

V – Razvoj aplikacije u Android studiu

SADRŽAJ

- 5.1** Elementi Android aplikacija
- 5.2** Primena servisa u Android OS
- 5.3** Primena prijemnika poruka
- 5.4** Primena provajdera sadržaja
- 5.5** Dodatni elementi Android aplikacija
- 5.6** Razvoj aplikacije u Android studiju
- 5.7** Struktura aplikacije u Android OS

5.1 - Elementi Android aplikacija

- U razvoju aplikacija se koriste komponente koje su povezane u **AndroidManifest.xml** datoteci.
- Ova datoteka **opisuje svaku komponentu** aplikacije i njihovo međusobno delovanje.
- Glavne komponente koje se koriste u Android aplikaciji su:
 - 1. Aktivnosti** – (*activities*) određuju **korisnički interfejs** i upravljaju korisničkim aktivnostima na ekranu telefona.
 - ✓ Aktivnost predstavlja (pojedinačan) **ekran** sa korisničkim interfejsom
 - ✓ Ova komponenta aplikacije **izvršava aktivnosti na ekranu**.

Primer: *email aplikacija može imati jednu aktivnost koja se odnosi na prikaz liste novih poruka, dok se druga aktivnost odnosi na čitanje poruka.*

- ✓ Ako aplikacija ima više od jedne aktivnosti, onda bi jedna od njih trebala biti **označena kao primarna aktivnost** koja će biti prikazana kada se pokrene aplikacija – **fokusirana aplikacija**.
- ✓ Aktivnost se implementira kao podklasa Activity klase:

```
public class MainActivity extends Activity {}
```

5.2 - Primena servisa u Android aplikacijama

2) Servisi (services)

- ✓ Servisi upravljaju **pozadinskim procesima** koji su vezani sa aplikacijom koja obavlja neke **rutinske operacije** u određenim intervalima

Primer: *servis komponenta može obavljati dugotrajnu operaciju prenosa podataka preko mreže, a da se pri tome ne zaustavlja korisničko izvršenje akcije.*

- ✓ Servis se implementira **kao podklasa Service** klase:

```
public class MyService extends Service {
```

- ✓ Izvršava **dugotrajne i zahtevne operacije** u **pozadini** za koje ne postoji potreba da ih aktivira korisnik tako da **nema neki korisnički interfejs**
- ✓ Druga komponenta aplikacije može startovati servis i on će nastaviti da se izvršava u pozadini **iako korisnik pređe na neku drugu aplikaciju.**
- ✓ Komponenta se može **povezati sa servisom** i **interagovati** sa njim i tako omogućiti njihovu komunikaciju (*interprocess communication, IPC*)

Primer: *servis može upravljati transakcijama preko mreže, puštati muziku, realizovati I/O operacije ili komunicirati sa content provajderom i to sve u pozadini.*

5.2 – Primena servisa u Android aplikacijama

➤ Servis se pojavljuje u dve forme:

1. Pokrenuti servis (*Started*) – kada ga komponenta aplikacije pozivom metode *startService()* pokrene. Jednom pokrenut servis može da se izvršava u pozadini neograničeno dugo, bez obzira na trenutni status komponente. Pokrenuti servisi najčešće ne vraćaju neki rezultat komponenti koja ga poziva, već samo izvrše zahtevani zadatak i zaustave se. Na primer, može *download-ovati* ili *upload-ovati* fajl preko mreže.

2. Povezani (*Bound*) – kada se komponenta aplikacije vezuje sa njim pomoću metode *bindService()*. Tada se servis izvršava dokle god je aktivna komponenta ili komponente koje su sa njim povezane. Vezani servisi kroz IPC omogućavaju *klijent-server interfejs*, *slanje zahteva*, *vraćanje rezultata*, i sl. Ovakvi servisi rade sve dok su vezani za neku komponentu i moguće je vezati ih *za više komponenti* istovremeno. Kada se u potpunosti odvežu od komponente servisi se ukidaju.

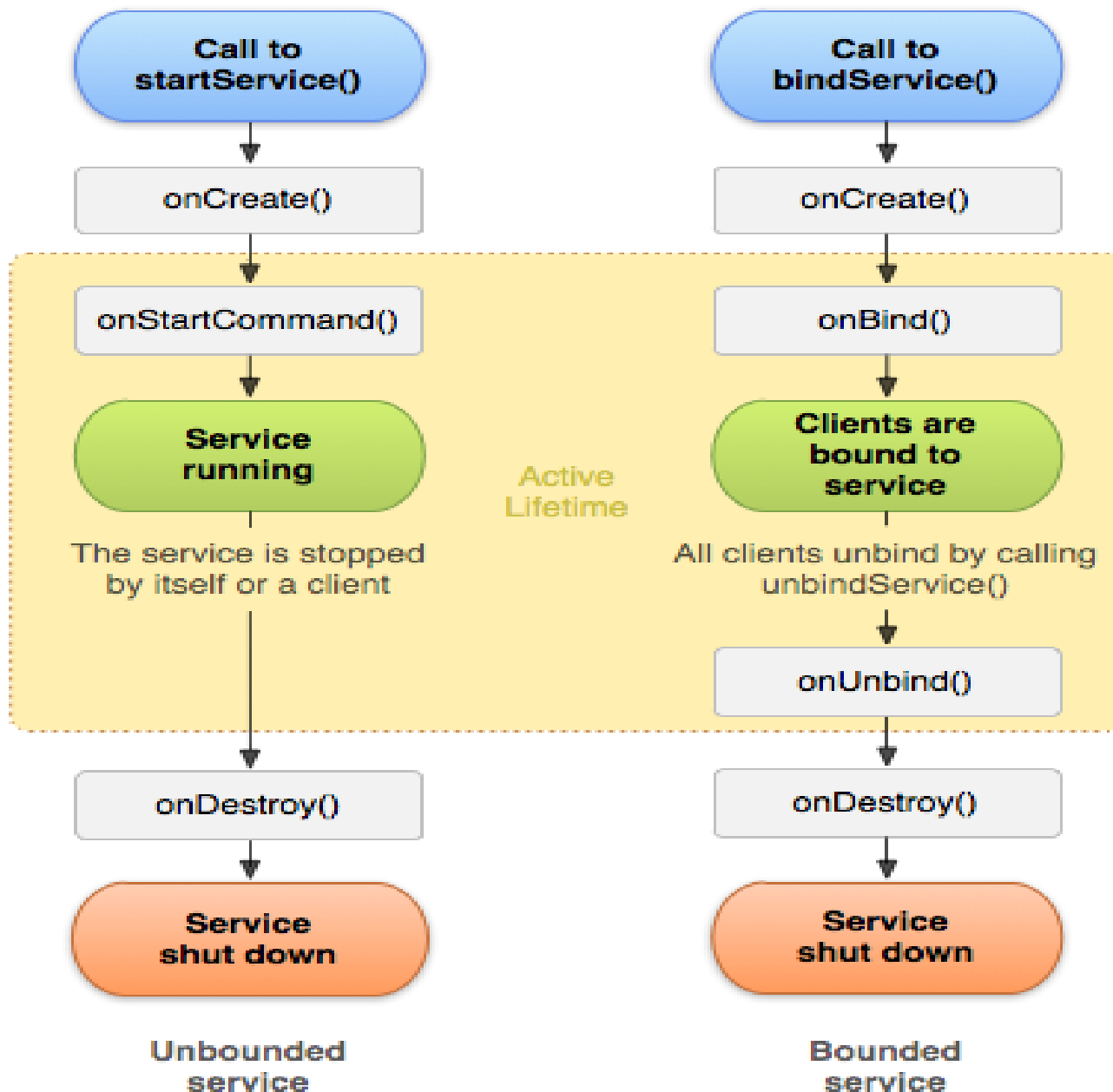
5.2 – Primena servisa u Android aplikacijama

