|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Politechnika Bydgoska im. Jana i Jędrzeja Śniadeckich  Wydział Telekomunikacji, Informatyki i Elektrotechniki  al. prof. S. Kaliskiego 7, 85-796 Bydgoszcz | | | |  | |
| Przedmiot | **Programowanie urządzeń mobilnych** | | | Kierunek/Tryb |  |
| Nr laboratorium | 2 | Data wykonania |  | Grupa |  |
| Ocena |  | Data oddania |  | Imię Nazwisko |  |
| Nazwa ćwiczenia | Cykl życia aktywności | | |

**Cel ćwiczenia laboratoryjnego**

Celem ćwiczenia laboratoryjnego jest zapoznanie się z cyklem życia aktywności w aplikacjach mobilnych na platformie Android. Studenci mają za zadanie stworzyć aplikację, która będzie zarządzać stanem aktywności i wyświetlać historię zmian stanu na ekranie oraz generować logi w LogCat. Ćwiczenie ma na celu zrozumienie, jak różne metody cyklu życia aktywności wpływają na działanie aplikacji oraz jak można efektywnie zarządzać zasobami w trakcie jej działania.

**Zadanie**

Zadaniem studentów jest utworzenie nowego projektu w Android Studio, skonfigurowanie odpowiednich plików XML i Java, a następnie implementacja metod cyklu życia aktywności takich jak onCreate, onStart, onResume, onPause, onStop, onRestart, onDestroy, onSaveInstanceState oraz onRestoreInstanceState. Studenci mają również za zadanie obserwować zmiany stanów aplikacji w oknie LogCat oraz testować aplikację poprzez przełączanie się między aplikacjami i zmianę orientacji ekranu.

**Oczekiwane wyniki**

Oczekiwane wyniki to poprawne działanie aplikacji, która będzie w stanie zachować i przywracać stan interfejsu użytkownika podczas zmian orientacji ekranu oraz przełączania się między aplikacjami. Studenci powinni również zrozumieć, jak zapisywać i przywracać stan aplikacji za pomocą mechanizmów takich jak SharedPreferences i Bundle. Efektem końcowym będzie aplikacja, która demonstruje pełne zrozumienie cyklu życia aktywności w Androidzie oraz umiejętność zarządzania stanem aplikacji.

Cykl życia aktywności w Androidzie składa się z kilku stanów, które odzwierciedlają różne etapy istnienia aktywności. Każdy stan jest reprezentowany przez odpowiednią metodę cyklu życia. Oto szczegółowe omówienie tych stanów:

**onCreate()**

* Opis: Wywoływana, gdy aktywność jest tworzona po raz pierwszy.
* Działanie: Inicjalizuje podstawowe komponenty aktywności, takie jak widoki, dane i inne zasoby. Jest to miejsce, gdzie ustawia się układ interfejsu użytkownika za pomocą setContentView().
* Przykład: Inicjalizacja TextView i ustawienie układu.

**onStart()**

* Opis: Wywoływana, gdy aktywność staje się widoczna dla użytkownika.
* Działanie: Aktywność jest przygotowywana do wejścia w stan interakcji z użytkownikiem. Może być używana do odświeżania danych lub zasobów, które mogą być zmienione podczas bycia w tle.
* Przykład: Aktualizacja interfejsu użytkownika.

**onResume()**

* Wywoływana, gdy aktywność zaczyna wchodzić w interakcję z użytkownikiem.
* Aktywność jest w pełni widoczna i gotowa do interakcji z użytkownikiem. Jest to miejsce, gdzie można uruchomić animacje, odtwarzanie wideo lub inne działania wymagające pełnej uwagi użytkownika.
* Uruchomienie animacji.

**onPause()**

* Wywoływana, gdy aktywność przestaje wchodzić w interakcję z użytkownikiem.
* Aktywność jest częściowo widoczna, ale użytkownik nie może z nią w pełni wchodzić w interakcję. Może być używana do zapisywania danych lub zatrzymywania działań, które nie powinny działać w tle.
* Zatrzymanie odtwarzania wideo.

