

# Biblioteka filmów

## Projektowanie użytkowej aplikacji w Android Studio

### 1. Cel oraz opis zadania.

Instrukcja ma charakter wykonywania projektu przy jednoczesnym poznawaniu elementów związanych z tworzeniem aplikacji. Celem instrukcji jest stworzenie użytkowej aplikacji, będącej biblioteką filmów. W aplikacji będzie można wykonywać podstawowe opcje takie jak np. dodawanie filmów, usuwanie, ocenianie. Tworzenie aplikacji ma pomóc w zrozumieniu projektowania aplikacji w środowisku *AndroidStudio* w języku Java.

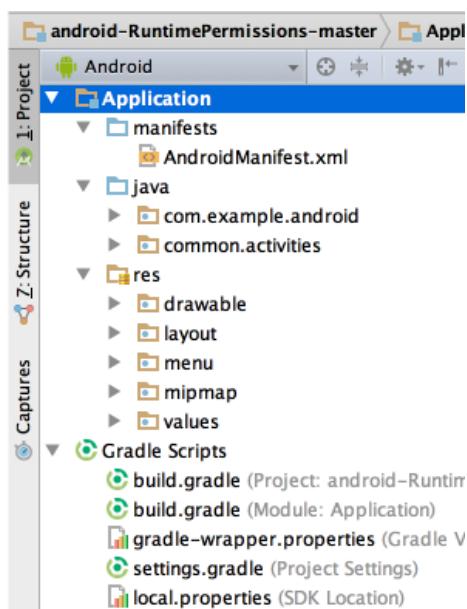
**Podczas wykonywania tej instrukcji poznasz:**

- ✓ Środowisko Android Studio oraz strukturę projektu aplikacji,
- ✓ Sposób definiowania wyglądu aplikacji - widoki, grupy widoków,
- ✓ Tworzenie layoutów oraz wykorzystywanie plików *.xml* w celu edycji wyświetlnych treści,
- ✓ Podstawowe komponenty aplikacji - aktywności, intenty, kontekst,
- ✓ Bazę danych *Google Firebase Realtime/Storage*, sposób jej konfiguracji oraz dodanie do projektu,
- ✓ Biblioteki takie jak: *RecyclerView*, *CardView*, *GridLayout*,
- ✓ Tworzenie systemu ocen w aplikacji,
- ✓ Tworzenie animacji,
- ✓ Wskazówki przydatne podczas pisania aplikacji w *AndroidStudio*.

### 2. Wstęp teoretyczny.

#### Struktura projektu aplikacji

W oknie programu Android Studio istnieje możliwość podglądu struktury projektu, ma ona postać hierarchiczną, najważniejsze z katalogów wyświetlane są na samej górze, natomiast pliki podobnych typów pogrupowane są wspólnie w folderze.



Rys.1. Przykładowa struktura projektu.

Źródło: <https://developer.android.com/studio/intro>

Każdy projekt stworzony w Android Studio musi posiadać odpowiednio opisany plik manifestu – *AndroidManifest.xml*. Plik ten przedstawia podstawowe informacje odnośnie pisanej aplikacji takie jak: minimalna wersja systemu Android wymagana do uruchomienia aplikacji, pozwolenia na np. korzystanie z Bluetooth, Internetu, czy orientację ekranów aktywności.

Katalog *Java* zawiera pliki z kodem źródłowym (logika programu), a także testy jednostkowe oraz testy automatyczne interfejsu użytkownika.

Katalog *res* zawiera zasoby aplikacji:

- **drawable** - folder zawierający grafikę/obrazy wykorzystywane w aplikacji,
- **layout** – folder przechowujący grupy widoków aplikacji zapisane za pomocą plików XML (ang. Extensible Markup Language),
- **mipmap** - obrazki, które są ikoną aplikacji,
- **anim** - zawiera animacje określone przy pomocy XML,
- **values** – tutaj przechowywane są inne zasoby aplikacji, jak na przykład ciągi znaków, kolory. Zapisane są one za pomocą plików XML,
- **menu** - zawiera listy menu aplikacji.

W strukturze projektu możemy także odnaleźć plik ***build.gradle***:

*Build.gradle(projektu)* - Plik komplikacji najwyższego poziomu, w którym można dodać opcje konfiguracji wspólne dla wszystkich podprojektów/modułów.

*Build.gradle (modułu)* - znajdujący się w każdym katalogu, umożliwia skonfigurowanie ustawień komplikacji dla konkretnego modułu, w którym się znajduje.

W plikach *gradle* określa się między innymi wersję narzędzi do komplikacji zestawu SDK.

### Sposób definiowania wyglądu aplikacji

Widoki: <https://developer.android.com/reference/android/view/View>

Layouty: <https://developer.android.com/guide/topics/ui/declaring-layout>

W Android Studio wygląd aplikacji definiuje się za pomocą widoków oraz grup widoków.

Widoki są to obiekty widoczne przez użytkownika podczas korzystania z aplikacji. Używa się ich do reprezentacji zawartości na ekranie. Do widoków można zaliczyć elementy takie jak:

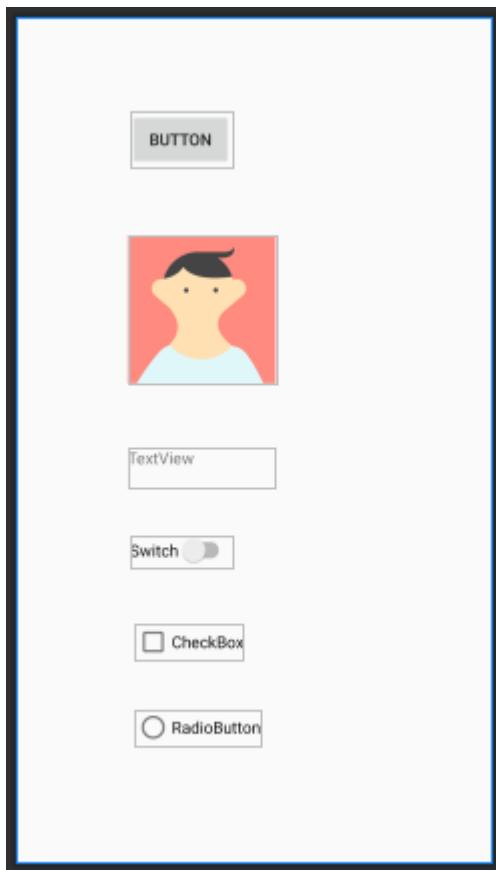
- Przyciski (*Button*),
- Pola obrazkowe (*ImageView*),
- Pola tekstowe (*TextView*, *EditView* itp.)
- Widgety.

Widoki grupuje się w grupy widoków (layouty). Są to pewnego rodzaju kontenery na elementy takie jak przyciski, pola tekstowe itp. Wyróżnia się kilka typów grup widoków:

- **Linear Layout** – jest to grupa widoków, w których widoki rozmieszczone są jeden pod drugim pionowo lub poziomo w zależności od wybrania orientacji. Istnieją wagi określające wielkość widoków.
- **Relative layout** – umożliwia tworzenie rozbudowanych hierarchii widoków, istnieje możliwość ustalenia poszczególnych widoków w relacjach do innych. Zamiast wykorzystywać wiele linear layoutów w projekcie, można wykorzystać jeden relative layout.

- **Constraint layout** – polega na zagnieżdzaniu widoków w relacjach do siebie. Konieczne jest zakotwiczenie widoku co najmniej po jednym zakotwiczeniu w pionie i poziomie.
- **FrameLayout** - powinien być używany do przechowywania pojedynczego widoku podziemnego.

Widoki oraz grupy widoków definiuje się w formacie XML. Przykładowe widoki przedstawione zostały na rysunku poniżej:



Rys. 2. Przykładowe widoki w Android Studio.

### Podstawowe komponenty aplikacji

Komponenty aplikacji są to podstawowe elementy składowe aplikacji na Androide, poniżej zostały wymienione niektóre z nich (więcej komponentów oraz informacji o nich na stronie:

<https://developer.android.com/guide/components/fundamentals>

#### Aktywności

Aktywności to podstawowy budulec do tworzenia zaplanowanej funkcjonalności aplikacji mobilnej i zawierający ich implementację. Oznacza to, że wywoływanie funkcji aplikacji mobilnej odbywa się przez uruchomienie Aktywności. W danej aktywności można wywoływać kolejne aktywności, dzięki czemu uzyskujemy możliwość realizacji nawet bardzo złożonych scenariuszy. Podstawową właściwością aktywności jest jej krótki czas wykonania oraz kontakt z użytkownikiem aplikacji lub innymi aplikacjami. Aktywność to element umożliwiający interakcję z użytkownikiem, reprezentuje pojedynczy ekran z interfejsem użytkownika. Na urządzeniu może być uruchomionych kilka aplikacji, jednak w danym momencie użytkownikowi udostępniany jest interfejs tylko jednej aplikacji (aktywności). System tworzy stos wywołań aktywności – uruchomiona jest tylko ta aktywność która znajduje się na wierzchu stosu i tylko jej interfejs jest dostępny dla użytkownika.

## Kontekst

Kontekst zapewnia dostęp do informacji o otoczeniu (kontekście) w jakim jest wykonywana aplikacja. Utworzony obiekt *Context* zapewnia między innymi dostęp do zasobów aplikacji (np. za pomocą metody *getResources()* ).

*Context* jest obiektem (klasy abstrakcyjnej *Context*), który przechowuje wszystkie istotne dane związane z aktualnym stanem naszej aplikacji lub danego obiektu (np. aktywności). Pozwala on m.in na rozeznanie się w sytuacji nowo utworzonym obiektom, które będą potrzebowaly określonych informacji związanych z funkcjonowaniem aplikacji. Dzięki niemu możemy też wykonywać operacje o charakterze „globalnym” (komunikacja pomiędzy poszczególnymi komponentami aplikacji), takie jak np. wysyłanie intentów, uruchamianie nowych aktywności, dostęp do zasobów aplikacji (*resources*) lub lokalnych baz danych.

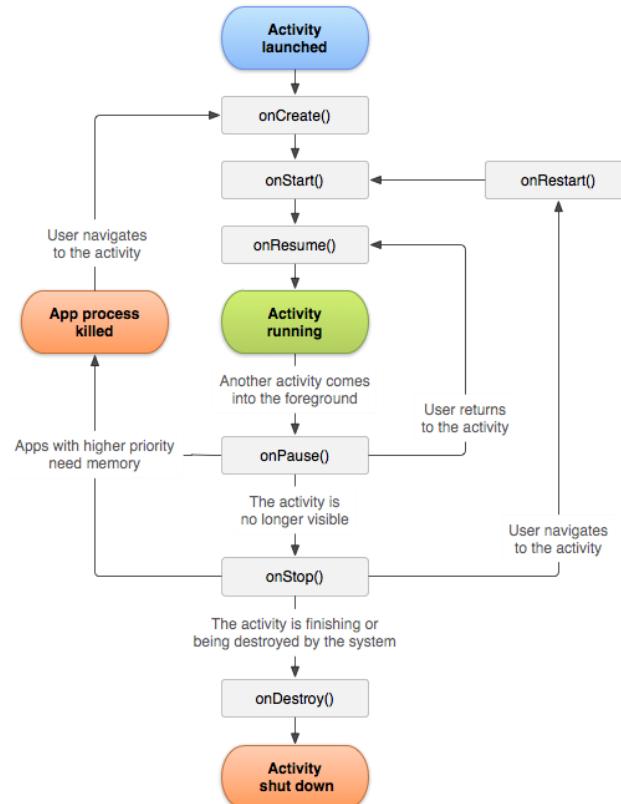
## Intenty

Intenty stanowią mechanizm obsługi zdarzeń występujących w aplikacji mobilnej. Z wykorzystaniem mechanizmu intencji możliwe jest uruchomienie aktywności lub serwisu w momencie wystąpienia określonego zdarzenia. Mechanizm intencji służy także do komunikacji między różnymi aplikacjami uruchomionymi na urządzeniu mobilnym oraz do obsługi zdarzeń z innych aplikacji (np. połączenie przychodzące) i wyposażenia urządzenia (np. niski stan baterii, wykonano zdjęcie). Intenty można podzielić na dwie podstawowe grupy: jawnie i niejawne.

- **wywołanie jawnie** – podawana jawnie aktywność do której zamierza się przejść,
- **wywołanie niejawne** – polega na określeniu jaka akcja ma zostać wykonana i z jakimi danymi. O tym jaka aktywność najlepiej pasuje decyduje system operacyjny. Związane z tym są filtry intencji (definiują możliwe do wykonania akcje). W przypadku niejawnego wykonania, gdy wiele aplikacji pasuje do zadanej akcji wyświetlane jest okno dialogowe z wyborem aplikacji obsługującej.

**Każdy typ służy innemu celowi i ma inny cykl życia, który definiuje sposób tworzenia i niszczenia komponentu.**

Istotnym jest poznanie cyklu życia aktywności, został on przedstawiony na rysunku poniżej:



Rys. 3. Cykl życia aktywności.

Źródło: <https://developer.android.com/guide/components/activities/activity-lifecycle>

Znając cykl życia aktywności możliwe jest efektywne zarządzanie zasobami w naszej aplikacji. W momencie, gdy użytkownik porusza się po aplikacji lub ją wyłącza i włącza ponownie, instancje aktywności przechodzą przez różne

stany w ich cyklu życia. Dobre wykorzystanie wywołań zwrotnych (metody wywoływanie podczas przechodzenia pomiędzy stanami aktywności) pozwala na uniknięcie niepożądanych zjawisk w aplikacji takich jak np. :

- Utrata postępów w aplikacji podczas rotacji ekranu,
- Wystąpienie błędu w momencie odebrania połączenia przez użytkownika podczas korzystania z aplikacji,
- Utrata postępów w momencie wyjścia z aplikacji.

Krótkie wyjaśnienie metod związanych z cyklem życia aktywności:

**Metoda onCreate()** – Konieczna jej implementacja. To odwołanie jest uruchamiane, gdy system tworzy aktywność.

Tutaj należy zainicjować wstępne parametry zmiennych, warunki początkowe działania aplikacji.

**Metoda onStart()** – Wywołanie uruchamiane w trakcie przygotowań aplikacji – powoduje wkroczenie aktywności na pierwszy plan i spowodowanie, żeby stała się interaktywna. Jej czas działania jest bardzo krótki. Po zakończeniu stanu określonego przez te wywołanie, aktywność przechodzi w stan *recovered* i wywoływana jest metoda *onResume()*. Jest to stan, w którym aplikacja współdziała z użytkownikiem. Stan ten trwa do tej pory, aż jakieś działanie nie odwróci jego uwagi (np. połączenie przychodzące).

**Metoda onPause()** – Metoda ta jest wywoływana jako pierwsza, w sytuacji gdy użytkownik opuszcza aktywność.

Wskazuje, to iż czynność nie jest już na pierwszym planie. Przykładowe czynniki, dla których aktywność może przejść w ten stan to:

- Zmiana aktywnego okna w urządzeniu mobilnym,
- Pojawienie się okna dialogowego.

W tej metodzie można zwolnić zasoby systemowe, aby zmniejszyć zużycie baterii.

**Metoda onStop()** – Wywoływana, gdy aktywność nie jest już widoczna dla użytkownika. W tej sytuacji nowo uruchomione zdarzenie obejmuje cały ekran. Występuje również, gdy aktualne działanie zostaje zakończone.

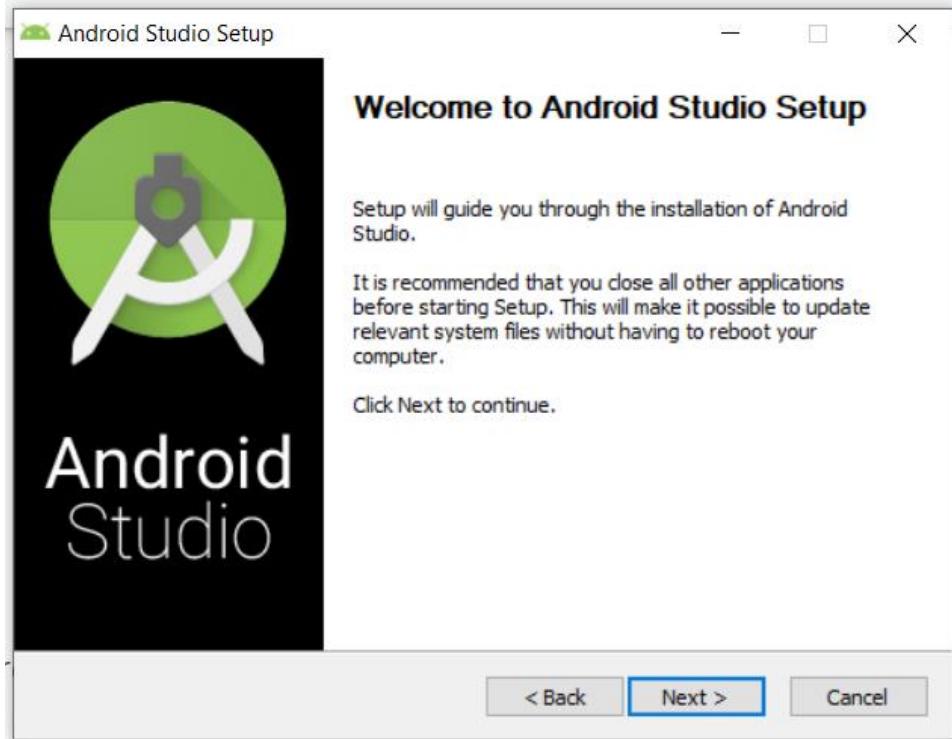
W tej metodzie powinno się zwolnić zasoby systemowe, o ile nie zostało to wykonane w wywołaniu *OnPause()*.

**Metoda onDestroy()** - Wywołana przed zniszczeniem aktywności.

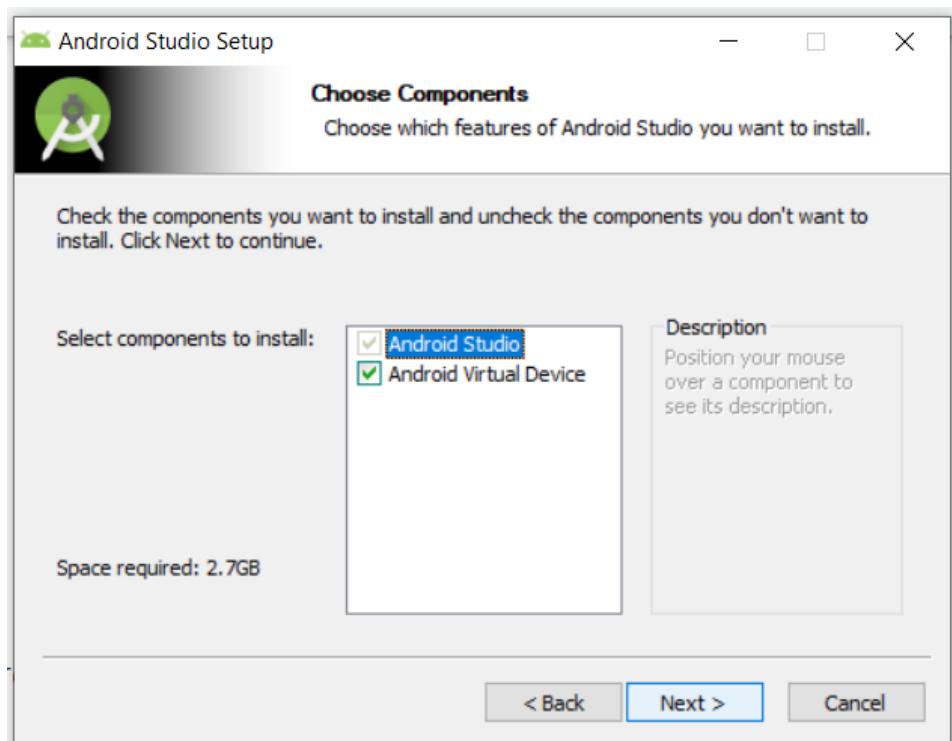
### 3. Zadania do wykonania.

#### 3.1. Instalacja i wstępna konfiguracja środowiska Android Studio.

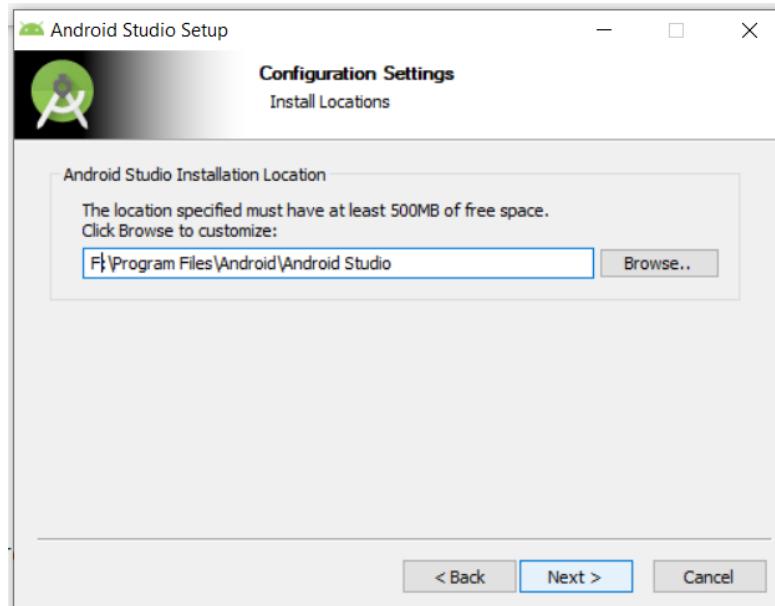
Pierwszym krokiem jest pobranie aktualnej wersji programu Android Studio (program można znaleźć na stronie: <https://developer.android.com/studio>). Po pomyślnym pobraniu programu należy przystąpić do instalacji, jej przebieg został przedstawiony na zrzutach poniżej:



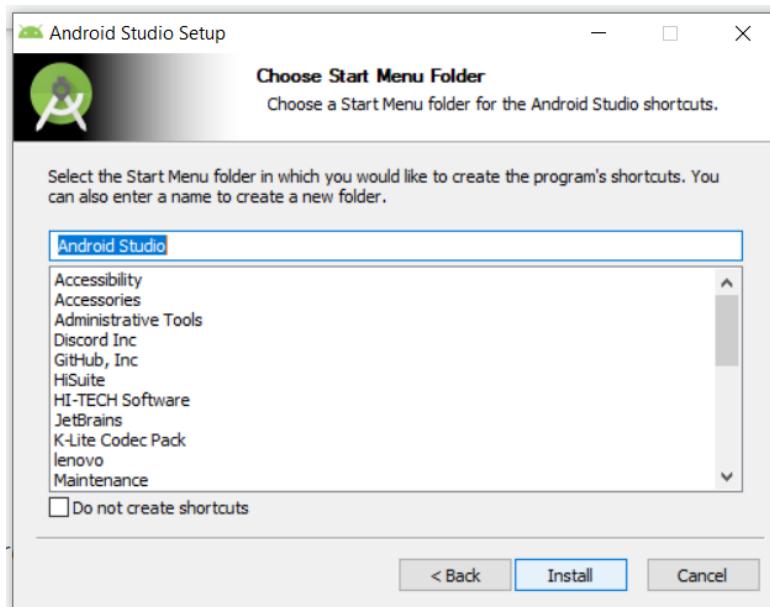
Rys. 4. Instalacja programu Android Studio.



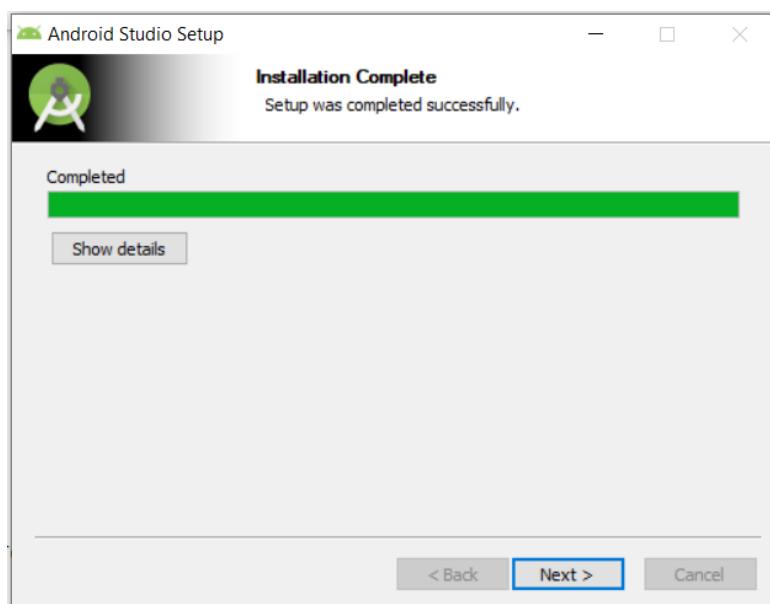
Rys. 5. Instalacja programu Android Studio.



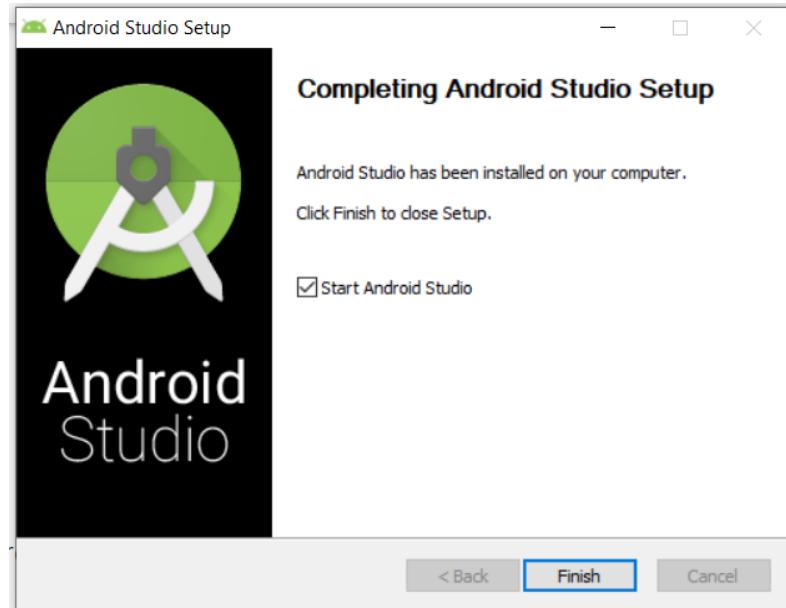
Rys. 6. Instalacja programu Android Studio.



Rys. 7. Instalacja programu Android Studio.

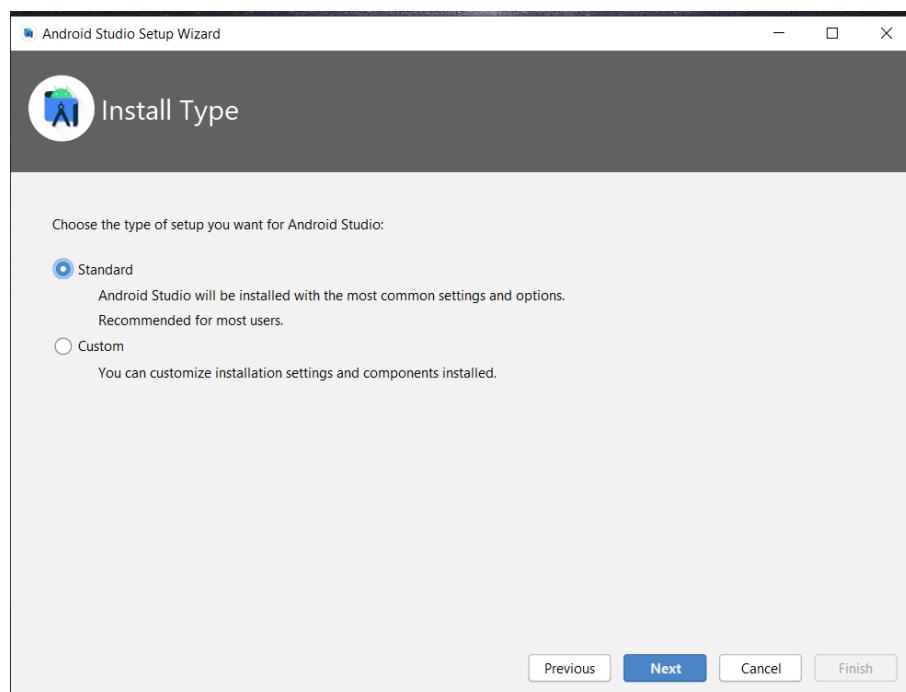


Rys. 8. Instalacja programu Android Studio.



Rys. 9. Instalacja programu Android Studio.

Po włączeniu zainstalowanego programu należy przejść przez okna konfiguracyjne. Do celów poradnika, zaleca się zastosowanie standardowych/rekomendowanych ustawień (rys. 10.).



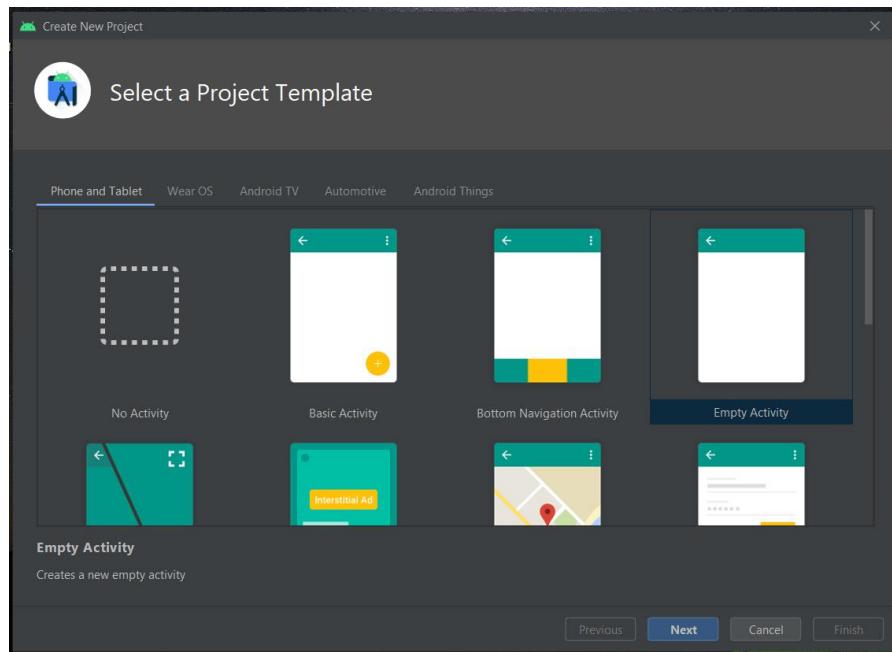
Rys. 10. Wybranie ustawień Android Studio.

Po wybraniu ustawień programu należy pobrać wymagane SDK zasugerowane przez instalator.

### 3.2.Tworzenie projektu.

Aby stworzyć nowy projekt, należy uruchomić program Android Studio oraz wybrać te opcje z ekranu startowego.

