



Technical Raport

AGH University of Science and Technology

LICZNIK STATYSTYKI TESKTU

Autor: Mikołaj Kołodziej

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie



Spis treści

1.WSTĘP.....	3
2.FUNKCJONALNOŚĆ (<i>FUNCTIONALITY</i>)	4
3.ANALIZA PROBLEMU (<i>PROBLEM ANALYSIS</i>)	5
4.PROJEKT TECHNICZNY (<i>TECHNICAL DESIGN</i>)	7
5.OPIS REALIZACJI (<i>IMPLEMENTATION REPORT</i>)	15
6.TESTY	17
7.PODRĘCZNIK UŻYTKOWNIKA (<i>USER'S MANUAL</i>).....	20
7.1 INSTRUKCJA OBSŁUGI	20
7.2 INSTALACJA (CMAKE)	24
8.METODOLOGIA ROZWOJU I UTRZYMANIA SYSTEMU (<i>SYSTEM MAINTENANCE AND DEPLOYMENT</i>)	25
9. BIBLIOGRAFIA.....	26

1. Wstęp

Celem projektu jest stworzenie aplikacji do analizy statystyk tekstowych w plikach o wybranych rozszerzeniach (np. .txt, .csv, .json, .cpp, .py). Aplikacja umożliwia zliczanie liczby znaków, wyrazów i linii w plikach znajdujących się w wybranych folderach i ich pod folderach, a wyniki prezentowane są w czytelnej formie.

2. Funkcjonalność (Functionality)

1. Wybór folderu i przeszukiwanie plików

- Użytkownik może wpisać nazwę folderu do przeszukania i rozszerzenie plików, które go interesują.
- Program rekurencyjnie wyszukuje pliki w folderze i pod folderach.
- Lista znalezionych plików jest wyświetlana w interfejsie.

2. Analiza pliku

- Po wybraniu pliku z listy użytkownik może przeprowadzić jego analizę.
- Plik jest wczytywany z zachowaniem odpowiedniego kodowania (np. obsługa polskich znaków i usuwanie BOM).
- Wyniki analizy (liczba znaków, słów i linii) są obliczane i wyświetlane w oknie dialogowym.

3. Obsługa rozszerzeń

- Aplikacja wspiera analizę plików o określonych rozszerzeniach (np. .txt, .csv, .json, .cpp, .py).
- Program sprawdza poprawność rozszerzenia wpisanego przez użytkownika oraz ostrzega, jeśli rozszerzenie jest nieobsługiwane.

4. Graficzny interfejs użytkownika

- Przyciski i pola tekstowe umożliwiają użytkownikowi wygodne wprowadzanie danych i nawigację.
- Komunikaty (błędy, ostrzeżenia, wyniki analizy) są prezentowane w oknach dialogowych.

3. **Analiza problemu (Problem Analysis)**

Definicja problemu

Problemem, który aplikacja rozwiązuje, jest potrzeba automatycznego przeszukiwania folderów oraz analizy zawartości plików tekstowych. W szczególności chodzi o zliczanie podstawowych statystyk tekstu (liczba znaków, słów, linii) dla wybranych plików o określonych rozszerzeniach.

Główne wyzwania to:

- Wydajne przeszukiwanie folderów i pod folderów.
- Obsługa plików o różnych rozszerzeniach.
- Analiza tekstu z uwzględnieniem różnych kodowań, w tym UTF-8.
- Prezentowanie wyników w przyjazny dla użytkownika sposób.

Wymagania funkcjonalne

- **Przeszukiwanie folderów:** Aplikacja musi umożliwiać użytkownikowi wskazanie folderu i rozszerzenia plików do przeszukania.
- **Obsługa rozszerzeń:** Program powinien obsługiwać z góry określoną listę rozszerzeń plików (np. .txt, .csv, .cpp).
- **Analiza plików:** Aplikacja powinna umożliwiać analizę wybranego pliku, podając statystyki dotyczące liczby znaków, słów i linii.
- **Interfejs graficzny:** Użytkownik powinien mieć możliwość korzystania z programu w sposób intuicyjny dzięki GUI.

Wymagania niefunkcjonalne

- **Wydajność:** Przeszukiwanie folderów i analiza plików muszą działać w rozsądnym czasie nawet dla dużych katalogów.
- **Przenośność:** Kod powinien działać na różnych platformach (Windows, Linux, macOS).
- **Obsługa błędów:** Program musi obsługiwać sytuacje wyjątkowe, takie jak brak dostępu do folderu, nieobsługiwane rozszerzenie lub brak wybranego pliku.

- **Łatwość obsługi:** Aplikacja powinna wyświetlać komunikaty informacyjne i błędów w sposób zrozumiały dla użytkownika.

Główne wyzwania

- **Rekurencyjne przeszukiwanie folderów:** Obsługa sytuacji, gdy foldery zawierają dużą liczbę pod folderów i plików.
- **Obsługa różnych rozszerzeń:** Każdy typ pliku może mieć inne specyficzne cechy (np. kodowanie tekstu).
- **Polskie znaki i kodowanie:** Należy zapewnić poprawne działanie z plikami zawierającymi znaki specyficzne dla języka polskiego.
- **Przyjazne GUI:** Intuicyjność interfejsu oraz informowanie użytkownika o błędach i wynikach w przystępny sposób.

4. Projekt Techniczny (Technical Design)

Opis klas:

DirectoryScanner - pozwala na rekurencyjne przeszukiwanie katalogów i podkatalogów w celu znalezienia plików o określonym rozszerzeniu.

