovremeno.
- [Service](#) – Servis je komponenta koja se izvršava u pozadini i obavlja operacije koje se dugo izvršavaju ili radi za neke vanjske procese. Servis ne obezbjeđuje korisnički interfejs. Primjer jednog servisa je preuzimanje podataka sa web servera za aplikaciju vremenske prognoze.
- [Content provider](#) – Content provider upravlja sa dijeljenim skupom podataka aplikacije. Takvi podaci mogu biti sačuvani na file sistemu, SQLite bazi, web-u ili nekoj drugoj lokaciji.
- [Broadcast receiver](#) – Broadcast receiver je komponenta koja odgovara na broadcast objave. Mnogi broadcasti potiču od samog sistema – npr. obavještenje da je baterija skoro prazna. Broadcasti nemaju svoj korisnički interfejs, ali mogu kreirati notifikaciju u statusnoj traci.
- [AndroidManifest.xml](#) – Sadrži sve informacije o aktivnostima, content providerima, permisijama, itd. Sve aplikacije sadrže ovu datoteku.

Android UI widgets

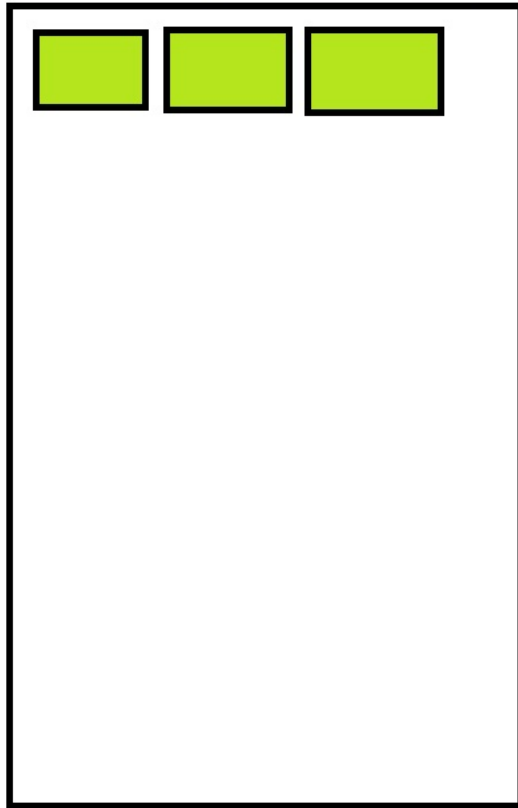
Svi elementi user interfejsa Android aplikacije su smješteni u objekte klase View i ViewGroup. **View** – direktno sadrži element UI koji će biti iscrtan na ekranu i sa kojim će korisnik imati interakciju. **View Group** – služi za grupisanje više View objekata. Strukturu prikaza UI definišemo sa layout-om. **Layout** možemo napraviti na dva načina pišući XML kôd i instancirati layout putem kôda koristeći View i ViewGroup.

Neki od osnovnih layout-a su: **Linear layout**, **Relative Layout**, **List view**, **Grid view**, **ConstraintLayout** i **RecyclerView**.

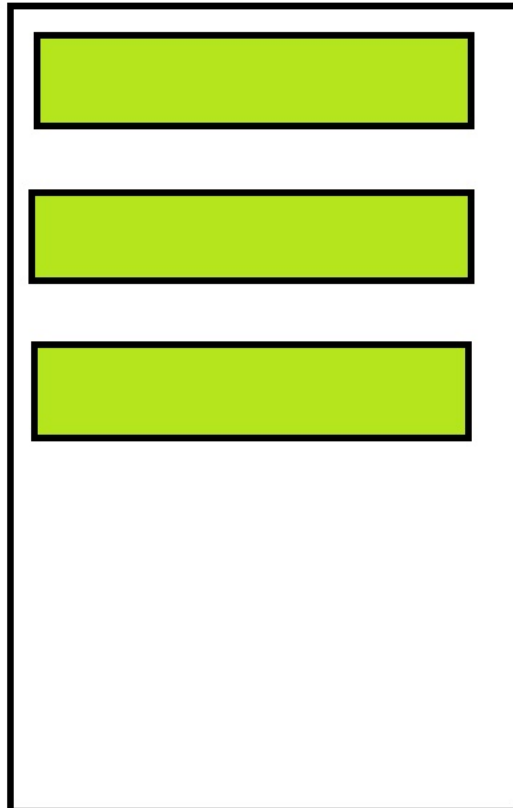
LinearLayout

LinearLayout grupiše view objekte tako da ih poravnava u jednom pravcu horizontalno ili vertikalno. Pravac poravnavanja se određuje atributom `android:orientation`.

Linear Layout Horizontal



Linear Layout Vertical



Linear layout

RelativeLayout

RelativeLayout grupiše view objekte na način da ih prikazuje koristeći relativne pozicije. Pozicija svakog view objekta se može zadati u odnosu na susjedne view objekte tako da se postavi npr. lijevo od nekog view-a, ispod i sl. Kako svaki view ima svoj jedinstveni broj omogućeno je da se pomoću njega definišu međusobne relacije između view-a. Neki od načina definisanja pozicija elemenata u odnosu na druge su pomoću atributa:

- **android:layout_below** = **"@+id/elementB"** - postavlja gornju ivicu trenutnog elementa direktno ispod elementa sa id-em "elementB".
- **android:layout_toRightOf** = **"@+id/elementB"** - postavlja lijevu ivicu trenutnog elementa pored elementa sa id-em "elementB" sa njegove desne strane.



Relative layout

ListView

ListView grupiše elemente i prikazuje ih kao listu koju je moguće skrolati. Elementi liste se automatski unose u listu koristeći Adapter koji dobavlja sadržaj iz izvora podataka poput niza ili upita prema bazi i konvertuje svaki element rezultata u jedan view koji se dodaje u listu.

GridView

GridView grupiše elemente i prikazuje ih na dvodimenzionalnoj mreži koju je moguće skrolati. Elementi GridView-a se automatski dodaju putem ListAdapttera.

ConstraintLayout

ConstraintLayout omogućava kreiranje velikih i kompleksnih layout-a bez hijerarhije. Sličan je RelativeLayout-u gdje su definisane veze između svih View-ova i parent layout-a, ali je daleko fleksibilniji i lakši za rad u Android Studio editor-u.

RecyclerView

RecyclerView predstavlja ViewGroup koja sadrži sve View-ove koji odgovaraju podacima.

Ulazne kontrole

Ulazne kontrole su komponente koje omogućavaju interakciju korisnika sa korisničkim interfejsom. Android posjeduje mnoge ulazne kontrole, a neke on najkorištenijih su:

- *Button* – dugme koje je moguće pritisnuti kako bi se obavila dodijeljena akcija, klasa koja omogućava ovu kontrolu je *Button*. Atributom `android:onclick` zadajemo koja će se funkcija prilikom pritiska dugmeta pozvati. U sklopu prvih vježbi ovo ćemo raditi kroz kôd.
- *Text field* – tekstualno polje koje dozvoljava unos teksta, klasa koja omogućava ovu kontrolu je *EditText*.
- *Checkbox* – prekidač tipa da/ne čije vrijednost korisnik može mjenjati. Ova kontrola se koristi kada je potrebno omogućiti izbor više opcija koje nisu međusobno isključive. Klasa u kojoj je ova kontrola implementirana je *Checkbox*.

- *Radio button* – kontrola slična Checkbox-u stim da samo jedna od opcija iz grupe može biti odabrana. Klase koje se koriste za implementaciju ove kontrole su `RadioGroup` i `RadioButton`.
- *Spinner* – drop-down lista koja omogućava korisniku izbor jedne vrijednosti iz skupa ponuđenih. Klasa kojom se implementira ova kontrola se naziva `Spinner`.
- *Pickers* – kontrole tipa dijaloškog okvira koje omogućavaju odabir vrijednosti iz skupa koristeći dugmad gore/dolje ili swipe pokret. Klase koje omogućavaju odabir datuma i vremena na ovaj način su `DatePicker` i `TimePicker`.