- Bez obzira na tip servisa, komponente aplikacije mogu koristiti servis **isto kao i aktivnost** – startujući ga pomoću **intent**-a.
- Servis u okviru Android Software Development Kit (SDK) može da znači **dve stvari**:
 1. može da znači **proces u pozadini**, koji obavlja neke korisne informacije u redovnim intervalima.
 2. može biti **interfejs za udaljeni objekat** koji se zove u okviru vaše aplikacije.
- U oba slučaja **servis objekat proširuje klasu *Service*** u okviru Android SDK, i to može da bude **samostalna komponenta** ili **deo aplikacije** sa kompletnim korisničkim interfejsom.
- Kreirani servis radi **u procesu komponente koja ga je pozvala**, dakle on **ne započinje novi proces** ili novi **thread** ukoliko drugačije nije navedeno.
- U slučaju da servis radi **veliki posao** ili **više njih istovremeno**, onda se preporučuje **stvaranje novih thread-ova** unutar njega.
- Time se **smanjuje rizik** od pada aplikacije.

5.2 - Životni ciklus servisa

- Pre nego što se krene u detaljno objašnjenje kako kreirati servis, mora se prvo razumeti **na koji način servis ima interakciju** sa Android OS.
- Prilikom kreiranja servisa, pomenuti servis mora **prvo da se registruje u manifest fajlu** koji aplikacija koristi u okviru taga **<service>**.
- Takođe potrebno je **definisati dozvole potrebne za pokretanje, zaustavljanje i povezivanje** na servis, kao i **neophodni poziv servisa**.
- Nakon što je **servis implementaran**, servis se može koristiti korišćenjem metode **Context.startService()**.
- Ako je usluga servisa već pokrenuta, kasnijem korišćenjem metode **startService()**, usluga se ne pokreće ponovo.
- Servis funkcioniše dok se ne pozove metoda **Context.stopService()** ili servis završi sa radom i sam pozove f-ju za prestanak rada **stopSelf()**
- Aplikacije koje žele da koriste servis **moraju da pozovu** funkciju **Context.bindService()** za uspostavljanje veze sa servisom.
- Ako servis **nije pokrenut**, on se **pokreće u tom trenutku**, ako je servis već pokrenut, aplikacije **mogu da šalju zahteve za izvršavanje** određenih usluga ako za to **imaju dozvolu**.

5.2 - Životni ciklus servisa



5.2 - Kreiranje servisa

- Životni ciklus servisa je **jednostavniji** od životnog ciklusa aktivnosti.
- Kod servisa treba više obratiti pažnju na samu **njihovu kreiranje** i **uništenje** jer korisnik najčešće **nije svestan** njihovog funkcionisanja i uglavnom nema kontrolu nad njima.
- Za kreiranje servisa, neophodno je **stvoriti klasu** sa **Service** proširenjem i deklarirati je u **Manifest fajlu**.
- U klasi treba implementirati **Callback metode** i tako definisati servis:
1. onStartCommand() - sistem poziva metodu kada neka od komponenti aplikacije zatraži kreaciju servisa pomoću metode **startService()**. Tada se kreira pokrenuti servis koji radi u pozadini **nedefinisano dugo**. Treba narediti servisu njegovo uništenje kada svoj posao odradi pomoću metode **stopSelf()** ili **stopService()** koja se poziva iz komponente. Kod vezanih (**bind**) servisa implementacija ovih metoda nije potrebna.

5.2 - Kreiranje servisa

- 2. onBind()** - sistem poziva ovu metodu kada neka komponenta zahteva kreaciju vezanog servisa pomoću metode *bindService()*. Implementacija ove metode treba pružiti klijentima (komponentama koje je pozivaju) interfejs preko koga će oni komunicirati sa servisom i to radi tako što metoda vraća vrednost **IBinder**. Ova metoda mora uvek biti implementirana a ukoliko nije potrebno stvaranje vezanog servisa ona treba vratiti vrednost *null*.
- 3. onCreate()** - ova metoda se poziva odmah nakon što komponenta zatraži kreaciju servisa (pre no što se **onStartCommand()** ili **onBind()** pozovu). U kodu ove metode se implementira sama svrha servisa. Ukoliko je servis već pokrenut i radi, ova metoda se ne poziva.
- 4. onDestroy()** - poziva se nakon što servis izvrši svoj zadatak (kroz **stopService()**), ili se odveže od svih komponenata (**unbindService()**), i uništava ga.