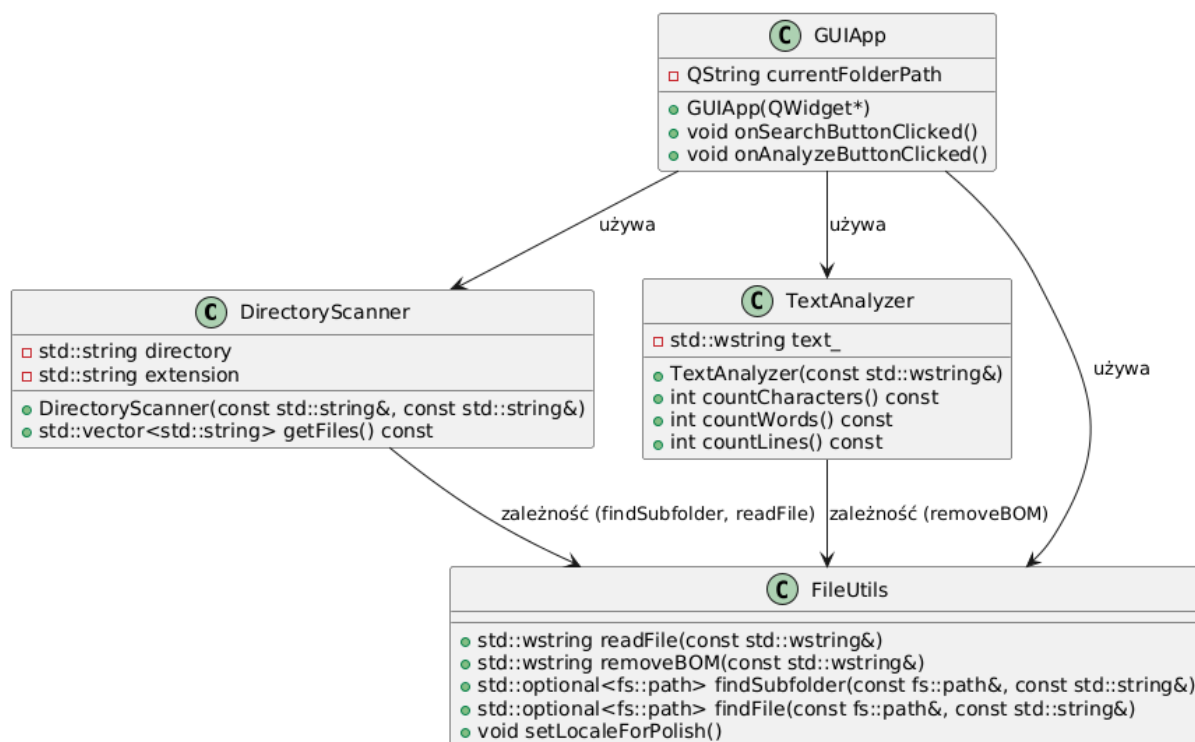
TextAnalyzer - służy do analizy tekstu przechowywanego w zmiennej wstring. Umożliwia liczenie znaków (pomijając spacje), słów i linii w tym tekście.

FileUtils - zawiera szereg funkcji pomocniczych do pracy z plikami i tekstami w różnych formatach i kodowaniach. Funkcje te obejmują m.in. odczyt plików z obsługą kodowania, lokalizacji, a także operacje na katalogach, takie jak znajdowanie podfolderów i plików.

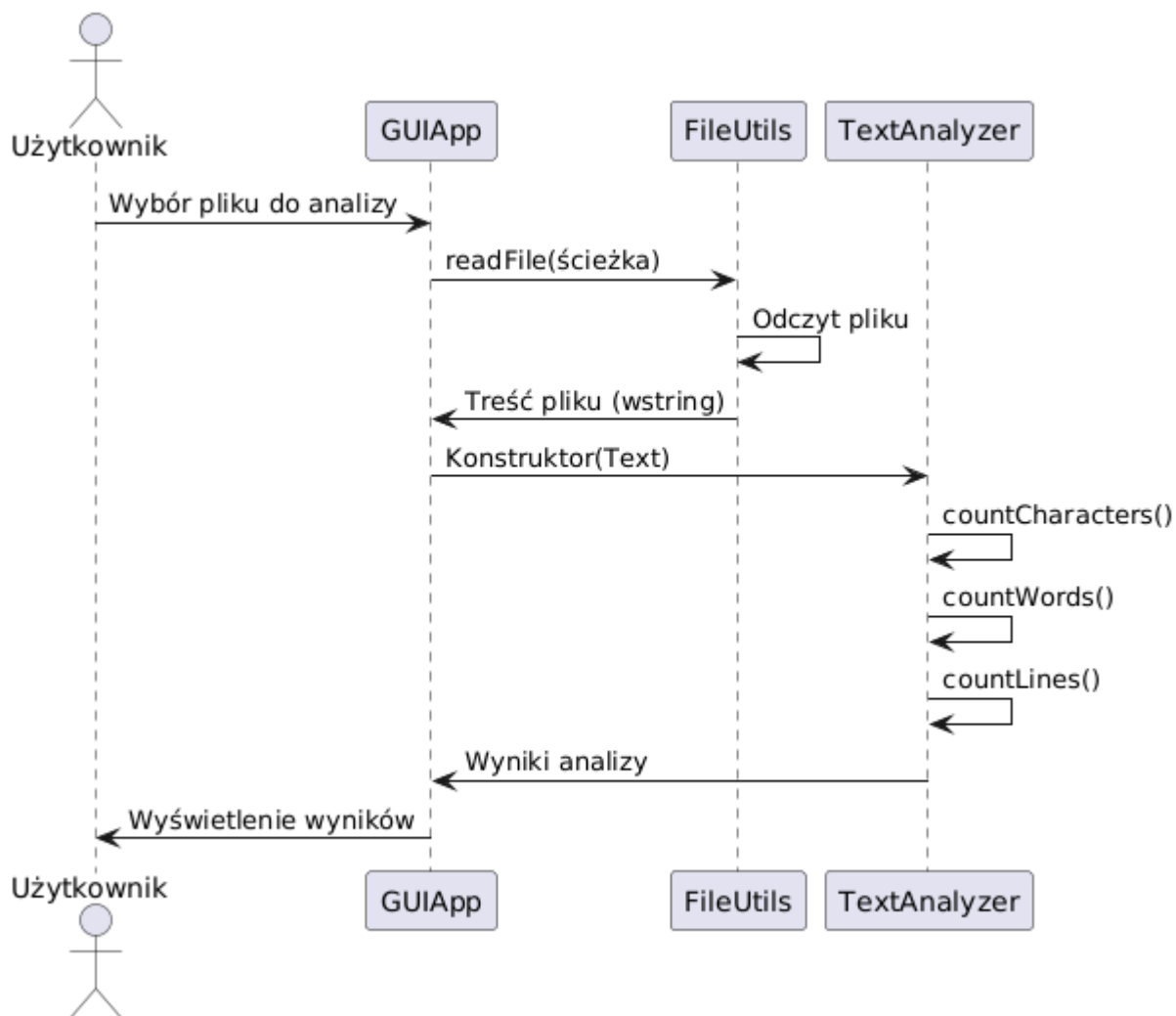
Poniżej zamieszczono diagramy UML:

- Diagram klas
- Diagram sekwencji
- Diagram użycia przez użytkownika
- Diagram stanów
- Diagram analizy plików
- Diagram aktywności
- Diagram komponentów

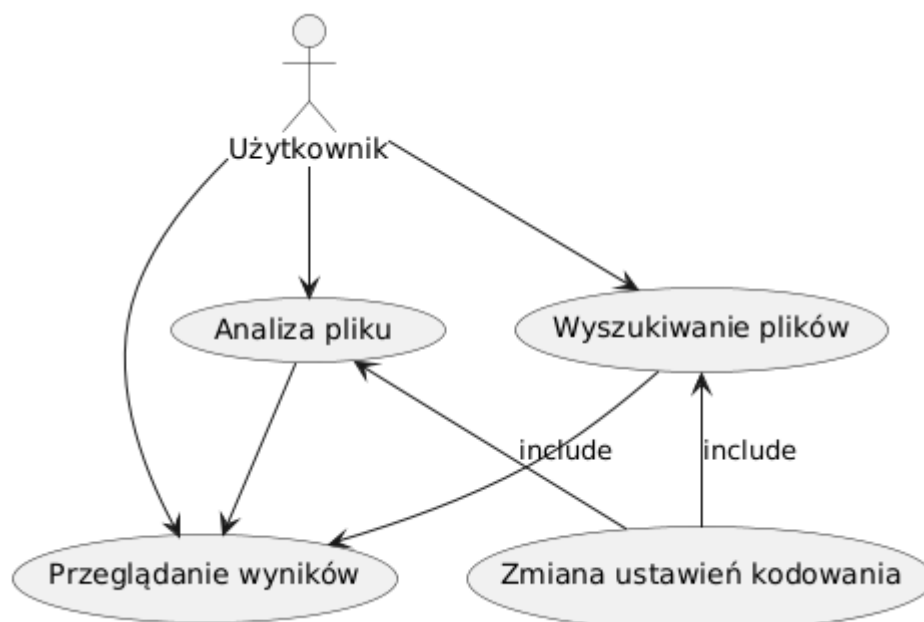
Diagramy wykonano za pomocą generatora [PlantUML](#)