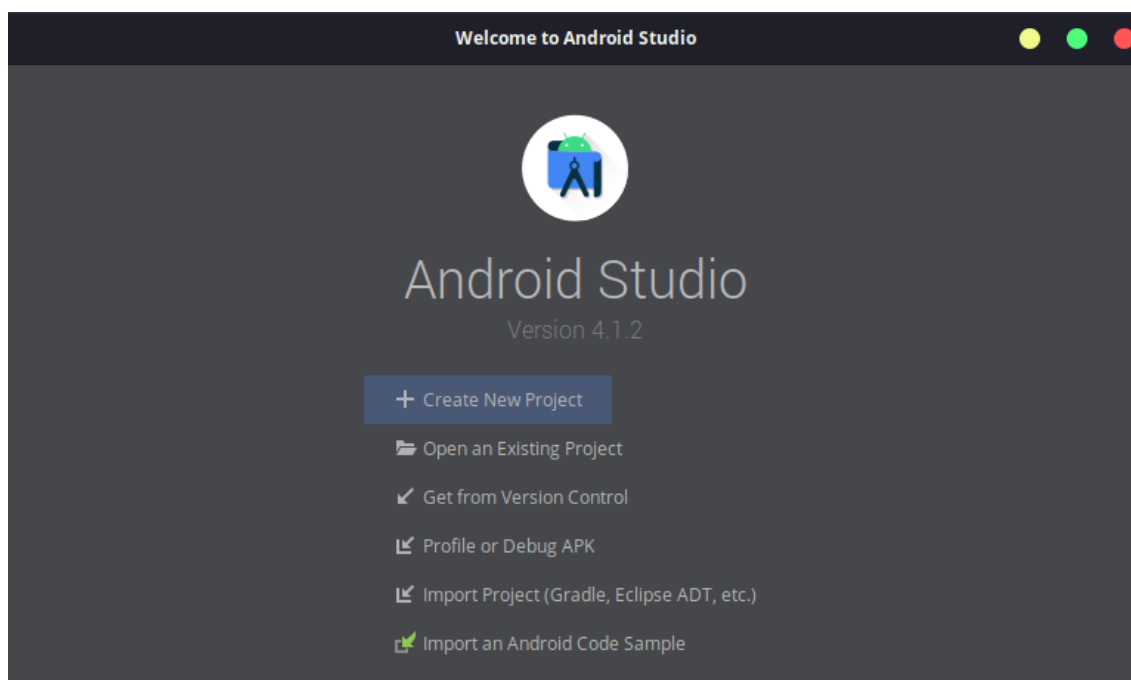
Ulazni događaj/event-i

Interakcija korisnika sa android aplikacijom se može detektovati na više načina. Princip je kada se dogodi odgovarajući event da se pozove funkcija koja je dodijeljena u event listener-u. Event listener je interfejs u View klasi koja sadrži jednu callback metodu. Ova metoda se poziva kada Android detektuje odgovarajuću interakciju korisnika sa aplikacijom. Više o event listenerima na sljedećem linku: [Input events overview](#)

