

# Kotlin & Android w Android Studio Giraffe (2022.3.1)

## Multimedia

Tym razem zajmiemy się multimediami w naszym telefonie. Utworzymy aplikację, która demonstrować będzie sposób obsługi aparatu fotograficznego, zajmiemy się też tworzeniem galerii wyświetlającej obrazy zawarte w naszym telefonie. Ponadto zbudujemy odtwarzacz plików dźwiękowych i filmów wideo.

### Tworzenie aplikacji wykorzystującej multimedia

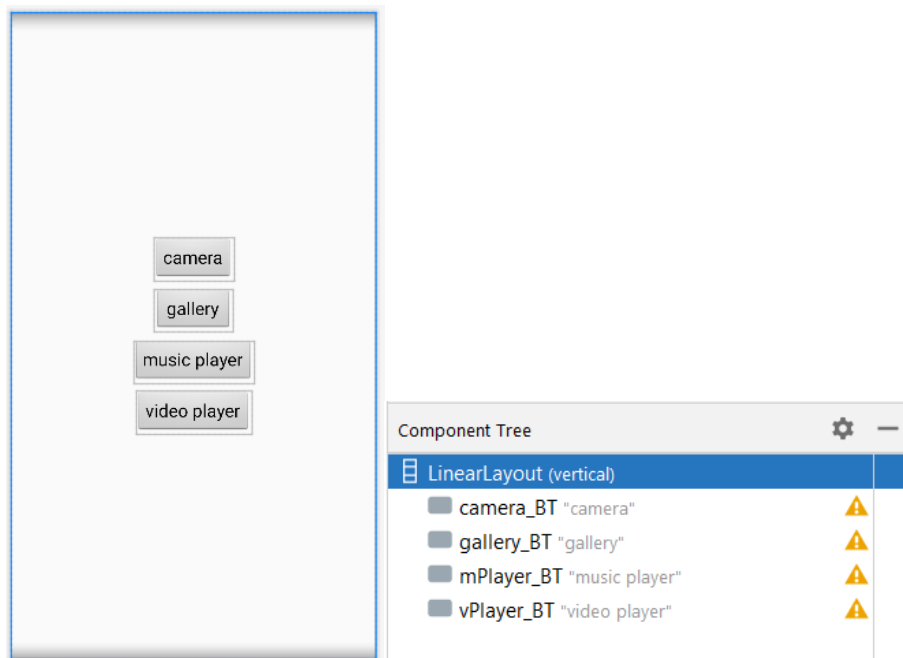
Nasza aplikacja bazować będzie na kilku aktywnościach i kilku layoutach. Zaczynamy od utworzenia głównego layoutu programu.

#### ZADANIE

Utwórz nowy projekt Android Studio o nazwie *MultimediaApp*, zawierający *EmptyActivity*. W pliku layoutu głównej aktywności użyj komponentu *LinearLayout(vertical)*. Ustaw dla layoutu parametr *gravity* na wartość *center*. Dodaj 4 przyciski *Button* zawierające następujące teksty:

- Camera (id = camera\_BT);
- Gallery (id = gallery\_BT);
- Music Player (id = mPlayer\_BT);
- Video Player (id = vPlayer\_BT).

Dla każdego przycisku ustaw: *layout\_margin = 5dp*, *layout\_width = wrap\_content*, *textSize = 20sp*.



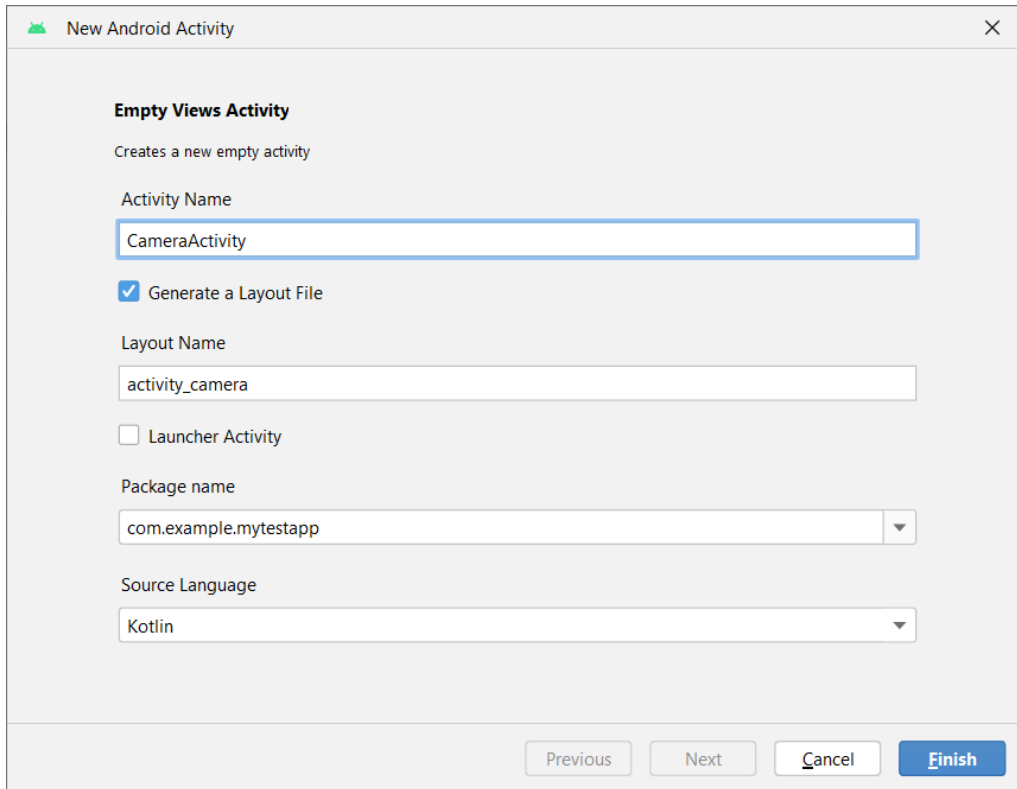
Przyciski posłużą nam do wywoływania aktywności odpowiedzialnych za realizację danych funkcjonalności naszej aplikacji.

