



POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Świątkowski Łukasz

Nr albumu 34532

Analiza porównawcza przenośnych bibliotek GUI dla języka C++

Praca magisterska napisana pod
kierunkiem:

dr inż. Mariusz Trzaska, prof.
PJATK

Warszawa, lipiec 2026

Streszczenie

Niniejsza praca dotyczy problemu wyboru biblioteki graficznego interfejsu użytkownika dla języka C++. Programiści stają przed wyborem jednej z wielu bibliotek, a decyzja ta znacząco wpływa na architekturę projektu oraz koszt utrzymania kodu. Brakuje przy tym opracowań porównujących te narzędzia w realistycznym scenariuszu, na podstawie pełnej implementacji aplikacji.

Praca przedstawia porównanie czterech popularnych bibliotek: Qt, wxWidgets, Dear ImGui oraz GTK. W tym celu zaimplementowano prototyp przeglądarki zdjęć tak, aby w każdej z bibliotek wyglądał i działał jak najbardziej podobnie. Biblioteki oceniono pod kątem wygody programowania, przenoszalności między systemami Windows i Linux, wydajności i zużycia zasobów, rozmiaru pliku wynikowego i zależności, zgodności ze standardem C++ oraz jakości dokumentacji.

Analiza pokazuje, że żadna z bibliotek nie jest najlepsza we wszystkich kryteriach. Ich wady i zalety wynikają z odmiennych rozwiązań projektowych przyjętych w każdej z nich. Na podstawie tych spostrzeżeń sformułowano rekomendacje doboru biblioteki w zależności od potrzeb projektu.

Słowa kluczowe: biblioteka GUI, C++, aplikacja, porównanie, testowanie, Windows, Linux

Podziękowania

Autor pracy wyraża podziękowanie promotorowi, dr inż. Mariuszowi Trzasce, za zaangażowanie oraz poświęcony czas.

Spis treści

1. WSTĘP	6
1.1. Cel pracy	6
1.2. Rozwiązania przyjęte w pracy	6
1.3. Rezultat pracy	7
1.4. Organizacja pracy	7
2. ISTNIEJĄCE ROZWIĄZANIA.....	8
2.1. Sposoby tworzenia GUI w C++	8
2.1.1 Tryb zachowany (<i>Retained Mode</i>).....	8
2.1.2 Tryb natychmiastowy (<i>Immediate Mode</i>)	9
2.1.3 Natywne kontrolki a renderowanie własne	9
2.2. Analiza architektur porównywanych bibliotek	10
2.2.1 Qt.....	10
2.2.2 wxWidgets	11
2.2.3 Dear ImGui	11
2.2.4 GTK.....	12
2.3. Istniejące porównania bibliotek GUI	12
2.3.1 Wiki wxWidgets	12
2.3.2 Artykuły online	13
2.3.3 Fora internetowe.....	13
3. KRYTERIA PORÓWNAWCZE I METODOLOGIA.....	14
3.1. Wybór aplikacji testowej	14
3.2. Wymagania funkcjonalne aplikacji testowej.....	14
3.2.1 Przeglądanie i wyświetlanie obrazów	14
3.2.2 Skalowanie i obracanie	15
3.2.3 Wyświetlanie metadanych	15
3.2.4 Eksplorator plików	15
3.2.5 Modyfikacja pliku.....	16
3.2.6 Opcja drukowania.....	16
3.2.7 Zakładka ustawień.....	16
3.2.8 Lista ostatnio otwieranych plików.....	17
3.3. Kryteria porównawcze	17
3.3.1 Łatwość implementacji i wygoda programowania	17
3.3.2 Przenoszalność między systemami operacyjnymi.....	17
3.3.3 Wydajność i zużycie zasobów	18
3.3.4 Rozmiar pliku binarnego	18
3.3.5 Zgodność ze standardem C++	19
3.3.6 Jakość dokumentacji	19
4. ZASTOSOWANE TECHNOLOGIE I NARZĘDZIA.....	21
4.1. C++20	21
4.2. CMake.....	21
4.3. Conan	21
4.4. SQLite	22
4.5. Technologie związane z zastosowanymi bibliotekami	22
4.5.1 Qt Charts.....	22
4.5.2 SDL2, OpenGL.....	22
4.5.3 STB do obsługi obrazów.....	23
4.6. Środowiska programistyczne	23
4.6.1 Visual Studio	23
4.6.2 Qt Creator	24

5.	PROJEKT I IMPLEMENTACJA PROTOTYPU	26
5.1.	Architektura rozwiązania z wymiennymi backendami	26
5.1.1	Budowanie dla wielu systemów operacyjnych.....	27
5.1.2	Plik conanfile i opcje kompilacji	28
5.1.3	Mechanizm wyboru backendu w czasie kompilacji	29
5.1.4	Konfiguracja CMake	30
5.1.5	Wspólny schemat bazy danych	31
5.2.	Implementacja Qt.....	31
5.2.1	Okno główne i układ interfejsu.....	32
5.2.2	System sygnałów i slotów	33
5.2.3	Przetwarzanie i wyświetlanie obrazów	34
5.2.4	Histogram z wykorzystaniem Qt Charts	36
5.2.5	Integracja z bazą danych QSqlDatabase	37
5.2.6	Drukowanie	38
5.3.	Implementacja wxWidgets.....	39
5.3.1	Okno główne.....	40
5.3.2	Przetwarzanie i wyświetlanie obrazów	41
5.3.3	Baza danych SQLite3	43
5.3.4	Drukowanie	44
5.4.	Implementacja Dear ImGui.....	45
5.4.1	Inicjalizacja z SDL2 i OpenGL	45
5.4.2	Okno główne.....	46
5.4.3	Wczytywanie obrazów biblioteką STB.....	47
5.4.4	Obracanie obrazów współrzędnymi UV.....	48
5.5.	Implementacja GTK.....	49
5.5.1	Okno główne.....	50
5.5.2	Sygnały GObject.....	51
5.5.3	Rysowanie z Cairo i GdkPixbuf.....	52
6.	TESTY I POMIARY	53
6.1.	Środowisko testowe	53
6.2.	Testy wydajności renderowania.....	54
6.2.1	Czas cyklu renderowania	54
6.2.2	Czas wczytywania plików.....	55
6.3.	Pomiary zużycia pamięci i procesora.....	56
6.3.1	Zużycie procesora	56
6.3.2	Zużycie pamięci	56
6.4.	Rozmiar plików binarnych i zależności	58
6.5.	Czas kompilacji.....	59
6.6.	Przenoszalność	60
6.7.	Ocena wygody programowania.....	62
6.8.	Porównanie dokumentacji.....	63
6.9.	Zestawienie wyników	64
6.10.	Rekomendacje w zależności od potrzeb projektu	65
7.	PODSUMOWANIE I WNIOSKI	67
7.1.	Wady i zalety rozwiązania	67
7.2.	Wnioski	68
PRACE CYTOWANE		70
SPIS RYSUNKÓW		74
SPIS TABEL.....		75
SPIS LISTINGÓW.....		76

1. Wstęp

Język C++ pozostaje jednym z dominujących wyborów przy tworzeniu aplikacji desktopowych [43]. Programiści korzystający z tego języka dysponują szerokim wyborem bibliotek służących do budowy graficznych interfejsów użytkownika (*Graphical User Interface*, GUI), z których każda reprezentuje odrębne podejście projektowe, filozofię API oraz kompromisy między łatwością użycia a elastycznością. Wybór odpowiedniej biblioteki ma istotny wpływ na architekturę projektu, czas jego realizacji, przenoszalność między platformami oraz długoterminowe koszty utrzymania kodu.

1.1. Cel pracy

Celem pracy jest przeprowadzenie wielokryterialnego porównania czterech popularnych bibliotek GUI dla języka C++: Qt, wxWidgets, Dear ImGui oraz GTK. Analiza obejmuje zarówno aspekty techniczne, jak i praktyczne, istotne z punktu widzenia programisty tworzącego aplikacje desktopowe. Porównanie jest realizowane poprzez implementację tej samej aplikacji, przeglądarki zdjęć przy użyciu każdej z wymienionych bibliotek. Jednolita specyfikacja funkcjonalna aplikacji testowej umożliwia obiektywne zestawienie bibliotek w kontekście rzeczywistych zadań programistycznych. Do badanych kryteriów należą:

- łatwość implementacji i wygoda programowania;
- przenoszalność między systemami operacyjnymi (Windows i Linux);
- wydajność i zużycie zasobów systemowych;
- rozmiar wynikowego pliku binarnego oraz zależności;
- zgodność ze standardem C++20;
- jakość dokumentacji.

1.2. Rozwiązania przyjęte w pracy

Realizacja celu pracy wymagała podjęcia szeregu decyzji projektowych dotyczących wyboru bibliotek, metody porównania oraz środowiska implementacyjnego. Poniższe podrozdziały omawiają przyjęte rozwiązania wraz z uzasadnieniem dokonanych wyborów.

Do analizy wybrano cztery biblioteki, które cieszą się szerokim zastosowaniem w środowisku programistów C++ i reprezentują odmienne podejścia do projektowania GUI.

Za podstawę porównania przyjęto implementację jednolitej aplikacji testowej, przeglądarki zdjęć, w każdej z wybranych bibliotek. Podejście to zapewnia porównywalność wyników, gdyż każda implementacja realizuje ten sam zestaw wymagań funkcjonalnych. Aplikacja testowa obejmuje zarówno podstawowe elementy interfejsu, jak i komponenty zaawansowane. Dzięki temu porównanie nie ogranicza się do prostych przykładów, lecz odzwierciedla realne warunki tworzenia oprogramowania desktopowego.

Prototyp zaprojektowano z myślą o wymiennych backendach, cały kod aplikacji jest współdzielony, a wybór biblioteki GUI następuje na etapie kompilacji. Takie podejście minimalizuje wpływ różnic w logice biznesowej na wyniki porównania i gwarantuje, że mierzone są właściwości samych bibliotek, a nie odmienności w implementacji funkcjonalności.

1.3. Rezultat pracy

Rezultatem niniejszej pracy są dwa powiązane ze sobą elementy: prototyp aplikacji oraz analiza porównawcza bibliotek GUI. Prototyp stanowi przeglądarkę zdjęć zaimplementowaną czterokrotnie, oddzielnie dla każdej z badanych bibliotek (Qt, wxWidgets, Dear ImGui oraz GTK), przy użyciu wspólnej bazy kodu i jednolitego zestawu wymagań funkcjonalnych.

Każda implementacja obsługuje:

- przeglądanie i wyświetlanie obrazów w formatach JPEG, PNG i BMP,
- skalowanie i obracanie obrazu bez modyfikacji pliku na dysku,
- wyświetlanie metadanych zdjęcia,
- eksplorator plików umożliwiający wybór zdjęcia z katalogu,
- usuwanie i zapis plików,
- zmianę jasności i kontrastu obrazu.

Analiza porównawcza obejmuje wyniki pomiarów wydajności renderowania, zużycia pamięci i czasu kompilacji, a także ocenę przenoszalności kodu na systemy Windows i Linux oraz subiektywną ocenę wygody programowania i jakości dokumentacji. Na podstawie zebranych danych sformułowano rekomendacje wskazujące, która biblioteka najlepiej odpowiada określonym typom projektów i wymaganiom zespołu.

1.4. Organizacja pracy

Rozdział 1. stanowi wprowadzenie, w którym przedstawiono cel, przyjęte rozwiązania, rezultat oraz strukturę pracy.

Rozdział 2. omawia istniejące rozwiązania w obszarze tworzenia GUI w C++. Przedstawiono w nim dwa główne paradygmaty, tryb zachowany i tryb natychmiastowy, oraz architektury analizowanych bibliotek.

Rozdział 3. opisuje kryteria porównawcze i metodologię badań. Uzasadniono w nim wybór aplikacji testowej, zdefiniowano jej wymagania funkcjonalne oraz określono mierzalne i jakościowe kryteria oceny bibliotek.

Rozdział 4. charakteryzuje technologie i narzędzia zastosowane w prototypie, w tym standard C++20, system budowania CMake, menedżer pakietów Conan oraz biblioteki pomocnicze.

Rozdział 5. przedstawia projekt i implementację prototypu. Opisano architekturę rozwiązania z wymiennymi backendami, a następnie omówiono szczegóły implementacji dla każdej z czterech bibliotek.

Rozdział 6. zawiera wyniki testów i pomiarów. Zestawiono dane dotyczące wydajności renderowania, zużycia zasobów, rozmiaru plików binarnych, czasu kompilacji i przenoszalności, a następnie sformułowano rekomendacje doboru biblioteki w zależności od potrzeb projektu.

Rozdział 7. podsumowuje pracę, prezentuje wnioski płynące z przeprowadzonego porównania oraz wskazuje kierunki możliwego dalszego rozwoju.

2. Istniejące rozwiązania

Niniejszy rozdział omawia kontekst techniczny pracy, prezentując fundamentalne podejścia do budowy interfejsów graficznych w języku C++, architektury porównywanych bibliotek oraz istniejące próby zestawienia tych narzędzi.

2.1. Sposoby tworzenia GUI w C++

W ciągu kilku dekad rozwoju graficznych interfejsów użytkownika ukształtowały się dwa różne podejścia do ich projektowania i implementacji. Wybór paradygmatu determinuje architekturę aplikacji, sposób zarządzania stanem oraz wymagania wydajnościowe. Poniższe podrozdziały omawiają te podejścia, a także odrębną kwestię sposobu renderowania kontrolki.

2.1.1 Tryb zachowany (*Retained Mode*)

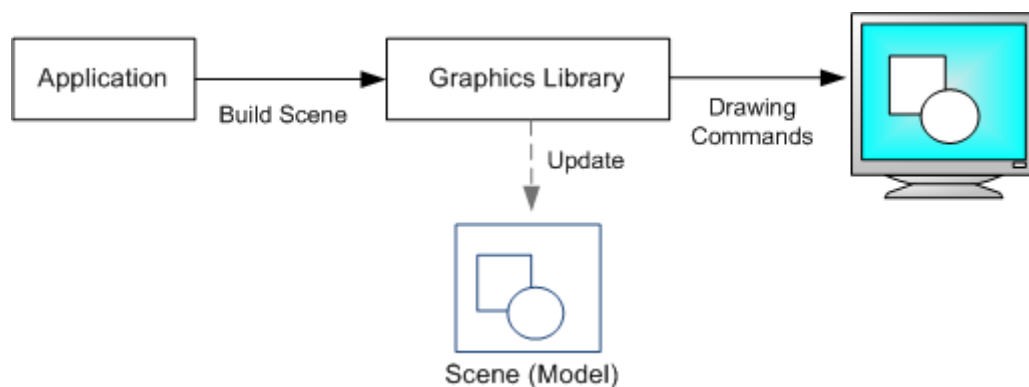
Tryb zachowany to klasyczne podejście do budowy GUI, wywodzące się z wczesnych bibliotek graficznych systemów okienkowych. W tym modelu biblioteka utrzymuje w pamięci trwałą strukturę obiektów, odzwierciedlającą bieżący stan interfejsu. Poszczególne elementy są tworzone jednokrotnie podczas inicjalizacji aplikacji i pozostają w pamięci przez cały czas jej działania, niezależnie od liczby wyrenderowanych klatek [1].