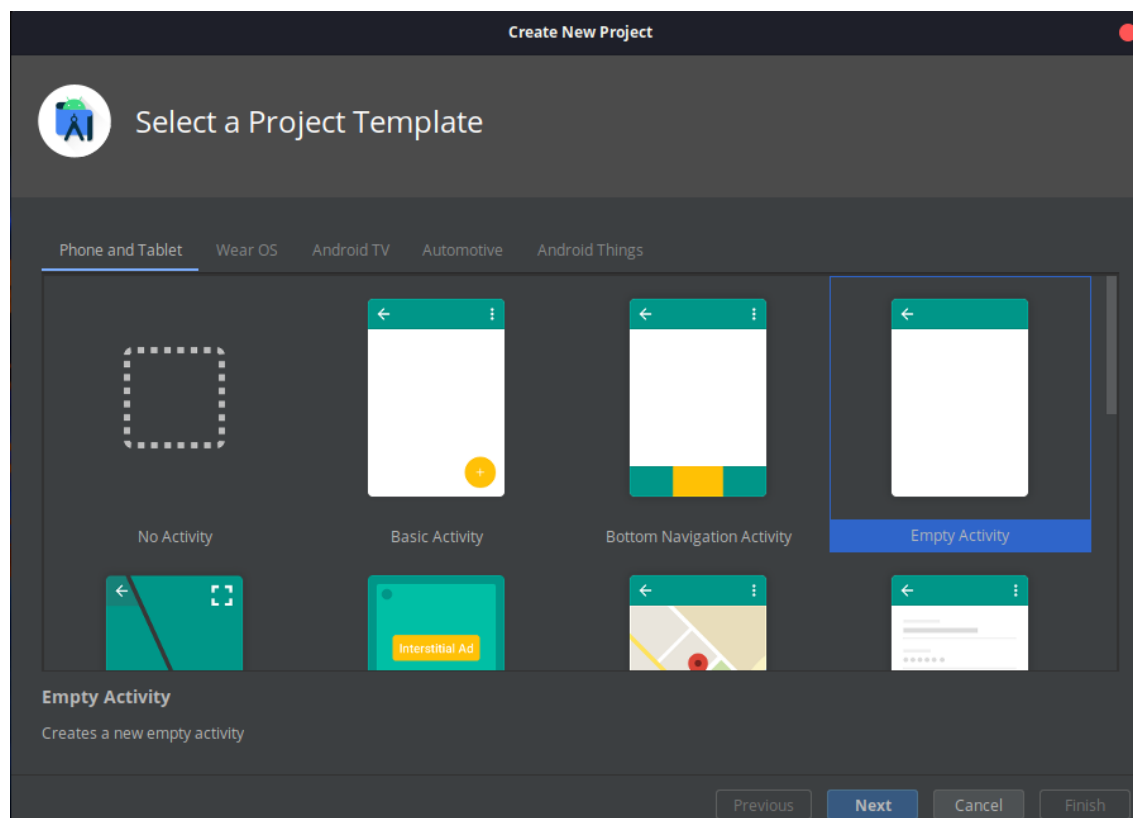
Kreiranje prve aplikacije

Zadatak 1. Kreiranje aplikacije

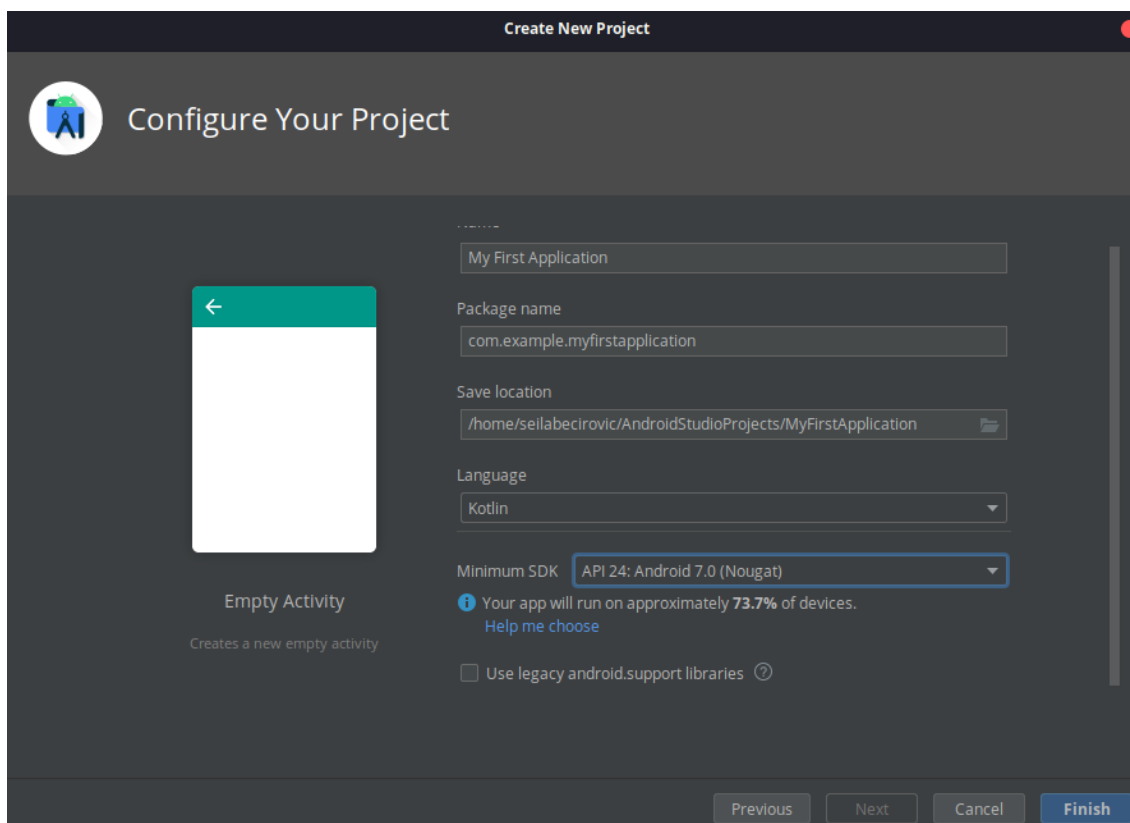
Kreirajte novi android projekat koristeći Android Studio:



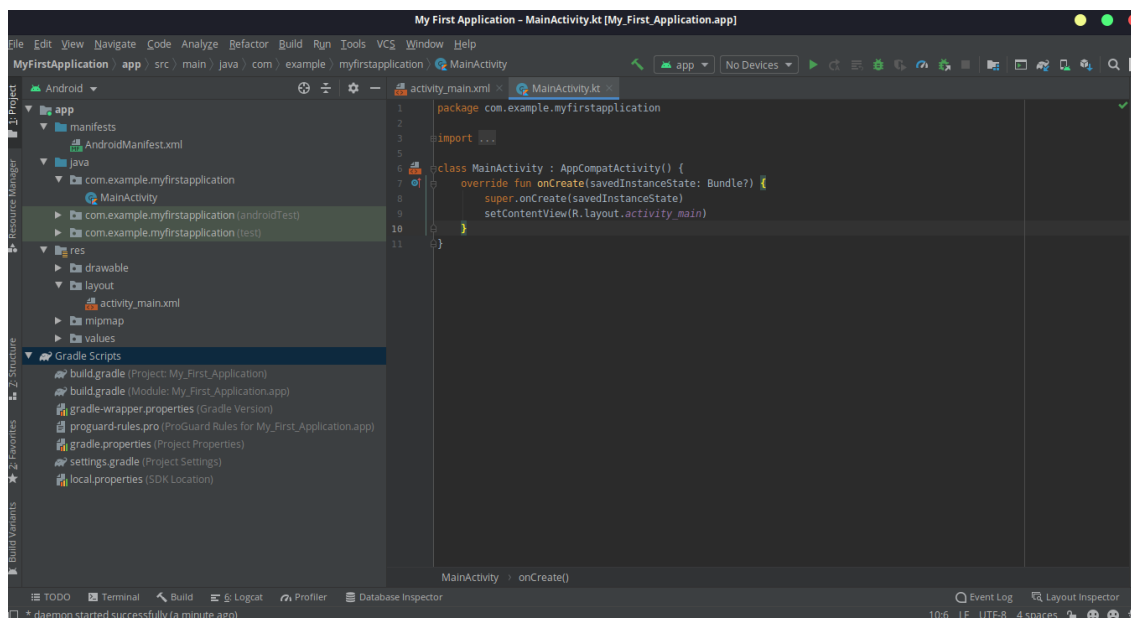
Izaberite **Empty Activity**



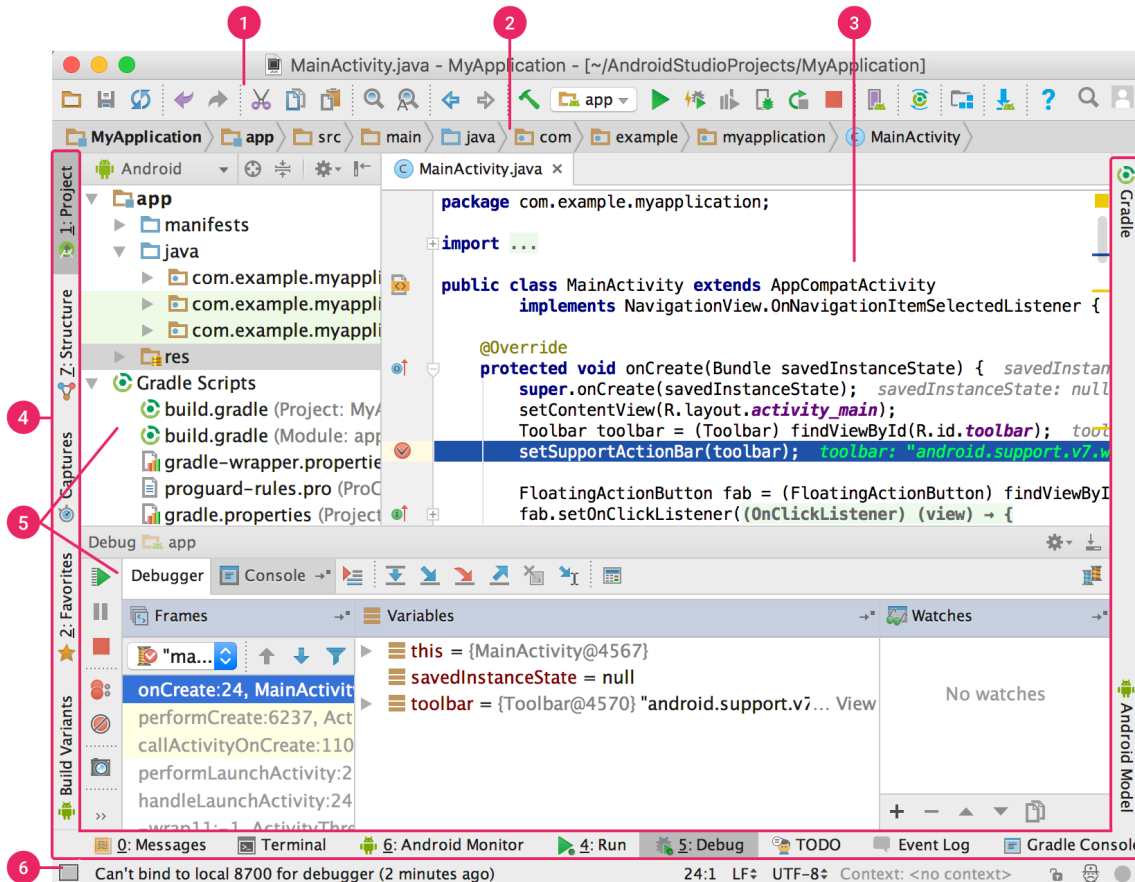
Konfigurirajte vaš projekat. Minimalni SDK postavite na API 24, ili niže na osnovu vlastitog uređaja.



