|  |  |
| --- | --- |
| Logo AMW Gdynia | **AKADEMIA MARYNARKI WOJENNEJ**  **im. BOHATERÓW WESTERPLATTE**  **WYDZIAŁ MECHANICZNO-ELEKTRYCZNY**  **KATEDRA INFORMATYKI** |

**SPRAWOZDANIE**

**Sortowanie liczb   
wykorzystując jedną aktywność**

**Android studio**

|  |  |
| --- | --- |
|  | Wykonawca: |
|  | **Patrycja Przybysz** |
|  | Grupa: |
|  | **215IC B2** |

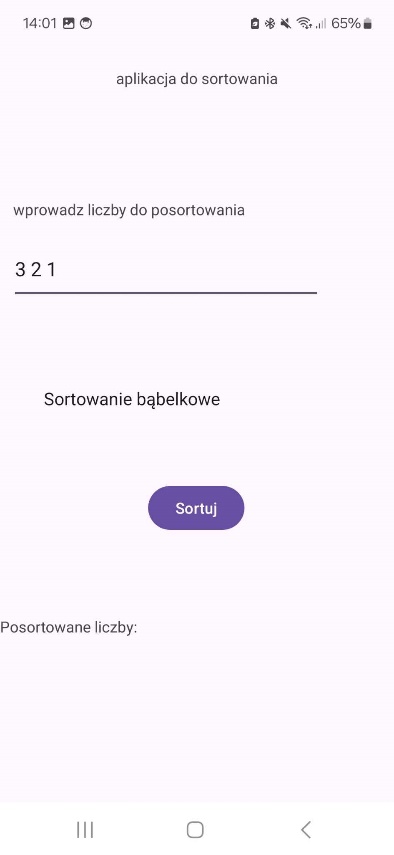
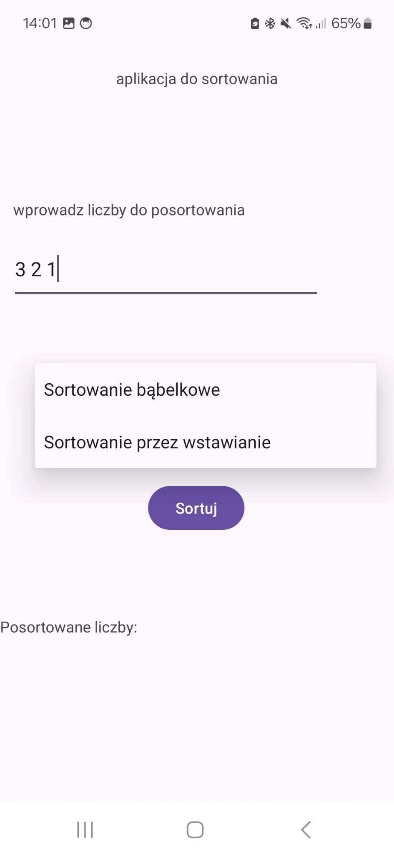
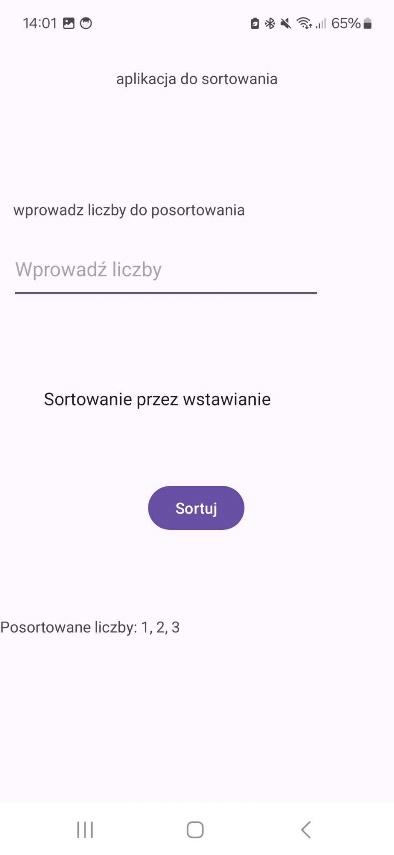
GDYNIA 2024

1. Projekt i architektura aplikacji

Projekt tej aplikacji składa się z dwóch głównych części: pliku XML definiującego układ interfejsu użytkownika oraz kodu źródłowego napisanego w języku Kotlin, który definiuje logikę działania aplikacji.

