

# PROJEKT NR 021

## NAPISY NA ZDJĘCIACH

Yuliya Zviarko, Aleksander Krajewski, Radosław Porębiński  
(WFiIS AGH)

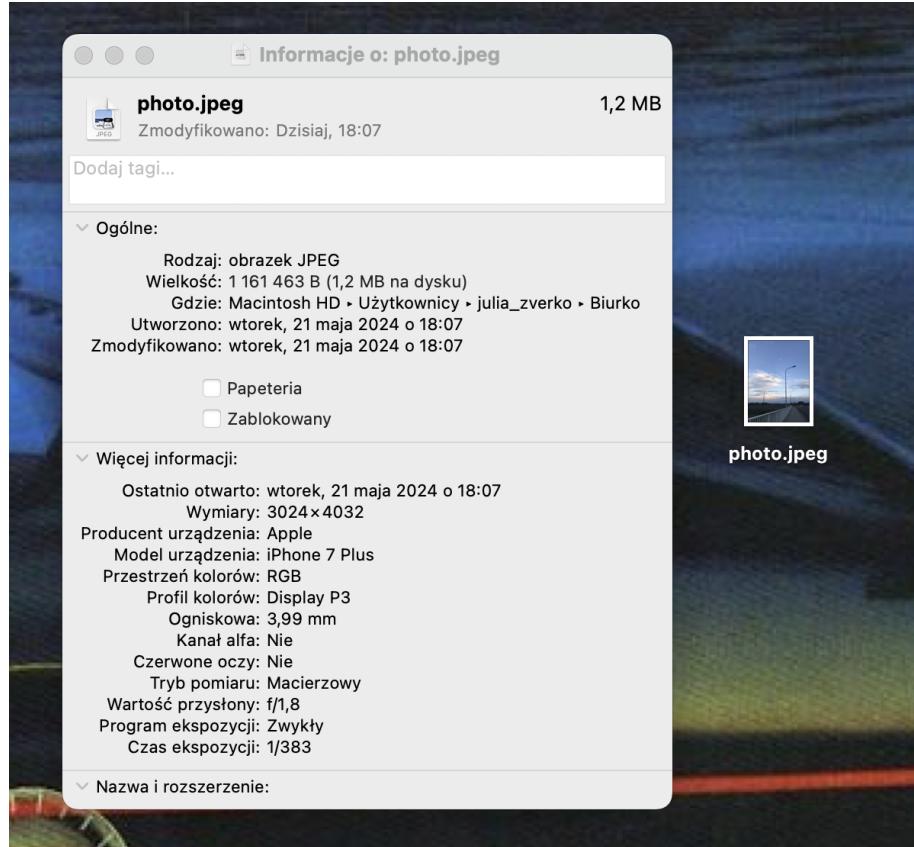
20.05.2024

### Spis treści

1 Opis Projektu	2
2 Założenia wstępne przyjęte w realizacji projektu	3
3 Analiza projektu	4
4 Podział pracy i analiza czasowa	7
5 Opracowanie i opis niezbędnych algorytmów	8
6 Struktury danych	10
7 Kodowanie	12
8 Testowanie	15
9 Wdrożenie, raport i wnioski	23

# 1 Opis Projektu

W dzisiejszych czasach, umieszczanie dodatkowych informacji w plikach z cyfrowymi zdjęciami stało się standardem. Dwa najpopularniejsze standardy, **EXIF**<sup>1</sup> i **IPTC**<sup>2</sup>, umożliwiają przechowywanie różnorodnych danych, jednakże te informacje nie są zawsze łatwo dostępne.



Rysunek 1: Wyświetlanie metadanych zdjęcia poprzez jego właściwości.

Wymagana jest specjalna przeglądarka, aby odczytać te dane. Naszym celem w ramach tego projektu było stworzenie właśnie takiej przeglądarki.

<sup>1</sup>EXIF (Exchangeable Image File Format) jest standardem służącym do określania formatu zapisu dodatkowych danych w plikach graficznych. EXIF przechowuje szereg informacji, w tym datę i godzinę wykonania zdjęcia, parametry użyte podczas jego tworzenia (takie jak czas naświetlania, przysłona, czułość ISO), a także dane dotyczące aparatu fotograficznego (takie jak model aparatu).

<sup>2</sup>IPTC (International Press Telecommunications Council) to standard często wykorzystywany w branży dziennikarskiej i wydawniczej. IPTC umożliwia dodawanie do zdjęć różnych informacji, takich jak nagłówki, opisy, słowa kluczowe, autorstwo oraz prawa autorskie. Dzięki IPTC proces katalogowania, wyszukiwania i zarządzania zdjęciami w systemach zarządzania treścią staje się łatwiejszy i bardziej efektywny.

W ramach projektu stworzyliśmy program do przeglądania zdjęć z wybranego katalogu. Ekran podzielony jest na dwie części: po lewej miniaturki zdjęć, po prawej informacje EXIF i IPTC wybranego zdjęcia. Po dwukrotnym kliknięciu zdjęcie powiększa się, zastępując miniaturki, z zachowaniem widoczności informacji. Kolejne kliknięcie przywraca miniaturki.

W rozszerzonej wersji programu dodaliśmy funkcje:

- Generowanie pliku z ścieżką i parametrami wszystkich zdjęć. Użytkownik wybiera, które parametry uwzględnić.
- Wpisywanie wybranych parametrów na zdjęciu i zapisywanie nowego pliku, pozostawiając oryginał nienaruszony.
- Automatyczne dodawanie parametrów na zdjęciach i zapisywanie ich pod nowymi nazwami dla wybranych zdjęć.
- Wczytywanie zewnętrznego pliku z nazwami plików i tekstami do umieszczenia na tych plikach (zdjęciach), co pozwala na hurtowe ich opisywanie.

## 2 Założenia wstępne przyjęte w realizacji projektu

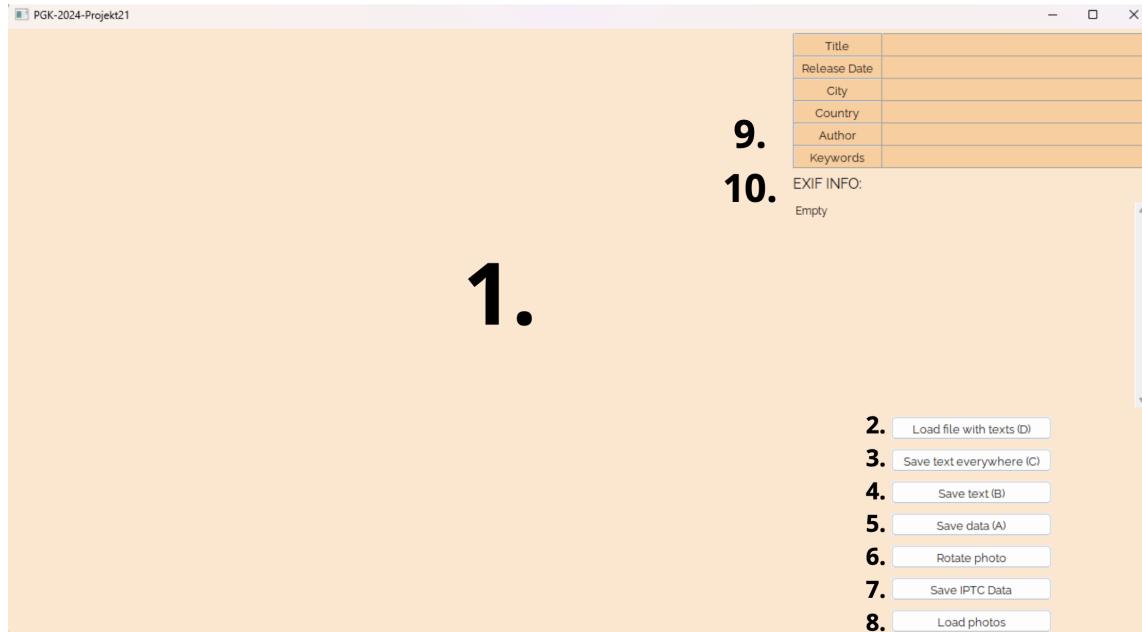
Nasz zespół podczas wykonywania projektu przyjął następujące założenia:

- **Wczytywanie folderu z plikami JPG:** Program umożliwia wczytanie wybranego przez użytkownika folderu, z którego będą przeglądane zdjęcia. Program wyświetla jedynie pliki z rozszerzeniem .jpg.
- **Minimalna wielkość okna:** Wielkość okna programu jest minimalnie ustawiona na 1300x720 pikseli, zapewniając wygodne korzystanie z aplikacji oraz odpowiednie wyświetlanie zawartości.
- **Wyświetlanie miniaturek:** Na podstawie przeskalowanych bitmap tworzone są "przyciski" w postaci BitmapButtonów, które są dodawane do FlexGridSizera. Umożliwia to wygodne przeglądanie miniatur zdjęć.
- **Wyświetlanie danych EXIF i IPTC:** Program umożliwia wyświetlanie informacji EXIF i IPTC dla aktualnie wybranego zdjęcia. Informacje te są wyświetlane po prawej stronie okna programu.
- **Obsługa podwójnego kliknięcia:** Podwójne kliknięcie na miniaturkę powoduje pokazanie dużego zdjęcia oraz ukrycie miniatur. Podwójne kliknięcie na dużym zdjęciu przywraca widok miniaturek.

Te założenia stanowiły podstawę dla dalszego rozwoju projektu i wpłynęły na sposób jego realizacji oraz interfejs użytkownika.

### 3 Analiza projektu

Po uruchomieniu programu pojawia się interfejs użytkownika:



Rysunek 2: Interfejs użytkownika.

który składa się z następujących bloków:

1. **Pole do wyświetlania zdjęć:** Wyświetlane są wszystkie zdjęcia z wczytywanego folderu.
2. **Load file with texts (D):** Pozwala na zapis danych z wczytywanego pliku .txt na obrazkach, których nazwy zostały podane w tym pliku.
3. **Save text everywhere (C):** Umożliwia zapis danych EXIF na wszystkich wybranych zdjęciach w folderze.
4. **Save text (B):** Służy do zapisu danych EXIF na wybranym zdjęciu.
5. **Save data (A):** Zapisuje w pliku .txt ścieżki do otwartych w programie zdjęć oraz informacje EXIF na temat każdego ze zdjęć.
6. **Rotate photo:** Służy do obracania zdjęcia oraz wyświetlania danych EXIF na temat każdego ze zdjęć.
7. **Save IPTC Data:** Pozwala na zapis edytowanych danych IPTC.
8. **Load photos:** Służy do wyboru folderu, z którego są wczytywane zdjęcia do programu.

9. **Tabela z danymi IPTC:** Zawiera następujące dane: tytuł (title), data wydania (release date), miasto (city), kraj (country), autor (author), słowa kluczowe (keywords).
10. **Pole do wyświetlania danych EXIF:** Przeznaczone do wyświetlania szczegółowych danych EXIF zdjęć.

Oprócz wymienionych bloków, w aplikacji niezbędne były także komponenty **Grid** oraz **StaticText**.

StaticText służy do wyświetlania danych EXIF, których ilość odpowiada liczbie zapisanych danych w pliku. Grid umożliwia wypisywanie oraz edytowanie danych IPTC. Po wprowadzeniu nowych danych, aby je zapisać, należy kliknąć przycisk "Save IPTC Data".

Aby program działał zgodnie z oczekiwaniami, trzeba wybrać folder zawierający pliki z rozszerzeniem .jpg. Pliki z innymi rozszerzeniami zostaną zignorowane i ich miniaturki nie będą wyświetlane. Podczas działania programu użytkownik ma możliwość wygenerowania pliku lub plików (w zależności od klikniętego przycisku) o rozszerzeniu .jpg. Obrazki będą zawierać oryginalne zdjęcie z wypisanymi danymi EXIF w lewym górnym rogu. Dodatkowo, użytkownik może swobodnie modyfikować dane IPTC obrazka. Możliwe jest także wygenerowanie pliku .txt, który zawiera ścieżki do wszystkich wyświetlonych miniaturek wraz z ich rozmiarami.

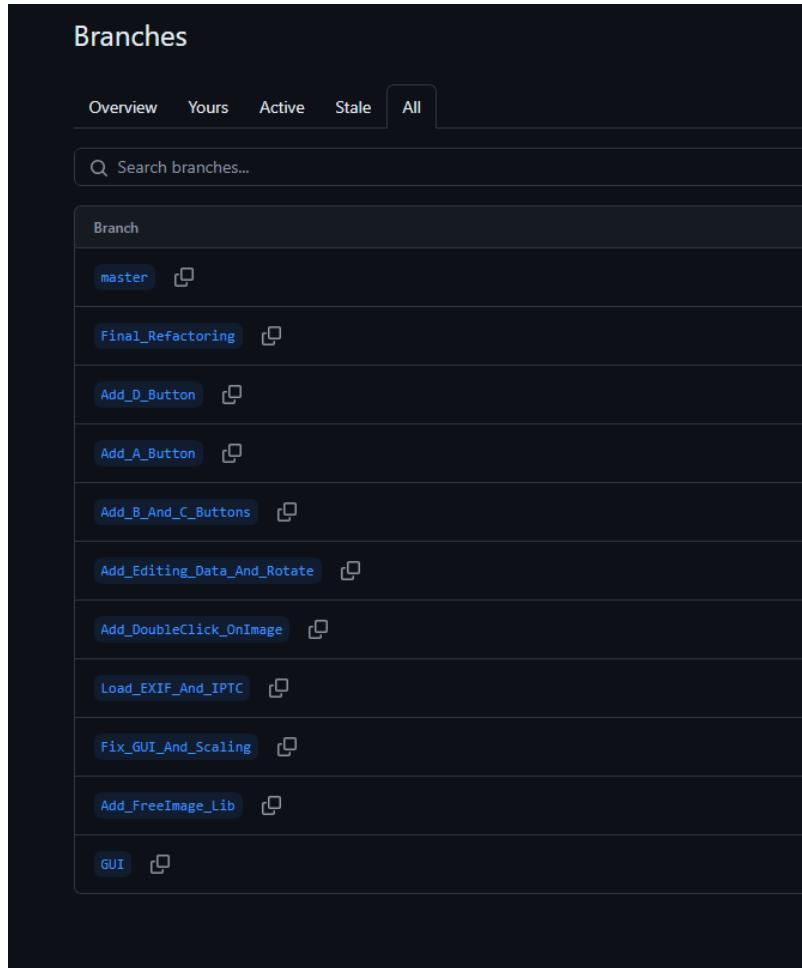
Program podzieliliśmy na następujące problemy:

- Wyświetlanie zdjęć z katalogu.
- Wyświetlanie danych IPTC po jednokrotnym kliknięciu na zdjęcie.
- Wyświetlanie dużego zdjęcia po dwukrotnym kliknięciu.
- Wyświetlanie danych EXIF.
- Zapisywanie danych na zdjęcia lub zdjęcie.
- Edytowanie danych IPTC.
- Zapisywanie informacji o zdjęciach do pliku.
- Dodanie napisów na zdjęcia w zależności od wybranego przycisku.