5.2 - Životni ciklus servisa

➤ Implementacijom **Callback metoda** mogu se uočiti **dva ugnježdena stadijuma** životnog ciklusa servisa:

1. celokupan život servisa nalazi se između poziva **onCreate()** i **onDestroy()** metoda. Podrazumevano, u **onCreate()** treba definisati servis i njegovo ponašanje a u **onDestroy()** osloboditi sve resurse koje je on zauzimao. Bez obzira na to da li je servis pokrenut ili vezan on mora biti kreiran i uništen.
2. aktivan život servisa počinje pozivom **onStartCommand()** ili **onBind()** metode. Ovim metodama se, prilikom pozivanja servisa, prosleđuje **Intent** objekat kroz **startService()** ili **bindService()** respektivno. Aktivan život pokrenutog servisa se završava u istom trenutku kada i celokupan život, dok se kod vezanih servisa on završava pozivom **onUnbind()** metode.

➤ **Pokrenuti i vezani tip** servisa nisu striktno razdvojeni, tj. servis pokrenut sa **startService()** može biti naknadno vezan sa komponentom **Primer**: možemo pokrenuti servis koji pušta muziku u pozadini koji se posle može vezati za aktivnost Music player-a ukoliko korisnik to želi.

5.2 - Životni ciklus servisa

- Servisi **moгу obaveštavati** korisnika o svom radu na dva načina:
 1. preko **toast poruka**
 2. preko **obaveštenja u statusnoj liniji i statusnom prozoru.**
- **Toast** poruke su kratke poruke u crnim pravougaonicima koji se **pojavljuju na ekranu** preko primarne aktivnosti (fokusiranog ekrana) i **nestaju** posle par sekundi.
- Statusna linija se kod standardnog Android interfejsa (i aplikacija koje nisu full-screen) **nalazi na samom vrhu ekrana** i moguće je njeno proširenje na statusni prozor.
- U statusnoj liniji se često mogu videti **obaveštenja o progresu servisa** (otpočeto preuzimanje sadržaja sa interneta, preuzimanje je u toku, preuzimanje je završeno itd.)
- U statusnom prozoru se može **videti više informacija** o progresu servisa i **eventualno interagovati** sa njihovim rezultatima (pokretanje ili pregled preuzetog sadržaja, otvaranje primljenog email-a u većem prozoru itd.)

5.3 - Primena prijemnika poruka

3. Prijemnici poruka

- ✓ Prijemnici poruka (*broadcast receivers*) **upravljaju komunikacijom** između Android OS-a i aplikacija.
- ✓ Oni odgovaraju na poruke dobijene od drugih aplikacija ili sistema.
Primer: *aplikacije mogu uputiti poruku drugim aplikacijama obavještavajući ih da su neki podaci preuzeti, da se nalaze na uređaju i da su raspoloživi za korištenje.*
- ✓ Prijemnici poruka će **prepoznati ovu komunikaciju** i pokrenuti odgovarajuću akciju.
- ✓ Prijem.poruka **se implementira** kao podklasa *BroadcastReceiver* klase, a svaka poruka je prijemnik poruke koji je *Intent* objekat.

Primer:

```
public class MyReceiver extends BroadcastReceiver {  
    public void onReceive(context, intent) {  
    }  
}
```

5.3 – Primena prijemnika poruka

- Prijemnici poruka su komponente koje reaguju **na emitovana obaveštenja i najave** u sistemu.
- Mnoge poruke stižu od samog sistema, kao što je obaveštenje da je **baterija prazna**, da je **ekran isključen**, da je *print screen* komanda **uspešno sačuvala sliku** itd.
- Aplikacije **takođe mogu emitovati obaveštenja**, na primer kada uspešno skinu sadržaj sa Interneta i žele da obaveste druge aplikacije da ga mogu koristiti.
- Iako prijemnici poruka **ne mogu pružiti korisnički interfejs**, oni mogu stvoriti obaveštenje **u statusnoj liniji** i tako obavestiti korisnika.
- Prijemnici poruka najčešće **samo signaliziraju** drugim komponentama da počnu sa izvršavanjem određenih zadataka i sami po sebi **ne rade velike poslove**.
- Prijemnici se implementiraju **u Java klasama** sa proširenjem *BroadcastReceiver* i aktivira ih prosleđeni *Intent* objekat, slično kao i aktivnosti.

5.3 - Mehanizmi interprocesne komunikacije

- Android OS karakteriše upotreba **inter-procesne komunikacije** (IPC) u okviru same aplikacije kao i između aplikacija međusobno.
- Android OS svaku aplikaciju pokreće u zasebnom procesu (**sandbox**) sa **jedinstvenim sistemskim identifikatorom (ID)**.
- Svaka aplikacija **ima svoju zasebnu instancu Dalvik VM**.
- Na ovaj način aplikacije su u **potpunosti nezavisne jedne od drugih** sa sopstvenim adresnim prostorom.
- Ovakva implementacija omogućava **sigurnost i bezbednost** jer jedan proces ne može direktno upravljati memorijom drugog procesa
- U mnogim slučajevima **komunikacija između procesa je neophodna**
- Zato OS **mora da obezbedi mehanizme** za inter-procesnu komunikaciju
- Android OS sadrži jedan oblik interprocesne komunikacije koji je **karakterističan samo za njega**, a to je **Binder okvir**.
- IPC na Android OS se **većinom odvija preko Binder** okvira i u **maloj meri na soketima** koji se koriste na najnižem nivou.

5.3 - Mehanizmi inerprocesne komunikacije

- Inter-procesna komunikacija u **Binder** okviru je implementirana kao **klijent-servis odnos**.
- Na Android platformi postoje **tri mehanizma**, tj. implementacije inter-procesne komunikacije koja se **odvija preko Binder okvira**:
 - 1. Intent** mehanizam,
 - 2. Messenger** mehanizam
 - 3. Mehanizam** baziran na jeziku za definisanje Android interfejsa-**AIDL**
- **Messenger** i **AIDL** implementacije rade sa **vezanim** (*bound*) servisima, dok **Intent** objekti rade sa **pokrenutim** (*started*) servisima.
- Tri od četiri tipa komponenti - **aktivnosti**, **servisi** i **prijemnici poruka** pokreću se pomoću asihrone poruke zvane **Intent**.
- **Intent objekat** spaja individualne komponente za vreme *runtime*-a **bez obzira** na to da li komponente pripadaju istoj aplikaciji.
- **Intent** objekat se može zamisliti kao glasnik koji **od komponente zahteva izvesnu akciju** ili delovanje.
- **Intent** se stvara kao **apstraktni objekat u kom se definiše poruka** koja se šalje komponenti.