Komunikacja między komponentami opiera się na mechanizmach powiadomień: zdarzenia generowane przez użytkownika (kliknięcie, zmiana wartości pola) są przekazywane do zarejestrowanych procedur obsługi. Biblioteka odpowiada za śledzenie zmian stanu i wywoływanie przerysowania wyłącznie tych obszarów ekranu, które uległy modyfikacji. Przykładem realizacji tego mechanizmu w Qt jest system sygnałów i slotów, opisany szczegółowo w dokumentacji frameworka [2].

Główne cechy trybu zachowanego to:

- trwałość obiektów widżetów w pamięci między kolejnymi klatkami;
- automatyczne zarządzanie przerysowaniem zmodyfikowanych obszarów;
- deklaratywne lub obiektowe tworzenie struktury interfejsu;
- obsługa zdarzeń oparta na mechanizmach sygnałów, slotów lub funkcji zwrotnych.

Podejście to jest naturalne dla typowych aplikacji desktopowych z menu, oknami dialogowymi i formularzami. Jego wadą jest jednak złożoność zarządzania stanem w przypadku dynamicznie zmieniających się interfejsów, a także narzut pamięciowy wynikający z utrzymywania pełnej sceny [1]. Rysunek 1 przedstawia trwałą strukturę obiektów utrzymywaną w pamięci w trybie zachowanym.



Rysunek 1. Wizualizacja trybu zachowanego. Źródło: [1]

2.1.2 Tryb natychmiastowy (*Immediate Mode*)

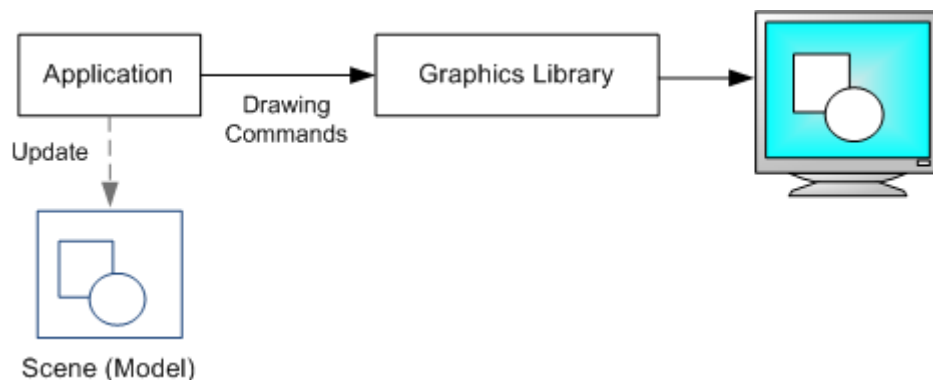
Tryb natychmiastowy to odmienne podejście do budowy GUI, w którym interfejs nie jest reprezentowany przez trwałą strukturę obiektów, lecz opisywany proceduralnie przy każdym rysowaniu klatki. W modelu tym aplikacja wywołuje kolejno funkcje rysujące poszczególne elementy (przyciski, pola tekstowe, suwaki) w każdej iteracji pętli głównej, a biblioteka natychmiast je renderuje i zwraca informację o interakcji użytkownika. Po zakończeniu klatki żaden widżet nie pozostaje w pamięci biblioteki [3].

Odpowiedzialność za przechowywanie stanu interfejsu spoczywa wyłącznie na kodzie aplikacji. Jeżeli przycisk ma być widoczny tylko w określonym kontekście, wystarczy nie wywoływać odpowiadającej mu funkcji. Biblioteka nie wymaga jawnego tworzenia ani niszczenia obiektów. Podejście to zostało spopularyzowane przez Caseya Muratoriego w 2005 roku [3].

Główne cechy trybu natychmiastowego to:

- brak trwałej struktury widżetów;
- stan interfejsu opisywany jest każdorazowo w kodzie aplikacji;
- synchroniczne zwracanie wyniku interakcji bezpośrednio z funkcji rysującej;
- minimalne wewnętrzne zarządzanie stanem po stronie biblioteki;
- pełna kontrola nad logiką wyświetlania po stronie programisty.

Tryb natychmiastowy szczególnie dobrze sprawdza się w aplikacjach z dynamicznie zmieniającym się interfejsem, narzędziach deweloperskich oraz grach komputerowych, gdzie pętla renderująca i tak wykonywana jest w każdej klatce. Rysunek 2 ilustruje proceduralne odtwarzanie interfejsu w każdej klatce, właściwe dla trybu natychmiastowego.



Rysunek 2. Wizualizacja trybu natychmiastowego. Źródło: [1]

2.1.3 Natywne kontrolki a renderowanie własne

Biblioteki GUI różnią się w sposobie prezentacji elementów interfejsu. Część z nich deleguje rysowanie widżetów do systemu operacyjnego, korzystając z jego natywnych kontroltek, podczas gdy inne samodzielnie renderują wszystkie elementy przy użyciu interfejsów graficznych.

Podejście oparte na natywnych kontrolkach polega na bezpośrednim osadzaniu widżetów dostarczanych przez platformę, takich jak przyciski, pola tekstowe czy listy rozwijane zdefiniowane na przykład przez Win32 API w systemie Windows. Biblioteka pełni wówczas rolę warstwy abstrakcji, tłumacząc wywołania niezależne od platformy na odpowiednie operacje systemowe. Reprezentantem tego podejścia jest wxWidgets, który używa natywnych widżetów zawsze, kiedy jest to możliwe [4].

Podejście oparte na renderowaniu własnym zakłada, że biblioteka samodzielnie rysuje wszystkie elementy interfejsu przy użyciu niskopoziomowych interfejsów graficznych. Qt realizuje ten model

poprzez abstrakcyjną klasę `QStyle`, odpowiedzialną za rysowanie widżetów niezależnie od platformy. Jak wskazuje dokumentacja frameworka, wbudowane widżety Qt korzystają z `QStyle` do niemal całego rysowania, co pozwala precyzyjnie odwzorować wygląd natywnych kontroltek [5]. Dear ImGui stosuje analogiczne podejście do samodzielnego renderowania: każdy element interfejsu rysowany jest bezpośrednio w dostarczonym buforze graficznym przy każdym wywołaniu klatki [3].

2.2. Analiza architektur porównywanych bibliotek

Wybór biblioteki GUI wiąże się nie tylko z dostępnymi widżetami czy obsługiwanymi platformami, lecz przede wszystkim z architekturą, która determinuje sposób organizacji kodu aplikacji, mechanizmy komunikacji między komponentami oraz możliwości rozbudowy projektu. Biblioteki różniące się paradygmatem, jak opisano w podrozdziale 2.1, narzucają odmienne wzorce projektowe, a ich zrozumienie jest warunkiem koniecznym do przeprowadzenia rzetelnego porównania.

Niniejszy podrozdział analizuje cztery biblioteki będące przedmiotem pracy: Qt, `wxWidgets`, Dear ImGui oraz GTK. Dla każdej z nich omówiono model obiektowy, mechanizm obsługi zdarzeń i komunikacji między komponentami, podejście do renderowania kontroltek oraz strukturę modułową i warunki licencyjne.

2.2.1 Qt

Qt to kompleksowy framework aplikacyjny dla języka C++, którego zakres wykracza daleko poza samą bibliotekę GUI [44]. Oprócz widżetów framework dostarcza moduły odpowiedzialne za obsługę sieci, multimediów, baz danych, drukowania, lokalizacji, komunikacji między procesami itp. Tak szeroka funkcjonalność sprawia, że Qt traktowany jest nie jako biblioteka, lecz jako pełna platforma do tworzenia aplikacji desktopowych, mobilnych i wbudowanych.

Fundamentem architektury Qt jest system metaobiektów (*Meta-Object System*), stanowiący rozszerzenie modelu obiektowego języka C++. Jego działanie opiera się na trzech elementach: klasie bazowej `QObject`, makrze `Q_OBJECT` oraz narzędziu MOC (*Meta-Object Compiler*), które przed właściwą kompilacją analizuje klasy oznaczone tym makrem i generuje na ich podstawie dodatkowy kod C++ [6]. System ten umożliwia mechanizm sygnałów i slotów, dostęp do informacji o typach w czasie działania programu (*runtime type information*), dynamiczny system właściwości obiektów oraz bezpieczne rzutowanie typów za pomocą `qobject_cast`. Zaletą tego podejścia jest udostępnianie mechanizmów wykraczających poza możliwości czystego języka C++ bez wprowadzania niestandardowych rozszerzeń składni. Dzięki temu kod działa tak samo na dowolnym kompilatorze [49]. Wadą jest dodatkowy etap przetwarzania kodu, który musi obsługiwać system budowania, co wydłuża czas kompilacji. Dodatkowym ograniczeniem jest brak możliwości stosowania makra `Q_OBJECT` w szablonach klas [50], co zawęża sposób korzystania z tego mechanizmu.

Mechanizm sygnałów i slotów, będący podstawowym sposobem komunikacji między komponentami Qt, opisano w podrozdziale 5.2.2.

W zakresie renderowania Qt stosuje własny silnik oparty na klasie `QStyle`, co opisano w podrozdziale 2.1.3 [5]. Widżety rysowane są niezależnie od systemu operacyjnego, przy czym domyślne style platformowe pozwalają odwzorować wygląd zbliżony do natywnych kontroltek.

Framework oferuje dwa podejścia do budowy interfejsu użytkownika, które mogą być stosowane łącznie w ramach jednej aplikacji:

- zestaw klasycznych widżetów desktopowych: okien, przycisków, list, formularzy, opartych na trybie zachowanym i hierarchii klas dziedziczących z `QWidget` [7]. Moduł ten wspiera ponadto architekturę Model/Widok (*Model/View*), umożliwiającą oddzielenie danych od ich reprezentacji graficznej;

- Qt Quick, opierający się na deklaratywnym języku QML (*Qt Modeling Language*), którego głównym przeznaczeniem są aplikacje wymagające płynnych animacji, interfejsów dotykowych lub dynamicznie zmieniającej się prezentacji danych [8].

Przenośność między platformami zapewnia warstwa abstrakcji QPA (*Qt Platform Abstraction*), która izoluje kod frameworka od szczegółów systemu operacyjnego. Dzięki temu ta sama baza kodu kompilowana jest dla systemów Windows, Linux, macOS, Android oraz iOS bez konieczności modyfikacji logiki aplikacji.

Framework dostępny jest na licencjach LGPL v3 i GPL v3, a także na licencji komercyjnej oferowanej przez Qt Group.

2.2.2 wxWidgets

wxWidgets to framework C++ przeznaczony do tworzenia aplikacji desktopowych i mobilnych z graficznym interfejsem użytkownika [4]. Jak wskazują autorzy biblioteki, zakres jej funkcjonalności wykracza poza samą warstwę GUI. Dostarcza również klasy do obsługi plików i strumieni, wielowątkowości, ustawień aplikacji, komunikacji między procesami, dostępu do baz danych oraz pomocy kontekstowej [4]. Projekt powstał w 1992 roku i przez ponad trzy dekady rozwijany jest jako oprogramowanie o otwartym kodzie źródłowym.

Kluczową cechą wyróżniającą wxWidgets spośród innych frameworków jest stosowanie natywnych kontrolki systemu operacyjnego. Biblioteka pełni rolę warstwy abstrakcji, tłumacząc niezależne od platformy wywołania API na odpowiednie operacje systemowe.

Model obiektowy wxWidgets oparty jest na hierarchii klas dziedziczących z `wxObject`. Klasy reprezentujące elementy interfejsu wywodzą się z `wxEvtHandler`, który odpowiada za przyjmowanie i przekazywanie zdarzeń systemowych [9]. Biblioteka nie wymaga dodatkowych narzędzi generujących kod na etapie budowania, w odróżnieniu od Qt, gdzie system metaobiektów wymaga przetworzenia kodu przez kompilator MOC przed właściwą kompilacją [6].

Obsługa zdarzeń realizowana jest dwoma sposobami: za pomocą statycznych tablic zdarzeń (*event tables*) definiowanych makrami lub nowszego mechanizmu `Bind()`, umożliwiającego dynamiczne podłączenie dowolnego wywołalnego obiektu jako procedury obsługi. Rozmieszczanie elementów interfejsu realizowane jest przez mechanizm `sizers`, obiektów odpowiedzialnych za automatyczne układanie i skalowanie widżetów, eliminując konieczność definiowania bezwzględnych pozycji elementów.

wxWidgets dystrybuowany jest na licencji wxWindows Library Licence, opartej na LGPL z klauzulą wyjątku zezwalającą na dystrybucję aplikacji komercyjnych bez ujawniania ich kodu źródłowego [10].

2.2.3 Dear ImGui

Dear ImGui to biblioteka GUI dla języka C++ stworzona przez Omara Cornuta i rozwijana jako oprogramowanie z otwartym kodem źródłowym [11]. Autor opisuje ją jako lekką i minimalistyczną bibliotekę GUI. Biblioteka realizuje paradygmat trybu natychmiastowego (*Immediate Mode*), opisany szczegółowo w podrozdziale 2.1.2, nie przechowuje trwałej struktury widżetów, lecz odtwarza cały interfejs przy każdym wywołaniu klatki.

Centralnym elementem architektury Dear ImGui są listy poleceń rysowania (*draw lists*) zawierające zoptymalizowane bufory wierzchołków (*vertex buffers*), które aplikacja przekazuje do własnego silnika graficznego. Biblioteka nie zarządza stanem graficznym ani nie komunikuje się bezpośrednio z żadnym API graficznym, generuje jedynie dane geometryczne, które aplikacja samodzielnie przekazuje do wybranego podsystemu renderowania [11]. Biblioteka dostarcza gotowe adaptory (*backends*) dla OpenGL, DirectX 9/10/11/12, Vulkan, Metal oraz innych interfejsów graficznych.

Istotną cechą projektową jest minimalizacja stanu po stronie aplikacji. Oznacza to, że logika wyświetlania i logika aplikacji mogą być pisane sekwencyjnie, bez konieczności utrzymywania oddzielnych struktur opisujących stan interfejsu.

Pod względem dystrybucji Dear ImGui wyróżnia się wyjątkową prostotą. Biblioteka składa się z kilku plików platformowo niezależnych, które wystarczy dołączyć bezpośrednio do projektu. Nie wymaga żadnego systemu budowania, narzędzi generujących kod ani zewnętrznych zależności.

Biblioteka przeznaczona jest przede wszystkim dla narzędzi deweloperskich i interfejsów diagnostycznych, a nie dla interfejsów użytkownika końcowego. Autor wskazuje docelowe zastosowania: narzędzia do wizualizacji danych i debugowanie, narzędzia do silników gier, aplikacje czasu rzeczywistego i aplikacje działające w trybie pełnoekranowym [11].

Dear ImGui dystrybuowana jest na licencji MIT.