Plik xml **(activity\_main.xml)** definiuje układ interfejsu użytkownika za pomocą elementów dostępnych w bibliotece Android.Głównym kontenerem jest ConstraintLayout, który umożliwia definiowanie układu za pomocą ograniczeń.Na ekranie znajdują się:

* TextView wyświetlający tytuł aplikacji.
* TextView z instrukcją wprowadzenia liczb do posortowania.
* EditText do wprowadzania liczb przez użytkownika.
* Button do wywołania procesu sortowania.
* Spinner do wyboru metody sortowania.
* TextView do wyświetlenia posortowanych liczb.



Rysunek 1 wygląd aplikacji

Rysunek 3 wygląd aplikacji

Rysunek 2 wygląd aplikacji

**Architektura:**

Aplikacja wykorzystuje architekturę jednoaktywnościową (single-activity architecture), co oznacza, że większość lub cała logika aplikacji jest zawarta w jednej głównej aktywności (MainActivity). Aktywność ta obsługuje wszystkie zdarzenia interfejsu użytkownika i zarządza logiką aplikacji.

* Sortowanie liczb odbywa się po naciśnięciu przycisku "Sortuj", a metoda sortowania jest wybierana na podstawie wyboru użytkownika z listy rozwijanej (Spinner).
* Logika sortowania została oddzielona od interfejsu użytkownika i zaimplementowana w dedykowanych funkcjach (bubbleSort i insertionSort).
* Aplikacja korzysta z biblioteki Android Jetpack, w tym z modułów AppCompatActivity oraz ConstraintLayout, co umożliwia wygodne tworzenie interfejsu użytkownika oraz obsługę cyklu życia aplikacji.

1. Implementacja
2. **MainActivity.kt**:

* MainActivity jest podklasą klasy AppCompatActivity, co oznacza, że jest to aktywność, która dziedziczy wszystkie właściwości i metody klasy bazowej z biblioteki AppCompat.
* **editText, sortButton, sortSpinner i sortedTextView** to zmienne, które przechowują odwołania do elementów interfejsu użytkownika (np. EditText, Button, Spinner, TextView).
* Metoda **onCreate** jest wywoływana, gdy aktywność jest tworzona. Inicjalizuje ona interfejs użytkownika, ustawiając odpowiednie widoki na podstawie zasobów z pliku XML (R.layout.activity\_main). Inicjalizuje adapter dla Spinnera na podstawie zasobów tablicy łańcuchów (R.array.sort\_methods).
* Przycisk sortowania przypisuje metodę **addNumberToList** do obsługi kliknięcia przycisku. Ta metoda dodaje liczby wprowadzone przez użytkownika do listy i wywołuje funkcję sortowania.
* Wybór opcji sortowania ustawia obiekt **OnItemSelectedListener** dla Spinnera, który wywołuje metodę **sortNumbers** po wyborze nowej metody sortowania.
* Metoda **addNumberToList** pobiera tekst z pola tekstowego, dzieli go na pojedyncze liczby, konwertuje je na liczby całkowite i dodaje do listy **numbersList**. Wyświetla komunikat o błędzie, jeśli wprowadzone dane nie są liczbami całkowitymi.
* Metoda **sortNumbers** pobiera wybraną metodę sortowania z Spinnera i wywołuje odpowiednią funkcję sortowania (bubbleSort lub insertionSort). Wyświetla posortowane liczby w polu tekstowym.
* **bubbleSort i insertionSort** to metody, które realizują odpowiednio sortowanie bąbelkowe i przez wstawianie.

class MainActivity : AppCompatActivity() {  
  
 // Deklaracje pól zmiennych dla elementów interfejsu użytkownika  
 private lateinit var editText: EditText  
 private lateinit var sortButton: Button  
 private lateinit var sortSpinner: Spinner  
 private lateinit var sortedTextView: TextView  
 private val numbersList = *mutableListOf*<Int>()  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *enableEdgeToEdge*() // Włączenie funkcji ustawiania obszaru krawędziowego  
 setContentView(R.layout.*activity\_main*) // Ustawienie widoku dla aktywności z pliku XML  
  
 // Inicjalizacja elementów interfejsu użytkownika  
 editText = findViewById(R.id.*editText*)  
 sortButton = findViewById(R.id.*sortButton*)  
 sortSpinner = findViewById(R.id.*sortSpinner*)  
 sortedTextView = findViewById(R.id.*sortedTextView*)  
  
