**Wyższa Szkoła Informatyki Stosowanej i Zarządzania**
**Grupa IO2**

# ALGORYTMY PRZETWARZANIA OBRAZÓW

( Architektura Projektu )

**Autor:**
Kacper Kaleta

**Prowadzący:**
dr hab. Korzyńska Anna

**Warszawa 2019/2020**

# Spis Treści:

# 1 Wprowadzenie

Program przetwarzania obrazów, nakładania zniekształceń radiometrycznych oraz ich korekcji. Przygotowany w semestrze letnim 2020 r. w ramach ukończenia uczelnianych zajęć: APO - Algorytmy Przetwarzania Obrazów.

Pełna nazwa projektu nr IO2_08:
Program prezentacji zasad przebiegu procesu wprowadzania i korekcji zniekształceń radiometrycznych z wykorzystaniem obrazów szaro-odcieniowych oraz ich fragmentów w postaci obrazowej a także tablic liczb.

Niniejszy dokument skupia się na budowie i strukturze oprogramowania pomijając opis funkcjonalności oraz praktycznych zastosowań aplikacji.

Aplikacja posiada budowę wielookienkową, z rozróżnieniem na 3 kategorie, *OknoNawigacji* (klasa **MainForm**), *OknaProjektu* (klasa **ImageForm**) oraz mniejsze okienka przyjmujące argumenty od użytkownika (klasa **PopupForm**).

# 2 Wymagania Systemowe

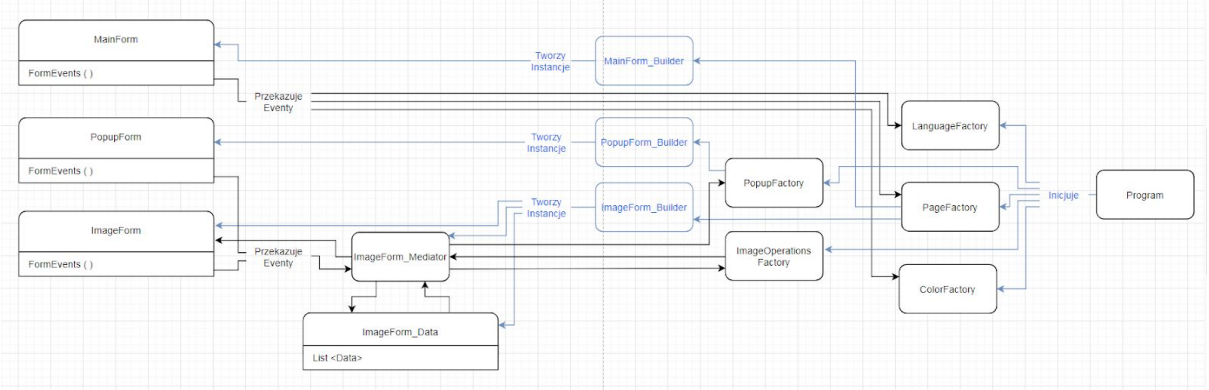
Uruchomienie programu wymaga dowolnego systemu operacyjnego Windows wspierającego framework .NET v4.7.2

# 3 Wykorzystane Narzędzia

- Visual Studio Community 2019 v16.5.4
- AutoMapper v9.0.0
- EmguCv v4.1.1.3497
- ZedGraph v5.1.7
- Kontrola wersji git w połączeniu z platformą github.com

# 4 Struktura Projektu

## 4.1 Spojrzenie Ogólne




[diagram załączony w pełnych wymiarach w osobnym pliku ***diagram.png***]

Podstawowym celem projektu jest zapewnienie elastycznej struktury pozwalającej na bezproblemowe dodawanie kolejnych funkcjonalności takich jak nowe wykresy, histogramy, tablice LUT (*Look up table*), nowe rodzaje okien, nowe motywy językowe i kolorystyczne. Jednak najważniejszym celem jest zapewnienie możliwości łatwego dodawania nowych operacji do przetwarzania obrazów. Operacje na obrazach były implementowane osobno poza aplikacją co ułatwiło rozbudowywanie oraz ewentualną refaktoryzacje obydwu. Scalenie w jedną aplikację nastąpiło bezproblemowo niedaleko terminu finalizacji programu.

