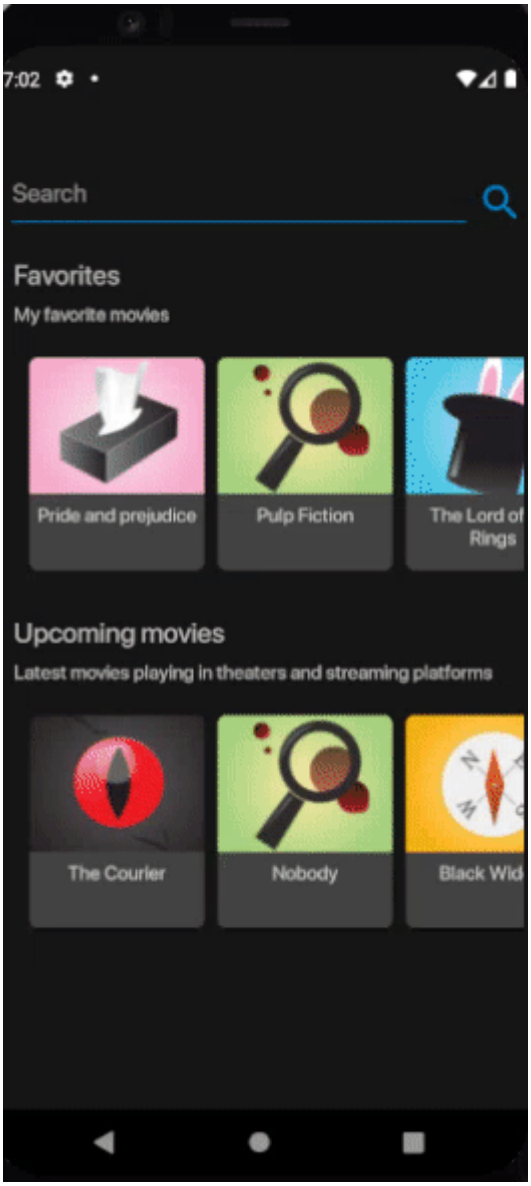
Laboratorijska vježba 3

Intent i broadcast receiver, testiranje mobilne aplikacije

Funkcionlanosti aplikacije nakon ove vježbe:

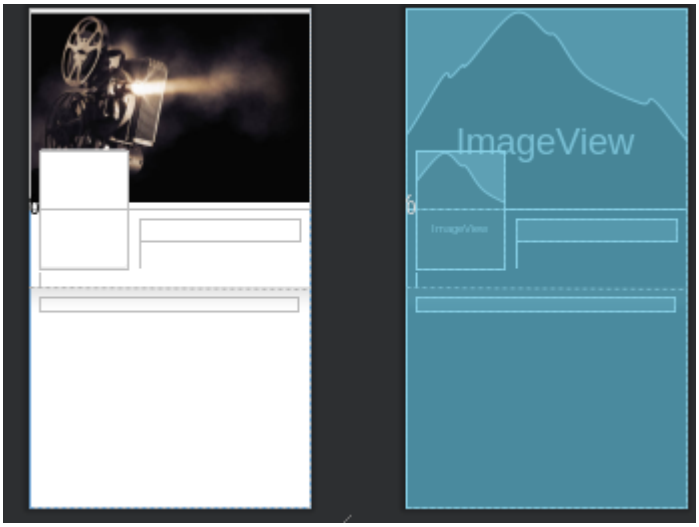


Zadatak 1.

Kreirati novu aktivnost u sklopu koje će biti prikazani detalji o filmu.

Desni klik na app i odabir stavki `New -> Activity -> EmptyActivity` otvaraju prozor za kreiranje aktivnosti. Kreirajte novu aktivnost `MovieDetailActivity`.

