Aplikacje mobilne. Projektowanie aplikacji Android. Definiowanie interfejsu użytkownika

Interfejs użytkownika UX.

Interfejs użytkownika aplikacji to wszystko, co użytkownik może zobaczyć i z czym może korzystać. System Android udostępnia wiele gotowych komponentów interfejsu użytkownika, takich jak obiekty układu strukturalnego i kontrolki interfejsu użytkownika, które umożliwiają zbudowanie graficznego interfejsu użytkownika dla aplikacji. Android udostępnia również inne moduły interfejsu użytkownika do specjalnych interfejsów, takich jak okna dialogowe, powiadomienia i menu. Definiować menu będziemy w kolejnych kartach pracy.

Układy Layout

Układ systemu Android służy do definiowania interfejsu użytkownika, który zawiera kontrolki interfejsu użytkownika lub widżety, które pojawią się na ekranie aplikacji lub działania na Androida. Zasadniczo każda aplikacja jest połączeniem klas View i ViewGroup. Jak wiemy, aplikacja na Androida zawiera dużą liczbę aktywności i możemy powiedzieć, że każda aktywność to jedna strona aplikacji. Zatem każde działanie zawiera wiele komponentów interfejsu użytkownika, a komponenty te są instancjami View i ViewGroup.

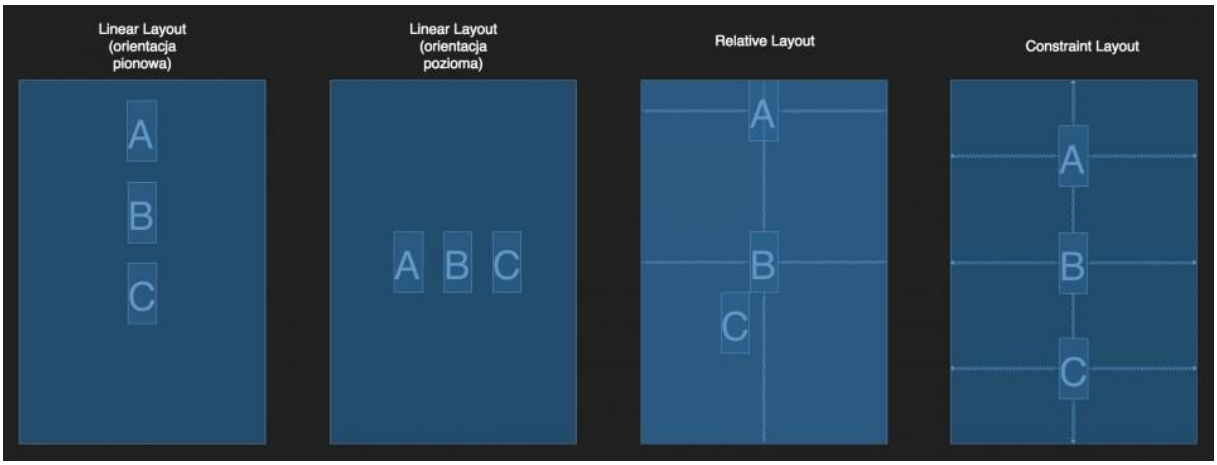
Widok jest zdefiniowany jako interfejs użytkownika, który jest używany do tworzenia interaktywnych elementów interfejsu użytkownika, takie jak TextView, EditText, przycisk opcji, itp i odpowiedzialny za obsługę zdarzeń i rysunku.

ViewGroup mogą zaś działać jako klasy bazowej dla układów i układach parametrówlub ViewGroups oraz określenie właściwości układu.

Layout definiuje wizualną strukturę interfejsu graficznego aplikacji, layouty deklarujemy na dwa sposoby:

- deklaracja w pliku XML
- deklaracja w pliku JAVA/KOTLIN

My na razie skupimy się na tworzeniu layoutów za pomocą XML. Wyobraźmy sobie layouty jako kontenery do których wkładamy następne elementy np.buttony czy pola tekstowe lub inne kontenery (layouty). Rozróżniamy kilka typów layoutów różnią się one tym jak rozkładane są w nich elementy.



