

LABORATORIUM nr 2 – Aplikacje mobilne (2 fragmenty)

Polecenie:

Wykonać aplikację mobilną wczytującą dane liczbowe oraz sortującą je zgodnie z wybranymi metodami sortowania. Aplikacja ma być wykonana w technologii obiektowej i umożliwiać wybór metody sortowania. Należy zaimplementować dwie wybrane metody sortowania:

- bąbelkowe,
- przez wstawianie,
- przez scalanie,
- przez wybór,
- szybkie

Efekty działania aplikacji:

- podanie metody sortowania
- krótki opis metody sortowania
- początkowy ciąg
- posortowany ciąg
- krótkie sprawozdanie w wersji elektronicznej

Należy wykorzystać poznane elementy Android Studio i języka Kotlin.

Aplikacja ma być wykonana w dwóch wersjach:

1. Zastosowanie dwóch aktywności (activity)

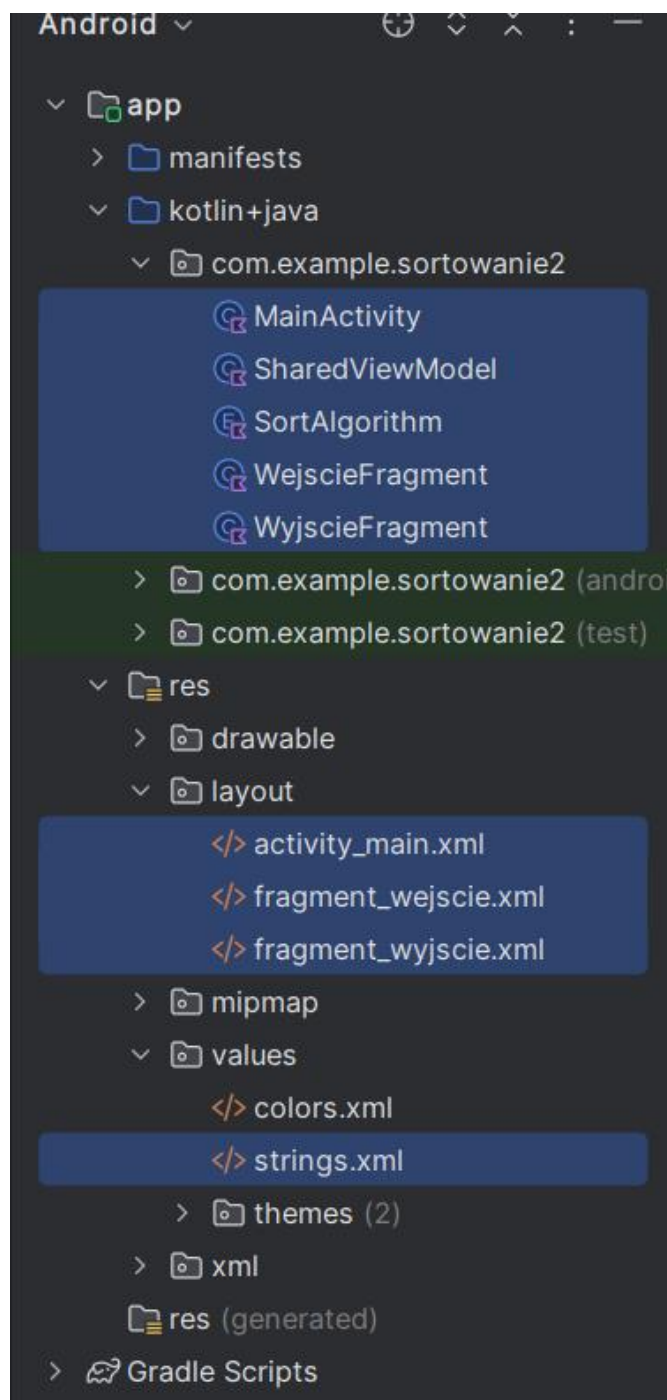
- a. Główna aktywność ustala co mamy do wykonania i przekazuje parametry do aktywności podrzędnej
- b. Aktywność wywoływana wykonuje wszystkie działania i wizualizuje efekty.