5.3 - Mehanizmi inerprocesne komunikacije

- Pod apstraktnim se misli na to da **komponente**, koje treba da izvrše akciju, **ne moraju biti definisane** u **Intent** objektu.
- **Intenti** se sastoje iz dva dela **akcije** i **podataka**.
- **Intent** objekti se mogu zamisliti **kao poruke koje se šalju širom OS** kako bi **izvršile određenu akciju** na nekoj komponenti koja ima odgovarajući filter za tu akciju.
- Osnovne komponente svake aplikacije **imaju definisane filtere** u AndroidManifest.xml fajlu te aplikacije.
- **Filter** jedne komponente **sadrži akcije** koje podržava ta komponenta.
- Kad se pošalje **intent**, Android OS **pronalaži sve komponente** koje sadrže **filter** za tu akciju.
- Ukoliko postoji više komponenti sa tim **filterom**, Android OS **prepušta korisniku** da **izabere** željenu komponentu.
- Postoje **dve vrste** Intent objekata **eksplicitni** i **implicitni**.
- **EksPLICITNI** gađaju određenu komponentu **po imenu** i **ograničen** je samo na komponente unutar jedne aplikacije.
- **Implicitni**, pokušavaju da **pronađu komponente po akciji** i **nisu ograničeni** na komponente jedne aplikacije.

5.3 - Mehanizmi inerprocesne komunikacije

- **EksPLICITNI** Intent objekat **može aktivirati određenu** komponentu dok **implicitni** može aktivirati **određeni tip** komponenti.
- **Intent** objekat prosleđen **aktivnostima i servisima** definiše **akciju koju treba da izvrše** (da prikažu ili pošalju nešto) i takođe može sadržati URI podataka **nad kojima ona treba da bude izvršena** (kao i ostale stvari koje bi novopokrenuta komponenta trebala da zna).
Primer: *intent-om se od aktivnosti može zatražiti da prikaže određenu sliku ili otvori određenu web stranicu.*
- U nekim slučajevima je **moгуće pokrenuti aktivnost** kako bi ona vratila rezultat, takođe kroz **Intent** objekat (na primer, on može pitati korisnika da odabere neku osobu iz kontakta i vraćeni **Intent** objekat će sadržati URI tog kontakta).
- **Intent** objekat prosleđen prijemniku poruka **samo definiše poruku** koja će biti prikazana kao obaveštenje.
- Poslednji tip komponenti, **dobavljači sadržaja**, se ne aktiviraju **Intent** objektima, već **zahtevima** koje mu **ContentResolver** može uputiti.

5.3 – Mehanizmi inerprocesne komunikacije

- **Content Resolver** kontroliše **sve direktne transakcije** između komponente i dobavljača kako komponenta **ne bi morala to da radi**.
- Ona samo **poziva metode** iz **ContentResolver** objekta i tako se stvara **apstraktni odnos** komponenta-dobavljač radi sigurnosti.
- Objekat **Intent** je **pasivna struktura podataka** koja drži apstraktan opis operacija koje treba da se izvrše, ili u slučaju **broadcast risivera**, **opis nečega** što se dogodilo ili se objavljuje.
- Svaki tip komponenti se **aktivira posebnim metodama**:
 - 1. Aktivnost** se pokreće (ili joj se zadaje novi zadatak) prosleđivanjem Intent objekta metodi **startActivity()**. Kada se od aktivnosti očekuje da vrati rezultat, on se prosleđuje metodi **startActivityForResult()**.
 - 2. Servis** se pokreće (ili mu se daju nove instrukcije ukoliko je već pokrenut) prosleđivanjem Intent objekta metodi **startService()**. Servis se može vezati prosleđivanjem Intent objekta metodi **bindService()**.
 - 3. Emitovanje** se može inicirati prosleđivanjem Intent objekta metodi **sendBroadcast()**, **sendOrderedBroadcast()** ili **sendStickyBroadcast()**.
 - 4. Dobavljaču sadržaja** se može postaviti upit pozivanjem metode **query()** kod **ContentResolver**-a.

5.4 - Primena provajdera sadržaja

4. Provajderi sadržaja

- ✓ Provajderi sadržaja (*content providers*) upravljaju deljenim podacima aplikacije.
- ✓ Ova komponenta (na zahtev) obezbeđuje podatke iz jedne aplikacije drugima.
- ✓ Podaci se mogu nalaziti u bazi podataka, datotečnom sistemu ili na nekom drugom mestu.
- ✓ Provajder sadržaja se implementira kao podklasa **ContentProvider** klase, i mora implementirati standardni skup API-a kako bi omogućio drugim aplikacijama izvršenje transakcija.

Primer:

```
public class MyContentProvider extends ContentProvider {  
    public void onCreate() {  
    }  
}
```