**onStop()**

* Wywoływana, gdy aktywność przestaje być widoczna dla użytkownika.
* Aktywność jest całkowicie niewidoczna. Może być używana do zwalniania zasobów, które nie są potrzebne, gdy aktywność nie jest widoczna.
* Zatrzymanie odtwarzania muzyki.

**onRestart()**

* Wywoływana, gdy aktywność jest ponownie uruchamiana po zatrzymaniu.
* Aktywność przechodzi z powrotem do stanu widocznego. Może być używana do ponownego inicjowania zasobów zwolnionych w onStop().
* Ponowne uruchomienie animacji.

**onDestroy()**

* Wywoływana, gdy aktywność jest niszczona.
* Aktywność jest usuwana z pamięci. Może być używana do zwalniania zasobów, które nie są już potrzebne.
* Zamykanie połączeń z bazą danych.

**onSaveInstanceState()**

* Wywoływana przed zniszczeniem aktywności, aby zapisać jej stan.
* Umożliwia zapisanie stanu aktywności, aby można go było przywrócić po ponownym utworzeniu aktywności.
* Zapisywanie stanu interfejsu użytkownika.

**onRestoreInstanceState()**

* Wywoływana po onCreate, jeśli aktywność jest przywracana po zniszczeniu.
* Umożliwia przywrócenie stanu aktywności zapisanego w onSaveInstanceState().
* Przywracanie stanu interfejsu użytkownika.

Cykl życia aktywności w Androidzie jest kluczowy dla zarządzania zasobami i zapewnienia płynnego działania aplikacji. Każdy stan cyklu życia ma swoje specyficzne zastosowanie i pozwala na odpowiednie reagowanie na zmiany w stanie aktywności. Dzięki zrozumieniu tych stanów możesz tworzyć bardziej efektywne i responsywne aplikacje.

**Przykład LifeCycle**

Aplikacja zarządza stanem i wyświetlać historię zmian stanu na ekranie i generuje logi w LogCat.

Utworzenie nowego projektu

1. Otwórz Android Studio i utwórz nowy projekt.
2. Wybierz “Empty Views Activity” jako szablon projektu.
3. Nazwij projekt np. LifeCycle.
4. Wprowadź zmiany w pliku activity\_main.xml

*<?*xml version="1.0" encoding="utf-8"*?>*<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:id="@+id/main"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".MainActivity">  
  
 <TextView  
 android:id="@+id/textview"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Hello World!"  
 android:textSize="24sp"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
</androidx.constraintlayout.widget.ConstraintLayout>

Wprowadź zmiany w pliku MainActivity.java:

package com.example.lifecycle;  
import android.os.Bundle;  
import androidx.activity.EdgeToEdge;  
import androidx.appcompat.app.AppCompatActivity;  
import android.util.Log;  
import android.widget.TextView;  
  
public class MainActivity extends AppCompatActivity {  
 private static final String *TAG* = "LifecycleTag";  
 private static final String *STATE\_HISTORY\_KEY* = "stateHistory";  
 private TextView textView;  
 private StringBuilder stateHistory;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 EdgeToEdge.*enable*(this);  
 setContentView(R.layout.*activity\_main*);  
 textView = findViewById(R.id.*textview*);  
 stateHistory = new StringBuilder();  
  
 if (savedInstanceState != null) {  
 String savedHistory = savedInstanceState.getString(*STATE\_HISTORY\_KEY*);  
 if (savedHistory != null) {  
 stateHistory.append(savedHistory);  
 textView.setText(savedHistory);  
 }  
 }  
  
 logAndAppend("onCreate called");  
 }  
  
 @Override  
 protected void onStart() {  
 super.onStart();  
 logAndAppend("onStart called");  
 }  
  
 @Override  
 protected void onResume() {  
 super.onResume();  
 logAndAppend("onResume called");  
 }  
  
 @Override  
 protected void onPause() {  
 super.onPause();  
 logAndAppend("onPause called");  
 }  
  