2. Zastosowanie jednej aktywności i dwóch fragmentów

- a. Pierwszy fragment ustala co trzeba wykonać
- b. Drugi fragment wykonuje zadanie i wizualizuje efekty

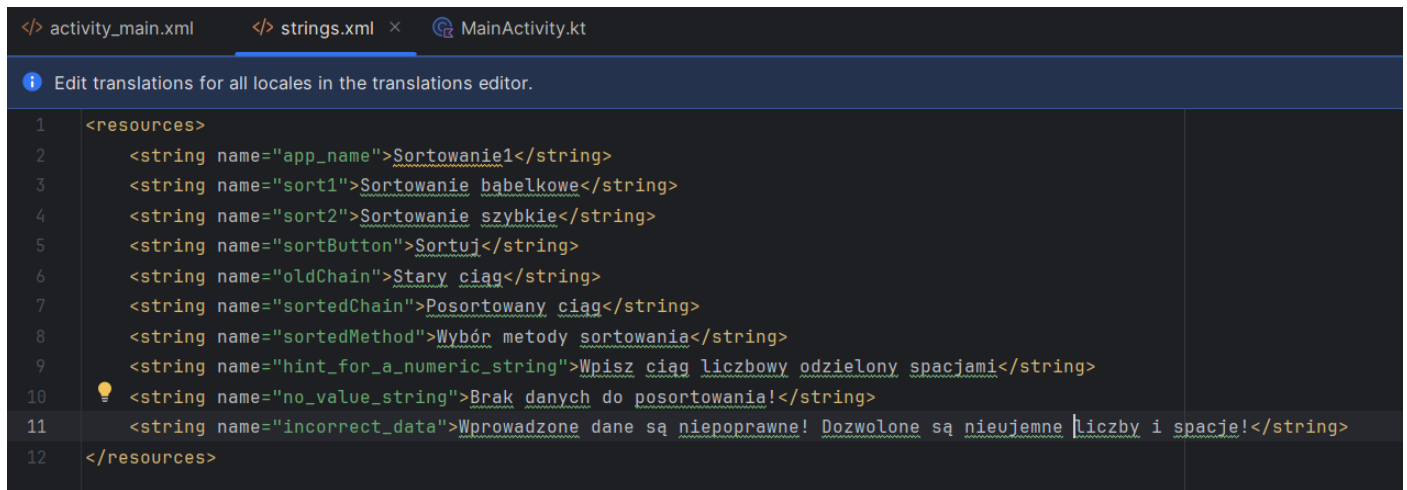
Opis

Podczas tworzenia aplikacji korzystano z wielu plików Android Studio, które zostały wyszczególnione niebieskim kolorem na Rys 1.



Rys. 1. Struktura projektu

W pliku *strings.xml* zainicjalizowano kilka ciągów znaków, które wykorzystano do tworzenia treści aplikacji.