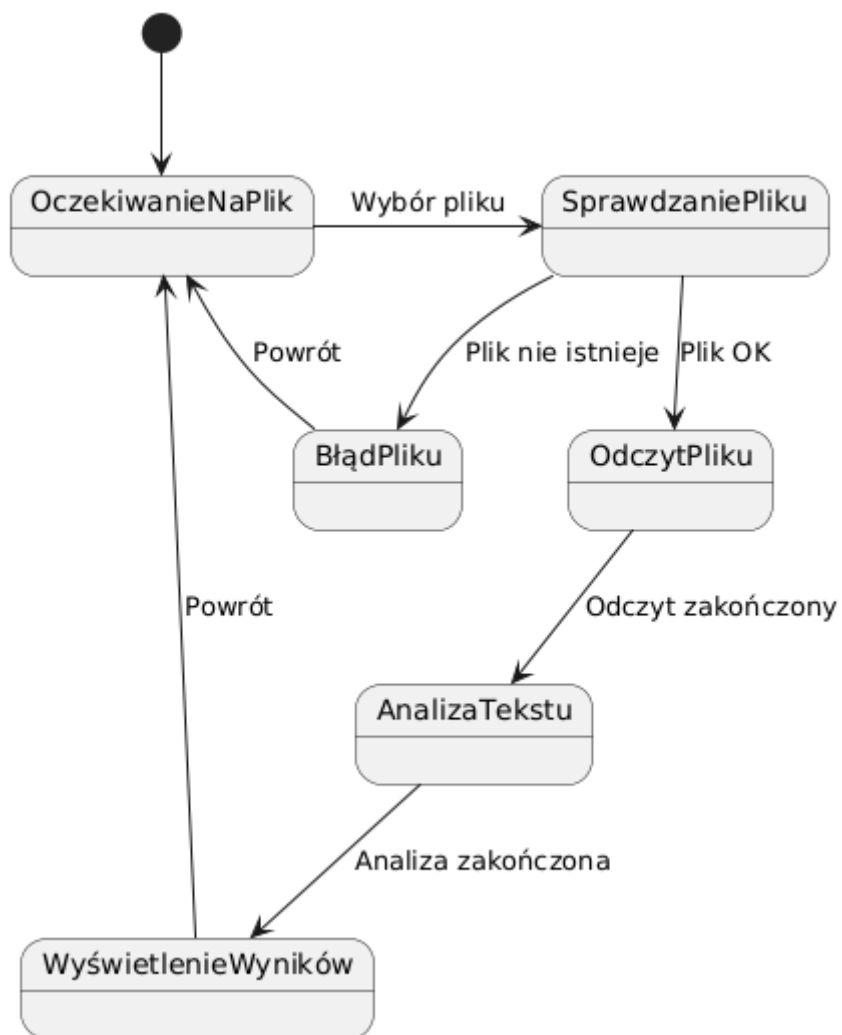
Rys. 1 Diagram klas



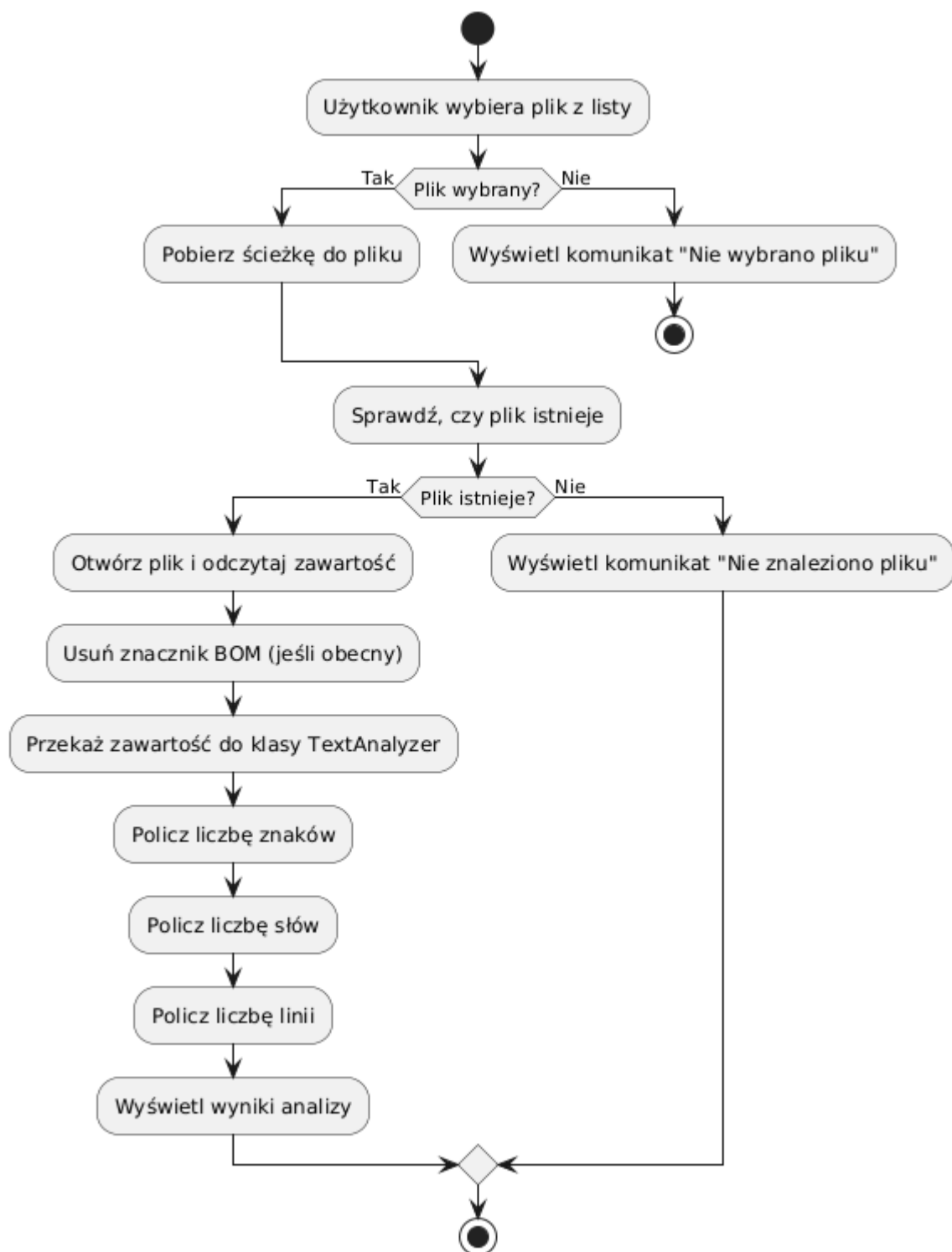
Rys. 2 Diagram sekwencji



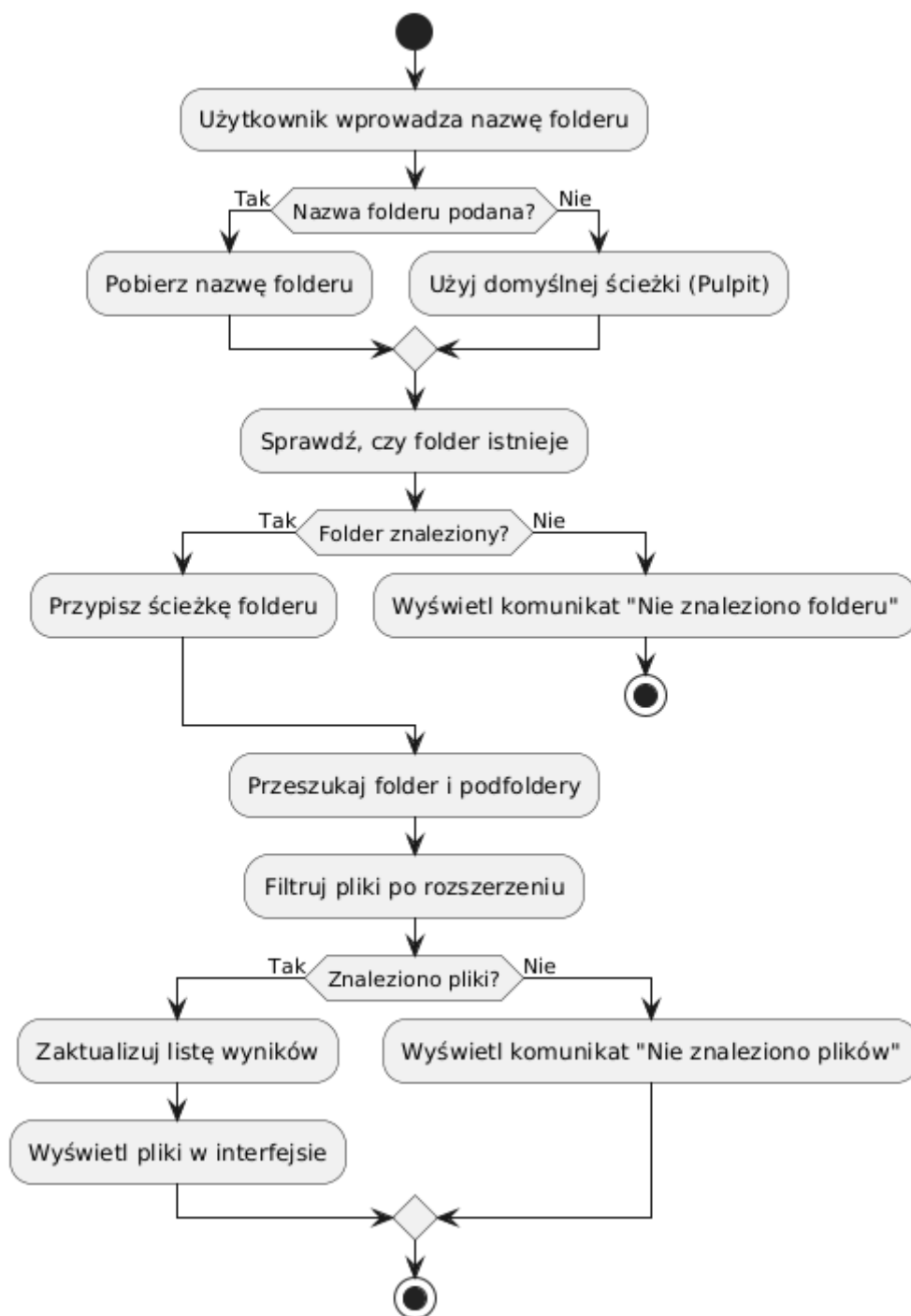
Rys. 3 Diagram użycia przez użytkownika



