Politechnika Krakowska

Wydział Informatyki i Telekomunikacji

Studia Stacjonarne

Sprawozdanie z przedmiotu:

Zaawansowane Techniki Programowania

Temat Projektu:

Aplikacja do automatycznej oceny testów

Wykonali:

Krystian Paluch

Michał Rosa

Miłosz Okarma

1. Wstęp

Cel i Funkcjonalność Aplikacji do Oceniania Testów

Celem niniejszej pracy jest stworzenie aplikacji komputerowej, która będzie służyła do automatycznego oceniania testów pisanych w języku programowania Python. Narzędzie to, wykorzystując zaawansowane techniki przetwarzania obrazu, będzie analizować zdjęcia kart odpowiedzi i porównywać je ze wzorcowym szablonem. Na podstawie ocenionych testów aplikacja wygeneruje statystyki dla całej grupy oraz szczegółowy raport w formacie PDF.

Główne Zadania i Korzyści

Głównym zadaniem aplikacji jest odciążenie nauczycieli i egzaminatorów poprzez eliminację żmudnego i czasochłonnego procesu ręcznego sprawdzania dużej liczby testów. Automatyzacja tego procesu pozwoli zaoszczędzić cenny czas, który będzie można poświęcić na inne ważne aspekty pracy dydaktycznej. Ponadto, aplikacja ma za zadanie minimalizować ryzyko błędu w procesie oceniania, gwarantując większą precyzję i obiektywizm, co bezpośrednio przełoży się na sprawiedliwe wyniki dla wszystkich uczestników.

2. Kryteria oceny

a. Backend

Cały program został stworzony w języku Python, który wybrano ze względu na jego intuicyjną prostotę, przejrzystość składni, dynamiczne typowanie, a także bogactwo dostępnych bibliotek i wsparcie aktywnej społeczności deweloperów.

W ramach implementacji backendu wykorzystano szereg specjalistycznych bibliotek Pythonowych, kluczowych dla osiągnięcia zamierzonej funkcjonalności w obszarze przetwarzania obrazu i analizy danych. OpenCV służy do zaawansowanych operacji na obrazach, w tym do wczytywania, konwersji i wstępnego przygotowania danych. NumPy jest niezbędny do efektywnego zarządzania danymi liczbowymi i macierzowymi, co jest kluczowe w operacjach na pikselach obrazów, natomiast Pillow wspiera podstawową manipulację plikami graficznymi. Do zaawansowanej analizy obrazów, w tym detekcji cech i segmentacji, zastosowano scikit-image, z wykorzystaniem algorytmów takich jak `matchTemplate`, `getPerspectiveTransform` i `warpPerspective` do dopasowywania szablonów i korekcji perspektywy. Metoda `structural_similarity` pozwala na precyzyjne porównywanie obrazów i wizualizację różnic, co jest fundamentalne dla dokładnego oceniania testów. Raporty w formacie PDF są generowane za pomocą biblioteki FPDF.

Logika backendu obejmuje kluczowe procesy, począwszy od wczytywania i wstępnego przetwarzania obrazów kart odpowiedzi i szablonów wzorcowych. System zarządza również konfiguracją ustawień

testu, takich jak punktacja, progi ocen i liczba pytań, co umożliwia elastyczne dostosowanie do różnorodnych egzaminów. Centralną funkcjonalnością jest automatyczne ocenianie testów, polegające na precyzyjnym porównywaniu zeskanowanych kart ze wzorcem, rozpoznawaniu zaznaczonych odpowiedzi i obliczaniu wyników. Backend agreguje i przetwarza te dane, a także oferuje możliwość włączenia podglądu dla testu w celu weryfikacji. Ostatnim etapem jest generowanie kompleksowych raportów PDF, zawierających zarówno statystyki zbiorcze dla całej grupy, jak i szczegółowe raporty indywidualne dla każdego uczestnika, wraz z wynikami, zaznaczonymi odpowiedziami i ewentualnymi błędami.

b. Frontend

Interfejs użytkownika aplikacji został zaprojektowany z myślą o intuicyjności i efektywności pracy, umożliwiając nauczycielom i egzaminatorom łatwe zarządzanie procesem oceniania testów. Całość frontendu jest realizowana jako aplikacja desktopowa, wykorzystująca bibliotekę **CustomTkinter**, co zapewnia nowoczesny wygląd i płynną interakcję. Framework ten został wybrany ze względu na jego zdolność do tworzenia estetycznych i intuicyjnych interfejsów graficznych, przy jednoczesnym zachowaniu prostoty i łatwości obsługi, co pozwala na zaprojektowanie spójnego i przejrzystego doświadczenia użytkownika.