Podczas tworzenia aplikacji korzystaliśmy z wxFormBuilder, co znaczco ułatwiło projektowanie interfejsu użytkownika. Kod programu został napisany w języku C++, przy użyciu bibliotek wxWidgets oraz FreeImage. Wykorzystanie biblioteki FreeImage okazało się niezbędne, umożliwiając nam łatwy dostęp do danych IPTC oraz EXIF.

Wszyscy członkowie zespołu pracowali w środowisku Visual Studio 2019. Aby zoptymalizować proces pracy, podzieliliśmy zadania pomiędzy siebie, każdy zajmując się innymi aspektami programu. Jednak warto zauważyć, że nie ograniczaliśmy się jedynie do swoich obszarów odpowiedzialności. W trakcie pracy nad projektem, często udzielaliśmy sobie nawzajem pomocy, szczególnie gdy ktoś dodawał nowe funkcjonalności lub napotykał problemy. Dzięki takiemu podejściu nasza praca była w pewnym sensie wspólnym wysiłkiem, gdzie każdy miał udział w rozwoju projektu.

Do koordynacji naszej pracy wykorzystaliśmy platformę GitHub. Stosowaliśmy praktykę pracy na różnych gałęziach (branches) w repozytorium. Każda gałąź była dedykowana konkretnemu zadaniu lub funkcjonalności.



Rysunek 3: Proces pracy na różnych gałęziach.

W przypadku gdy każda gałąź działała poprawnie i testy były zaliczone, łączymy ją z główną gałęzią master. Dzięki temu nowe funkcje i poprawki były stopniowo wprowadzane do głównej wersji projektu. Proces ten umożliwiał nam prowadzenie pracy równolegowej nad różnymi elementami aplikacji, zachowując przy tym stabilność głównej gałęzi i minimalizując ryzyko wprowadzenia błędów do głównej wersji kodu.

## 4 Podział pracy i analiza czasowa

Zadanie	Yuliya Z.	Radosław P.	Aleksander K.
Repozytorium	15 min	30 min	10 min
Planowanie	50 min	50 min	50 min
wxFormBuilder	X	2 h	X
Wymagania podstawowe	6 h	2 h	1 h
Pierwsze testy i poprawki	30 min	6 h	25 min
Wymagania rozszerzone	4 h	1 h	5h
Ostateczne testy i poprawki	30 min	4 h	2 h
Dokumentacja	6 h	2 h	2 h
Kosmetyka kodu i przygotowanie do wysłania	20 min	2 h	X

Ważne jest zauważenie, że powyższe wartości są przybliżone, a większość zadań wykonywaliśmy wspólnie.

Chociaż projekt był realizowany razem, każdy z nas miał swoje indywidualne obszary odpowiedzialności:

- Yuliya Z.: Zajmowała się wczytywaniem zewnętrznych plików zawierających nazwy plików oraz proste teksty, które miały zostać umieszczone na zdjęciach. Ponadto pracowała nad możliwością modyfikacji danych IPTC i ich przypisywaniem do zdjęć. Przygotowała także pliki zawierające niezbędne dane IPTC dla każdego zdjęcia. Dodatkowo, dodała funkcjonalność wyboru zdjęć dla przycisku C oraz zapewniła, że w panelu aplikacji wyświetlane są tylko oryginalne zdjęcia. Do przycisku B dodała opcję tworzenia nowego pliku z metadanymi na kopii zdjęcia. Brała udział w testowaniu, debugowaniu i wprowadzaniu niezbędnych poprawek w celu osiągnięcia wspólnego celu. Odpowiada za punkty 2, 3, 4 i 9 w sprawozdaniu
- Radosław P.: Zajmował się konfiguracją środowiska jak i projektu, odpowiednie załadowanie biblioteki FreeImage. Przygotował również oprawę graficzną za pomocą programu wxFormBuilder, pracował nad mechanizmem skalowania okna i obliczania prawidłowych wielkości zdjęć. Większość czasu był odpowiedzialny za bieżące testowanie i naprawianie powstałych błędów oraz refaktoryzację i optymalizację całego kodu aby był zgodny z przyjętymi schematami. Odpowiada za punkty 6 i 9 w sprawozdaniu.
- Aleksander K.: Zajmował się stworzeniem wymagań rozszerzonych: (A) zapisywanie wybranych atrybutów zdjęć do pliku tekstowego, (B) rotację przybliżonych obrazów, (C) nanoszenie tekstu na obrazy używając biblioteki FreeImage oraz (D) nanoszenie tekstu na wiele wybranych obrazów. Stworzył klasę IPTCCheckboxDialog odpowiedzialną za okno dialogowe do wyboru atrybutów do zapisania. Odpowiada za punkty 5 i 7 w sprawozdaniu.

## 5 Opracowanie i opis niezbędnych algorytmów

Poniżej znajduje się opis każdego ważnego algorytmu wraz z pseudokodem.

- Algorytm wczytywania zdjęć z katalogu:

```
ustaw defaultPath na "/"
utwórz wxDirDialog z tytułem "Choose directory" i defaultPath
jeśli dialog.ShowModal() == wxID_OK
    wyczysć pathArray
    jeśli buttonsSizer istnieje
        wyczysć buttonsSizer (true)
    ustaw buttonsSizer na nullptr
    wyczysć bitmapVector

    pobierz ścieżkę z dialog.GetPath()
    utwórz wxDir z dialog.GetPath()

    utwórz wxArrayString files
    dir.GetAllFiles(dialog.GetPath(), &files, "*.jpg", wxDIR_FILES)

    dla każdego pliku w files
        jeśli plik nie kończy się na "_modified.jpg"
            dodaj plik do pathArray

    ustaw fileCount na pathArray.GetCount()
    wywołaj funkcję Repaint()
w przeciwnym razie
    wyświetl komunikat wxMessageBox("Unable to open the folder.")
```

- Algorytm odświeżania:

```
funkcja Repaint(bool loading=true)
utwórz wxClientDC dla m_scrolledWindowPhotos

jeśli liczba plików (fileCount) jest większa od 0
    jeśli ładowanie jest prawdą
        ustaw tekst "LOADING IMAGES..."
        ustaw czcionkę dla dc na 20-punktową czcionkę Raleway
        oblicz szerokość i wysokość tekstu
        narysuj tekst na środku dc (zarówno w poziomie, jak i w pionie)

    dla każdego pliku w liczbie plików
        wczytaj obraz z pathArray[i]
        przeskaluj obraz do 240x180 pikseli używając jakości NEAREST
```

```

utwórz bitmapę z obrazu
dodaj bitmapę do bitmapVector

ustaw tło dc na kolor (255, 229, 204)
wyczyść dc
wywołaj funkcję printBitmapButtons

```

- Algorytm rysowania miniaturek:

```

jeśli liczba plików jest większa od 0
    rows = 1
    cols = (szerokość okna - 40) / 260
    jeśli liczba kolumn nie jest równa 0
        rows = liczba plików / liczba kolumn + 1

jeśli buttonsSizer istnieje
    wyczyść buttonsSizer

utwórz nowy wxFlexGridSizer z liczbą wierszy, liczbą kolumn
ustaw kierunek zmiany wielkości wxFlexGridSizer na oba kierunki (poziomy i pionowy)

dla każdego pliku w liczbie plików
    utwórz nowy wxBitmapButton z obrazem z bitmapVector[i] i ścieżką z pathArray[i]
    dodaj przycisk do buttonsSizer

ustaw menedżer rozkładu m_scrolledWindowPhotos na buttonsSizer
dopasuj m_scrolledWindowPhotos do rozmiaru zawartości buttonsSizer
ustaw prędkość przewijania m_scrolledWindowPhotos na 25