Utwórzmy teraz pliki aktywności i layoutów dla każdej z funkcjonalności. Kod tych plików będziemy modyfikować nieco później.

### **ZADANIE**

Utwórz w naszym projekcie 4 nowe pliki aktywności *EmptyActivity* o następujących nazwach

- *CameraActivity* (z plikiem layoutu *activity\_camera*);
- *GalleryActivity* (z plikiem layoutu *activity\_gallery*);
- *MusicActivity* (z plikiem layoutu *activity\_music*);
- *VideoActivity* (z plikiem layoutu *activity\_video*).



New Android Activity

**Empty Views Activity**  
Creates a new empty activity

Activity Name  
CameraActivity

☒ Generate a Layout File

Layout Name  
activity\_camera

☐ Launcher Activity

Package name  
com.example.mytestapp

Source Language  
Kotlin

Previous Next Cancel Finish

Przejdźmy do oprogramowania przycisków głównego layoutu aby uruchamiały wyżej utworzone aktywności.

### **ZADANIE**

Zaimplementuj kod obsługi przycisków głównej aktywności.

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.layout)

    findViewById<Button>(R.id.camera_BT).setOnClickListener { it: View!
        val intentCamera = Intent(applicationContext, CameraActivity::class.java)
        startActivity(intentCamera)
    }

    findViewById<Button>(R.id.gallery_BT).setOnClickListener { it: View!
        val intentGallery = Intent(applicationContext, GalleryActivity::class.java)
        startActivity(intentGallery)
    }

    findViewById<Button>(R.id.mPlayer_BT).setOnClickListener { it: View!
        val intentMusic = Intent(applicationContext, MusicActivity::class.java)
        startActivity(intentMusic)
    }

    findViewById<Button>(R.id.vPlayer_BT).setOnClickListener { it: View!
        val intentVideo = Intent(applicationContext, VideoActivity::class.java)
        startActivity(intentVideo)
    }
}

```

Ponieważ w naszej aplikacji będziemy potrzebować dostępu do aparatu oraz do plików dodajmy w pliku manifestu odpowiednie uprawnienia.

### **ZADANIE**

Dodaj w pliku *AndroidManifest* poniższy kod uprawnień naszej aplikacji:

```

<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera" android:required="true" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

```

Ponadto wewnątrz znacznika *<application* umieść kod:

```

<application
    android:preserveLegacyExternalStorage="true"
    android:requestLegacyExternalStorage="true"