5.5 – Dodatne komponente Android aplikacije

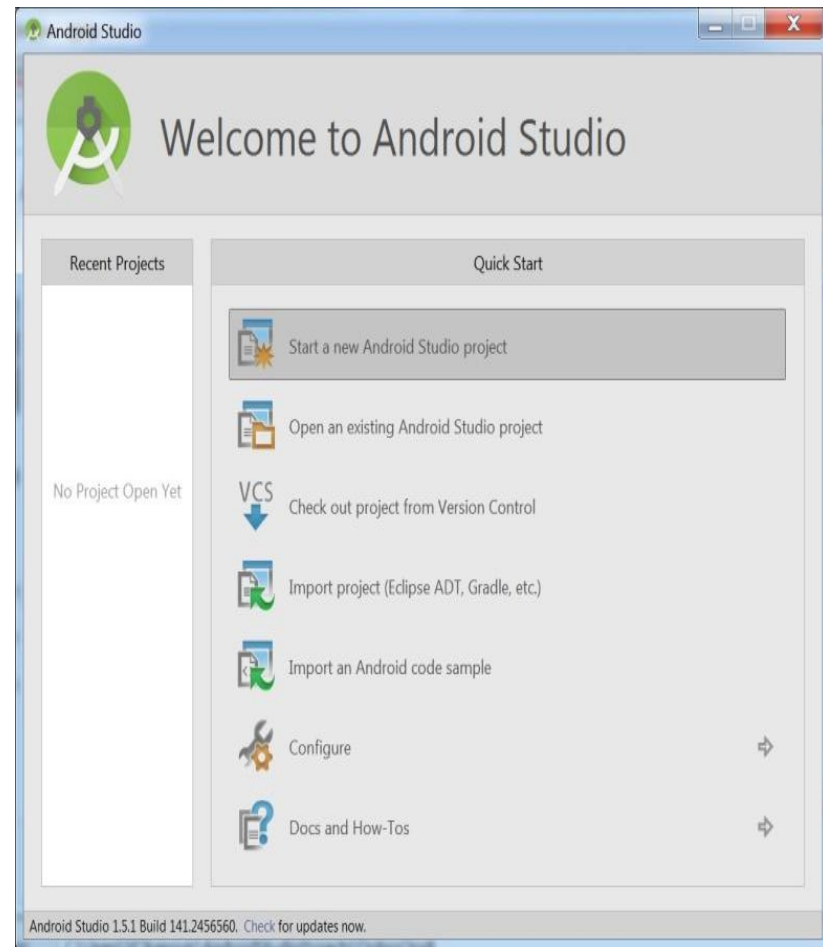
- Postoje **dodatne komponente** koje se mogu koristiti u razvoju aplikacija, odnosno pripadajućih entiteta, njihove logike i međusobnog povezivanja.
- Dodatne komponente su:
 - 1. Fragmenti** (*fragments*) – predstavljaju **deo korisničkog interfejsa** u nekoj aktivnosti
 - 2. Pogledi** (*views*) – **elementi korisničkog interfejsa** koji su iscrtani, odnosno predstavljeni na ekranu, kao što su **dugmići, liste, kvadratići za izbor** (*checkbox*)
 - 3. Prikazi** (*layouts*) – hijerarhija pogleda koja **kontrolira format ekrana i izgled** pogleda
 - 4. Namere** (*intents*) – **poruke** koje povezuju komponente
 - 5. Resursi** (*resources*) – **eksterni elementi**
 - 6. Manifest** – **konfiguraciona datoteka** za aplikaciju

5.6- Razvoj aplikacije u Android studiju

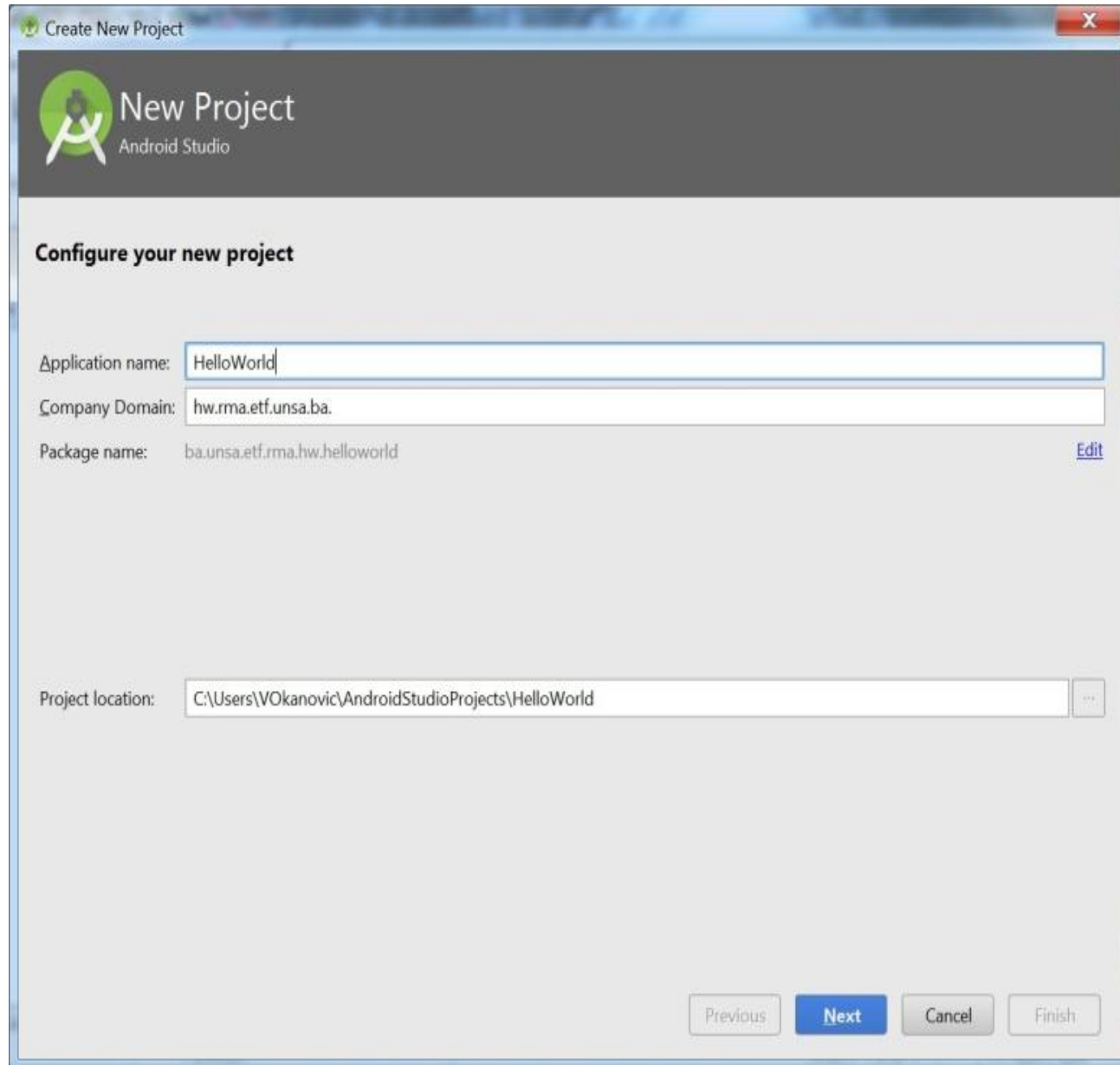
- Pre početka razvoja aplikacije pomoću Android SDK potrebno je **postaviti Android razvojno okruženje**.
- Nova aplikacija/projekat u Android Studio-u se može kreirati tako da se nakon pokretanja Android Studio-a izabere **Start a new Android Studio project**
- Nova aplikacija/projekat se može kreirati i u radnom okruženju Android Studia, izborom **File ->New -> New project....** nakon čega je (u **Configure your new project**) potrebno uneti podatke koji se odnose na:

1. Application name (HelloWorld) ,
2. Company Domain (vtsnis.edu.rs.)
3. Project location


...\\AndroidStudioProjects\\HelloWorld



5.6- Razvoj aplikacije u Android studiju



Create New Project

 New Project
Android Studio

Configure your new project

Application name:

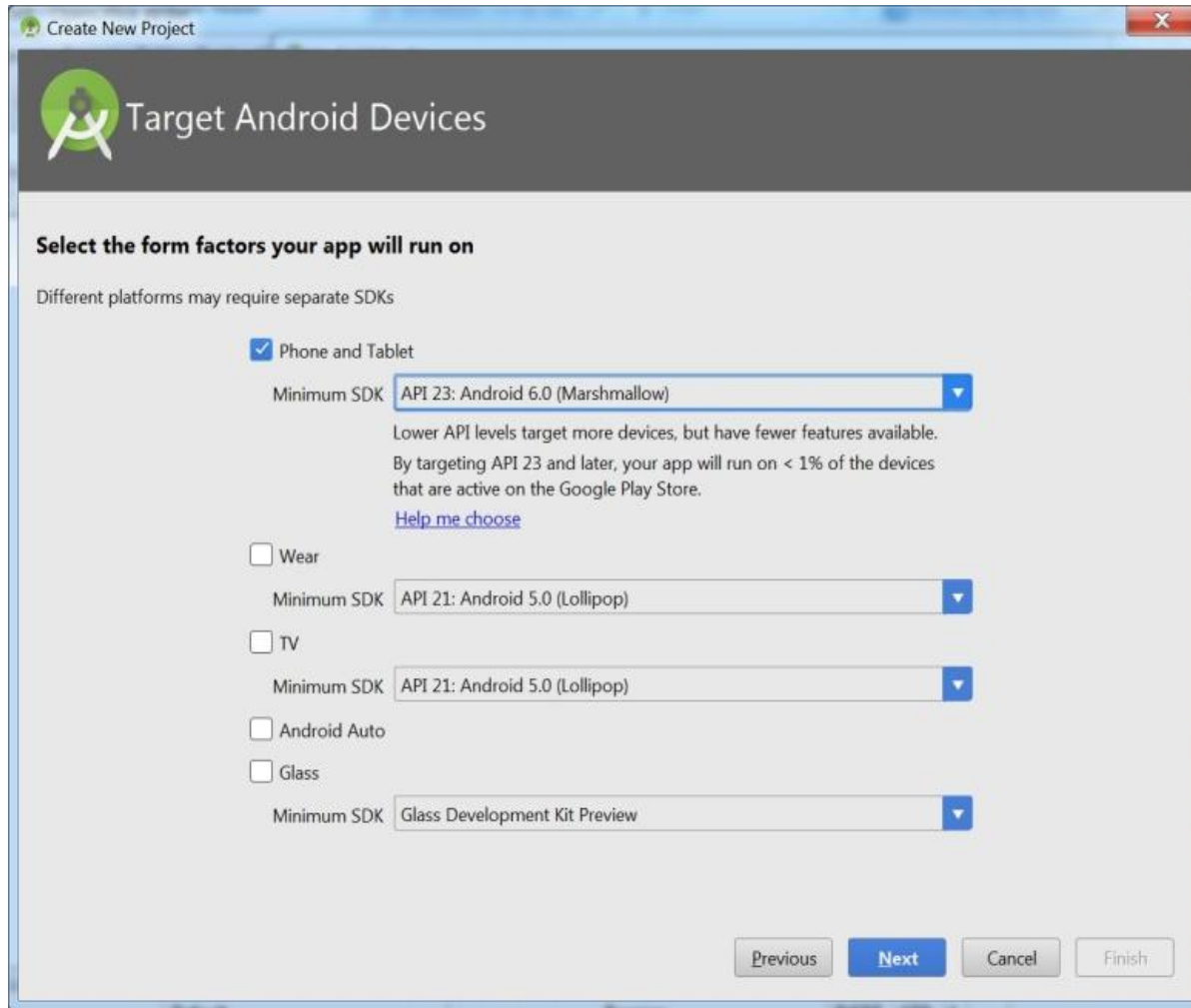
Company Domain:

Package name: [Edit](#)

Project location: ...