2.2.4 GTK

GTK to wieloplatformowa biblioteka widżetów z otwartym kodem źródłowym. Biblioteka powstała pierwotnie jako narzędzie do tworzenia interfejsu programu GIMP (*GNU Image Manipulation Program*) [46], a następnie rozwinęła się w samodzielną bibliotekę graficzną ogólnego przeznaczenia. GTK napisane jest w języku C, jednak od początku projektowane było z myślą o wiązaniach (*bindings*) dla innych języków programowania. Biblioteka dostępna jest dla języków Python, JavaScript, C++, Rust i innych [45].

Fundamentem architektury GTK jest system GObject, wchodzący w skład biblioteki GLib. GObject dostarcza mechanizmy programowania obiektowego w języku C: system typów z dziedziczeniem, metody wirtualne, właściwości obiektów oraz system sygnałów. Sygnały w GTK pełnią funkcję analogiczną do sygnałów i slotów w Qt, lecz realizowane są wyłącznie przy użyciu standardowych konstrukcji języka C [12].

W zakresie renderowania GTK 4 korzysta z warstwy GSK, opisywanej w dokumentacji jako abstrakcja renderowania sceny graficznej [13]. GSK udostępnia między innymi implementacje renderowania: Vulkan, OpenGL oraz Cairo. Styl wizualny widżetów definiowany jest za pomocą arkuszy CSS, obsługiwanych przez klasę *CssProvider*, co umożliwia szeroką personalizację wyglądu aplikacji [14].

Choć GTK dostępne jest na systemach Windows i macOS, jego podstawowym środowiskiem jest Linux, gdzie stanowi fundament środowiska graficznego GNOME.

GTK dystrybuowane jest na licencji LGPL w wersji 2.1 lub nowszej.

2.3. Istniejące porównania bibliotek GUI

W obszarze porównań bibliotek GUI dla języka C++ dostępne są nieliczne niezależne opracowania uwzględniające łącznie wydajność, łatwość użycia oraz przenoszalność między platformami. Istniejące materiały przyjmują zazwyczaj formę stron wiki, artykułów technicznych lub dyskusji społecznościowych. Poniższe podrozdziały omawiają najczęściej przywoływane przykłady z każdej z tych kategorii.

2.3.1 Wiki wxWidgets

Projekt wxWidgets utrzymuje stronę wiki zatytułowaną *WxWidgets Compared To Other Toolkits*, zestawiającą bibliotekę wxWidgets z innymi narzędziami, w tym Qt, GTK, Tk oraz Lazarus [15]. Strona skupia się głównie na różnicach w modelu licencyjnym, sposobie obsługi widżetów natywnych oraz zakresie dostępnych klas pomocniczych. Brakuje w niej przykładów kodu, pomiarów wydajności oraz analizy kosztów integracji w rzeczywistym projekcie. Opisy mają charakter jakościowy i nie precyzują,

na jakich wersjach bibliotek ani jakich platformach przeprowadzono ewentualne testy. Z uwagi na to, że stronę redaguje społeczność projektu wxWidgets, zawarte w niej zestawienie nie jest w pełni niezależne od biblioteki, którą opisuje. Strona jest również rzadko aktualizowana, ostatnie edycje miały miejsce w 2018 i 2012 roku.

2.3.2 Artykuły online

Wśród artykułów opublikowanych w Internecie często cytowanym przykładem jest tekst na stronie e-con Systems, porównujący Qt, GTK oraz wxWidgets [16]. Artykuł omawia wsparcie dla platform docelowych, model licencyjny i historię każdego z projektów. Wnioski mają charakter opisowy i nie są poparte przykładami kodu ani pomiarami ilościowymi.

Artykuł serwisu Baeldung on Linux ogranicza zakres porównania wyłącznie do GTK oraz Qt, skupiając się na różnicach w modelu programowania w środowisku Linux [17]. Biblioteki wxWidgets i Dear ImGui nie są w nim uwzględnione.

Zestawienie autorstwa Philippe’a Groarke’a obejmuje dużą liczbę bibliotek GUI dla języka C++, oferując przy każdej z nich krótki opis wsparcia platformowego, typu licencji, przykładowego zrzutu ekranu i poziomu skomplikowania zależności [18]. Ze względu na szeroki zakres opracowania, każda biblioteka jest opisana zwięźle, bez przykładów implementacji ani analizy konkretnych scenariuszy użycia.

Wspólną cechą wymienionych artykułów jest brak praktycznych przykładów aplikacji oraz brak pomiarów wydajności. Żaden z nich nie analizuje konsekwencji wyboru trybu natychmiastowego w porównaniu z trybem zachowanym w kontekście aplikacji desktopowej.

2.3.3 Fora internetowe

Liczne dyskusje na platformach Reddit oraz Quora poruszają kwestię wyboru biblioteki GUI dla języka C++. Wypowiedzi mają zazwyczaj charakter anegdotyczny i wynikają z indywidualnych doświadczeń autorów, bez ustrukturyzowanej metodologii porównawczej ani weryfikacji twierdzeń w oparciu o mierzalne kryteria. Platformy te nie są nastawione na prezentację kodu, wobec czego dyskusje ograniczają się do ogólnych rekomendacji bez szczegółowego uzasadnienia technicznego.

Przegląd istniejących porównań wskazuje na lukę w literaturze: brakuje opracowania, które zestawiałoby wybrane biblioteki na podstawie implementacji tej samej aplikacji, z uwzględnieniem pomiarów wydajności oraz różnic wynikających z odmiennych paradygmatów tworzenia GUI.

3. Kryteria porównawcze i metodologia

Podstawą porównania bibliotek GUI opisanego w niniejszej pracy jest stworzenie czterech implementacji tej samej aplikacji, po jednej dla każdej z analizowanych bibliotek. Takie podejście pozwala na zestawienie bibliotek w identycznych warunkach funkcjonalnych, eliminując wpływ różnic w specyfice problemu na wyniki analizy. Niniejszy rozdział opisuje uzasadnienie wyboru aplikacji testowej, kryteria przyjęte do oceny bibliotek oraz metodologię przeprowadzenia pomiarów i analizy.

3.1. Wybór aplikacji testowej

Jako aplikację testową wybrano przeglądarkę zdjęć. Jest to aplikacja powszechnie zrozumiała, co pozwala na jednoznaczną ocenę poprawności działania bez konieczności znajomości specyficznej dziedziny problemu. Spośród typowych aplikacji desktopowych przeglądarka zdjęć wyróżnia się szczególnie szerokim zakresem komponentów GUI, jakich wymaga jej implementacja. Obejmuje zarówno elementy podstawowe, takie jak przyciski i etykiety, jak i bardziej złożone, takie jak eksplorator plików, panel metadanych, zakładki ustawień oraz obszar renderowania obrazu. Dzięki temu pojedyncza implementacja pozwala ocenić bibliotekę w wielu wymiarach jednocześnie.

Istotnym aspektem jest obecność obszaru renderowania obrazu, który wymaga od biblioteki obsługi wyświetlania, skalowania i transformacji treści graficznej. Jest to szczególnie wartościowe kryterium porównawcze, ponieważ każda z analizowanych bibliotek realizuje rysowanie w odmienny sposób. Różnice te wynikają między innymi z przyjętego paradygmatu interfejsu: Qt, wxWidgets i GTK korzystają z trybu zachowanego (*Retained Mode*), natomiast Dear ImGui opiera się wyłącznie na trybie natychmiastowym (*Immediate Mode*).

Przeglądarka zdjęć wymaga również integracji z systemem operacyjnym: obsługi systemu plików, natywnych okien dialogowych oraz drukowania. Są to obszary, w których biblioteki GUI w sposób szczególnie wyraźny różnią się podejściem do abstrakcji platformowej. Obsługa tych funkcji w aplikacji testowej pozwala ocenić, jak każda z bibliotek radzi sobie z wykraczaniem poza samo rysowanie interfejsu.

Dodatkowym argumentem przemawiającym za wyborem przeglądarki zdjęć jest właściwa proporcja między złożonością a powtarzalnością implementacji. Aplikacja jest wystarczająco rozbudowana, aby ujawnić rzeczywiste różnice między bibliotekami, a jednocześnie na tyle ograniczona zakresem, aby cztery niezależne implementacje pozostały porównywalne i możliwe do wykonania.

3.2. Wymagania funkcjonalne aplikacji testowej

Wymagania funkcjonalne aplikacji testowej określają zestaw zachowań przyjęty jako podstawa porównania bibliotek. Każda z czterech implementacji realizuje te wymagania w zakresie, na jaki pozwalają możliwości danej biblioteki. Przypadki, w których biblioteka nie oferuje wsparcia dla określonej funkcji lub wymaga uzupełnienia o zewnętrzne rozwiązania, są odnotowane jako ograniczenie danej biblioteki i omówione w odpowiednich podrozdziałach. Poniższe podrozdziały opisują poszczególne wymagania wraz z uzasadnieniem ich wyboru w kontekście celów porównawczych niniejszej pracy.

3.2.1 Przeglądanie i wyświetlanie obrazów

Wymaganie obejmuje wczytanie pliku obrazu z dysku oraz jego wyświetlenie w obszarze roboczym aplikacji. Obsługiwane formaty to JPEG, PNG oraz BMP. Są to formaty powszechnie

stosowane w codziennym użytkowaniu, obejmujące zarówno kompresję stratną (JPEG), bezstratną (PNG), jak i zapis nieskompresowany (BMP).

Cecha ta jest ważna w przypadku przeglądarki zdjęć. Bez sprawnego wczytywania i wyświetlania obrazów żadna z pozostałych funkcji aplikacji nie ma zastosowania. Stanowi jednocześnie kryterium porównawcze, ponieważ analizowane biblioteki realizują ten mechanizm w odmienny sposób.

3.2.2 Skalowanie i obracanie

Obsługa zmiany rozmiaru oraz obracania wyświetlanego obrazu to jeden z ważniejszych obszarów oceny bibliotek GUI w kontekście pracy z przeglądarką zdjęć. Operacje te obejmują mechanizmy transformacji graficznych, zarządzanie pamięcią przy przetwarzaniu danych pikselowych oraz jakość dostępnych algorytmów interpolacji.

Aplikacja testowa obsługuje dwa tryby skalowania:

- tryb dopasowania (*Fit*) polegający na automatycznym przeskalowaniu obrazu do rozmiarów dostępnego obszaru wyświetlania przy zachowaniu oryginalnych proporcji;
- tryb powiększenia (*Zoom*) umożliwiający ręczną zmianę skali wyrażonej procentowo względem rzeczywistych wymiarów pliku.

Zmiana poziomu powiększenia realizowana jest za pomocą suwaków w interfejsie użytkownika. Przy powiększeniach przekraczających rozmiar okna wymagana jest możliwość przewijania w obrębie wyświetlanego obrazu.

Obracanie obrazu odbywa się skokowo, w krokach co 90 stopni, zarówno zgodnie z ruchem wskazówek zegara, jak i w kierunku przeciwnym. Operacja nie powoduje modyfikacji pliku źródłowego na dysku. Wymaganie to pozwala zbadać, w jaki sposób biblioteki podchodzą do transformacji obrazu.

3.2.3 Wyświetlanie metadanych

Wymaganie obejmuje odczyt i prezentację podstawowych informacji o wczytanym pliku obrazu w postaci okna dialogowego wywoływanego na żądanie użytkownika. Wyświetlane dane obejmują: nazwę pliku, rozmiar na dysku, wymiary w pikselach oraz dodatkowe informacje, które udostępnia użyta biblioteka.

Biblioteka Qt umożliwia rozszerzenie tego widoku o histogram RGB, przedstawiający rozkład wartości składowych czerwonej, zielonej i niebieskiej dla wszystkich pikseli obrazu. Histogram wyznaczany jest przez jednokrotne przejście po danych pikselowych i zliczanie wartości każdego kanału do osobnych tablic. Wizualizacja realizowana jest przy użyciu modułu `Qt Charts`, będącego częścią standardowej dystrybucji Qt. Pozostałe analizowane biblioteki nie oferują porównywalnego modułu wykresów, wobec czego analogiczna funkcjonalność wymagałaby własnej implementacji renderowania lub integracji zewnętrznej biblioteki. Z perspektywy porównawczej wymaganie to pozwala ocenić zarówno stopień dostępu do niskopoziomowych danych pikselowych, jak i bogactwo wbudowanych komponentów interfejsu użytkownika.

3.2.4 Eksplorator plików

Wymaganie obejmuje dwa mechanizmy nawigacji po plikach: otwieranie pliku obrazu za pomocą okna dialogowego wyboru pliku oraz przejście do katalogu bieżącego pliku w menedżerze plików systemu operacyjnego.

Otwieranie pliku realizowane jest przez okno dialogowe (*file dialog*). Qt, wxWidgets i GTK udostępniają własne komponenty wywołujące natywne okno systemu operacyjnego. Dear ImGui nie posiada takiego komponentu w swojej standardowej dystrybucji, co wynika z charakteru tej biblioteki skupionej na interfejsie narzędziowym i diagnostycznym. W prototypie konieczne było dołączenie

zewnętrznej biblioteki *Portable File Dialogs*, która na każdej platformie wywołuje natywne okno systemowe.

Przejsięcie do folderu bieżącego pliku polega na otwarciu katalogu zawierającego wczytany obraz w domyślnym menedżerze plików systemu operacyjnego. Funkcjonalność ta wymaga integracji z usługami platformy, ponieważ sposób uruchamiania menedżera plików różni się zależnie od systemu operacyjnego. Biblioteki różnią się pod względem poziomu abstrakcji oferowanej programiście. Część z nich dostarcza gotowe, przenośne mechanizmy wywołania domyślnej aplikacji systemowej, inne natomiast nie oferują takiej funkcjonalności, co wymusza napisanie kodu zależnego od platformy.

Wymaganie to pozwala ocenić, w jakim stopniu każda z bibliotek zapewnia gotowe mechanizmy integracji z usługami systemu operacyjnego, a w jakim zakresie programista zmuszony jest sięgać po zewnętrzne zależności lub pisać kod specyficzny dla danej platformy.

3.2.5 Modyfikacja pliku

Wymaganie obejmuje dwie operacje trwale zmieniające stan pliku na dysku: zapis obrazu do pliku oraz jego usunięcie.

Zapis realizowany jest w formie funkcji „Zapisz jako” (*Save As*), która umożliwia wyeksportowanie bieżącego obrazu do wybranego przez użytkownika pliku w jednym z obsługiwanych formatów. Istotnym aspektem tej operacji jest konieczność uwzględnienia przekształceń naniesionych przez użytkownika w trakcie pracy z aplikacją: obrotu oraz zmian jasności i kontrastu. Oznacza to, że przed zapisem biblioteka lub kod aplikacji musi zastosować te przekształcenia bezpośrednio na danych pikselowych, a nie jedynie na warstwie prezentacji. Jest to wyraźne kryterium różnicujące biblioteki, ponieważ niektóre z nich oferują mechanizmy wysokiego poziomu obsługujące takie przekształcenia, inne natomiast wymagają ręcznego przetworzenia bufora pikselowego przez programistę.

