

Wprowadzenie do intencji (Android Intents)

Intencje (`android.content.Intent`) to system przesyłania wiadomości, za pomocą którego jedna aktywność (`activity`) może uruchomić inną aktywność.

Aktywność może na przykład zażądać uruchomienia innej aktywności będącej częścią tej samej aplikacji.

Można również zarządzać uruchomieniem dowolnej innej odpowiednio zarejestrowanej aktywności na urządzeniu, dla której skonfigurowane są uprawnienia.

Rozważmy na przykład działanie zawarte w aplikacji, które wymaga załadowania i wyświetlenia użytkownikowi strony internetowej.

Zamiast tworzenia drugiej aktywności w tej aplikacji, kod może wysłać intencję do środowiska wykonawczego Androida, prosząc o wykonanie przez aplikację, która zarejestrowała możliwość wyświetlenia strony internetowej.

System wykonawczy (`runtime system`) dopasuje żądanie do dostępnych aktywności na urządzeniu i albo uruchomi pasującą aktywność, albo, w przypadku wielu możliwości, pozwoli użytkownikowi zdecydować, której aktywności użyć.

Intencje umożliwiają również przesyłanie danych z aktywności wysyłającej do aktywności odbierającej.

W opisanym wcześniej przykładzie aktywność wysyłająca musiałaby wysłać do drugiej aktywności adres URL strony internetowej, która ma zostać wyświetlona.

Podobnie aktywność odbierającą można skonfigurować tak, aby zwracała dane do aktywności wysyłającej po ukończeniu wymaganych zadań.

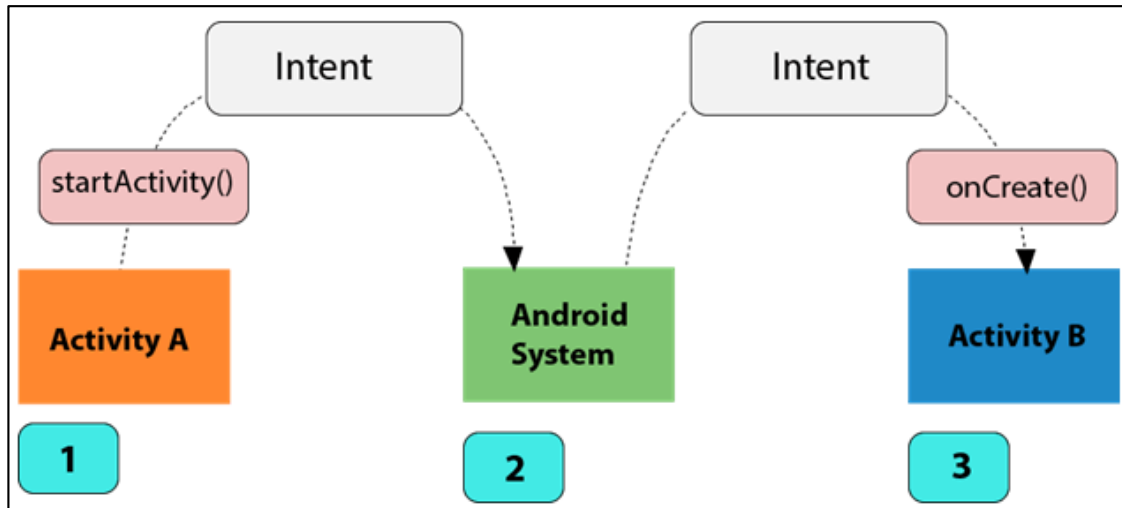
Warto również podkreślić, że oprócz uruchamiania aktywności, intencje są również wykorzystywane do uruchamiania usług (`service`) i odbiorników transmisji (`broadcast receivers`) oraz komunikowania się z nimi.

Intencje dzieli się na jawne (*explicit*) i ukryte-domniemane (*implicit*).

Jawne intencje (Explicit Intents)

Żądanie wykonania jawnej (explicit) intencji żąda uruchomienia określonej aktywności (nazwy komponentu, która jest nazwą klasy).

To podejście jest najbardziej powszechne podczas uruchamiania aktywności znajdującej się w tej samej aplikacji, co aktywność wysyłająca (nazwa klasy jest znana programiście).



Jawna intencja jest wykonywana poprzez utworzenie instancji klasy `Intent`, wykorzystując kontekst (context) aktywności i nazwę aktywności, która ma zostać uruchomiona.

Następnie wywoływana jest metoda `startActivity()`, przekazując intencję jako argument.

Poniższy fragment kodu generuje intencję uruchamiającą aktywność o nazwie `ActivityB`:

```
val i = Intent(this, ActivityB::class.java)
startActivity(i)
```

Dane mogą być przesyłane do aktywności odbierającej poprzez dodanie ich do intencji przed jego uruchomieniem wywołując metodę `putExtra()`; dane dodajemy w formacie para klucz-wartość.