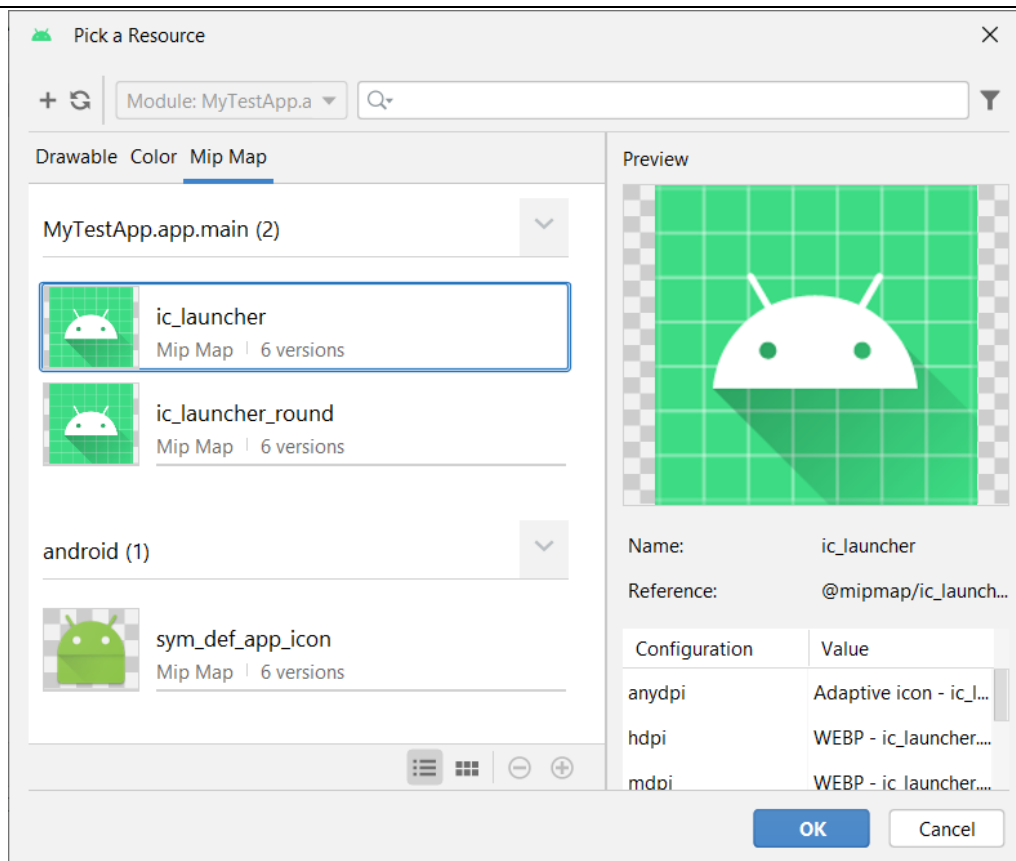
```

### **Obsługa aparatu**

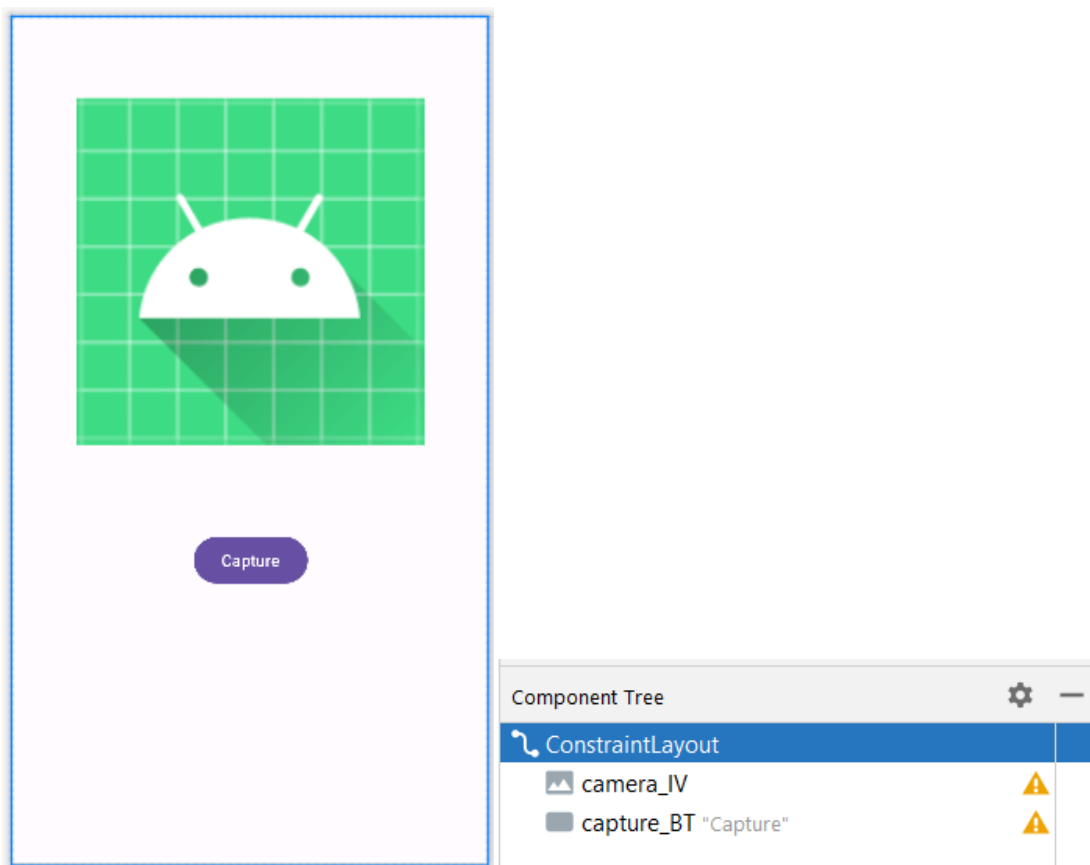
W tej sekcji zajmiemy się obsługą aparatu. Przygotujmy layout dla tej aktywności.

### **ZADANIE**

Dodaj do layoutu *activity\_camera* komponent *ImageView* i ustaw w nim obraz *ic\_launcher*.



Dodaj również przycisk *Button* zawierający tekst *Capture*. Ustaw identyfikatory komponentów: dla *ImageView* *id=camera\_IV*, dla *Button* *id=capture\_BT*. Dopasuj rozmiar komponentów i ich położenie.



Layout aktywności obsługi aparatu jest gotowy. Teraz trzeba zaimplementować obsługę przycisku *Capture*. Skorzystamy tutaj z intencji wywołującej aplikację Aparat.

#### ZADANIE

Dodaj w *CameraActivity* kod odpowiedzialny za wykonywanie zdjęć.

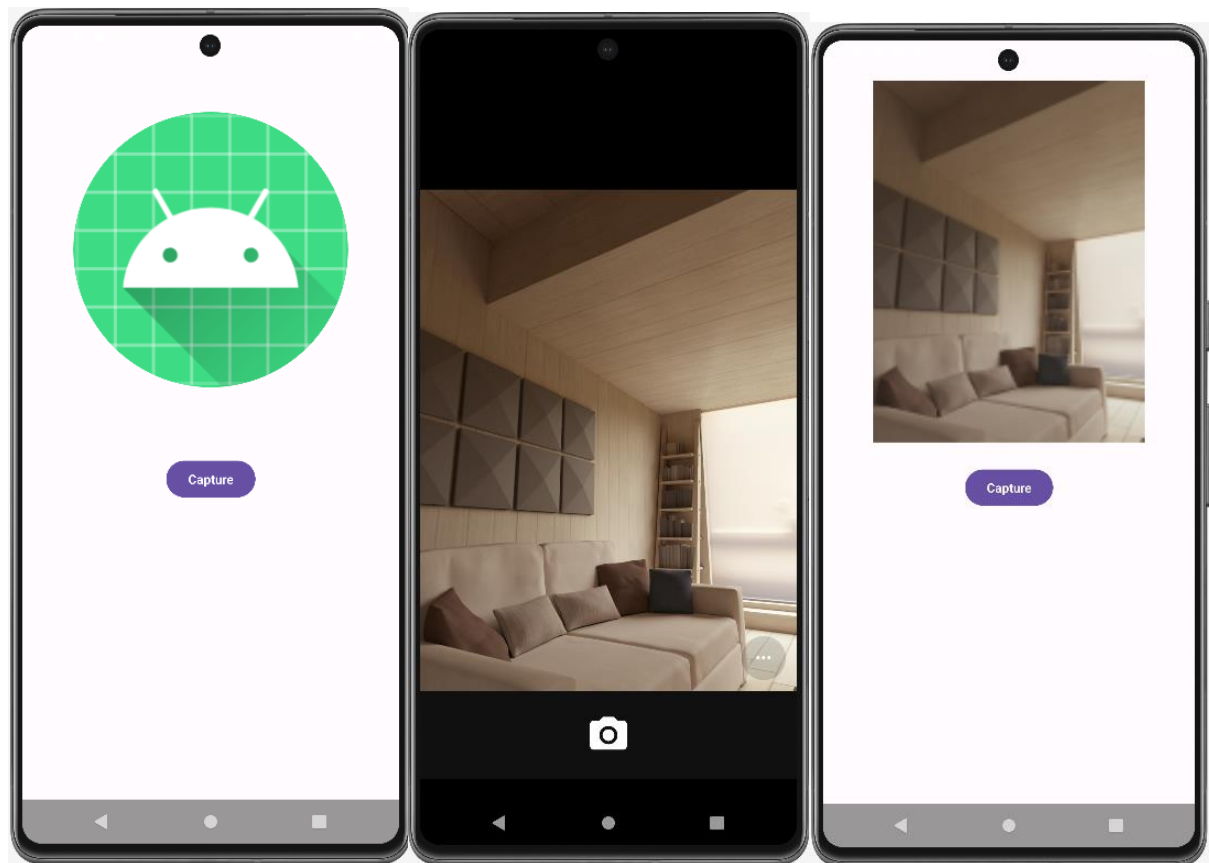
```
class CameraActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_camera)