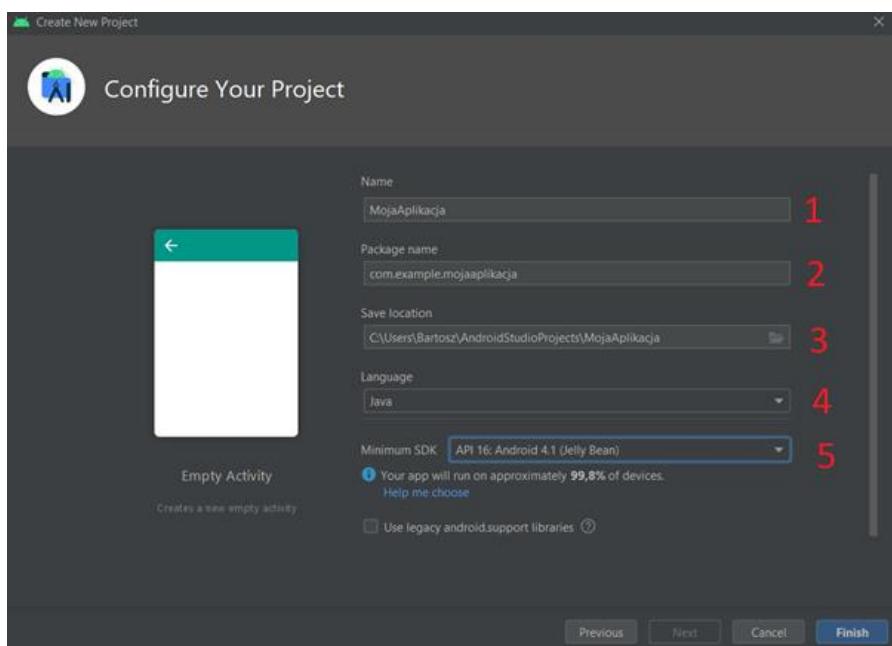
W kolejnym oknie, można wybrać szablon projektu, do celów tworzonej aplikacji należy wybrać pustą aktywność oraz kliknąć przycisk *next* (Rys.11).



Rys. 11. Wybranie szablonu projektu.

W kolejnym oknie przedstawionym na rysunku poniżej, mamy do dyspozycji kilka ustawień, odpowiednio:

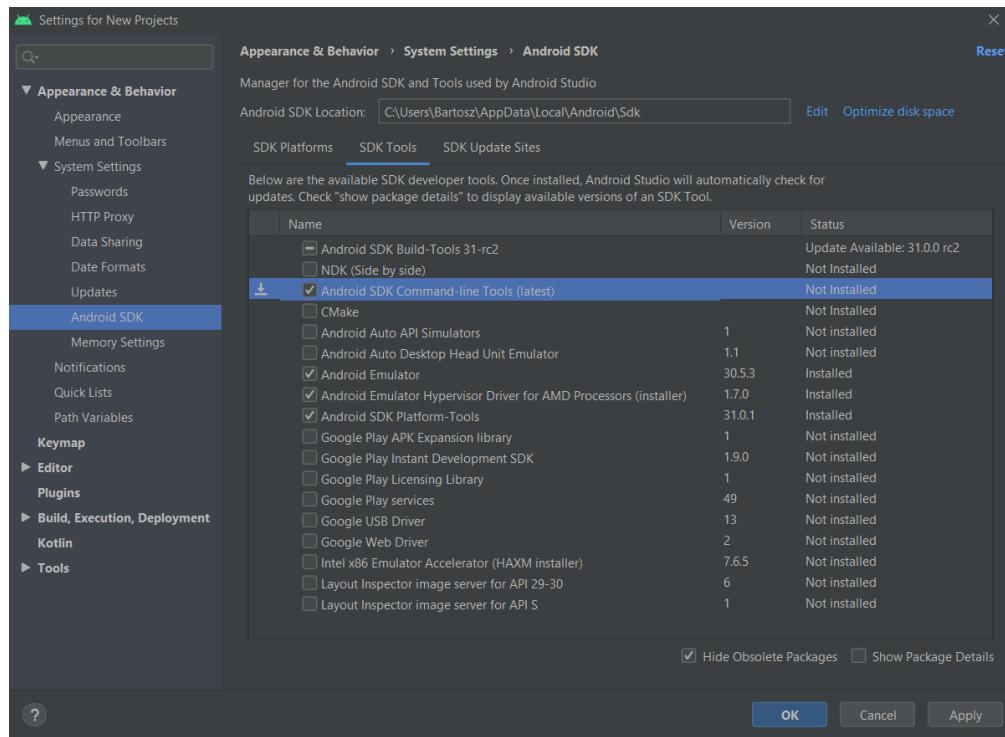
- 1 - Nazwa naszej aplikacji
- 2 – Nazwa paczki (zupełniane automatycznie)
- 3 – Lokalizacja projektu
- 4 – Język, w którym będzie pisana aplikacja (w tym skrypcie korzystamy z języka Java)
- 5 – Jest to wybór minimalnej wersji Androida, która będzie kompatybilna z naszą aplikacją



Rys. 12. Konfiguracja projektu.

Ważne jest, aby upewnić się że mamy zainstalowane *Android SDK Command-line Tools* (Rys.13).

Aby znaleźć menadżer SDK należy wejść w *tools -> SDK Manager* z poziomu opcji w górnej części programu.

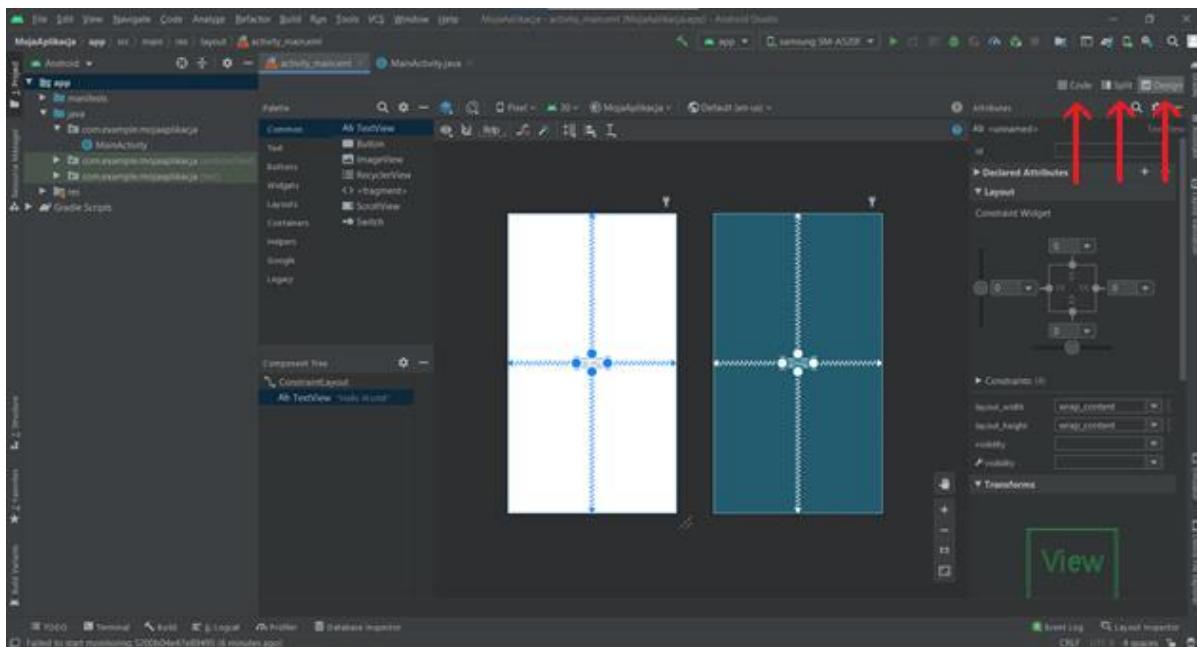


Rys. 13. Menadżer SDK.

### 3.2.1. Pierwszy layout.

Tworzenie layoutów aplikacji odbywać się może za pomocą trzech trybów:

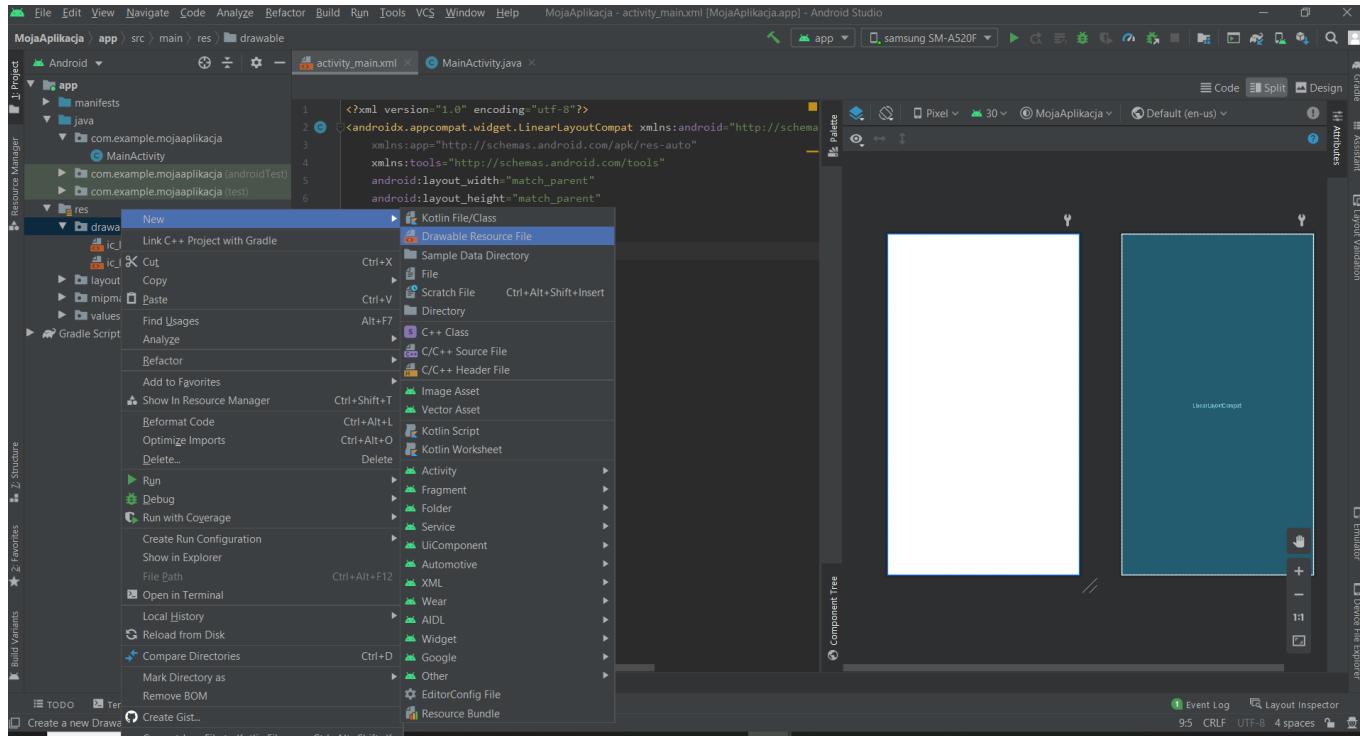
- Tworzenie poprzez używanie kodu w XML,
- Tworzenie poprzez graficzny interfejs (wstawianie elementów, ustawianie ich atrybutów),
- Tworzenie korzystając z trybu dzielonego.



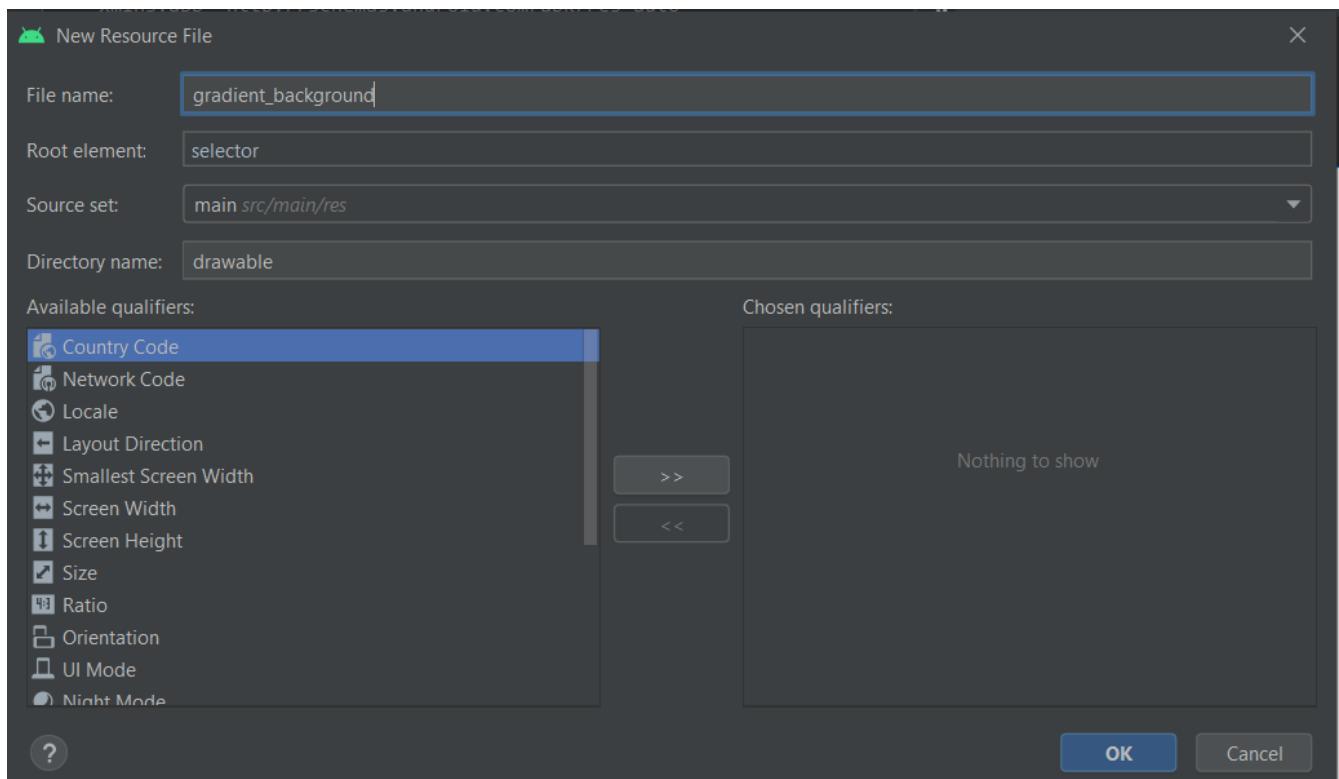
Rys. 14. Widok edycji Layoutu, tryby edycji oznaczone strzałkami.

Pierwszym zadaniem do wykonania jest usunięcie znajdującego się pola tekstuowego (*TextView*) z naszego Layoutu, aby otrzymać pusty layout. W tym celu można usunąć to pole bezpośrednio z kodu XML, lub wykorzystać edytor graficzny (zakładka *design*) i kliknąć na element po czym usunąć wciskając klawisz *delete* na klawiaturze.

Kolejnym zadaniem jest stworzenie gradientowego tła w layout. W tym celu należy utworzyć nowy plik *Drawable* z rozszerzeniem XML. Tworzenie pliku przedstawione zostało na rysunkach poniżej:

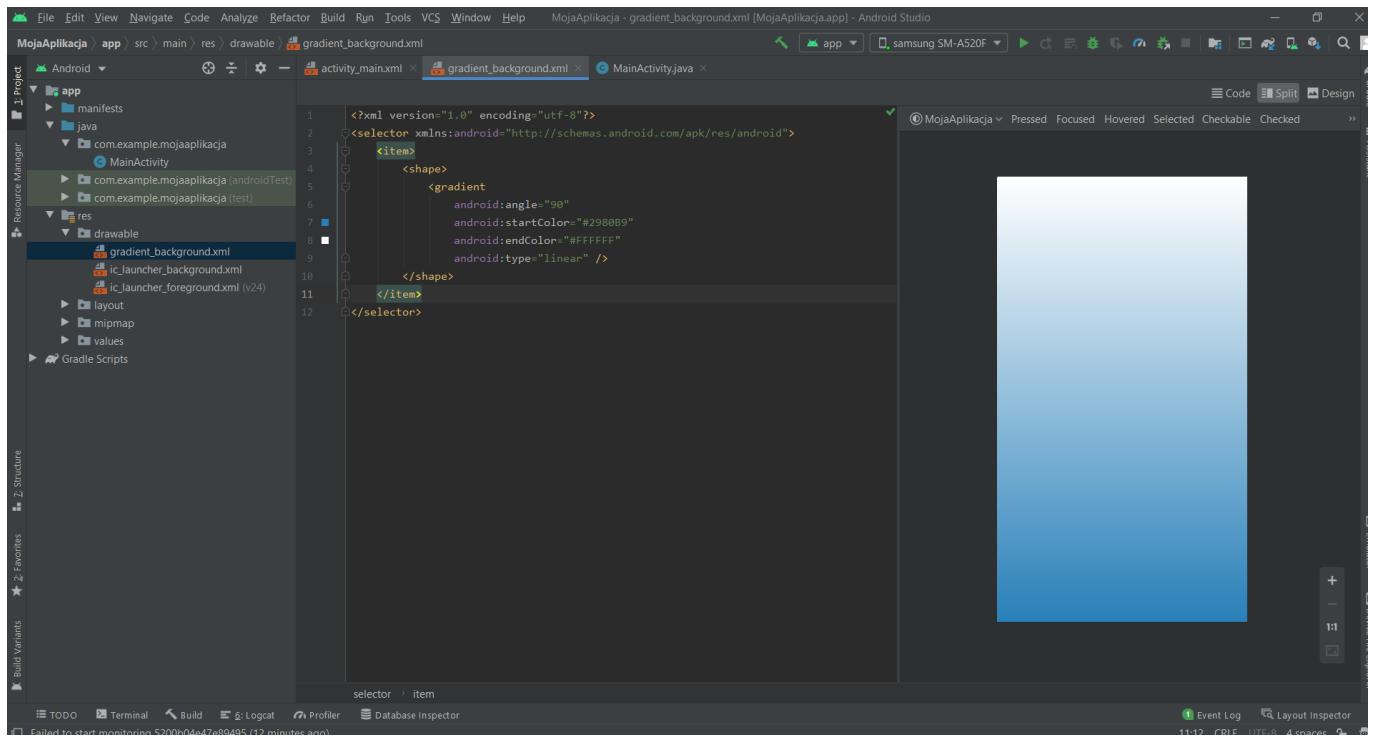


Rys. 15. Tworzenie pliku zawierającego definicję gradientowego tła layoutu.



Rys. 16. Tworzenie pliku zawierającego definicję gradientowego tła layoutu.

W utworzonym przez nas pliku, należy stworzyć element (*item*) i zdefiniować jego kształt (*shape*), w którym określmy gradient. Atrybuty jakie zastosujemy do naszego gradientu to kąt (*angle*), który ustawiony został na 90 stopni. Spowoduje to, że zmiana koloru odbędzie się w pionie. Dodatkowo należy wybrać kolor początku oraz końca i typ gradientu. W tworzeniu kolorystyk gradientów można posłużyć się stroną: <https://uigradients.com/>, która pomaga dobrą odpowiednie kolory w tworzonym gradiencie. Zmodyfikowany plik powinien wyglądać w sposób jak to zostało przedstawione na rysunku 17:



Rys. 17. Wygląd pliku *gradient\_background.xml*.

Kolejno zajmować się będziemy na razie pustym layoutem. W tym celu należy przejść do katalogu *layout* i wybrać plik *activity\_main.xml*. Jest to nasz główny layout, wyświetlany w pierwszej aktywności (aktywność główna).

Pierwszą rzeczą, którą należy zrobić w tym pliku jest zmiana jego layoutu na *linear layout* (np. poprzez tryb tekstowy, w znaczniku określającym typ layoutu zmienić go na *linear layout*). Wybrano ten layout, ponieważ na tym ekranie poszczególne widoki mają być umieszczone jeden pod drugim. Dlatego dodatkowym parametrem jaki należy zastosować dla layoutu jest *android:orientation="vertical"*. W android Studio możliwe jest autouzupełnianie wprowadzanej przez nas składni, wystarczy wpisać część atrybutu, który chcemy zastosować a następnie wybrać z proponowanych opcji, tę która nas interesuje i kliknąć klawisz *tab*.

Ważnymi parametrami naszego layoutu jest wysokość oraz szerokość. Istnieje kilka sposobów definiowania tych atrybutów:

- Zastosowanie opcji *match\_parent*: dopasuje ona szerokość/wysokość do rodzica, czyli do widoku lub grupy widoków, w którym znajduje się nasz widok;
- Zastosowanie opcji *wrap\_content*: dopasuje ona szerokość/wysokość na tyle, aby obejmowana była tylko zawartość widoku. Np. Tworząc przycisk z tekstem „save”, dopasuje on wymiary przycisku do tekstu znajdującego się wewnątrz niego;
- Zastosowanie jednostek: wymiary widoków lub grup widoków można również definiować za pomocą konkretnych wymiarów z odpowiednimi jednostkami. Do takich jednostek należy np. *pixel* (px), jednakże nie zaleca się stosowania tej jednostki ze względu na to, że różne ekranы mają różne gęstości pikseli. Jedną z najczęściej wykorzystywanych jednostek jest *dp*, czyli piksele niezależne od gęstości. Jest to jednostka, która w przybliżeniu jest równa jednemu pikselowi na ekranie, którego średnia gęstość pikseli wynosi 160 dpi.

Kolejnym dodanym atrybutem jest *android:weighSum*, który pozwala na określanie wag poszczególnych widoków w grupie (więcej informacji na: <https://developer.android.com/reference/android/widget/Layout>).

Następnie do naszego layoutu należy dodać widoki takie jak *ImageView*, *TextView* oraz *Button*. Dwa ostatnie umieścić należy w drugiej grupie, tym razem *RelativeLayout*, aby można było je rozmielić w relacjach do siebie (*należy pamiętać o połączeniach/constraints!* W przeciwnym wypadku widok wyląduje w lewym, górnym rogu rodzica).

Aby użyć obraz w naszym *ImageView* należy skorzystać z atrybutu *app:srcCompat* oraz podać odpowiednią ścieżkę do niego. W tym celu należy skopiować plik *main\_activity\_image.png* do folderu *drawable* (poprzez wklejenie go bezpośrednio do folderu, w którym się zawiera lub po skopiowaniu pliku, kliknięcie na katalog w strukturze projektu i wykorzystaniu opcji *ppm->paste*). Każdy widok powinien posiadać swoje unikalne *ID*, do którego można się odnosić w plikach z rozszerzeniem *.java* zawierających logikę aktywności.

Inne wykorzystane atrybuty w kodzie:

- *android:layout\_weight*: pozwala na ustawienie wagi widoku,
- *android:layout\_alignParent(Top/End/Right)*: jest to wyrównanie do rodzica do podanych krawędzi,
- *android:layout\_margin(Top/End/Right)*: ustawienie marginesów w odniesieniu do rodzica,
- *android:fontFamily*: typ czcionki,
- *android:text*: wyświetlany tekst,
- *android:textSize*: rozmiar tekstu (zaleca się stosowanie jednostek *sp*),
- *android:textStyle*: styl tekstu (np. pogrubiony, kursywa itp.).

Atrybuty wstawionych widoków, ich wartości oraz cały kod przedstawione zostały poniżej:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.appcompat.widget.LinearLayoutCompat
    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:background="@drawable/gradient_background"
    android:orientation="vertical"
    android:weightSum="10"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/movieImageViewId"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_marginTop="100dp"
        android:layout_weight="6"
        app:srcCompat="@drawable/main_activity_image" />

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="4">

        <TextView
            android:id="@+id/checkMyMovieTextViewId"
            android:layout_width="386dp"
            android:layout_height="wrap_content"
            android:layout_alignParentTop="true"
            android:layout_alignParentEnd="true"
            android:layout_alignParentRight="true"
            android:layout_marginTop="30dp"
            android:fontFamily="casual"
            android:text="Check my movie App!"
            android:textSize="36sp" />

        <Button
            android:id="@+id/clickHereButtonId"
            android:layout_width="wrap_content"
```

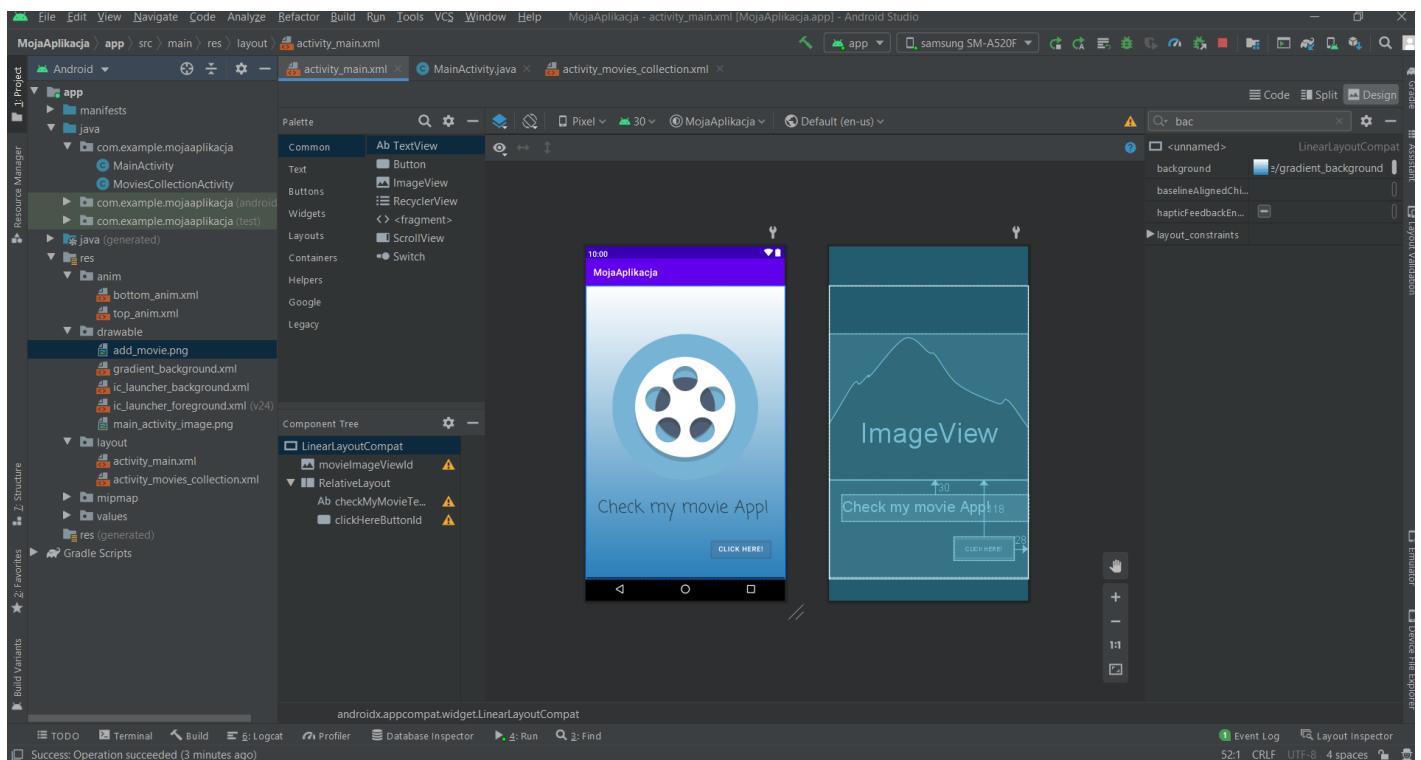
```

        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_alignParentEnd="true"
        android:layout_alignParentRight="true"
        android:layout_marginTop="118dp"
        android:layout_marginEnd="28dp"
        android:layout_marginRight="28dp"
        android:text="Click here!"
        app:backgroundTint="#3959E7C"/>
    
```

*Wskazówka: chcąc uporządkować kod w AndroidStudio nie trzeba robić tego ręcznie. Kod zostaje automatycznie uporządkowany po użyciu kombinacji klawiszy **ctrl+alt+l**.*

Widoczne żółte zaznaczenia na rysunkach powyżej oznaczają ostrzeżenia wydawane przez Android Studio – czasami nie są one istotne, jednakże mogą okazać się przydatne. Dla przykładu żółte pole na atrybutie *android:text* oznajmia, iż tekst powinien zostać wyeksportowany do zasobów określonych w pliku *strings.xml*. Dobrą praktyką jest eksportowanie ciągów znaków do tego pliku, ponieważ w momencie, gdy w naszej aplikacji zastosujemy kilka wersji językowych, znacznie łatwiej jest operować na *stringach* z pliku *strings.xml*, jednakże w naszej aplikacji nie jest to konieczne.

Stworzony layout powinien wyglądać w taki sposób jak zostało to przedstawione na rys.18:



Rys. 18. Utworzony layout *activity\_main.xml*.

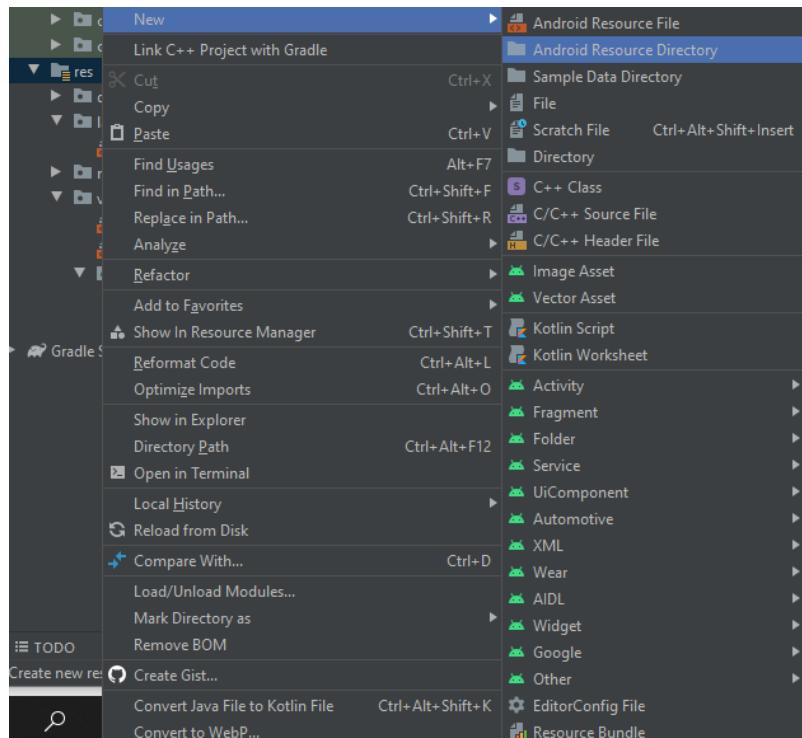
W tworzonej aplikacji nie potrzebujemy paska u góry ekranu (*ActionBar*) dlatego zostanie on usunięty, w tym celu należy otworzyć plik *themes.xml* i *themes.xml (night)* znajdujący się w *app->res->themes* i zmienić linijkę jak przedstawiono poniżej:

```
<style name="Theme.MojaAplikacja" parent="Theme.MaterialComponents.DayNight.NoActionBar">
```

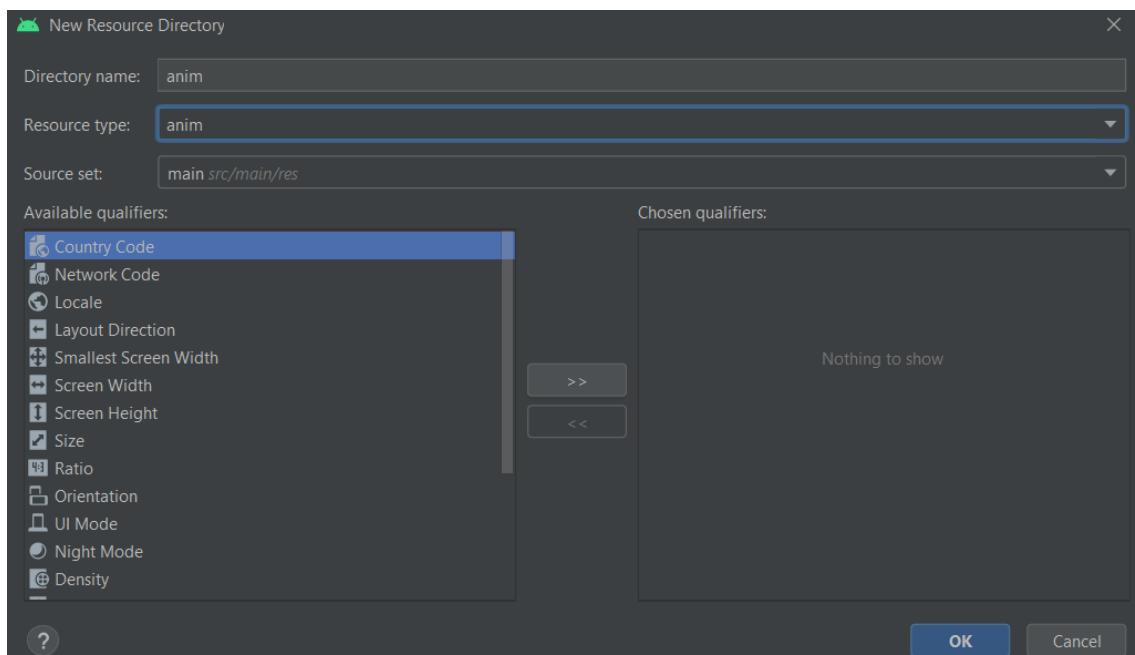
Rys. 19. Usunięcie *ActionBar*.

### 3.2.2. Tworzenie animacji.

Kolejnym etapem jest stworzenie animacji na ekranie startowym w aplikacji. W tym celu należy utworzyć katalog (*Android Resource Directory*) animacji jak przedstawiono na rys.20. i rys.21.

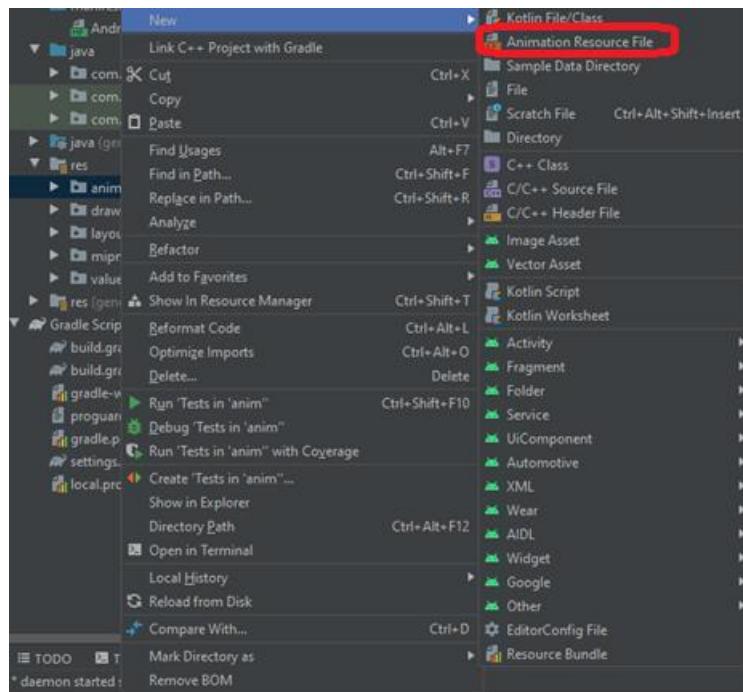


Rys. 20. Tworzenie katalogu do animacji.

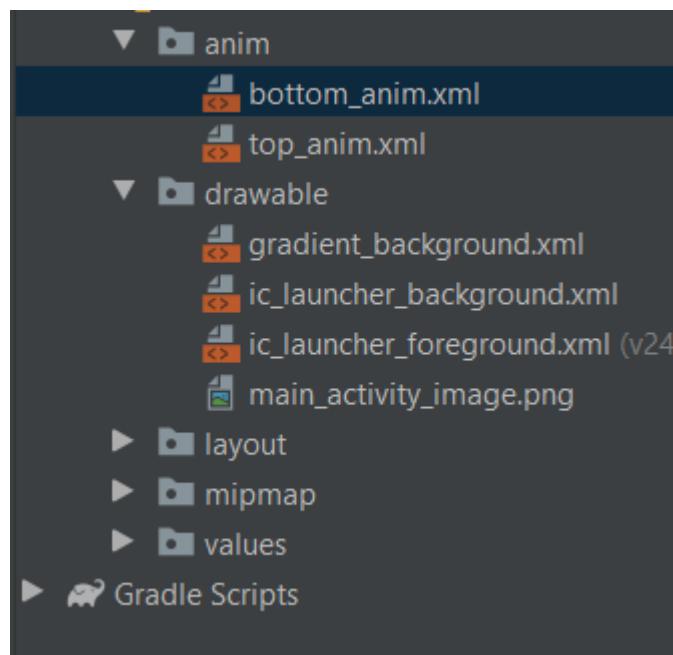


Rys. 21. Tworzenie katalogu do animacji.

Oraz utworzenie dwóch plików (*bottom\_anim.xml* i *top\_anim.xml*) związanych z animacjami:



Rys. 22. Tworzenie pliku animacji.



Rys. 23. Utworzono pliki animacji.

Będą to animacje odpowiadające za górną animację *ImageView* oraz dolną animację *TextView*.

W celu ustawienia parametrów animacji, zajmiemy się najpierw animacją górną - *top\_anim.xml* (rys.23).

Zamysłem jest, aby wykonać animację obiektu *ImageView*, która polegałaby na tym, że nasz obraz pojawił się od góry do środka ekranu. Dodatkowo animacja obrazu miałaby wykorzystywać efekt zmiany przezroczystości (od przezroczystego do braku przezroczystości). W tym celu pomiędzy znacznikami *<set>* *</set>* należy umieścić odpowiednie atrybuty oraz ich parametry. Za sterowanie ruchem obiektu, odpowiada atrybut *translate*. W naszym przykładzie ustawiliśmy aby obiekt poruszał się w kierunku pionowym – od pozycji -50% wysokości obiektu do jego naturalnej pozycji. Czas animacji ustowany został na 2500 ms w celu zachowania płynności animacji. Kolejnym wykorzystanym atrybutem jest *alpha*. Parametry tego atrybutu określają poziom alfa obiektu, tj. zmianę poziomu

przezroczystości. Jest to przydatny atrybut do celów wtapiania i wygaszania obiektów na ekranie. Parametr 1.0 określa zerową przezroczystość. W przykładzie wykorzystano zmianę od 10% widoczności obiektu do 100% w takim samym czasie jaki został określony w atrybucie *translate*. Plik powinien wyglądać w sposób jaki zostało to przedstawione poniżej:

```
<?xml version="1.0" encoding="utf-8"?>

<!-- top image animation of first screen -->

<set xmlns:android="http://schemas.android.com/apk/res/android">
    <translate
        android:fromXDelta="0%"
        android:fromYDelta="-50%"
        android:duration="2500" />
    <alpha
        android:fromAlpha="0.1"
        android:toAlpha="1.0"
        android:duration="2500" />
</set>
```

W przypadku dolnej animacji, ustawiony zostanie tylko efekt „wtapiania” za pomocą parametrów atrybutu *alpha*:

```
<?xml version="1.0" encoding="utf-8"?>

<!-- bottom image animation of first screen -->

<set xmlns:android="http://schemas.android.com/apk/res/android">
    <alpha
        android:fromAlpha="0.01"
        android:toAlpha="1.0"
        android:duration="3000" />
</set>
```

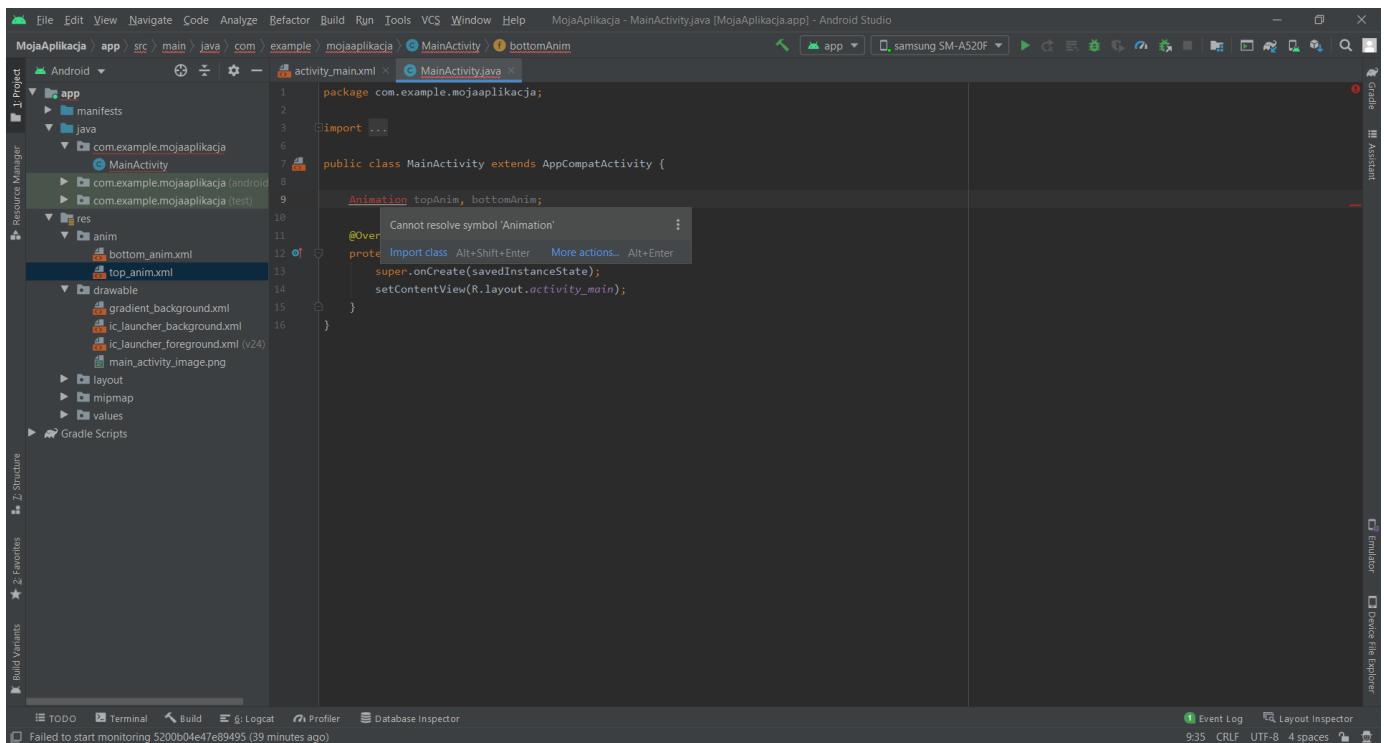
### 3.2.3. Tworzenie logiki aktywności.

W tym miejscu zajmiemy się tworzeniem funkcjonalności tworzonej aktywności. Aktywność tak jak zostało to opisane na początku skryptu jest w uproszczeniu „pojedynczym ekranem”, który zawiera interfejs graficzny i zaimplementowaną logikę działań. Na początku zajmiemy się główną aktywnością *MainActivity.java*. W tym celu należy przejść do katalogu przedstawionego na rys.24, oraz otworzyć plik związany z główną aktywnością.

Początkowo utworzymy obiekt klasy *Animation* w głównej klasie *MainActivity*, określający utworzone animację (rys.24). Podkreślenie *Animation* wynika z braku zimportowanej klasy związanej z animacjami. W AndroidStudio w momencie, gdy pojawi się błąd w syntax’ie możemy zastosować dwa skróty klawiszowe:

- *alt+shift+enter* – podejmuje próbę naprawienia błędu poprzez rekommendowaną zmianę,
- *alt+enter* – wyświetla możliwe do wykonania akcje, w celu naprawy błędu.

W naszym przypadku wystarczy zaimportować klasę *Animation*. Po poprawnym importie w miejscu *import* powinny pojawić się odpowiednie klasy ( w naszym przypadku: *import android.view.animation.Animation;*).



Rys. 24. Tworzenie logiki aktywności – animacje.

Kolejno pod nowo stworzonymi obiektami należy dodać kolejne obiekty reprezentujące nasze widoki w layout:

```
ImageView movieImage;
TextView checkMyMovieTextView;
```

W tym miejscu przechodzimy do metody `onCreate()`, która została opisana na początku instrukcji. Jest ona wywoływana w momencie „tworzenia” aktywności. Zauważalne jest iż, w tej metodzie wczytywany jest nasz plik layoutu. Dzieje się to za pomocą: `setContentView(R.layout.activity_main);`

W celu uruchomienia animacji wykorzystamy następujący syntax:

```
topAnim = AnimationUtils.LoadAnimation(this,R.anim.top_anim);
bottomAnim = AnimationUtils.LoadAnimation(this,R.anim.bottom_anim);
```

- *AnimationUtils* - Definiuje typowe narzędzia do pracy z animacjami (<https://developer.android.com/reference/android/view/animation/AnimationUtils>)
- *LoadAnimation* – ładuje animację zasobu określonego w nawiasie. *This* odnosi się do wykorzystywanego kontekstu, natomiast zasób określony jest za pomocą *R.anim.top\_anim*. „R” w tym przypadku jest to klasa – stanowi skrót od *Resources*. Używamy jej, gdy chcemy odwołać się do konkretnych zasobów w naszym projekcie. *Top\_anim* oraz *bottom\_anim* oznaczają nazwy naszych plików ze zdefiniowanymi animacjami.

Jednakże taka składnia nie uruchomi jeszcze naszej animacji. Konieczne jest przypisanie jej do konkretnych widoków. W tym celu wykonujemy podobny proces jak wyżej, przypisujemy do naszych obiektów widoki zadeklarowane w naszym layout:

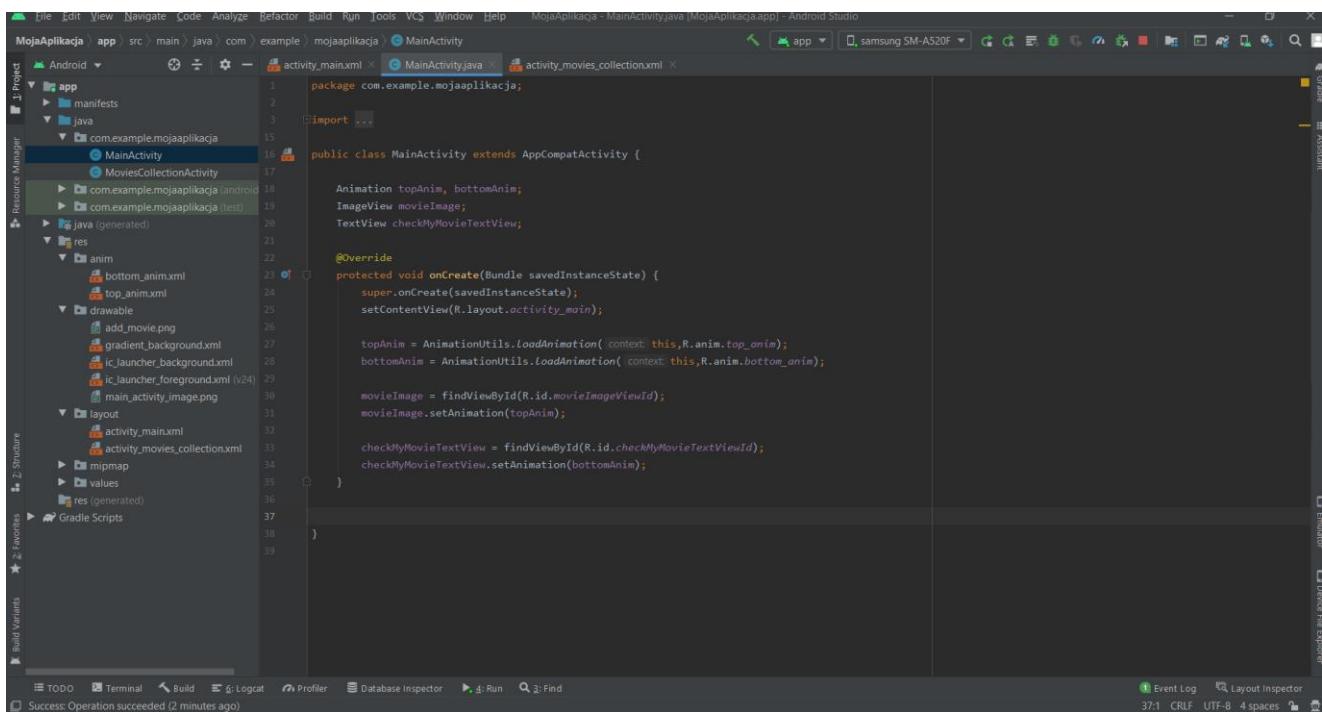
```
movieImage = findViewById(R.id.movieImageViewId);
checkMyMovieTextView = findViewById(R.id.checkMyMovieTextViewId);
```

Odnajdywanie widoków odbywa się w tym przypadku po *id*, które zostało ustanowione w pliku *activity\_main.xml*.

Ustawienie animacji odbywa się sposób przedstawiony poniżej:

```
movieImage.setAnimation(topAnim);
checkMyMovieTextView.setAnimation(bottomAnim);
```

Cały plik powinien na te chwilę wyglądać w następujący sposób:



The screenshot shows the Android Studio interface with the project 'MojaAplikacja' open. The code editor displays the MainActivity.java file, which contains Java code for an AppCompatActivity. The code includes imports, class definition, onCreate method, and various UI component initializations. The project structure on the left shows files like activity\_main.xml, activity\_movies\_collection.xml, and various resource XML files. The bottom status bar indicates a successful build.

```
package com.example.mojaaplikacja;
import ...;
public class MainActivity extends AppCompatActivity {
    Animation topAnim, bottomAnim;
    ImageView movieImage;
    TextView checkMyMovieTextView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        topAnim = AnimationUtils.loadAnimation(this, R.anim.top_anim);
        bottomAnim = AnimationUtils.loadAnimation(this, R.anim.bottom_anim);

        movieImage = findViewById(R.id.movieImageViewById);
        movieImage.setAnimation(topAnim);

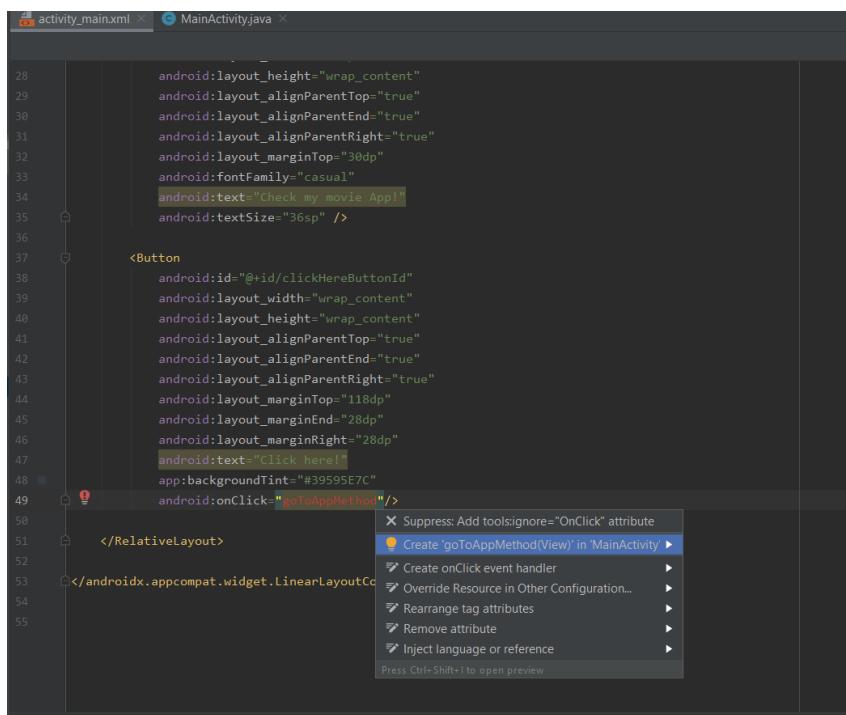
        checkMyMovieTextView = findViewById(R.id.checkMyMovieTextViewId);
        checkMyMovieTextView.setAnimation(bottomAnim);
    }
}
```

Rys. 25. Tworzenie logiki aktywności – utworzony kod.

Następną rzeczą, jest stworzenie metody, która będzie odpowiedzialna za przeniesienie nas do innej aktywności (ekranu) aplikacji. W tym celu chcielibyśmy wykorzystać stworzony przez nas przycisk – aby po kliknięciu na niego została wykonana metoda, której zadaniem jest otwarcie nowej aktywności. Możliwości wykonania takiej operacji są dwie:

- Stworzenie obiektu *button*, przypisanie do niego widoku z layout'u, a następnie stworzenie metody „nasłuchującej” kliknięcie przycisku,
- Druga metoda, polega na przypisaniu metody dla przycisku w pliku *activity\_main.xml* a następnie stworzeniu jej logiki w *MainActivity.java*.

Wykorzystamy możliwość drugą:



The screenshot shows the XML layout editor for activity\_main.xml. A button element is selected, and a context menu is open at its position. The menu includes options like 'Create > Click event handler' and 'Override Resource in Other Configuration...'. The code editor shows the XML structure with the button definition.

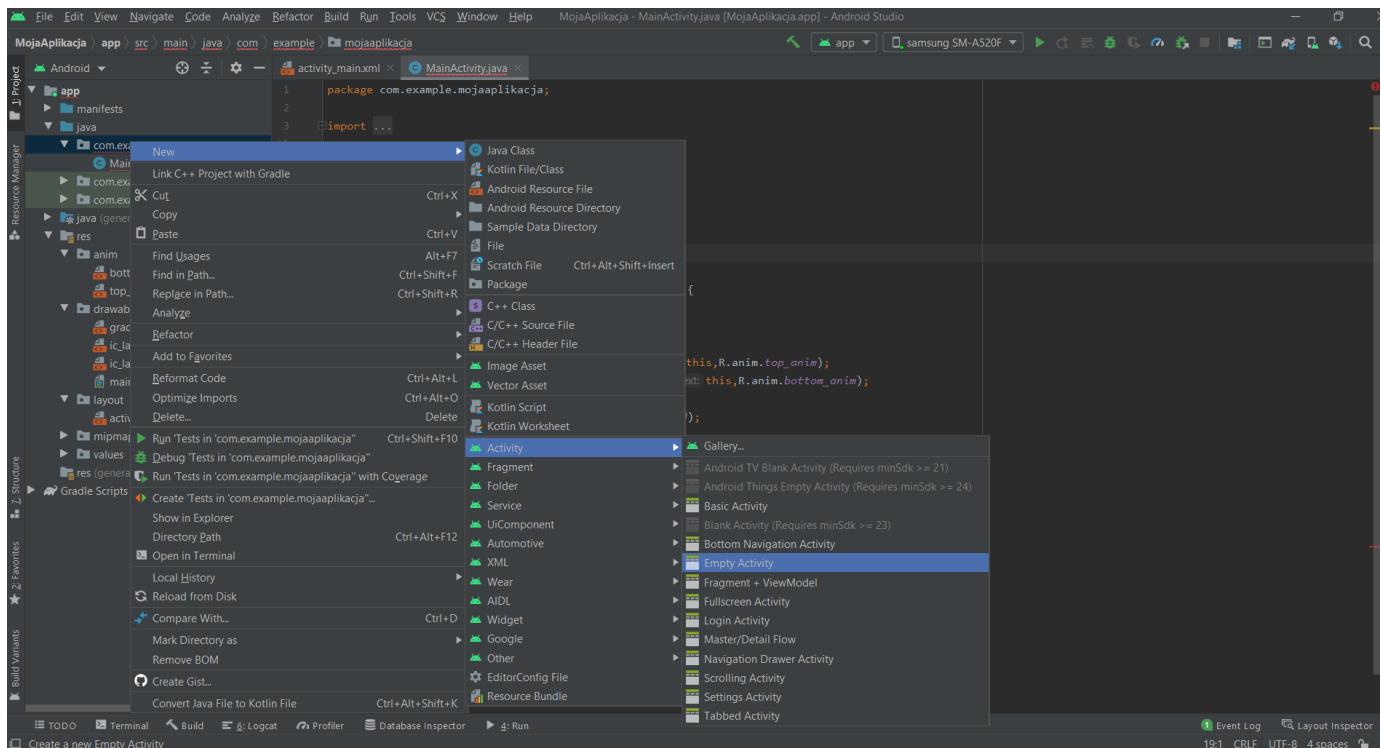
```
<Button
    android:id="@+id/clickHereButtonId"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_alignParentEnd="true"
    android:layout_alignParentRight="true"
    android:layout_marginTop="30dp"
    android:fontFamily="casual"
    android:text="Check my movie App!"
    android:textSize="36sp" />
```

Rys. 26. Tworzenie metody wykonywanej po kliknięciu przycisku.

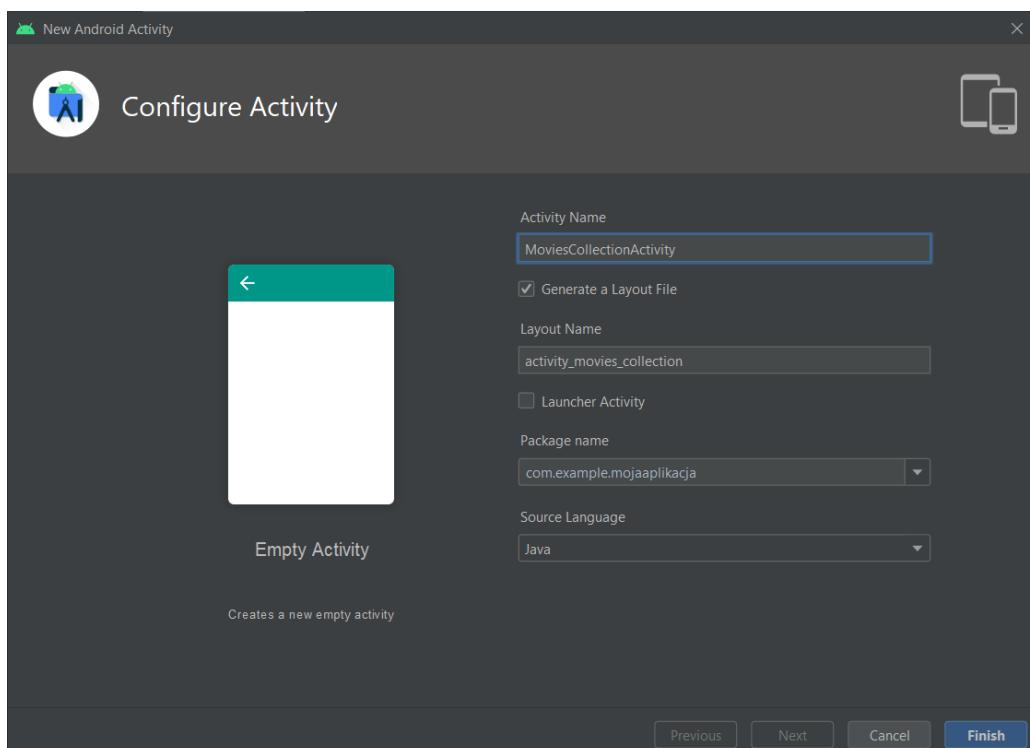
W tym celu użyto parametru *onClick*, a w cudzysłowie wpisano nazwę metody *goToAppMethod*. Środowisko AndroidStudio zgłosi nam błąd, ponieważ funkcja nie jest zdefiniowana w pliku *.java*. W tym celu należy wykorzystać wyżej opisany skrót klawiszowy i wybrać opcję stworzenia metody w *MainActivity.java*. Na tę chwilę należy pozostawić ją pustą – wróćmy do niej po stworzeniu nowej aktywności.

### 3.2.4. Tworzenie nowej aktywności.

W tym miejscu należy stworzyć nową aktywność, w której zostaną wyświetcone filmy oraz będzie możliwość dodania/usunięcia filmu. Tworzenie aktywności przedstawione zostało poniżej:



Rys. 27. Tworzenie nowej aktywności.



Rys. 28. Tworzenie nowej aktywności – okno konfiguracyjne.

Na Rys.28 przedstawione zostało okno konfiguracji nowo dodawanej aktywności. Podczas tworzenia, dodatkowo istnieje możliwość wygenerowania pliku layout'u z czego skorzystamy.

Po pomyślnym utworzeniu aktywności można przystąpić do definiowania utworzonej metody `goToAppMethod()` w pliku `MainActivity.java`.

Zadaniem jest, aby wykonanie metody powodowało przejście do nowej aktywności (`MoviesCollectionActivity`).

W tym celu wykorzystany zostanie `Intent`.

Intencje można wykorzystywać w różnych celach (np. komunikacja pomiędzy aplikacjami). Jednak jedną z ważniejszych funkcji jest uruchamianie nowych aktywności. W tym celu należy uzupełnić metodę `goToAppMethod` w następujący sposób:

```
public void goToAppMethod(View view) {  
    Intent goToMenuItem;  
    goToMenuItem = new Intent(getApplicationContext(), MoviesCollectionActivity.class);  
    startActivity(goToMenuItem);  
}
```

Początkowo deklarujemy obiekt `Intent nazwa`, następnie tworzymy ten obiekt podając `Context` oraz aktywność (z dopiskiem `.class`) do której chcielibyśmy przejść. Tak stworzony intent można wystartować za pomocą `startActivity(nazwa_intencji)`.

Więcej na temat intencji na: <https://developer.android.com/reference/android/content/Intent>

Nowo stworzona aktywność nie posiada jeszcze odpowiedniego layoutu. Należy przejść do pliku odpowiedzialnego za layout (`activity_movies_collection.xml`) jeśli nie został utworzony podczas konfiguracji należy stworzyć nowy i dodać go do pliku z rozszerzeniem `.java`. W tej grupie widoków wykorzystamy stworzone tło gradientowe. Jako layout wykorzystamy `LinearLayout`. Dodatkowo użyjemy kilku nowych widoków i parametrów, niektóre z nich opisane poniżej:

**parametry:**

- `android:background="?attr/selectableItemBackgroundBorderless", android:clickable="true, android:focusable="true"` - wykorzystywane do wyświetlania animacji kliknięcia w momencie naciśnięcia na widok;
- `android:foregroundGravity="left"` – umieszcza widok po lewej stronie rodzica,
- `android:scaleType="fitCenter"` – ten parametr powoduje dopasowanie naszego obrazu w `ImageView` do środka.

**Widoki:**

- `<androidx.appcompat.widget.AppCompatImageButton />` - Jest to przycisk obrazkowy – możliwość wykorzystania własnej grafiki.
- `<androidx.recyclerview.widget.RecyclerView />` - opisany zostanie poniżej.

Kod w *activity\_movies\_collection.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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:background="@drawable/gradient_background"
    android:orientation="vertical"
    tools:context=".MoviesCollectionActivity">

    <androidx.appcompat.widget.LinearLayoutCompat
        android:layout_width="match_parent"
        android:layout_height="80dp">

        <androidx.appcompat.widget.AppCompatImageButton
            android:id="@+id/addMovieImageButtonId"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="end|bottom"
            android:layout_margin="5dp"
            android:adjustViewBounds="true"
            android:background="?attr/selectableItemBackgroundBorderless"
            android:clickable="true"
            android:focusable="true"
            android:foregroundGravity="left"
            android:scaleType="fitCenter"
            android:src="@drawable/add_movie" />

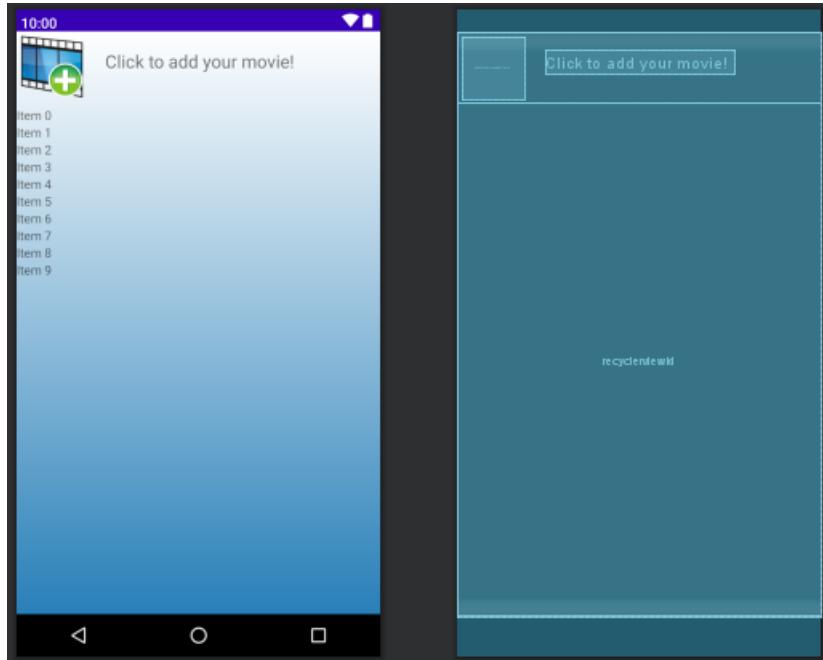
        <TextView
            android:id="@+id/clickToAddTextViewById"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="20dp"
            android:layout_marginLeft="20dp"
            android:layout_marginTop="20dp"
            android:background="?attr/selectableItemBackgroundBorderless"
            android:clickable="true"
            android:focusable="true"
            android:text="Click to add your movie!"
            android:textSize="20sp" />
    </androidx.appcompat.widget.LinearLayoutCompat>

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerviewId"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:paddingTop="5dp">
    </androidx.recyclerview.widget.RecyclerView>

</LinearLayout>
```

Jako obraz przycisku wykorzystana została grafika znajdująca się w katalogu *resources* o nazwie *add\_movie.png*.