Sačekajte da se kreira vaš projekat. Nakon kreiranja projekta isti će imati osnovne stavke kreirane: odgovarajuću strukturu sa manifest, glavnom aktivnošću i layout-om kreiranim.



Prođimo kroz osnovni prozor Android Studio.



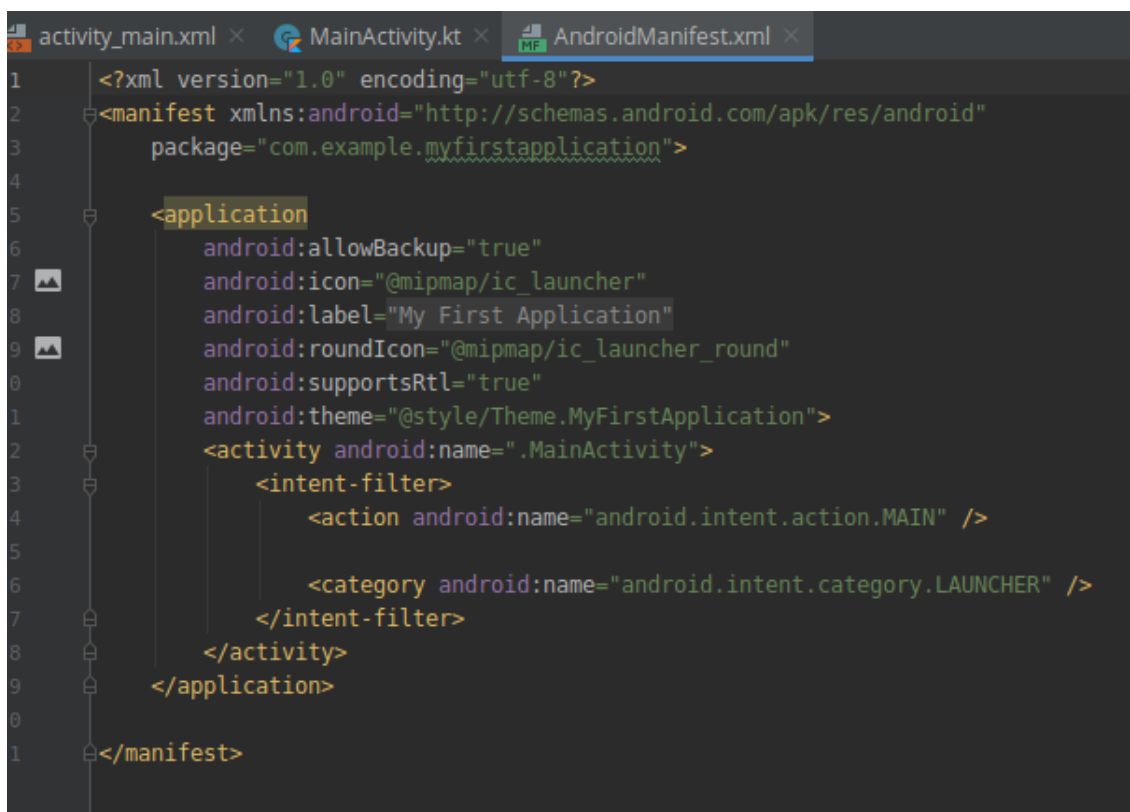
U sklopu njega imamo sljedeće:

1. Toolbar koji omogućava izvršavanje raznih tipova akcija, kao što su pokretanje aplikacije i Android alata
2. Navigacijski bar omogućava prolaz kroz projekat i otvorene datoteke za uređivanje. Kompaktni pregled je dostupan u *Project* prozoru.
3. Editor služi za kreiranje i uređivanje kôda. Postoje dva tipa editor-a za kôd i za layout.
4. Toolbar omogućava prikazivanje određenih alata.
5. Alati projekta podrazumijevaju: menadžment projekta, pretraga, verzija, terminal, debug itd.
6. Statusna traka prikazuje trenutni status projekta i IDE.

Struktura android projekta je sljedeća:

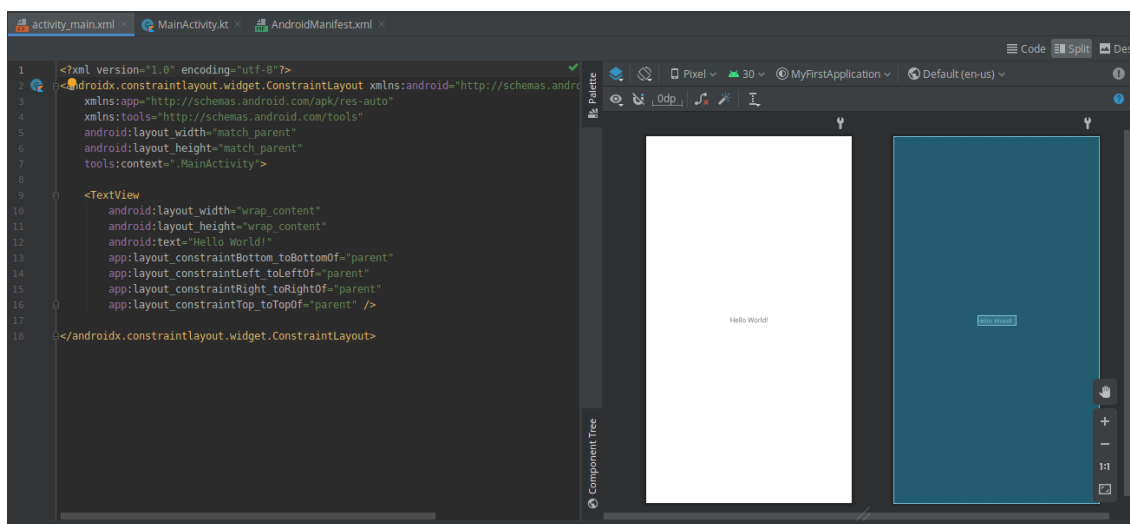
- **manifests** - Sadrži `AndroidManifest.xml` dokument.
- **java** - Sadrži izvorni kôd koji se razdvaja korištenjem paketa. Ovdje se nalaze i testovi
- **res** - Sadrži sve resurse, kao što su XML layout-i, UI stringovi i slike koji su podijeljeni u odgovarajuće direktorije.

Struktura `AndroidManifest.xml` dokumenta je data u nastavku. U sklopu nje se definiše aplikacija i osnovne komponente android aplikacije.