Nakon kreiranja aktivnosti definišime layout iste. Primjer izgleda je dat u nastavku:



XML file nove aktivnosti je dat u nastavku:

```
<androidx.core.widget.NestedScrollView
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"
    android:fillViewport="true">
    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <ImageView
            android:id="@+id/movie_backdrop"
            android:layout_width="0dp"
            android:layout_height="0dp"
            app:layout_constraintBottom_toBottomOf="@+id/backdrop_guideline"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            app:srcCompat="@drawable/backdrop" />
        <androidx.cardview.widget.CardView
            android:id="@+id/movie_poster_card"
            android:layout_width="128dp"
            android:layout_height="172dp"
            android:layout_marginStart="16dp"
            android:layout_marginEnd="8dp"
            app:cardCornerRadius="4dp"
            app:layout_constraintBottom_toBottomOf="@+id/backdrop_guideline"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@id/backdrop_guideline">
            <ImageView
                android:id="@+id/movie_poster"
                android:layout_width="match_parent"
```

```
            android:layout_height="match_parent" />
    </androidx.cardview.widget.CardView>
    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/backdrop_guideline"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        app:layout_constraintGuide_percent="0.4" />
    <TextView
        android:id="@+id/movie_title"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="16dp"
        android:layout_marginEnd="16dp"
        android:textColor="@android:color/white"
        android:textSize="24sp"
        android:textStyle="bold"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/movie_poster_card"
        app:layout_constraintTop_toBottomOf="@+id/backdrop_guideline" />
    <TextView
        android:id="@+id/movie_release_date"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#757575"
        android:textSize="14sp"
        app:layout_constraintStart_toStartOf="@+id/movie_title"
        app:layout_constraintTop_toBottomOf="@+id/movie_title" />
    <TextView
        android:id="@+id/movie_genre"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#757575"
        android:textSize="14sp"
        app:layout_constraintStart_toStartOf="@id/movie_release_date"
        app:layout_constraintTop_toBottomOf="@id/movie_release_date" />
    <TextView
        android:id="@+id/movie_website"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:textColor="@color/highlightColor"
        android:textSize="14sp"
        app:layout_constraintStart_toStartOf="@id/movie_poster_card"
        app:layout_constraintTop_toBottomOf="@id/movie_poster_card" />
    <androidx.constraintlayout.widget.Barrier
        android:id="@+id/movie_poster_title_barrier"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:barrierDirection="bottom"
```

```
app:constraint_referenced_ids="movie_release_date,movie_poster_card,movie_genre,movie_we

            tools:layout_editor_absoluteY="379dp" />
        <TextView
            android:id="@+id/movie_overview"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginStart="16dp"
            android:layout_marginTop="16dp"
            android:layout_marginEnd="16dp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/movie_poster_title_barrier"
            tools:textSize="16sp" />
    </androidx.constraintlayout.widget.ConstraintLayout>
</androidx.core.widget.NestedScrollView>
```

Zadatak 2.

Potrebno je dodati akciju koja se dešava prilikom klika na listu filmova, takvu da se otvara aktivnost koja prikazuje detalje o filmu. Potrebno je proslijediti naziv filma iz liste u sljedeću aktivnost te korištenjem ViewModela-a pronaći film i prikazati vrijednosti u odgovarajućim poljima.

Izvršimo prvo kreiranje intenta. U glavnoj aktivnosti definišimo funkciju koja će se izvršiti na click. Unutar funkcije se kreira intent i dodatni parametri.

```
private fun showMovieDetails(movie: Movie) {
    val intent = Intent(this, MovieDetailActivity::class.java).apply {
        putExtra("movie_title", movie.title)
    }
    startActivity(intent)
}
```

Nakon definisanja funkcije, odredimo gdje ćemo postaviti `onListItemClickListener` ili `onClickListener`. Preporuka je da se isti deifniše unutar `onBindViewHolder-a`.