Struktura interfejsu opiera się na modułowym układzie komponentów, które odzwierciedlają kolejne etapy pracy z aplikacją. Górna belka nawigacyjna zawiera główne zakładki operacyjne, takie jak "Opcje", "Ustawienia testu", "Wczytaj szablon", "Wczytaj testy", "Oceń" oraz "Stwórz raport", co pozwala użytkownikowi na logiczne przechodzenie przez poszczególne etapy procesu. Na pierwszym ekranie, dedykowanym przygotowaniu danych, znajduje się wizualizacja "Karty odpowiedzi" służąca jako podgląd wczytanego szablonu, a także lista wczytanych plików testowych z opcją ich usunięcia i dodawania kolejnych. Drugi ekran prezentuje szczegółowe wyniki ocenionych testów w formie czytelnej listy, gdzie każdy wpis zawiera nazwę pliku, liczbę punktów, ocenę oraz status (Zaliczony/Niezaliczony, wyraźnie oznaczony kolorem), z możliwością włączenia podglądu dla każdego testu. Dodatkowo, panel "PODSUMOWANIE" po prawej stronie ekranu dostarcza zagregowanych danych dla całej grupy, w tym łączną liczbę ocenionych testów, liczbę testów zaliczonych i niezliczonych, średnią ocenę oraz średnią ilość punktów, zapewniając szybki wgląd w ogólną efektywność.

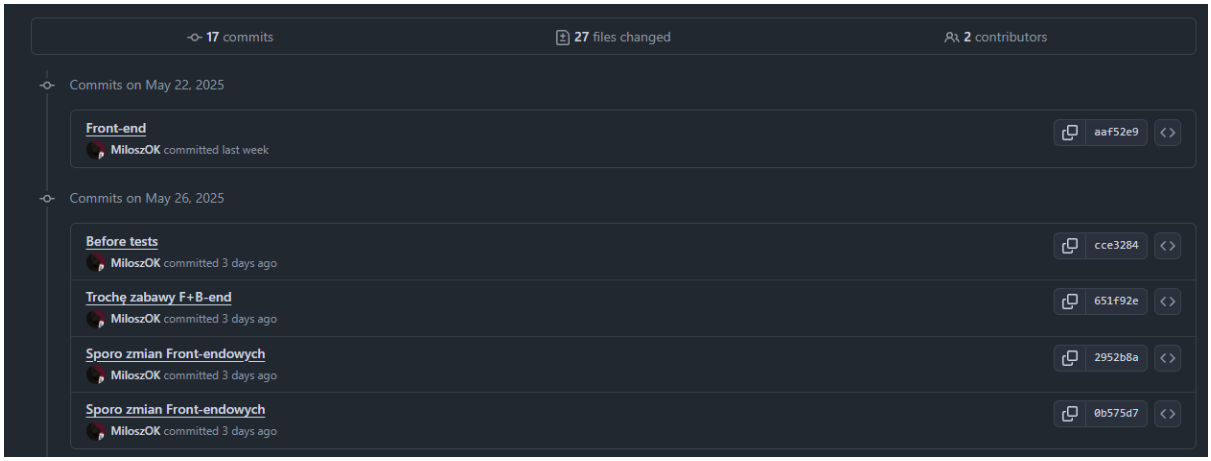
Komunikacja między frontendem a backendem odbywa się w sposób synchroniczny, umożliwiając efektywne przekazywanie danych wejściowych, takich jak zdjęcia kart odpowiedzi czy konfiguracja testu, do modułów przetwarzających w backendzie. Frontend jest odpowiedzialny za wysyłanie żądań do backendu w celu uruchomienia procesów analizy i obliczeń, a następnie za odbieranie i wizualizowanie zwracanych przez backend danych, takich jak obliczone wyniki testów czy wygenerowane raporty, co zapewnia użytkownikowi bieżące informacje zwrotne i dostęp do pełnej funkcjonalności. Cały projekt interfejsu skupia się na czytelności i użyteczności poprzez wyraźne etykiety, intuicyjne przyciski i wizualne wyróżnienie statusów.

c. Zarządzanie projektem

Zarządzanie projektem aplikacji do automatycznego oceniania testów odbywa się w sposób zwinny, z wykorzystaniem platformy GitHub jako centralnego repozytorium kodu źródłowego. Takie podejście gwarantuje efektywną współpracę zespołu, transparentność postępów oraz możliwość szybkiego reagowania na zmiany i pojawiające się wyzwania.

W ramach repozytorium utrzymywane są dwie główne gałęzie (branches): `main` oraz `testowa`. Gałąź `main` stanowi stabilną wersję produkcyjną kodu, zawierającą wyłącznie w pełni przetestowane i zatwierdzone funkcjonalności. Wszelkie nowe funkcje, poprawki błędów czy eksperymentalne rozwiązania są rozwijane w gałęzi `testowa`. Dopiero po gruntownym przetestowaniu i upewnieniu się, że dany fragment kodu jest stabilny i spełnia założenia, następuje jego integracja z gałęzią `main`. Taki dwugałęziowy model pracy minimalizuje ryzyko wprowadzenia niestabilnego kodu do głównej wersji aplikacji.