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="com.example.myfirstapplication">
4
5     <application
6         android:allowBackup="true"
7         android:icon="@mipmap/ic_launcher"
8         android:label="My First Application"
9         android:roundIcon="@mipmap/ic_launcher_round"
10        android:supportsRtl="true"
11        android:theme="@style/Theme.MyFirstApplication">
12        <activity android:name=".MainActivity">
13            <intent-filter>
14                <action android:name="android.intent.action.MAIN" />
15
16                <category android:name="android.intent.category.LAUNCHER" />
17            </intent-filter>
18        </activity>
19    </application>
20
21 </manifest>
```

U nastavku je prikazan odgovarajući *LayoutEditor* koji ima dva vida prikaza: Uređivanje XML dokumenta i direktno uređivanje korisničkog interfejsa.

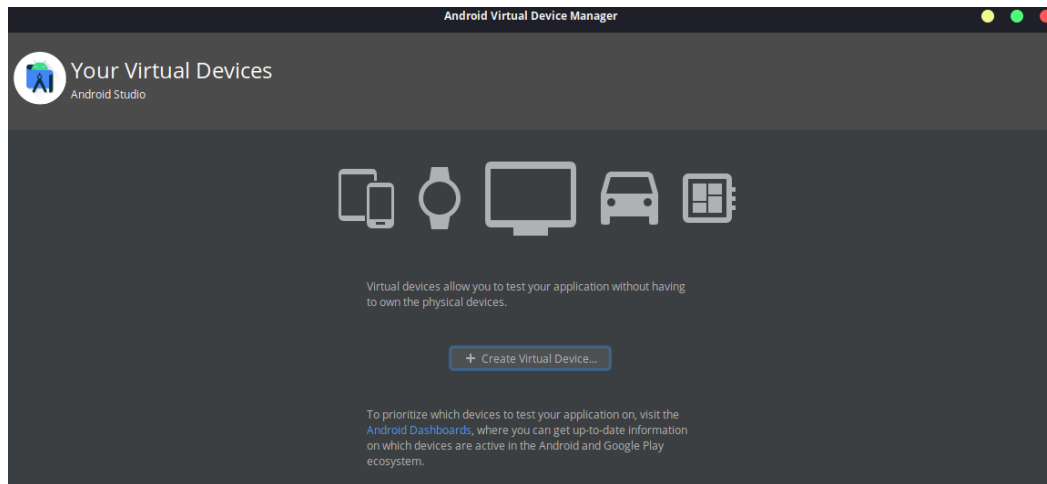


Zadatak 2. Pokretanje aplikacije

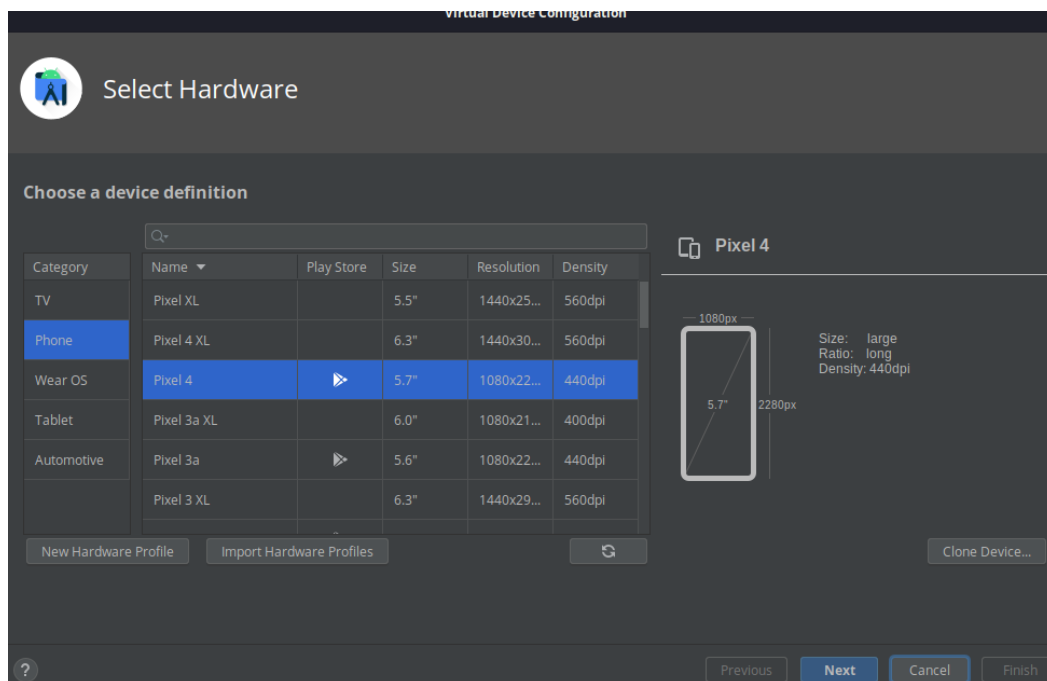
Pokrenimo sada našu aplikaciju. Aplikaciju možete pokretati na vlastitom uređaju pri čemu morate uključiti **USB DEBUGGING**. Da biste ovo uključili morate omogućiti razvojne opcije (**Developer options**) klikom 7 puta na **VERSION NUMBER** u sklopu informacija o telefonu. Pored ovog načina, aplikaciju možete pokretati na emulatoru. U slučaju pokretanja na emulatoru, savjetuje se da se isti ne gasi, te da se samo aplikacija