Usunięcie pliku polega na trwałym skasowaniu bieżącego pliku z dysku po uzyskaniu potwierdzenia od użytkownika. Operacja wymaga wyświetlenia okna dialogowego z pytaniem o potwierdzenie, co angażuje mechanizmy tworzenia okien modalnych właściwe dla danej biblioteki. Po pomyślnym usunięciu interfejs użytkownika powinien zostać odpowiednio zaktualizowany. Kryterium to pozwala ocenić, jak biblioteki wspierają operacje na systemie plików oraz w jaki sposób obsługują interakcję z użytkownikiem przy operacjach nieodwracalnych.

3.2.6 Opcja drukowania

Funkcjonalność drukowania umożliwia wydruk bieżącego obrazu za pośrednictwem systemowego okna dialogowego (*print dialog*).

Obsługa drukowania wymaga integracji z podsystemem drukowania systemu operacyjnego, co stawia przed bibliotekami GUI wymagania wykraczające poza samo rysowanie interfejsu. Z perspektywy porównawczej ważne jest, czy dana biblioteka dostarcza wbudowane komponenty obsługujące komunikację z podsystemem drukowania, czy też wymaga to sięgnięcia po dodatkowe moduły lub zewnętrzne zależności.

3.2.7 Zakładka ustawień

Aplikacja testowa udostępnia panel regulacji parametrów obrazu, obejmujący zmianę jasności (*brightness*) oraz kontrastu (*contrast*). Oba parametry przyjmują wartości z zakresu od -100 do +100 i sterowane są suwakami w oknie dialogowym. Modyfikacje widoczne są na bieżąco w obszarze roboczym bez konieczności zatwierdzania: każda zmiana położenia suwaka skutkuje natychmiastowym przeliczeniem i odświeżeniem wyświetlanego obrazu. Użytkownik może zatwierdzić zmiany lub je anulować, przywracając wartości sprzed otwarcia okna.

Kryterium to pozwala ocenić wydajność dostępu do danych pikselowych w poszczególnych bibliotekach oraz sposób wyzwalania odświeżenia widoku.

3.2.8 Lista ostatnio otwieranych plików

Funkcjonalność przechowuje historię plików otwieranych przez użytkownika i udostępnia ją w postaci podmenu paska menu aplikacji. Lista ograniczona jest do dziesięciu najnowszych pozycji, posortowanych według czasu ostatniego dostępu. Wybranie pozycji z listy powoduje wczytanie odpowiedniego pliku bez konieczności korzystania z okna dialogowego. Dane przechowywane są w lokalnej bazie danych, co zapewnia ich trwałość między uruchomieniami aplikacji.

Kryterium to ilustruje różnicę w zakresie funkcjonalności oferowanych przez biblioteki poza samym interfejsem graficznym. Dostępność wbudowanego modułu baz danych wpływa na objętość kodu oraz poziom abstrakcji, na którym programista operuje przy implementacji funkcji wymagających trwałego przechowywania danych.

3.3. Kryteria porównawcze

Poza wymaganiami funkcjonalnymi, porównanie bibliotek obejmuje szereg kryteriów dotyczących procesu tworzenia oprogramowania. Kryteria te obejmują zarówno mierzalne właściwości, takie jak wydajność czy zużycie zasobów, jak i jakościowe aspekty pracy z biblioteką, mające wpływ na jej wybór w kontekście rzeczywistego projektu. Poniższe podrozdziały omawiają te kryteria oraz uzasadniają ich wybór jako podstawy oceny.

3.3.1 Łatwość implementacji i wygoda programowania

Kryterium ocenia, jak wiele wysiłku wymaga realizacja poszczególnych funkcjonalności przy użyciu danej biblioteki oraz jak przystępne jest jej API dla programisty. Ocena obejmuje kilka wzajemnie powiązanych aspektów.

Pierwszym z nich jest projekt API (*Application Programming Interface*) biblioteki: jego spójność, czytelność nazewnictwa oraz poziom abstrakcji oferowany programiście. Dobrze zaprojektowane API umożliwia wyrażenie intencji w sposób zwięzły i bezpośredni, bez konieczności znajomości szczegółów wewnętrznej implementacji.

Drugim aspektem jest ilość kodu pomocniczego (*boilerplate*) niezbędnego do uruchomienia podstawowych funkcji. Niektóre biblioteki wymagają jawnej inicjalizacji wielu podsystemów, ręcznego zarządzania zasobami i obsługi zdarzeń na niskim poziomie, podczas gdy inne dostarczają gotowych mechanizmów eliminujących powtarzalny kod.

Trzecim aspektem jest dostępność i jakość dokumentacji oraz narzędzi wspierających programistę: środowisk programistycznych (*IDE*), generatorów kodu, projektantów interfejsów graficznych (*GUI designer*) oraz systemów budowania. Dostępność takich narzędzi bezpośrednio przekłada się na tempo i komfort pracy.

Ocena tego kryterium opiera się na obserwacjach zebranych podczas implementacji prototypu i ma charakter subiektywny. Stanowi jednak istotne uzupełnienie pomiarów, ponieważ wygoda programowania należy do najczęściej wymienianych argumentów za wyborem biblioteki GUI.

3.3.2 Przenoszalność między systemami operacyjnymi

Przenoszalność określa zdolność biblioteki do budowania i uruchamiania aplikacji na różnych systemach operacyjnych bez konieczności wprowadzania większych zmian w kodzie źródłowym. W niniejszej pracy kryterium oceniane jest na podstawie dwóch platform docelowych: systemu Windows oraz Linux.

Ocena przenoszalności obejmuje kilka wymiarów. Pierwszy dotyczy tego, czy biblioteka jest oficjalnie wspierana na obu platformach i czy jej instalacja oraz konfiguracja środowiska kompilacji przebiega w porównywalny sposób. Drugi wymiar odnosi się do ilości kodu specyficznego dla danej

platformy, który musi napisać programista przy implementacji funkcji wymagających integracji z usługami systemu operacyjnego, takich jak okna dialogowe, drukowanie czy uruchamianie widżetów zewnętrznych. Im więcej takich miejsc wymaga rozgałęzień warunkowych za pomocą dyrektyw preprocesora, tym niższy poziom przenoszalności oferuje biblioteka w praktyce.

Trzeci wymiar dotyczy spójności wizualnej i behawioralnej interfejsu między platformami. Biblioteki opierające się na natywnych kontrolkach systemowych automatycznie dostosowują wygląd aplikacji do stylu danego systemu operacyjnego, co jest oczekiwanym zachowaniem na każdej z platform, lecz może prowadzić do różnic w układzie i wyglądzie interfejsu. Biblioteki renderujące interfejs we własnym silniku graficznym zapewniają identyczny wygląd na wszystkich platformach, kosztem odstępstwa od natywnej estetyki systemu.

3.3.3 Wydajność i zużycie zasobów

Kryterium obejmuje pomiar i porównanie mierzalnych właściwości każdej z implementacji w obszarach mających bezpośredni wpływ na odczuwalną responsywność aplikacji oraz jej wymagania sprzętowe.

Pierwszym wskaźnikiem jest zużycie pamięci operacyjnej w stanie spoczynku oraz podczas wyświetlania obrazu. Różne biblioteki stosują odmienne strategie zarządzania buforami graficznymi i danymi pikselowymi, co przekłada się na różne całkowite zużycie pamięci przez proces aplikacji.

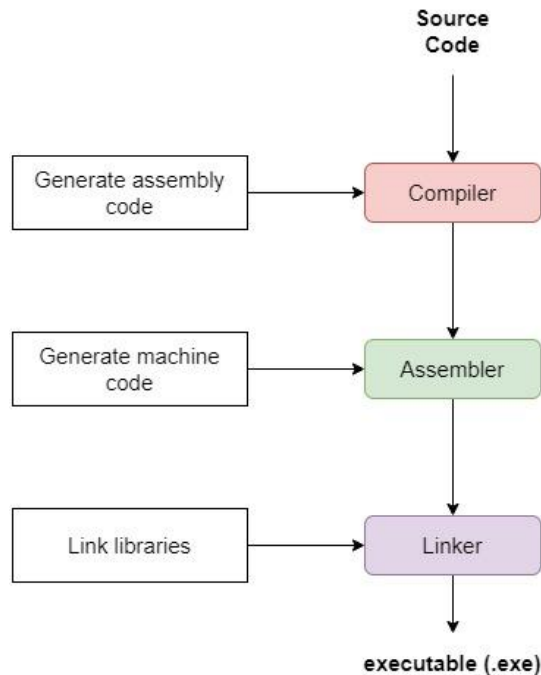
Drugim obszarem pomiaru jest wydajność operacji intensywnych obliczeniowo wykonywanych po stronie procesora: skalowania obrazu oraz przetwarzania danych pikselowych przy regulacji jasności i kontrastu. Operacje te angażują bibliotekę bezpośrednio, a różnice w dostępnych algorytmach i wewnętrznej reprezentacji danych mogą prowadzić do mierzalnych różnic w czasie przetwarzania.

Osobnym aspektem jest obciążenie procesora w stanie bezczynności aplikacji. Biblioteki działające w trybie natychmiastowym (*Immediate Mode*) przeliczają interfejs w każdej klatce niezależnie od tego, czy nastąpiła jakakolwiek zmiana stanu, co generuje stały koszt obliczeniowy. Biblioteki oparte na trybie zachowanym (*Retained Mode*) odświeżają widok wyłącznie w odpowiedzi na zdarzenia, co powinno skutkować minimalnym obciążeniem procesora podczas bezczynności.

3.3.4 Rozmiar pliku binarnego

Rozmiar pliku binarnego to kryterium istotne z perspektywy dystrybucji i wdrażania aplikacji. Na ostateczny rozmiar pakietu dystrybucyjnego wpływa zarówno rozmiar samego pliku wykonywalnego, jak i zestaw bibliotek wymaganych do uruchomienia aplikacji na docelowym systemie.

Kluczowym czynnikiem jest sposób linkowania (*linking*) bibliotek z kodem aplikacji. Linkowanie statyczne (*static linking*) włącza kod biblioteki bezpośrednio do pliku wykonywalnego, zwiększając jego rozmiar, lecz eliminując zależności zewnętrzne w czasie uruchomienia. Linkowanie dynamiczne (*dynamic linking*) skutkuje mniejszym plikiem wykonywalnym, jednak wymaga obecności odpowiednich bibliotek współdzielonych w środowisku uruchomieniowym [19]. W przypadku dystrybucji na systemy bez preinstalowanych bibliotek konieczne jest dołączenie ich do pakietu instalacyjnego. Schemat procesu budowania i linkowania aplikacji przedstawia Rysunek 3.



Rysunek 3. Proces budowania i linkowania aplikacji. Źródło: [19]

Poszczególne biblioteki różnią się pod tym względem. Duże frameworki dostarczają wiele modułów, z których aplikacja może korzystać tylko częściowo, jednak przy linkowaniu dynamicznym wszystkie wymagane biblioteki współdzielone muszą towarzyszyć aplikacji. Biblioteki o minimalnym zakresie funkcji generują mniejsze pliki wynikowe, lecz mogą wymagać uzupełnienia o zewnętrzne zależności realizujące funkcje, których same nie oferują.

Kryterium to jest szczególnie wymowne w przypadku prostej aplikacji testowej, która nie zawiera dodatkowych zasobów graficznych ani rozbudowanej logiki biznesowej. W takim przypadku rozmiar pakietu dystrybucyjnego odzwierciedla niemal wyłącznie narzut samej biblioteki, co umożliwia bezpośrednie porównanie jej wpływu na rozmiar końcowego produktu.

3.3.5 Zgodność ze standardem C++

Kryterium to obejmuje trzy powiązane aspekty: obsługiwane przez bibliotekę wersje standardu C++, obecność lub brak rozszerzeń wykraczających poza standard oraz wynikającą z tego niezależność od kompilatora.

Biblioteka korzystająca wyłącznie z konstrukcji przewidzianych przez standard języka działa poprawnie z każdym kompilatorem zgodnym z daną wersją standardu. Oznacza to możliwość budowania projektu zarówno przy użyciu GCC, Clang, jak i MSVC bez modyfikacji kodu źródłowego. Biblioteka wymagająca niestandardowych rozszerzeń składni lub dodatkowych narzędzi przetwarzania kodu ogranicza tę swobodę i może wymagać dostosowania procesu budowania do konkretnego środowiska.

Wymóg konkretnej wersji standardu, takiej jak C++11, C++17 czy C++20, wyznacza dolną granicę obsługiwanych kompilatorów i projektów. Wyższe wymagania wersji mogą ograniczać zastosowanie biblioteki w projektach z kompilatorem nieobsługującym nowszych wersji, jednocześnie umożliwiając bibliotece korzystanie z nowszych mechanizmów języka.

3.3.6 Jakość dokumentacji

Jakość dokumentacji wpływa bezpośrednio na czas potrzebny do opanowania biblioteki oraz na efektywność pracy podczas jej codziennego stosowania. Kryterium to obejmuje kilka wymiarów: kompletność i aktualność oficjalnej dokumentacji API, dostępność poradników wprowadzających i przykładów kodu, a także zasoby społeczności, takie jak fora dyskusyjne, strony pytań i odpowiedzi oraz materiały tworzone przez użytkowników.

Kompletna dokumentacja API powinna opisywać każdą klasę, metodę i sygnał dostępny w bibliotece, wraz z przykładami ilustrującymi typowe zastosowania. Brak dokumentacji dla części interfejsu zmusza programistę do analizy kodu źródłowego biblioteki lub polegania na nieoficjalnych źródłach, co zwiększa ryzyko błędnego użycia.

Równie ważna jest dostępność materiałów wprowadzających skierowanych do nowych użytkowników. Dokumentacja ograniczona wyłącznie do opisu API nie przedstawia zalecanego sposobu korzystania z biblioteki ani typowych wzorców projektowych stosowanych w danym ekosystemie.

Zasoby tworzone przez społeczność, takie jak odpowiedzi na forach czy artykuły techniczne, uzupełniają oficjalną dokumentację. Ich liczba i aktualność stanowią również pośrednią miarę aktywności i dojrzałości projektu.

4. Zastosowane technologie i narzędzia

Niniejszy rozdział opisuje technologie i narzędzia zastosowane podczas realizacji prototypu. Omówiono w nim kolejno język programowania, system budowania, menedżer pakietów oraz bazę danych, wykorzystywaną do przechowywania listy ostatnio otwartych plików.

4.1. C++20

C++ to kompilowany język programowania ogólnego przeznaczenia o statycznym typowaniu, stworzony przez Bjarne’a Stroustrupa. Prace nad językiem rozpoczęły się w 1979 roku, a jego oficjalne wydanie nastąpiło w 1985 roku [36]. Wywodzi się on z języka C i rozszerza go o programowanie obiektowe (*object-oriented programming*), programowanie generyczne (*generic programming*) za pomocą szablonów oraz zarządzanie zasobami poprzez idiom RAII (*Resource Acquisition Is Initialization*) [37]. Dzięki tym cechom C++ jest stosowany w obszarach wymagających wysokiej wydajności, takich jak systemy operacyjne, silniki gier, oprogramowanie wbudowane (*embedded*) oraz biblioteki graficzne.