Struktura repozytorium



Fragment historii commitów i statystki

d. Testowanie

W ramach prac nad aplikacją przeprowadzono testy jednostkowe przy użyciu biblioteki `unittest`. Testy obejmują kluczowe funkcje logiki przetwarzania obrazu, interfejsu użytkownika oraz mechanizmów oceny wyników.

Rodzaje przeprowadzonych testów:

- **Testy funkcjonalne** metod takich jak `convert_to_opencv`, `find_corners_closest_to_edges`, `process_image` – weryfikujące poprawność działania przekształceń obrazu oraz detekcji narożników.
- **Testy GUI (z użyciem mocków)** funkcji z modułu `buttons`, takich jak `exit_program`, `toggle_help_text`, `show_page1`, `show_page2`, `toggle_settings_option` – testujące zachowanie widżetów i obsługę interakcji użytkownika.

- **Testy logiki oceny** – sprawdzające przekształcanie punktowych wyników testów na oceny końcowe według zadanych progów (`grade_table`).

- **Testy inicjalizacji wyglądu aplikacji** – weryfikujące, czy przy starcie programu ustawiana jest odpowiednia kolorystyka i tryb (mockowanie `ctk.set_appearance_mode` i `set_default_color_theme`).

Pokrycie kodu testami:

- Pokryto testami wszystkie główne ścieżki funkcjonalne aplikacji: przetwarzanie obrazu, logika oceny oraz zachowania komponentów GUI.

- Testy obejmują zarówno przypadki poprawne, jak i sytuacje graniczne (np. nietypowe rozkłady narożników, różne proporcje obrazów, interakcje użytkownika z widżetami w różnych stanach).

- Ze względu na trudności związane z bezpiecznym testowaniem funkcji związanych z hasłami – testy tych fragmentów zostały zaniechane.

Narzędzia testowe:

- **Python `unittest`** – główna biblioteka wykorzystywana do budowy i uruchamiania testów.

- **`unittest.mock` (MagicMock, patch)** – użyte do izolowania testowanych komponentów od zależności zewnętrznych, takich jak GUI, system plików, czy zewnętrzne moduły.

- **`Pillow, NumPy, OpenCV`** – biblioteki pomocnicze, używane do przygotowania danych wejściowych dla testów funkcji obrazowych.

Testy zostały zorganizowane w jednej klasie (`TestLogicFunctions`) i można je uruchamiać niezależnie od reszty aplikacji. Dzięki zastosowaniu mocków możliwe było przetestowanie logiki nawet w przypadku elementów silnie zależnych od GUI, co zwiększa niezawodność i jakość całego systemu.

e. Bezpieczeństwo aplikacji

Autoryzacja użytkownika:

- Aplikacja wykorzystuje **lokalne uwierzytelnianie hasłem** przed uzyskaniem dostępu do głównej funkcjonalności.
- Lista dozwolonych haseł przechowywana jest w pliku `Assets/passwords.json`.
- Hasło jest weryfikowane lokalnie przy użyciu prostego formularza logowania w GUI (customtkinter).
- W przypadku błędnego hasła, użytkownik nie ma dostępu do aplikacji.

Ochrona danych:

- Aplikacja nie przechowuje danych wrażliwych użytkownika.
- Przechowywane są jedynie tymczasowe dane ocenianych testów (lokalnie).

Bezpieczeństwo plików graficznych:

- Importowane obrazy są analizowane lokalnie — brak przesyłania przez sieć.
- Obsługa jedynie plików PNG minimalizuje ryzyko wstrzyknięcia szkodliwego kodu.

3. Architektura aplikacji

Typ architektury: Monolityczna aplikacja desktopowa.

Główne komponenty:

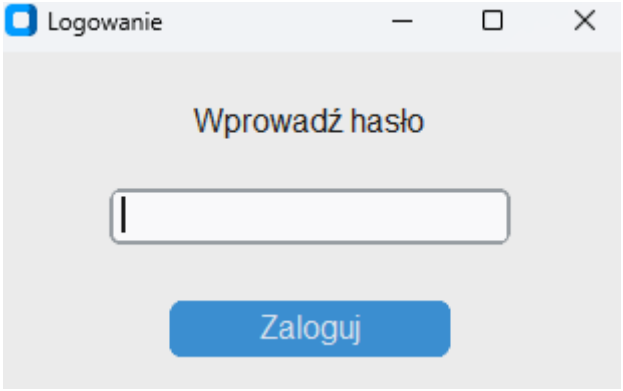
- **Frontend:**
  - Panele interaktywne (załaduj szablon, testy, ustawienia testu)
  - Obsługa widoku wyników i generowania raportu

- Komunikacja synchronizowana z backendem

- **Backend:**
  - Przetwarzanie obrazów z użyciem OpenCV (m.in. transformacja perspektywiczna, szukanie narożników)
  - Porównywanie fragmentów z użyciem `cv2.matchTemplate`
  - Analiza różnic z `structural_similarity` i tworzenie map błędów
  - Generowanie raportu PDF