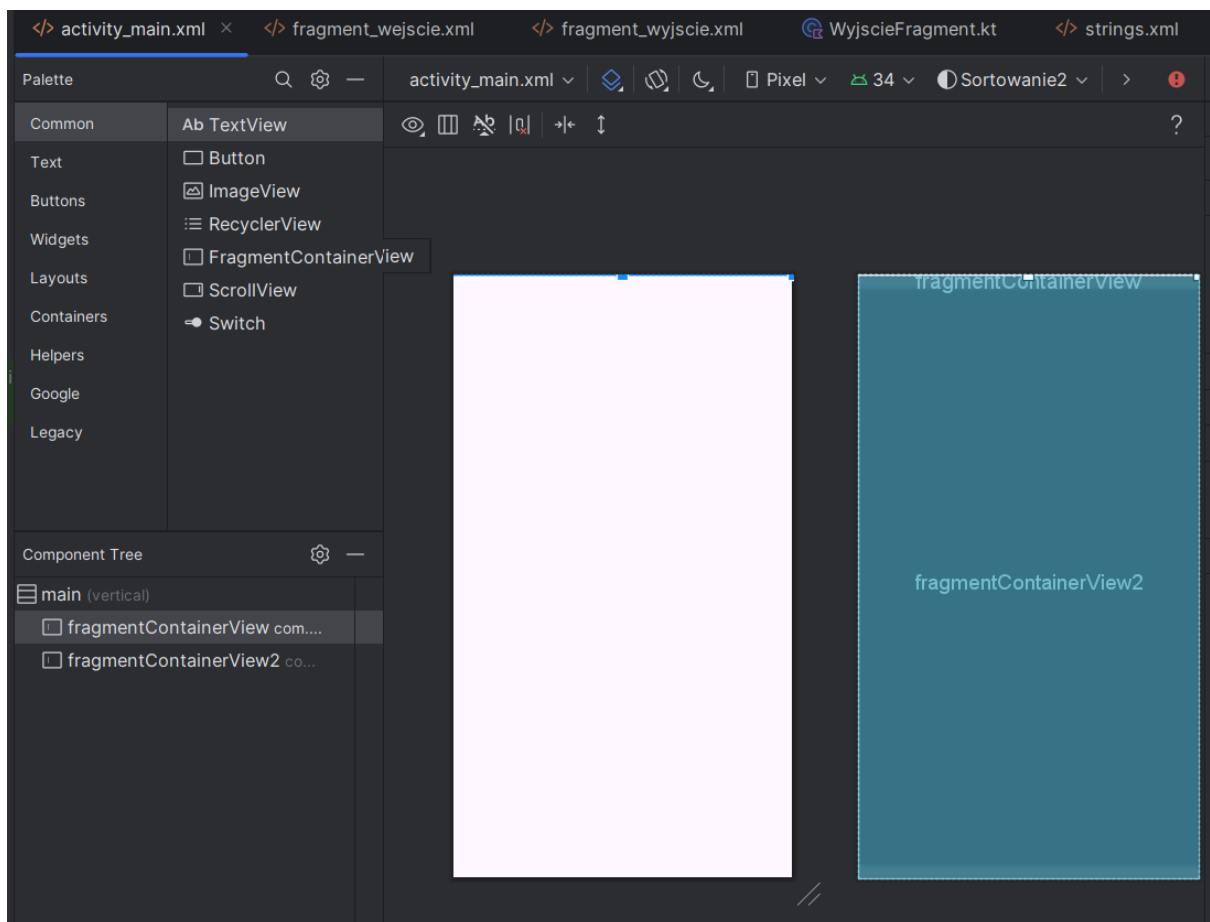


```
1 <resources>
2   <string name="app_name">Sortowanie1</string>
3   <string name="sort1">Sortowanie bąbelkowe</string>
4   <string name="sort2">Sortowanie szybkie</string>
5   <string name="sortButton">Sortuj</string>
6   <string name="oldChain">Stary ciąg</string>
7   <string name="sortedChain">Posortowany ciąg</string>
8   <string name="sortedMethod">Wybór metody sortowania</string>
9   <string name="hint_for_a_numeric_string">Wpisz ciąg liczbowy oddzielony spacjami</string>
10  <string name="no_value_string">Brak danych do posortowania!</string>
11  <string name="incorrect_data">Wprowadzone dane są niepoprawne! Dozwolone są nieujemne liczby i spacje!</string>
12 </resources>
```