Poniższy kod rozszerza poprzedni przykład, dodając wartości typu `String` i `integer`:

```
val i = Intent(this, ActivityB::class.java)
i.putExtra("message", "Wiadomość z ActivityB")
i.putExtra("liczbaInt", 100)

startActivity(i)
```

Aktywność docelowa odbiera dane jako część obiektu `Bundle`, który można uzyskać poprzez wywołanie metody `getIntent().getExtras()`.

Metoda `getIntent()` (klasy `Activity`) zwraca intencję, która rozpoczęła aktywność, natomiast metoda `getExtras()` (klasy `Intent`) zwraca obiekt `Bundle` zawierający dane.

Na przykład, aby wyodrębnić wartości danych przekazane do Działania B:

```
val extras = intent.extras ?: return

val tekst = extras.getString("message")
int liczba = extras.getInt("liczbaInt")
```

W przypadku używania intencji do uruchamiania innych aktywności w tej samej aplikacji, aktywności te muszą być wymienione w **pliku manifestu aplikacji**.

Poprawnie skonfigurowana zawartość pliku `AndroidManifest.xml` dla aplikacji zawierającej aktywności o nazwach `ActivityA` i `ActivityB`:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.amw.intent1.intent1" >

    <application
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name="com.amw.intent1.intent1.ActivityA" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="ActivityB"
            android:label="ActivityB" >
        </activity>
    </application>
</manifest>
```

Zwracanie Danych z Aktywności

Jak widać w poprzednim przykładzie, podczas przesyłania danych do ActivityB nie ma możliwości zwrócenia danych do pierwszej aktywności (ActivityA).

Można to jednak osiągnąć poprzez uruchomienie ActivityB jako pod-aktywności ActivityA.

Aktywność jest uruchamiana jako aktywność podrzędna poprzez utworzenie instancji *ActivityResultLauncher*.

Instancja *ActivityResultLauncher* jest tworzona przez wywołanie metody *podraz* przekazanie do niej procedury obsługi wywołania zwrotnego w postaci lambdy.

Ta procedura obsługi zostanie wywołana i przekaże dane zwrotne po powrocie do aktywności głównej.

Po utworzeniu instancji *ActivityResultLauncher* można ją wywołać z parametrem intencji w celu uruchomienia sub-activity.

Kod tworzący instancję *ActivityResultLauncher* zazwyczaj wygląda następująco:

```
val startForResult = registerForActivityResult(  
    ActivityResultContracts.StartActivityForResult()) {  
        result: ActivityResult ->  
    if (result.resultCode == Activity.RESULT_OK) {  
        val data = result.data  
        data?.let {  
            // Kod obsługi danych zwrotnych  
        }  
    }  
}
```

Gdy program uruchamiający będzie gotowy, można go wywołać i przekazać intencję uruchomienia w następujący sposób:

```
val i = Intent(this, ActivityB::class.java)  
.  
.  
startForResult(intent)
```

Aby zwrócić dane do aktywności nadrzędnej, pod-aktywność (sub-activity) musi zaimplementować metodę *finisz()*.

Kod wyniku to zazwyczaj *RESULT_OK* lub *RESULT_CANCELED*, ale może to być również wartość niestandardowa w zależności od wymagań programisty.

Jeśli aktywność podrzędna ulegnie awarii, aktywność nadrzędna otrzyma kod wyniku `RESULT_CANCELED`.

Poniższy przykład ilustruje kod typowej metody `finish()` pod-aktywności:

```
override fun finish() {  
    val data = Intent()  
  
    data.putExtra("tekstZwrotny", "Wiadomość do aktywności nadrzędnej")  
  
    setResult(RESULT_OK, data)  
    super.finish()  
}
```

Intencje ukryte – domniemane (Implicit Intents)

W przeciwieństwie do jawnych intencji (explicit), które odwołują się do nazwy klasy aktywności, która ma zostać uruchomiona, ukryte intencje identyfikują aktywność, która ma zostać uruchomiona, określając akcję do wykonania i typ danych, które mają być obsługiwane przez aktywność odbierającą.

Na przykład akcja typu ACTION_VIEW, której towarzyszy adres URL strony internetowej w postaci obiektu URI, poinstruuje system Android, aby wyszukał, a następnie uruchomił aktywność obsługującą przeglądarkę internetową.

Poniższa ukryta intencja, wykonana na urządzeniu z systemem Android, spowoduje wyświetlenie określonej strony internetowej w aktywności przeglądarki internetowej:

```
val intent = Intent(Intent.ACTION_VIEW,
    Uri.parse("http://www.amw.gdynia.pl"))
startActivity(intent)
```

Gdy aktywność zażąda wykonania domniemanej intencji (implicit), system Android zacznie szukać w urządzeniu takiej aktywności, która zarejestrowała możliwość obsługi żądań ACTION_VIEW na danych schematu HTTP, korzystając z procesu określanego jako rozpoznawanie intencji (*intent resolution*).

Zanim system uruchomi aktywność, korzystając z domniemanej intencji (implicit intent), użytkownik musi ją zweryfikować lub włączyć.

Jeżeli żaden z tych warunków nie zostanie spełniony, zamierzona aktywność nie zostanie wykonana.

Zanim omówimy te dwie opcje, najpierw musimy poznać **filtry intencji**.

Użycie filtrów intencji (Using Intent Filters)

Filtry intencji to mechanizm, dzięki któremu aktywności „reklamują” obsługiwane działania i możliwości obsługi danych w procesie rozpoznawania intencji Androida.

Deklaracje te zawierają także ustawienia niezbędne do przeprowadzenia procesu weryfikacji linków.

Poniższy plik AndroidManifest.xml ilustruje konfigurację aktywności o nazwie WebActivity w aplikacji o nazwie MyWebView z odpowiednio skonfigurowanym filtrem intencji:

```
<?xml version="1.0" encoding="utf-8"?>
.
.
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.MyWebView">
    <activity
        android:name="WebActivity"
        android:exported="true">
        <intent-filter android:autoVerify="true">
            <action android:name="android.intent.action.VIEW" />
            <category android:name="android.intent.category.BROWSABLE" />
            <category android:name="android.intent.category.DEFAULT" />
            <data android:scheme="https" />
            <data android:host="www.amw.gdynia.pl" />
        </intent-filter>
    </activity>
</application>
</manifest>
```

Powyższy plik manifestu konfiguruje aktywność WebActivity, która ma zostać uruchomiona w odpowiedzi na intencję implicit (ukrytą) pochodzącą z innej aktywności, gdy intencja zawiera adres URL <https://www.amw.com>.

Poniższy kod uruchomi aktywność WebActivity (zakładając, że aplikacja MyWebView została zweryfikowana lub włączona przez użytkownika):

```
val intent = Intent(Intent.ACTION_VIEW,
    Uri.parse("https://www.amw.gdynia.pl"))
startActivity(intent)
```

Automatyczna weryfikacja Linku

Korzystanie z łącza internetowego w celu uruchomienia aktywności na urządzeniu z systemem Android jest uważane za potencjalne zagrożenie bezpieczeństwa.

Aby zminimalizować to ryzyko, link użyty do uruchomienia intencji musi zostać automatycznie zweryfikowany lub ręcznie dodany jako link obsługiwany przez użytkownika.

Aby włączyć automatyczną weryfikację, odpowiednia deklaracja intencji w aktywności docelowej musi ustawić `autoVerify` na `true` w następujący sposób:

```
<intent-filter android:autoVerify="true">
.
.
</intent-filter>
```

Następnie adres URL linku musi być powiązany ze stroną internetową, na której bazuje link do aplikacji.

Osiąga się to poprzez utworzenie pliku **Digital Assets Link** o nazwie *assetslinks.json* i zainstalowanie go w folderze witryny internetowej.

Plik łącza do zasobu cyfrowego (digital asset link) składa się z instrukcji umożliwiającej udzielenie pozwolenia na uruchomienie aplikacji docelowej przy użyciu adresów URL oraz instrukcji docelowej deklarującej nazwę pakietu aplikacji towarzyszącej i odcisk palca (certificate fingerprint) certyfikatu SHA-256 dla tego projektu.

Typowy plik łącza do zasobów może na przykład mieć następującą treść:

```
[{
  "relation": ["delegate_permission/common.handle_all_urls"],
  "target": {
    "namespace": "android_app",
    "package_name": "com.amw.mywebview",
    "sha256_cert_fingerprints":
      ["<your certificate fingerprint here>"]
  }
}]
```

Ten plik można utworzyć ręcznie lub wygenerować za pomocą narzędzia online dostępnego pod adresem URL:

<https://developers.google.com/digital-asset-links/tools/generator>

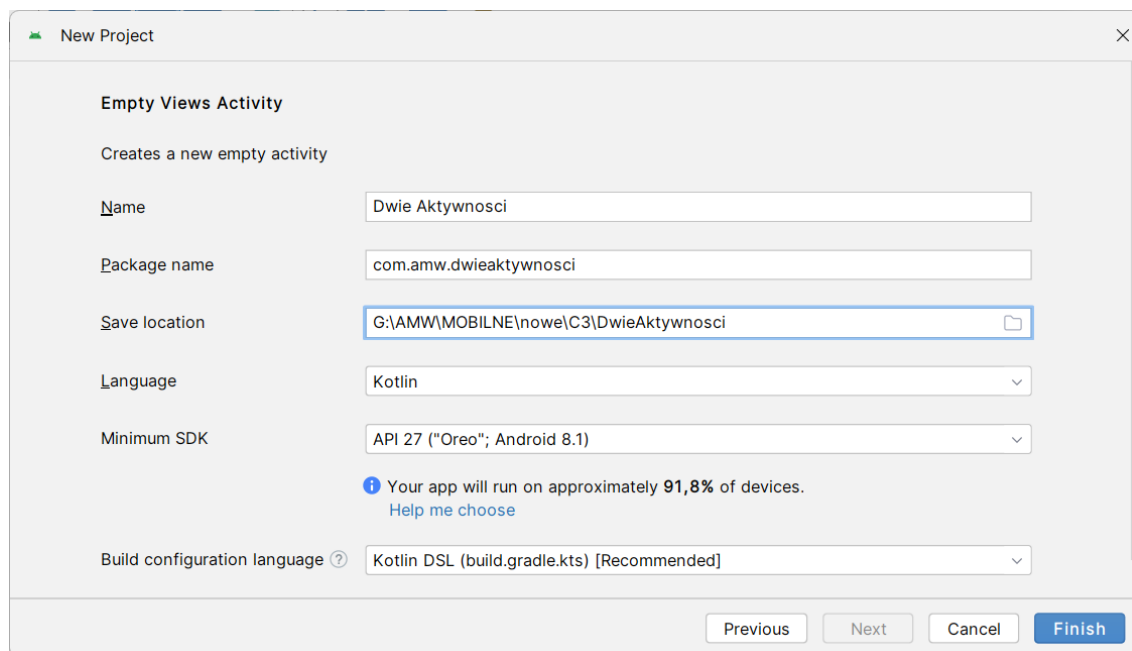
Android Explicit Intents - Przykład

Zbudujemy przykładową aplikację demonstrującą użycie Explicit Intent do uruchomienia drugiej aktywności, w tym transferu danych pomiędzy działaniami wysyłającymi i odbierającymi.

Tworzymy aplikację

Tworzymy nowy projekt (szablon Empty Views Activity).

Nazwa projektu: *DwieAktywnosci*



New Project

Empty Views Activity

Creates a new empty activity

Name: Dwie Aktywnosci

Package name: com.amw.dwieaktywnosci

Save location: G:\AMW\MOBILNE\nowe\C3\DwieAktywnosci

Language: Kotlin

Minimum SDK: API 27 ("Oreo"; Android 8.1)

ⓘ Your app will run on approximately 91,8% of devices.
[Help me choose](#)

Build configuration language ? Kotlin DSL (build.gradle.kts) [Recommended]

Previous Next Cancel Finish

Skonwertuj projekt aby użyć view binding (przypomnienie na końcu tekstu)

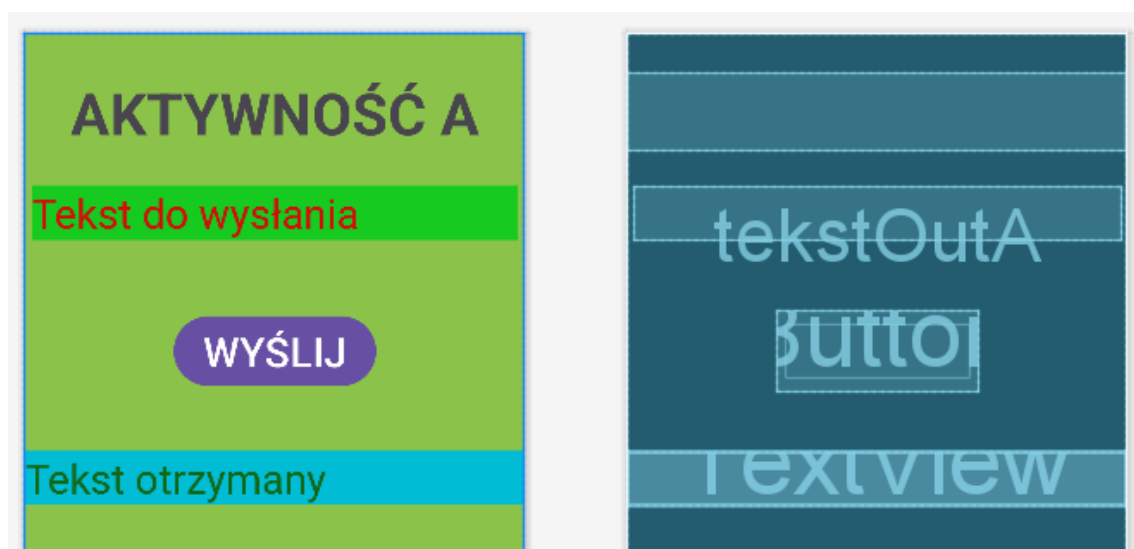
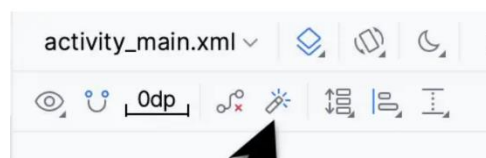
Tworzymy interfejs użytkownika MainActivity

Interfejs użytkownika głównej aktywności (MainActivity) będzie oparty o układ ConstraintLayout zawierający EditText (Plain Text), TextView x 2, oraz Button.