Standard języka jest zarządzany przez komitet ISO i regularnie aktualizowany. Obraz aktualnego stanu ekosystemu C++ przedstawia badanie *The State of C++ 2025* [20], przeprowadzone przez JetBrains wśród 1800 programistów. Wynika z niego, że najszerzej stosowanym standardem pozostaje C++17, używany przez 43% respondentów, a C++20 stosuje 34% z nich. Wśród powodów niekorzystania z nowszych wersji dominuje opinia, że aktualny standard jest wystarczający do realizowanych zadań, wskazana przez 71% respondentów. Badanie wskazuje również, że dominującym kompilatorem jest GCC, używany przez 70% programistów, następnie Clang (45%) i MSVC (27%), co odzwierciedla wieloplatformowy charakter języka.

4.2. CMake

CMake to wieloplatformowe narzędzie do zarządzania procesem budowania projektów. Nie jest samodzielnym systemem budowania, lecz generatorem plików wejściowych dla narzędzi natywnych: Makefile, projektów Visual Studio, Ninja i innych [38]. Dzięki temu ten sam opis projektu zapisany w plikach `CMakeLists.txt` umożliwia budowanie aplikacji na różnych systemach operacyjnych i w różnych środowiskach programistycznych bez modyfikacji konfiguracji. Według badania *The State of C++ 2025* [20] CMake jest najczęściej stosowanym systemem budowania wśród programistów C++, używanym przez 59% ankietowanych.

4.3. Conan

Conan to menedżer pakietów z otwartym kodem źródłowym, przeznaczony dla języka C++, umożliwiający deklaratywne zarządzanie zależnościami projektu [47]. Konfiguracja opisana jest w pliku `conanfile.py`, w którym programista definiuje wymagane biblioteki wraz z ich wersjami. Conan pobiera pakiety z centralnego repozytorium, kompiluje je ze źródeł zgodnie z profilem docelowej platformy i generuje pliki integracyjne dla systemów budowania [39]. Pakiety dostępne w repozytorium są tworzone i aktualizowane przez społeczność, co zapewnia szeroki zasób bibliotek.

Według badania *The State of C++ 2025* [20] menedżery pakietów systemowych używane są przez 36% respondentów. Są one jednak zależne od dystrybucji i nie gwarantują dostępności konkretnych wersji bibliotek na wszystkich platformach. Conan rozwiązuje ten problem przez

budowanie pakietów ze źródeł z zachowaniem określonych wersji, co jest istotne w projekcie wymagającym spójnego środowiska na systemach Windows i Linux.

4.4. SQLite

SQLite to wbudowana relacyjna baza danych (*embedded relational database*) napisana w języku C, udostępniana jako biblioteka dołączana bezpośrednio do kodu aplikacji [48]. W odróżnieniu od klasycznych systemów zarządzania bazami danych, takich jak PostgreSQL czy MySQL, SQLite nie wymaga osobnego procesu serwera: cała baza danych przechowywana jest w jednym pliku na dysku, a dostęp do niej odbywa się przez wywołania funkcji bibliotecznych [21]. Dzięki temu rozwiązanie jest szczególnie przydatne w aplikacjach desktopowych, gdzie wymagana jest trwałość danych bez konieczności konfigurowania zewnętrznej infrastruktury.

4.5. Technologie związane z zastosowanymi bibliotekami

Każda z porównywanych bibliotek korzysta z dodatkowych komponentów ściśle związanych ze swoim ekosystemem lub modelem renderowania. Poniższe podrozdziały opisują wybrane technologie wykorzystane w prototypie: moduł Qt Charts stosowany w wariantcie Qt, biblioteki SDL2 i OpenGL stanowiące fundament wariantu Dear ImGui oraz bibliotekę STB do obsługi obrazów również w Dear ImGui.

4.5.1 Qt Charts

Qt Charts to osobny moduł frameworka Qt, udostępniający komponenty do tworzenia interaktywnych wykresów i wizualizacji danych [22]. Moduł opiera się na mechanizmie *Qt Graphics View Framework* i jest zintegrowany z systemem widżetów Qt, dzięki czemu wygenerowane wykresy mogą być osadzone w dowolnym miejscu hierarchii interfejsu użytkownika jak zwykle widżety.

Ponieważ Qt Charts to opcjonalny moduł, jego użycie wymaga jawnego włączenia zarówno na etapie konfiguracji zależności, jak i budowania projektu. W pliku konfiguracyjnym Conan moduł aktywowany jest przez opcję przekazywaną do biblioteki Qt, a w CMake konieczne jest dołączenie komponentu Charts do polecenia `find_package` oraz jawne zlinkowanie biblioteki `Qt6::Charts` [22].

4.5.2 SDL2, OpenGL

Dear ImGui jest biblioteką zajmującą się wyłącznie generowaniem interfejsu użytkownika: nie tworzy własnego okna ani nie obsługuje zdarzeń systemowych [11]. Wymaga zewnętrznej warstwy, która dostarczy kontekst okienkowy oraz moduł renderujący (*renderer*). W prototypie tę rolę pełnią dwie biblioteki: SDL2 i OpenGL.

SDL (*Simple DirectMedia Layer*) to wieloplatformowa biblioteka, zapewniająca ujednolicony dostęp do urządzeń wejściowych, dźwięku i innych zasobów systemowych [23]. W wersji 2.x (SDL2), zastosowanej w projekcie, biblioteka odpowiada za stworzenie okna z obsługą OpenGL, zarządzanie kontekstem graficznym oraz obsługę pętli zdarzeń. Inicjalizacja przebiega przez ustawienie parametrów kontekstu OpenGL, a następnie utworzenie okna.

OpenGL (*Open Graphics Library*) to wieloplatformowe i wolne od opłat licencyjnych (*royalty-free*) API graficzne, utrzymywane przez organizację Khronos Group [24]. Jako standard niezależny od systemu operacyjnego i systemu okienkowego, OpenGL definiuje interfejs programistyczny umożliwiający komunikację z procesorem graficznym (GPU) w celu renderowania grafiki 2D i 3D. Niezależność od platformy oznacza jednak, że OpenGL nie dostarcza mechanizmów do tworzenia okna ani kontekstu graficznego: tę rolę przejmuje SDL2.

4.5.3 STB do obsługi obrazów

Dear ImGui, jako biblioteka skupiona wyłącznie na generowaniu interfejsu użytkownika, nie dostarcza żadnych mechanizmów wczytywania plików graficznych [28]. ImGui operuje wyłącznie na identyfikatorach tekstur GPU (*ImTextureID*), pozostawiając programiście pełną odpowiedzialność za załadowanie pikseli z dysku i przesłanie ich do pamięci karty graficznej. Aby wyświetlić obraz, należy wykonać trzy kroki: zdekodować plik graficzny do tablicy bajtów, przesłać dane do GPU jako teksturę OpenGL, a następnie przekazać identyfikator tej tekstury do polecenia rysowania ImGui. Do realizacji pierwszego kroku zastosowano bibliotekę `stb_image`.

STB to kolekcja bibliotek nagłówkowych dla języków C i C++, opracowana przez Seana Barretta i udostępniana na licencji public domain lub MIT [40]. W prototypie wykorzystano dwa pliki z tej kolekcji: `stb_image.h` do wczytywania plików graficznych oraz `stb_image_write.h` do ich zapisywania.

4.6. Środowiska programistyczne

Poniższe podrozdziały opisują środowiska programistyczne (*Integrated Development Environments*) zastosowane podczas tworzenia prototypu. Wybór środowiska był częściowo podyktowany używaną biblioteką: Qt Creator oferuje natywną integrację z ekosystemem Qt, natomiast Visual Studio jest naturalnym środowiskiem dla projektów C++ na platformie Windows.

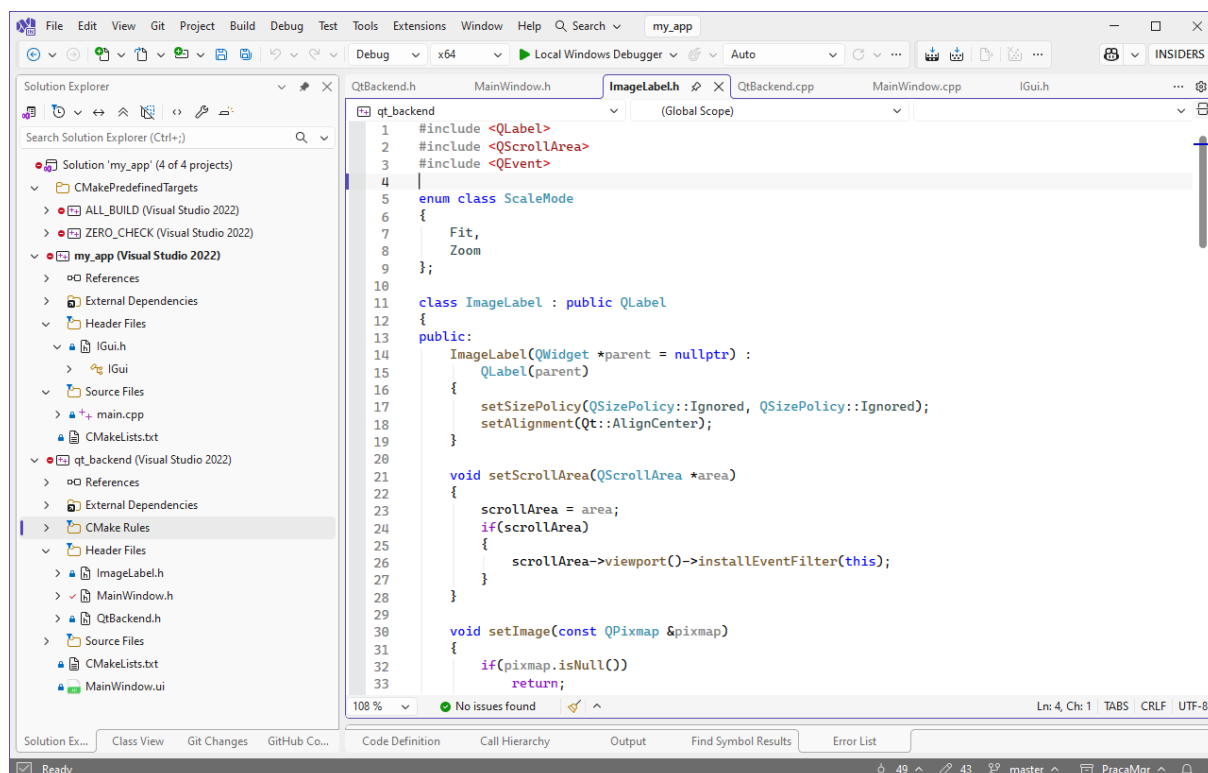
4.6.1 Visual Studio

Visual Studio to zintegrowane środowisko programistyczne firmy Microsoft, przeznaczone dla systemu Windows. Środowisko obsługuje wiele języków programowania, w tym C++, C#, Python, JavaScript oraz TypeScript, i integruje w jednym miejscu wszystkie etapy tworzenia oprogramowania: edycję kodu, kompilację, debugowanie, testowanie i wdrażanie [25].

Dostępne jest w trzech edycjach:

- *Community*, będącej bezpłatną wersją przeznaczoną dla studentów, programistów indywidualnych i projektów z otwartym kodem źródłowym;
- *Professional*, skierowanej do małych zespołów;
- *Enterprise*, przeznaczonej dla dużych organizacji.

Rysunek 4 przedstawia Visual Studio 2026 w edycji *Community*.



Rysunek 4. Okno Visual Studio 2026

Podstawową funkcją wspierającą pracę z kodem jest mechanizm IntelliSense, realizujący uzupełnianie kodu, podpowiedzi dotyczące parametrów wywoływanych funkcji oraz wyświetlanie pełnej deklaracji dowolnego identyfikatora po najechaniu na niego kursorem [26].

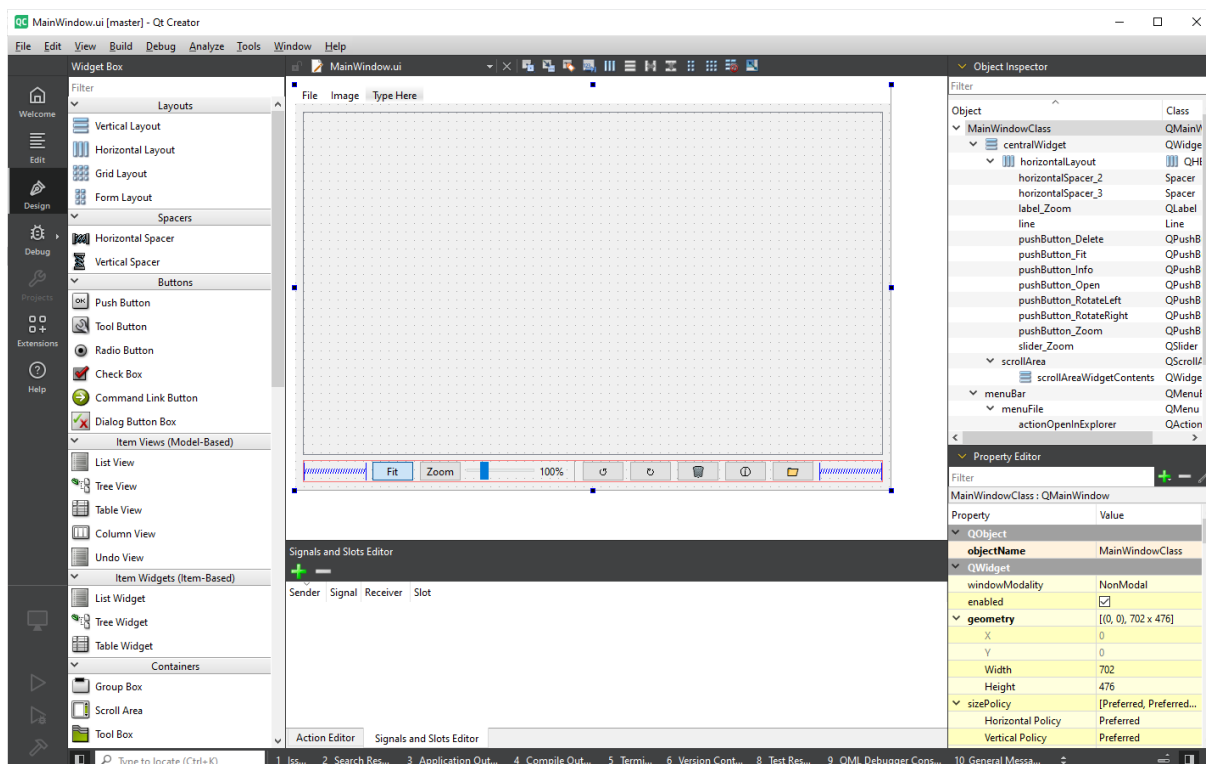
Funkcjonalność środowiska można rozszerzać przez system wtyczek dostępnych w *Visual Studio Marketplace*. Visual Studio integruje się również natywnie z systemem kontroli wersji Git, oferując graficzny interfejs do zarządzania gałęziami, przeglądania historii zmian i rozwiązywania konfliktów.