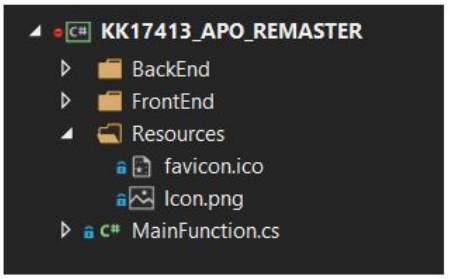
Struktura projektu wzorowana była architekturą *MVC*.
Rolę *Modelu* stanowią dane i główna logika aplikacji.
Za dane odpowiada klasa **ImageForm_Data** oraz klasy konfigurujące obecny stan **LanguageFactory** i **ColorFactory**. Główną logikę reprezentują klasy **ImageOperationsFactory**, **PopupFactory** oraz **PageFactory**.
Elementy *Modelu* uaktualniają *Widok* (*view*), czyli interfejs graficzny oparty na formularzach zbudowany z klas **ImageForm**, **PopupForm**, **MainForm**.
Formularze odbierają input od użytkownika i przesyłają go do *Kontrolera*.
Rola *Kontrolera* jest reprezentowana jedynie w postaci metod **FormEvents()**.
Eventy, bądź wydarzenia modyfikują *Widok* lub wywołują wybrane operacje głównej logiki aplikacji, modyfikując *Model*.

Klasy **PopupFactory** oraz **ImageOperationsFactory** są wykorzystywane tylko przy eventach związanych z modyfikacją konkretnych obrazów, konkretnych *OkienProjektu* (klasa **ImageForm**). Posiadają dostęp do danych obrazu **ImageForm_Data** poprzez klasę **ImageForm_Mediator** (wzorzec projektowy *Mediator*) której obowiązkiem jest udzielanie dostępu do danych oraz komunikacja między danymi a logiką modyfikującą dane.
Klasy **LanguageFactory**, **ColorFactory**, **PageFactory** otrzymują eventy bezpośrednio od *OknaNawigacji* (klasa **MainForm**) i odpowiadają za modyfikację globalną w tym konfiguracje aplikacji. Nie posiadają dostępu do danych z **ImageForm_Data**.

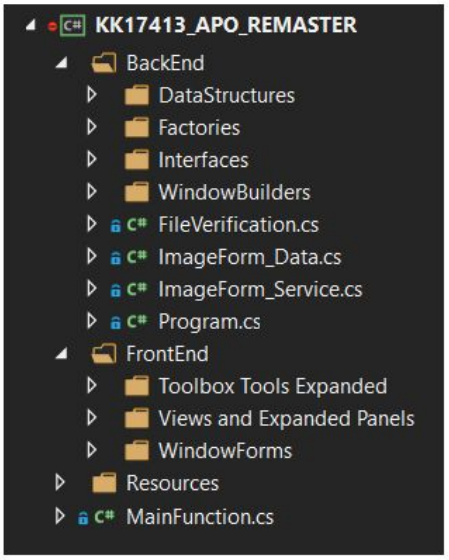
### 4.2 Porządek wewnątrz projektu - elementy wizualne

Porządek został zapewniony poprzez odseparowanie logiki elementów graficznych od reszty programu.

Formularze programu umożliwiają interakcję użytkownika z programem. Ich implementacje znajdują się wewnątrz katalogu **FrontEnd / WindowForms**.




Najbardziej skomplikowane formularze składają się z widoków (ang. Views - nie mylić z modelem MVC), a te z kolei są zbudowane z grupy paneli.
Panele odpowiedzialne są za jedną funkcjonalność jednocześnie. Przykładowo mogą określać położenie histogramu lub określać rozmieszczenie przycisków.
Zarówno widoki jak i panele znajdują się w katalogu **FrontEnd / Views and Expanded Panels**.