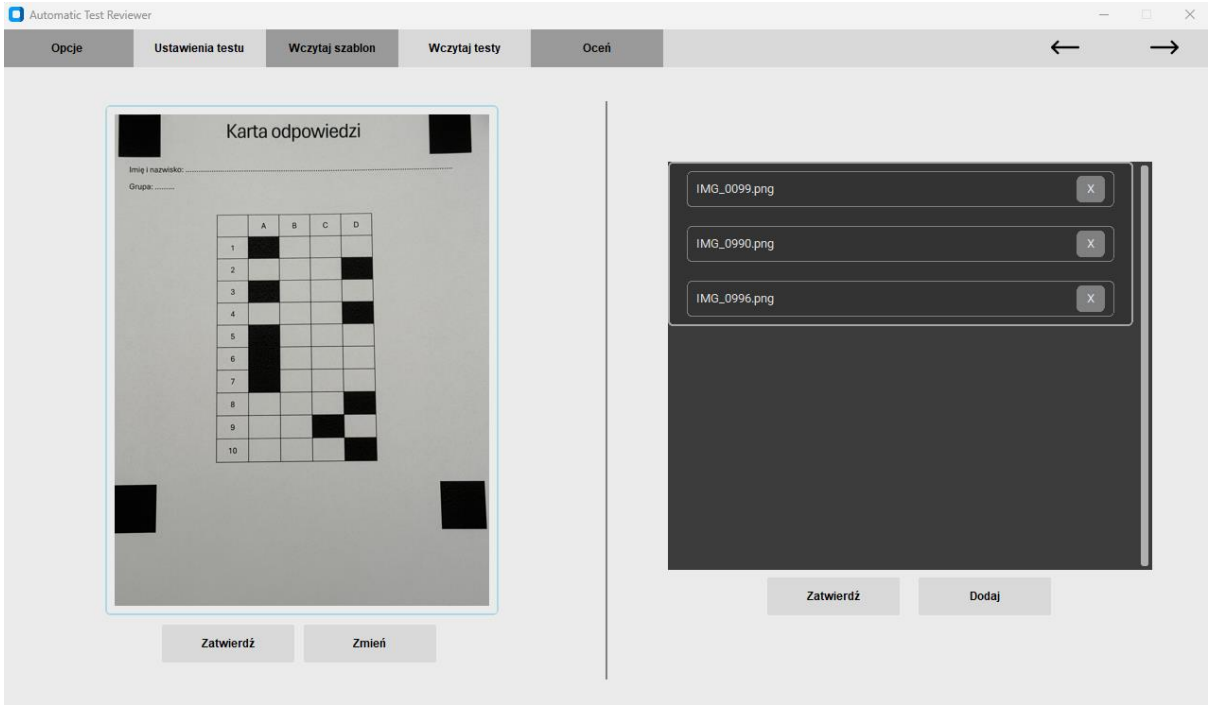
Model architektury: Opis zastosowanego podejścia (monolit, mikroserwisy), główne komponenty aplikacji i ich interakcje.