- Linear Layout – najprostszy layout jaki mamy. Linear z angielskiego oznacza „liniowy" – ta nazwa idealnie oddaje jego działanie. Linear Layout po prostu układa elementy jeden za drugim. Wyróżniamy jego dwa warianty – Linear Layout z orientacją pionową, gdzie elementy układane są jeden pod drugim (zaczynając od góry) oraz z orientacją poziomą, gdzie elementy układane są od lewej do prawej.
- Relative Layout – słowo Relative możemy przetłumaczyć jako „względny" lub „zależny" – te dwa słowa również świetnie oddają jego działanie. W tym layoucie układamy widoki na podstawie położenia innych elementów. Dla przykładu jeśli mamy 3 widoki A, B oraz C to, chcąc ułożyć je jeden pod drugim, musimy powiedzieć systemowi coś w stylu: „Przypnij widok A do góry, widok B do widoku A, a widok C do widoku B". W ten sposób powstanie nam taka wieża. Relative Layout jest używany w sytuacjach, gdzie chcemy osiągnąć efekt niemożliwy do zrealizowania przy użyciu Linear Layout.
- Constraint Layout – powstał on stosunkowo niedawno – tak naprawdę jest to ulepszony Relative Layout. Każdy widok opisywany jest przez tak zwane constrainty, czyli opis tego w jaki sposób widok ma być przypięty do innego widoku. Na początku nauka Constraint Layouta może wydawać się ciężka, ale Android Studio posiada tryb graficzny, w którym możesz zdefiniować średnio zaawansowany interfejs użytkownika używając jedynie myszki.

W Androidzie praktycznie nie używamy pikseli jako jednostki wielkości. Zamiast tego używamy jednostki „dp" czyli „density independent pixel" co można przetłumaczyć na pixel niezależny od gęstości. Tworząc interfejs deklarujemy na przykład, że nasz przycisk będzie miał 30dp szerokości i Android dba o to, żeby ten przycisk był jednakowej wielkości zarówno na urządzeniu z rozdzielczością HVGA jak i 4K.

Układ w XML

Zapoznajmy się atrybutami layoutów XML.

1. android: id - Służy do określania identyfikatora widoku.
2. android: layout_width - Służy do deklarowania szerokości elementów View i ViewGroup w układzie.
3. android: layout_height - Służy do deklarowania wysokości elementów View i ViewGroup w układzie.

4. android: layout_marginLeft - Służy do deklarowania dodatkowej przestrzeni używanej po lewej stronie elementów View i ViewGroup.
5. android: layout_marginRight - Służy do deklarowania dodatkowej przestrzeni używanej po prawej stronie elementów View i ViewGroup.
6. android: layout_marginTop - Służy do deklarowania dodatkowej przestrzeni używanej w górnej części elementów View i ViewGroup.
7. android: layout_marginBottom - Służy do deklarowania dodatkowej przestrzeni używanej w dolnej części elementów View i ViewGroup.
8. android: layout_gravity - Służy do definiowania położenia widoków potomnych w układzie.

Sprawdzamy w praktyce LinearLayout

Z LinearLayout mieliśmy już okazję spotkać się w poprzedniej karcie pracy.

Deklaracja :

```
<?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="vertical" >

   // Add another child elements here like
   // EditText, button etc

</LinearLayout>
```

1. Utwórzmy nowy project o nazwie LinearLayoutApp. Program powinien składać się z widżetów TextView, EditText oraz Button.
2. Przykładowy kod, który może pomóc zrozumieć ideę LinearLayout.

```
<?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:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity">

   <TextView
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_margin="16dp"
      android:text="Enter your name here:"
      android:textSize="24dp"
      android:id="@+id/txtVw"/>

   <EditText
      android:id="@+id/editText"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_margin="16dp"
```

```
    android:hint="Name"
    android:inputType="text"/>

  <Button
    android:id="@+id/showInput"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:text="show"
    android:backgroundTint="@color/colorPrimary"
    android:textColor="@android:color/white"/>

</LinearLayout>
```