        findViewById<Button>(R.id.capture_BT).setOnClickListener { it: View!
            // intencja wywołująca aplikację Aparat
            val intentCapture = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
            // uruchomienie intencji z możliwością dostępu do rezultatu
            // jej działania - w tym przypadku do wykonanego zdjęcia
            startActivityForResult(intentCapture, requestCode: 123)
        }
    }

    // funkcja przetwarzająca wynik działania intencji
    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)
        var bmp: Bitmap? = null
        if (data != null && requestCode == 123) {
            bmp = data.extras?.get("data") as Bitmap // pobranie obrazu
            // ustawienie obrazu w ImageView
            findViewById<ImageView>(R.id.camera_IV).setImageBitmap(bmp)
        }
    }
}
```

Korzystamy z metody *startActivityForResult()*, która nie tylko uruchamia intencję ale również pozwala na dostęp do wyniku jej działania. Do naszej intencji przypisujemy kod '123' aby później móc odnieść się do niej. Następnie w metodzie *onActivityResult()* sprawdzamy czy dane będące rezultatem działania intencji są niepuste (*data!=null*) oraz zgodność kodu nadanego wcześniej intencji (wynikowi jej działania). Ostatecznie przetwarzamy otrzymany rezultat – w naszym przypadku wyświetlamy zdjęcie w *ImageView*.

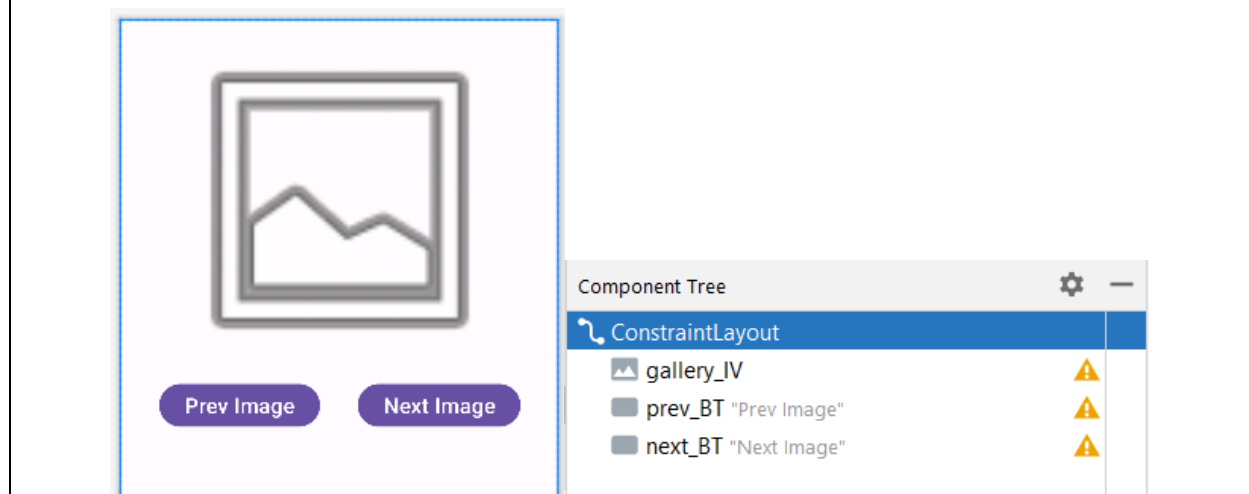


## Tworzenie galerii zdjęć

Teraz zajmiemy się galerią zdjęć. Zaczynamy od zbudowania layoutu. Nasza galeria zbudowana będzie z komponentu *ImageView* do wyświetlania obrazów oraz 2 przycisków umożliwiających nawigowanie.

### ZADANIE

Dodaj do layoutu *activity\_gallery* komponent *ImageView* oraz 2 przyciski *Button* wg poniższego rysunku. Wprowadź identyfikatory komponentów i ustaw wyświetlany na nich tekst.



Mając gotowy layout uzupełniamy kod aktywności galerii. Na początku musimy zbudować listę wszystkich plików graficznych (ścieżek do plików) znajdujących się w pamięci telefonu aby następnie

móc swobodnie nawigować pomiędzy nimi. W tym celu utworzymy funkcję, która pozwoli nam utworzyć odpowiednią strukturę.

#### **ZADANIE**

Zaimplementuj w `Gallery_Activity` poniższą funkcję.