4. Aplikacja

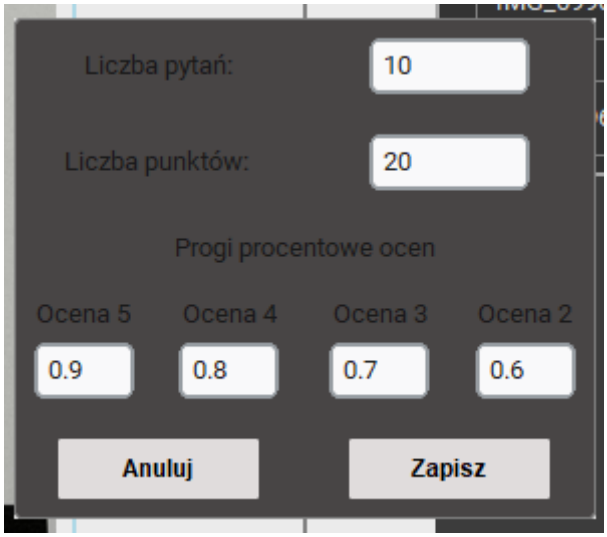


Walidacja przy włączaniu aplikacji

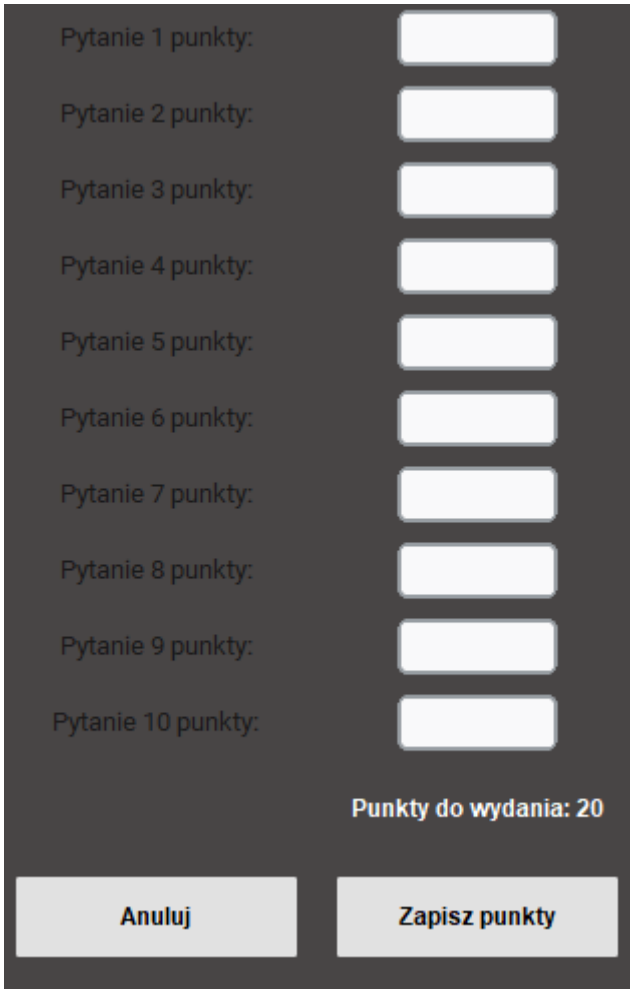
Ekran startowy aplikacji



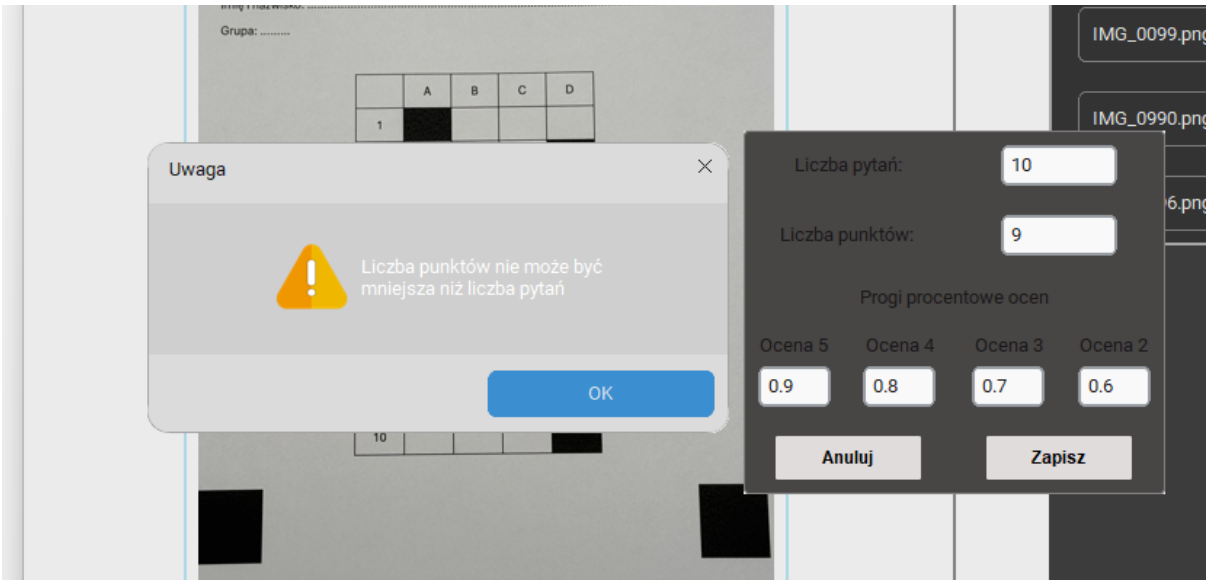
Ekran aplikacji po załadowaniu szablonu testu i testów gotowych do oceny