4.6.2 Qt Creator

Qt Creator to wieloplatformowe zintegrowane środowisko programistyczne, zaprojektowane z myślą o projektach opartych na frameworku Qt. Obsługuje projekty CMake, oferując natywną integrację z mechanizmami Qt, takimi jak nawigacja po połączeniach sygnał-slot czy podgląd dokumentacji klas Qt bezpośrednio w edytorze [41].

Głównym elementem środowiska jest Qt Designer, czyli wbudowane narzędzie graficzne, które służy do projektowania interfejsów użytkownika. Narzędzie to umożliwia stworzenie pliku `.ui`, który jest dokumentem XML opisującym hierarchię widżetów, ich właściwości i rozmieszczenie. Podczas kompilacji narzędzie `uic` (*UI Compiler*) generuje na podstawie takiego pliku nagłówki C++ [42].

Rozdzielenie definicji układu interfejsu od logiki aplikacji stanowi jeden z przykładów separacji odpowiedzialności promowanej przez framework Qt. Zmiany w układzie widżetów nie wymagają ingerencji w kod C++, a nagłówek z widżetami jest odtwarzany automatycznie przy każdej kompilacji. Rysunek 5 pokazuje narzędzie Qt Designer w środowisku Qt Creator z otwartym plikiem prototypu.



Rysunek 5. Qt Designer w Qt Creator otwarty z plikiem prototypu

5. Projekt i implementacja prototypu

Niniejszy rozdział opisuje projekt i implementację prototypowej aplikacji przeglądarki zdjęć, stworzonej na potrzeby porównania czterech bibliotek GUI dla języka C++. Kolejne podrozdziały omawiają przyjętą architekturę z wymiennymi modułami renderującymi, sposób budowania aplikacji dla różnych systemów operacyjnych oraz szczegóły implementacji poszczególnych backendów.

5.1. Architektura rozwiązania z wymiennymi backendami

Prototyp zbudowano z założeniem identycznej funkcjonalności zrealizowanej czterokrotnie, przy użyciu czterech różnych bibliotek GUI. Aby umożliwić rzetelne porównanie, głównym wymaganiem projektowym było oddzielenie logiki aplikacji od kodu odpowiedzialnego za interfejs użytkownika. W tym celu przyjęto architekturę opartą na abstrakcyjnym interfejsie.

Rdzeniem architektury jest klasa abstrakcyjna `IGui`, zdefiniowana w pliku `IGui.h`. Interfejs ten deklaruje jedną czysto wirtualną metodę `run()`, przyjmującą argumenty wiersza poleceń i zwracającą kod zakończenia procesu. Celowo ograniczona definicja wynika z charakteru porównywanych bibliotek. Każda z nich narzuca własny cykl życia aplikacji, własny mechanizm obsługi zdarzeń oraz własny model inicjalizacji. Próba ujednolicenia tych różnic na poziomie interfejsu prowadziłaby do sztucznych abstrakcji. Zamiast tego każdy backend przejmuje pełną kontrolę w momencie wywołania metody `run()` i samodzielnie zarządza cyklem życia okna, pętlą zdarzeń i zasobami. Kod źródłowy tego interfejsu przedstawia Listing 1.

Listing 1. Kod źródłowy interfejsu `IGui`

```
class IGui
{
public:
    virtual ~IGui() = default;
    virtual int run(int argc, char **argv) = 0;
};
```

Punkt wejścia aplikacji, zawarty w pliku `main.cpp`, jest świadomie minimalistyczny. Deklaruje on jedynie funkcję `create_gui()`, zwracającą inteligentny wskaźnik `std::unique_ptr<IGui>`, a następnie wywołuje metodę `run()` na zwróconym obiekcie. Funkcja `create_gui()` nie jest zdefiniowana w `main.cpp`. Jej implementacja znajduje się w pliku źródłowym wybranego backendu i zwraca konkretny typ implementujący `IGui`. Dzięki temu `main.cpp` przedstawione na Listing 2 nie zawiera żadnych odwołań do kodu konkretnej biblioteki i pozostaje niezmienny niezależnie od wybranego backendu.

Listing 2. Kod źródłowy punktu wejścia prototypu

```
#include "gui/IGui.h"

#include <memory>

std::unique_ptr<IGui> create_gui();

int main(int argc, char **argv)
{
    auto gui = create_gui();
    return gui->run(argc, argv);
}
```

Każdy backend kompilowany jest jako osobna biblioteka statyczna: `imgui_backend`, `qt_backend`, `wx_backend` lub `gtk_backend`. Plik wykonywalny linkowany jest wyłącznie z jedną z nich, wybraną na etapie kompilacji. Takie podejście eliminuje narzut wynikający z jednoczesnego linkowania wielu zależności, a ponadto odpowiada realistycznemu scenariuszowi tworzenia aplikacji, w którym wyboru biblioteki dokonuje się raz, przed wdrożeniem.

5.1.1 Budowanie dla wielu systemów operacyjnych

Prototyp zbudowano z założeniem kompilacji i uruchomienia na dwóch platformach: Windows i Linux. Do zarządzania zależnościami i budowania zastosowano CMake oraz Conan, opisane w podrozdziałach 4.2 i 4.3.

CMake pełni rolę centralnego narzędzia budowania dla wszystkich czterech implementacji prototypu. Plik główny definiuje standard języka na poziomie C++20 z wyłączonymi rozszerzeniami kompilatora (`CMAKE_CXX_EXTENSIONS OFF`), co zapewnia zgodność z czystym standardem C++ niezależnie od użytego kompilatora.

Do realizacji prototypu został zastosowany Conan w wersji 1, mimo że jest to wersja wycofywana na rzecz Conana 2. Przyczyną tego wyboru był brak pakietu GTK zgodnego z Conanem 2 w repozytorium pakietów Conan. Ponieważ wszystkie cztery moduły renderujące wymagały wspólnego środowiska budowania, zdecydowano się na wersję 1 jako jedyną obsługującą kompletny zestaw wymaganych bibliotek.

Wybór backendu odbywa się przez opcję `gui` przekazywaną do Conana podczas instalacji zależności:

```
conan install . --build=missing -s build_type=Debug -o gui=qt
```

Wartościami dozwolonymi dla opcji `gui` są: `imgui`, `qt`, `wx` oraz `gtk`. Wartość ta jest następnie przekazywana przez `conanfile.py` do CMake jako definicja `GUI_BACKEND`, na podstawie której CMake dołącza odpowiedni podkatalog z kodem backendu.

Mimo że CMake i Conan biorą na siebie obsługę większości różnic platformowych, kilka konfiguracji wymagało jawnych warunków uzależnionych od systemu operacyjnego. Różnice te dotyczą trzech obszarów: biblioteki `wxWidgets`, `Qt` oraz `Dear ImGui`.

Pierwsza zmiana dotyczy `wxWidgets` na Linuksie. `wxWidgets` oferuje opcjonalne funkcje, takie jak obsługa bezpiecznego przechowywania haseł (opcja *secretstore*) oraz alternatywny silnik rysowania grafiki oparty na bibliotece `Cairo`. Funkcje te opierają się na zewnętrznych bibliotekach systemowych (`libsecret-1-dev` oraz `libcairo2-dev`), których Conan nie dostarcza samodzielnie i które muszą być zainstalowane w systemie operacyjnym. Ponieważ żadna z tych opcji nie jest

wykorzystywana w prototypie, a ich obecność powodowała błędy budowania na Linuksie, zostały one wyłączone w konfiguracji Conana.

Druga zmiana dotyczy Qt na Linuksie. Bibliotekę Qt można zlinkować z aplikacją na dwa sposoby:

- statycznie, gdzie cały kod Qt trafia do jednego pliku wykonywalnego;
- dynamicznie, gdzie aplikacja korzysta z osobnych plików bibliotek w trakcie działania.

Na Linuksie Qt skompilowane statycznie, przy włączonej obsłudze protokołu wyświetlania Wayland, powoduje znany błąd po stronie Conana uniemożliwiający poprawną kompilację. Z tego powodu na Linuksie Qt budowane jest dynamicznie. Na Windows ograniczenie to nie występuje i zastosowano linkowanie statyczne. Funkcję konfiguracyjną pliku Conan z opisanymi warunkami platformowymi przedstawia Listing 3.

Listing 3. Kod źródłowy funkcji konfiguracyjnej pliku Conan

```
def configure(self):
    if self.settings.os == "Linux":
        self.options["wxwidgets"].secretstore = False
        self.options["wxwidgets"].cairo = False

        self.options["qt"].shared = True
```

Dodatkowym krokiem specyficznym dla Windows jest automatyczne uruchomienie narzędzia `windeployqt` po kompilacji backendu Qt w trybie bibliotek współdzielonych (*shared*). Narzędzie to kopiuje do katalogu z plikiem wykonywalnym wszystkie wymagane biblioteki DLL oraz wtyczki Qt.

Na poziomie CMake różnice platformowe pojawiają się w backendzie ImGui, gdzie biblioteka OpenGL eksportowana jest pod inną nazwą celu w zależności od systemu operacyjnego: `opengl::opengl` na Linuksie oraz `OpenGL::GL` na Windows.

Przyjęte podejście pozwala na kompilację prototypu na obu docelowych platformach bez modyfikacji kodu źródłowego, przy różnicach ograniczonych do kilku warunków w pliku Conan i plikach CMake poszczególnych backendów.

Kompilacja odbywa się komendą:

```
conan build .
```

5.1.2 Plik `conanfile` i opcje kompilacji

Plik `conanfile.py` to centralny punkt konfiguracji projektu. Definiuje on zależności, opcje kompilacji oraz kroki budowania dla wszystkich czterech backendów, tworząc jednolity punkt zarządzania procesem przygotowania środowiska.

Klasa główna `MyApp`, dziedzicząca po `ConanFile`, deklaruje pole `settings` zawierające standardowe parametry Conana: system operacyjny, kompilator, typ budowania oraz architekturę. Pole `options` definiuje jedną opcję specyficzną dla projektu: `gui`, której dozwolone wartości to `imgui`, `qt`, `wx` oraz `gtk`. Sekcja `default_options` zawiera ponadto wartości domyślne dla opcji samych bibliotek. Dla Qt zdefiniowano: `qt:shared` ustawione na `False` (linkowanie statyczne jako wartość domyślna), `qt:with_libjpeg` ustawione na `libjpeg` (obsługa formatu JPEG przez wtyczkę Qt) oraz `qt:qtcharts` ustawione na `True` (moduł wykresów wymagany przez widok histogramu w informacjach o pliku).

Metoda `requirements()` deklaruje zależności warunkowo, na podstawie wybranego backendu. Takie podejście eliminuje pobieranie zbędnych bibliotek i skraca czas przygotowania środowiska budowania. Dla backendu `imgui` wymagane są: biblioteka `imgui` w wersji 1.91.4, biblioteka `stb` zapewniająca dekodowanie obrazów rastrowych, `sdl` w wersji 2.30.8 odpowiedzialna za obsługę okna i wejścia, `opengl/system` dostarczający interfejs graficzny oraz `portable-file-dialogs` realizujący systemowe okna dialogowe wyboru pliku. Dla backendu `qt` jedyną zależnością jest `qt` w wersji 6.7.3, ponieważ biblioteka Qt zawiera wszystkie wymagane komponenty. Dla backendu `wx` wymagane są `wxwidgets` w wersji 3.2.6 oraz `sqlite3` w wersji 3.47.0. Sytuacja backendu `gtk` jest bardziej złożona: dostępna w repozytorium *ConanCenter* wersja 4.7.0 pochodzi z 2022 roku i deklaruje zależności od przestarzałych wersji bibliotek pomocniczych, z których część posiada nieaktualne adresy URL uniemożliwiające automatyczne pobranie. Z tego powodu wersje bibliotek `zlib`, `libffi`, `libxml2` oraz `libpng` zostały jawnie przypięte do nowszych wersji, nadpisując wersje wskazywane przez pakiet GTK. Zestawienie zależności backendów przedstawia Tabela 1.

Tabela 1. Zależności backendów

BACKEND	ZALEŻNOŚCI
DEAR IMGUI	<code>imgui/1.91.4</code> , <code>stb/cc1.20240531</code> , <code>sdl/2.30.8</code> , <code>opengl/system</code> , <code>portable-file-dialogs/0.1.0</code>
QT	<code>qt/6.7.3</code>
WXWIDGETS	<code>wxwidgets/3.2.6</code> , <code>sqlite3/3.47.0</code>
GTK	<code>gtk/4.7.0</code> , <code>zlib/1.2.13</code> , <code>libffi/3.4.4</code> , <code>libxml2/2.9.14</code> , <code>libpng/1.6.44</code>

Metoda `imports()` w `conanfile.py`, wywoływana po instalacji zależności, kopiuje pliki wiążące (bindings) biblioteki Dear ImGui do katalogu roboczego. Dear ImGui odpowiada wyłącznie za rysowanie elementów interfejsu użytkownika i nie integruje się samodzielnie z systemem okien ani podsystemem graficznym. Integracja z konkretną platformą realizowana jest przez osobne pliki wiążące dystrybuowane razem z biblioteką. Dla backendu `imgui` wymagane są pliki implementacji dla SDL2 oraz OpenGL3. Pliki te kopiowane są z katalogu `res/bindings` pakietu do podkatalogu `bindings` w katalogu budowania, skąd korzysta z nich konfiguracja CMake backendu ImGui.

Metoda `build()` wywołuje CMake, przekazując zmienną `GUI_BACKEND` z wartością wybranego backendu. Na tej podstawie CMake dołącza odpowiedni podkatalog z kodem modułu i linkuje właściwą bibliotekę statyczną. Dla Qt przekazywana jest dodatkowo zmienna `QT_SHARED`, informująca o trybie linkowania.

5.1.3 Mechanizm wyboru backendu w czasie kompilacji

Wybór backendu realizowany jest jako przepływ wartości przez kolejne warstwy narzędzi budowania: od parametru wiersza poleceń, przez system Conan, aż do zmiennej CMake, która determinuje, który kod jest kompilowany i linkowany.

Użytkownik podaje wybrany backend jako wartość opcji `gui` podczas instalacji zależności przez Conan. Wartość ta jest dostępna w pliku `conanfile.py` jako `self.options.gui` i przekazywana do CMake w metodzie `build()` jako definicja zmiennej `GUI_BACKEND`.

Na poziomie CMake zmienna `GUI_BACKEND` trafia do pliku `src/CMakeLists.txt`, gdzie decyduje o tym, który podkatalog zostaje dołączony do procesu budowania. Każdy z czterech wariantów skutkuje wywołaniem `add_subdirectory()` dla odpowiedniego backendu oraz zlinkowania powstałej biblioteki statycznej z plikiem wykonywalnym. W przypadku podania nieznanej wartości CMake przerywa proces budowania komunikatem błędu krytycznego (`FATAL_ERROR`), co zabezpiecza przed przypadkowym pominięciem konfiguracji.