```
fun getAllImages(): ArrayList<String> {  
    val fileList: ArrayList<String> = ArrayList()  
    // katalog główny urządzenia  
    var path: String = Environment.getExternalStorageDirectory().absolutePath  
    // pętla przechodząca po wszystkich katalogach, podkatalogach i plikach  
    File(path).walk().forEach { it: File  
        if(it.toString().endsWith(suffix: ".jpg")) // filtr plików JPG  
            fileList.add(it.toString())  
    }  
    return fileList  
}
```

Powyższa funkcja tworzy `ArrayList` ścieżek do plików graficznych w formacie JPG. Teraz wystarczy wywołać naszą funkcję a następnie oprogramować akcje przycisków nawigacyjnych naszej galerii.

#### **ZADANIE**

Zaimplementuj w metodzie `onCreate()` następujący kod:

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_gallery)  
  
    val gallery_IV = findViewById<ImageView>(R.id.gallery_IV)  
    val prev_BT = findViewById<Button>(R.id.prev_BT)  
    val next_BT = findViewById<Button>(R.id.next_BT)  
  
    // ArrayLista wszystkich obrazów  
    val allImagesPaths = getAllImages()  
    // indeks aktualnie wyświetlanego obrazu  
    var currImgIndex = 0  
  
    // wyświetlenie pierwszego obrazu (indeks 0)  
    gallery_IV.setImageURI(Uri.parse(allImagesPaths.get(currImgIndex)))  
  
    // przejście do wyświetlania poprzedniego obrazu  
    prev_BT.setOnClickListener { it: View!  
        gallery_IV.setImageURI(Uri.parse(allImagesPaths.get(currImgIndex - 1)))  
        currImgIndex--  
    }  
  
    // przejście do wyświetlania kolejnego obrazu  
    next_BT.setOnClickListener { it: View!  
        gallery_IV.setImageURI(Uri.parse(allImagesPaths.get(currImgIndex + 1)))  
        currImgIndex++  
    }  
}
```

Nasza aplikacja wyszukuje obrazy i wyświetla je oraz umożliwia nawigowanie pomiędzy nimi. Jednak trzeba dodać pewne zabezpieczenia:

- Nie możemy wyświetlać obrazów jeżeli nie ma ich w telefonie. Wówczas przyciski powinny być nieaktywne.
- Jeśli istnieje tylko 1 obraz to wyświetlamy go, ale przyciski pozostają nieaktywne.
- Przy wyświetlaniu pierwszego obrazu z listy przycisk *Prev Image* powinien pozostać nieaktywny.
- Przy wyświetlaniu ostatniego pliku z listy przycisk *Next Image* powinien być nieaktywny.
- Odblokowanie przycisków nawigacyjnych powinno nastąpić po opuszczeniu skrajnych pozycji listy – odpowiednio *Prev Image* gdy jesteśmy za indeksem 0 oraz *Next Image* będąc przed ostatnim indeksem.

### **ZADANIE**

Zmodyfikuj kod metody *onCreate()* wg poniższych obrazów.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_gallery)

    val gallery_IV = findViewById<ImageView>(R.id.gallery_IV)
    val prev_BT = findViewById<Button>(R.id.prev_BT)
    val next_BT = findViewById<Button>(R.id.next_BT)

    // ArrayLista wszystkich obrazow
    val allImagesPaths = getAllImages()
    // indeks aktualnie wyswietlanego obrazu
    var currImgIndex = 0