```

- Algorytm przy zmianie rozmiaru okna:

```

ustaw szerokość okna na obecną - 400
jeżeli wyświetla się przybliżone zdjęcia
    Wyświetl folder miniatur

```

- Algorytm wyświetlania przybliżonego zdjęcia:

```

w = pobierzSzerokośćOkna()
h = pobierzWysokośćOkna()
img_w = pobierzSzerokośćObrazu()
img_h = pobierzWysokośćObrazu()
scale_h = h / img_h      //skala wysokości

```

```

scale_x = w / img_w      //skala szerokości

jeśli (img_w > w) LUB (img_h > h)
    jeśli (scale_h < scale_x)
        przeskaluj obraz do szerokości = scale_h * img_w, wysokość = scale_h * img_h
    w przeciwnym razie
        przeskaluj obraz do szerokości = img_w * scale_x, wysokość = img_h * scale_x

utwórz bitmapę z obrazu
utwórz kontekst urządzenia dla tego okna
ustaw tło kontekstu urządzenia na kolor (255, 229, 204)
wyczyść kontekst urządzenia
narysuj bitmapę na kontekście urządzenia, wyśrodkowaną w oknie

```

## 6 Struktury danych

```

class MyFrame1 : public wxFrame {...}

class GUIMyFrame1 : public MyFrame1 {...}

```

GUIMyFrame1 to klasa odpowiedzialna za główną ramkę programu, która dziedziczy po MyFrame1, która przechowuje wszystkie elementy GUI, takie jak przyciski, pola tekstowe oraz panele. W klasie GUIMyFrame1 znajduje się cała logika programu, w tym obsługa kliknięć przycisków, wyświetlanie odpowiednich elementów oraz skalowanie okna.

Posiada ona takie pola jak:

- int windowHeight //szerokość panelu w którym wyświetlane są zdjęcia
- size\_t fileCount //ilość załadowanych zdjęć
- wxArrayString pathArray //Podczas załadowania folderu ze zdjęciami wszystkie ścieżki do tych zdjęć zostają przechowane tutaj
- std::vector<wxBitmap> bitmapVector //Podczas załadowania folderu ze zdjęciami wszystkie bitmapy stworzone na podstawie przeskalowanych zdjęć przechowujemy tutaj
- wxFlexGridSizer\* buttonsSizer //Sizer do którego dodawane są powstałe przyciski w celu zapewnienia odpowiedniego skalowania się okna programu
- BigDisplay\* currentBigDisplay //klasa z aktualnie wyświetlonym zdjęciem w powiększeniu
- wxString currentImgPath //ścieżka do aktualnie zaznaczonego zdjęcia

Oraz takie metody jak:

- void printBitmapButtons //Odpowiedzialna za odpowiednie obliczanie rozmiarów zdjęć oraz ilości wierszy i kolumn ze zdjęciami, oraz utworzenie naszych przycisków ze zdjęciami

- void Repaint //Odpowiedzialna za rysowanie całego panelu ze zdjęciami
- static wxString getImageExifData //Zwraca dane EXIF zdjęcia na podstawie danej bitmapy
- static wxString getImageIPTCData //Zwraca dane IPTC zdjęcia na podstawie danej bitmapy
- static wxString getImageIPTCData //Zwraca dana wartość IPTC na podstawie podanego tagu i danej bitmapy
- static void AddText //Odpowiedzialna za obliczanie odpowiedniego rozmiaru czcionki i zapisanie zmodyfikowanego zdjęcia
- static void DrawTextOnWxImage //Odpowiedzialna za naniesie odpowiedniego tekstu na zdjęcie
- void DisplayBigIm //Tworzy oraz wyświetla obiekt odpowiedzialny za powiększenie zdjęcia
- void DisplayMetaData //Wyświetla dane IPTC oraz EXIF w odpowiednich miejscach
- void DisplayFolder //Wraca do widoku miniatur

```
class BigDisplay : public wxPanel {...}
```

Klasa BigDisplay odpowiada za utworzenie panelu z powiększonym zdjęciem. Przechowuje ona wybrane zdjęcie i ścieżkę do niego. Wśród najważniejszych elementów tej klasy można uznać metody:

- void PaintBigDisplay //Odpowiednio skaluje i wyświetla powiększone zdjęcie
- void RotateImage //Obraca zdjęcie z każdym wywołaniem o kolejne 90 stopni

```
class ImageButton : public wxBitmapButton {...}
```

Klasa ImageButton jest przyciskiem z bitmapą danego zdjęcia. Przechowuje informacje o ścieżce zdjęcia. Pomaga wywoływać (podczas klikania) metody odpowiedzialne za wyświetlanie metadanych zdjęcia, oraz powiększenie zdjęcia.

```
class ImageSelectionDialog : public wxDialog {...}
```

ImageSelectionDialog Jest to klasa, która oferuje okienko do wyboru zdjęć na których chcemy umieścić napisy. Wykorzystywane przez odpowiednie przyciski.

```
class IPTCCheckboxDialog : public wxDialog {...}
```

IPTCCheckboxDialog Jest to klasa, która oferuje okienko do wyboru parametrów które chcemy umieścić na zdjęciach. Wykorzystywane przez odpowiednie przyciski.

## 7 Kodowanie

Poniżej opisano zaimplementowanie najważniejszych metod wykorzystywanych przez stworzone klasy:

- GUIMyFrame1

```
void Repaint(bool loading)
```

Metoda odpowiadająca za ponowne narysowanie interfejsu graficznego. Wyświetla komunikat "LOADING IMAGES..." oraz przetwarza obrazy, gdy loading jest true. Skaluje obrazy do rozmiaru 240x180 pikseli i zapisuje je w wektorze bitmapVector. Na końcu ustawia tło, czyści okno i wywołuje metodę printBitmapButtons().

```
static wxString getImageExifData(FIBITMAP* bitmap)
```

Metoda pobierająca dane EXIF z bitmapy, używa biblioteki FreeImage. Iteruje przez metadane EXIF, zbiera kluczowe wartości i dodaje je do imgData. Na końcu zamknięta uchwytu do metadanych i zwraca zebrane dane EXIF w formie wxString.

```
static wxString getImageIPTCData(FIBITMAP* bitmap)
```

Metoda pobierająca dane IPTC z bitmapy. Iteruje przez zdefiniowane pola IPTC, pobiera odpowiadające im wartości z metadanych i dodaje je do IPTCData. Zwraca zebrane dane IPTC w formie wxString.

```
static wxString getImageIPTCData(FIBITMAP* bitmap, const char* tagName)
```

Metoda pobierająca określone dane IPTC z bitmapy na podstawie nazwy tagu. Pobiera wartość z metadanych IPTC i zwraca ją jako wxString. Jeśli tag nie istnieje, zwraca pusty string.

```
static void AddText(wxImage image, const wxString& text, const wxString& path)
```

Metoda dodająca tekst do obrazu i zapisującą zmodyfikowany obraz. Ustawia rozmiar czcionki na podstawie wymiarów obrazu, rysuje tekst na obrazie, a następnie zapisuje go w formacie JPEG pod nową nazwą. Usuwa metadane, co może wymagać naprawy.

```
static void DrawTextOnWxImage(wxImage& image, const wxString& text,
                               int x, int y, const wxFont& font, const wxColour& textColor)
```