 @Override  
 protected void onStop() {  
 super.onStop();  
 logAndAppend("onStop called");  
 }  
  
 @Override  
 protected void onRestart() {  
 super.onRestart();  
 logAndAppend("onRestart called");  
 }  
  
 @Override  
 protected void onDestroy() {  
 super.onDestroy();  
 logAndAppend("onDestroy called");  
 }  
  
 @Override  
 protected void onSaveInstanceState(Bundle outState) {  
 super.onSaveInstanceState(outState);  
 outState.putString(*STATE\_HISTORY\_KEY*, stateHistory.toString());  
 logAndAppend("onSaveInstanceState called");  
 }  
  
 @Override  
 protected void onRestoreInstanceState(Bundle savedInstanceState) {  
 super.onRestoreInstanceState(savedInstanceState);  
 String savedHistory = savedInstanceState.getString(*STATE\_HISTORY\_KEY*);  
 if (savedHistory != null) {  
 stateHistory.append(savedHistory);  
 textView.setText(savedHistory);  
 }  
 logAndAppend("onRestoreInstanceState called");  
 }  
  
 private void logAndAppend(String message) {  
 Log.*d*(*TAG*, message);  
 stateHistory.append("\n").append(message);  
 textView.append("\n"+message);  
 }  
}

Po uruchomieniu aplikacji możemy zauważyć, że przed wywołaniem metody onCreate wyświetli się Layout z TextView i napisem HelloWorld! Możesz przełączać się między aplikacjami w emulatorze, obracać ekran i obserwować jakie stany przyjmuje nasza aplikacja. Dodatkowo można dodać filtr do Logcat i obserwować zmiany stanów. W tym celu wybierz:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

lub

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, numer

Opis wygenerowany automatycznie

Teraz ustaw filtr, wpisując ustalony ciąg, w naszym przypadku będzie to ‘LifecycleTag’

Obraz zawierający tekst, oprogramowanie, linia, zrzut ekranu

Opis wygenerowany automatycznie

Teraz możemy obserwować zmiany stanu aplikacji w oknie Logcat dzięki umieszczeniu w naszej aplikacji instrukcji Log.*d*(*TAG*, message) gdzie TAG to ciąg ‘LifecycleTag’. Zauważ że przy zmianie orientacji ekranu stan naszej aplikacji nie jest zachowywany.

**Przykład LifeCycleEditTextApp**

Następny przykład demonstruje sposób zachowania swojego stanu nawet po zamknięciu aplikacji lub jej ponownym uruchomieniu, możemy zapisać jego zawartość w preferencjach aplikacji. SharedPreferences to idealne narzędzie do tego celu w Androidzie, ponieważ pozwala na trwałe przechowywanie małych ilości danych.

Utworzenie nowego projektu

Otwórz Android Studio i utwórz nowy projekt.

Wybierz “Empty Views Activity” jako szablon projektu.

Nazwij projekt np. LifeCycleEditTextApp.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Wprowadź zmiany w pliku activity\_main.xml

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:orientation="vertical"  
 android:padding="16dp">  
   
 <EditText  
 android:id="@+id/editTextNote"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:hint="Enter your notes here"  
 android:textSize="16sp"  
 android:layout\_marginTop="16dp"/>  
 <TextView  
 android:id="@+id/textview"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Lifecycle States"  
 android:textSize="16sp" />  
   
</LinearLayout>

MainActivity,java

package com.example.lifecycleedittextapp;  
  
import android.content.SharedPreferences;  
import android.os.Bundle;  
import androidx.activity.EdgeToEdge;  
import androidx.appcompat.app.AppCompatActivity;  
import android.util.Log;  
import android.widget.EditText;  
import android.widget.TextView;  
  
public class MainActivity extends AppCompatActivity {  
 private static final String *TAG* = "LifecycleTag";  
 private static final String *STATE\_HISTORY\_KEY* = "stateHistory";  
 private static final String *NOTES\_KEY* = "notes";  
  
 private TextView textView;  
 private EditText editTextNote;  
 private StringBuilder stateHistory;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 EdgeToEdge.*enable*(this);  
 setContentView(R.layout.*activity\_main*);  
  
 textView = findViewById(R.id.*textview*);  
 editTextNote = findViewById(R.id.*editTextNote*);  
 stateHistory = new StringBuilder();  
  