 // Pobranie opcji sortowania z zasobów i ustawienie adaptera dla spinnera  
 val sortMethods = *resources*.getStringArray(R.array.*sort\_methods*)  
 val sortAdapter = ArrayAdapter(this, android.R.layout.*simple\_spinner\_item*, sortMethods)  
 sortAdapter.setDropDownViewResource(android.R.layout.*simple\_spinner\_dropdown\_item*)  
 sortSpinner.*adapter* = sortAdapter  
  
 // Obsługa zdarzenia kliknięcia przycisku sortowania  
 sortButton.setOnClickListener **{** addNumberToList() // Dodanie liczb wprowadzonych przez użytkownika do listy  
 sortNumbers() // Posortowanie liczb i wyświetlenie wyniku  
 **}** // Obsługa zdarzenia wyboru opcji sortowania w spinnerze  
 sortSpinner.*onItemSelectedListener* = object : AdapterView.OnItemSelectedListener {  
 override fun onItemSelected(parent: AdapterView<\*>, view: View?, position: Int, id: Long) {  
 sortNumbers() // Ponowne posortowanie liczb po wybraniu nowej metody sortowania  
 }  
  
 override fun onNothingSelected(parent: AdapterView<\*>) {}  
 }  
 }  
  
 // Metoda dodająca liczby wprowadzone przez użytkownika do listy  
 private fun addNumberToList() {  
 val input = editText.*text*.toString() // Pobranie tekstu z pola tekstowego  
 val numbers = input.*split*("\\s+".*toRegex*()) // Podział tekstu na pojedyncze liczby  
 for (numberStr in numbers) {  
 if (numberStr.*isNotEmpty*()) {  
 try {  
 val number = numberStr.*toInt*() // Konwersja tekstu na liczbę całkowitą  
 numbersList.add(number) // Dodanie liczby do listy  
 } catch (e: NumberFormatException) {  
 editText.*error* = "Wprowadź liczby całkowite oddzielone spacjami" // Wyświetlenie komunikatu o błędzie  
 return  
 }  
 }  
 }  
 editText.*text*.clear() // Wyczyszczenie pola tekstowego po dodaniu liczb do listy  
 }  
  
 // Metoda sortująca liczby w zależności od wybranej metody sortowania  
 private fun sortNumbers() {  
 val sortMethod = sortSpinner.*selectedItem*.toString() // Pobranie wybranej metody sortowania  
 val sortedNumbers = when (sortMethod) {  
 "Sortowanie bąbelkowe" -> bubbleSort(numbersList.*toList*()) // Wywołanie metody sortowania bąbelkowego  
 "Sortowanie przez wstawianie" -> insertionSort(numbersList.*toList*()) // Wywołanie metody sortowania przez wstawianie  
 else -> *listOf*() // Domyślnie zwrócenie pustej listy  
 }  
 sortedTextView.*text* = "Posortowane liczby: ${sortedNumbers.*joinToString*(", ")}" // Wyświetlenie posortowanych liczb  
 }  
  
 // Metoda realizująca sortowanie bąbelkowe  
 private fun bubbleSort(list: List<Int>): List<Int> {  
 val sortedList = list.*toMutableList*() // Skopiowanie listy liczb do sortowania  
 for (i in 0 *until* sortedList.size - 1) {  
 for (j in 0 *until* sortedList.size - i - 1) {  
 if (sortedList[j] > sortedList[j + 1]) { // Porównanie sąsiednich liczb  
 val temp = sortedList[j]  
 sortedList[j] = sortedList[j + 1] // Zamiana miejscami liczb  
 sortedList[j + 1] = temp  
 }  
 }  
 }  
 return sortedList // Zwrócenie posortowanej listy  
 }  
  
 // Metoda realizująca sortowanie przez wstawianie  
 private fun insertionSort(list: List<Int>): List<Int> {  
 val sortedList = list.*toMutableList*() // Skopiowanie listy liczb do sortowania  
 for (i in 1 *until* sortedList.size) {  
 val key = sortedList[i]  
 var j = i - 1  
 while (j >= 0 && sortedList[j] > key) {  
 sortedList[j + 1] = sortedList[j] // Przesuwanie elementów większych od klucza  
 j--  
 }  
 sortedList[j + 1] = key // Wstawienie klucza w odpowiednie miejsce  
 }  
 return sortedList // Zwrócenie posortowanej listy  
 }  
}