iznova pokreće na njemu. U nastavku slijede koraci instalacije jednog virtuelnog uređaja. **Napomena:** U BIOS-u opcije virtualizacije moraju biti uključene.

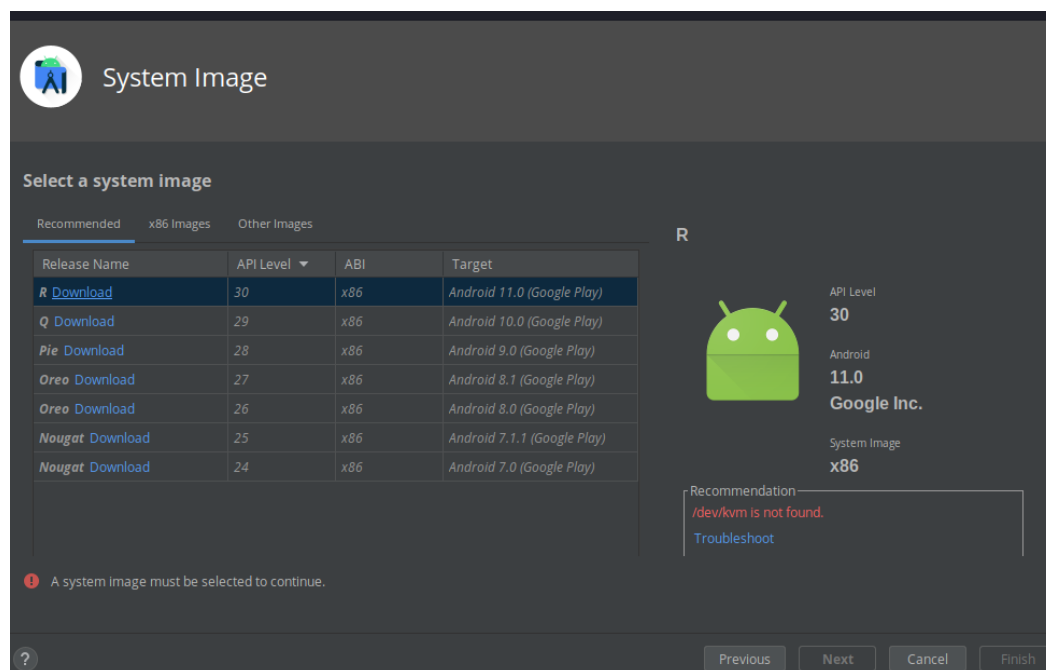
1. U sklopu Devices opcije pokrenite AVD manager (Android Virtual Device Manager).



2. Odaberite odgovarajući model uređaja.

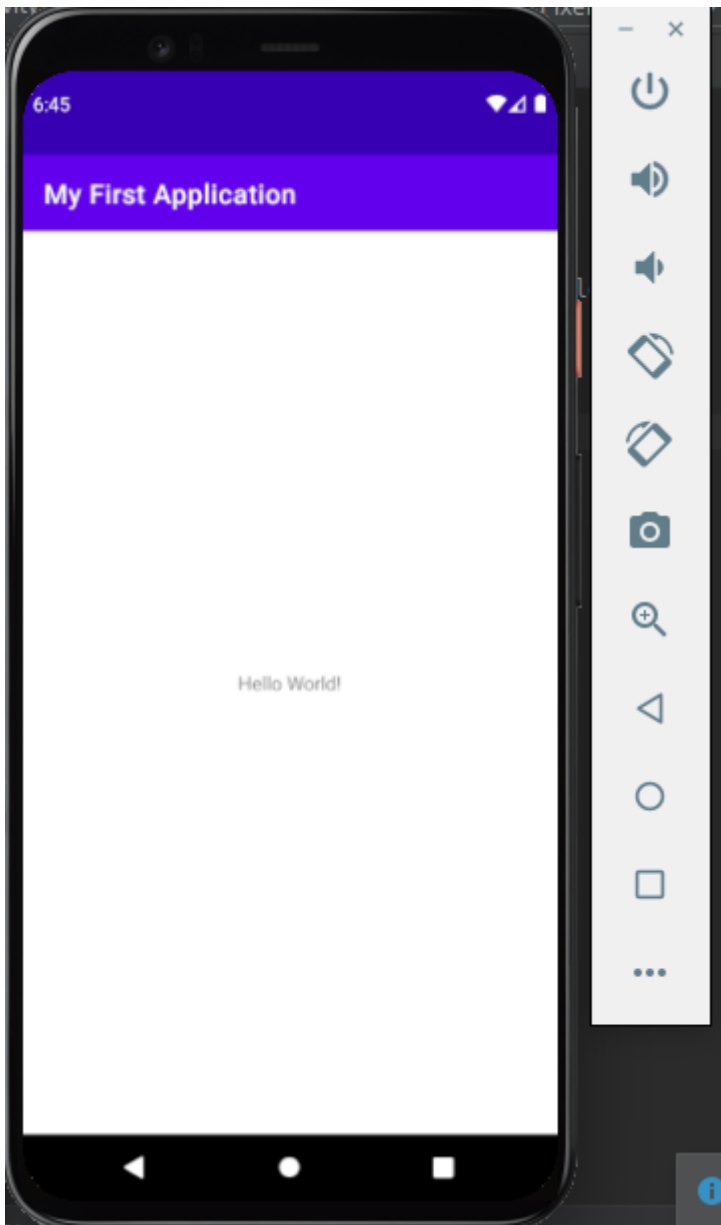


3. Odaberite operativni sistem (preporučuje se Android 10.0)



4. Sačekajte da se instalacija završi.

Nakon što se instalacija završi, pokretanje aplikacije treba biti omogućeno. Pokrenite je na odgovarajućem uređaju. Emulator će se pokrenuti i nakon određenog vremena će se aplikacija build-ati i instalirati na uređaj.



Uspješno je pokrenuta prva aplikacija.

Zadatak 3. Korištenje ulaznih kontrola

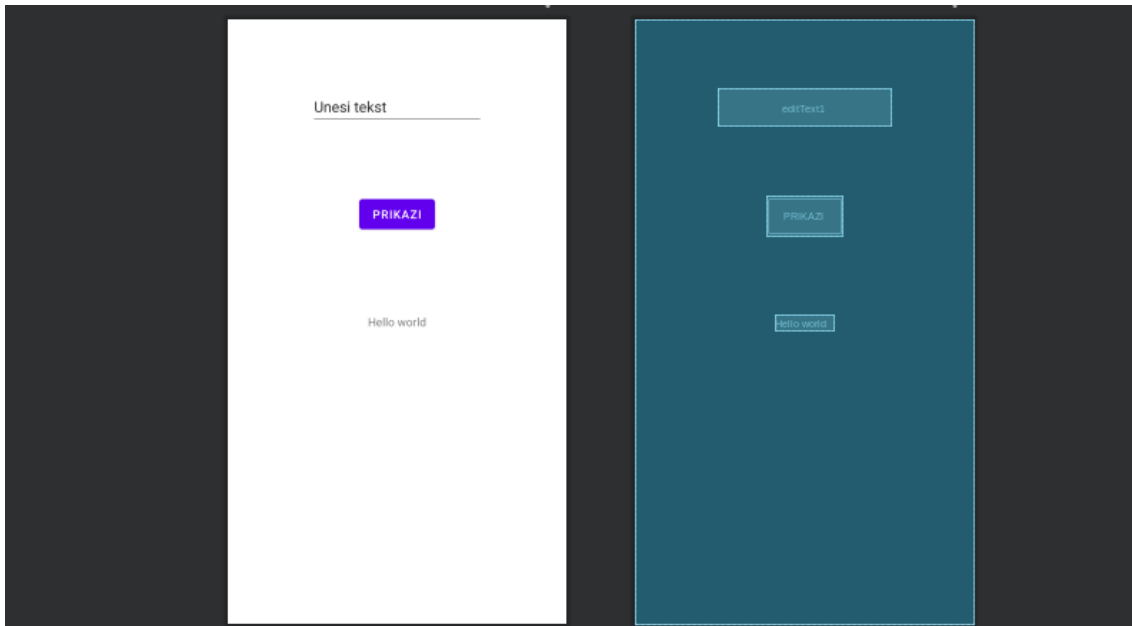
Potrebno je omogućiti unos teksta koji će s klikom na dugme biti prikaz u odgovarajućem TextView-u.

Korištenjem editora kreirat ćemo dodatne dvije kontrole:

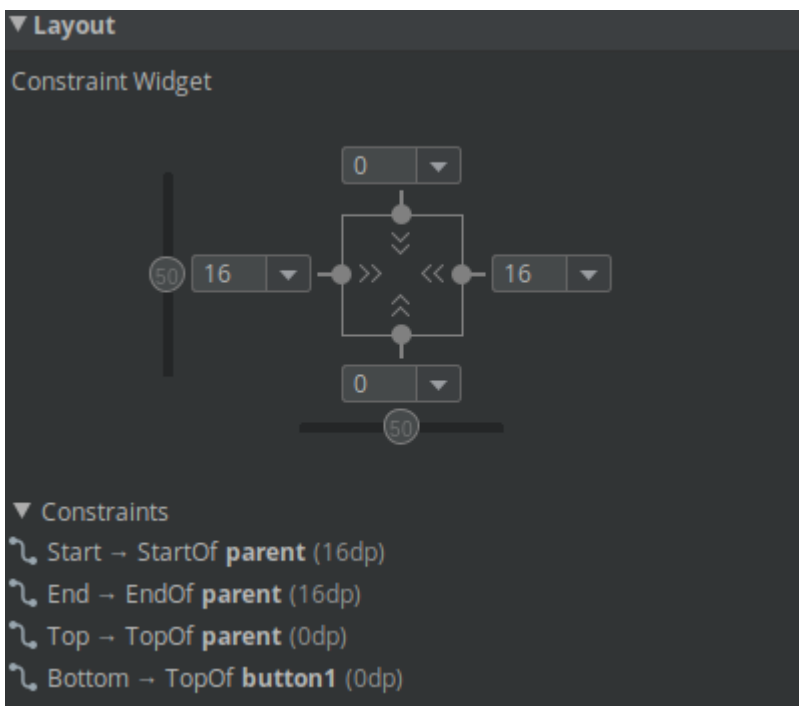
- EditText s ID: `editText1`
- Button s ID: `button1`

Postojeći TextView će imati ID: `textView1`.

Prikaz odgovarajućeg korisničkog interfejsa je dat na slici.



Napomena: Obzirom da je default layout `ConstraintLayout` za sve stavke definišite odgovarajuća ograničenja. Prikaz definisanja layout-a je dat na sljedećoj slici.



Nakon što smo kreirali komponente potrebno je da omogućimo izvršenje traženog kroz kôda. Trenutno klasa glavne aktivnosti glasi:

```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
    }  
}
```

```
}  
}
```

Ovaj kôd označava da prilikom kreiranja aktivnosti će se prikazati odgovarajući layout `activity_main`. Layout-u se pristupa kroz klasu `R` koja označava sve interne resurse aplikacije. Resursi android-a su dostupni kroz `android.R`.

U sklopu odgovarajuće funkcije ćemo dohvatiti referencu na dugme i dodijeliti mu osluškivač na klik.

```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
        //Dohvatit ćemo referencu na view button-a preko id-a  
        val button = findViewById<Button>(R.id.button1);  
        //Definisat ćemo akciju u slučaju klik akcije  
        button.setOnClickListener {  
            //akcije nakon klik-a  
        }  
    }  
}
```

Kreirajmo privatnu funkciju unutar klase koja će dohvatiti tekst iz `EditText`-a i postaviti na `TextView`.

```
private fun showMessage() {  
    // Pronaći ćemo naš edit text i text view na osnovu id-a  
    val editText = findViewById<EditText>(R.id.editText1)  
    val textView = findViewById<TextView>(R.id.textView1)  
    // Tekst ćemo prebaciti u varijablu  
    val message = editText.text.toString()  
    // Postavimo tekst  
    textView.text = message  
}
```

Naš cjelokupni kôd sada izgleda ovako:

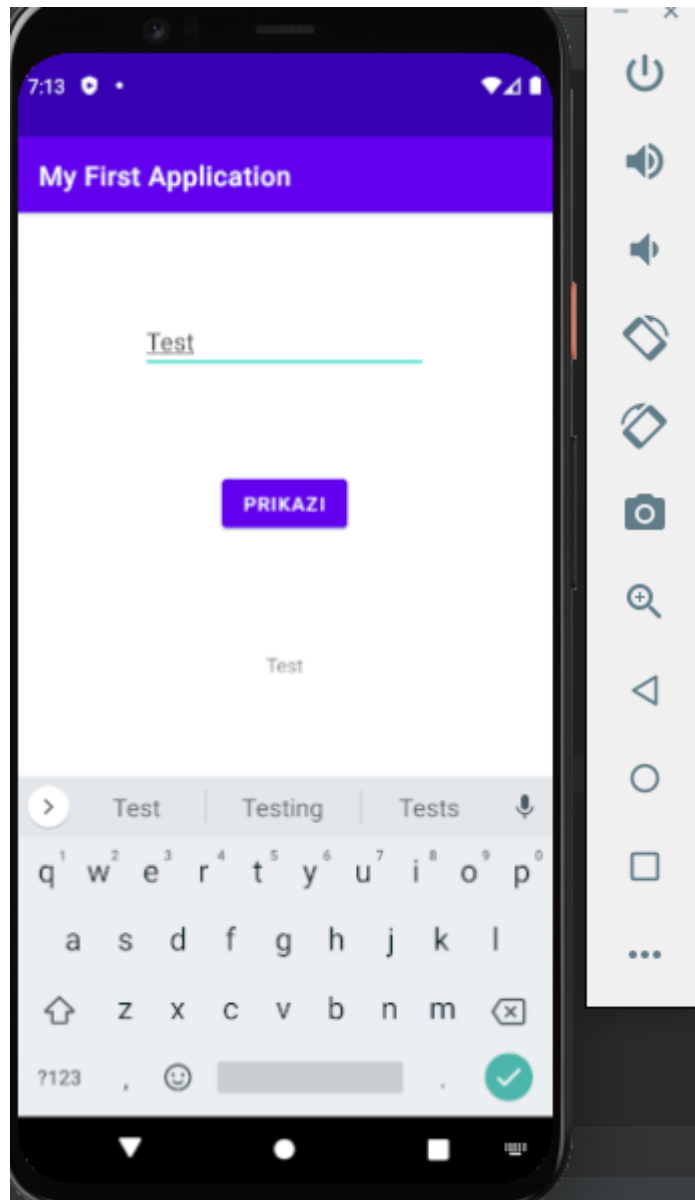
```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
        //Dohvatit ćemo referencu na view button-a preko id-a  
        val button = findViewById<Button>(R.id.button1);  
        //Definisat ćemo akciju u slučaju klik akcije  
        button.setOnClickListener {  
            showMessage()  
        }  
    }  
    //Poziva se na klik dugmenta  
    private fun showMessage() {  
        // Pronaći ćemo naš edit text i text view na osnovu id-a  
        val editText = findViewById<EditText>(R.id.editText1)  
        val textView = findViewById<TextView>(R.id.textView1)
```