 SharedPreferences preferences = getPreferences(*MODE\_PRIVATE*);  
 String savedNote = preferences.getString(*NOTES\_KEY*, "");  
 editTextNote.setText(savedNote);  
  
 if (savedInstanceState != null) {  
 String savedHistory = savedInstanceState.getString(*STATE\_HISTORY\_KEY*);  
 if (savedHistory != null) {  
 stateHistory.append(savedHistory);  
 textView.setText(savedHistory);  
 }  
 }  
  
 logAndAppend("onCreate called");  
 }  
  
 @Override  
 protected void onStart() {  
 super.onStart();  
 logAndAppend("onStart called");  
 }  
  
 @Override  
 protected void onResume() {  
 super.onResume();  
 logAndAppend("onResume called");  
 }  
  
 @Override  
 protected void onPause() {  
 super.onPause();  
  
 SharedPreferences preferences = getPreferences(*MODE\_PRIVATE*);  
 SharedPreferences.Editor editor = preferences.edit();  
 editor.putString(*NOTES\_KEY*, editTextNote.getText().toString());  
 editor.apply();  
  
 logAndAppend("onPause called");  
 }  
  
 @Override  
 protected void onStop() {  
 super.onStop();  
 logAndAppend("onStop called");  
 }  
  
 @Override  
 protected void onRestart() {  
 super.onRestart();  
 logAndAppend("onRestart called");  
 }  
  
 @Override  
 protected void onDestroy() {  
 super.onDestroy();  
 logAndAppend("onDestroy called");  
 }  
  
 @Override  
 protected void onSaveInstanceState(Bundle outState) {  
 super.onSaveInstanceState(outState);  
 outState.putString(*STATE\_HISTORY\_KEY*, stateHistory.toString());  
 logAndAppend("onSaveInstanceState called");  
 }  
  
 @Override  
 protected void onRestoreInstanceState(Bundle savedInstanceState) {  
 super.onRestoreInstanceState(savedInstanceState);  
 String savedHistory = savedInstanceState.getString(*STATE\_HISTORY\_KEY*);  
 if (savedHistory != null) {  
 stateHistory.append(savedHistory);  
 textView.setText(savedHistory);  
 }  
 logAndAppend("onRestoreInstanceState called");  
 }  
  
 private void logAndAppend(String message) {  
 Log.*d*(*TAG*, message);  
 stateHistory.append("\n").append(message);  
 textView.append("\n" + message);  
 }  
}

SharedPreferences są używane do trwałego zapisu zawartości EditText. Zawartość EditText jest wczytywana przy każdym uruchomieniu aplikacji (onCreate). Zapis do SharedPreferences odbywa się w onPause, co gwarantuje, że dane zostaną zapisane przed przejściem aplikacji do stanu uśpienia lub zakończenia. Dzięki temu zawartość pola tekstowego pozostanie bez zmian nawet po zamknięciu i ponownym uruchomieniu aplikacji.

**Przykład LifeCycleEditTextApp (Bundle)**

Aby osiągnąć efekt, że tekst z pola tekstowego będzie przechowywany tylko podczas sesji aplikacji i zniknie po usunięciu aplikacji z listy ostatnio otwartych (czyli po całkowitym zamknięciu procesu aplikacji), ale nadal będzie dostępny podczas normalnego cyklu życia aplikacji (np. przejście do tła i powrót), musimy zastosować zmienny mechanizm przechowywania danych.

Rozwiązanie polega na przechowywaniu zawartości tekstu w Bundle poprzez mechanizm onSaveInstanceState, zamiast w SharedPreferences. W ten sposób tekst będzie przechowywany podczas aktywności aplikacji (np. podczas zmiany orientacji ekranu), ale zostanie utracony po pełnym zamknięciu aplikacji.