Metoda rysująca tekst na obrazie, wykorzystywana przez metodę AddText. Tworzy tymczasową bitmapę na podstawie obrazu, następnie tworzy pamięciowy kontekst urządzenia (wxMemoryDC), ustawia czcionkę i kolor tekstu, rysuje tekst na bitmapie, a na końcu przypisuje zmodyfikowany obraz z powrotem do obiektu image.

```
void DisplayBigIm()
```

Metoda wyświetlająca duży obraz. Ukrywa kontrolkę przewijania, ustawia rozmiar panelu na pełny rozmiar okna, a następnie tworzy obiekt BigDisplay i przekazuje mu ścieżkę do obrazu oraz panel, na którym ma być wyświetlony.

```
void DisplayMetaData()
```

Metoda wyświetlająca metadane obrazu. Ładuje obraz z podanej ścieżki za pomocą biblioteki FreeImage, czyści kontrolkę tekstową, ustawia jej czcionkę, a następnie wyświetla dane EXIF i IPTC na kontrolce tekstowej oraz siatce. Na końcu zwalnia załadowany obraz za pomocą FreeImage\_Unload.

```
void DisplayFolder()
```

Metoda wyświetlająca zawartość folderu. Jeśli istnieje aktualne wyświetlanie dużego obrazu, ukrywa je i usuwa. Następnie ukrywa panel pełnego wyświetlania i pokazuje panel przewijania.

- BigDisplay

```
void PaintBigDisplay()
```

Metoda rysująca duży obraz na panelu. Oblicza skale dla szerokości i wysokości obrazu oraz skaluje obraz, jeśli jest większy niż dostępna przestrzeń. Następnie rysuje obraz na panelu za pomocą wxClientDC.

```
void RotateImage()
```

Metoda obracająca obraz o 90 stopni w prawo i ponownie rysująca duży obraz na panelu za pomocą metody PaintBigDisplay()

- ImageButton

```
void OnMouseLeftDown(wxMouseEvent& event)
```

Metoda obsługująca zdarzenie naciśnięcia lewego przycisku myszy. Wywołuje metodę DisplayMetaData() klasy frame (prawdopodobnie GUIMyFrame1) w celu wyświetlenia metadanych dla obrazu o ścieżce path.

```
void OnMouseDoubleClick(wxMouseEvent& event)
```

Metoda obsługująca podwójne kliknięcie lewym przyciskiem myszy. Wywołuje metodę DisplayBigImg() klasy frame w celu wyświetlenia dużej wersji obrazu o ścieżce path.

- ImageSelectionDialog

```
ImageSelectionDialog(wxWindow* parent, const wxArrayString& images)
```

Konstruktor klasy ImageSelectionDialog, który tworzy okno dialogowe do wyboru obrazów. Przyjmuje okno nadziedzne parent oraz tablicę nazw obrazów images. Tworzy interfejs użytkownika składający się z listy obrazów oraz przycisków OK i Anuluj.

```
Metoda GetSelectedImages()
```

Metoda zwracająca indeksy wybranych obrazów z listy. Pobiera indeksy zaznaczonych pozycji z listy m\_listBox i zwraca je w postaci tablicy wxArrayInt.

- IPTCCheckboxDialog

```
IPTCCheckboxDialog(wxWindow* parent)
```

Konstruktor klasy IPTCCheckboxDialog, który tworzy okno dialogowe do wyboru pól IPTC. Przyjmuje okno nadrzędne parent i tworzy interfejs użytkownika z polami wyboru dla różnych pól IPTC oraz przyciskami OK i Anuluj.

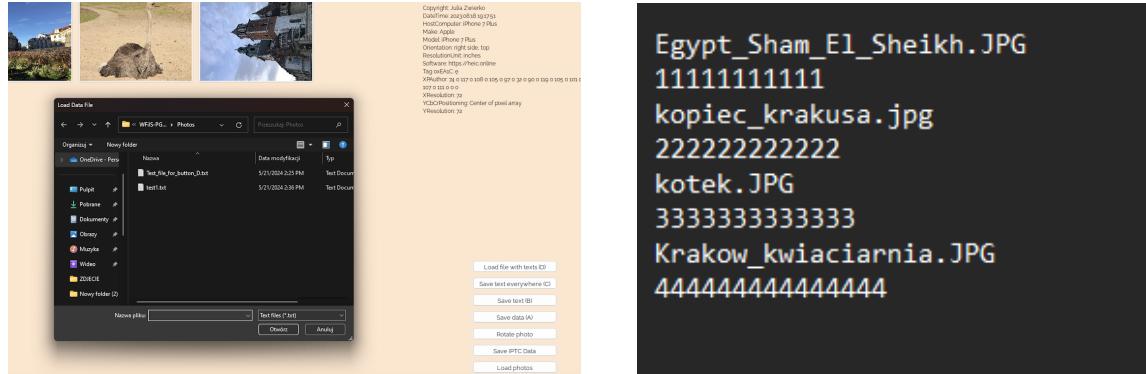
**Metoda GetCheckedItems()**

Metoda zwracająca wektor zawierający identyfikatory pól IPTC wybranych przez użytkownika. Sprawdza, które pola IPTC są zaznaczone, i dodaje ich identyfikatory do wektora checkedItems. Następnie zwraca ten wektor.

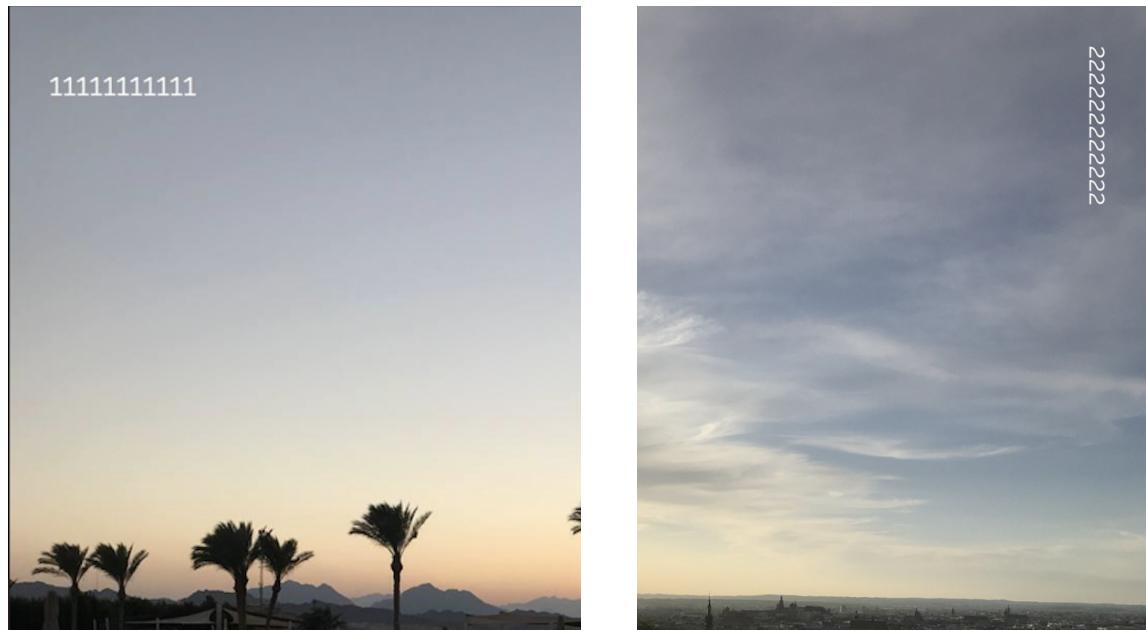
## 8 Testowanie

Podczas testowania aplikacji sprawdziliśmy działanie następujących funkcji:

- Load file with texts (D):

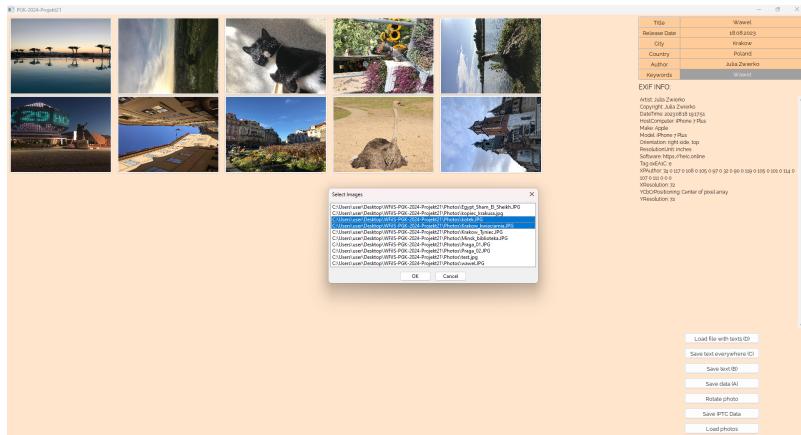


Rysunek 4: Działanie przycisku **Load file with texts (D)**

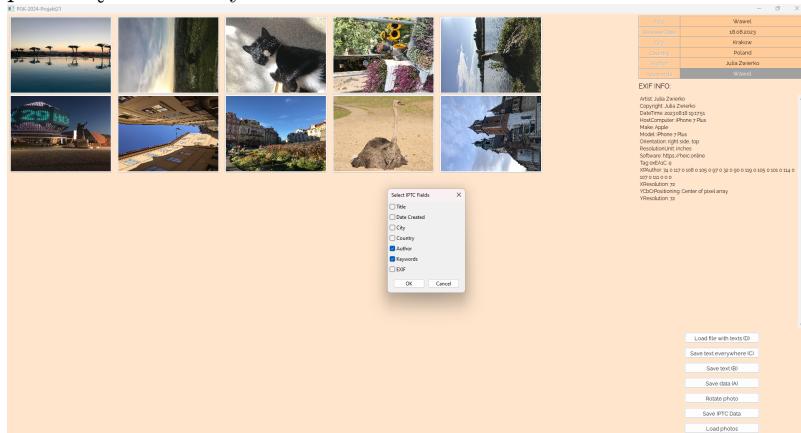


Rysunek 5: Wyniki: Wszystko działa poprawnie.

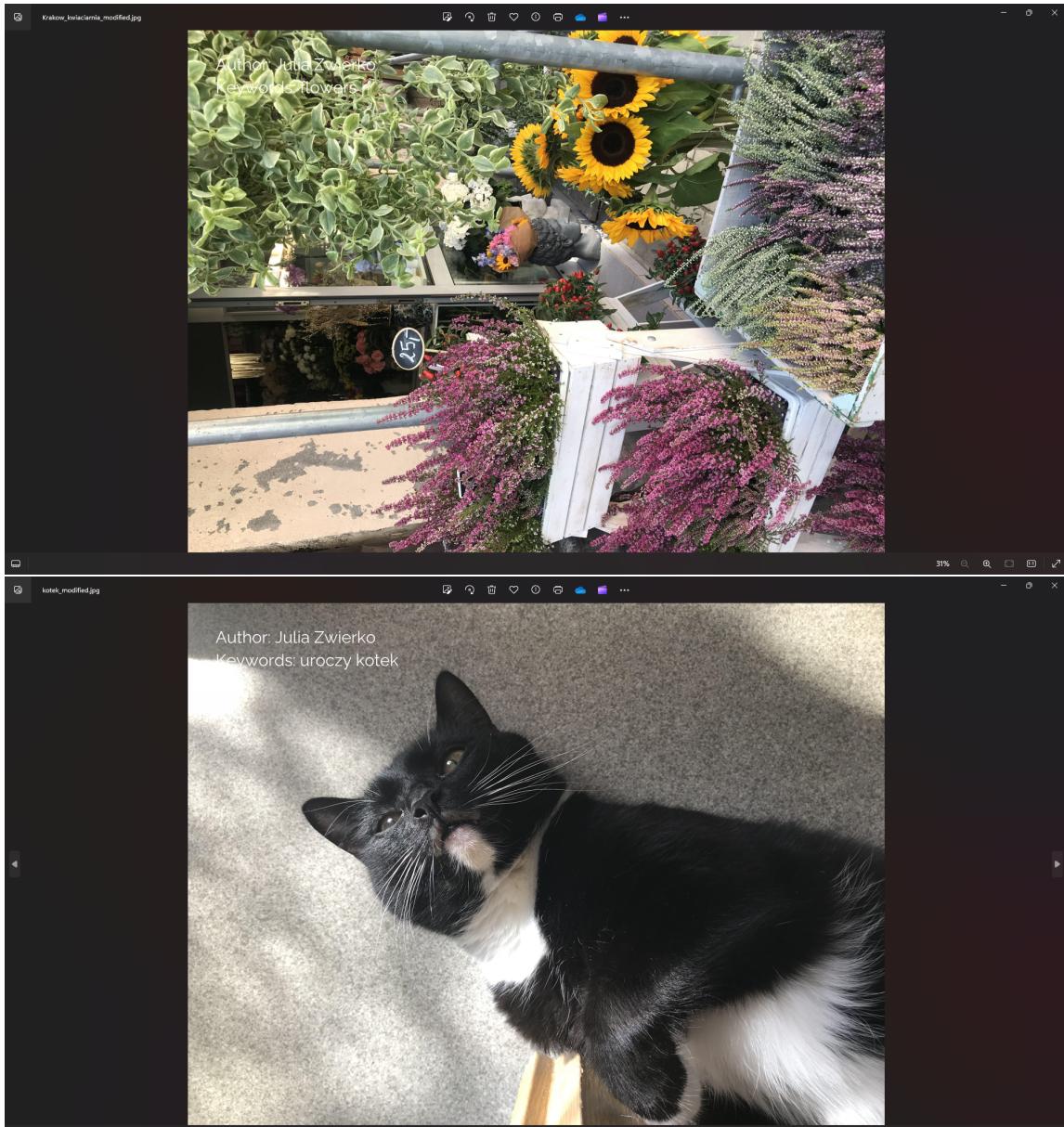
- Save text everywhere (C):



Rysunek 6: Sprawdzanie poprawności wyboru plików, na których kopiach będzie dodany tekst.