3.W pliku MainActivity.kt zadeklaruj widżety o nazwie showButton, editText oraz textView.

```
  class MainActivity : AppCompatActivity() {

  override fun onCreate(savedInstanceState: Bundle?)
  {
    super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)
      // finding the UI elements
      val showButton = findViewById<Button>(R.id.showInput)
      val editText = findViewById<EditText>(R.id.editText)
     val textView = findViewById<TextView>(R.id.txtVw)
  }
}
```

4. Nie zapomnij o importach.

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.EditText
import android.widget.TextView
```

5. Zaproponuj stworzenie własnej aplikacji wykorzystując wiedzę z poprzednich zadań.

Sprawdzamy w praktyce RelativeLayout

Deklaracja:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingLeft="10dp"
  android:paddingRight="10dp">

  // Add other view or ViewGroup here
</RelativeLayout>
```

2. Utwórz nowy project o nazwie RelativeLayoutApp. Program powinien składać się z widżetów z 2 TextView, 2 EditText oraz Button. Przeanalizuj poniższy kod xml.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingLeft="10dp"
   android:paddingRight="10dp">

   <TextView
      android:id="@+id/textView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
      android:text="First name:"
      android:layout_marginTop="20dp"
      android:textSize="20dp"/>

   <EditText
      android:id="@+id/editText1"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_toRightOf="@id/textView1"
      android:layout_marginTop="8dp"/>

   <TextView
      android:id="@+id/textView2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
      android:layout_below="@+id/textView1"
      android:layout_marginTop="10dp"
      android:text="Last name:"
      android:textSize="20dp"/>

   <EditText
      android:id="@+id/editText2"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_centerHorizontal="true"
      android:layout_toRightOf="@id/textView2"
      android:layout_marginTop="45dp"/>

   <Button
      android:id="@+id/btn4"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_centerHorizontal="true"
      android:layout_below="@id/textView2"
      android:layout_marginTop="20dp"
      android:text="Submit" />
```

```
</RelativeLayout>
```

3. Nie zapomnij o importach:

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
```

4. Klasa MainActivity powinna wyglądać tak:

```
class MainActivity : AppCompatActivity() {

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

      // below access the UI elements


   }
}
```

5. Zaproponuj stworzenie własnej aplikacji wykorzystując wiedzę z poprzednich zadań.

Style i tematy:

Style i motywy w Androidzie pozwalają oddzielić szczegóły projektu aplikacji od struktury i zachowania interfejsu użytkownika, podobnie jak arkusze stylów w projektowaniu stron internetowych.

Styl jest zbiorem atrybutów określających wygląd pojedynczej View. Styl może określać atrybuty, takie jak kolor czcionki, rozmiar czcionki, kolor tła i wiele innych.

Motyw to rodzaj stylu, który jest stosowany do całej aplikacji, działalności, lub widoku hierarchii, a nie tylko pojedynczego widoku. Gdy zastosujesz styl jako motyw, każdy widok w aplikacji lub działaniu zastosuje każdy obsługiwany atrybut stylu. Motywy mogą także stosować style do elementów nie wyświetlających, takich jak pasek stanu i tło okna.

Style i motywy są zadeklarowane w stylu pliku zasobów res/values/, zwykle o nazwie styles.xml.

Ważniejsze informacje:

Kolor : Istnieją różne atrybuty związane z kolorem. Poniżej znajduje się opis wszystkich tych atrybutów:

- colorPrimary: jest to podstawowy kolor używany w twojej aplikacji. To będzie twoja podstawa lub możesz powiedzieć kolor nadrzędny.
- colorOnPrimary: służy do elementów umieszczonych nad kolorem podstawowym. Jest to zatem sprzeczne z kolorem podstawowym. Jest to bardzo ważne, ponieważ jeśli ustawisz kolor podstawowy i element znajdujący się w kolorze podstawowym na taki sam, spowoduje to złe wrażenia użytkownika, ponieważ użytkownik nie będzie w stanie rozróżnić tych dwóch.
- colorPrimaryVarient: Jest to nieco jaśniejszy lub ciemniejszy wariant twojego podstawowego koloru.
- colorSecondary: jest to kolor najczęściej używany po kolorze podstawowym w dowolnej aplikacji.