Zaktualizuj MainActivity.java:

package com.example.lifecycleedittextapp;  
import android.content.SharedPreferences;  
import android.os.Bundle;  
import androidx.activity.EdgeToEdge;  
import androidx.appcompat.app.AppCompatActivity;  
import android.util.Log;  
import android.widget.EditText;  
import android.widget.TextView;  
  
import android.os.Bundle;  
import androidx.activity.EdgeToEdge;  
import androidx.appcompat.app.AppCompatActivity;  
import android.util.Log;  
import android.widget.EditText;  
import android.widget.TextView;  
  
public class MainActivity extends AppCompatActivity {  
 private static final String *TAG* = "LifecycleTag";  
 private static final String *STATE\_HISTORY\_KEY* = "stateHistory";  
 private static final String *NOTES\_KEY* = "notes";  
  
 private TextView textView;  
 private EditText editTextNote;  
 private StringBuilder stateHistory;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 EdgeToEdge.*enable*(this);  
 setContentView(R.layout.*activity\_main*);  
  
 textView = findViewById(R.id.*textview*);  
 editTextNote = findViewById(R.id.*editTextNote*);  
 stateHistory = new StringBuilder();  
  
 if (savedInstanceState != null) {  
 String savedNote = savedInstanceState.getString(*NOTES\_KEY*);  
 String savedHistory = savedInstanceState.getString(*STATE\_HISTORY\_KEY*);  
  
 if (savedNote != null) {  
 editTextNote.setText(savedNote);  
 }  
  
 if (savedHistory != null) {  
 stateHistory.append(savedHistory);  
 textView.setText(savedHistory);  
 }  
 }  
  
 logAndAppend("onCreate called");  
 }  
  
 @Override  
 protected void onStart() {  
 super.onStart();  
 logAndAppend("onStart called");  
 }  
  
 @Override  
 protected void onResume() {  
 super.onResume();  
 logAndAppend("onResume called");  
 }  
  
 @Override  
 protected void onPause() {  
 super.onPause();  
 logAndAppend("onPause called");  
 }  
  
 @Override  
 protected void onStop() {  
 super.onStop();  
 logAndAppend("onStop called");  
 }  
  
 @Override  
 protected void onRestart() {  
 super.onRestart();  
 logAndAppend("onRestart called");  
 }  
  
 @Override  
 protected void onDestroy() {  
 super.onDestroy();  
 logAndAppend("onDestroy called");  
 }  
  
 @Override  
 protected void onSaveInstanceState(Bundle outState) {  
 super.onSaveInstanceState(outState);  
 outState.putString(*NOTES\_KEY*, editTextNote.getText().toString());  
 outState.putString(*STATE\_HISTORY\_KEY*, stateHistory.toString());  
 logAndAppend("onSaveInstanceState called");  
 }  
  
 @Override  
 protected void onRestoreInstanceState(Bundle savedInstanceState) {  
 super.onRestoreInstanceState(savedInstanceState);  
 String savedNote = savedInstanceState.getString(*NOTES\_KEY*);  
 String savedHistory = savedInstanceState.getString(*STATE\_HISTORY\_KEY*);  
  
 if (savedNote != null) {  
 editTextNote.setText(savedNote);  
 }  
  
 if (savedHistory != null) {  
 stateHistory.append(savedHistory);  
 textView.setText(savedHistory);  
 }  
  
 logAndAppend("onRestoreInstanceState called");  
 }  
  
 private void logAndAppend(String message) {  
 Log.*d*(*TAG*, message);  
 stateHistory.append("\n").append(message);  
 textView.append("\n" + message);  
 }  
}

**onSaveInstanceState** przechowuje dane (notatkę i historię stanu) w obiekcie Bundle w momencie, gdy aktywność może zostać zniszczona (np. podczas zmiany orientacji ekranu lub wyjścia aplikacji na pierwszy plan).