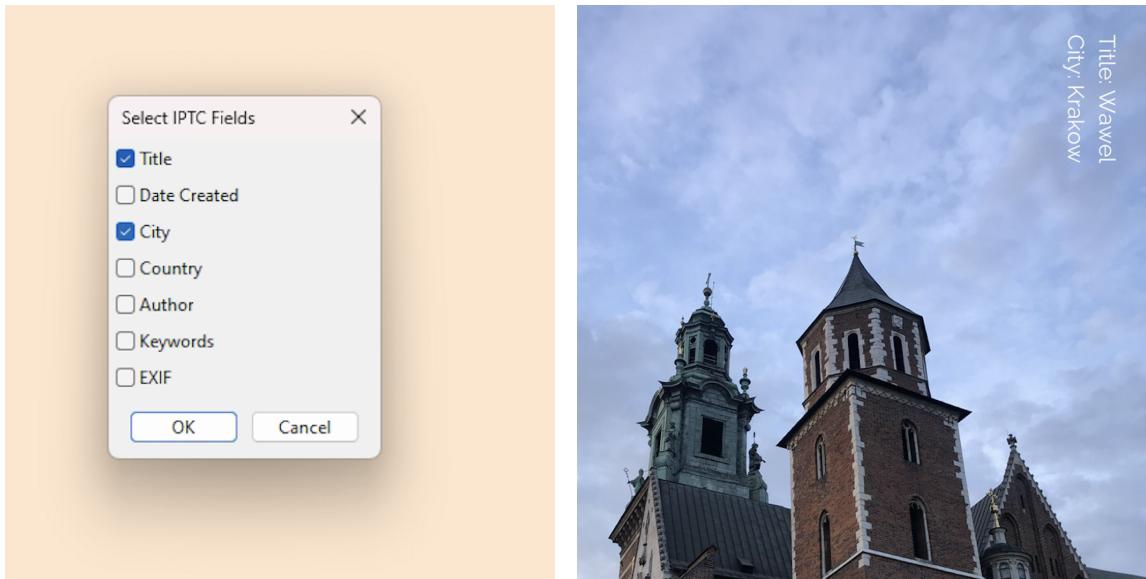


Rysunek 7: Sprawdzanie poprawności wyboru parametrów, które zostaną umieszczone w lewej górnej części kopii oryginalnych zdjęć wybranych przez użytkownika w poprzednim kroku.



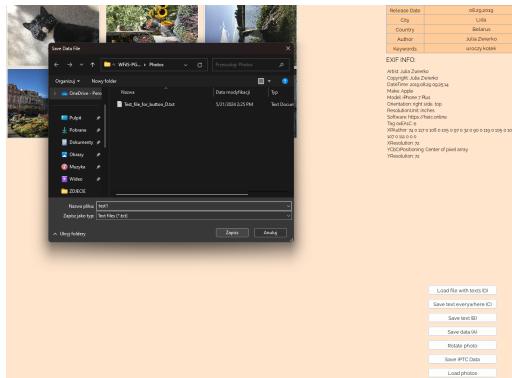
Rysunek 8: Wyniki testowania przycisku **Save text everywhere (C)** są poprawne.

- Save text (B):

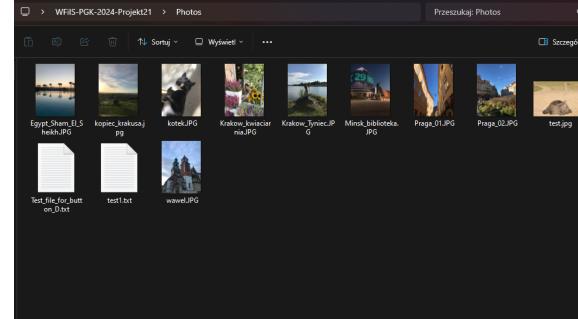


Rysunek 9: Wyniki poprawnego działania przycisku **Save text (B)**

- Save data (A):



Rysunek 10: Sprawdzanie działania wyboru folderu, w którym zostanie utworzony nowy plik tekstowy o nazwie wprowadzonej przez użytkownika.



Rysunek 11: Wynik: Plik został poprawnie utworzony we wskazanym folderze.

```

Path: C:\Users\user\Desktop\WFIS-PGK-2024-Projekt21\Photos\Egypt_Sharm_El_Sheikh.JPG

IPTC data:
Title: Sharm el-Sheikh - Al Nabq
Creation Date: 26.06.2021
City: Sharm el-Sheikh
Country: Egypt
Author: Julia Zwierko
Keywords: Egypt

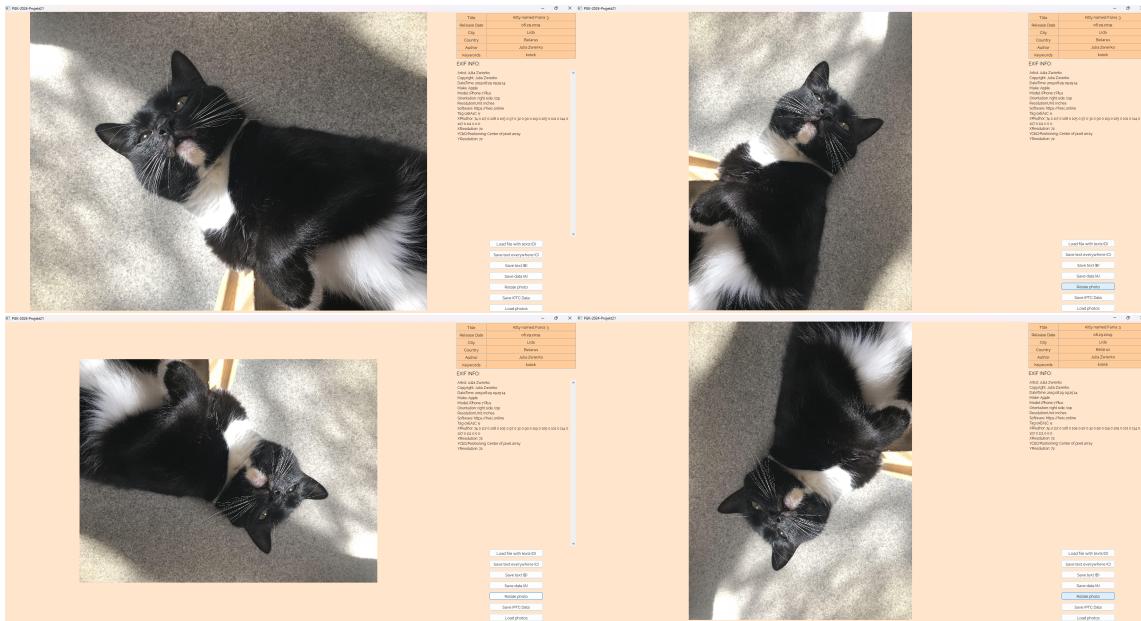
EXIF data:
Artist: Julia Zwierko
Copyright: Julia Zwierko
Tag 0xA1C: @e
XPAuthor: 74 0 117 0 108 0 105 0 97 0 32 0 90 0 119 0 105 0 101 0 114 0 107 0 111 0 0 0
-----
Path: C:\Users\user\Desktop\WFIS-PGK-2024-Projekt21\Photos\kopiec_krakusa.jpg

IPTC data:
Title: Kopiec Krakusa
Creation Date: 02.05.2024
City: Krakow
Country: Poland
Author: Julia Zwierko
Keywords: kopiec

EXIF data:
Artist: Julia Zwierko
Copyright: Julia Zwierko
Datetime: 2024-05-02 19:10:04
HostComputer: iPhone 7 Plus
Make: Apple
Model: iPhone 7 Plus
Orientation: right side, top
ResolutionUnit: inches
Software: https://heic.online
Tag 0xA1C: @e
XPAuthor: 74 0 117 0 108 0 105 0 97 0 32 0 90 0 119 0 105 0 101 0 114 0 107 0 111 0 0 0
-----
```

Rysunek 12: Kawałek tekstu z pliku stworzonego w poprzednim kroku. Wynik: Wszystko działa poprawnie.