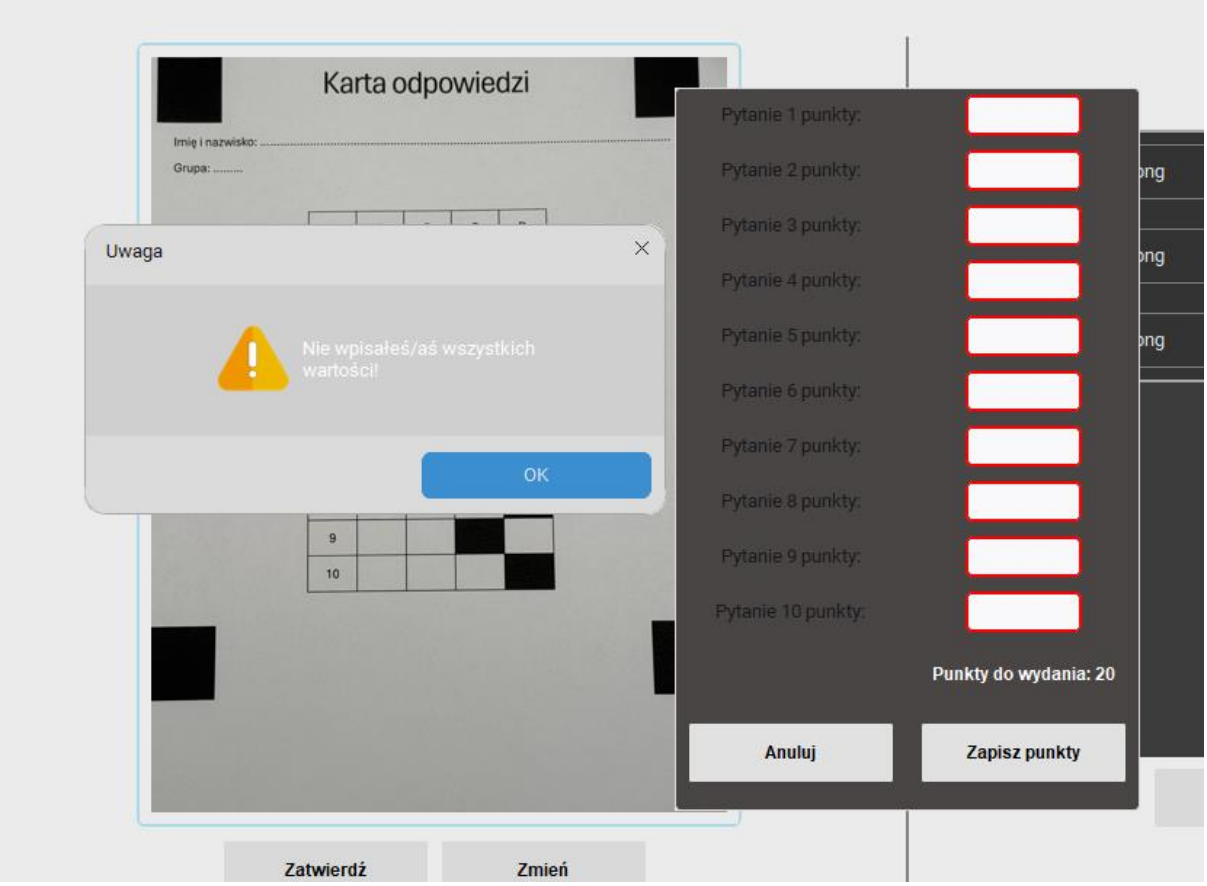
Zakładka Ustawienia Testu w której można ustawiać wszystkie parametry testu