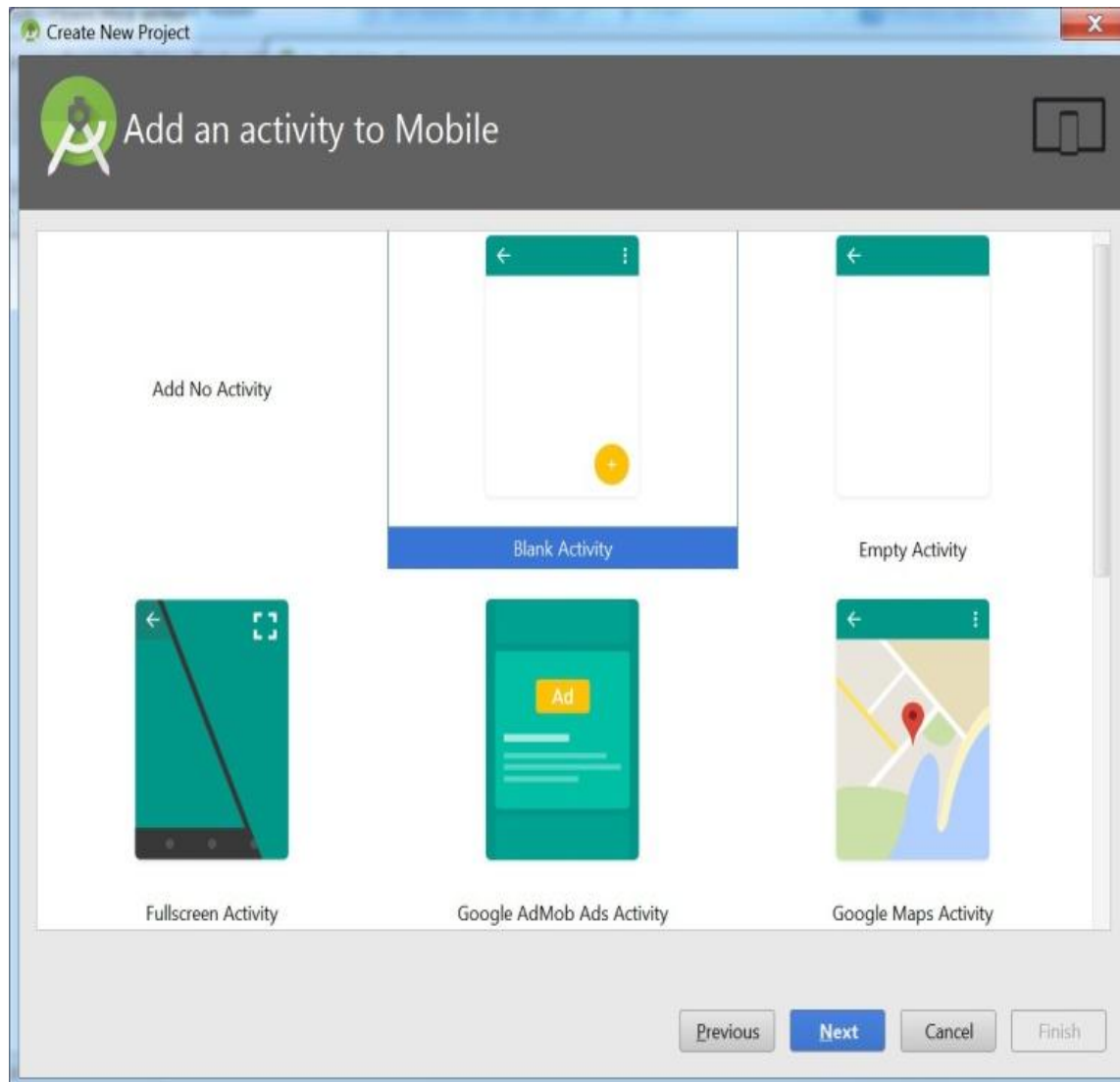
5.6- Razvoj aplikacije u Android studiju

- U narednom koraku (**Select the form factors your app will run on**) je potrebno odabrati **platformu** na kojoj će se aplikacija izvršavati, te **verziju** Android SDK.



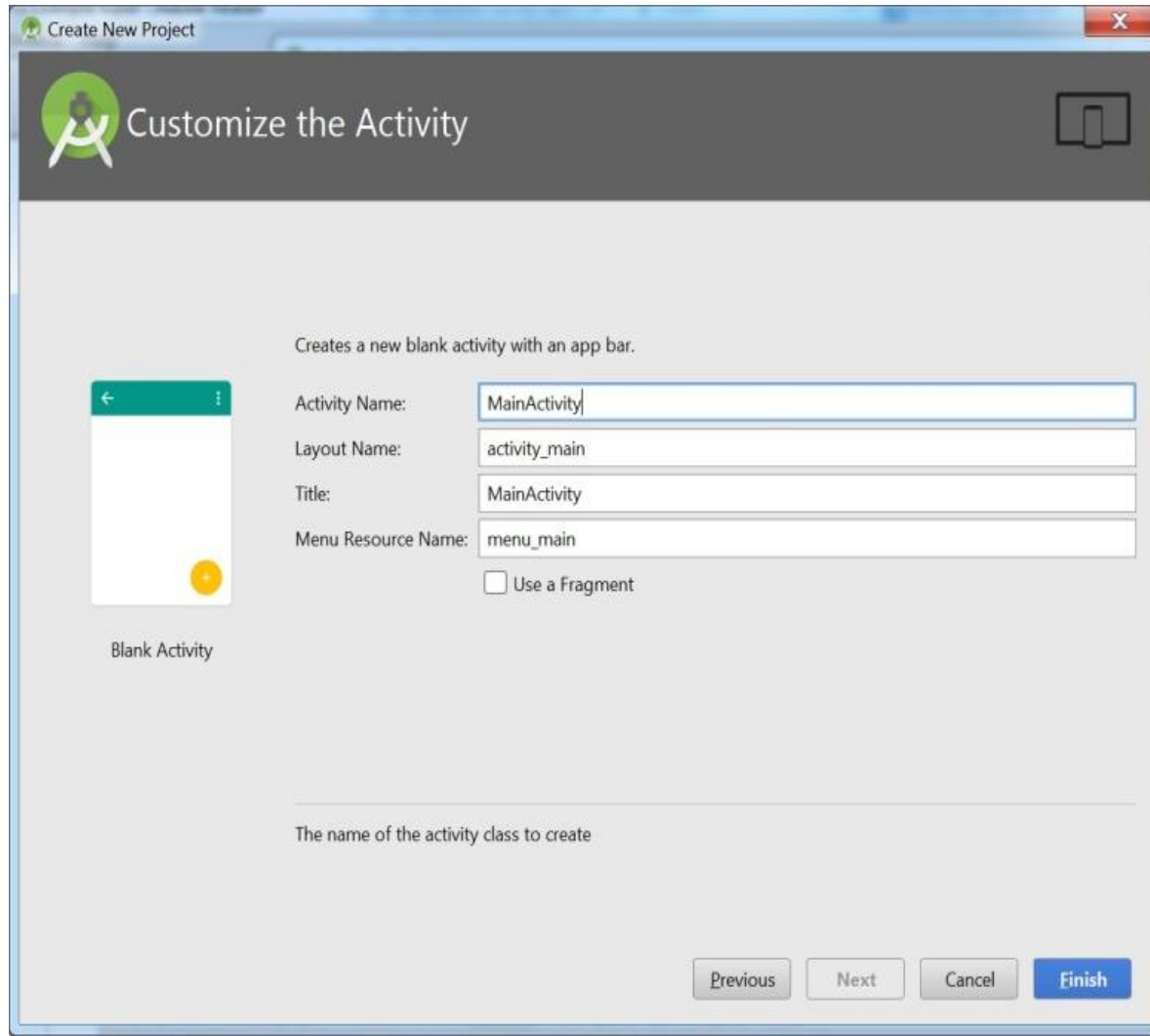
5.6- Razvoj aplikacije u Android studiju

- U **Add an activity to Mobile** je omogućeno dodavanje i izbor nove aktivnosti za mobilni uređaj.