Dodatkowo wykorzystano *RecyclerView*, gdzie odpowiednio ustawiono jego *id*, szerokość, wysokość oraz padding. Możemy teraz przejść do edytora graficznego i zobaczyć jak wygląda stworzony layout (rys. 29).



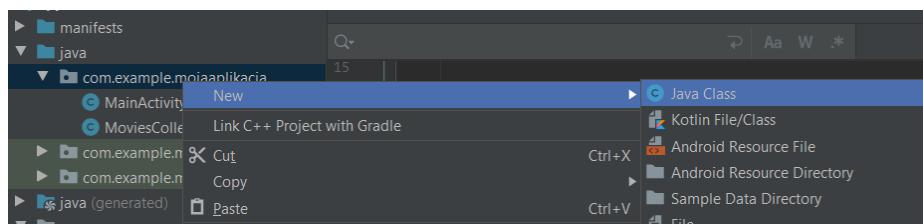
Rys. 29. Stworzony layout `activity_movies_collection.xml` w zakładce edytora graficznego.

### 3.2.5. RecyclerView.

Jest to komponent, którego zadaniem jest wyświetlanie listy elementów/obiektów o konkretnym układzie. `RecyclerView` za pomocą adaptera łączy widoki oraz inne dane używając `ViewHolder`.

Więcej na temat `RecyclerView`: <https://developer.android.com/guide/topics/ui/layout/recyclerview>.

Po stworzeniu `RecyclerView` w pliku `xml` należy utworzyć adapter, który zostanie później wykorzystany do załadowania danych - wyświetlenia kolekcji filmów. Dodajmy teraz nową klasę o nazwie `RecyclerViewAdapter` do naszego projektu (rys. 29)



Rys. 30. Tworzenie nowej klasy

Aby poprawnie wyświetlały się dane tworzony przez nas adapter musi rozszerzać klasę `RecyclerView.Adapter`, przekazując klasę `RecyclerViewAdapter`, gdzie zaimplementowany jest wzorzec `ViewHolder`. Obiekt `ViewHolder` opisuje i zapewnia dostęp do widoków. Jest to pewnego rodzaju wzorzec, który usprawnia renderowanie widoków aplikacji, ponieważ przechowuje on referencję do obiektów widoku `View`. Implementacja została przedstawiona na rys. 31.

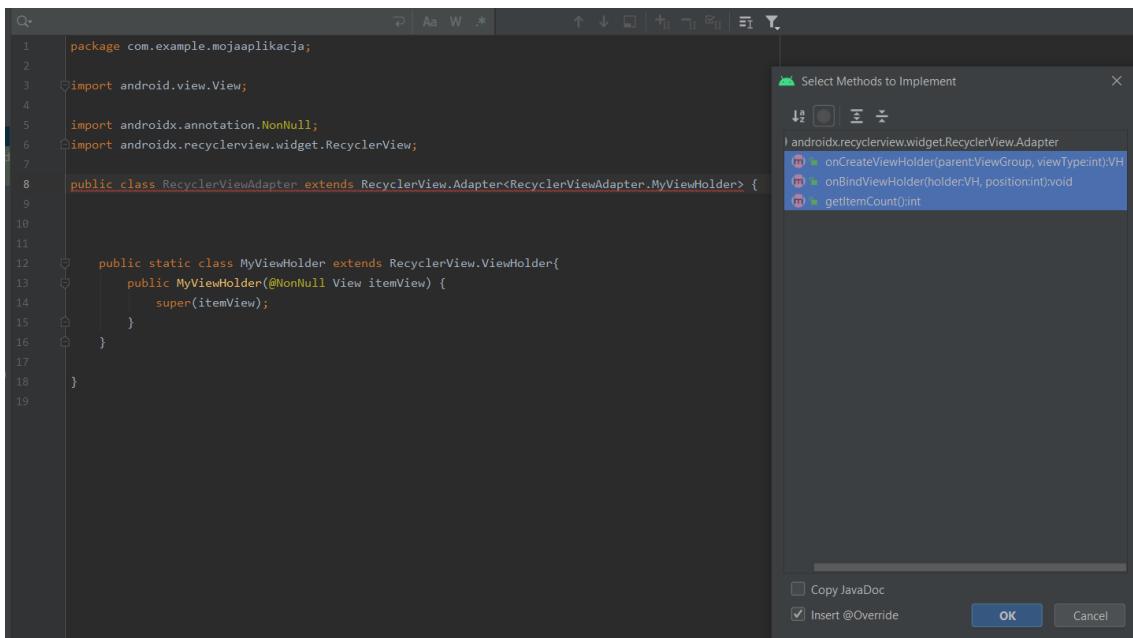
Aby dowiedzieć się więcej o `RecyclerView.ViewHolder` przejdź na:

<https://developer.android.com/reference/android/support/v7/widget/RecyclerView.ViewHolder.html>

```
activity_main.xml MainActivity.java activity_movies_collection.xml RecyclerViewAdapter.java
1 package com.example.mojaaplikacja;
2
3 import android.view.View;
4
5 import androidx.annotation.NonNull;
6 import androidx.recyclerview.widget.RecyclerView;
7
8 public class RecyclerViewAdapter extends RecyclerView.Adapter<RecyclerViewAdapter.MyViewHolder> {
9
10
11     public static class MyViewHolder extends RecyclerView.ViewHolder{
12         public MyViewHolder(@NonNull View itemView) {
13             super(itemView);
14         }
15     }
16 }
17
18 }
```

Rys. 31. Klasa `RecyclerViewAdapter` - rozszerzenie klasy oraz stworzenie klasy `MyViewHolder`, gdzie zaimplementowany zostanie wzorzec usprawniający renderowanie widoków.

W linii 8 (rys. 31) zasygnalizowany został błąd, w celu sprawdzenia sugerowanych propozycji naprawy, należy kliknąć na „czerwoną żarówkę” lub zastosować skrót klawiszowy alt+enter. Następnie wybieramy opcję *Implement methods*, otworzy się okno, gdzie wypisane będą sugerowane metody, które należy dodać. Należy zaznaczyć wszystkie, jak przedstawiono na rys. 32 i nacisnąć przycisk OK.



Rys. 32. Zaznaczenie dodania niezbędnych metod.

Po naciśnięciu przycisku OK powinniśmy otrzymać następujący rezultat:

```

4   import android.view.ViewGroup;
5
6   import androidx.annotation.NonNull;
7   import androidx.recyclerview.widget.RecyclerView;
8
9   public class RecyclerAdapter extends RecyclerView.Adapter<RecyclerAdapter.MyViewHolder> {
10
11     @NonNull
12     @Override
13     public MyViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
14       return null;
15     }
16
17     @Override
18     public void onBindViewHolder(@NonNull MyViewHolder holder, int position) {
19     }
20
21     @Override
22     public int getItemCount() {
23       return 0;
24     }
25
26     public static class MyViewHolder extends RecyclerView.ViewHolder{
27       public MyViewHolder(@NonNull View itemView) {
28         super(itemView);
29       }
30     }
31
32   }
33
34 }

```

Rys. 33. Dodanie niezbędnych metod.

Błąd spowodowany był brakiem kluczowych metod, które muszą być zawarte w każdej definicji adaptera. W adapterze najważniejsze są trzy metody:

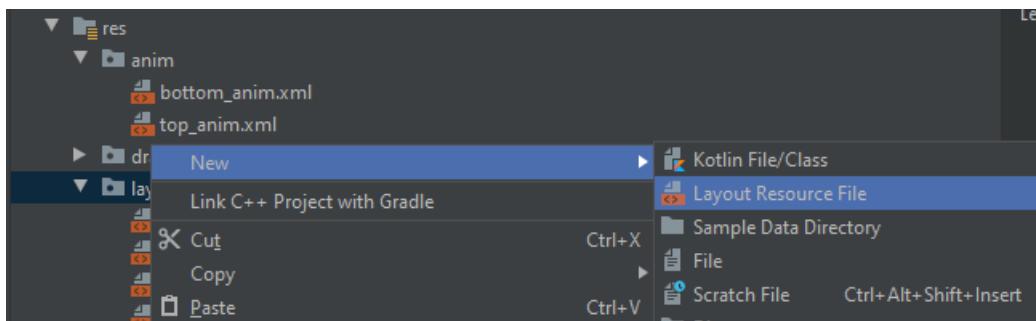
- **onCreateViewHolder()** - metoda ta wywoływana jest za każdym razem, gdy musi zostać stworzony nowy obiekt ViewHolder,
- **onBindViewHolder()** - jest wywoływana w celu uzupełnienia elementu odpowiednimi danymi, metoda ta pobiera odpowiednie dane i wykorzystuje je do wypełnienia widoku, do którego dostęp zapewnia ViewHolder,
- **getItemCount()** – metoda zwracająca ilość wszystkich elementów.

Spójrzmy jeszcze raz na rys. 29 - w stworzonym *RecyclerView* znajdują się kolejne pozycje. Chcemy, aby na poszczególnych pozycjach wyświetlały się filmy reprezentowane przez zdjęcie oraz tytuł filmu. W tym celu stworzymy nowy layout oparty na *CardView*.

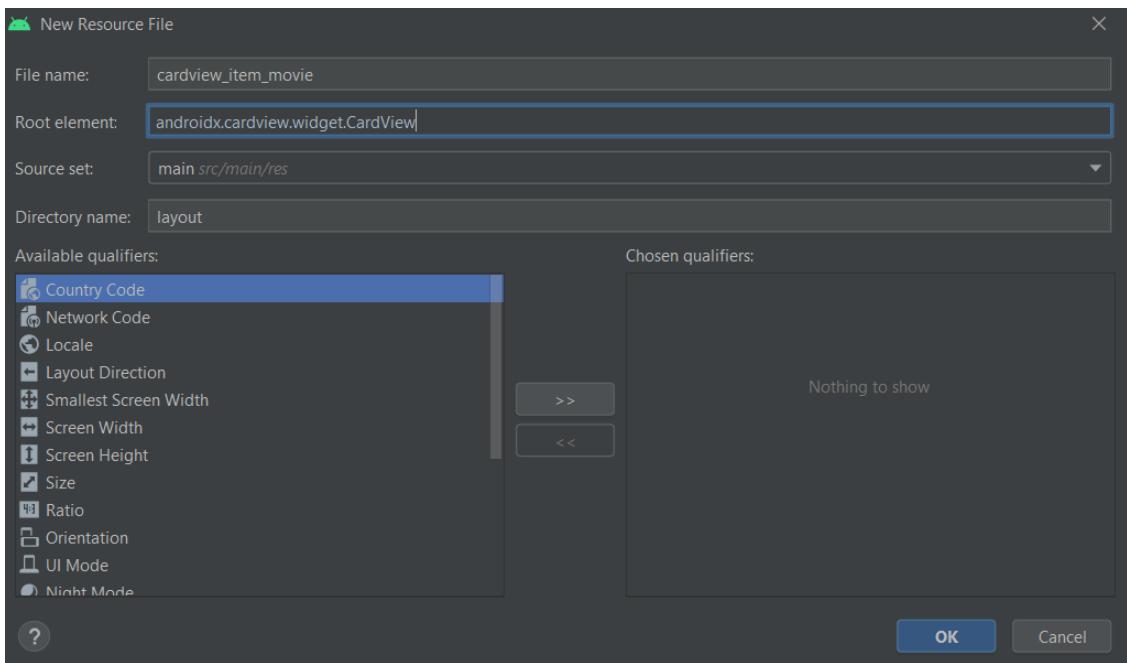
*CardView* to składnik interfejsu użytkownika, który umożliwia przedstawienie tekstu i obrazu w widoku przypominającym kartę.

### 3.2.6. Tworzenie nowego widoku *CardView*.

W celu stworzenia nowego layoutu należy kliknąć prawym przyciskiem myszy na folder layout i stworzyć nowy plik *Layout Resource File* (rys. 34). Na rys. 35 przedstawione jest okno konfiguracyjne tworzonego *CardView*.



Rys. 34. Tworzenie nowego widoku aplikacji.



Rys. 35. Tworzenie CardView - okno konfiguracyjne.

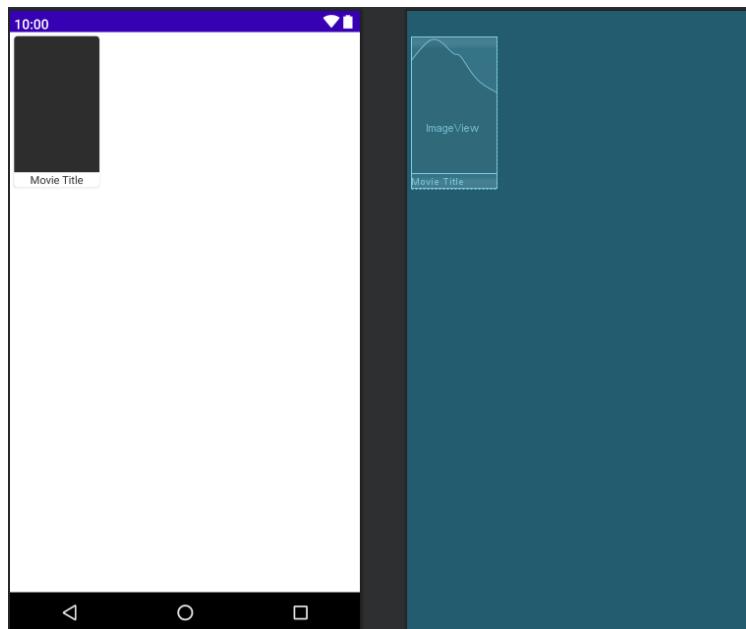
Po kliknięciu przycisku OK (rys. 35) od razu zostaniemy przeniesieni do stworzonego pliku, możemy przejść teraz do jego edycji. W celu stworzenia layoutu należy wykorzystać kod znajdujący się poniżej:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/cardViewId"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="5dp"
    android:clickable="true"
    android:focusable="true"
    android:foreground="?android:attr/selectableItemBackground"
    android:padding="5dp"
    app:cardCornerRadius="4dp">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">
        <ImageView
            android:id="@+id/movieThumbnailImageViewId"
            android:layout_width="100dp"
            android:layout_height="160dp"
            android:background="#2d2d2d"
            android:scaleType="centerCrop" />
        <TextView
            android:id="@+id/movieTitleTextViewId"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:gravity="center"
            android:text="Movie Title"
            android:textColor="#2d2d2d"
            android:textSize="13sp" />
    </LinearLayout>
</androidx.cardview.widget.CardView>
```

Graficzny efekt stworzonego kodu przedstawiono na rys. 36. Wykorzystane parametry oraz widoki zostały opisane podczas tworzenia poprzednich layoutów. Nowe parametry, które nie pojawiły się wcześniej to:

- `app:cardCornerRadius="4dp"` – parametr *CardView* odpowiedzialny za zaokrąglenie rogów „karty”, im większa wartość tym rogi robią się bardziej zaokrąglone,
- `android:scaleType="centerCrop"` - parametr *ImageView* odpowiedzialny za opcję skalowania obrazu do granic widoku, ustawienie go na *centerCrop* powoduje wyśrodkowanie obrazu w widoku i zachowanie jego proporcji.

Ważne, aby nie zapomnieć o przypisaniu *id* dla poszczególnych widoków umieszczonych w layoucie, ponieważ za ich pośrednictwem będziemy odwoływać się do widoków, np. w celu ustawienia obrazu filmu oraz tytułu jaki ma się wyświetlić w późniejszym czasie na ekranie smartfona użytkownika aplikacji.

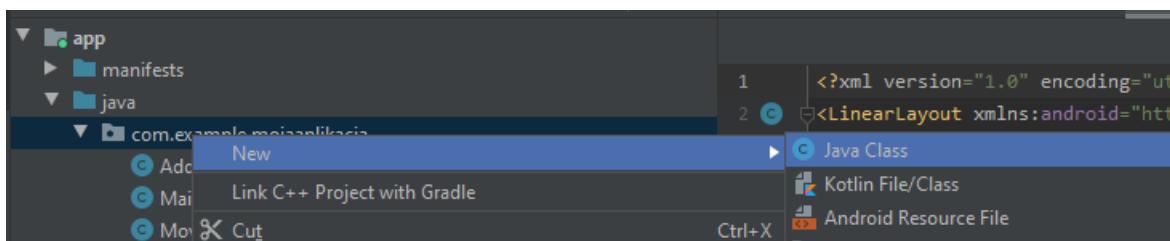


Rys. 36. Stworzony layout *cardview\_item\_movie.xml* w zakładce edytora graficznego.

### 3.2.7. Tworzenie klasy Movie.

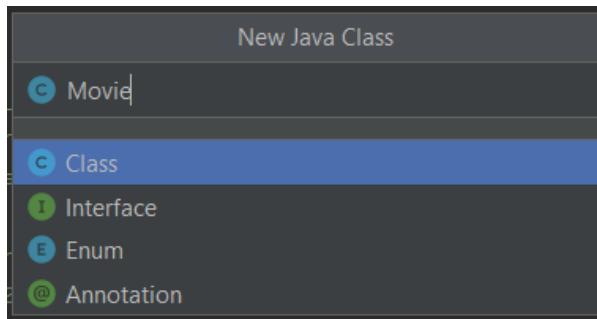
Chcielibyśmy, aby oprócz możliwości wyświetlania poszczególnych tytułów filmów wraz z obrazkami można było przeczytać opis filmu oraz sprawdzić jaka jest kategoria filmu. Dodatkowo też chcielibyśmy zapewnić, aby użytkownik aplikacji mógł ocenić film. Zatem każdy z filmów będzie posiadał swój tytuł, obrazek, kategorię filmu, opis, ocenę. Każdy film powinien mieć także swój unikalny klucz. Stworzymy klasę o nazwie *Movie*, która będzie posiadała pola określające właściwości danego filmu – będzie to reprezentacja pojedynczego filmu, który chcemy wyświetlać.

Na rysunku 37 przedstawiono tworzenie nowej klasy.



Rys. 37. Dodawanie nowej klasy do projektu.

Po wyborze *New -> Java Class* na ekranie pojawi się okno, gdzie należy wpisać nazwę tworzonej klasy (rys.38) oraz zatwierdzić naciskając przycisk *enter*.



Rys. 38. Dodawanie nowej klasy - ustawienie nazwy klasy.

W nowej klasie stworzymy następujące pola:

- key - identyfikator filmu,
- title - tytuł filmu,
- category - kategoria filmu,
- description - opis filmu,
- thumbnail - gdzie przechowywany będzie łańcuch znaków odnoszący się do minaturki filmu (obrazek),
- rate - ocena filmu.

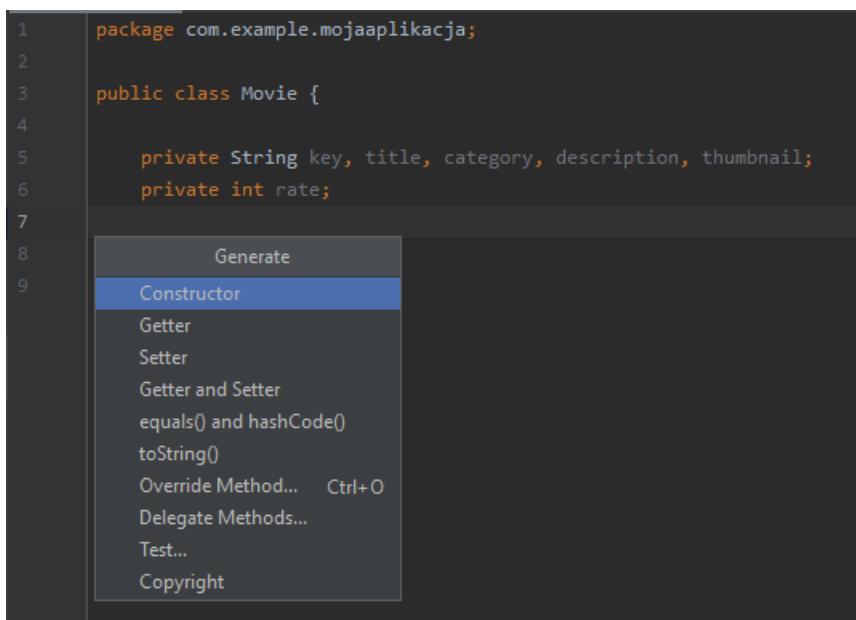
Utworzone pola powinny wyglądać następująco:

```
private String key, title, category, description, thumbnail;  
private int rate;
```

Wszystkie pola klasy należy ustawić jako prywatne. Aby klasa była gotowa należy jeszcze stworzyć konstruktor oraz dodać metody, które umożliwią nam pobranie wartości prywatnego pola na zewnątrz (*getter*) oraz takie, które pozwolą nam na zmianę wartości pola prywatnego z zewnątrz (*setter*).

Najpierw stworzymy pusty konstruktor, w tym celu skorzystamy ze skrótu klawiszowego alt+insert (rys. 39).

Wybieramy *Constructor*, następnie w oknie dialogowym wybieramy *Select None*.



Rys. 39. Efekt wykorzystania skrótu klawiszowego alt+insert.

Po dodaniu konstruktora przechodzimy do dodania metod *Getter* oraz *Setter*, korzystamy z tego samego skrótu klawiszowego (*alt + insert*). W okienku *Generate* (rys. 39) wybieramy *Getter and Setter*. Zaznaczamy wszystkie pozycje i zatwierdzamy klikając przycisk *OK* (rys. 40).

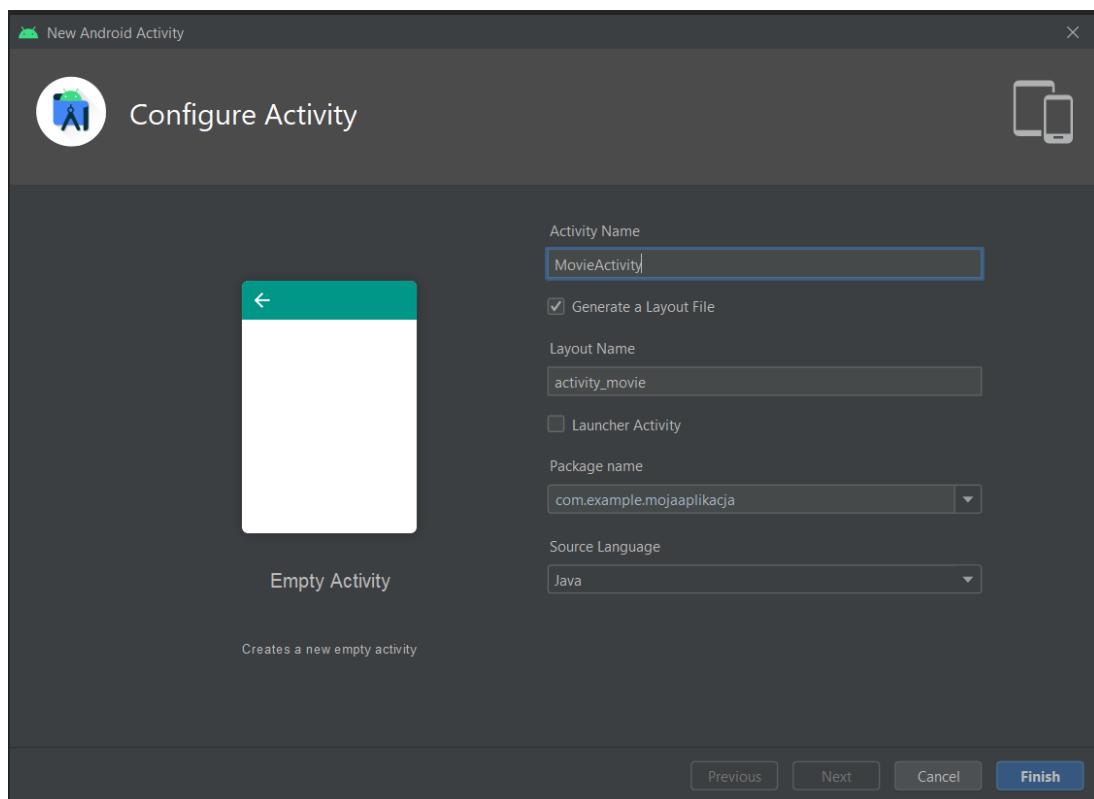
The screenshot shows the IntelliJ IDEA code editor with a Java file named `Movie.java`. The code defines a `Movie` class with fields: `key`, `title`, `category`, `description`, `thumbnail`, and `rate`. A tooltip from the 'Generate' tool window is displayed, showing the 'Select Fields to Generate Getters and Setters' dialog. The dialog lists the fields and their types: `key:String`, `title:String`, `category:String`, `description:String`, `thumbnail:String`, and `rate:int`. The 'Getter template' and 'Setter template' dropdowns are set to 'IntelliJ Default'. At the bottom right of the dialog are 'OK' and 'Cancel' buttons.

```
1 package com.example.mojaaplikacja;
2
3 public class Movie {
4
5     private String key, title, category, description, thumbnail;
6     private int rate;
7
8     public Movie() {
9     }
10
11 }
12
13
14
```

Rys. 40. Klasa Movie, tworzenie Getterów i Setterów.

### 3.2.8. Tworzenie aktywności pojedynczego filmu.

Stworzymy teraz nową aktywność, w której wyświetlona będzie miniaturka filmu, tytuł, kategoria, opis filmu, a także dostępna będzie możliwość oceny filmu. Podczas tworzenia aktywności wygenerujemy również layout powiązany z tą aktywnością. Na rys. 41 przedstawiono okno konfiguracji tworzenia nowej aktywności.



Rys. 41. Tworzenie nowej aktywności MovieActivity - okno konfiguracyjne.

Przejdźmy teraz do nowo stworzonego layoutu: *activity\_movie.xml*. Jako layout wykorzystany zostanie *LinearLayout*. Chcielibyśmy, aby na tym ekranie wyświetlana była miniatura filmu, a więc musimy dodać *ImageView*. Tytuł, kategoria oraz opis filmu będą wyświetlane w *TextView*. Dodatkowo, aby umożliwić użytkownikowi aplikacji ocenę danego filmu dodamy *RatingBar*.

*RatingBar* przedstawia ocenę w gwiazdkach. Dzięki niemu, użytkownik może dotknąć lub przeciągnąć palcem po ekranie w celu ustawienia oceny. W pliku *.xml* mamy możliwość ustawienia liczby gwiazdek za pośrednictwem parametru *android:numStars="6"* (ustawienie liczby gwiazdek na 6). Dodatkowo mamy możliwość manipulowania wartością kroku oceny poprzez zmianę parametru: *android:stepSize="0.5"* (ustawienie kroku oceny na 0.5). Pod *RatingBar* dodamy także *TextView*, w którym wyświetlona zostanie informacja o naszej ocenie filmu.

Ekran smartfona jest niewielki i w przypadku dodania bardzo długiego opisu filmu, część widoków mogłyby nie zmieścić się na ekranie, dlatego też skorzystamy z *androidx.core.widget.NestedScrollView*, dzięki czemu cały ekran będzie przewijalny.

Opisany layout można stworzyć poprzez wykorzystanie kodu znajdującego się poniżej:

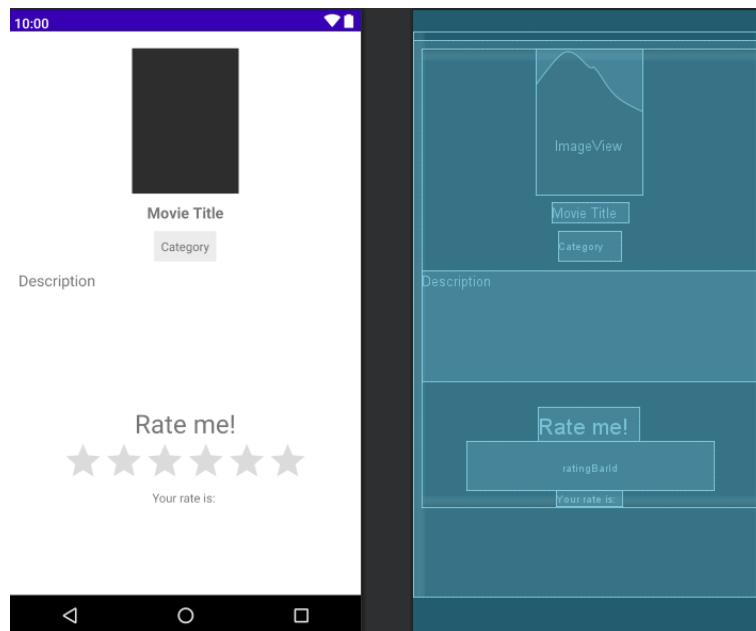
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MovieActivity">

    <androidx.core.widget.NestedScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginTop="10dp"
        android:padding="10dp">
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:gravity="center"
            android:orientation="vertical">
            <ImageView
                android:id="@+id/thumbnailInActivityImageViewById"
                android:layout_width="125dp"
                android:layout_height="170dp"
                android:background="#2d2d2d"
                android:scaleType="centerCrop" />
            <TextView
                android:id="@+id/titleTextViewId"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginTop="10dp"
                android:text="Movie Title"
                android:textSize="18sp"
                android:textStyle="bold" />
            <TextView
                android:id="@+id/categoryTextViewId"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginTop="10dp"
                android:background="#e8e8e8"
                android:padding="8dp"
                android:text="Category" />
            <TextView
                android:id="@+id/descriptionTextViewId"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginTop="10dp"
                android:background="#e8e8e8"
                android:padding="8dp"
                android:text="Description" />
        </LinearLayout>
    </NestedScrollView>
</LinearLayout>
```

```

        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:gravity="top|left"
        android:inputType="textMultiLine"
        android:maxLines="9"
        android:minLines="6"
        android:scrollbars="vertical"
        android:text="Description"
        android:textSize="18sp" />
<TextView
    android:id="@+id/rateTextViewId"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="30dp"
    android:text="Rate me!"
    android:textSize="30sp" />
<RatingBar
    android:id="@+id/ratingBarId"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:numStars="6"
    android:stepSize="0.5" />
<TextView
    android:id="@+id/userRateTextViewId"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:text="Your rate is: " />
</LinearLayout>
</androidx.core.widget.NestedScrollView>
</LinearLayout>
```

Po wklejeniu powyższego kodu powinniśmy otrzymać następujący rezultat:



Rys. 42. Stworzony layout activity\_movie.xml - zakładka edytora graficznego.