- colorOnSecondary: Jest używany, gdy chcesz ustawić kolor takiego elementu, który jest obecny na górze jakiegoś dodatkowego koloru.
- colorSecondaryVarient: Jest to nieco jaśniejszy lub ciemniejszy wariant twojego drugiego koloru.
- colorSurface: Jest używany, gdy chcesz ustawić kolor powierzchni. Ta powierzchnia może być podobna do Arkuszy Materiałów.
- colorOnSurface: ten kolor jest używany dla elementów obecnych na powierzchni colorSurface.
- colorError: Ten kolor służy do wyświetlania niektórych komunikatów o błędach. Zasadniczo ma kolor czerwony.
- colorOnError: Ten kolor jest używany dla elementów wyświetlanych na górze colorError.
- colorBackground: ten kolor jest używany dla treści, która znajduje się za wszystkimi innymi treściami ekranów.
- colorOnBackground: służy do elementów umieszczonych na górze colorBackground.

Możesz użyć powyższych kolorów w motywie aplikacji, używając poniższego kodu:

```
<style name="AppTheme" parent="Theme.MaterialComponents.Light">
   <item name="colorPrimary">#0336ff</item>
   <item name="colorOnPrimary">#ffffff</item>
   <item name="colorPrimaryVariant">#0035c9</item>
   <item name="colorSecondary">#ffde03</item>
   <item name="colorOnSecondary">#000000</item>
   <item name="colorSecondaryVariant">#ffc000</item>
   <item name="colorSurface">#ffffff</item>
   <item name="colorOnSurface">#000000</item>
   <item name="colorError">#b00020</item>
   <item name="colorOnError">#FFFFFF</item>
   <item name="android:colorBackground">@color/background</item>
   <item name="colorOnBackground">#212121</item>
</style>
```

TextView

TextView to podstawowy element interfejsu użytkownika, który pomaga wyświetlać tekst użytkownikowi. Treść tekstu możemy wpisać bezpośrednio w kodzie aplikacji w pliku klasy .kt albo w pliku definiującym interfejs .xml. Ta druga opcja jest polecana.

TextView ma wiele właściwości, które określają sposób jego wyświetlania w działaniu.

- layout_width: wrap_content - Szerokość TextView jest zawijana do szerokości treści
- layout_height: wrap_content - Wysokość TextView jest zawijana do wysokości treści
- tekst: „Witaj świecie!” - to tekst, który powinien być wyświetlany w Activity for TextView

Przykład z poziomu pliku activity_main.xml:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Jeśli chcielibyśmy tekst wyjustować to proponuję zrobić w ten sposób, ustawiając margines wewnętrzny:

```
android:padding="20sp"
android:justificationMode="inter_word"
```

Wpis "na sztywno" – bezpośrednio z kodu w MainActivity.kt wyglądałby tak :

```
val tv_dynamic = TextView(this)
tv_dynamic.textSize = 12f
tv_dynamic.text = "Hello World!"
// add TextView to LinearLayout
    ll_main_layout.addView(tv_dynamic)
```

Dodatkowo musimy zadeklarować „ll_main_layout.addView(tv_dynamic)", że utworzony TextView jest dołączany na końcu wszystkich widoków potomnych obecnych w LinearLayout.