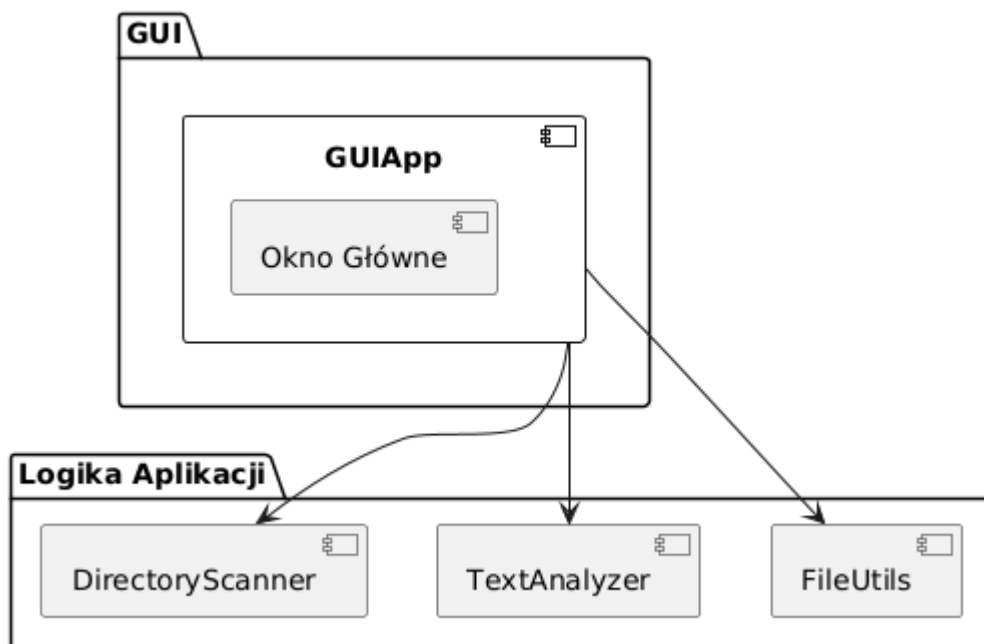
Rys. 4 Diagram stanów



Rys. 5 Diagram analizy plików



Rys. 6 Diagram aktywności



Rys. 6 Diagram komponentów

5. Opis realizacji (Implementation report)

Uwagi wstępne:

Do wykonania projektu, podczas procesu pisania programu:

- wykorzystano/zaadaptowano kod i wiedzę z książki “Programowanie w języku C++. Wprowadzenie dla inżynierów”, autor: prof. dr hab. inż. Bogusław Cyganek
- wykorzystywano/zaadaptowano kod wygenerowany przez narzędzie ChatGPT 4
- wykorzystano/zaadaptowano kod ze strony cppreference.com, zawierającą dokumentację i przykłady użycia dla języka C++

Etapy implementacji:

1. Stworzenie klasy DirectoryScanner do przeszukiwania folderów i podfolderów

- Zaimplementowanie funkcji do skanowania katalogów i podkatalogów w celu znalezienia odpowiednich plików.

2. Stworzenie klasy TextAnalyzer do odczytu i analizy plików

- Opracowanie metod do analizy tekstu w plikach, np. liczenia słów, znaków i linii.

3. Stworzenie aplikacji konsolowej w celu przetestowania poprawności działania programu

- Napisanie prostego interfejsu konsolowego do testowania działania klas DirectoryScanner i TextAnalyzer.

4. Stworzenie klasy FileUtils z brakującymi funkcjami do prawidłowego działania projektu

- Dodanie pomocniczych funkcji, takich jak odczyt plików w różnych kodowaniach, obsługa polskich znaków, oraz funkcje do pracy z plikami i katalogami.

5. Zaimplementowanie interfejsu graficznego w Qt

- Tworzenie aplikacji z interfejsem graficznym w Qt, który będzie wykorzystywał wcześniej stworzone klasy do analizy plików i wyświetlania wyników.

Technologie:

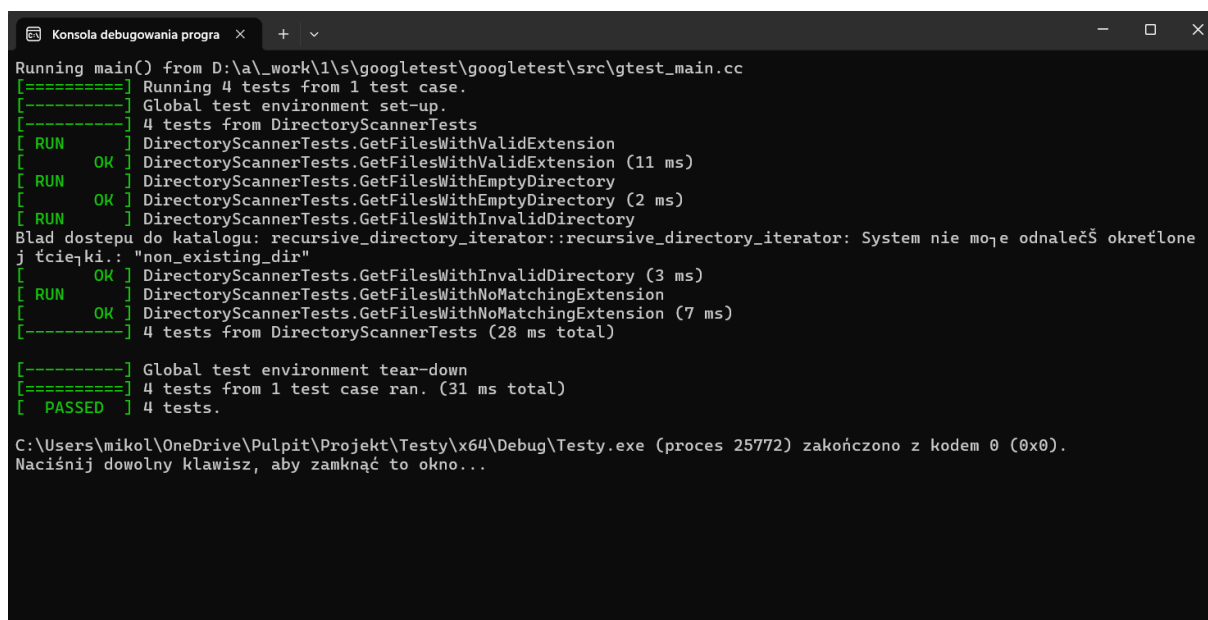
- Język: C++ 20
- Biblioteka GUI: Qt
- Środowisko IDE: Visual Studio 2022

6. Testy

Testy zostały przeprowadzone w Visual Studio 2022 (Google Test). Sprawdzone zostały wszystkie funkcje każdej z klas wykorzystanej w projekcie.