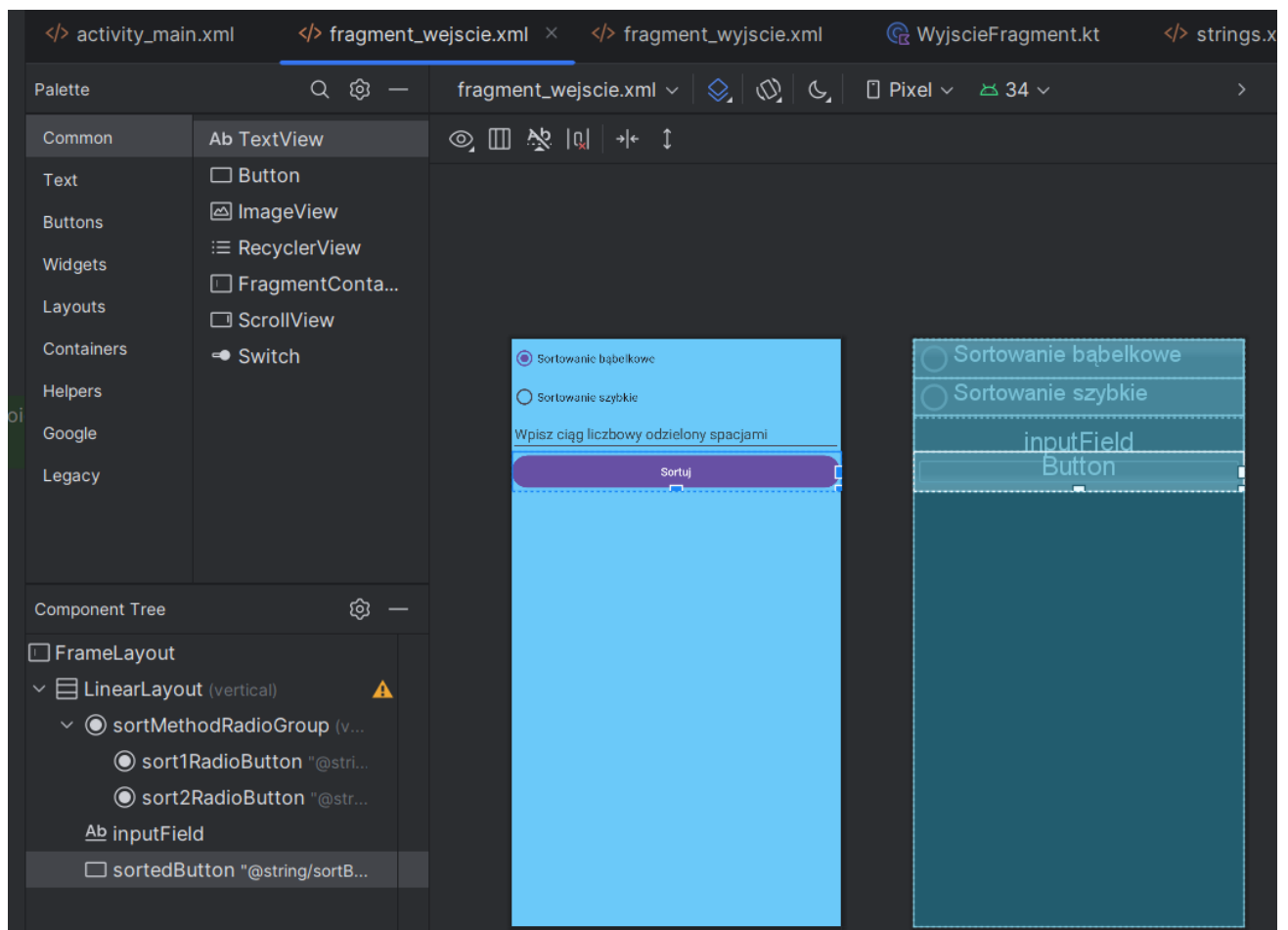
Rys. 2. Podgląd zawartości pliku strings.xml

Do pliku *activity_main.xml* dołączono 2 fragmenty, które są odpowiedzialne za wyświetlanie odpowiednich komponentów graficznych.