U prvom koraku u konstruktor našeg adaptera ćemo dodati funkciju višeg reda (**high-order function**) koja prima funkciju i vraća void. U Javi bi se dodao interfejs.

```
class MovieListAdapter(
    private var movies: List<Movie>,
    private val onItemClicked: (movie:Movie) -> Unit
    ) : RecyclerView.Adapter<MovieListAdapter.MovieViewHolder>() {
```

Nakon što smo dodali istu u konstruktor, potrebno je sada da definišemo da se ista koristi u `onBindViewHolder-u`. Na kraj metode ćemo dodati sljedeće:

```
holder.itemView.setOnClickListener{ onItemClicked(movies[position]) }
```

Obzirom da smo izvršili izmjenu nad konstruktorom, sada je potrebno da drugačije inicijaliziramo naš adapter. Konkretnu implementaciju funkcije ćemo sada proslijediti.

```
favoriteMoviesAdapter = MovieListAdapter(arrayListOf()) { movie ->
showMovieDetails(movie) }
recentMoviesAdapter = MovieListAdapter(arrayListOf()) { movie ->
showMovieDetails(movie) }
```

Omogućili smo pokretanje nove aktivnosti na klik. Definišimo sada ponašanje nove aktivnosti. Prvo ćemo implementirati MovieDetailViewModel preko kojeg ćemo filtrirati postojeće liste iz repozitorija kako bi dobili odgovarajući film.

```
class MovieDetailViewModel {
    fun getMovieByTitle(name:String):Movie{
        var movies: ArrayList<Movie> = arrayListOf()
        movies.addAll(MovieRepository.getRecentMovies())
        movies.addAll(MovieRepository.getFavoriteMovies())
        val movie= movies.find { movie -> name.equals(movie.title) }
        //ako film ne postoji vratimo testni
        return movie?:Movie(0,"Test","Test","Test","Test","Test")
    }
}
```

U nastavku je dat prikaz klase `MovieDetailActivity`.

```
class MovieDetailActivity : AppCompatActivity() {
    private var movieDetailViewModel =  MovieDetailViewModel()
    private lateinit var movie: Movie
    private lateinit var title : TextView
    private lateinit var overview : TextView
    private lateinit var releaseDate : TextView
    private lateinit var genre : TextView
    private lateinit var website : TextView
    private lateinit var poster : ImageView
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_movie_detail)
        title = findViewById(R.id.movie_title)
        overview = findViewById(R.id.movie_overview)
        releaseDate = findViewById(R.id.movie_release_date)
        genre = findViewById(R.id.movie_genre)
        poster = findViewById(R.id.movie_poster)
        website = findViewById(R.id.movie_website)
        val extras = intent.extras
        if (extras != null) {
           movie =
movieDetailViewModel.getMovieByTitle(extras.getString("movie_title",""))
            populateDetails()
        } else {
            finish()
        }
    }
    private fun populateDetails() {
        title.text=movie.title
        releaseDate.text=movie.releaseDate
```

```
        genre.text=movie.genre
        website.text=movie.homepage
        overview.text=movie.overview
        val context: Context = poster.getContext()
        var id: Int = context.getResources()
            .getIdentifier(movie.genre, "drawable", context.getPackageName())
        if (id===0) id=context.getResources()
            .getIdentifier("picture1", "drawable", context.getPackageName())
        poster.setImageResource(id)
    }
}
```

Nakon pokretanja dobijemo sljedeći izgled detalja filma.

Zadatak 3.

Potrebno je napraviti da se pokrene web preglednik kada se klikne na link web stranice filma i da se učita navedena stranica.

Postavimo prvo `setOnClickListener`:

```
website.setOnClickListener{
        showWebsite()
}
```

Definišimo funkciju `showWebsite()` na osnovu uputa iz teoretskog uvoda.

```
private fun showWebsite(){
    val sendIntent = Intent().apply {
        action = Intent.ACTION_VIEW
        setData(Uri.parse(movie.homepage))
    }
    if (sendIntent.resolveActivity(packageManager) != null) {
        startActivity(sendIntent)
    }
}
```

Zadatak 4.

Potrebno je napraviti da vaša aplikacija odgovara na akciju tipa `ACTION_SEND`, ako se putem te akcije prosljeđuje tekst. Navedeni tekst treba da popuni editText u početnoj aktivnosti koji se nalazi iznad button-a pretrage.