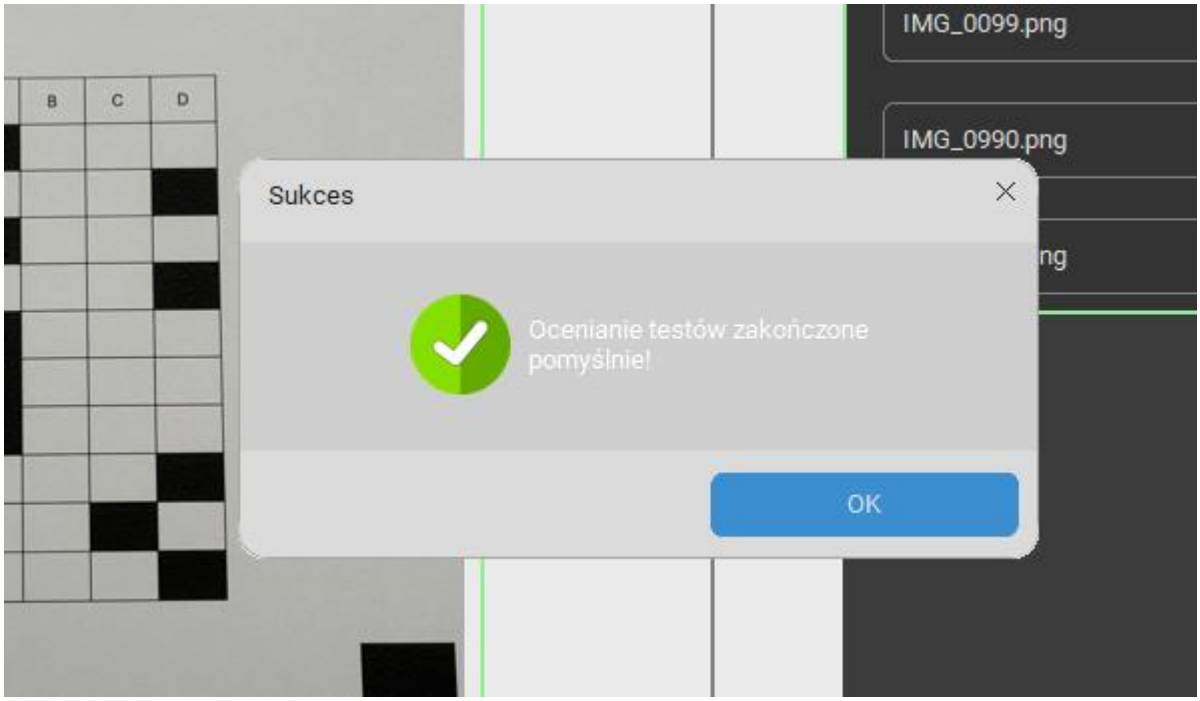
Menu przypisania punktów



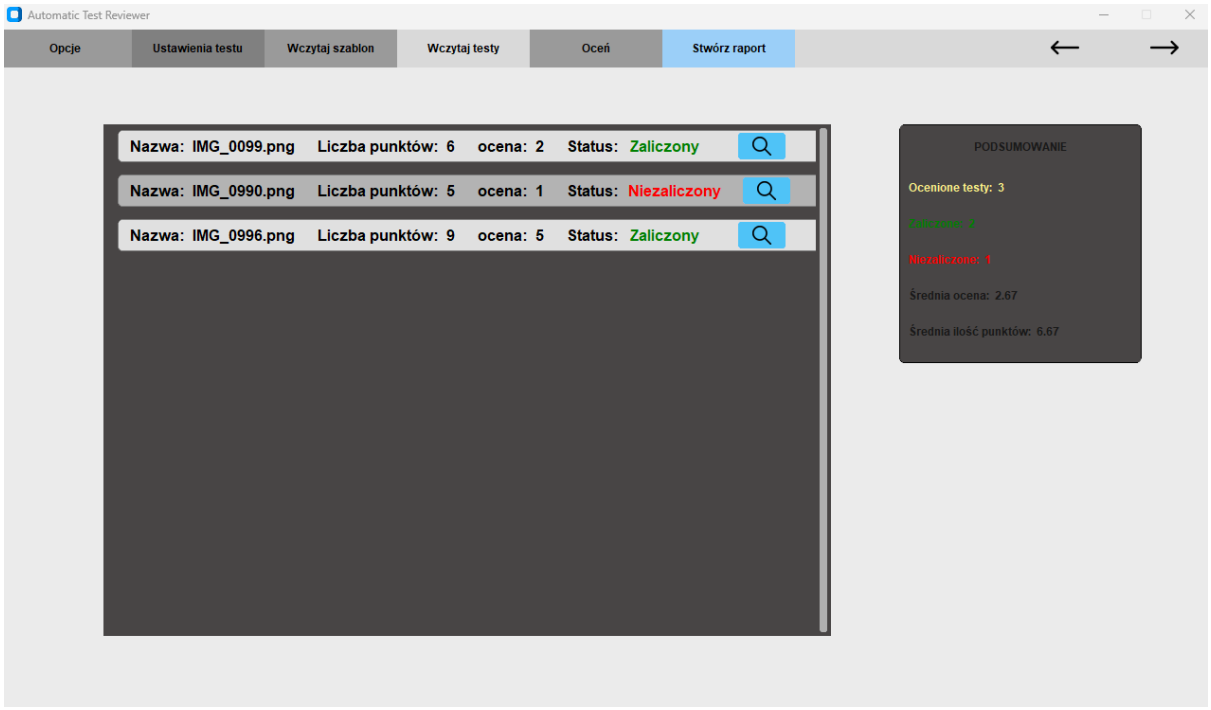
Ostrzeżenie gdy chcemy przypisać mniej punktów niż pytań.