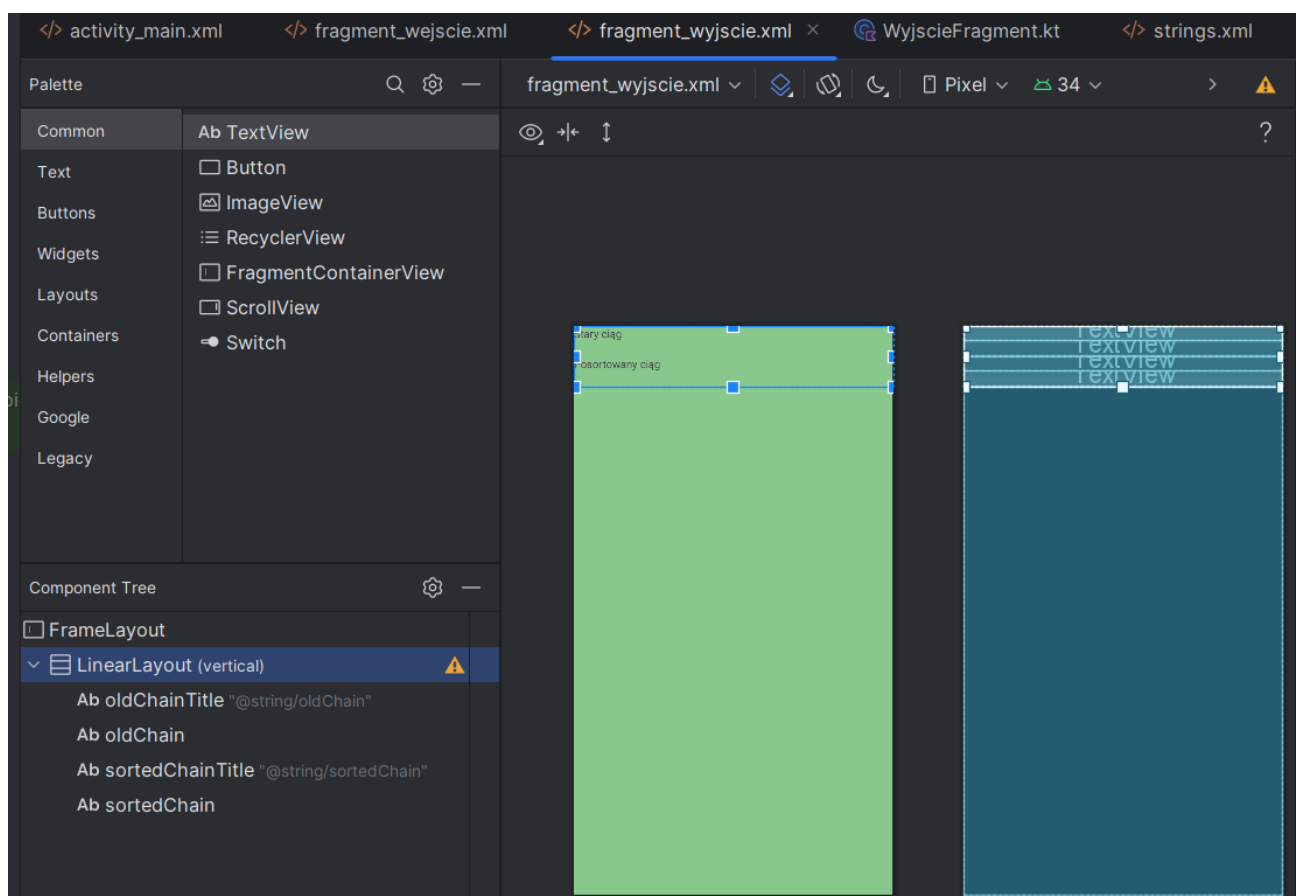
W plikach *fragment_wejście.xml* i *fragment_wyjście.xml* opisano właściwości graficzne aplikacji. Dodano m.in. pola tekstowe wyświetlające, pole tekstowe pobierającą wartość od użytkownika, 2 przyciski typu radio button oraz zwykły przycisk, który inicjalizuje wykonanie operacji.