Tworzeniem logiki aktywności *MovieActivity.java* zajmiemy się w dalszej części instrukcji.

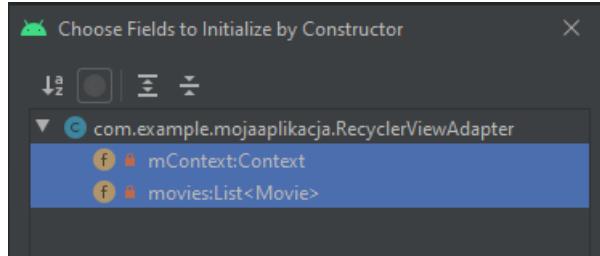
### 3.2.9. Stworzenie zmiennych oraz uzupełnienie metod w klasie RecyclerViewAdapter.

Dla przypomnienia: *RecyclerViewAdapter* chcemy wykorzystać do wyświetlania kolekcji filmów. Każdy film z kolekcji ma być przedstawiony za pomocą „karty” (*CardView*), na której widoczna będzie miniaturka filmu oraz jego tytuł.

Przejdzmy teraz do pliku *RecyclerViewAdapter.java* i stwórzmy w tej klasie dwa prywatne pola jedno będzie zawierało kontekst, a drugie będzie listą obiektów *Movie* (kolekcja naszych filmów):

```
private Context mContext;
private List<Movie> movies;
```

Następnie korzystamy ze skrótu klawiszowego *alt+insert*, wybieramy możliwość utworzenia konstruktora i wybieramy wszystkie proponowane pozycje (rys. 43).



Rys. 43. Tworzenie konstruktora klasy *RecyclerViewAdapter.java*.

Po dodaniu konstruktora kod powinien wyglądać jak na rys. 44.

```
1 package com.example.mojaaplikacja;
2
3 import android.content.Context;
4 import android.view.View;
5 import android.view.ViewGroup;
6
7 import androidx.annotation.NonNull;
8 import androidx.recyclerview.widget.RecyclerView;
9
10 import java.util.List;
11
12 public class RecyclerViewAdapter extends RecyclerView.Adapter<RecyclerViewAdapter.MyViewHolder> {
13
14     private Context mContext;
15     private List<Movie> movies;
16
17
18     public RecyclerViewAdapter(Context mContext, List<Movie> movies) {
19         this.mContext = mContext;
20         this.movies = movies;
21     }
22
23     @NonNull
24     @Override
25     public MyViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) { return null; }
26
27     @Override
28     public void onBindViewHolder(@NonNull MyViewHolder holder, int position) {
29
30     }
31
32     @Override
33     public int getItemCount() { return 0; }
34
35     public static class MyViewHolder extends RecyclerView.ViewHolder {
36
37
38
39     }
40 }
```

Rys. 44. Dodanie pól do klasy *RecyclerViewAdapter* (linia 14,15) oraz utworzenie konstruktora (linie 18-21).

Uzupełnijmy teraz klasę *MyViewHolder*. Każdy film z kolekcji ma być przedstawiony za pomocą „karty” (*CardView*), na której widoczna będzie miniaturka filmu oraz jego tytuł, a więc nasz *ViewHolder* musi posiadać obiekty, które będą reprezentować widoki znajdujące się w layout (*cardview\_item\_movie.xml*). Należy stworzyć obiekty typu *CardView*, *ImageView* oraz *TextView*, a następnie przypisać do nich widoki zadeklarowane w layout. Aby odnaleźć odpowiednie widoki wykorzystuje się unikalne *id* przypisane w pliku *cardview\_item\_movie.xml*.

Uzupełniona klasa powinna wyglądać następująco:

```
public static class MyViewHolder extends RecyclerView.ViewHolder {  
    TextView movieTitleTextView;  
    ImageView movieThumbImageView;  
    CardView cardView;  
  
    public MyViewHolder(@NonNull View itemView) {  
        super(itemView);  
        movieTitleTextView = itemView.findViewById(R.id.movieTitleTextViewId);  
        movieThumbImageView = itemView.findViewById(R.id.movieThumbnailImageViewId);  
        cardView = itemView.findViewById(R.id.cardViewId);  
    }  
}
```

Przejdźmy teraz do uzupełnienia zawartości trzech kluczowych metod, które musi posiadać adapter. Rozpoczniemy od `getCount()`, jak już wcześniej wspomniano, metoda ta zwraca ilość wszystkich elementów. Po utworzeniu metody automatycznie przypisane zostało, że metoda ta zwraca wartość 0, a więc musimy to zmienić. Chcemy zwrócić ilość wszystkich elementów, a więc zwrócić rozmiar naszej listy `movies`, gdzie zapisane będą obiekty `Movie` - cała nasza kolekcja filmów. Zawartość metody po edycji powinna wyglądać następująco:

```
@Override  
public int getItemCount() {  
    return movies.size();  
}
```

Zajmijmy się teraz `onCreateViewHolder()` metoda ta wywoływana jest zaraz po utworzeniu adaptera i służy do inicjowania elementów `ViewHolder`.

`onCreateViewHolder` zwraca nowy `ViewHolder`, który przechowuje widoki w danym typie. Wewnątrz tej metody wykorzystamy `LayoutInflater`, który tworzy instancję pliku XML. Tutaj więcej na temat `LayoutInflater`:  
<https://developer.android.com/reference/android/view/LayoutInflator>.

Zawartość metody `onCreateViewHolder` powinna wyglądać następująco:

```
@NonNull  
@Override  
public MyViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {  
    View view;  
    LayoutInflater mInflater = LayoutInflater.from(mContext);  
    view = mInflater.inflate(R.layout.cardview_item_movie, parent, false);  
    return new MyViewHolder(view);  
}
```

Do uzupełnienia została ostatnia z trzech metod, które są wymagane do poprawnego działania adaptera, jest to `onBindViewHolder()`. Metoda ta jest wywoływana przez `RecyclerView` w celu wyświetlenia danych na określonej pozycji. Metoda ta przyjmuje dwa argumenty: `holder`, który reprezentuje zawartość elementu w danej pozycji w zestawie danych. Drugim parametrem jest `position`, czyli pozycja elementu w zestawie danych adaptera.

Za pośrednictwem `holdera` na określonej pozycji będziemy prezentowali dane, którymi będzie miniatura filmu oraz jego tytuł.

Miniatury filmów zapisywane będą w obiekcie `Movie` jako łańcuch znaków, będący adresem URI. Z tego względu wykorzystamy bibliotekę `Android Picasso`, która umożliwia ładowanie obrazu z URI. Aby umożliwić korzystanie z tej biblioteki musimy dodać ją do projektu. W tym celu przejdźmy do pliku `build.gradle` modułu i w `dependencies` dodajmy poniższą implementację (rys. 45 linia 39):

```
implementation 'com.squareup.picasso:picasso:2.71828'
```

Zawartość `dependencies` w pliku `build.gradle` modułu powinna wyglądać następująco:

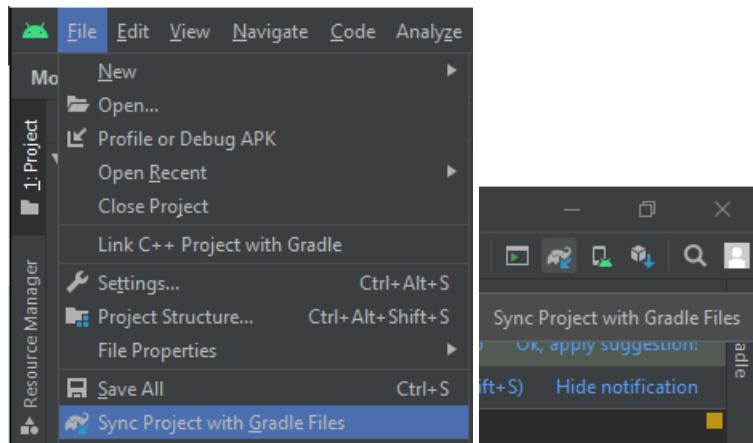
```

dependencies {
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'com.google.android.material:material:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
    implementation 'com.squareup.picasso:picasso:2.71828'
}

```

Rys. 45. Dodanie biblioteki Picasso w pliku build.gradle modułu (linia 39).

Po dodaniu implementacji należy dokonać synchronizacji projektu z plikiem gradle. W tym celu można w lewym górnym rogu wybrać *File -> Sync Project with Gradle Files* lub w prawym górnym rogu wybrać ikonkę przedstawioną na rys. 46 po prawej stronie.



Rys. 46. Synchronizacja projektu z plikami gradle.

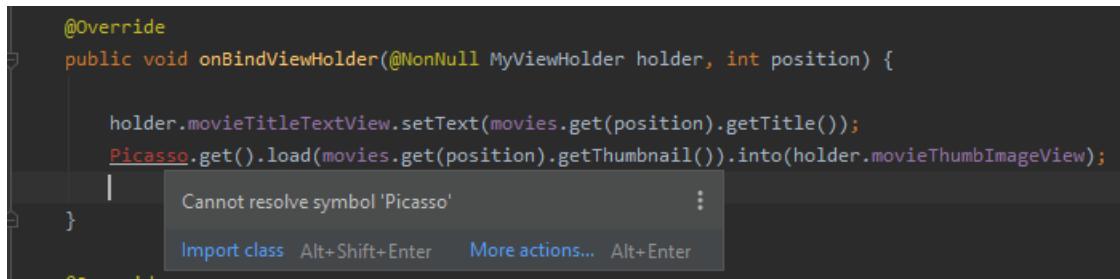
Po dokonaniu synchronizacji możemy wrócić do uzupełniania *onBindViewHolder* w pliku *RecyclerViewAdapter.java*. Przy wykorzystaniu holdera ustawmy w *TextView* tytuł filmu pobrany z listy filmów *movies* oraz w *ImageView* ustawmy miniaturkę filmu wykorzystując zainportowaną bibliotekę Picasso. Implementacja przedstawiona została poniżej:

```

@Override
public void onBindViewHolder(@NonNull MyViewHolder holder, int position) {
    holder.movieTitleTextView.setText(movies.get(position).getTitle());
    Picasso.get().load(movies.get(position).getThumbnail()).into(holder.movieThumbImageView);
}

```

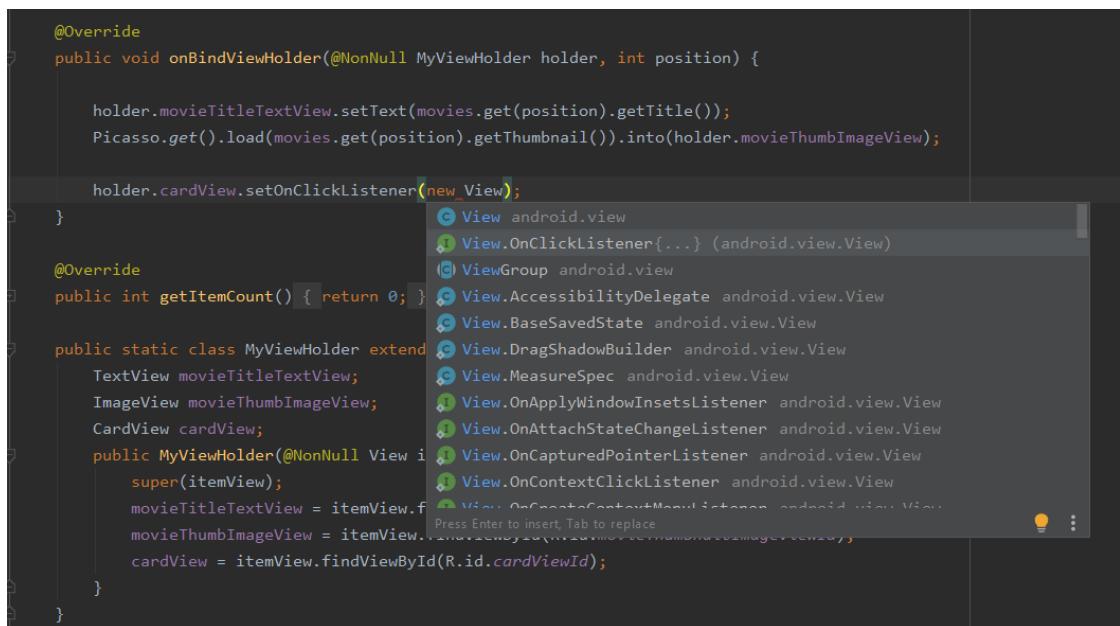
Po dodaniu dwóch brakujących linii do kodu, słowo „Picasso” będzie oznaczone kolorem czerwonym, aby zobaczyć co może powodować błąd przesunięty kurSOR na słowo Picasso (rys.47).



Rys. 47. Błąd spowodowany wykorzystaniem Picasso.

Aby naprawić błąd musimy zaimportować klasę, w tym celu użyjmy sugerowanego skrótu klawiszowego (rys. 47) *Alt+Shift+Enter* po zaimportowaniu klasy błąd zniknie.

Chcielibyśmy także, aby na każdy z filmów można było kliknąć, aby uzyskać więcej szczegółowych informacji. W tym celu utworzymy metodę *onClickListener()* (rys. 48).



Rys. 48. Tworzenie onClickListenera dla CardView.

Jeśli zaczniemy wpisywać jak pokazano na rys. 48 wyświetli nam się podpowiedź, należy wybrać zaznaczoną pozycję i kliknąć klawisz *enter*, wtedy automatycznie stworzy się „szkielet” metody. Chcielibyśmy, aby po kliknięciu na dany film w *CardView* przejść do aktywności *MovieActivity*. W tym celu wykorzystamy *Intent*, którego zadaniem będzie uruchomienie nowej aktywności. Także za pomocą *intent.putExtra* przekażemy dodatkowe dane, które zawarte są w danym obiekcie *Movie* na liście *movies*. Po stworzeniu intentu i przypisaniu odpowiednich wartości możemy przejść do nowej aktywności poprzez wywołanie: *mContext.startActivity(intent)*. Poniżej przedstawiono zawartość metody *onBindViewHolder* wraz z uzupełnionym *onClickListenerem*:

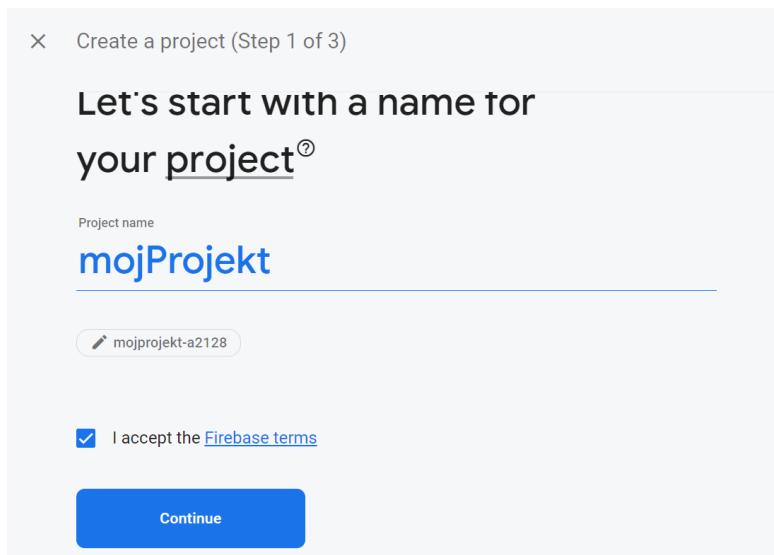
```
@Override  
public void onBindViewHolder(@NonNull MyViewHolder holder, int position) {  
  
    holder.movieTitleTextView.setText(movies.get(position).getTitle());  
  
    Picasso.get().load(movies.get(position).getThumbnail()).into(holder.movieThumbImageView);  
  
    holder.cardView.setOnClickListener(new View.OnClickListener() {  
        @Override
```

```

public void onClick(View v) {
    Intent intent = new Intent(mContext, MovieActivity.class);
    intent.putExtra("MovieTitle", movies.get(position).getTitle());
    intent.putExtra("Description", movies.get(position).getDescription());
    intent.putExtra("Thumbnail", movies.get(position).getThumbnail());
    intent.putExtra("Category", movies.get(position).getCategory());
    intent.putExtra("Rate", movies.get(position).getRate());
    intent.putExtra("Key", movies.get(position).getKey());
    //start the activity
    mContext.startActivity(intent);
}
});
```

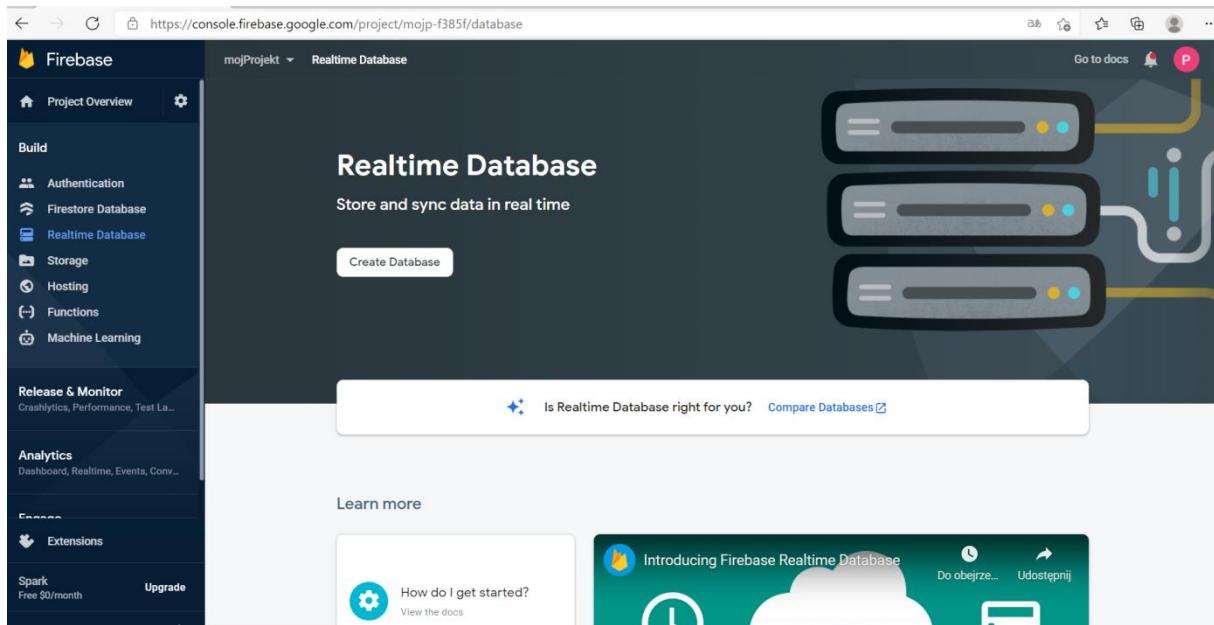
### 3.2.10. Konfiguracja Google Firebase.

Zajmiemy się teraz konfiguracją bazy danych czasu rzeczywistego *Google Firebase*, która będzie potrzebna do gromadzenia danych na temat konkretnych filmów. Aby wykonać tę część instrukcji konieczne jest posiadanie konta Google. Jeśli jesteś zalogowany na konto Google przejdź na stronę konsoli: <https://console.firebaseio.google.com/>. Następnie wybierz opcję tworzenia nowego projektu, po wybraniu tej propozycji powinno pojawić się okno z trzema krokami tworzenia projektu (rys. 49).



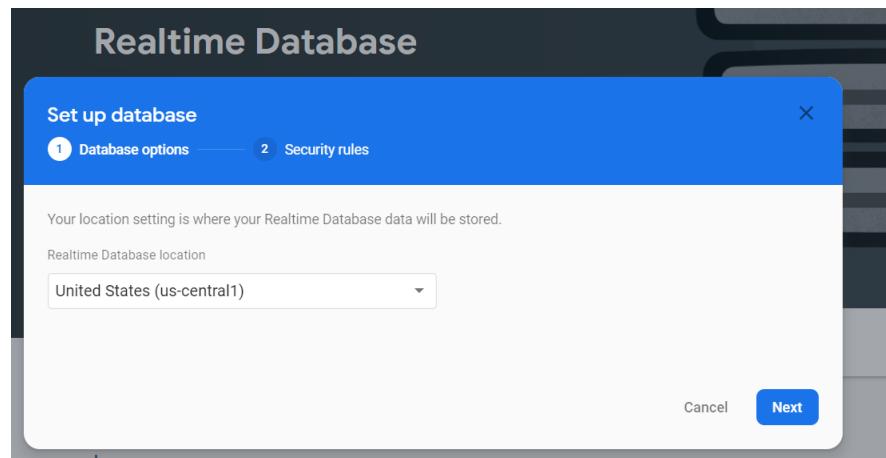
Rys. 49. Pierwszy krok tworzenia projektu.

W pierwszym kroku należy wpisać nazwę projektu i zaakceptować warunki korzystania z Firebase. Kolejne kroki należy przejść zostawiając zaznaczone rekommendowane ustawienia. Po przejściu kroków przez które prowadzi kreator należy przejść do konsoli nowo stworzonego projektu oraz kliknąć z zakładkę *Build -> Realtime Database* (rys. 50).

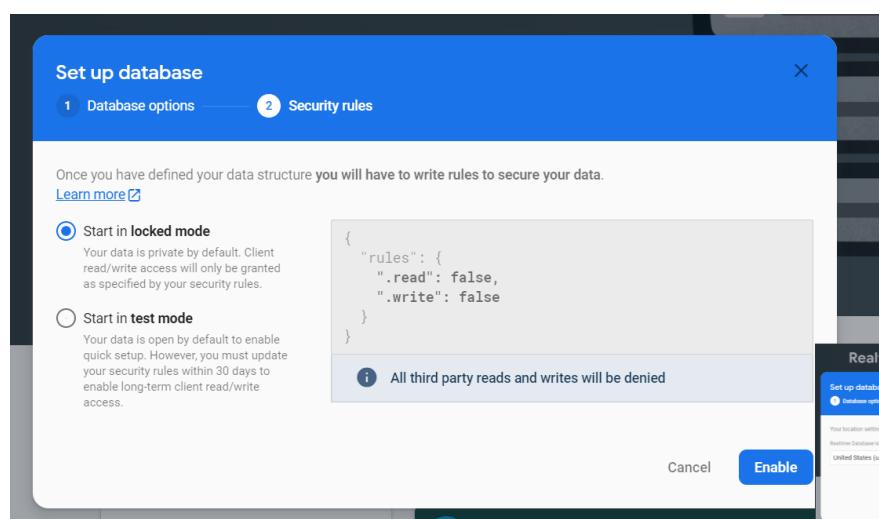


Rys. 50. Konsola nowo stworzonego projektu -> zakładka Realtime Database.

W zakładce *Build* -> *Realtime Database* (rys. 50) naciśnij przycisk *Create Database* w celu utworzenia bazy danych, która będzie dołączona do tworzonej przez nas aplikacji. Po zasygnalizowaniu chęci stworzenia bazy danych wyświetli się okno dialogowe, gdzie można wybrać lokalizację bazy danych (rys. 51) oraz w drugim kroku konfiguruje się zasady bezpieczeństwa (rys. 52).

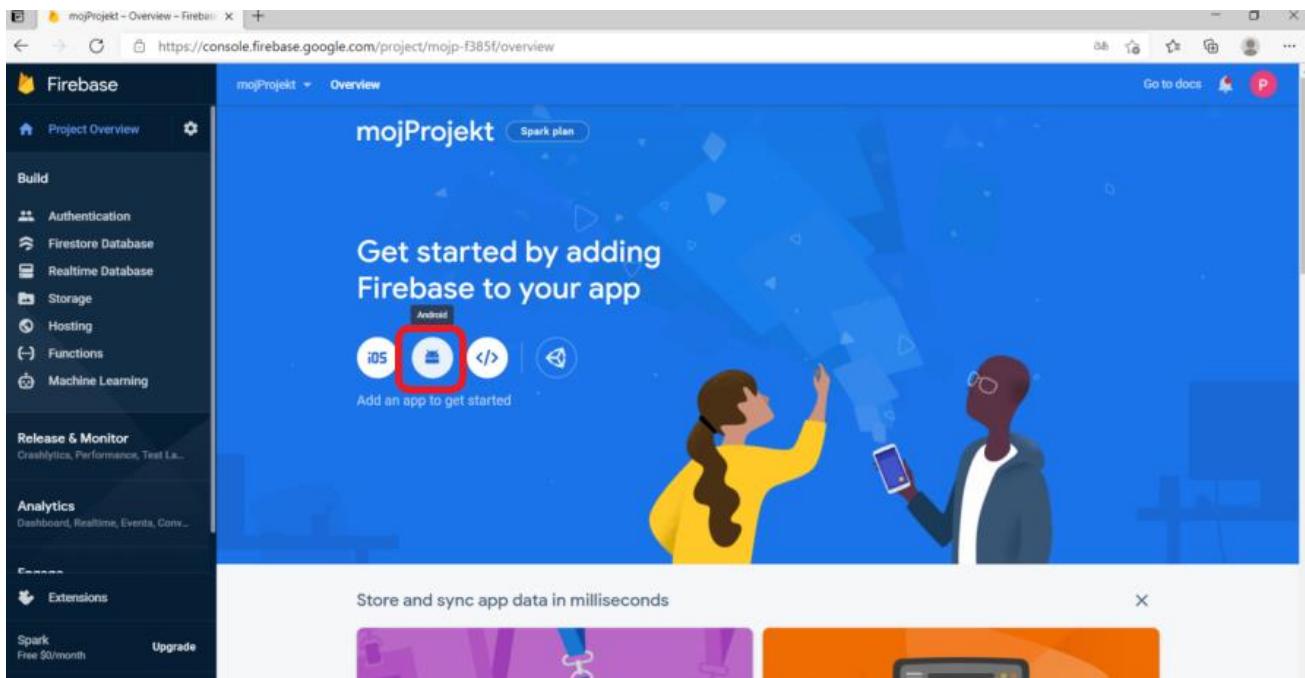


Rys. 51. Tworzenie Realtime Database - wybór lokalizacji bazy danych.



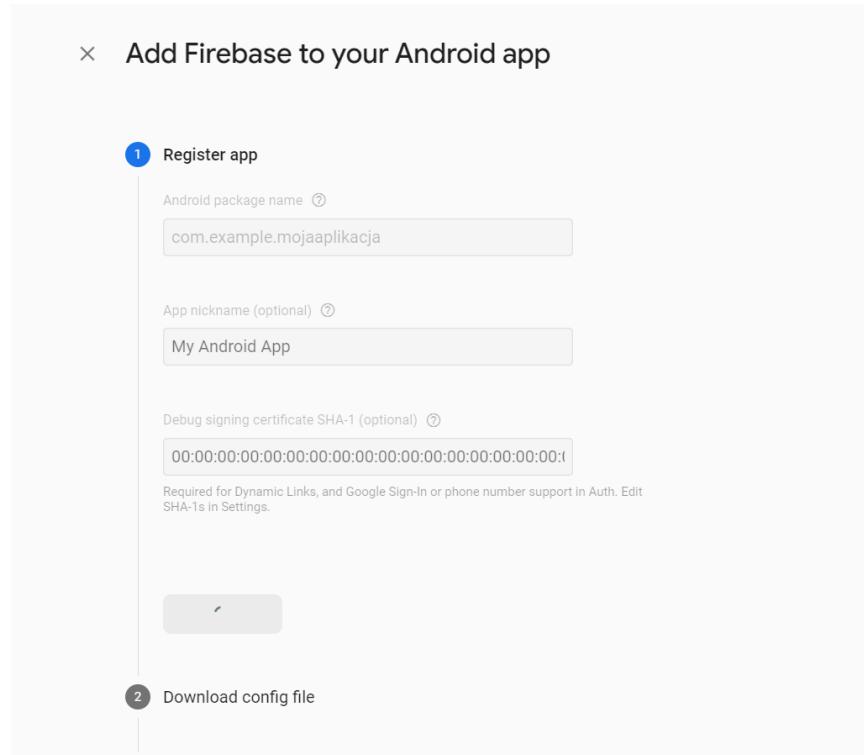
Rys. 52. Tworzenie Realtime Database - zasady bezpieczeństwa.

Po naciśnięciu przycisku *Enable* (rys. 51) mamy już stworzoną bazę danych, teraz musimy dodać ją do swojego projektu, w tym celu należy nacisnąć przycisk z ikonką androida tak jak przedstawiono na rys. 53.



Rys. 53. Dodawanie bazy danych do naszego projektu.

Aby dodać bazę danych do aplikacji, należy postępować według kroków, przez które prowadzi kreator. Na rys. 54 przedstawiony jest pierwszy krok. Nazwę którą należy wpisać do первого поля możemy znaleźć w pliku *AndroidManifest.xml* projektu (rys. 55).