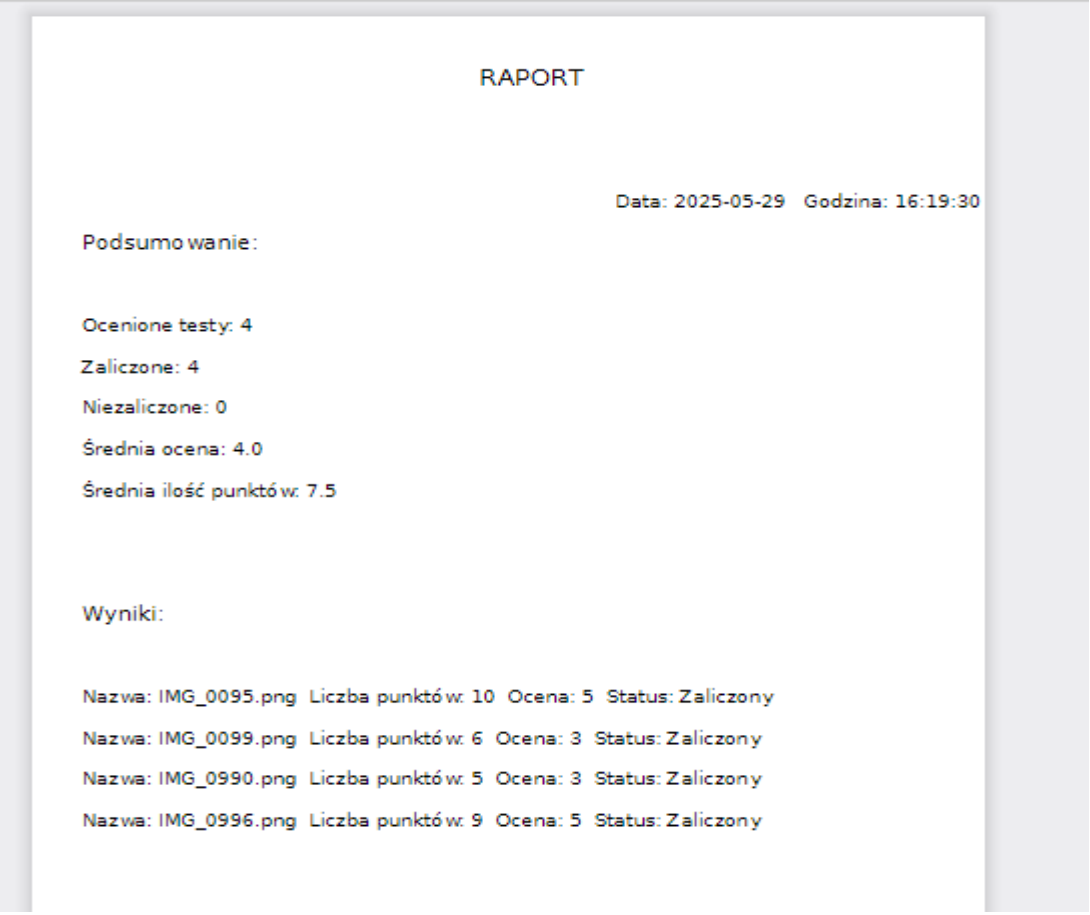
Ostrzeżenie w przypadku próby zapisania punktów bez ich przydzielenia



Komunikat po kliknięciu Oceń



Drugi ekran po wykonaniu oceny gdzie znajdują sie zbiorcze jak i pojedyncze statystki dla testów



RAPORT

Data: 2025-05-29 Godzina: 16:19:30

Podsumowanie:

Ocenione testy: 4

Zaliczone: 4

Niezaliczone: 0

Średnia ocena: 4.0

Średnia ilość punktów: 7.5

Wyniki:

Nazwa: IMG_0095.png Liczba punktów: 10 Ocena: 5 Status: Zaliczony

Nazwa: IMG_0099.png Liczba punktów: 6 Ocena: 3 Status: Zaliczony

Nazwa: IMG_0990.png Liczba punktów: 5 Ocena: 3 Status: Zaliczony

Nazwa: IMG_0996.png Liczba punktów: 9 Ocena: 5 Status: Zaliczony

Fragment raportu który powstaje po kliknięciu Stwórz raport

5. Podsumowanie

Aplikacja została zrealizowana zgodnie z założeniami projektowymi. Zbudowano narzędzie umożliwiające automatyczne ocenianie testów jednokrotnego wyboru na podstawie wzorca, analizujące zeskanowane odpowiedzi i generujące podsumowanie wyników. Użytkownik może z łatwością ustawić liczbę pytań, przypisać punkty, zdefiniować progi ocen, porównać testy z szablonem, a następnie wygenerować raport w formacie PDF.

Projekt osiągnął wszystkie cele funkcjonalne. System działa stabilnie, a interfejs graficzny jest intuicyjny i prosty w obsłudze. Przeprowadzone testy jednostkowe potwierdziły poprawność kluczowych funkcji aplikacji, takich jak przetwarzanie obrazu, ocena punktowa, czy funkcje GUI.

Problem	Rozwiązanie
Trudności w odczycie nieczytelnych znaków na skanach	Zastosowanie filtrów progowych i poprawa algorytmu binarizacji obrazu
Różne proporcje szablonów	Wprowadzenie funkcji ręcznego dopasowania oraz kalibracji punktów kontrolnych
Wykrywanie zaznaczeń w różnych odcieniach	Wprowadzenie progowania pikseli i eliminacja fałszywych detekcji
Duplikaty nazw plików testowych	Dodanie mechanizmu sprawdzania i unikania duplikatów przy ocenie

W przyszłości możliwe są następujące ulepszenia:

- **Integracja z chmurą lub bazą danych** – przechowywanie wyników i synchronizacja wielu urządzeń.
- **Ulepszone OCR** – automatyczne rozpoznawanie imion i nazwisk uczniów.
- **Obsługa testów wielokrotnego wyboru** – rozszerzenie funkcjonalności.
- **Mobilna wersja aplikacji** – umożliwiająca robienie zdjęć testów i ocenę w czasie rzeczywistym.