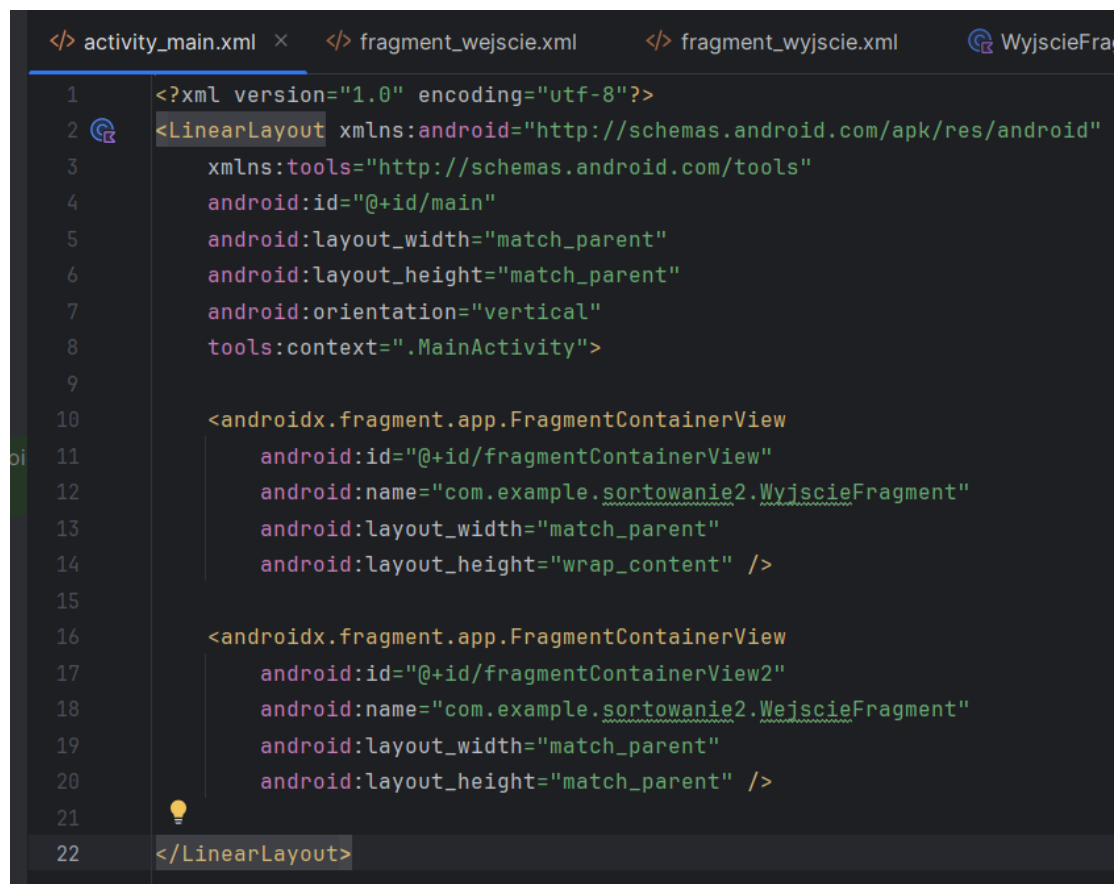
Rys. 3. Podgląd kreatora części wizualnej - *activity_main.xml*.



Rys. 4. Podgląd kreatora części wizualnej - *fragment_wejscie.xml*.



Rys. 5. Podgląd kreatora części wizualnej - i *fragment_wyjście.xml*.