- Rotate photo:



Rysunek 13: Poprawne działanie przycisku **Rotate photo**

- Save IPTC Data:

Title	Kitty named Fania 3
Release Date	08.29.2019
City	Lida
Country	Belarus
Author	Julia Zwierko
Keywords	uroczy kotek

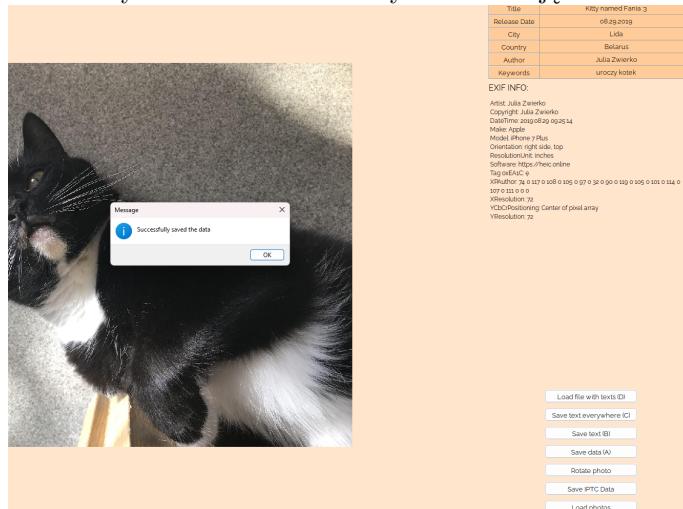
EXIF INFO:

```

Artist: Julia Zwierko
Copyright: Julia Zwierko
DateTime: 2019:08:29 09:25:14
Make: Apple
Model: iPhone 7 Plus
Orientation: right side, top
ResolutionUnit: inches
Software: https://heic.online
Tag oxEA1C:  
XPAuthor: 74 0 117 0 108 0 105 0 97 0 32 0 90 0 119 0 105 0 101 0 114 0
107 0 111 0 0
XResolution: 72
YCbCrPositioning: Center of pixel array
YResolution: 72

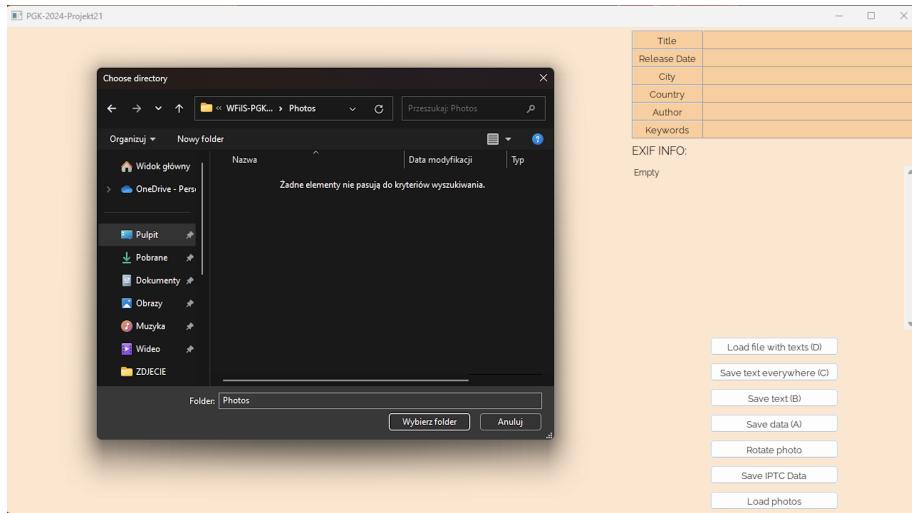
```

Rysunek 14: Zmiana danych IPTC zdjęcia.

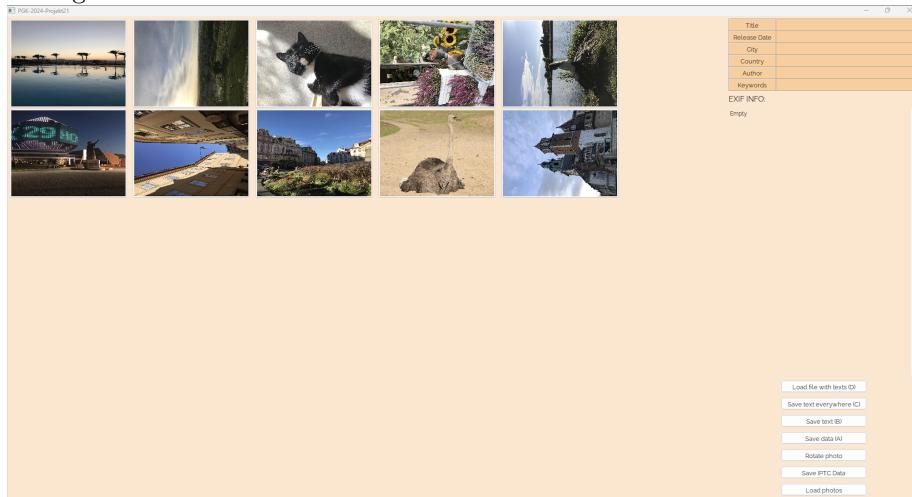


Rysunek 15: Wynik: Dane IPTC zdjęcia zostały pomyślnie zmienione.

- Load photos:



Rysunek 16: Po kliknięciu na przeciek, użytkownik ma możliwość wyboru dowolnego folderu.



Rysunek 17: Interfejs użytkownika wyświetla załadowane zdjęcia z wybranego folderu.

Dodatkowo przeprowadziliśmy serię dodatkowych testów, których wyniki zostały przedstawione na poniższych obrazkach.



Rysunek 18: Wyświetlenie danych EXIF a IPTC oraz obsługa podwójnego kliknięcia.

Oprócz tego, przetestowaliśmy również scenariusze błędne, takie jak próba załadowania folderów bez zdjęć JPEG oraz inne potencjalne błędy. W każdym z tych przypadków użytkownik otrzymuje odpowiednie komunikaty o błędach, które działają poprawnie. Ponadto, zauważliśmy, że rozmiar okna poprawnie się zmienia w zależności od preferencji użytkownika.

Na zakończenie, każdy z tych elementów przeszedł pomyślnie testy, co pozwoliło nam uznać projekt za gotowy.

## 9 Wdrożenie, raport i wnioski

Cały projekt udało się zrealizować w ciągu około 1,5 tygodnia, spełniając zarówno podstawowe wymagania, jak i rozszerzone oraz uznane przez nas dodatkowe funkcjonalności, które mogą być przydatne dla użytkownika, takie jak obracanie zdjęć czy edytowanie danych IPTC.

Przy tworzeniu naszego programu nauczyliśmy się lepiej wykorzystywać możliwości platformy GitHub, w tym obsługę repozytoriów Git oraz funkcjonalności takie jak pull requesty i ocena kodu innych osób. To zmuszało nas wszystkich do utrzymania naszego kodu w należytym porządku, aby każdy mógł łatwo zrozumieć kod napisany przez innych.

Największym wyzwaniem podczas tworzenia programu było odpowiednie zapoznanie się z dokumentacją oraz użycie metod z biblioteki FreeImage. Wspomniane metody były bardzo obciążające dla systemu (operacje I/O), przez co trzeba było zadbać o odpowiednią optymalizację programu, na przykład poprzez stworzenie odpowiednich metod wspomagających, takich jak pobieranie danych na podstawie istniejącej już bitmapy.

Podczas tworzenia programu można było napotkać wiele niespodziewanych błędów, takich jak ładowanie zmodyfikowanych już zdjęć do programu, co według nas nie było pożądanym zjawiskiem. Wszystkie te błędy trzeba było naprawić, co zajęło znaczna część czasu, ale można to nazwać zwykłym procesem tworzenia oprogramowania.

Wszystkie funkcjonalności, według nas, udało się ostatecznie zaimplementować bez błędów, a program wyświetla wszystkie zdjęcia i okienka w satysfakcyjującym nas tempie.