Katalog **FrontEnd / Toolbox Tools Expanded** zawiera w sobie implementacje nowych, autorskich narzędzi oraz implementacje poszerzonych wersji już istniejących, wbudowanych wewnątrz standardu *VisualStudio WindowsForms*.
Znajdują się tam autonomiczne komponenty takie jak klasa **Histogram** lub rozszerzona wersja wbudowanej klasy formularza (klasa **Form**).

### 4.3 Porządek wewnątrz projektu - odseparowanie konstruktorów

Najczęściej wykorzystywanym wewnątrz tego projektu wzorcem projektowym jest *Builder*. Niemalże każda klasa niebędąca odpowiedzialna za trzymanie danych lub niebędąca fabryką posiada przynajmniej jedną implementację buildera.

Logika buildera nie jest potrzebna głównej logice samego obiektu.
Odseparowanie procesu konstruowania obiektów od samych obiektów zwiększyło hermetyczność kodu oraz czytelność projektu, ułatwiło ponowne wykorzystanie kodu i pozwoliło na sprawniejszą nawigację między plikami projektu.
Dodatkowo pozwoliło na nieprzejmowanie się konfiguracją jak i implementacją obiektów. Nawet w projekcie jednoosobowym, zdarza się zapomnieć jak obsługiwać pewne fragmenty kodu, i na szczęście nie musiałem sobie przypominać.
Przykładowa klasa histogramu składa się z ponad 250 współdziałających ze sobą elementów graficznych. Wszelka konfiguracja, kolejność inicjowania, argumenty do konstruktorów, tym wszystkim zajął się za mnie gotowy builder.

Co większe, bardziej złożone obiekty powstawały za pomocą wzorca *ClassicBuilder*.
W przypadku mniejszych, mniej wymagających wystarczało zastosowanie *FluentBuildera*.

### 4.4 Elastyczność projektu

Najważniejszym wymaganiem nałożonym na strukturę programu było umożliwienie dodawania dużej ilości nowych funkcjonalności przy minimalnym nakładzie pracy oraz minimalnej ilości modyfikacji do wprowadzenia.

Wymaganie to zostało spełnione za pomocą kilkukrotnie
wykorzystanego wzorca projektowego *Fabryki*.
Wewnątrz projektu znajduje się fabryka odpowiedzialna za operacje na obrazach, fabryka konstruująca system okienek popup, fabryka trzymająca mapy tłumaczeń językowych, fabryka trzymająca mapy motywów kolorystycznych programu.
Każda z fabryk działa niezależnie od reszty.
Największą zaletą takiego podejścia jest automatyzacja pisania nowego kodu.
Utworzenie nowej operacji dla fabryki kończyło się na implementacji danej operacji.
Wszelkie podpięcie nowego fragmentu kodu odbywało się automatycznie tym samym znacznie zmniejszając nakład pracy oraz zwiększając czytelność projektu.

Negatywną cechą *Wzorców Kreacyjnych*, w tym również *Fabryk*, jest wzrost ilości kodu na początku projektu. Nie jest to w żadnym razie duża cena do zapłacenia. Korzyści wynikające z użycia fabryk są nieproporcjonalnie większe do kosztów.

# 5 Zebrane Statystyki O Projekcie

| | |
|---|---|
| **Ilość plików:** | 41 |
| **Ilość katalogów:** | 12 |
| **Ilość linijek napisanego kodu:** | 6 418 |
| **Ilość linijek kodu po kompilacji:** | 1 967 |
| **Ilość przeprowadzanych refaktoryzacji:** | 1 |
| **Ilość commitów na git:** | 248 |
| **Wykorzystane wzorce projektowe:**<br>- Fabryka<br>- ClassicBuilder<br>- FluentBuilder<br>- Mediator<br>- Wrapper | 5 |
| **Czas powstawania projektu:**<br>od **22.03.2020** do **13.06.2020**<br>2 miesiące i 22 dni | 83dni |