Ostatnim ogniwem łączącym `main.cpp` z kodem konkretnej biblioteki jest funkcja fabryczna `create_gui()`. Zadeklarowana jako zewnętrzna w `main.cpp`, zdefiniowana jest w pliku źródłowym każdego backendu i zwraca instancję właściwej klasy implementującej `IGui`. Dzięki temu `main.cpp` nie zawiera żadnych dyrektyw warunkowych ani odwołań do konkretnych bibliotek i pozostaje niezmienny niezależnie od wybranego backendu.

Przyjęte podejście wyboru w czasie kompilacji jest świadomą decyzją projektową. Alternatywą byłoby dynamiczne ładowanie bibliotek w trakcie działania programu, co umożliwiłoby zmianę backendu bez rekompilacji. Dla celów porównawczych niniejszej pracy takie podejście byłoby jednak nadmiarowe i komplikowałoby infrastrukturę budowania. Kompilacja z jedną wybraną biblioteką eliminuje ponadto jakikolwiek narzut wynikający z obecności kodu pozostałych backendów w pliku wykonywalnym, co odpowiada realistycznemu scenariuszowi tworzenia aplikacji produkcyjnej.

5.1.4 Konfiguracja CMake

Konfiguracja CMake projektu składa się z trzech poziomów: głównego pliku `CMakeLists.txt` w katalogu projektu, pliku `src/CMakeLists.txt` odpowiedzialnego za budowanie pliku wykonywalnego i wybór backendu, oraz osobnych plików `CMakeLists.txt` dla każdego backendu.

Główny plik `CMakeLists.txt` ustawia minimalną wymaganą wersję CMake, definiuje projekt, wybiera standard C++20 dla wszystkich celów oraz włącza eksport bazy danych kompilacji (`compile_commands.json`).

Plik `src/CMakeLists.txt` definiuje cel główny `my_app` jako plik wykonywalny składający się z `main.cpp` i nagłówka `IGui.h`. Katalog bieżący dodawany jest do ścieżek wyszukiwania nagłówków, co umożliwia stosowanie ścieżek względem katalogu `src`. Następnie, na podstawie zmiennej `GUI_BACKEND`, dołączany jest podkatalog odpowiedniego backendu i wynikowa biblioteka statyczna linkowana z plikiem wykonywalnym. Katalog wyjściowy pliku wykonywalnego ustawiany jest na `${CMAKE_BINARY_DIR}/bin`.

Każdy backend posiada własny plik `CMakeLists.txt`, wyszukujący wymagane zależności i definiujący bibliotekę statyczną. Wszystkie backendy eksportują katalog `src` projektu jako publiczną ścieżkę nagłówków, dzięki czemu plik wykonywalny może odwoływać się do `IGui.h` bez powielania konfiguracji.

Backend Qt wyszukuje pakiet Qt6 z komponentami `Core`, `Widgets`, `Gui`, `PrintSupport`, `Charts`, `OpenGLWidgets` oraz `Sql`. Biblioteka tworzona jest poleceniem `qt_add_library()`. Do celów linkowania dołączana jest zawsze wtyczka `Qt6::QJpegPlugin`, zapewniająca obsługę formatu JPEG. W przypadku budowania statycznego linkowane są dodatkowo: wtyczka sterownika bazy danych `Qt6::SQLiteDriverPlugin` oraz wyłącznie na Windows, wtyczka integracji z systemem okien `Qt6::QWindowsIntegrationPlugin`. Wtyczki te są wymagane przy statycznym linkowaniu, ponieważ Qt nie może wtedy ładować ich w trakcie działania programu.

Backend `wxWidgets` wyszukuje pakiety `wxWidgets` oraz `SQLite3` i tworzy bibliotekę statyczną `wx_backend` linkowaną z `wxWidgets::wxWidgets` oraz `SQLite::SQLite3`. Biblioteka `SQLite3` wykorzystywana jest przez backend do przechowywania historii ostatnio otwieranych plików.

Backend Dear ImGui wyszukuje pakiety `imgui`, `stb`, `SDL2`, `portable-file-dialogs` oraz `opengl`. Biblioteka statyczna `imgui_backend` obejmuje oprócz własnych plików źródłowych również pliki wiążące skopiowane przez Conan do katalogu `bindings`: `imgui_impl_opengl3.cpp` oraz `imgui_impl_sdl2.cpp`. Katalog `bindings` dołączany jest jako publiczna ścieżka nagłówków.

Backend GTK wyszukuje pakiet `gtk` i tworzy bibliotekę statyczną `gtk_backend` linkowaną wyłącznie z `gtk::gtk`. Pakiet GTK dostarcza wszystkie wymagane zależności pośrednio, w ramach własnej definicji pakietu Conan, co upraszcza konfigurację CMake tego backendu.

5.1.5 Wspólny schemat bazy danych

W niniejszym prototypie baza danych służy do przechowywania historii ostatnio otwieranych plików graficznych. Po każdym otwarciu obrazu ścieżka do pliku i data ostatniego otwarcia zapisywana jest w tabeli `recent_files`, a menu aplikacji jest aktualizowane na podstawie zapytania zwracającego dziesięć najnowszych wpisów.

Spośród porównywanych bibliotek jedynie Qt udostępnia wbudowany moduł bazodanowy. W wariancie `wxWidgets` wykorzystano w tym celu zewnętrzną bibliotekę `SQLite3`. Funkcję tę zrealizowano w obu tych wariantach, a pominięto w Dear ImGui i GTK, traktując ją jako ilustrację możliwości wykraczających poza sam interfejs użytkownika.

Schemat bazy danych składa się z jednej tabeli, tworzonej przy pierwszym uruchomieniu aplikacji. Zapytanie tworzące tę tabelę zawiera Listing 4.

Listing 4. Polecenie SQL tworzące tabelę w bazie danych

```
CREATE TABLE IF NOT EXISTS recent_files (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    file_path TEXT UNIQUE,
    last_opened DATETIME DEFAULT CURRENT_TIMESTAMP
)
```

Kolumna `file_path` posiada ograniczenie unikalności, dzięki czemu ponowne otwarcie tego samego pliku nie powoduje duplikacji wiersza, lecz aktualizację znacznika czasu z użyciem klauzuli `ON CONFLICT ... DO UPDATE`.

Szczegóły implementacji bazy danych dla poszczególnych bibliotek znajdują się w podrozdziałach 5.2.5 i 5.3.3.

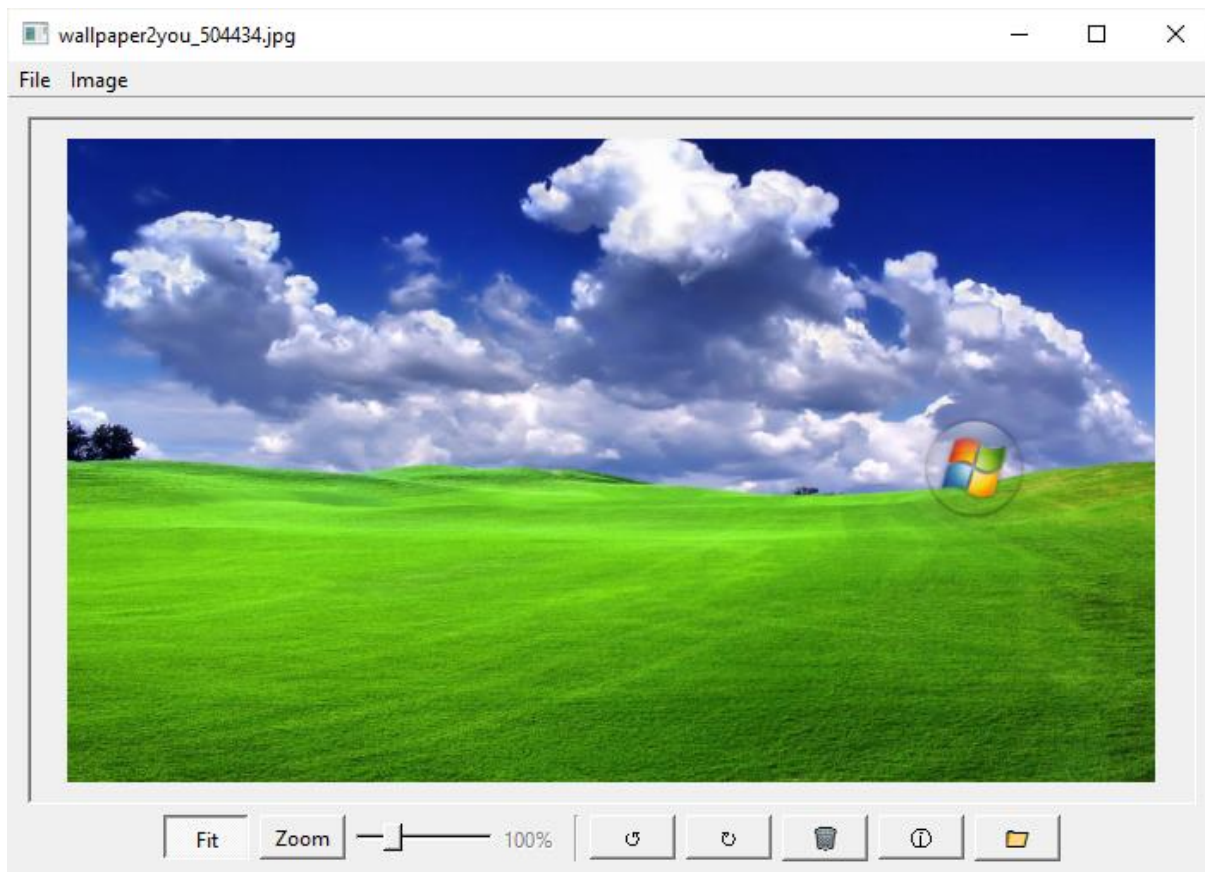
5.2. Implementacja Qt

Niniejszy podrozdział omawia szczegółowo realizację prototypu przeglądarki zdjęć w bibliotece Qt. Implementacja reprezentuje podejście oparte na trybie zachowanym (*Retained Mode*), w którym hierarchia obiektów interfejsu utrzymywana jest przez bibliotekę, a aplikacja jedynie reaguje na zdarzenia. Logikę aplikacji rozdzielono pomiędzy dwie klasy: `MainWindow`, dziedziczącą po `QMainWindow` i odpowiadającą za okno główne oraz obsługę poszczególnych funkcji, a także `ImageLabel`, dziedziczącą po `QLabel` i zajmującą się prezentacją oraz skalowaniem obrazu.

Cały moduł budowany jest jako biblioteka statyczna `qt_backend` i udostępnia wspólny interfejs `IGui`, dzięki czemu można go wymieniać z pozostałymi modułami graficznymi na etapie kompilacji, zgodnie z przyjętą architekturą wymiennych modułów. Implementacja wykorzystuje kilka

modułów biblioteki Qt, odpowiadających kolejnym funkcjom aplikacji: Qt Widgets do budowy interfejsu, Qt Charts do prezentacji histogramu, Qt SQL do współpracy z bazą danych oraz Qt Print Support do obsługi drukowania.

Kolejne punkty przedstawiają budowę okna głównego i układ interfejsu, mechanizm sygnałów i slotów łączący kontrolki z logiką aplikacji, sposób przetwarzania i wyświetlania obrazów, prezentację histogramu, integrację z bazą danych oraz obsługę drukowania. Rysunek 6 przedstawia okno główne prototypu zrealizowanego w bibliotece Qt.



Rysunek 6. Okno główne prototypu Qt uruchomionego w systemie Windows

5.2.1 Okno główne i układ interfejsu

Okno główne aplikacji zaprojektowano w narzędziu Qt Designer i zapisano w pliku `MainWindow.ui`. Podczas kompilacji narzędzie uic przekształca ten opis w nagłówek `ui_MainWindow.h`, zawierający klasę `Ui::MainWindowClass` z polami odpowiadającymi poszczególnym kontrolkom. Klasa `MainWindow` przechowuje instancję tego obiektu w polu `ui` i inicjuje cały interfejs jednym wywołaniem `ui.setupUi(this)`. Takie rozdzielanie opisu wyglądu od logiki aplikacji jest typowym dla Qt sposobem budowy interfejsu i umożliwia modyfikowanie układu kontrolki bez ingerencji w kod źródłowy.

Rozmieszczeniem kontrolki zarządzają menedżery układu (*layout managers*), które samodzielnie wyznaczają położenie i rozmiar elementów oraz dostosowują je do zmian wielkości okna. Dzięki temu interfejs zachowuje poprawny wygląd niezależnie od rozdzielczości ekranu czy ręcznego skalowania okna, bez potrzeby określania współrzędnych poszczególnych elementów. Główny obszar okna tworzy pionowy układ `QVBoxLayout`, w którym umieszczono obszar przewijania `QScrollArea`, prezentujący obraz, oraz poziomy pasek narzędzi `QHBoxLayout` z przyciskami sterującymi.

Pasek narzędzi zawiera komplet operacji dostępnych dla wyświetlanego obrazu. Znajdują się na nim dwa przełączane przyciski wyboru trybu skalowania (`pushButton_Fit` oraz `pushButton_Zoom`), suwak powiększenia `slider_Zoom` wraz z etykietą procentową `label_Zoom`, a także przyciski obrotu w lewo i w prawo, usunięcia pliku, wyświetlenia informacji o obrazie oraz otwarcia nowego pliku. Aby spośród dwóch trybów skalowania aktywny pozostawał zawsze tylko jeden, przyciski `pushButton_Fit` i `pushButton_Zoom` powiązano w grupę `QButtonGroup`, która automatycznie pilnuje ich wzajemnego wykluczania.

Operacje rzadziej używane udostępniono w pasku menu, reprezentowane przez obiekty `QAction` rozmieszczone w dwóch menu:

- `File`, zawierającym polecenia drukowania, zapisu pod nową nazwą oraz otwarcia pliku w eksploratorze systemowym;
- `Image`, obejmującym korekcję jasności i kontrastu.

Część poleceń otrzymała skróty klawiszowe, na przykład `Ctrl+P` dla drukowania. Do menu `File` dodawane jest dodatkowo, już z poziomu kodu, podmenu `Recent Files` z listą ostatnio otwieranych plików.

Sam obraz prezentuje obiekt klasy `ImageLabel`, tworzony dynamicznie w konstruktorze klasy `MainWindow` i dodawany do układu wewnątrz obszaru przewijania. Rozwiązanie to oddziela kontrolkę odpowiadającą za wyświetlanie i skalowanie obrazu od pozostałych elementów interfejsu zdefiniowanych w pliku `MainWindow.ui`, co porządkuje strukturę kodu i ułatwia zarządzanie sposobem prezentacji obrazu.