Rys. 54. Dodawanie bazy danych do aplikacji - krok pierwszy.

```

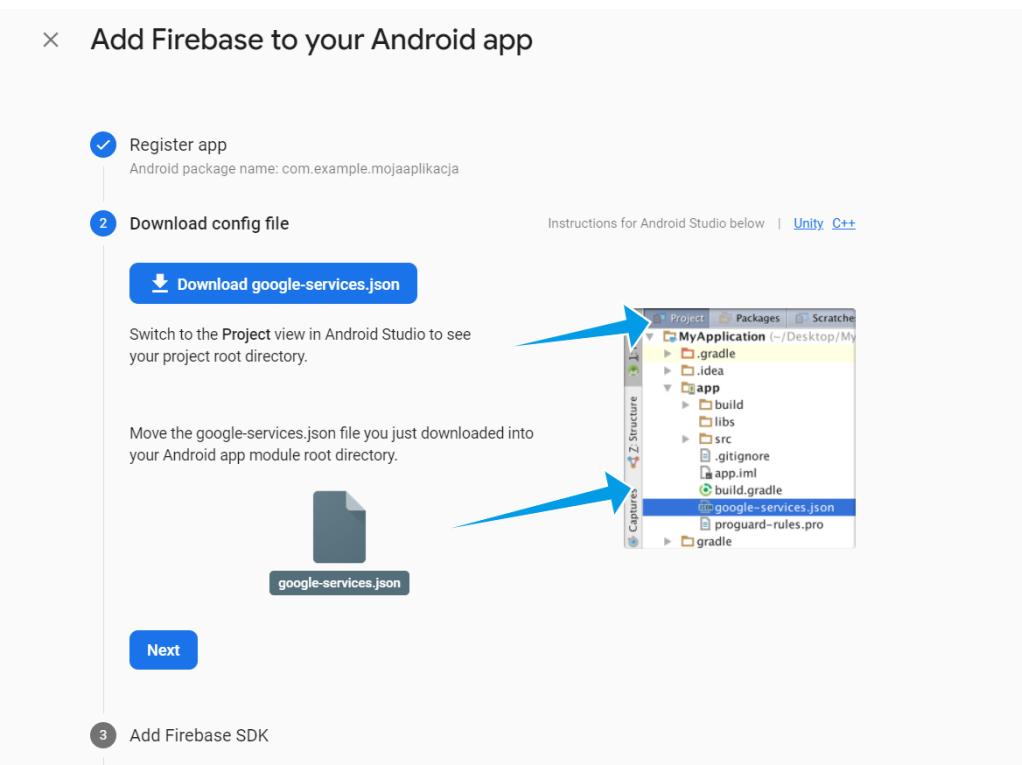
MojaAplikacja > app > src > main > AndroidManifest.xml
1: Project
  - app
    - manifests
      - AndroidManifest.xml (highlighted)
    - java
    - res
    - res (generated)
  - Gradle Scripts
    - build.gradle (Project: MojaAplikacja)
    - build.gradle (Module: MojaAplikacja.app)
    - gradle-wrapper.properties (Gradle Version)
    - proguard-rules.pro (ProGuard Rules for MojaAplikacja.app)
    - gradle.properties (Project Properties)
    - settings.gradle (Project Settings)
    - local.properties (SDK Location)

MovieActivity.java x activity_movies_collection.xml x
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
  <package android:name="com.example.mojaaplikacja" />
  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="MojaAplikacja"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.MojaAplikacja">
    <activity android:name=".AddMovieActivity" />
    <activity android:name=".MovieActivity" />
    <activity android:name=".MoviesCollectionActivity" />
    <activity android:name=".MainActivity" />
  </application>
</manifest>

```

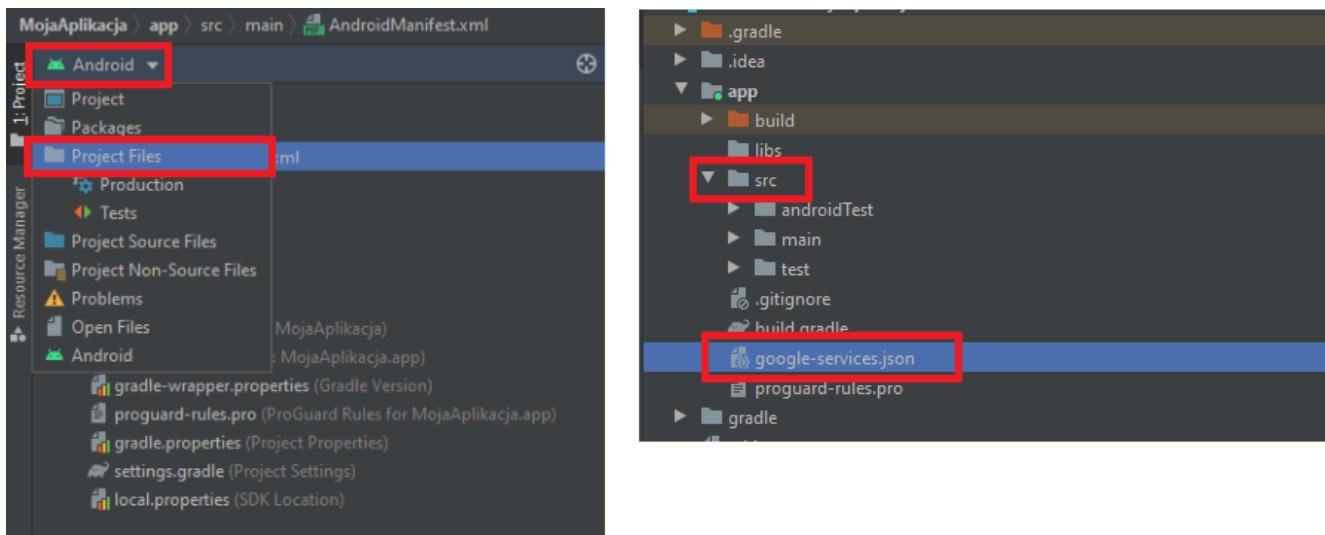
Rys. 55. Plik AndroidManifest.xml - nazwa package po prawej stronie.

W kolejnym kroku należy pobrać plik konfiguracyjny i umieścić go w projekcie tak jak pokazano na rys. 56.



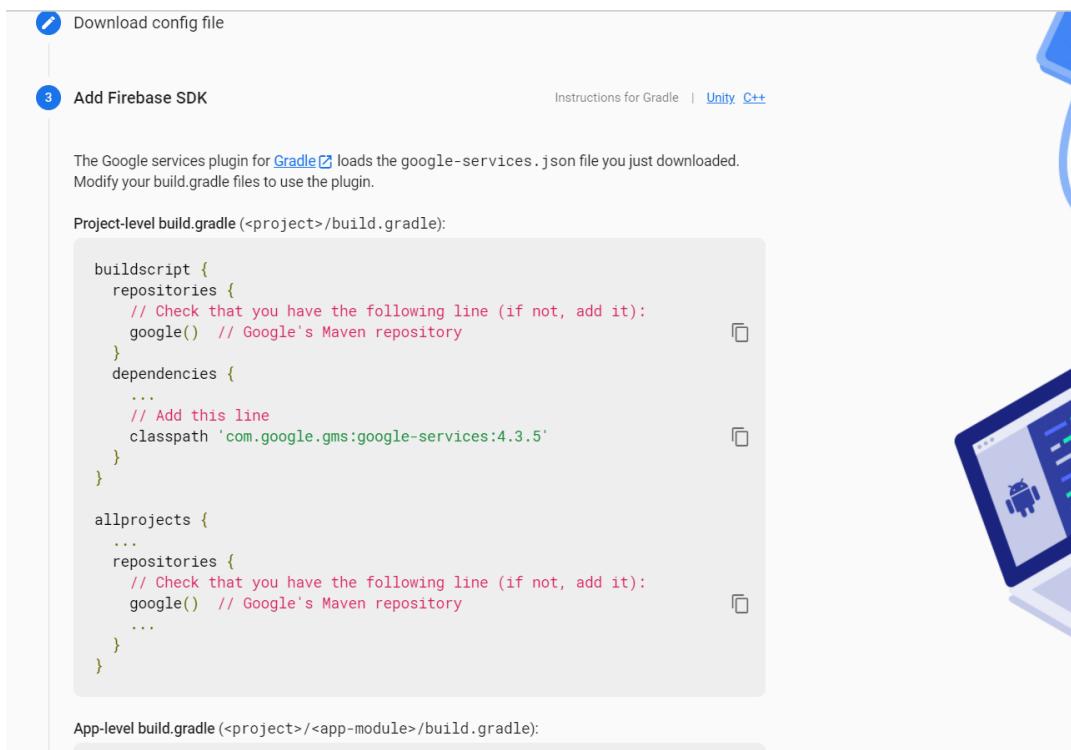
Rys. 56. Dodawanie bazy danych do aplikacji - krok drugi, dodanie pliku konfiguracyjnego.

Aby zobaczyć czy plik konfiguracyjny umieszczony został w odpowiednim miejscu należy w lewym górnym rogu przełączyć się na widok plików projektu, a następnie zobaczyć czy pobrany plik został umieszczony w folderze `src` (rys. 57).



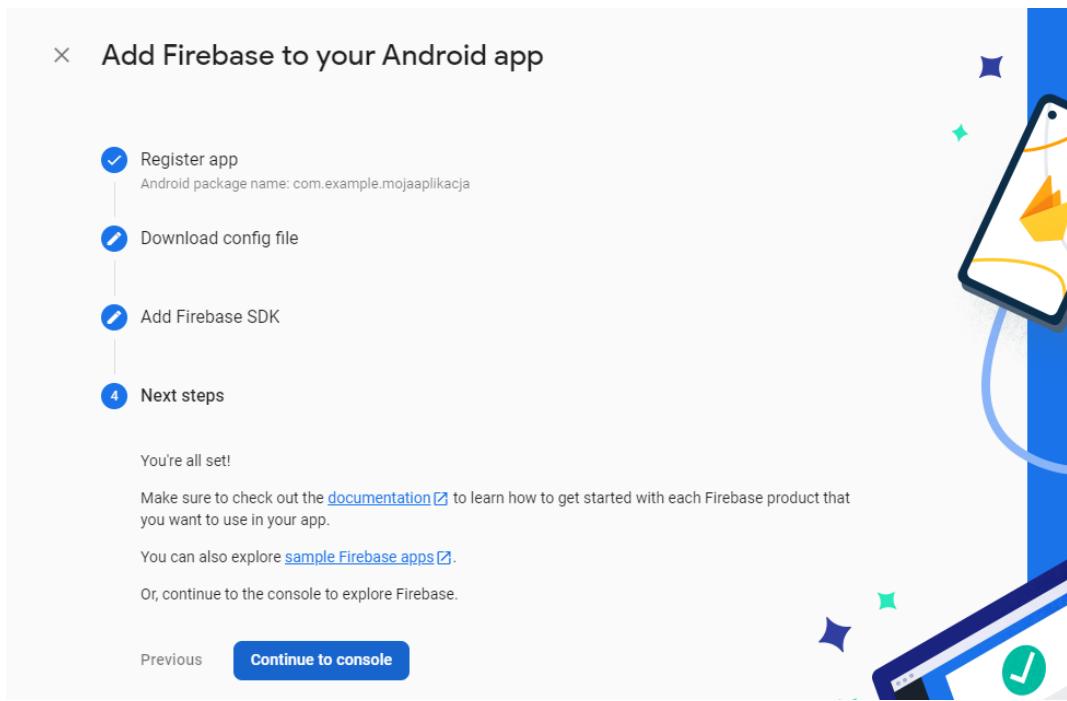
Rys. 57. Sprawdzenie poprawności dodania pliku konfiguracyjnego do projektu.

Jeśli plik został poprawnie umieszczony w projekcie możemy przejść do kolejnych kroków. Należy dodać Firebase SDK w tym celu należy zmodyfikować pliki *gradle* (dokładnie opisane w kreatorze dodawania bazy danych do projektu rys. 57).



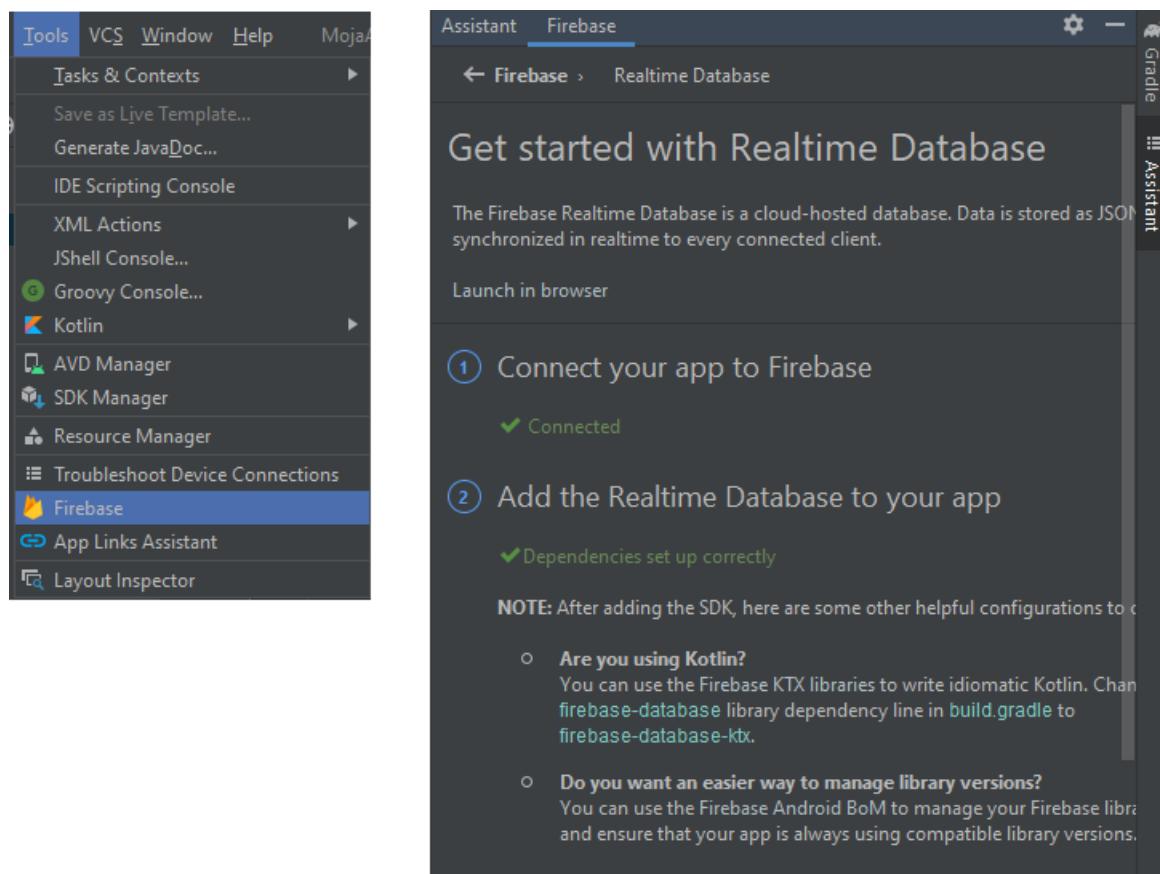
Rys. 58. Dodawanie bazy danych do aplikacji - krok trzeci, dodanie Firebase SDK.

Po dodaniu SDK i przejściu dalej wyświetli się następujące okno:



Rys. 59. Dodawanie bazy danych do aplikacji - krok czwarty.

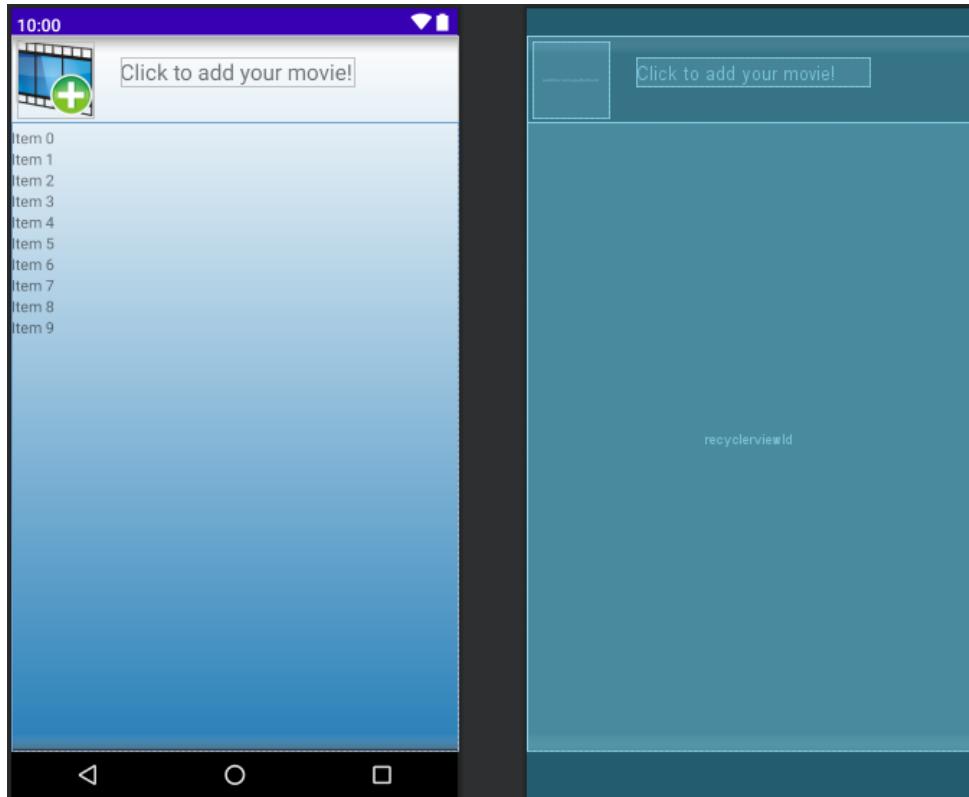
Po przejściu wszystkich kroków baza danych powinna być już połączona do naszej aplikacji. Poprawność połączenia możemy sprawdzić w Android Studio wybierając na górnym pasku *Tools* -> *Firebase*. Następnie wybieramy *Realtime Database* (rys. 59).



Rys. 59. Zakładka *Tools* -> *Firebase* (po lewej stronie), po wybraniu *Realtime Database* możemy sprawdzić poprawność połączenia bazy danych do aplikacji (po prawej stronie).

### 3.2.11. Uzupełnianie logiki aktywności *MoviesCollectionActivity.java*.

Przypomnijmy sobie jak wygląda layout (*activity\_movies\_collection.xml*) wykorzystywany w aktywności *MoviesCollectionActivity.java*. Głównymi elementami wchodząymi w skład layoutu (rys.60) jest część odpowiedzialna za dodanie nowego filmu do naszej kolekcji (*ImageButton*, *TextView*) oraz *RecyclerView*, gdzie wyświetlana będzie kolekcja filmów (miniatura filmu oraz jego tytuł). Teraz przejdziemy do tworzenia logiki aktywności powiązanej z tym layoutem.



Rys. 60. Layout *activity\_movies\_collection.xml* - zakładka edytora graficznego.

Należy przejść do pliku *MoviesCollectionActivity.java* i stworzyć listę filmów, referencję do bazy danych oraz obiekty reprezentujące nasze widoki w layout:

```
public class MoviesCollectionActivity extends AppCompatActivity {

    List<Movie> movieList;
    ImageButton addMovieBtn;
    RecyclerViewAdapter myAdapter;
    DatabaseReference databaseReference;

    @Override
    protected void onCreate(Bundle savedInstanceState) {...}
}
```

Przejdźmy teraz do metody *onCreate()* i przypiszmy do stworzonych obiektów konkretne widoki, stwórzmy listę filmów, a także adapter, do którego przekażemy listę filmów. W tym miejscu stworzymy również „węzeł” - child w naszej bazie danych. Aby to zrobić najpierw musimy pobrać instancję bazy danych wykorzystując metodę *getInstance()*, a następnie *getReference()*, która umożliwia odczyt lub zapis danych z/do bazy.

Więcej na temat zapisu/odczytu danych z Firebase: <https://firebase.google.com/docs/database/android/read-and-write>

W naszym *RecyclerView*, każda pozycja, czyli każdy film reprezentowany będzie przez *CardView*, jednak chcemy, aby w jednym wierszu mieściły się trzy filmy. Aby dokonać takich ustawień wykorzystamy *LayoutManager*, który

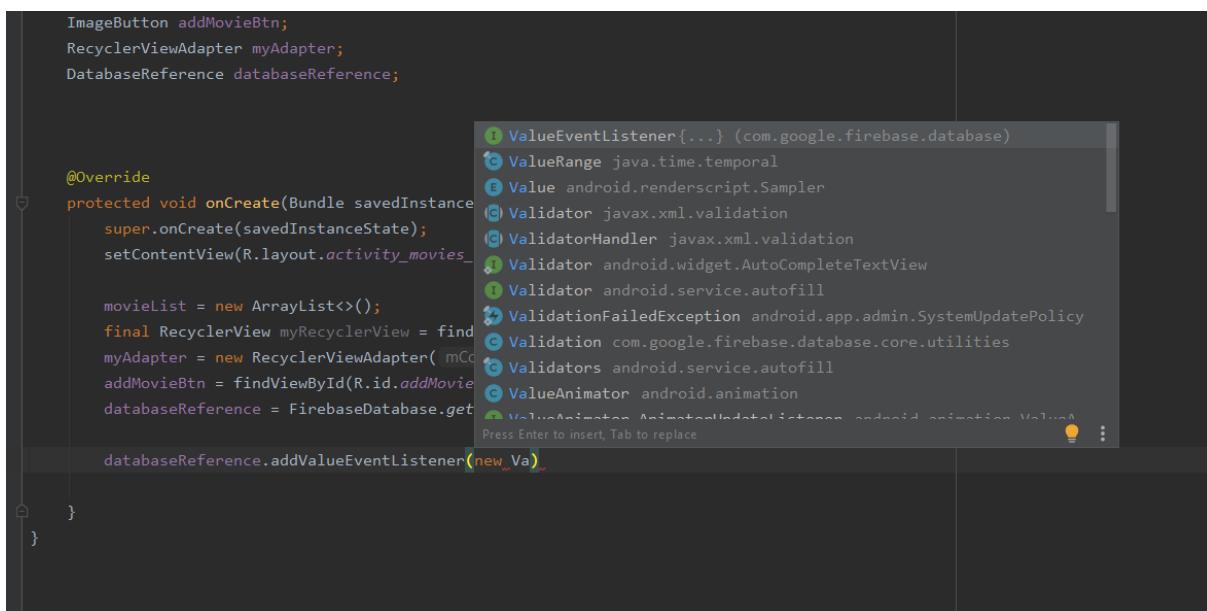
odpowiedzialny jest za pozycjonowanie widoku w układzie *RecyclerView*. Więcej na temat *LayoutManager*:  
<https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView.LayoutManager>

Uzupełnijmy więc nasz *onCreate()* o część przedstawioną poniżej:

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_movies_collection);  
  
    movieList = new ArrayList<>();  
    final RecyclerView myRecyclerView = findViewById(R.id.recyclerviewId);  
    myAdapter = new RecyclerViewAdapter(this, movieList);  
    myRecyclerView.setLayoutManager(new GridLayoutManager(this, 3));  
    addMovieBtn = findViewById(R.id.addMovieImageButtonId);  
    databaseReference = FirebaseDatabase.getInstance().getReference().child("movies");  
  
}
```

Informacje dotyczące filmów przechowywane będą w bazie danych. Ponadto użytkownik aplikacji może dodać nowy film lub usunąć jakiś film z kolekcji (ta funkcjonalność zostanie dodana później). Kolekcja naszych filmów musi być więc cały czas aktualniana zgodnie z zawartością bazy danych, dlatego też koniecznie będzie zastosowanie metody *addValueEventListener()*, gdzie za pomocą dostępnej metody *onDataChange()* będziemy odczytywali zawartość bazy, za każdym razem, gdy dane zostaną zmodyfikowane, dzięki czemu dane w naszej aplikacji będą zawsze aktualne.

W celu dodania *addValueEventListener()* wpiszmy tak jak pokazano na rys. 61 i naciśnijmy klawisz enter. Na rys. 62 przedstawiono co powinniśmy otrzymać. Automatycznie zostały dodane dwie funkcje *onDataChange()* oraz *onCancelled()*.



```
ImageButton addMovieBtn;  
RecyclerViewAdapter myAdapter;  
DatabaseReference databaseReference;  
  
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_movies_collection);  
  
    movieList = new ArrayList<>();  
    final RecyclerView myRecyclerView = findViewById(R.id.recyclerviewId);  
    myAdapter = new RecyclerViewAdapter(this, movieList);  
    myRecyclerView.setLayoutManager(new GridLayoutManager(this, 3));  
    addMovieBtn = findViewById(R.id.addMovieImageButtonId);  
    databaseReference = FirebaseDatabase.getInstance().getReference().child("movies");  
  
    databaseReference.addValueEventListener(new ValueEventListener() {  
        @Override  
        public void onDataChange(DataSnapshot dataSnapshot) {  
            // Data changes have been made  
            // ...  
        }  
        @Override  
        public void onCancelled(DatabaseError error) {  
            // Failed to read value  
            // ...  
        }  
    });  
}
```

Rys. 61. Dodawanie *addValueEventListener()*.

```

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

    movieList = new ArrayList<>();
    final RecyclerView myRecyclerView = findViewById(R.id.recyclerviewId);
    myAdapter = new RecyclerViewAdapter(mContext, movieList);
    myRecyclerView.setLayoutManager(new GridLayoutManager(context, spanCount: 3));
    addMovieBtn = findViewById(R.id.addMovieImageButtonId);
    databaseReference = FirebaseDatabase.getInstance().getReference().child("movies");

    databaseReference.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {

        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {

        }
    });
}

```

Rys. 62. Dodanie `ValueEventListener()` oraz metody utworzone automatycznie.

Uzupełnijmy teraz metodę `onDataChange()`. W tej metodzie odczytane muszą zostać wszystkie dane zapisane w bazie. Jednak, aby nie powielić elementów za każdym wywołaniem, najpierw musimy wyczyścić listę naszych filmów (`movieList`). Następnie tworzymy pętlę przechodzącą po wszystkich rekordach w bazie danych i tworzymy obiekt `Movie`, do tworzonego obiektu przypisujemy unikalny klucz oraz dodajemy go do listy filmów `movieList`. Jeżeli wszystkie rekordy będą zapisane już na naszej liście `movieList` przechodzimy do ustawienia danych na `RecyclerView`. Zawartość metody `onDataChange()` przedstawiona została poniżej:

```

@Override
public void onDataChange(@NonNull DataSnapshot snapshot) {
    movieList.clear(); //preventing duplication of objects

    for(DataSnapshot dataSnapshot1:snapshot.getChildren())
    {
        Movie movie = dataSnapshot1.getValue(Movie.class);
        movie.setKey(dataSnapshot1.getKey()); //get movie id/key from database to assign it
        to created object
        movieList.add(movie);
    }
    myRecyclerView.setAdapter(myAdapter);
}

```

Wypełnijmy także metodę `onCancelled()`, która wywoływana jest w przypadku anulowania odczytu. W tym wypadku wyświetlimy wiadomość typu `Toast` na ekranie użytkownika:

```

@Override
public void onCancelled(@NonNull DatabaseError error) {
    Toast.makeText(MoviesCollectionActivity.this, "Oppss..Something is wrong",
    Toast.LENGTH_SHORT).show();
}

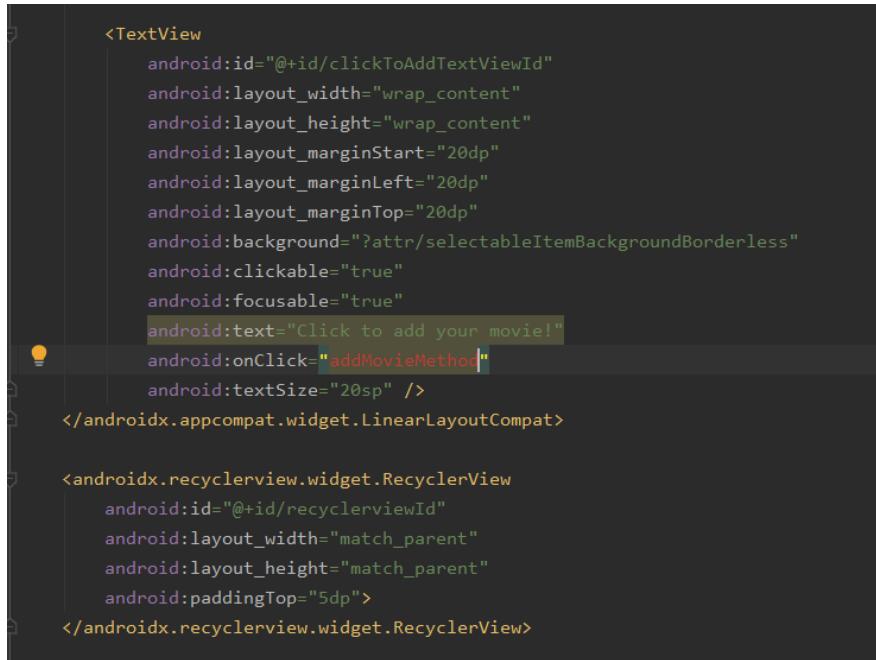
```

Jak już wcześniej wspomniano chcielibyśmy, aby nasza aplikacja posiadała funkcję dodawania nowego filmu. W pliku *layout activity\_movies\_collection.xml* (rys. 60) użyliśmy *ImageButton* oraz *TextView*, gdzie ustawiono napis „Click to add your movie!”. Aby po naciśnięciu tych widoków można było przejść do dodawania nowego filmu należy stworzyć metodę *onClick()*, która po kliknięciu przeniesie nas do nowej aktywności.

Przejdźmy, więc do pliku *activity\_movies\_collection.xml* i zarówno w *TextView* jak i w *ImageButton* dodajmy:

```
| android:onClick="addMovieMethod"
```

Dodanie tej linijki powinno spowodować błąd, jak pokazano na rys. 63.



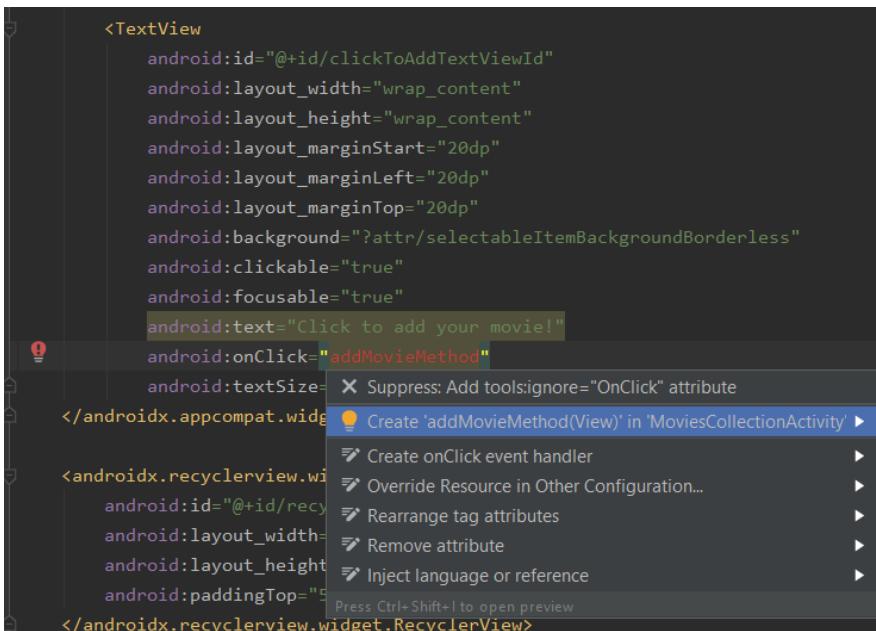
```
<TextView
    android:id="@+id/clickToAddTextViewId"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="20dp"
    android:layout_marginLeft="20dp"
    android:layout_marginTop="20dp"
    android:background="?attr/selectableItemBackgroundBorderless"
    android:clickable="true"
    android:focusable="true"
    android:text="Click to add your movie!"
    android:onClick="addMovieMethod"
    android:textSize="20sp" />
</androidx.appcompat.widget.LinearLayoutCompat>