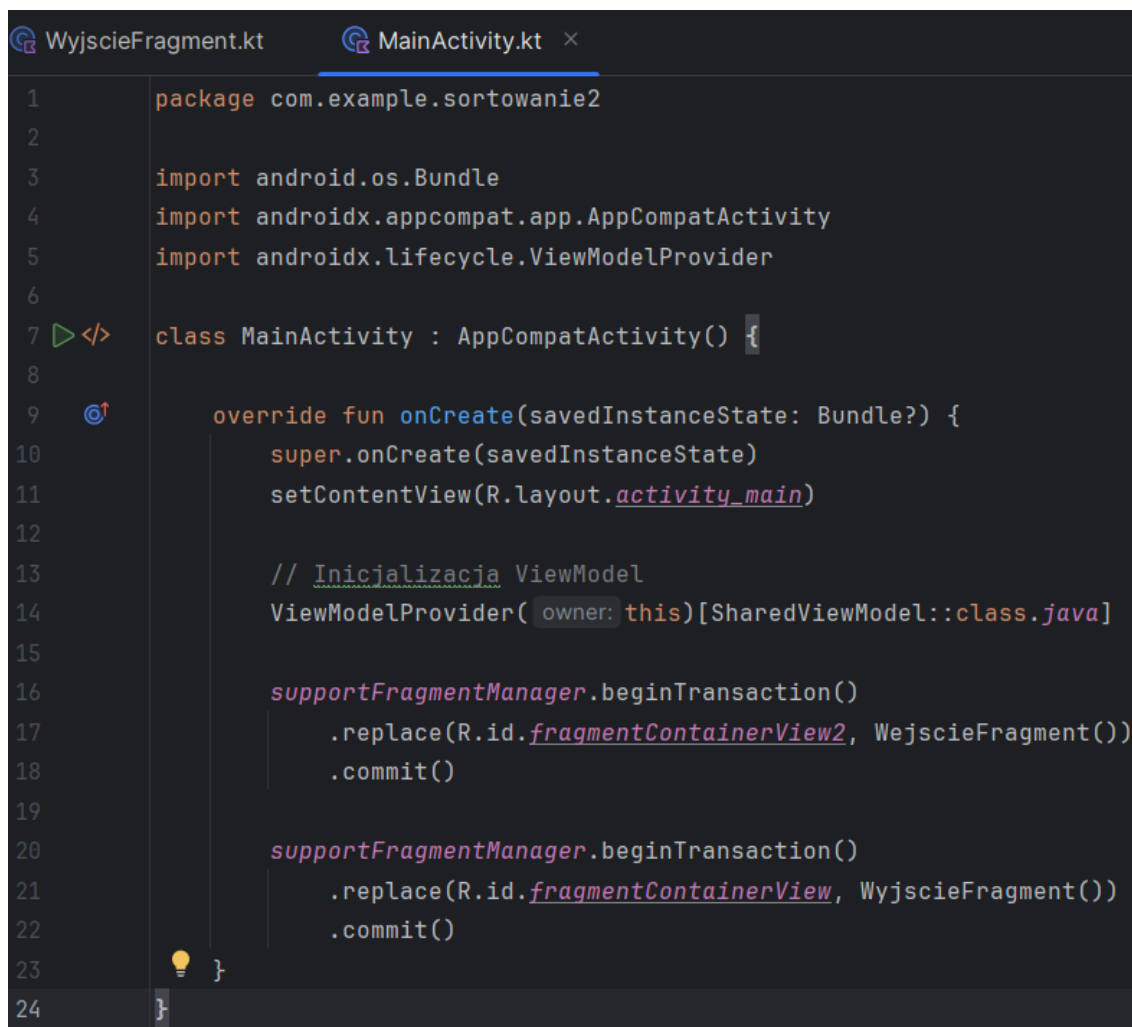
```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:tools="http://schemas.android.com/tools"
4     android:id="@+id/main"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     android:orientation="vertical"
8     tools:context=".MainActivity">
9
10    <androidx.fragment.app.FragmentContainerView
11        android:id="@+id/fragmentContainerView"
12        android:name="com.example.sortowanie2.WyjścieFragment"
13        android:layout_width="match_parent"
14        android:layout_height="wrap_content" />
15
16    <androidx.fragment.app.FragmentContainerView
17        android:id="@+id/fragmentContainerView2"
18        android:name="com.example.sortowanie2.WejścieFragment"
19        android:layout_width="match_parent"
20        android:layout_height="match_parent" />
21
22 </LinearLayout>
```

Rys. 6. Podgląd fragmentu kodu z pliku activity_main.xml

Jak widać, rysunki nr 2 - 6 dowodzą, że tworzenie elementów aplikacji możliwe jest zarówno z poziomu kodu XML, jak i kreatora elementów.

W plikach `.kt` zaimplementowano mechanikę aplikacji.

Plik `MainActivity.kt` jest odpowiedzialny za inicjalizację `ViewModel` oraz za dodanie dwóch fragmentów.



```
1 package com.example.sortowanie2
2
3 import android.os.Bundle
4 import androidx.appcompat.app.AppCompatActivity
5 import androidx.lifecycle.ViewModelProvider
6
7 class MainActivity : AppCompatActivity() {
8
9     override fun onCreate(savedInstanceState: Bundle?) {
10         super.onCreate(savedInstanceState)
11         setContentView(R.layout.activity_main)
12
13         // Inicjalizacja ViewModel
14         ViewModelProvider(owner: this)[SharedViewModel::class.java]
15
16         supportFragmentManager.beginTransaction()
17             .replace(R.id.fragmentContainerView2, WejscieFragment())
18             .commit()
19
20         supportFragmentManager.beginTransaction()
21             .replace(R.id.fragmentContainerView, WyjscieFragment())
22             .commit()
23     }
24 }
```

Rys. 7. Podgląd fragmentu kodu pliku `MainActivity.kt` napisanego w języku Kotlin