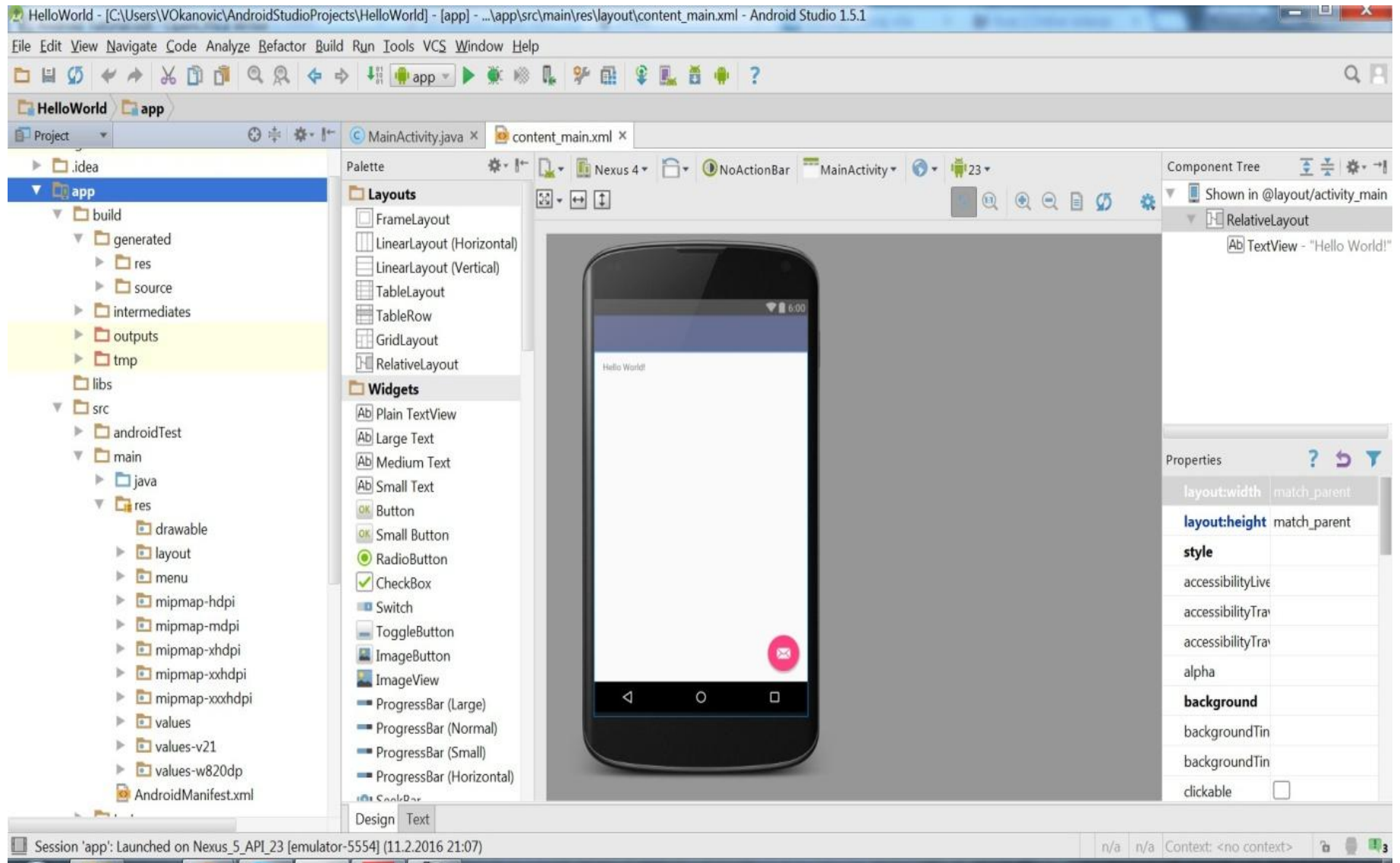
5.6- Razvoj aplikacije u Android studiju

- U nastavku (**Customise the Activity**) treba dati naziv određenom projektu/aktivnosti, te izborom **Finish** završiti proces kreiranja novog projekta/aplikacije, sa sledećim izgledom:



5.6- Razvoj aplikacije u Android studiju

➤ Konačan izgled unetog projekta je:



5.7 - Struktura aplikacije u Android-u

➤ Struktura Android projekta se sastoji od nekoliko foldera i datoteka:

- 1. build** – sadrži generisane datoteke, kao što su Aidl, Build konfiguracija i R (R.java)
- 2. lib** – folder za biblioteke potrebne za razvoj android aplikacija
- 3. src** – sadrži izvorne (.java) datoteke projekta. Ovde se nalazi datoteka MainActivity.java koja sadrži klasu aktivnosti, a koja se izvršava pri pokretanju aplikacije.
- 4. res** –sadrži slike, prikaze, vrednosti i Android manifest datoteku
- 5. res/drawable** – folder za slike, odnosno objekte dizajnirane za ekrane
- 6. res/layout** – folder koji sadrži datoteke koje određuju korisnički interfejs aplikacije
- 7. res/menu** – folder za objekte menija koji su dizajnirani za gradnju menija u aplikaciji
- 8. res/values** – folder za razne XML datoteke koje sadrže resurse, kao što su stringovi i boje
- 9. AndroidManifest.xml** – manifest datoteka koja opisuje bitne osobine aplikacije i definiše svaku njenu komponentu

Hvala na pažnji !!!



Pitanja

? ? ?