Prvo ćemo u `Manifest` file unutar tag-a glavne aktivnosti definisati `intent-filter` na koji odgovara naša aplikacija.

```
<intent-filter>
    <action android:name="android.intent.action.SEND" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="text/plain" />
</intent-filter>
```

Unutar `onCreate` glavne aktivnosti provjerit ćemo da li postoji intent s odgovarajućim tipom i proslijediti ga funkciji ako postoji:

```
if(intent?.action == Intent.ACTION_SEND && intent?.type == "text/plain")
    handleSendText(intent)
```

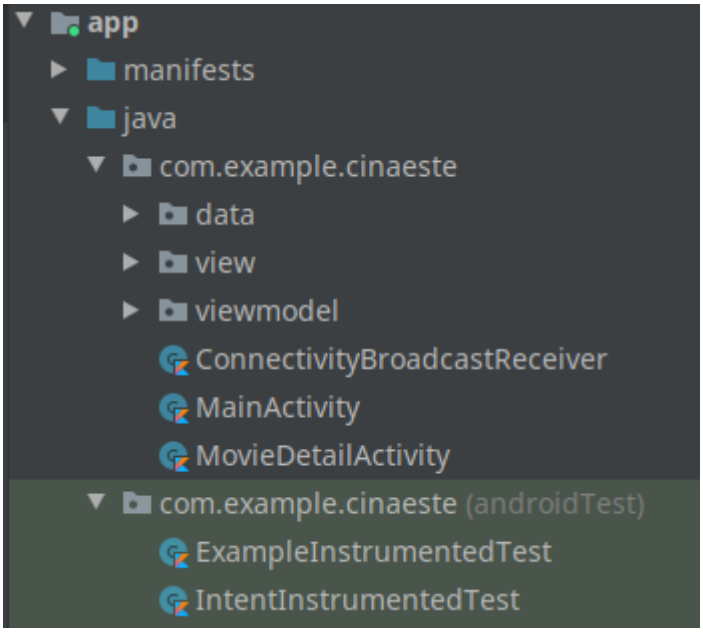
Funkcija koja će handle intent:

```
private fun handleSendText(intent: Intent) {
    intent.getStringExtra(Intent.EXTRA_TEXT)?.let {
        searchText.setText(it)
    }
}
```

Zadatak 5.

Potrebno je napraviti instrumented testove za aktivnost `MovieDetailsActivity`. U testovima provjerite da li se ispravno popunjava korisnički interfejs aktivnosti sa podacima odgovarajućeg filma čiji naziv je proslijeđen u extra podacima intenta. Nakon što testirate instanciranje aktivnosti testirajte i to da li klik na link poziva intent sa akcijom `Intent.ACTION_VIEW`

Prvo je potrebno dodati dependency u `app/gradle.build` i dodati klasu za testove u folder `app/src/androidTest/` koju ćemo nazvati `IntentInstrumentedTest`.



U klasi dodajte `IntentTestRule` koji ima tri parametra, prvi je specifikacija klase `MovieDetailsActivity::class.java`, drugi i treći imaju vrijednost false i ovime kažemo da se sama aktivnost ne pokreće na početku. Aktivnost ćemo pokretati u testovima.

Testno pravilo:

```
@get:Rule
var intentsRule: IntentsTestRule<MovieDetailActivity> =
IntentsTestRule(MovieDetailActivity::class.java,false,false)
```

Ovi početni koraci su opisani u teorijskom dijelu vježbe.

Testiranje instanciranja klase i popunjavanja korisničkog interfejsa: Nakon što je klasa pripremljena dodajmo metodu prvog testa:

```
@Test
fun testDetailActivityInstantiation(){
    val pokreniDetalje: Intent = Intent(MovieDetailActivity::javaClass.name)
    pokreniDetalje.putExtra("movie_title","Pulp Fiction")
    intentsRule.launchActivity(pokreniDetalje)
    onView(withId(R.id.movie_title)).check(matches(withText("Pulp Fiction")))
    onView(withId(R.id.movie_genre)).check(matches(withText("crime")))
    onView(withId(R.id.movie_overview)).check(matches(withSubstring("pair of diner
bandits")))
}
```

U prve dvije linije pripremamo intent kojim ćemo otvoriti aktivnost detalja filma sa nazivom *'Pulp fiction'*. Intent pokrećemo koristeći `intentsRule` i metodu `launchActivity()`. U testu provjeravamo da li `TextView` elementi naziva, žanra i opisa filma imaju odgovarajuće vrijednosti.