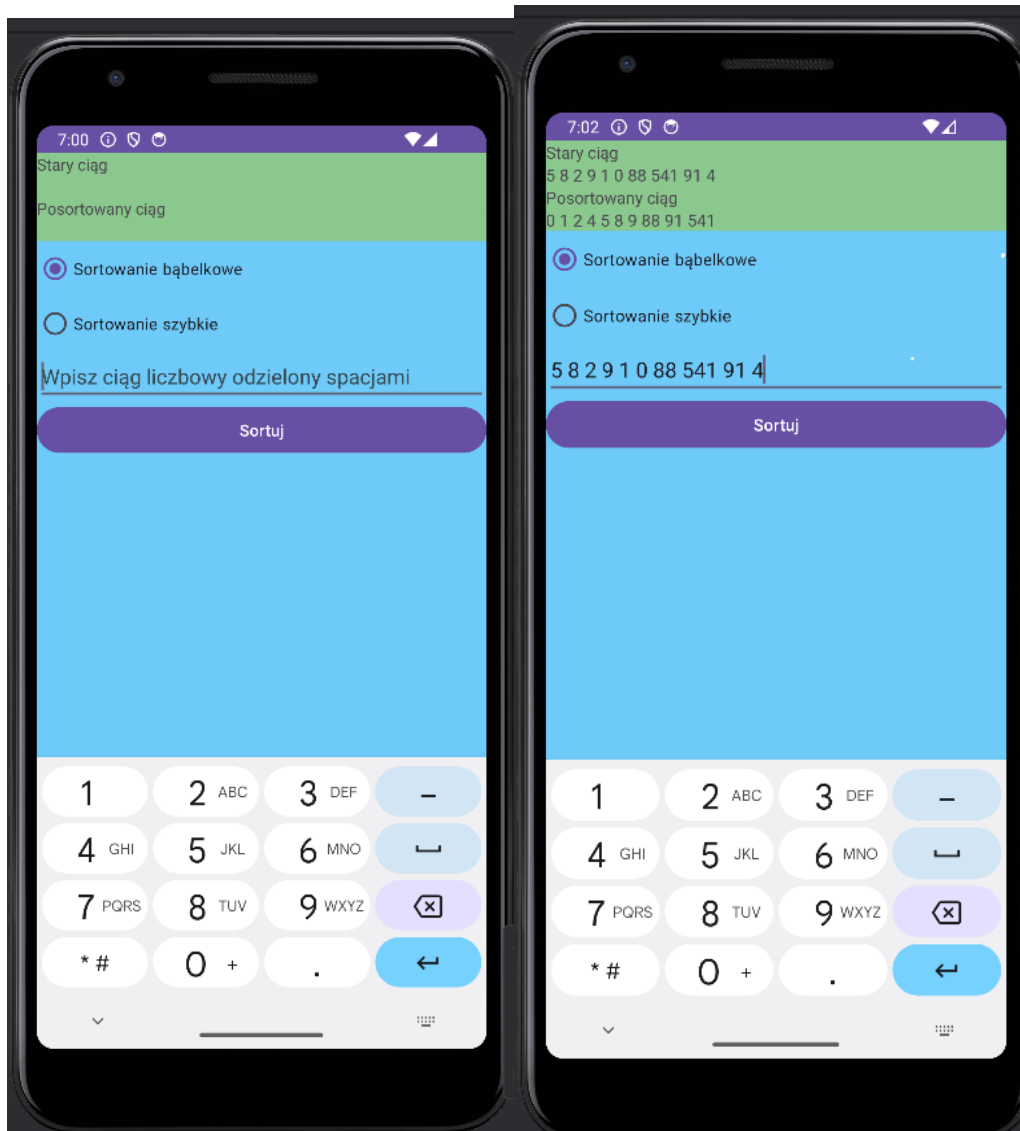
Kolejne opisy plików:

`SharedViewModel.kt` - Jest odpowiedzialny za pośredniczenie w wymianie danych pomiędzy fragmentami.

`SortAlgorithm.kt` - przyjmuje wartości Enum, oznaczając przez nie wybrany algorytm sortowania.

`WejscieFragment.kt` - pobiera dane z wejściowego fragmentu tj. ciąg liczbowy, wybrany algorytm, po czym wysyła je do klasy pośredniczącej.

`WyjscieFragment.kt` - pobiera dane od pośrednika, po czym wykonuje stosowne działania w oparciu o wybrany algorytm sortowania. Zapisuje wyniki do odpowiednich etykiet w części graficznej fragmentu wyjściowego.



Rys. 8 i 9. Kolejno widok uruchomionej aplikacji i widok po kliknięciu przycisku sortowania.