Możemy dodawać ciągi w pliku strings.xml, dzięki czemu łatwo można używać ich w innych plikach, wywołując je z ich nazwami.

```
<resources>
  <string name="app_name">TextViewInKotlinApp</string>
  <string name="text_view">Devillecloud</string>
  <string name="text_on_click">Hello World</string>
</resources>
```

Wtedy w pliku xml odwołujemy się za pomocą : android:text="@string/text_view" . W miejsce android:text musimy się odwołać do nazwy stringa np. app_name, text_view itp, które znajdują się w pliku strings.xml.

```
<TextView
    android:id="@+id/text_view_id"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:text="@string/text_view"
    android:textColor="#008000"
    android:textSize="40dp"
    android:textStyle="bold"/>
</LinearLayout>
```

Tekst można formatować np. ustawiając mu kolor, rozmiar, oraz style np. pogrubionej czcionki.

```
android:textColor="#008000"
```

android:textSize="40dp"
android:textStyle="bold"

1. Zaproponuj stworzenie własnej aplikacji wykorzystując wiedzę z poprzednich zadań. Aplikacja powinna składać się z ImageView z Twoim zdjęciem profilowym w górnej części aplikacji z marginesami wewnętrznymi. Nie co niżej powinien znajdować się TextView z twoim opisem/biografią z różnym formatowaniem czcionki np. kolorem, stylem.

AlertDialog

Klasa Android AlertDialog służy do wyświetlania okna dialogowego ostrzegającego użytkownika za pomocą przycisków dodatnich i ujemnych. Przycisk dodatni służy do kontynuowania określonej akcji. Przycisk ujemny służy do odrzucenia zaalarmowanego działania. Możesz podać własny kod po kliknięciu przycisku dodatniego lub ujemnego.
AlertDialog pojawia się na górze układu aktywności. Nie możesz fizycznie uzyskiwać dostępu do innych elementów interfejsu użytkownika innych niż okno dialogowe Alert. AlertDialog można utworzyć tylko w wątku interfejsu użytkownika.Okno dialogowe alertu zawiera następujące trzy elementy:Tytuł,Wiadomość, Przyciski (które wyzwalają działania negatywne i pozytywne)

Sprawdźmy w praktyce AlertDialog:

1.Utwórz program o nazwie AlertDialogApp oraz zdefiniuj widżet Button przy wykorzystaniu LinearLayour. Po wciśnięciu przycisku powinien wyświetlić się AlertDialog. Button powinien zawierać text: Show Me Alert Dialog.
2. Wzorcowy plik .xml możesz zobaczyć 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:id="@+id/root_layout"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  android:padding="16dp"
  >
  <Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Show Me Alert Dialog"
    />
</LinearLayout>
```

2. W aplikacji chcemy by AlertDialog pytał się nas czy chcemy zmienić kolor tła naszej aplikacji na czerwony. Jeśli tak to musimy ustawić kolor tła z poziomu kodu w MainActivity na czerwony.

```
    // Set a click listener for button widget
    button.setOnClickListener{
      val builder = AlertDialog.Builder(this@MainActivity)
      builder.setTitle("App background color")
      builder.setMessage("Are you want to set the app background color to RED?")
```

```
        builder.setPositiveButton("YES"){dialog, which ->
            Toast.makeText(applicationContext,"Ok, we change the app
background.",Toast.LENGTH_SHORT).show()

            root_layout.setBackgroundColor(Color.RED)
        }
        builder.setNegativeButton("No"){dialog,which ->
            Toast.makeText(applicationContext,"You are not agree.",Toast.LENGTH_SHORT).show()
        }
        builder.setNeutralButton("Cancel"){_,_ ->
            Toast.makeText(applicationContext,"You cancelled the
dialog.",Toast.LENGTH_SHORT).show()
        }
        val dialog: AlertDialog = builder.create()
        dialog.show()
    }
  }
}
```

3. Sprawdź czy Twój program działa poprawnie. Sprawdź czy kod znajduje się klasie MainActivity oraz czy dołączyłeś poprawnie importy.

Dodatkowe zadania:

1. **GridLayout** https://www.youtube.com/watch?v=H6F54t7hzkc
2. **NightMode Theme – tryb nocny w aplikacji Android** https://www.youtube.com/watch?v=3pKqA3YvEaQ