U ovaj test trebamo dodati i provjeru da li je slika žanra filma ispravno popunjena. Da bi ovo uradili trebamo implementirati vlastiti view matcher. Za implementaciju koristit ćemo apstraktnu klasu `TypeSafeMatcher<View>`. Ova klasa zahtjeva da budu implementirane metode `describeTo` i `matchesSafely`. Metoda `describeTo(decrtiption:Description)` dodaje poruku u rezultat testa ukoliko provjera nije rezultirala sa true vrijednosti, a u metodi `matchesSafely(item:View)` vršimo odgovarajuću provjeru. Ovaj matcher vrši provjeru da li je item koji se testira ispravnog tipa i da nije null.

Provjera da li slika žanra odgovara slici žanra iz resursa android aplikacije:

```
fun withImage(@DrawableRes id: Int) = object : TypeSafeMatcher<View>(){
    override fun describeTo(description: Description) {
        description.appendText("Drawable does not contain image with id: $id")
    }

    override fun matchesSafely(item: View): Boolean {
        val context:Context = item.context
        val bitmap: Bitmap? = context.getDrawable(id)?.toBitmap()
        return item is ImageView && item.drawable.toBitmap().sameAs(bitmap)
    }

}
```

Ovaj mathcer možete dodati unutar klase testa ili ga izdvojiti u posebnu utility klasu. Provjera koja se vrši u matcheru je da je item koji se testira tipa `ImageView` i da bitmap zapis slike koji se nalazi u tom itemu odgovara bitmap zapisu slike drawable resursa sa id-em koji je proslijeđen kao parametar našeg custom `withImage(id:Int)` matchera.

U test `testDetailActivityInstantiation()` možemo sada dodati i provjeru da li je slika žanra filma ispravno popunjena:

```
    onView(withId(R.id.movie_poster)).check(matches(withImage(R.drawable.crime)))
```

Ovaj test bi sada trebao ispravno proći. Ukoliko izmijenite drawable i stavite npm `R.drawable.family` test će pasti i vidjet ćete poruku koja se generiše u metodi `describeTo` našeg custom matchera.

Testiranje da li klik na link aktivira intent sa akcijom `Intent.ACTION_VIEW`

Dodajte novi test u prethodnu klasu `IntentInstrumentedTest` kojeg ćemo nazvati `testLinksIntent()` :

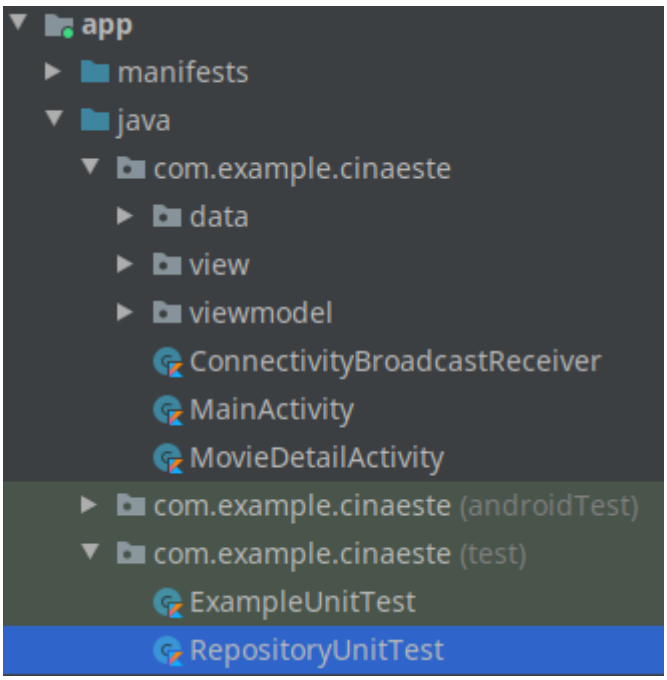
```
@Test
fun testLinksIntent(){
    val pokreniDetalje: Intent = Intent(MovieDetailActivity::javaClass.name)
    pokreniDetalje.putExtra("movie_title","Pulp Fiction")
    intentsRule.launchActivity(pokreniDetalje)
    onView(withId(R.id.movie_website)).perform(click())
    intended(hasAction(Intent.ACTION_VIEW))
}
```