Test klasy DirectoryScanner

Testy obejmowały różne przypadki użycia klasy, takie jak działanie w katalogach z pasującymi i niepasującymi plikami, pustych katalogach oraz nieistniejących ścieżkach. Testy sprawdzały poprawność działania w typowych i nietypowych scenariuszach.



```

Konsola debugowania progra x + v
Running main() from D:\a\_work\1\s\googletest\googletest\src\gtest_main.cc
[=====] Running 4 tests from 1 test case.
[-----] Global test environment set-up.
[-----] 4 tests from DirectoryScannerTests
[ RUN ] DirectoryScannerTests.GetFilesWithValidExtension
[ OK ] DirectoryScannerTests.GetFilesWithValidExtension (11 ms)
[ RUN ] DirectoryScannerTests.GetFilesWithEmptyDirectory
[ OK ] DirectoryScannerTests.GetFilesWithEmptyDirectory (2 ms)
[ RUN ] DirectoryScannerTests.GetFilesWithInvalidDirectory
Błąd dostępu do katalogu: recursive_directory_iterator::recursive_directory_iterator: System nie może odnaleźć określone
j ścieżki.: "non_existing_dir"
[ OK ] DirectoryScannerTests.GetFilesWithInvalidDirectory (3 ms)
[ RUN ] DirectoryScannerTests.GetFilesWithNoMatchingExtension
[ OK ] DirectoryScannerTests.GetFilesWithNoMatchingExtension (7 ms)
[-----] 4 tests from DirectoryScannerTests (28 ms total)

[-----] Global test environment tear-down
[=====] 4 tests from 1 test case ran. (31 ms total)
[ PASSED ] 4 tests.

C:\Users\mikal\OneDrive\Pulpit\Projekt\Testy\x64\Debug\Testy.exe (proces 25772) zakończono z kodem 0 (0x0).
Naciśnij dowolny klawisz, aby zamknąć to okno...

```

Zrzut ekranu przedstawiający wyniki testów dla klasy DirectoryScanner

Test klasy TextAnalyzer

Wykonano testy sprawdzające poprawność metod klasy, obejmujące liczenie znaków, słów i wierszy w różnych scenariuszach, takich jak tekst pusty, z polskimi znakami, czy z nadmiarowymi spacjami. Dodatkowo zweryfikowano, czy tekst jest poprawnie przechowywany i zwracany.

```
[-----] 4 tests from TextAnalyzerTests
[ RUN      ] TextAnalyzerTests.CountCharacters
[      OK   ] TextAnalyzerTests.CountCharacters (1 ms)
[ RUN      ] TextAnalyzerTests.CountWords
[      OK   ] TextAnalyzerTests.CountWords (0 ms)
[ RUN      ] TextAnalyzerTests.CountLines
[      OK   ] TextAnalyzerTests.CountLines (0 ms)
[ RUN      ] TextAnalyzerTests.GetText
[      OK   ] TextAnalyzerTests.GetText (0 ms)
[-----] 4 tests from TextAnalyzerTests (6 ms total)
```

Zrzut ekranu przedstawiający wyniki testów klasy TextAnalyzer

Test klasy FileUtils

Przeprowadzono testy dla funkcji klasy, sprawdzając poprawność odczytu plików (istniejących i nieistniejących), usuwania BOM z tekstu, odnajdywania podfolderów i plików w katalogu, a także ustawiania polskiej lokalizacji. Testy obejmowały różne scenariusze, takie jak brak wymaganych zasobów czy nietypowe dane wejściowe.

```
[-----] 2 tests from ReadFileTests
[ RUN      ] ReadFileTests.ReadValidFile
[      OK   ] ReadFileTests.ReadValidFile (1 ms)
[ RUN      ] ReadFileTests.ReadInvalidFile
[      OK   ] ReadFileTests.ReadInvalidFile (1 ms)
[-----] 2 tests from ReadFileTests (3 ms total)

[-----] 2 tests from RemoveBOMTests
[ RUN      ] RemoveBOMTests.RemoveBOMFromText
[      OK   ] RemoveBOMTests.RemoveBOMFromText (0 ms)
[ RUN      ] RemoveBOMTests.NoBOMInText
[      OK   ] RemoveBOMTests.NoBOMInText (1 ms)
[-----] 2 tests from RemoveBOMTests (2 ms total)

[-----] 2 tests from FindSubfolderTests
[ RUN      ] FindSubfolderTests.FindSubfolderExists
[      OK   ] FindSubfolderTests.FindSubfolderExists (3 ms)
[ RUN      ] FindSubfolderTests.FindSubfolderDoesNotExist
[      OK   ] FindSubfolderTests.FindSubfolderDoesNotExist (1 ms)
[-----] 2 tests from FindSubfolderTests (7 ms total)

[-----] 2 tests from FindFileTests
[ RUN      ] FindFileTests.FindFileExists
[      OK   ] FindFileTests.FindFileExists (3 ms)
[ RUN      ] FindFileTests.FindFileDoesNotExist
[      OK   ] FindFileTests.FindFileDoesNotExist (0 ms)
[-----] 2 tests from FindFileTests (5 ms total)

[-----] 1 test from SetLocaleForPolishTests
[ RUN      ] SetLocaleForPolishTests.SetLocaleForPolish
[      OK   ] SetLocaleForPolishTests.SetLocaleForPolish (38 ms)
[-----] 1 test from SetLocaleForPolishTests (42 ms total)
```

Zrzut ekranu przedstawiający wyniki testu klasy FileUtils

7. Podręcznik użytkownika (User's manual)

7.1 INSTRUKCJA OBSŁUGI

1. Uruchomienie aplikacji

1. Uruchom plik wykonywalny aplikacji:

- Kliknij dwukrotnie na plik wykonywalny lub uruchom go z terminala, wpisując:

./GUIApp

2. Po uruchomieniu aplikacji zobaczysz główne okno, które zawiera:

- Pole tekstowe do wpisania nazwy folderu.
- Pole tekstowe do podania rozszerzenia plików.
- Listę wyników znalezionych plików.
- Przycisk **Szukaj**.
- Przycisk **Analizuj**.