**onRestoreInstanceState** odtwarza zapisane dane po ponownym utworzeniu aktywności.

Zamiast trwałego zapisu w pamięci urządzenia, tekst notatki jest przechowywany tylko podczas sesji aplikacji. Zostanie usunięty po pełnym zamknięciu aplikacji.

**Podczas zmiany orientacji ekranu z**awartość notesu będzie zachowywana, ponieważ onSaveInstanceState przechowuje dane w trakcie normalnego cyklu życia aktywności.

**Po zamknięciu aplikacji (usunięciu z listy ostatnich aplikacji)** tekst notatki zostanie utracony, ponieważ nie jest zapisany w pamięci trwałej, jak to było w przypadku SharedPreferences.

To podejście zapewnia zachowanie danych w trakcie działania aplikacji, ale usuwa je po pełnym zamknięciu.

**ZADANIA**

Zadanie

Zadaniem, które dobrze obrazuje mechanizmy działania aplikacji na Androida, jest prosty **licznik kliknięć**.

Aplikacja licznika kliknięć będzie miała przycisk, który po kliknięciu zwiększa licznik, oraz TextView, który wyświetla jego aktualną wartość. Wartość licznika zostanie zachowana podczas zmiany orientacji ekranu, a także po zamknięciu aplikacji.

Obraz zawierający tekst, zrzut ekranu, Czcionka, System operacyjny

Opis wygenerowany automatycznie

**Sprawozdanie**

Sprawozdanie z ćwiczenia w ramach nauki programowania aplikacji mobilnych powinno zawierać istotne informacje i dokumentację dotyczącą zadania oraz jego realizacji. Oto kilka kluczowych elementów, które powinny być uwzględnione w sprawozdaniu:

1. **Tytuł i informacje ogólne**
   * Tytuł ćwiczenia.
   * Imię i nazwisko studenta.
   * Data realizacji ćwiczenia.
2. **Cel ćwiczenia**
   * Krótka informacja o celu i znaczeniu ćwiczenia, jak również o tym, czego studenci mieli się nauczyć.
3. **Opis projektu**
   * Opis funkcjonalności projektowanej aplikacji (przeliczanie stopni Celsjusza na Fahrenheita).
   * Krótka charakteryzacja interfejsu użytkownika, jakie widoki i elementy zostały użyte.
4. **Implementacja**
   * Opis procesu tworzenia projektu.
   * Omówienie wykorzystanych narzędzi i technologii (Android Studio, język Java/Kotlin).
   * Przedstawienie kodu źródłowego aplikacji, zarówno XML (layout) jak i kodu Java/Kotlin.
5. **Funkcje** 
   * Omówienie kluczowych funkcji aplikacji, takich jak obsługa przycisku, przeliczanie temperatury, obsługa błędów.
6. **Testowanie**
   * Opis testowania aplikacji, w tym przykłady testów, jakie przeprowadzono.
   * Raport z wynikami testów.
7. **Wyniki**
   * Przedstawienie wyników działania aplikacji.
   * Włączanie zrzutów ekranu lub opis demonstrujący, jak aplikacja działa.
8. **Podsumowanie**
   * Krótka ocena projektu i osiągnięć.
   * Wnioski wynikające z ćwiczenia.
9. **Trudności i błędy**
   * Informacje na temat ewentualnych problemów, na jakie napotkano oraz jak zostały one rozwiązane.
10. **Źródła i odniesienia**
    * Jeśli korzystano z materiałów lub źródeł zewnętrznych, warto je tu uwzględnić.
11. **Dodatkowe materiały**
    * Ewentualne dodatkowe materiały, takie jak kody źródłowe, zrzuty ekranu itp.

Warto zachować przejrzystość i czytelność w sprawozdaniu, używać odpowiednich nagłówków, numeracji stron, a także opisywać kroki projektowania i implementacji w sposób logiczny. Sprawozdanie powinno być dostatecznie szczegółowe, aby inny programista mógł zrozumieć projekt i ewentualnie go udoskonalić lub wykorzystać w przyszłości.