<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recyclerviewId"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingTop="5dp">
</androidx.recyclerview.widget.RecyclerView>
```

Rys. 63. Błąd spowodowany dodaniem `android:onClick="addMovieMethod"`.

Naciśnijmy na czerwoną żarówkę (rys. 64) i zobaczymy co spowodowało błąd.

Sygnalizacja błędu spowodowana była brakiem odpowiedniej metody. Wybierzmy -> *Create 'addMovieMethod(View)' in 'MoviesCollectionActivity'*.

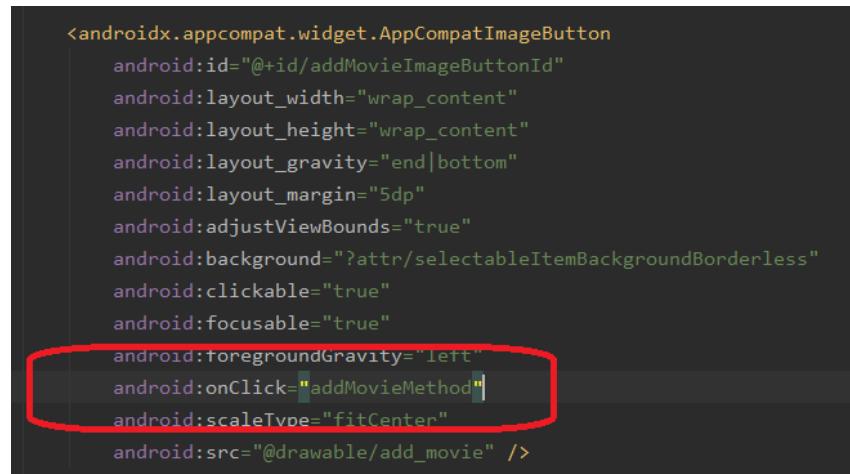


```
<TextView
    android:id="@+id/clickToAddTextViewId"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="20dp"
    android:layout_marginLeft="20dp"
    android:layout_marginTop="20dp"
    android:background="?attr/selectableItemBackgroundBorderless"
    android:clickable="true"
    android:focusable="true"
    android:text="Click to add your movie!"
    android:onClick="addMovieMethod"
    android:textSize="20sp" />
</androidx.appcompat.widget.LinearLayoutCompat>

<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recyclerviewId"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingTop="5dp">
</androidx.recyclerview.widget.RecyclerView>
```

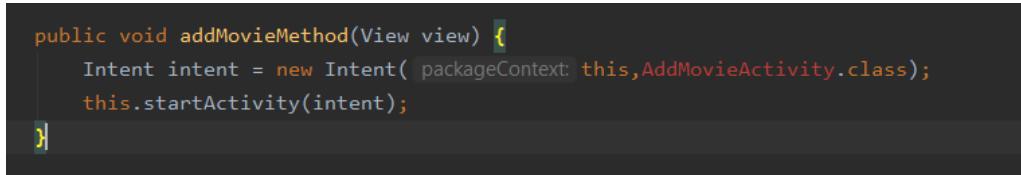
Rys. 64f. Sugerowane sposoby naprawienia sygnalizowanego błędu.

Po dodaniu metody `addMovieMethod()` do `MoviesCollectionActivity.java` nazwa metody powinna zmienić swój kolor na zielony (rys. 65), co oznacza naprawienie błędu.



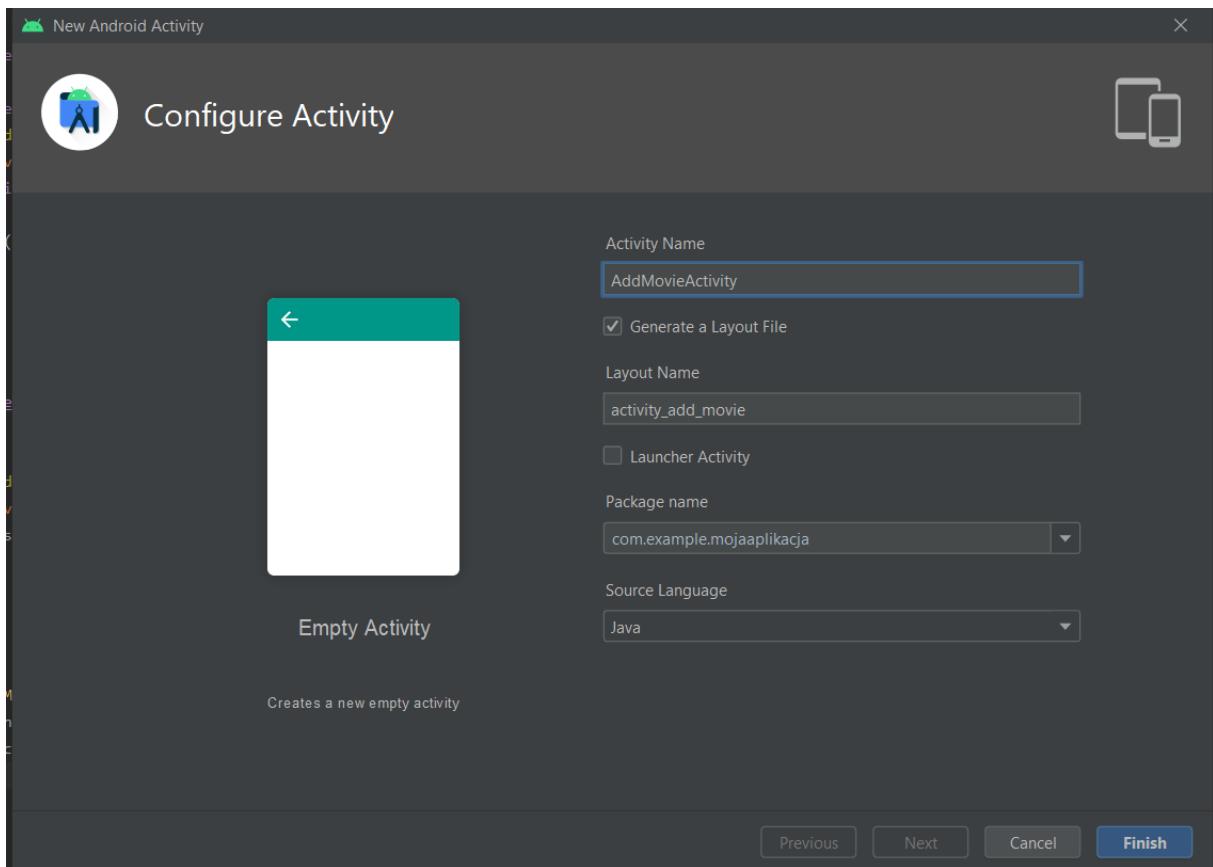
Rys. 65. Po dodaniu metody `addMovieMethod()` do `MoviesCollectionActivity.java`.

Przejdzmy teraz do `MoviesCollectionActivity.java` i uzupełnijmy stworzoną metodę `addMovieMethod()`. Chcemy aby po kliknięciu na `ImageButton` lub na napis „Click to add your movie!” przeniesiono nas do nowej aktywności odpowiedzialnej za dodawanie nowego filmu, gdzie będzie możliwość podania tytułu i innych właściwości filmu. Jak już wcześniej wspominano, do przechodzenia pomiędzy aktywnościami służą intenty, właśnie z tego skorzystamy. Uzupełnijmy więc metodę w następujący sposób (rys. 66):



Rys. 66. Uzupełniona metoda `addMovieMethod()` -> przejście do nowej aktywności.

Po uzupełnieniu metody jak na rys. 66 otrzymaliśmy błąd spowodowany tym, że podaliśmy nazwę aktywności, do której chcemy się przenieść, a taka aktywność nie istnieje. Musimy więc ją stworzyć. Na rys. 67 przedstawione zostało okno konfiguracyjne nowej aktywności. Nazwa aktywności musi być taka sama jako została podana w metodzie `addMovieMethod` (rys. 66). Podczas tworzenia nowej aktywności stworzymy także od razu plik `.xml`, gdzie później zdefiniujemy layout powiązany z tą aktywnością.

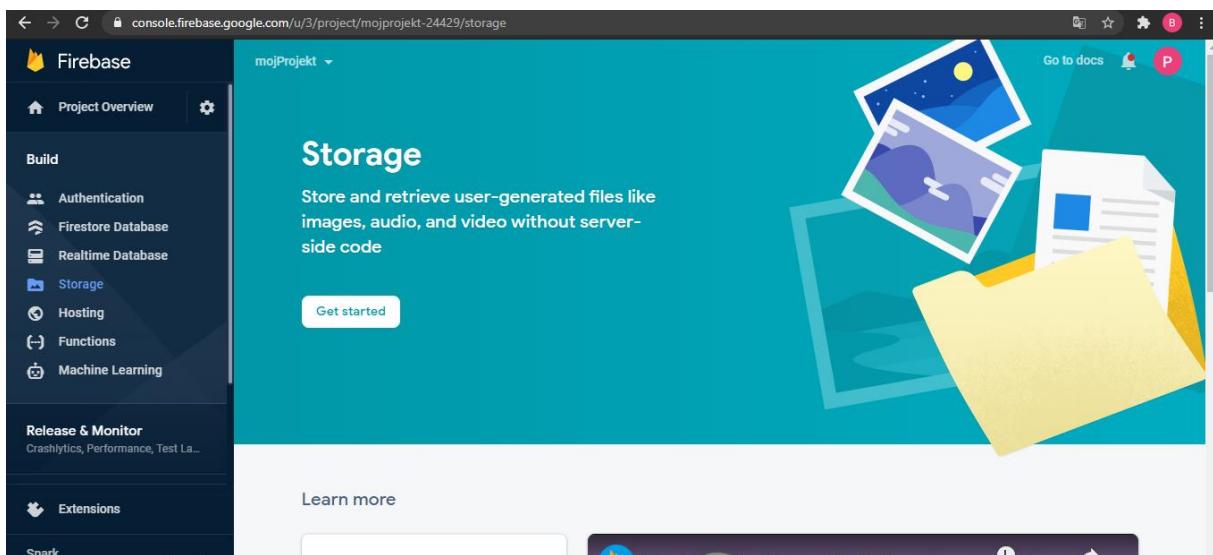


Rys. 67. Tworzenie nowej aktywności: AddMovieActivity oraz layoutu: activity\_add\_movie.

Zanim przejdziemy do definiowania wyglądu dodanej aktywności oraz tworzenia jej logiki dodamy *Firebase Storage* do naszego projektu w celu gromadzenia miniaturek filmów.

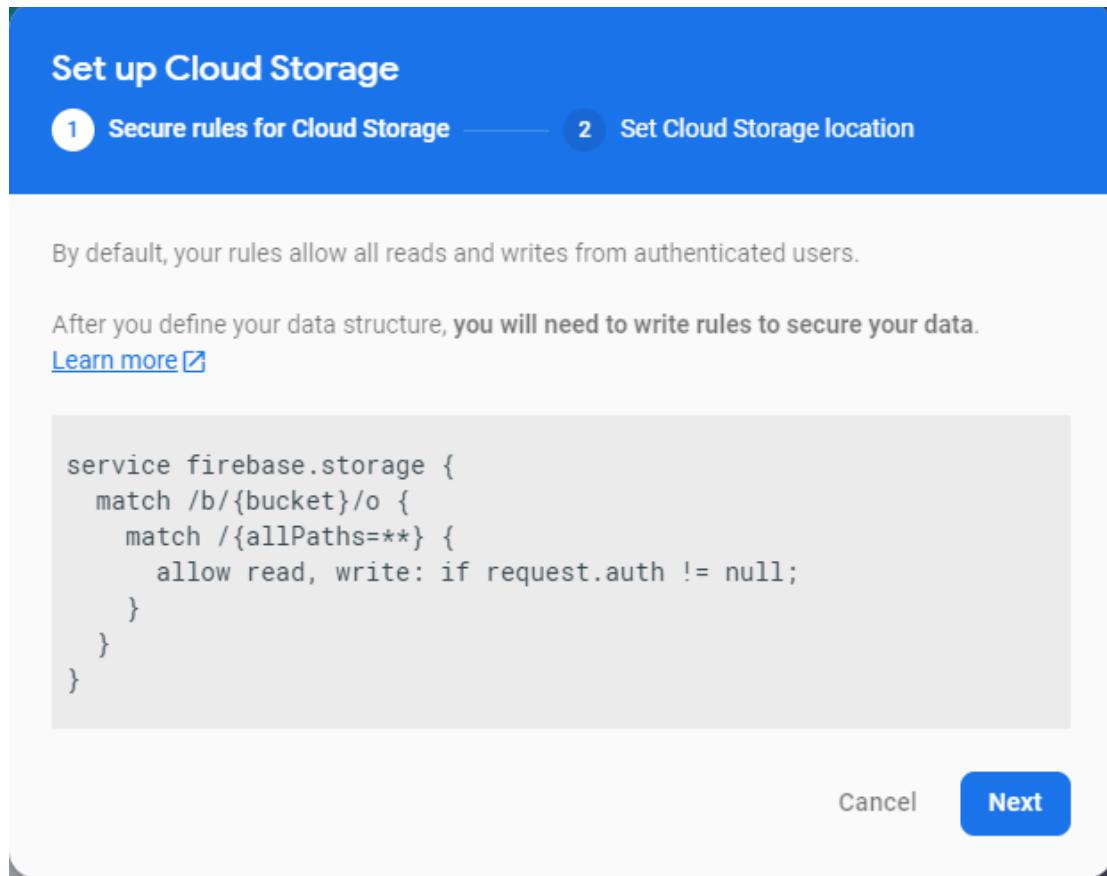
### 3.2.12. Dodanie Firebase Storage do projektu.

W celu podpięcia pod projekt bazy *Storage* należy skorzystać z konsoli *Firebase*, wybrać zakładkę *Storage* oraz kliknąć przycisk *Get Started* (rys.68).



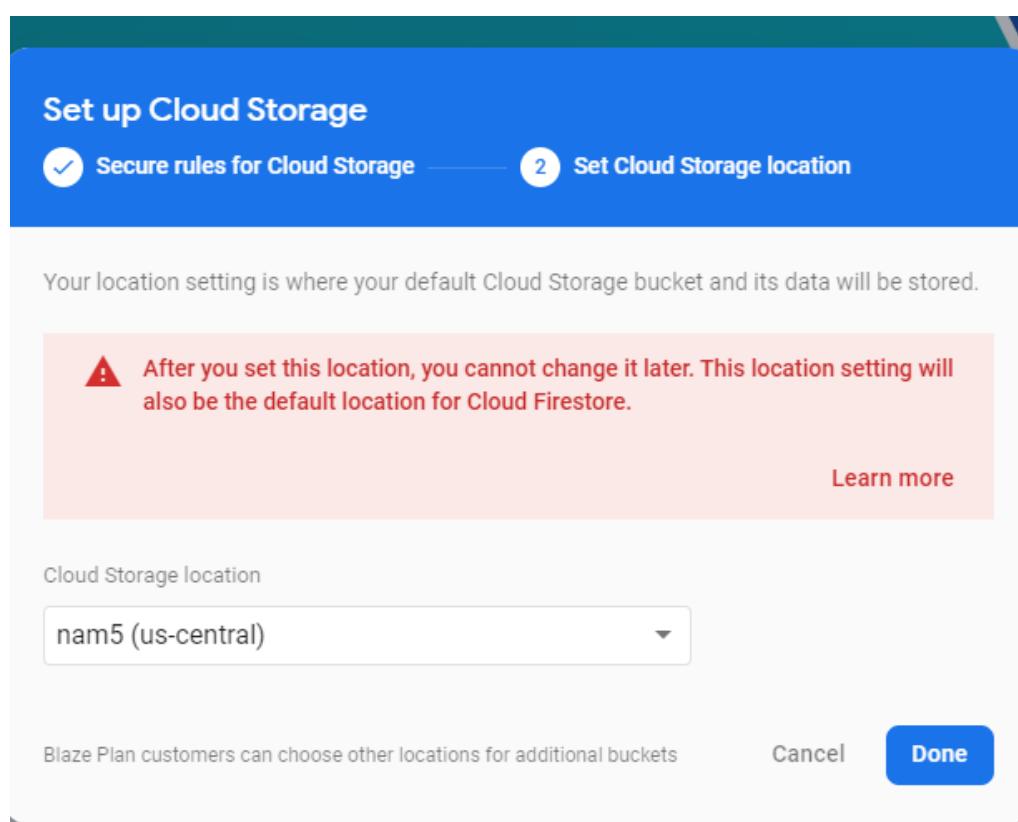
Rys. 68. Zakładka Firebase Storage.

Następnie należy zdefiniować zasady bezpieczeństwa – pozostawimy domyślne:



Rys. 69. Definiowanie zasad Firebase Storage.

Oraz wybranie lokalizacji bazy (również pozostawimy domyślne opcje):



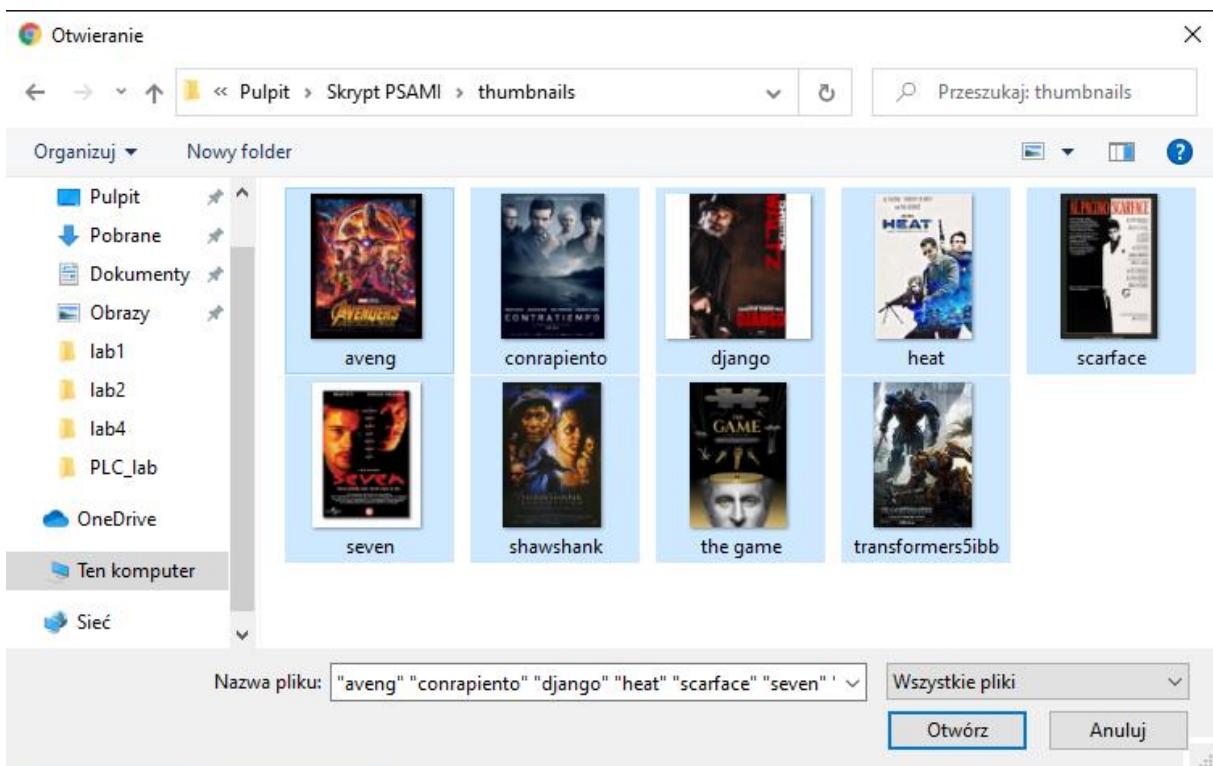
Rys. 70. Konfiguracja Firebase Storage.

Po pomyślnym utworzeniu bazy danych powinniśmy otrzymać następujący rezultat:

The screenshot shows the Firebase Storage interface for a project named 'mojProjekt'. On the left, there's a sidebar with various services like Authentication, Firestore Database, and Storage. The Storage section is selected. The main area is titled 'Storage' and shows a table with columns: Name, Size, Type, and Last modified. A message at the bottom says 'There are no files here yet'. At the top right, there are buttons for 'Upload file' and other options.

Rys. 71. Okno Firebase Storage.

Należałooby uzupełnić dodaną przez nas bazę miniaturkami filmów, które są zawarte w folderze *thumbnails* (opcja *upload file* w oknie *Storage*):



Rys. 72. Miniaturki filmów do dodania.

Po umieszczeniu obrazów w naszej bazie, należy zwrócić uwagę na obszar *Access Token* zawarty w *File location* (rys.73):

The screenshot shows the Firebase Storage interface for a project named 'mojProjekt'. On the left, there's a sidebar with various services like Authentication, Firestore Database, Realtime Database, Storage, Hosting, Functions, Machine Learning, Release & Monitor, and Extensions. The Storage section is selected. The main area displays a list of files in the 'files' folder:

Name	Size	Type	Last modified
aveng.jpg	212.3 KB	image/jpeg	Mar 27, 2021
conrapiento.jpg	73.47 KB	image/jpeg	Mar 27, 2021
django.jpg	101.43 KB	image/jpeg	Mar 27, 2021
heat.jpg	164.03 KB	image/jpeg	Mar 27, 2021
scarface.jpg	195.9 KB	image/jpeg	Mar 27, 2021
seven.jpg	138.55 KB	image/jpeg	Mar 27, 2021
shawshank.jpg	664.57 KB	image/jpeg	Mar 27, 2021
the game.jpg	76.59 KB	image/jpeg	Mar 27, 2021
transformers5ibb.jpg	328.84 KB	image/jpeg	Mar 27, 2021

A detailed view of the 'transformers5ibb.jpg' file is shown on the right, with its properties:

- Name: transformers5ibb.jpg
- Size: 336,736 bytes
- Type: image/jpeg
- Created: Mar 27, 2021, 12:14:37 PM
- Updated: Mar 27, 2021, 12:14:37 PM
- File location: Storage location: gs://mojprojekt-24429.appspot.com/transformers5ibb...
- Access token: [revoke](#) 9aaa0433-5ce0-4584-9261-e2c78ea979b9
- [Create new access token](#)

Rys. 73. Access token danego elementu w bazie.

Ten ciąg znaków wykorzystamy w późniejszej części jako Uri reprezentujące obraz konkretnego filmu.

Pod wykonywane ćwiczenie przygotowana została przykładowa baza, znajdująca się w dołączonym folderze, która nosi nazwę *ExampleDatabase.json*. Jest to plik w formacie *.json* przypominającym składnię tworzenia obiektów w języku *JavaScript*. Import bazy odbywa się w sposób jaki przedstawiony został poniżej:

The screenshot shows the Firebase Realtime Database interface for a project named 'mojProjekt'. The main area displays the database structure under the 'Data' tab:

```
mojprojekt-24429-default-rtdb: null
```

To the right, there's a context menu with the following options:

- Export JSON
- Import JSON
- Show legend
- Disable database
- Create new database  
Available with an upgrade to the Blaze plan

Rys. 74. Import przygotowanej bazy danych.

Jednakże, w polu *thumbnail* w naszej bazie danych widnieje puste miejsce (rys.75), do którego należy skopiować wyżej wymieniony adres Uri. Każdy adres konkretnego obrazu należy skopiować do odpowiedniego filmu.

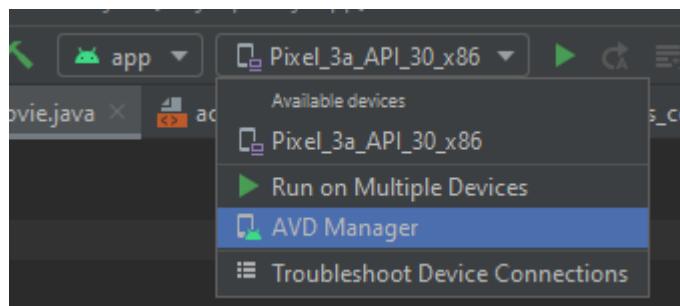
The screenshot shows the Firebase Realtime Database interface. At the top, there's a navigation bar with 'mojProjekt' and other links like 'Go to docs', 'P', and '?'. Below it, tabs for 'Data', 'Rules', 'Backups', and 'Usage' are visible, with 'Data' being the active tab. The URL in the address bar is 'https://mojprojekt-24429-default.firebaseio.com/'. The main area displays a hierarchical database structure under 'mojprojekt-24429-default-rtdb':

- movies
  - 1
    - category: "action sci-fi"
    - description: "High-school student Sam Witwicky buys his first..."
    - rate: 6
    - thumbnail: [Empty input field]
    - title: "Transformers"
  - 2
  - 3
  - 4
  - 6
  - 7

At the bottom left, it says 'Database location: United States (us-central1)'. There are also standard browser controls for zooming and navigating.

Rys. 75. Struktura zimportowanej bazy danych.

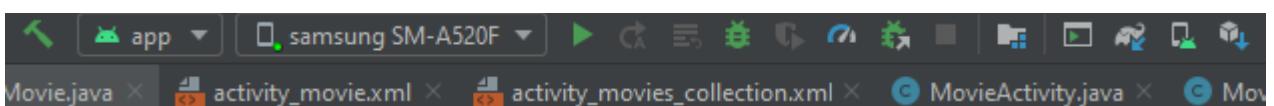
W przypadku poprawnego wykonania wszystkich kroków, można uruchomić stworzoną przez nas aplikację. Istnieje możliwość uruchomienia poprzez emulator w środowisku *AndroidStudio* (rys.76) lub poprzez fizyczne urządzenie.



Rys.76. Emulator urządzenia mobilnego w Android Studio.

W przypadku chęci uruchomienia aplikacji na rzeczywistym smartfonie, należy najpierw odpowiednio go skonfigurować. Wymagane jest, aby uaktywnić w nim tryb programisty oraz aktywować debugowanie USB. W zależności od modelu telefonu może to przebiegać w różny sposób, przykładowe uaktywnienie debugowania oraz cała konfiguracja przedstawiona została na: <https://developer.android.com/studio/run/device>

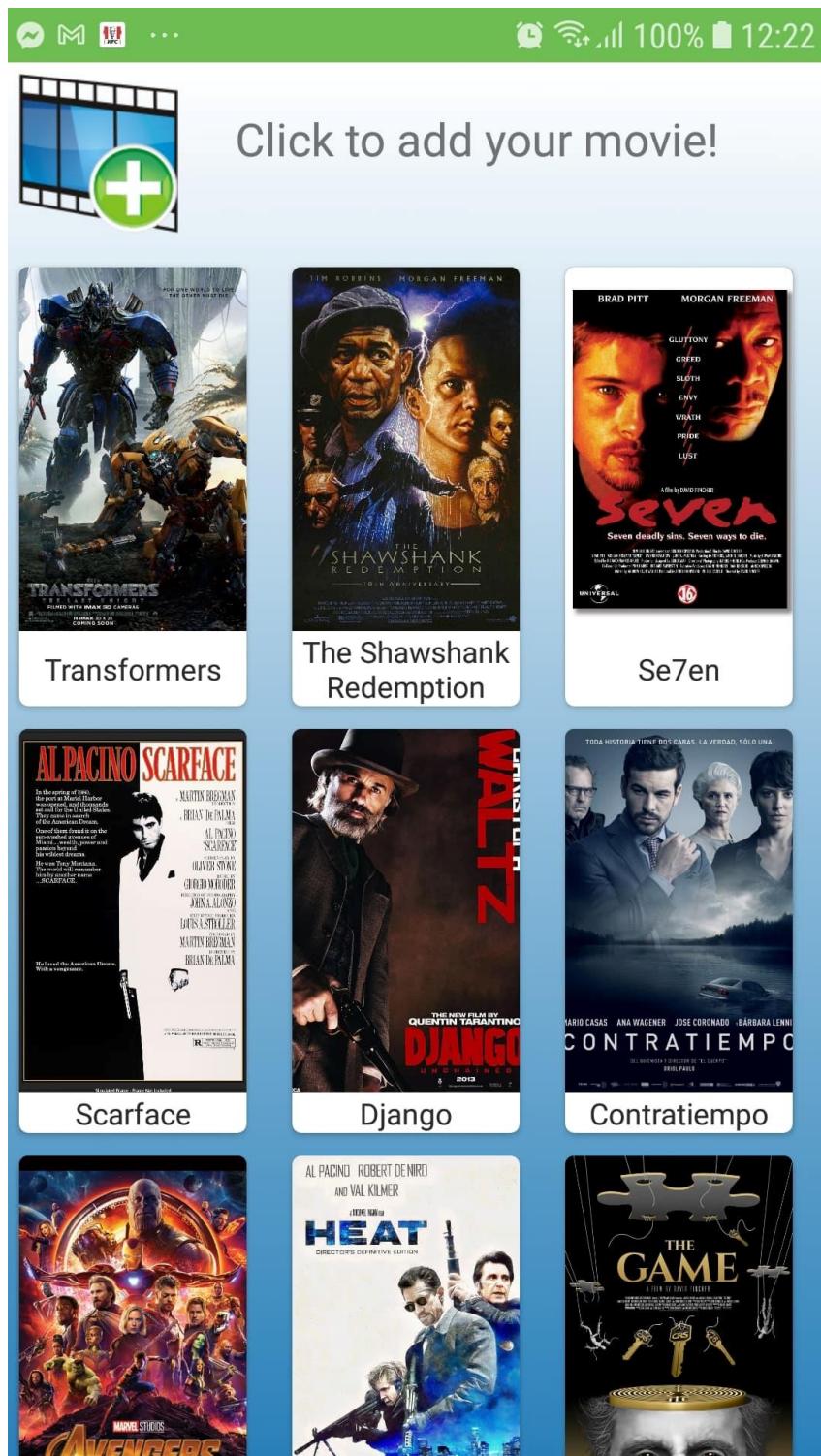
Po pomyślnym włączeniu opcji debugowania, należy podłączyć telefon pod komputer, na którym zainstalowane jest środowisko *Android Studio*, a następnie zezwolić na debugowanie oraz transfer danych. Po krótkiej chwili w miejscu przedstawionym na rys.77 powinna pojawić się nazwa podłączonego przez nas urządzenia:



Rys.77. Wyświetlony model podłączonego telefonu w programie Android Studio.

Aby zainstalować i uruchomić aplikację na wybranym telefonie należy kliknąć na zielony trójkąt po prawej stronie listy z wyświetlonymi podłączonymi urządzeniami, po chwili aplikacja powinna zostać zainstalowana oraz uruchomiona.

Rezultat wykonanej przez nas aplikacji przedstawiony został poniżej (tak powinna wyglądać aplikacja po poprawnym wykonaniu kroków powyżej):



Rys. 78. Widok aktywności z listą filmów w tworzonej aplikacji.

Jednakże nadal brakuje implementacji kilku zdarzeń – np. wyświetlania odpowiednich informacji po kliknięciu na konkretny film.

### 3.2.13. Dodawanie funkcjonalności do wyświetlania informacji poszczególnych filmów.

Aby umożliwić wyświetlanie odpowiednich danych po kliknięciu na film, należy przejść do pliku *MovieActivity.java* oraz zainicjować poniższe obiekty:

```
TextView titleTextView, descriptionTextView, categoryTextView, rateTextView;
ImageView movieImageView;
DatabaseReference referenceDatabase;
RatingBar ratingBar;
```

W tym pliku, należy wykonać kilka kluczowych zadań:

- przypisać do poszczególnych obiektów odpowiadające im widoki,
- odebrać dane z przesłanej intencji,
- użyć „nasłuchiwacza” zmiany stanu *RatingBar*,
- wysyłać do bazy danych zaktualizowaną ocenę przez użytkownika,
- ustawiać poszczególne widoki danymi otrzymanymi z intencji.

Aby zrealizować powyższe funkcję, należy zastosować kod, który przedstawiono poniżej:

```
package com.example.mojaaplikacja;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.RatingBar;
import android.widget.TextView;

import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.squareup.picasso.Picasso;

public class MovieActivity extends AppCompatActivity {

    TextView titleTextView, descriptionTextView, categoryTextView, rateTextView;
    ImageView movieImageView;
    DatabaseReference referenceDatabase;
    RatingBar ratingBar;

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

        titleTextView = findViewById(R.id.titleTextViewId);
        descriptionTextView = findViewById(R.id.descriptionTextViewId);
        categoryTextView = findViewById(R.id.categoryTextViewId);
        movieImageView = findViewById(R.id.thumbnailInActivityImageViewId);
        ratingBar = findViewById(R.id.ratingBarId);
        rateTextView = findViewById(R.id.userRateTextViewId);

        Intent receivedIntent = getIntent();
        String title = receivedIntent.getExtras().getString("MovieTitle");
        String description = receivedIntent.getExtras().getString("Description");
        String category = receivedIntent.getExtras().getString("Category");
        String image = receivedIntent.getExtras().getString("Thumbnail");
        String key = receivedIntent.getExtras().getString("Key");
        int rate = receivedIntent.getExtras().getInt("Rate");

        ratingBar.setOnRatingBarChangeListener(new RatingBar.OnRatingBarChangeListener() {
```

```

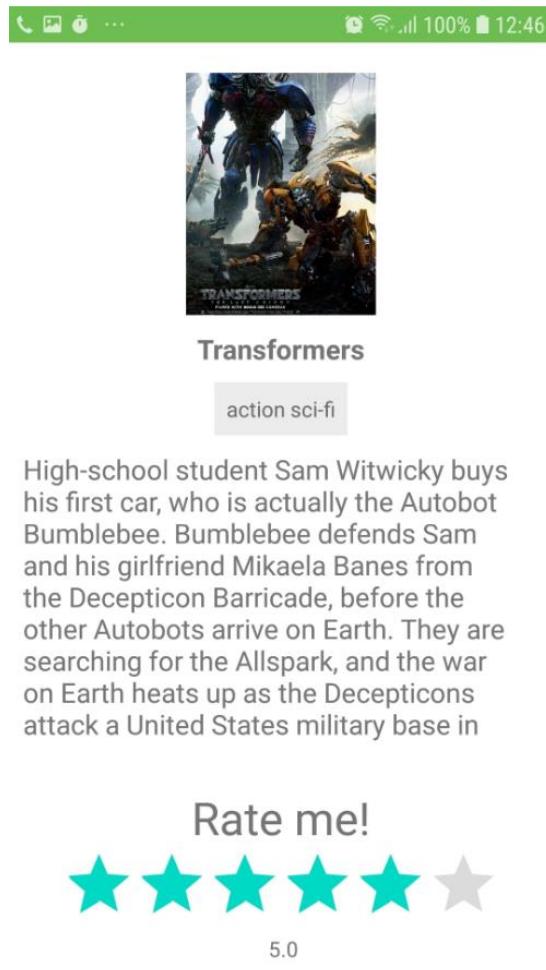
@Override
public void onRatingChanged(RatingBar ratingBar, float rating, boolean fromUser)
{
    referenceDatabase =
        FirebaseDatabase.getInstance().getReference().child("movies");
    referenceDatabase.child(key).child("rate").setValue(rating);
    rateTextView.setText(" " + rating);
}
});
titleTextView.setText(title);
descriptionTextView.setText(description);
categoryTextView.setText(category);
Picasso.get().load(image).into(movieImageView);
ratingBar.setRating(rate);
}
}

```

Jak widać na kodzie przedstawionym powyżej, otrzymaliśmy dane z Intencji poprzez metodę `getIntent()`. Następnie konkretne dane przypisywane były do zmiennych za pomocą metody `getExtras().GetString()` lub `getExtras().getInt()`.

W aktywności zastosowano *listener* do *RatingBar* oraz metodę `onRatingChanged()`, która zostaje wykonywana każdorazowo, gdy zmieni się wartość oceny na naszym *RatingBar*. W tej metodzie, tworzymy instancję naszej bazy danych. Dodatkowo w miejscu `rate` dla konkretnego filmu przypisujemy nową wartość. Uzupełnianie widoków odbywa się poprzez przypisanie odebranych danych z intencji.

Po pomyślnym zaimplementowaniu kodu, można sprawdzić działanie aplikacji. Po naciśnięciu na jakikolwiek film powinna otwierać się nowa aktywność, która może wyglądać w ten sposób:



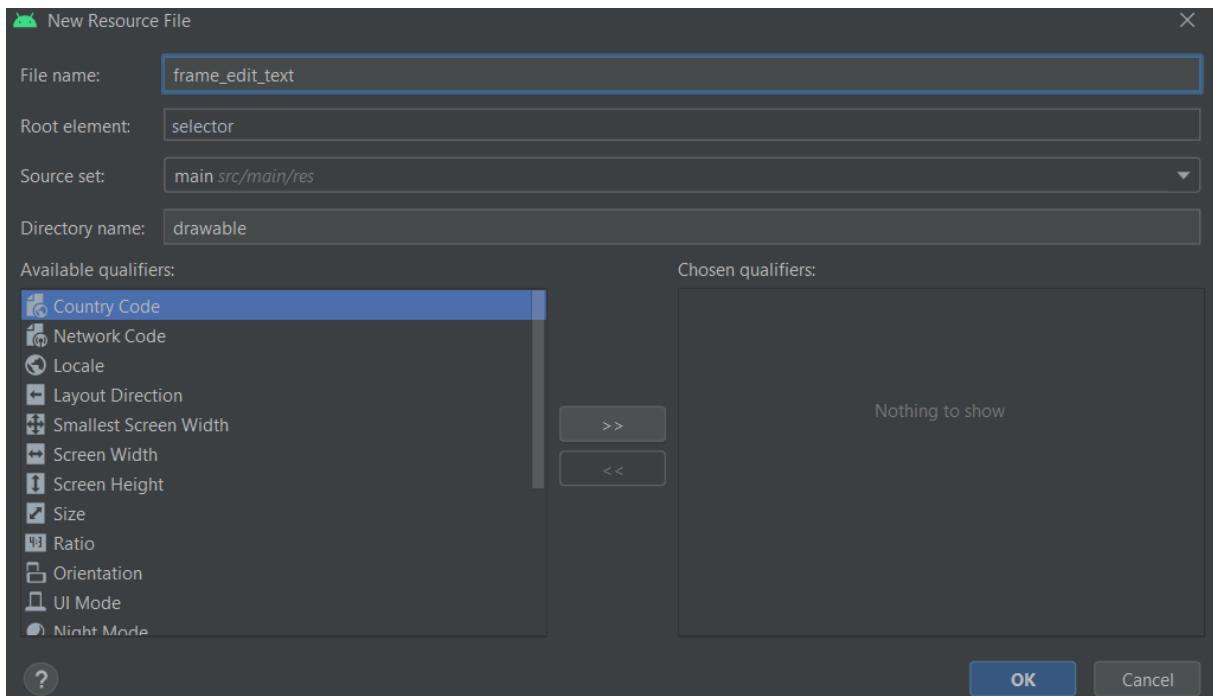
Rys.79. Widok ekranu dotyczącego filmu w tworzonej aplikacji.

Kolejną rzeczą jest funkcjonalność dodawania nowych filmów jak zostało to wspomniane powyżej.

### 3.2.14. Dodawanie aktywności związanej z tworzeniem nowych filmów.

Przed rozpoczęciem tworzenia wyglądu tej aktywności, przystąpmy do utworzenia nowego pliku `frame_edit_text.xml` (rys.80). Będzie to plik, w którym zdefiniujemy kontury tworzonych przez nas pól tekstowych.

W tym celu należy stworzyć plik jak poniżej:



Rys.80. Tworzenie pliku `frame_edit_text.xml`.

Kod, który należy zastosować dla tego pliku przedstawiony poniżej:

```
<?xml version="1.0" encoding="utf-8"?>
<shape
    xmlns:android="http://schemas.android.com/apk/res/android">
    <stroke
        android:width="2dp"
        android:color="#2f333f"
        >
    </stroke>

    <corners
        android:radius="20dp">
    </corners>

    <padding
        android:top="15dp"
        android:bottom="15dp"
        android:left="15dp"
        android:right="15dp">
    </padding>
</shape>
```

- Atrybut `stroke` odnosi się do linii obrysu kształtu – ustawimy jej szerokość na 2dp i kolor jak wyżej.,
- Atrybut `corners` tworzy zaokrąglone rogi kształtu – ustawimy jego wartość na 20dp,
- Atrybut `padding` jest to wypełnienie jakie ma zostać zastosowane do zawierającego elementu widoku.

Po stworzeniu pliku możemy zająć się edytowaniem *activity\_add\_movie.xml*. W tym celu należy stworzyć kod jak poniżej:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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:background="@drawable/gradient_background"
    android:orientation="vertical"
    tools:context=".AddMovieActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:fontFamily="casual"
        android:padding="10dp"
        android:text="Add Movie!"
        android:textColor="#000"
        android:textSize="30sp"
        android:textStyle="bold" />

    <ImageView
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_gravity="center"
        android:scaleType="fitCenter"
        android:src="@drawable/popcorn" />

    <EditText
        android:id="@+id/addTitleEditTextId"
        android:layout_width="match_parent"
        android:layout_height="70dp"
        android:layout_margin="10dp"
        android:background="@drawable/frame_edit_text"
        android:hint="Title of movie..."
        android:textSize="15sp" />

    <EditText
        android:id="@+id/addCategoryEditTextId"
        android:layout_width="match_parent"
        android:layout_height="70dp"
        android:layout_margin="10dp"
        android:background="@drawable/frame_edit_text"
        android:hint="Category of movie..."
        android:textSize="15sp" />

    <EditText
        android:id="@+id/addDescriptionEditTextId"
        android:layout_width="match_parent"
        android:layout_height="150dp"
        android:layout_margin="10dp"
        android:autofillHints=""
        android:background="@drawable/frame_edit_text"
        android:gravity="left"
        android:hint="Description of movie..."
        android:overScrollMode="always"
        android:scrollbarStyle="insideInset"
        android:scrollbars="vertical" />

    <Button
```

```

        android:id="@+id/addButtonId"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:layout_marginEnd="10dp"
        android:layout_marginRight="10dp"
        android:text="add movie"
        app:backgroundTint="#39595E7C" />
    </LinearLayout>

```

Layout składa się z następujących elementów umieszczonych w *LinearLayout*:

- *TextView* – w którym wyświetlimy tekst „Add movie”,
- *Imageview* – będzie przechowywał obraz, który został dołączony do plików wraz z instrukcją,
- *Trzy pola EditText* – w których będziemy wpisywali: tytuł filmu, kategorię filmu oraz opis filmu. Dodatkowo w widoku przewidzianym na wpisywanie opisu filmu zastosowano atrybuty umożliwiające „scrollowanie” w przypadku dłuższych opisów. Do tego celu wykorzystano parametry:

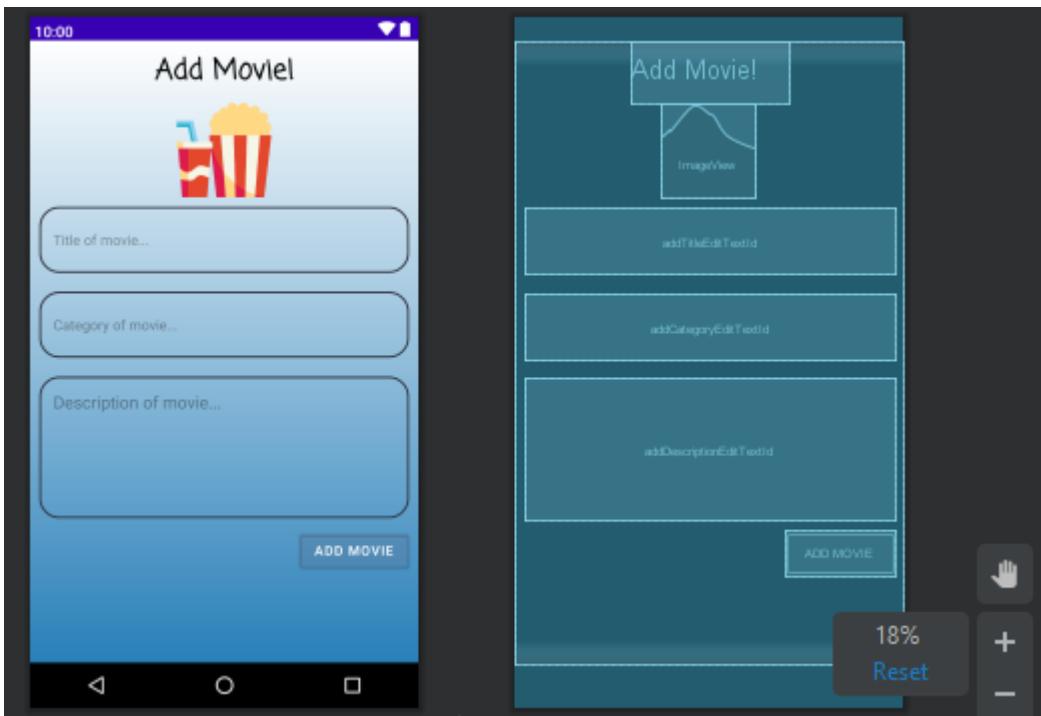
```

        android:overScrollMode="always"
        android:scrollbarStyle="insideInset"
        android:scrollbars="vertical"

```

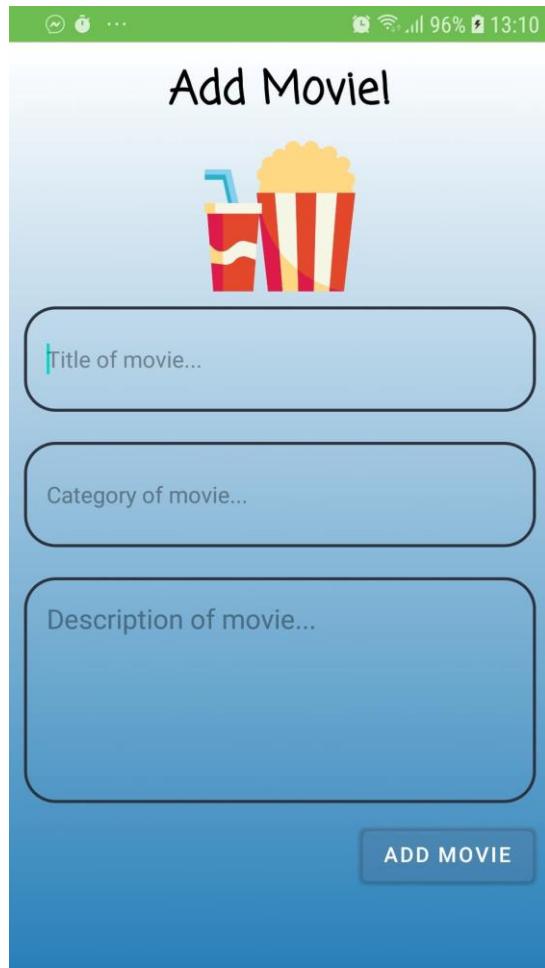
- *Button* – po jego naciśnięciu film ma zostać dodany do bazy danych.

Utworzony layout powinien wyglądać w następujący sposób w programie:



Rys.81. wygląd layout'u activity\_add\_movie.xml w programie Android Studio.

Natomiast widok z pozycji telefonu powinien wyglądać jak przedstawiono na rysunku poniżej:



Rys.82. Wygląd aktywności związanej z dodawaniem filmu na ekranie smartfona.

W tym miejscu zajmiemy się tworzeniem logiki aktywności dodawania filmu. W tym celu należy stworzyć pola w pliku *AddMovieActivity.java*:

```
Movie movie;
Button addButon;
EditText titleEditText, descriptionEditText, categoryEditText;
DatabaseReference referenceDatabase;
```

W metodzie *onCreate()* natomiast należy zatrzymać poniższy kod:

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

    addButon = findViewById(R.id.addButtonId);
    titleEditText= findViewById(R.id.addTitleEditTextId);
    descriptionEditText = findViewById(R.id.addDescriptionEditTextId);
    categoryEditText = findViewById(R.id.addCategoryEditTextId);
    referenceDatabase = FirebaseDatabase.getInstance().getReference().child("movies");
    movie = new Movie();

    addButon.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            movie.setCategory(categoryEditText.getText().toString().trim());
            movie.setDescription(descriptionEditText.getText().toString().trim());
            movie.setTitle(titleEditText.getText().toString().trim());
        }
    });
}
```

```

    //-----set default thumbnail-----
-- 
// you should copy URI from default picture in storage database
movie.setThumbnail("https://firebasestorage.googleapis.com/v0/b/mojprojekt-
24429.appspot.com/o/krists-luhanders-AtPWhYNDJnM-unsplash.jpg?alt=media&token=1a37c640-01dd-
497c-849a-901906d808b6");
//-----

referenceDatabase.child(UUID.randomUUID().toString()).setValue(movie);
Toast.makeText(getApplicationContext(),"Video was added
successfully!",Toast.LENGTH_SHORT).show();
}
});

}

```

W metodzie tej poza przypisaniem widoków do obiektów, stworzeniem referencji do bazy danych oraz stworzenia obiektu klasy *movie* dodatkowo wykorzystujemy *listener* dla przycisku *addButton*.

Za pomocą metody *onClick()* wykonywane jest przypisanie do obiektu *movie* następujących rzeczy:

- tytuł,
- kategoria,
- opis.

Dodatkowo ustawiana jest domyślna miniaturka umieszczona w bazie danych *FirebaseStorage*.

Za pomocą *UUID.randomUUID()* tworzony jest unikalny klucz, mający zapobiegać duplikowania nazw rekordów w bazie danych. Gdy pomyślnie zostanie dodany film do bazy, wyświetlany jest *Toast Message*.

### 3.2.15. Usuwanie elementu (filmu).

Poza dodawaniem nowych filmów, dokonamy implementacji funkcjonalności dotyczącej usuwania istniejących filmów z naszej aplikacji. Usuwanie filmów odbywać się będzie po dłuższym przyciśnięciu konkretnego filmu z listy. W tym celu należy zastosować na naszym *holder.cardview* metodę *setOnLongClickListener()*. Implementacja kodu została przedstawiona poniżej:

```

holder.cardView.setOnLongClickListener(new View.OnLongClickListener() {
    @Override
    public boolean onLongClick(View v) {
        new AlertDialog.Builder(mContext).setTitle("Delete")
            .setMessage("Are you sure to delete this
movie?").setPositiveButton(android.R.string.ok, new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialogInterface, int i) {
                    String movieID = movies.get(position).getKey();

                    FirebaseDatabase.getInstance().getReference("movies").child(movieID).removeValue();
                }
            })
            .setNegativeButton(android.R.string.cancel,
null).setIcon(android.R.drawable.ic_dialog_alert).show();
        return true;
    }
});
```

Metoda `setOnLongClickListener()` wywoływana jest w momencie, gdy użytkownik przytrzyma dłużej dany obiekt. W metodzie `onLongClick()` zostało zaimplementowane okno dialogowe, w którym napisaliśmy zapytanie czy użytkownik jest zdecydowany usunąć film. Więcej na temat okien dialogowych:

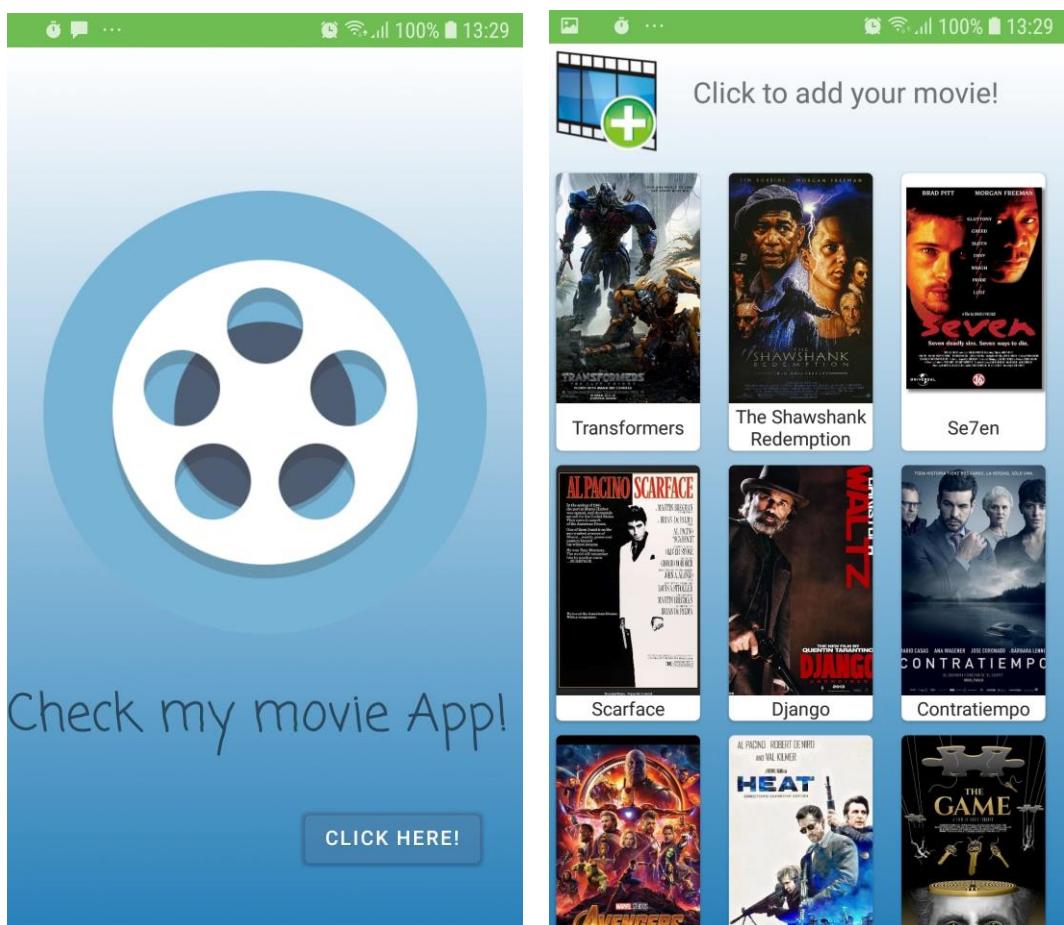
<https://developer.android.com/guide/topics/ui/dialogs>

W momencie wybrania opcji `ok` pobierane jest `id` rekordu w bazie danych i film o tym `id` jest z niej usuwany. W momencie kliknięcia `cancel` okno dialogowe jest zamknięte.

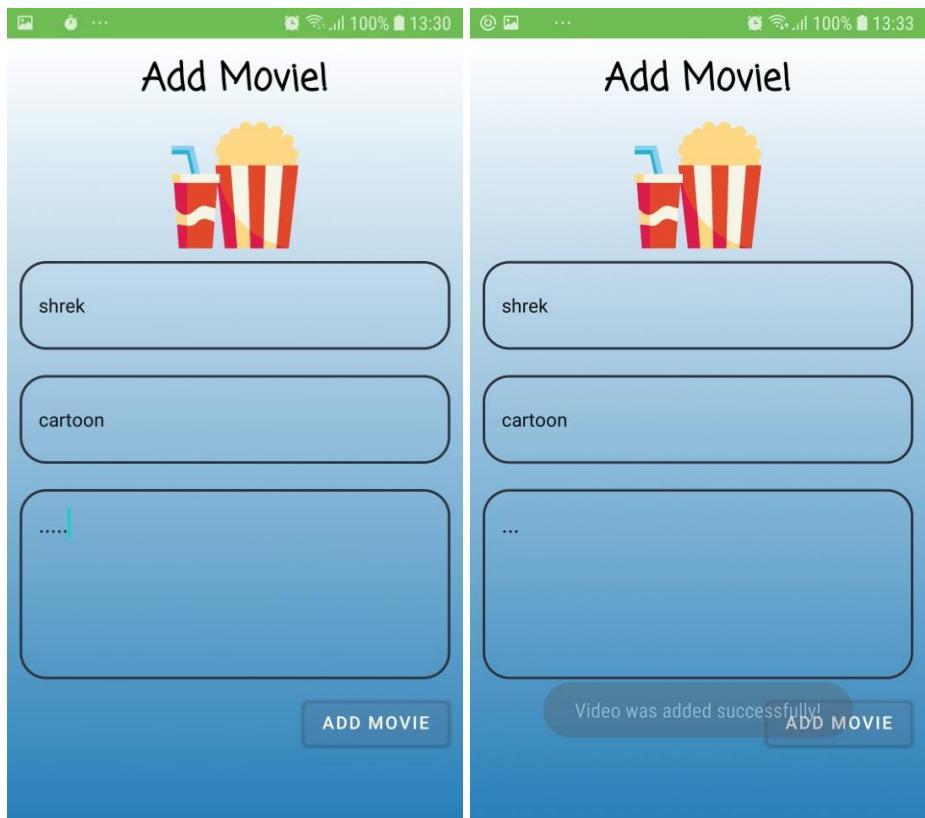
W aplikacji należałyby jeszcze zdefiniować wygląd poszczególnych aktywności dla orientacji ekranu poziomej. Jednakże w tym skrypcie nie będziemy się już tym zajmować. Można zastosować opcję, że aplikacja działać będzie tylko w trybie pionowym co definiowane jest w pliku `AndroidManifest.xml`. Dla poszczególnych aktywności należy zastosować parametr: `android:screenOrientation="portrait"`.

W tym miejscu nasza aplikacja może zostać uznana za skończoną. Tworzenie praktycznej aplikacji w tym poradniku miało pomóc w zrozumieniu istotnych aspektów podczas projektowania aplikacji w środowisku Android Studio w języku Java.

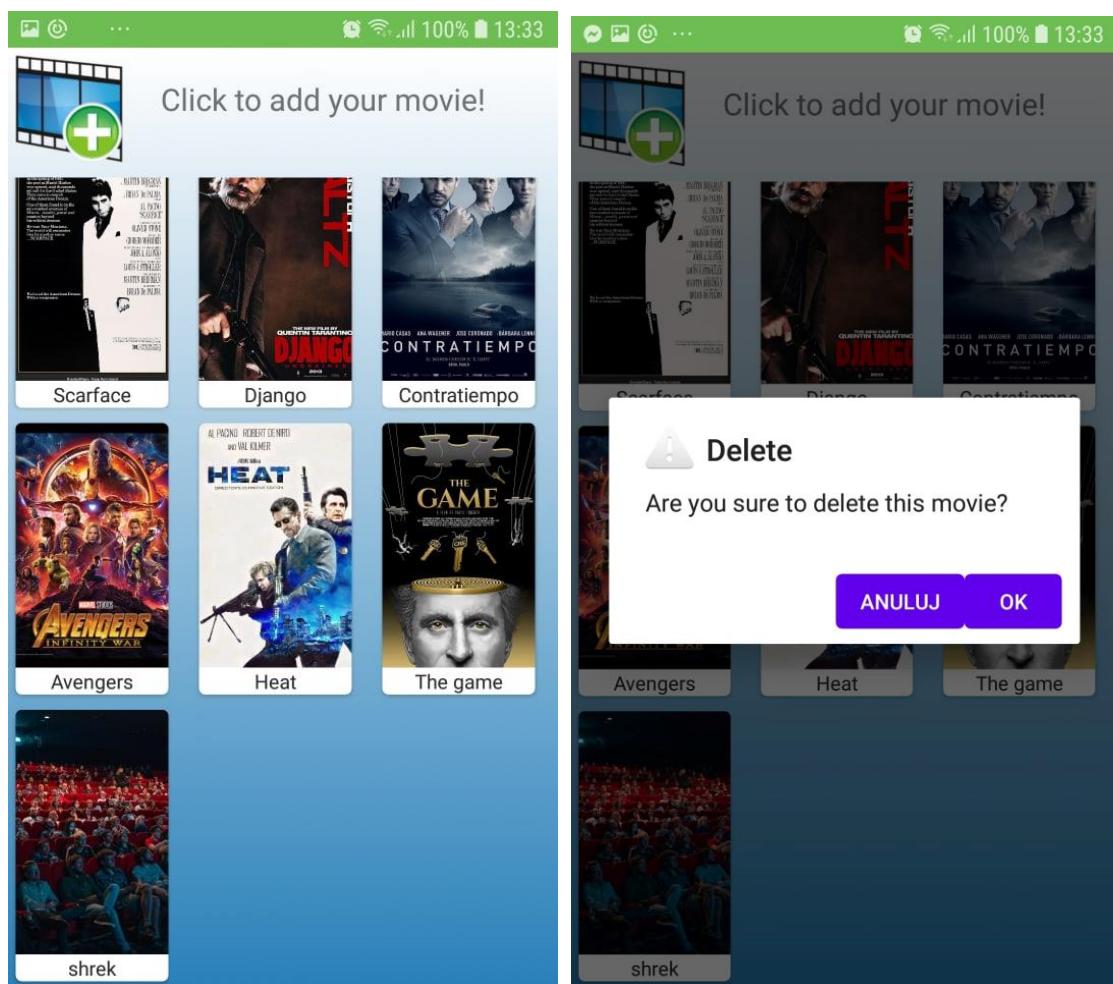
Aplikacja końcowa powinna wyglądać w sposób jaki zostało to przedstawione poniżej:



Rys.83. Ekran główny aplikacji (po lewej stronie) oraz widok menu głównego aplikacji (po prawej stronie).



Rys.85. Ekrany aktywności odpowiedzialne za dodawanie nowych filmów. Po prawej stronie sytuacja po pomyślnym dodaniu filmu.



Rys.85. Nowo dodany film „shrek” (po lewej stronie) oraz okno dialogowe do usuwania filmu (po stronie prawej)

Baza danych po dodaniu konkretnego filmu:

The screenshot shows the Firebase Realtime Database interface. At the top, there are tabs for Data, Rules, Backups, and Usage. The Data tab is selected. Below the tabs, the URL is https://mojprojekt-24429-default-rtdb.firebaseio.com/. The main area displays a tree structure under the 'movies' node. The 'movies' node contains ten child nodes labeled 1 through 10. Node 10 has a child node 'e96f404e-c7cd-4e94-9de8-820699111890' which represents the newly added movie 'Shrek'. This node contains the following fields: category: "cartoon", description: "...", rate: 0, thumbnail: "https://fbsstorage.googleapis.com/v0/b/moj.../", and title: "shrek". A tooltip at the bottom left indicates the database location is United States (us-central1).

Rys.86. Nowo dodany film „shrek” – okno bazy danych.