```

    // Tekst ćemo prebaciti u varijablu
    val message = editText.text.toString()
    // Postavimo tekst
    textView.text = message
}
}

```

Nakon pokretanja i testiranja vidjet ćemo da aplikacija uradi traženo, kao što je

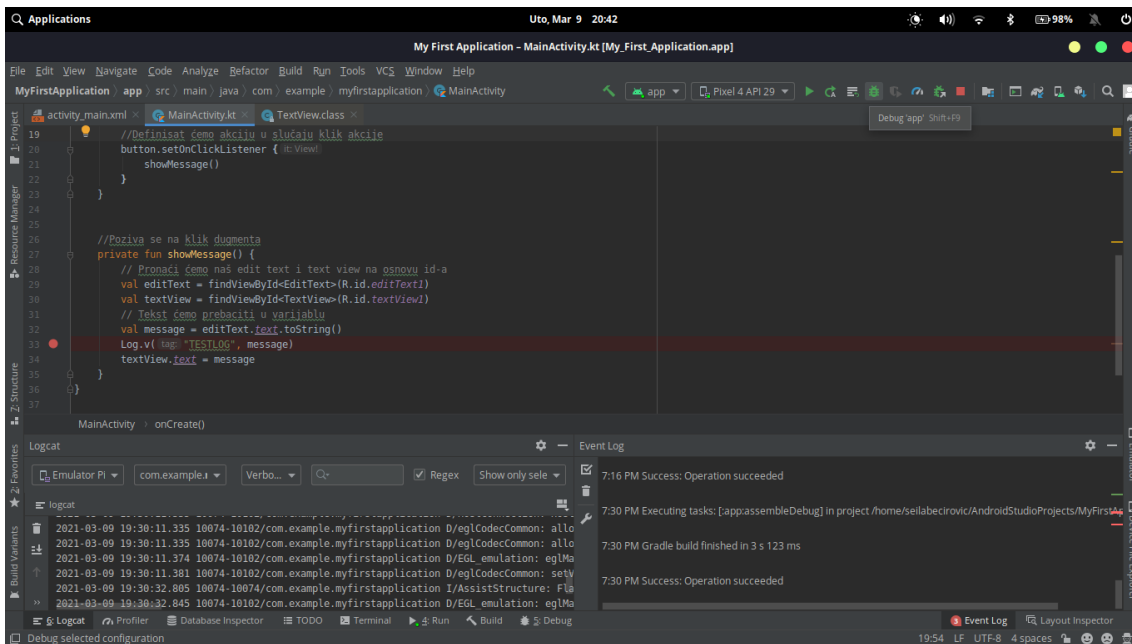


prikazano na slici.

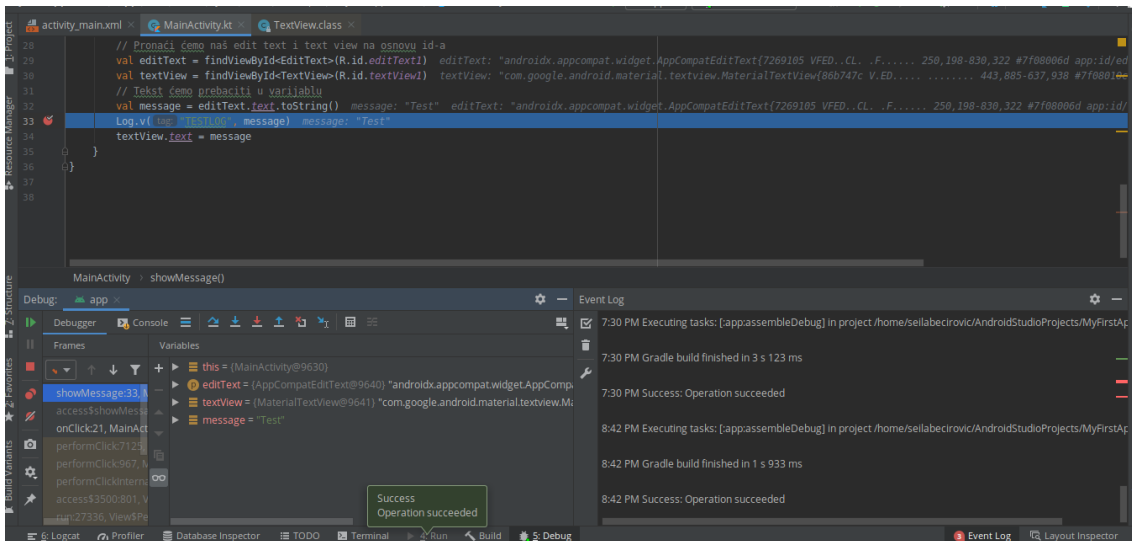
Zadatak 4: Debug aplikacije i logiranje

Postaviti breakpoint na aplikaciji i izvršiti logiranje teksta iz EditText-a.

U nastavku je prikazano kako se dodaje breakpoint, kako glasi odgovarajuća linija logiranja podataka i gdje se pokreće debug mode.



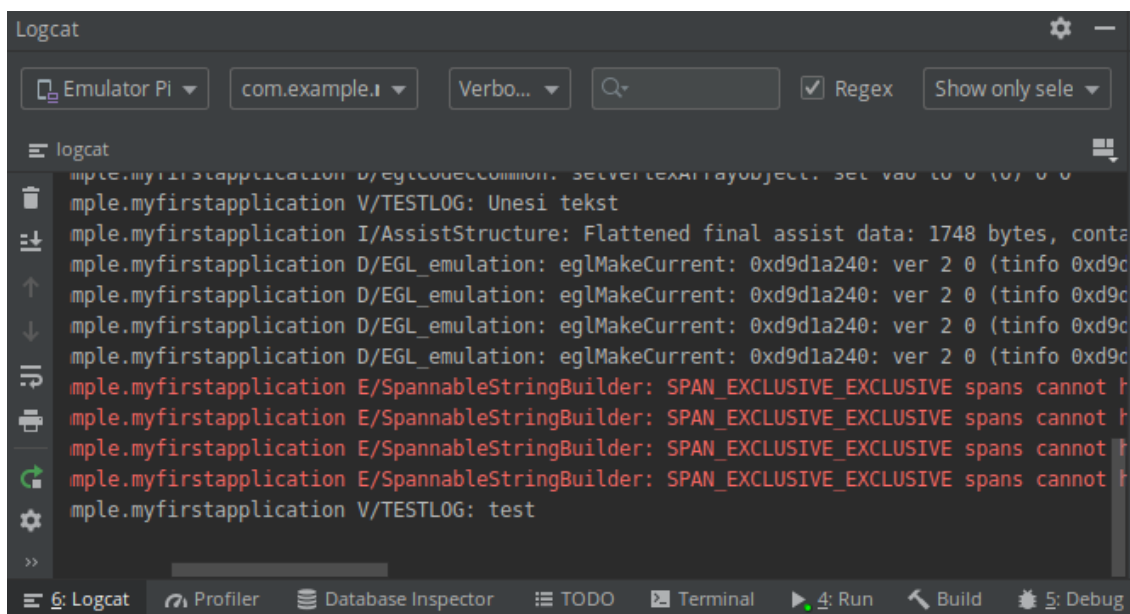
Nakon što se izvrše odgovarajuće akcije na aplikaciji, breakpoint će zaustaviti izvršavanje. Otvorit će se **DEBUG** tab u sklopu kojeg se vidi trenutno stanje aplikacije. Izvršavanje se može nastaviti liniju po liniju ili do samog kraja.



Logiranje u sklopu ovog alata se vrši korištenjem **Logcat-a**. Logovi se mogu vidjeti u sklopu odgovarajućeg **Logcat** tab-a. Postoje sljedeće vrste logova:

- Log.e(String, String) (error)
- Log.w(String, String) (warning)
- Log.i(String, String) (information)
- Log.d(String, String) (debug)
- Log.v(String, String) (verbose)

Prvi parametar predstavlja TAG, a drugi poruku. Prikaz odgovarajućeg loga aplikacije je dat na sljedećoj slici.



Zadatak 5

Projekat postaviti na vlastiti Bitbucket git repozitorij. Link na repozitorij trebate podijeliti s asistentima korištenjem njihovih fakultetskih e-mail adresa.

Za postavljanje projekta na git repozitorij možete koristiti VCS (Version Control) u sklopu Android Studia, ili možete koristiti komandnu liniju. **Napomena:** Obzirom na način rada na predmetu RPR smatra se da dovoljno poznajete rad git-a. U nastavku će biti prikazane osnovne naredbe.

Git Cheat Sheet



GIT BASICS

| | |
|---|---|
| <code>git init</code> <directory> | Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository. |
| <code>git clone</code> <repo> | Clone repo located at <repo> onto local machine. Original repo can be located on the local filesystem or on a remote machine via HTTP or SSH. |
| <code>git config</code> user.name <name> | Define author name to be used for all commits in current repo. Devs commonly use <code>--global</code> flag to set config options for current user. |
| <code>git add</code> <directory> | Stage all changes in <directory> for the next commit. Replace <directory> with a <file> to change a specific file. |
| <code>git commit -m</code> "message" | Commit the staged snapshot, but instead of launching a text editor, use <message> as the commit message. |
| <code>git status</code> | List which files are staged, unstaged, and untracked. |
| <code>git log</code> | Display the entire commit history using the default format. For customization see additional options. |
| <code>git diff</code> | Show unstaged changes between your index and working directory. |

UNDOING CHANGES

| | |
|-------------------------------------|--|
| <code>git revert</code> <commit> | Create new commit that undoes all of the changes made in <commit>, then apply it to the current branch. |
| <code>git reset</code> <file> | Remove <file> from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes. |
| <code>git clean -n</code> | Shows which files would be removed from working directory. Use the <code>-f</code> flag in place of the <code>-n</code> flag to execute the clean. |

REWRITING GIT HISTORY

| | |
|---|---|
| <code>git commit</code> <code>--amend</code> | Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit's message. |
| <code>git rebase</code> <base> | Rebase the current branch onto <base>. <base> can be a commit ID, branch name, a tag, or a relative reference to HEAD. |
| <code>git reflog</code> | Show a log of changes to the local repository's HEAD. Add <code>--relative-date</code> flag to show date info or <code>--all</code> to show all refs. |

GIT BRANCHES

| | |
|--|---|
| <code>git branch</code> | List all of the branches in your repo. Add a <branch> argument to create a new branch with the name <branch>. |
| <code>git checkout -b</code> <branch> | Create and check out a new branch named <branch>. Drop the <code>-b</code> flag to checkout an existing branch. |
| <code>git merge</code> <branch> | Merge <branch> into the current branch. |

REMOTE REPOSITORIES

| | |
|---|---|
| <code>git remote add</code> <name> <url> | Create a new connection to a remote repo. After adding a remote, you can use <name> as a shortcut for <url> in other commands. |
| <code>git fetch</code> <remote> <branch> | Fetches a specific <branch>, from the repo. Leave off <branch> to fetch all remote refs. |
| <code>git pull</code> <remote> | Fetch the specified remote's copy of current branch and immediately merge it into the local copy. |
| <code>git push</code> <remote> <branch> | Push the branch to <remote>, along with necessary commits and objects. Creates named branch in the remote repo if it doesn't exist. |



Visit atlassian.com/git for more information, training, and tutorials

Zadaci za samostalni rad

1. Pratite navedene zadatke.
2. Uredite aplikaciju da koristi `LinearLayout`, pa nakon toga i `RelativeLayout`.