Sprawdzamy czy tryb autoconnect jest zezwolony i usuwamy domyślny TextView, po czym dodajemy widgety TextView, PlainText, Button, TextView:

parametr	TextView	PlainText	Button	TextView	układ
id	tytułA	tekstOutA	klawisza	tekstInA	main
text	AKTYWNOŚĆ A		WYŚLIJ	Tekst otrzymany	
text-hint		Tekst do wysłania			
width	match_parent	wrap_content	wrap_content	match_parent	match_parent
height	wrap_content	wrap_content	wrap_content	wrap_content	match_parent
textColor		#673AB7		#126716	
textColorHint		#D80A0A			
textSize	48sp	34sp	34sp	34sp	
textStyle	bold				
background		#16CA1F		#00bcd4	#8BC34A
gravity	center				
onClick			wyslij		

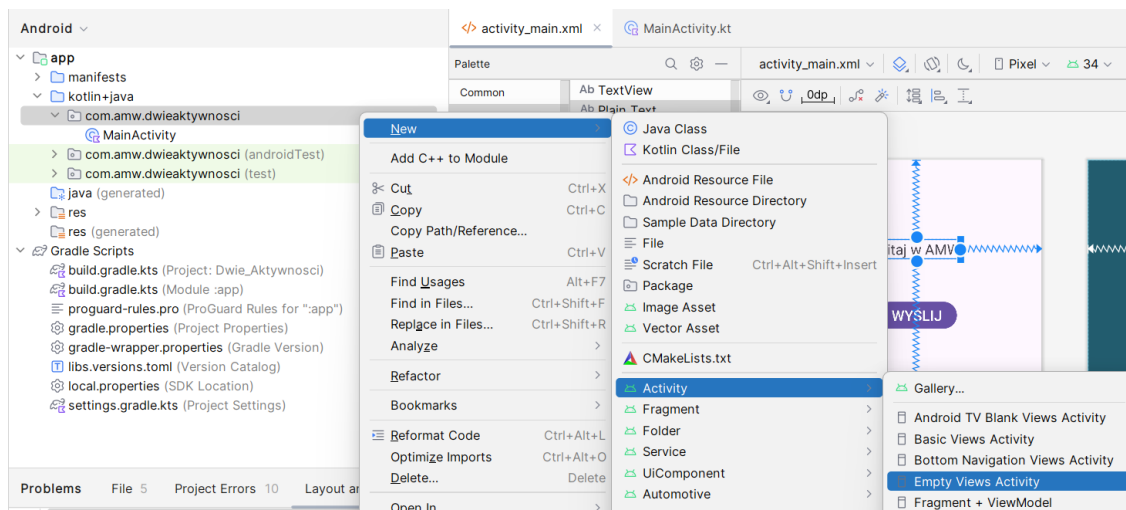
Użyj przycisku *Infer constraints* aby dodać brakujące powiązania, układ powinien wyglądać jak na rysunku:



Tworzymy drugą aktywność

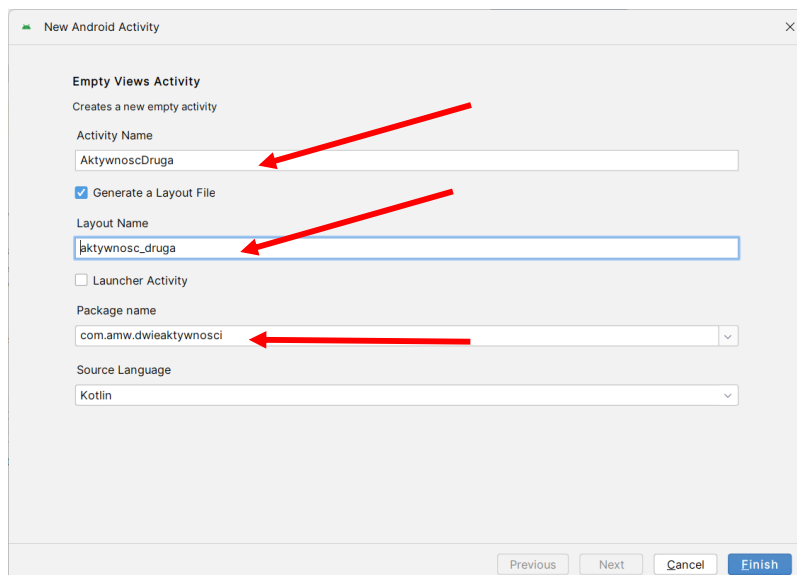
Po kliknięciu przycisku „WYŚLIJ” zostanie utworzona intencja z prośbą o uruchomienie drugiej aktywności, w której użytkownik przyjmie i wyświetli tekst oraz będzie mógł wpisać odpowiedź.

W oknie narzędzi Project tool klikamy prawym przyciskiem myszy na *com.amw.dwieaktywnosci* (app -> kotlin+java i wybieramy opcję New -> Activity -> Empty Views Activity)



, po czym wyświetla się okno w którym wpisujemy: *AktywnoscDruga* w polu Activity Name oraz *aktywnosc_druga* w polu Layout Name.

Ta ktywność nie będzie aktywnością startową więc pole Launcher Activity nie powinno być zaznaczone.



Projektujemy układ UI dla Drugiej Aktywności

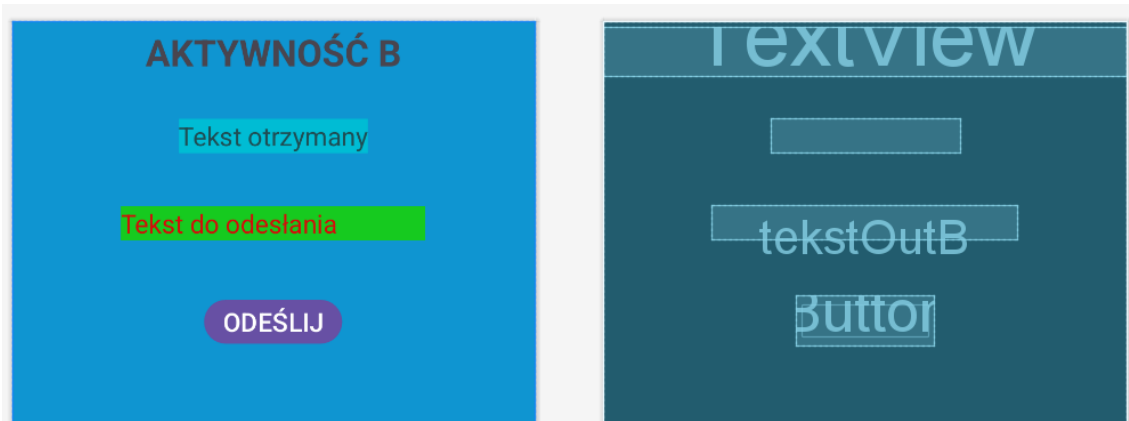
Będziemy potrzebować: Plain Text EditText, TextView oraz Button.

Ładujemy plik *aktywnosc_druga.xml* do edytora i dodajemy elementy z nazwami odpowiednio:

parametr	TextView	TextView	PlainText	Button	układ
id	tytulB	tekstInB	tekstOutB	klawiszB	main
text	AKTYWNOŚĆ B	Tekst otrzymany		ODEŚLIJ	
text-hint			Tekst do odesłania		
width	match_parent	wrap_content	wrap_content	wrap_content	match_parent
height	wrap_content	wrap_content	wrap_content	wrap_content	match_parent
textColor		#204D53	#673AB7		
textColorHint			#D80A0A		
textSize	48sp	34sp	34sp	34sp	
textStyle	bold				
background		#00bcd4	#034006		#0F95D1
gravity	center		Center		
onClick				odeslij	

Po zakończeniu projektowania UI, nie zapomnij o kliknięciu klawisza *Infer constraints*

UI drugiej aktywności powinno wyglądać:



Przyjrzyjmy się plikowi Application Manifest File

Aby MainActivity mogło uruchomić Drugą Aktywność (przy użyciu intencji), w pliku *AndroidManifest.xml* musi znajdować się wpis AktywnoscDruga.

Sprawdźmy czy Android Studio automatycznie dodało punkt wejściowy aktywności:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.DwieAktywnosci"
        tools:targetApi="31">
        <activity
            android:name=".AktywnoscDruga"
            android:exported="false" />
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Mając drugą aktywność utworzoną i umieszczoną w pliku manifestu napiszemy kodu intentu w pliku MainActivity.

Tworzymy intencję (Intent)

Celem MainActivity jest utworzenie i uruchomienie intencji po kliknięciu klawisza “WYŚLIJ”.

Podczas tworzenia interfejsu użytkownika dla MainActivity, klawisz został tak skonfigurowany, aby po „kliknięciu” wywoływał metodę o nazwie *wyslij()*.

Tę metodę należy teraz dodać do klasy MainActivity pliku MainActivity.kt:

```
package com.amw.dwieaktywnosci

import android.content.Intent
import android.os.Bundle
import android.view.View
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import com.amw.dwieaktywnosci.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()

        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets
->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom)
            insets
        }

        fun wyslij(view: View) {
            val i = Intent(this, AktywnoscDruga::class.java) A

            val danaOut = binding.tekstOutA.text.toString()
            i.putExtra("tekst", danaOut) B
            startActivity(i) C
        }
    }
}
```


Najpierw tworzona jest nowa instancja Intent, przekazująca bieżącą aktywność i nazwę klasy AktywnoscDruga jako argumenty. **A**

Następnie tekst wprowadzony do obiektu EditText jest dodawany do obiektu intencji jako para klucz-wartość **B** a intencja jest uruchamiana poprzez wywołanie metody *startActivity()*, przekazując obiekt intencji jako argument. **C**


Skompiluj i uruchom aplikację, a następnie kliknij klawisz „WYŚLIJ”, aby uruchomić AktywnoscDruga. Wróć do ekranu MainActivity za pomocą przycisku Wstecz (znajdującego się na pasku narzędzi u dołu ekranu) lub przesuwając palcem w prawo od krawędzi ekranu w nowszych wersjach Androida.

Wydobycie danych z intencji (Intent)

Teraz, gdy AktywnoscDruga jest uruchamiana z MainActivity, musimy odczytać dane zawarte w intencji i wyświetlić je w interfejsie użytkownika AktywnoscDruga.

Wiąże się to z dodaniem kodu do metody *onCreate()* w pliku AktywnoscDruga.kt, a także przystosowaniem aktywności do korzystania z ViewBinding: 


```
package com.amw.dwieaktywnosci

import android.os.Bundle
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import com.amw.dwieaktywnosci.databinding.AktywnoscDrugaBinding 

class AktywnoscDruga : AppCompatActivity() {

    private lateinit var binding: AktywnoscDrugaBinding 

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()

        binding = AktywnoscDrugaBinding.inflate(layoutInflater) 
        setContentView(binding.root) 

        val dane_in = intent.extras ?: return
        val tekst = dane_in.getString("tekst")
        binding.tekstInB.text = tekst

        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
                systemBars.bottom)
            insets
        }
    }
}
```

Skompiluj i uruchom aplikację w emulatorze lub na fizycznym urządzeniu z Androidem. Wpisz tekst w polu tekstowym w MainActivity przed kliknięciem klawisza „WYŚLIJ”. Otrzymany tekst powinien pojawić się w interfejsie użytkownika AktywnoscDruga.

Uruchomienie drugiej aktywności jako pod-aktywności (Sub-Activity)

Aby AktywnoscDruga mogła zwracać dane do MainActivity, musi zostać uruchomiona jako sub-aktywność MainActivity.

Oznacza to, że musimy wywołać metodę *RegisterForActivityResult()* i zadeklarować procedurę obsługi wywołania zwrotnego, która będzie wywoływana po powrocie z AktywnoscDruga.

Wywołanie zwrotne (callback) wyodrębni dane zwrócone przez AktywnoscDruga i wyświetli je w MainActivity.

Wywołanie metody *RegisterForActivityResult()* zwraca instancję *ActivityResultLauncher*, którą można wywołać z poziomu metody *wyslij()* uruchamiając intencję.

Edytuj plik *MainActivity.kt*:

```
package com.amw.dwieaktywnosci

import android.app.Activity
import android.content.Intent
import android.os.Bundle
import android.view.View
import androidx.activity.enableEdgeToEdge
import androidx.activity.result.contract.ActivityResultContracts
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import com.amw.dwieaktywnosci.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()

        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
                systemBars.bottom)
            insets
        }

        val startForResult = registerForActivityResult(
            ActivityResultContracts.StartActivityForResult()) {
            result ->
            if (result.resultCode == Activity.RESULT_OK) {
                val data = result.data
                data?.let {
                    if (it.hasExtra("daneZwrotne")) {
                        val returnString = it.extras?.getString("daneZwrotne")
                        binding.tekstInA.text = returnString
                    }
                }
            }
        }
    }
}
```



```

fun wyslij(view: View) {
    val i = Intent(this, AktywnoscDruga::class.java)

    val tekstOut = binding.tekstOutA.text.toString()
    i.putExtra("tekst", tekstOut)
    startForResult.launch(i)
}
}

```

Zwracanie danych z pod-aktywności (Sub-Activity)

AktywnoscDruga jest teraz uruchamiana jako pod-aktywność MainActivity, zmodyfikowanej w celu obsługi zwracanych danych. Pozostaje tylko zmodyfikować plik AktywnoscDruga.kt, aby zaimplementować metodę *finish()* i dodać metodę o nazwie **odeslij()**. Metoda **finish()** jest wywoływana po zakończeniu działania (na przykład, gdy użytkownik wybierze przycisk Wstecz na urządzeniu):