2. Wyszukiwanie plików

1. Podanie nazwy folderu (*opcjonalne*):

- W polu "**Nazwa folderu**" wpisz nazwę pod folderu, w którym chcesz rozpocząć wyszukiwanie.
- Jeśli nie podasz nazwy, wyszukiwanie rozpocznie się od domyślnej lokalizacji – **pulpitu użytkownika**.

2. Podanie rozszerzenia plików (wymagane):

- W polu "**Rozszerzenie plików**" wpisz rozszerzenie plików, które chcesz znaleźć (np. .txt, .csv).
- Jeśli nie wpiszesz rozszerzenia, aplikacja wyświetli ostrzeżenie: *"Musisz podać rozszerzenie pliku."*
- Obsługiwane są tylko określone rozszerzenia. W przypadku podania nieobsługiwanego rozszerzenia pojawi się komunikat: *"Nieobsługiwane rozszerzenie pliku."*

3. Rozpoczęcie wyszukiwania:

- Kliknij przycisk **Szukaj**.
- Aplikacja przeszuka folder i jego pod foldery w poszukiwaniu plików o podanym rozszerzeniu.
- Wyniki zostaną wyświetlone w liście wyników z nazwami znalezionych plików.

4. Wyświetlenie komunikatu o braku wyników:

- Jeśli w katalogu nie znaleziono żadnych plików o podanym rozszerzeniu, aplikacja poinformuje o tym za pomocą komunikatu: *"Nie znaleziono plików o rozszerzeniu ."*

3. Analiza pliku

1. Wybór pliku do analizy:

- Wybierz plik z listy wyników, klikając na jego nazwę.
- Jeśli nie wybierzesz żadnego pliku i klikniesz **Analizuj**, aplikacja wyświetli ostrzeżenie: *"Nie wybrano pliku do analizy."*

2. Rozpoczęcie analizy:

- Kliknij przycisk **Analizuj**.
- Aplikacja przeprowadzi analizę tekstu w wybranym pliku, w tym:
 - Zliczy liczbę znaków (bez spacji).
 - Zliczy liczbę słów.
 - Zliczy liczbę linii tekstu.

3. Wyświetlenie wyników:

- Wyniki analizy zostaną przedstawione w oknie informacyjnym, zawierającym:
 - Ścieżkę do analizowanego pliku.
 - Liczbę znaków.
 - Liczbę słów.
 - Liczbę linii.

4. Obsługa błędów podczas analizy:

- Jeśli podczas analizy wystąpi błąd (np. plik został usunięty), aplikacja wyświetli komunikat o błędzie.

4. Dodatkowe informacje

- **Ścieżka folderu lub pliku:**
 - Aktualnie przeszukiwany folder lub analizowany plik jest wyświetlany w dolnej części interfejsu.
- **Obsługiwane rozszerzenia plików:**
 - *.txt, .csv, .json, .xml, .ini, .md, .yaml, .yml, .h, .cpp, .c, .py, .java, .html.*
- **Obsługa języka polskiego:**

- Aplikacja poprawnie obsługuje polskie znaki w nazwach plików i ich zawartości.
 - **Domyślna lokalizacja:**
 - Jeśli nie podasz nazwy folderu, aplikacja przeszukuje **pulpit użytkownika**.
-

7. Zakończenie pracy

Aby zamknąć aplikację, kliknij ikonę zamknięcia w prawym górnym rogu okna lub użyj skrótu klawiaturowego Alt + F4.

7.2 INSTALACJA (CMAKE)

Proces do przeprowadzenia instalacji CMAKE na innym komputerze (Windows).

1. Wszystkie niezbędne pliki znajdują się w folderze GUIApp.
2. Po pobraniu folderu należy stworzyć w nim nowy folder np. build
3. W CMakeLists.txt musimy podać nazwę folderu który stworzyliśmy w komendzie „include_directories(build) „ – inaczej podczas budowania kompilator nie będzie mógł znaleźć pliku **ui_GUIApp.h**
4. Wchodzimy do stworzonego folderu build(z poziomu terminala) i wykonujemy w nim polecenie ***cmake ..***
5. W następnym kroku możemy zbudować nasz projekt. Możemy zrobić to z poziomu terminala komendą ***cmake -build .*** lub wejść do pliku GUIApp.sln stworzonego w folderze build.
6. Gdy zdecydujemy się na wybranie opcji 1 z punktu 5 program możemy wywołać wchodząc do folderu Debug/Release (zależy od ustawienia kompilatora) i tam wywołujemy komendę ***./GUIApp***.

8. METODOLOGIA ROZWOJU I UTRZYMANIE SYSTEMU

Rozwój systemu:

- Nowe klasy, które byłyby odpowiedzialne za obsługę i dekodowanie zakodowanych plików takich jak: **pdf, csv, xml**.
- Moduł obsługi wielojęzyczności, aby umożliwić analizę tekstu w różnych językach, uwzględniając specyficzne znaki i zasady gramatyczne.
- Wizualizacja wyników analizy np. w formie wykresów.
- Rozbudowa filtracji plików np. data utworzenia, rozmiar pliku.

Utrzymanie systemu:

- Regularne testy jednostkowe(np. przy użyciu Google Test) oraz testy integracyjne.
- Zapewnienie wsparcia dla przyszłych wersji bibliotek Qt oraz kompilatorów.

9. Bibliografia

1. Cyganek B.: Programowanie w języku C++. Wprowadzenie dla inżynierów. PWN, 2023.
2. Strona internetowa cppreference.com