    if(allImagesPaths.size>0) {
        // wyswietlenie pierwszego obrazu (indeks 0)
        gallery_IV.setImageURI(Uri.parse(allImagesPaths.get(currImgIndex)))
    }
    prev_BT.isEnabled = false // na starcie przycisk nieaktywny
    if(allImagesPaths.size<=1)
        next_BT.isEnabled=false
}
```



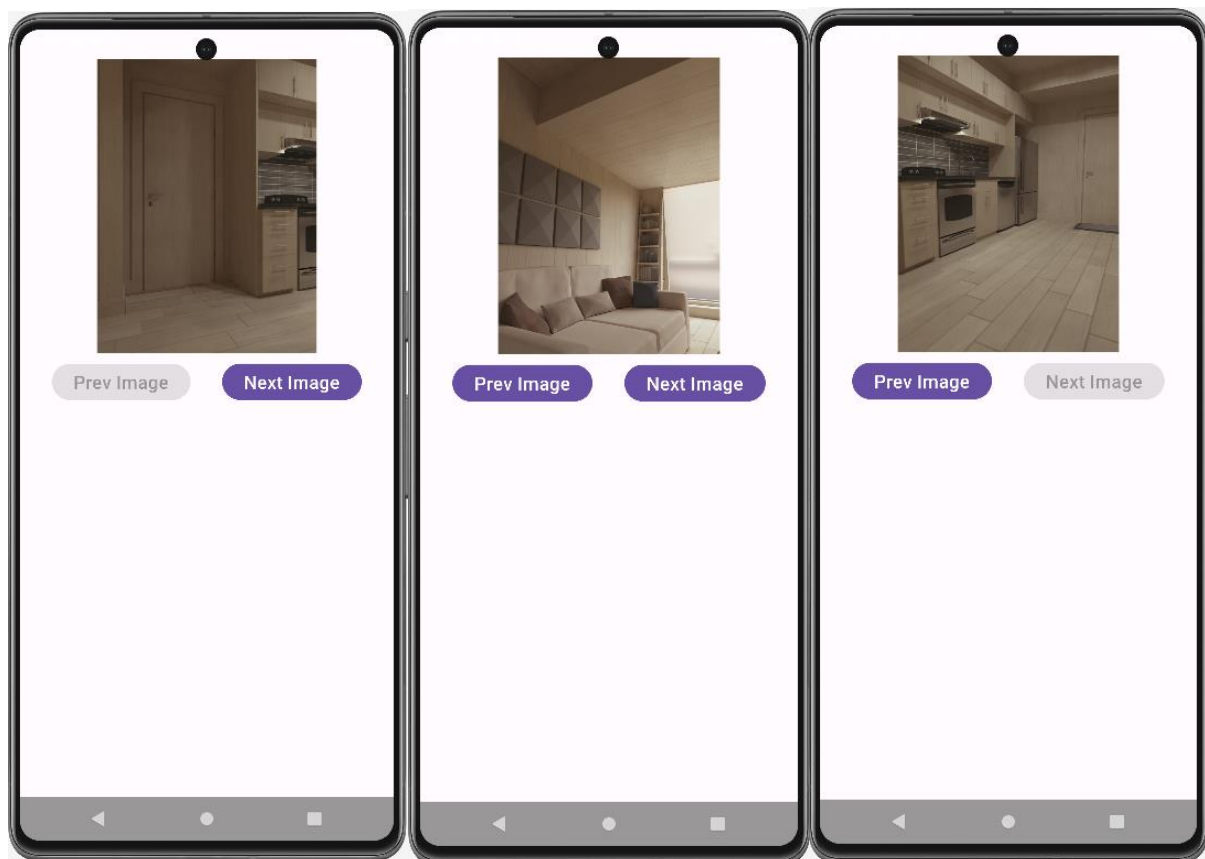
```

// przejście do wyświetlania poprzedniego obrazu
prev_BT.setOnClickListener { it: View!
    if (currImgIndex > 0) {
        gallery_IV.setImageURI(Uri.parse(allImagesPaths.get(currImgIndex - 1)))
        currImgIndex--
        if (!next_BT.isEnabled)
            next_BT.isEnabled = true
    }
    if (currImgIndex == 0)
        prev_BT.isEnabled = false
}

// przejście do wyświetlania kolejnego obrazu
next_BT.setOnClickListener { it: View!
    if (currImgIndex < allImagesPaths.size - 1) {
        gallery_IV.setImageURI(Uri.parse(allImagesPaths.get(currImgIndex + 1)))
        currImgIndex++
        if (!prev_BT.isEnabled)
            prev_BT.isEnabled = true
    }
    if (currImgIndex == allImagesPaths.size - 1)
        next_BT.isEnabled = false
}
}

```

Galeria gotowa, możemy bezpiecznie poruszać się między obrazami.

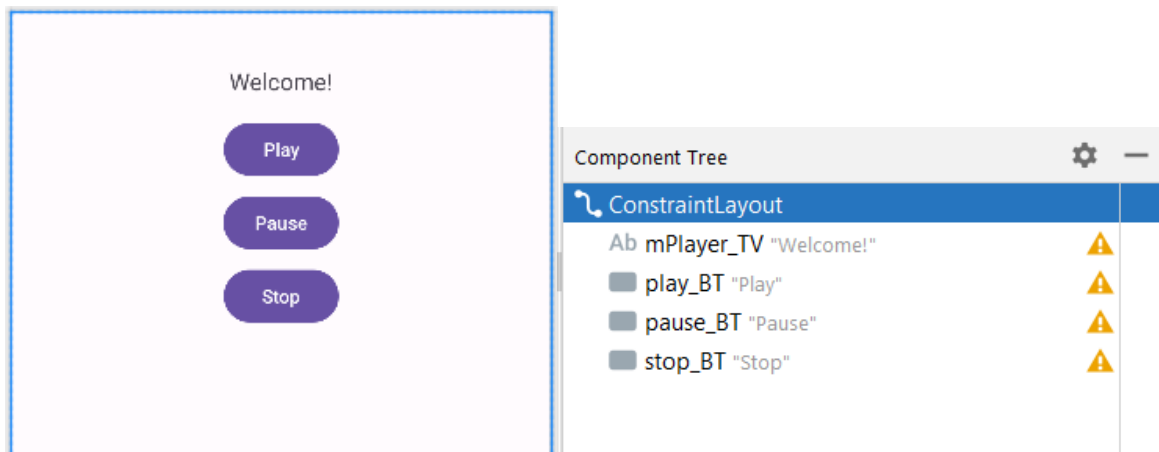


## Odtwarzanie muzyki

W celu utworzenia prostego odtwarzacza muzyki skorzystamy z klasy *MediaPlayer*, która udostępnia gotowe metody do odtwarzania dźwięku. Zanim jednak zajmiemy się samym dźwiękiem przygotujmy layout naszego playera. Będzie on zbudowany z komponentu *TextView* do wyświetlania informacji o stanie playera oraz 3 przycisków *Button* - odpowiednio do rozpoczynania odtwarzania, pauzowania i zatrzymywania odtwarzania.

### ZADANIE

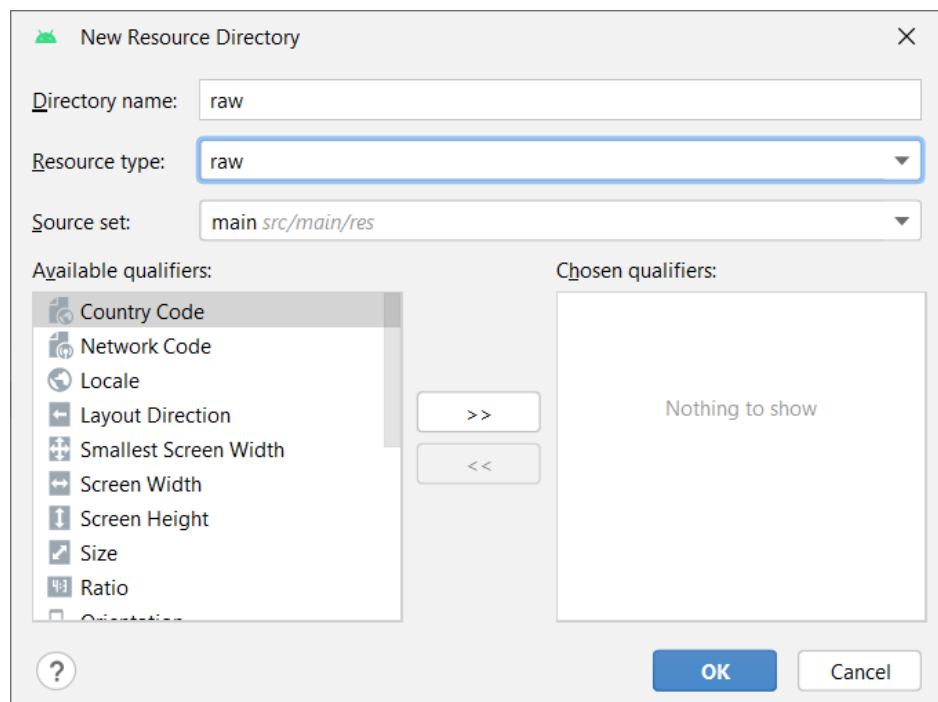
Zmodyfikuj layout *activity\_music.xml* wg poniższych obrazów.



Docelowo nasz player będzie odtwarzał plik, który będzie znajdował się w jednym z katalogów aplikacji. Utwórzmy zatem potrzebny folder i umieśćmy w nim plik dźwiękowy.

### ZADANIE

Utwórz nowy katalog *raw* w naszym projekcie. W tym celu kliknij PPM na folder *res* i wybierz *New/Android Resource Directory*. Następnie wypełnij pola kreatora tworzenia katalogu wg poniższego rysunku.



Dalej dodaj do utworzonego katalogu plik dźwiękowy.

Możemy teraz przystąpić do oprogramowania kontrolek naszego playera. Na początek utworzymy zmienną klasy *MediaPlayer* i zajmiemy się przyciskiem *Play*.

### ZADANIE

W klasie głównej aktywności zadeklaruj obiekt klasy *MediaPlayer*:

```
class MusicActivity : AppCompatActivity() {  
  
    private var mediaPlayer: MediaPlayer? = null
```

Następnie zaimplementuj funkcję do odtwarzania dźwięku:

```
fun play(){  
    if(mediaPlayer==null) {  
        mediaPlayer = MediaPlayer.create(context: this, R.raw.my_music_file)  
    }  
    mediaPlayer?.start() // rozpoczęcie odtwarzania  
}
```

Oprogramuj przycisk *Play* wywołując utworzoną funkcję *play()*:

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_music)  
  
    findViewById<Button>(R.id.play_BT).setOnClickListener { it: View!  
        play()  
    }  
}
```

Uruchom aplikację i sprawdź działanie playera.

W metodzie *play()* za pomocą konstrukcji *MediaPlayer.create()* tworzymy obiekt odpowiedzialny za odtwarzanie dźwięku. Jako parametry podajemy kontekst oraz plik, który ma zostać odtworzony. Dalej rozpoczynamy odtwarzanie dźwięku.

Utwórzmy teraz pozostałe funkcje niezbędne w naszym odtwarzaczu oraz wywołajmy je dla akcji przycisków *Pause* i *Stop*.

### ZADANIE

Zaimplementuj kod poniższych metod:

```

fun pause(){
    if(mediaPlayer!=null)
        mediaPlayer?.pause()
}

fun stop(){
    if(mediaPlayer!=null) {
        mediaPlayer?.release() // zwolnienie zasobu (pliku) powiazanego z playerem
        mediaPlayer = null
    }
}

```

Dopisz również kod przycisków do pauzowania i zatrzymywania odtwarzania:

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_music)

    findViewById<Button>(R.id.play_BT).setOnClickListener { it: View!
        play()
        findViewById<TextView>(R.id.mPlayer_TV)
            .setText("Playing: "+getResources().getResourceEntryName(R.raw.my_music_file))
    }

    findViewById<Button>(R.id.pause_BT).setOnClickListener { it: View!
        pause()
        findViewById<TextView>(R.id.mPlayer_TV)
            .setText("Paused")
    }

    findViewById<Button>(R.id.stop_BT).setOnClickListener { it: View!
        stop()
        findViewById<TextView>(R.id.mPlayer_TV)
            .setText("Stopped")
    }
}

```

W funkcji `stop()` zwalniamy najpierw zasoby powiązane z playerem, po czym „kasujemy” sam player przypisując do niego wartość `null`.

Z kolei w kodzie przycisków dodane zostały linie, które odpowiadają za wyświetlenie nazwy odtwarzanego pliku (przycisk *Play*) oraz informacji o stanie playera.

Zadbajmy jeszcze o zwolnienie zasobów playera w momencie zatrzymania odtwarzania z powodu zakończenia pliku. W tym celu wprowadźmy modyfikację metody `play()`. Zatrzymajmy także odtwarzanie dźwięku przy wyjściu z aktywności i zwolnijmy zasoby.

#### **ZADANIE**

Zmodyfikuj kod metody `play()`:

```

fun play(){
    if(mediaPlayer==null) {
        mediaPlayer = MediaPlayer.create(context: this, R.raw.my_music_file)
        mediaPlayer?.setOnCompletionListener(MediaPlayer.OnCompletionListener { it: MediaPlayer!
            fun onCompletion(mediaPlayer: MediaPlayer) {
                stop()
            }
        })
    }
    mediaPlayer?.start() // rozpoczęcie odtwarzania
}

```

Dodaj funkcję `onStop()`:

```

override fun onStop() {
    super.onStop()
    stop()
}

```

Przetestuj działanie playera.

Nasz player jest gotowy.

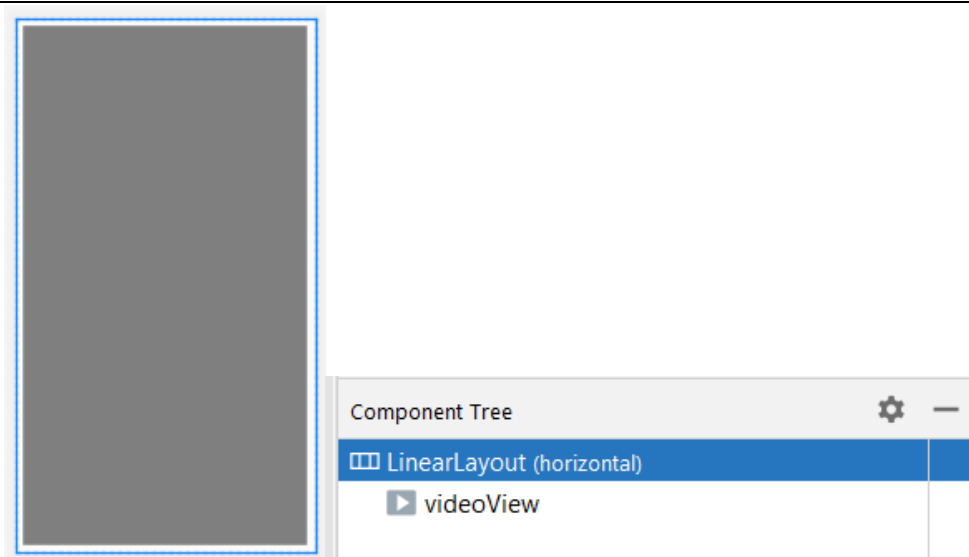


## Odtwarzanie filmów

Przed nami ostatnia aktywność – player video. Rozpoczynamy od utworzenia layoutu.

### ZADANIE

Zmodyfikuj layout `activity_video.xml` wg poniższych obrazów.



Ustaw dla komponentu *videoView* następujące parametry:

```
<VideoView
    android:id="@+id/videoView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="10dp" />
```

Potrzebujemy teraz pliku video, który będziemy odtwarzać. W naszej aplikacji używać będziemy plików w formacie *.mp4*. UWAGA: Nazwa pliku może zawierać wyłącznie małe litery, cyfry i znaki podkreślenia.

#### **ZADANIE**

Dodaj do pakietu *raw* plik video w formacie *mp4*.

Przystępujemy do implementacji funkcjonalności odtwarzania filmów. Aby odtwarzać plik video wystarczy dla komponentu *VideoView* ustawić zasób (podać plik), który ma być odtwarzany oraz wywołać metodę *start()*. My jednak chcemy nie tylko móc rozpocząć odtwarzanie filmu ale również jego pauzowanie i zatrzymywanie. Dlatego dla *VideoView* ustawimy również *MediaController*.

#### **ZADANIE**

Dodaj poniższy kod w metodzie *onCreate()* w *VideoActivity*:

```
val videoView = findViewById<VideoView>(R.id.videoView)
val path = "android.resource://" + packageName + "/" + R.raw.vid_001
val uri: Uri = Uri.parse(path)
videoView.setVideoURI(uri)
videoView.setMediaController(MediaController(context, this))
```

Przetestuj działanie playera video.

Po wywołaniu aktywności *VideoActivity* naszym oczom ukazuje się komponent *VideoView*, który jest czarny – film nie jest odtwarzany. Dopiero po kliknięciu na ten komponent w dolnej części ekranu widzimy *MediaController* - pasek z przyciskami umożliwiającymi sterowanie odtwarzaniem.



Nasz player działa prawidłowo, jednak położenie *MediaControllera* sprawia, że użytkowanie odtwarzacza jest uciążliwe. Dlatego wprowadzimy modyfikację layoutu oraz drobne zmiany w kodzie aktywności.

#### **ZADANIE**

Wprowadź poniższe zmiany w layoucie aktywności video playera.

```

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

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp">

        <VideoView
            android:id="@+id/videoView"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />
        </FrameLayout>
    </LinearLayout>

```

Ponadto dla *LinearLayout* ustaw wartość parametru *gravity* na *center*.

Zmodyfikuj również kod *VideoActivity*:

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_video)

    val videoView = findViewById<VideoView>(R.id.videoView)
    val path = "android.resource://" + packageName + "/" + R.raw.vid_001
    val uri: Uri = Uri.parse(path)
    videoView.setVideoURI(uri)
    val mediaController = MediaController(context: this)
    videoView.setMediaController(mediaController)
    mediaController.setAnchorView(videoView)
}

```

Teraz nasz video player jest gotowy.