```

package com.amw.dwieaktywnosci

import android.content.Intent
import android.os.Bundle
import android.view.View
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import com.amw.dwieaktywnosci.databinding.AktywnoscDrugaBinding

class AktywnoscDruga : AppCompatActivity() {

    private lateinit var binding: AktywnoscDrugaBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()

        binding = AktywnoscDrugaBinding.inflate(layoutInflater)
        setContentView(binding.root)

        val dane_in = intent.extras ?: return
        val tekst = dane_in.getString("tekst")
        binding.tekst2a.text = tekst

        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets
->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom)
            insets
        }
    }

    fun odeslij(view: View) {
        finish()
    }

    override fun finish() {
        val data = Intent()
        val tekstZwrotny = binding.tekst.text.toString()
        data.putExtra("daneZwrotne", tekstZwrotny)

        setResult(RESULT_OK, data)
        super.finish()
    }
}

```

Metoda *finish()* tworzy nowy intent, dodaje dane zwrotne (para klucz-wartość) i wywołuje metodę *setResult()*.
Metoda *returnText()* wywołuje metodę *finish()*.

Otwórz plik *activity_second.xml*, wybierz element button i ustaw atrybut `onClick` tak, aby wywoływał metodę *odeslijtekst()*.

Testowanie aplikacji

Skompiluj i uruchom aplikację. Sprawdź poprawność działania.

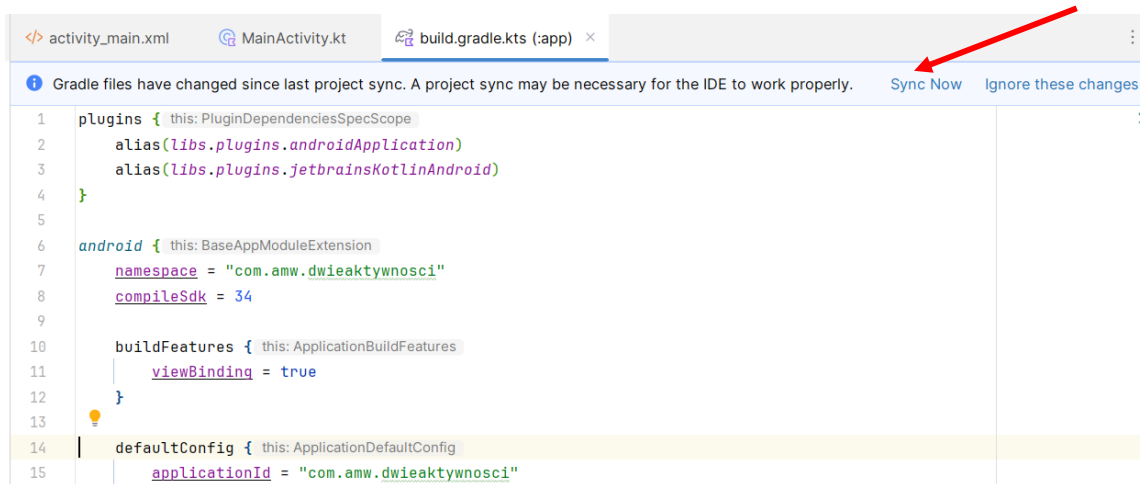
VIEW BINDING

1. Zezwalamy na View Binding

Edytujemy plik *build.gradle.kts* (*Gradle Scripts* -> *build.gradle.kts (Module: app)*) i zezwalamy na ViewBinding:

```
plugins {  
    alias(libs.plugins.androidApplication)  
    alias(libs.plugins.jetbrainsKotlinAndroid)  
}  
  
android {  
    namespace = "com.amw.dwieaktywnosci"  
    compileSdk = 34  
  
    buildFeatures {  
        viewBinding = true  
    }  
  
    defaultConfig {  
        .....
```

Synchronizujemy ponownie aplikację (klasa binding musi zostać wygenerowana):



2. Używamy View Binding w kodzie

Edytujemy plik *MainActivity.kt*

```
package com.amw.dwieaktywnosci

import android.os.Bundle
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import com.amw.dwieaktywnosci.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v,
insets ->
            val systemBars =
insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom)
            insets
        }
    }
}
```

Synchronizujemy ponownie aplikację. Z menu Build wykonaj Rebuild Project oraz Clean Project.

<https://www.answertopia.com/android-studio/an-android-intents-overview/>

<https://www.javatpoint.com/kotlin-android-implicit-intent>