Ovaj test pokreće aktivnost detalja filma kao i prethodni. Nakon pokretanja aktivnosti simulira se klik na link (element sa id-em R.id.moview_website). Sa metodom `intended()` provjeravamo da li pozvani intent ima akciju `Intent.ACTION_VIEW`. Ovaj test bi trebao ispravno proći, ali mu može biti potrebno više vremena da prođe nego prethodni jer test čeka da vidi rezultat intenta. Ako promijentite akciju u testu i stavite npr. `Intent.ACTION_SEND` test će pasti nakon što istekne time-out jer se nije pojavio intent sa ovakvom akcijom.

Zadatak 6.

Potrebno je napraviti unit test za testiranje repozitorija

Za razliku od instrumented testova, unit testovi se pokreću lokalno na računaru gdje razvijate aplikaciju. Testna klasa lokalnih unit testova se treba kreirati u folderu `app/src/test/`, nazvat ćemo je `RepositoryUnitTest`.



Da bi koristili matchere koji nam olakšavaju provjere stanja komponenti trebate u `app/build.gradle` dodati dependency:

```
testImplementation("org.hamcrest:hamcrest:2.2")
```

U testnu klasu dodajte metodu `testGetFavoriteMovies`:

```
@Test
fun testGetFavoriteMovies(){
    val movies = MovieRepository.getFavoriteMovies()
    assertEquals(movies.size,6)
    assertThat(movies, hasItem<Movie>(hasProperty("title", Is("Pulp Fiction"))))
    assertThat(movies, not(hasItem<Movie>(hasProperty("title", Is("Black Widow")))))
}
```

U ovom testu provjeravamo da li metoda movie repozitorija `getFavoriteMovies()` vraća odgovarajući niz. Prva provjera je veličine rezultujućeg niza koja treba biti 6. Druga provjera je da niz filmova sadrži element tipa `Movie` koji ima atribut `title` sa vrijednosti *"Pulp Fiction"*, a treća provjera gleda da niz ne sadrži element tipa `Movie` sa atributom `title` i vrijednosti *"Black Widow"*.

Matcher `is` uključite na sljedeći način:

```
import org.hamcrest.CoreMatchers.`is` as Is
```

Osmislite test za metodu `getRecentMovies` repozitorija `MovieRepository` koja će na sličan način testirati rezultujući niz.

ZSR

- Napraviti da kada se klikne na naziv filma, da se pokrene youtube aplikacija, kojoj je potrebno proslijediti string naziva filma i string "trailer" po kojem će se izvršiti pretraga.
- Napraviti button na aktivnosti detalji o filmu i tom dugmetu dodjeliti akciju klika, kojom se prosljeđuje kratak tekst o filmu nekoj od aplikacija za dijeljenje podataka *(Facebook messenger, Google+ i sl.)*. Koristiti akciju tipa `Intent.ACTION_SEND.` Razlika između ovog zadatka i onog sa vježbi je što ovdje aplikacija inicira akciju, a u Zadatku 4 sa vježbi aplikacija odgovara na akciju send.
- Napraviti `BroadcastReceiver` koji će obavijestiti korisnika putem Toast-a da je uređaj odspojen sa interneta u slučaju da se izgubi konekcija.
- Testirajte da li je raspored elemenata na `MovieDetailsActivity` ispravan
- Testirajte da li klik na naslov filma otvara YouTube aplikaciju