5.2.2 System sygnałów i slotów

Komunikację pomiędzy kontrolkami a logiką aplikacji zrealizowano przy użyciu mechanizmu sygnałów i slotów (*signals and slots*), stanowiącego podstawowy sposób obsługi zdarzeń w bibliotece Qt. Sygnał jest emitowany przez obiekt w reakcji na określone zdarzenie, na przykład naciśnięcie przycisku, natomiast slot to funkcja wywoływana w odpowiedzi na ten sygnał. Połączenie pomiędzy nimi tworzy statyczna metoda `QObject::connect`, która wiąże nadawcę z odbiorcą bez wprowadzania bezpośredniej zależności między tymi obiektami. Dzięki temu kontrolka emitująca sygnał nie musi wiedzieć nic o obiekcie, który go obsługuje, co sprzyja luźnemu powiązaniu elementów aplikacji.

Aby klasa mogła brać udział w tym mechanizmie, musi dziedziczyć po `QObject` i zawierać makro `Q_OBJECT`, tak jak klasa `MainWindow`. Makro to włącza obsługę systemu metaobiektów (*Meta-Object System*), na którym opiera się działanie sygnałów i slotów. Na jego podstawie kompilator metaobiektów generuje, przed właściwą kompilacją, dodatkowy kod odpowiedzialny za obsługę połączeń w czasie działania programu [6].

W prototypie zastosowano składnię połączeń opartą na wskaźnikach do funkcji składowych, sprawdzaną podczas kompilacji. Większość połączeń korzysta z wyrażeń lambda, które przekazują obsługę zdarzenia do odpowiedniej metody klasy `MainWindow`. Przykładowo, naciśnięcie przycisku obrotu w lewo wiązane jest z wywołaniem metody `RotateImageLabel`. Połączenie to zawiera Listing 5.

Listing 5. Połączenie sygnału przycisku obrotu z metodą

```
connect(ui.pushButton_RotateLeft, &QPushButton::clicked, this,
[this] () {
    RotateImageLabel(90);
});
```

Część sygnałów przekazuje wraz ze zdarzeniem dodatkowe dane. Przykładowo, suwak powiększenia emituje sygnał `valueChanged` z nową wartością, którą slot wykorzystuje do ustawienia poziomu skalowania obrazu oraz aktualizacji etykiety procentowej.

Ten sam mechanizm wykorzystywany jest również w innych miejscach aplikacji. Pozycje podmenu `Recent Files` tworzone dynamicznie otrzymują połączenia wczytujące wskazany plik, a suwaki w oknie korekcji jasności i kontrastu aktualizują podgląd obrazu na bieżąco poprzez sygnał `valueChanged`. Dzięki jednolitemu zastosowaniu sygnałów i slotów obsługa zdarzeń w całej aplikacji opiera się na spójnym i czytelnym schemacie.

5.2.3 Przetwarzanie i wyświetlanie obrazów

W prototypie wczytany obraz przechowywany jest w polu `originalPixmap` typu `QPixmap`, czyli w reprezentacji zoptymalizowanej pod kątem wyświetlania na ekranie. Operacje modyfikujące zawartość pikseli, takie jak korekcja jasności i kontrastu, wykonywane są jednak na obiekcie `QImage`, który zapewnia bezpośredni dostęp do danych obrazu w pamięci. Taki podział wynika z odmiennego przeznaczenia obu klas: `QImage` służy do przetwarzania pikseli, natomiast `QPixmap` do efektywnego rysowania na ekranie. Wszystkie operacje wykonywane są na kopii obrazu znajdującej się w pamięci, bez modyfikowania pliku źródłowego na dysku.

Za prezentację obrazu odpowiada autorska klasa `ImageLabel`, dziedzicząca po `QLabel` i rozszerzona o obsługę skalowania. Klasa ta przechowuje oryginalny obraz oraz udostępnia dwa tryby skalowania określone wyliczeniem (*enum*) `ScaleMode`: dopasowanie do okna (*Fit*) oraz powiększenie o zadany procent (*Zoom*). Wybór trybu następuje za pomocą przycisków paska narzędzi, a właściwe przeliczenie rozmiaru obrazu realizuje prywatna metoda `updateScaledPixmap`. Jej kod źródłowy zawiera Listing 6.

Listing 6. Funkcja dopasowująca rozmiar obrazu do okna

```
void updateScaledPixmap()
{
    if(originalPixmap.isNull())
        return;

    if(scaleMode == ScaleMode::Fit)
    {
        QSize targetSize = scrollArea ? scrollArea->viewport()-
>size() : size();
        if(targetSize.isEmpty())
            targetSize = size();
        setPixmap(originalPixmap.scaled(targetSize,
Qt::KeepAspectRatio, Qt::SmoothTransformation));
    }
    if(scaleMode == ScaleMode::Zoom)
    {
        QSize zoomedSize = originalPixmap.size() * zoomPercent
/ 100;
        setPixmap(originalPixmap.scaled(zoomedSize,
Qt::KeepAspectRatio, Qt::SmoothTransformation));
        setFixedSize(zoomedSize);
    }
}
```

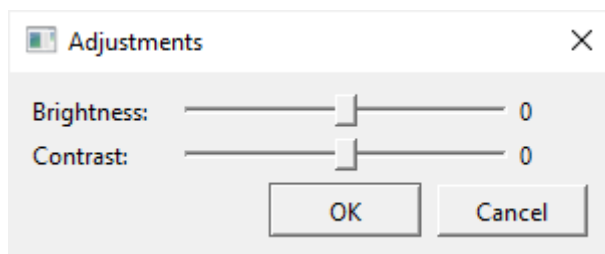
W trybie dopasowania obraz skalowany jest do rozmiaru widoku obszaru przewijania (QScrollArea) z zachowaniem proporcji. Wykorzystywana jest do tego metoda `scaled` z opcjami `Qt::KeepAspectRatio`, zapobiegającą zniekształceniu obrazu oraz `Qt::SmoothTransformation`, zapewniającą lepszą jakość. Aby obraz dostosowywał się do zmian wielkości okna, klasa `ImageLabel` reaguje na zdarzenia zmiany rozmiaru na dwa sposoby: przesłania metodę `resizeEvent`, wywoływaną przy zmianie rozmiaru samej kontrolki oraz instaluje filtr zdarzeń (*event filter*) na widoku obszaru przewijania, dzięki czemu wykrywa również zmiany rozmiaru jego widoku. W trybie powiększenia rozmiar obrazu wyznaczany jest na podstawie wartości procentowej pobranej z suwaka, a kontrolce nadawany jest ustalony rozmiar metodą `setFixedSize`, co pozwala przewijać powiększony obraz w obrębie obszaru przewijania.

Obrót obrazu realizuje metoda `RotateImageLabel`, która zmienia przechowywany kąt `rotateAngle` o wielokrotność dziewięćdziesięciu stopni, po czym wywołuje metodę `UpdateDisplayedImage` odpowiedzialną za ponowne zbudowanie wyświetlanej zawartości. Metoda ta pobiera oryginalny obraz, stosuje aktualne ustawienia, a następnie obraca obraz przy użyciu obiektu `QTransform`. Tak przygotowany obraz przekazywany jest do kontrolki `ImageLabel`.

Parametry jasności i kontrastu ustawiane są w osobnym oknie dialogowym, otwieranym z menu `Image`. Okno to zawiera dwa suwaki o zakresie od -100 do +100, których przesunięcie, dzięki opisanemu wcześniej mechanizmowi sygnałów i slotów, natychmiast aktualizuje podgląd obrazu. Zatwierdzenie okna utrwała wprowadzone ustawienia, natomiast jego anulowanie przywraca wartości sprzed otwarcia i odtwarza pierwotny wygląd obrazu.

Korekcję jasności i kontrastu zawiera metoda `ApplyAdjustments`, działająca bezpośrednio na obiekcie `QImage`. Metoda iteruje po kolejnych wierszach obrazu, uzyskując wskaźnik do danych wiersza za pomocą metody `scanLine`, i dla każdego piksela przelicza składowe czerwoną, zieloną i

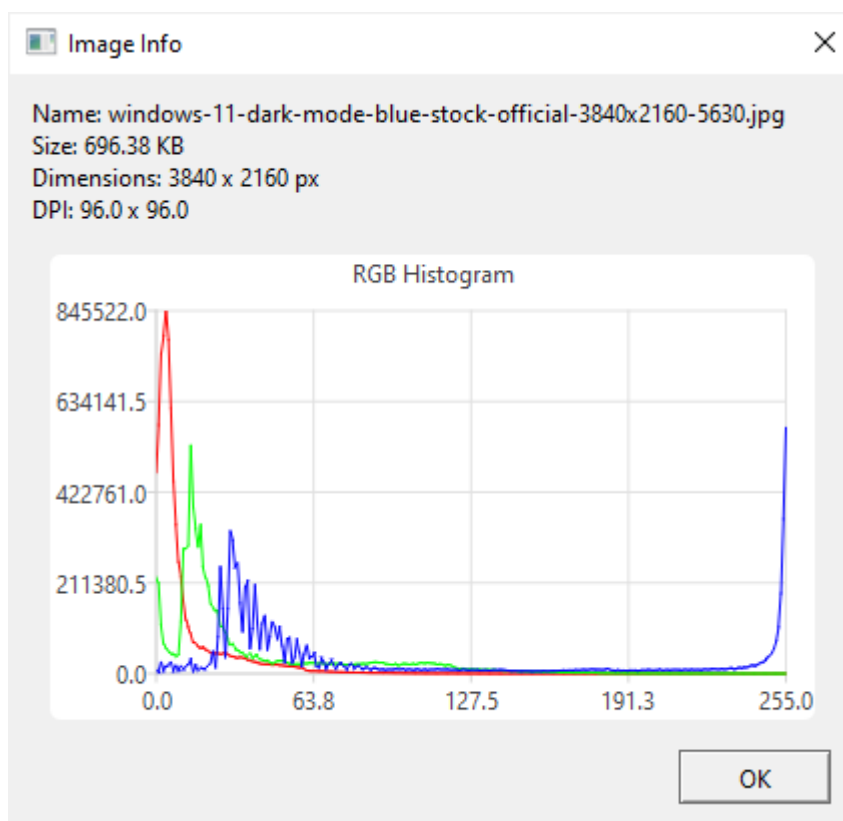
niebieską, zachowując przy tym wartość kanału przezroczystości. Jasność stosowana jest jako przesunięcie wartości składowych, a kontrast jako współczynnik skalujący odchylenie od wartości środkowej. Wartości wynikowe ograniczane są do zakresu od zera do 255 funkcją `qBound`, co zapobiega przekłamaniam barw przy skrajnych ustawieniach. Jeżeli oba parametry mają wartość zerową, metoda kończy działanie bez modyfikowania obrazu, unikając zbędnych obliczeń. Wygląd okna korekcji jasności i kontrastu przedstawia Rysunek 7.



Rysunek 7. Okno dialogowe korekcji jasności i kontrastu w Qt

5.2.4 Histogram z wykorzystaniem Qt Charts

Jednym z elementów funkcjonalnych prototypu jest histogram prezentujący rozkład wartości składowych barwnych wyświetlanego obrazu. Do jego wizualizacji wykorzystano moduł Qt Charts, dostarczający gotowe komponenty do tworzenia wykresów. Histogram generowany jest w metodzie `ShowImageInfo`, wspólnie z metadany pliku i prezentowany w oknie dialogowym. Rysunek 8 przedstawia okno dialogowe z metadanymi obrazu oraz histogramem RGB.



Rysunek 8. Okno dialogowe Qt z metadanymi i histogramem RGB obrazu

Obliczenia rozpoczyna analiza obrazu reprezentowanego przez obiekt `QImage`. Dla każdego z trzech kanałów barwnych (czerwonego, zielonego i niebieskiego) przygotowano 256-elementową

tablicę zliczającą liczbę wystąpień poszczególnych poziomów jasności. Przejście po wszystkich pikselach obrazu pozwala odczytać składowe za pomocą funkcji `qRed`, `qGreen` oraz `qBlue` i zwiększyć odpowiednie liczniki, co przedstawia Listing 7. Histogram budowany jest po nałożeniu korekcji obrazu, dzięki czemu odwzorowuje aktualny stan jasności i kontrastu, a nie wyłącznie dane pierwotne.

Listing 7. Kod źródłowy iteracji po każdym pikselu w Qt

```
for (int y = 0; y < img.height(); y++)
{
    for (int x = 0; x < img.width(); x++)
    {
        QRgb pixel = img.pixel(x, y);
        histR[qRed(pixel)]++;
        histG[qGreen(pixel)]++;
        histB[qBlue(pixel)]++;
    }
}
```

Zgromadzone dane przekształcane są na postać graficzną. Dla każdego kanału tworzona jest osobna seria `QLineSeries`, której przypisywany jest odpowiadający kolor. Punkty serii powstają przez dodanie par wartości, w których współrzędna pozioma oznacza poziom jasności (od 0 do 255), a pionowa liczbę pikseli o tej jasności. Trzy serie dodawane są do obiektu `QChart`, dla którego automatycznie generowane są osie. Gotowy wykres prezentowany jest w komponencie `QChartView` z włączonym wygładzaniem krawędzi (*antialiasing*), co poprawia czytelność linii.

Dostępność rozbudowanych modułów w ramach jednego, spójnego ekosystemu to duża zaleta Qt. Wizualizacja histogramu nie wymaga sięgania po zewnętrzne zależności ani samodzielnego rysowania wykresu, co upraszcza implementację.

5.2.5 Integracja z bazą danych QSqlDatabase

Oprócz komponentów odpowiedzialnych za interfejs graficzny, biblioteka Qt udostępnia szereg modułów rozszerzających jej zastosowanie poza warstwę prezentacji. W prototypie wykorzystano moduł Qt SQL, którego centralnym elementem jest klasa `QSqlDatabase`. Celem tej części implementacji było zaprezentowanie dodatkowych możliwości biblioteki Qt, niezwiązanych bezpośrednio z budową interfejsu, lecz stanowiących element kompletnego ekosystemu.

Funkcjonalność oparta na bazie danych polega na zapamiętywaniu ostatnio otwieranych plików graficznych. Dzięki temu po ponownym uruchomieniu aplikacji użytkownik uzyskuje dostęp do listy wcześniej przeglądanych obrazów, dostępnej z poziomu podmenu w głównym menu programu.

Jako mechanizm składowania danych zastosowano bazę SQLite, obsługiwaną przez wbudowany w Qt sterownik. Plik bazy umieszczany jest w systemowym katalogu danych aplikacji, co zapewnia zgodność z konwencjami obu wspieranych systemów operacyjnych. Każde otwarcie obrazu powoduje zapisanie jego ścieżki wraz ze znacznikiem czasu, przy czym ponowne otwarcie znanego już pliku jedynie aktualizuje ten znacznik, zamiast tworzyć duplikat. Listing 8 przedstawia ten proces. Przy budowie podmenu pobieranych jest dziesięć najnowszych pozycji, uporządkowanych według czasu otwarcia, a każdej z nich przypisywana jest akcja umożliwiająca ponowne wczytanie obrazu.

Listing 8. Dodanie rekordu z użyciem QSqlDatabase

```
QSqlQuery query(db);
query.prepare(R"(
    INSERT INTO recent_files (file_path, last_opened)
    VALUES (:path, CURRENT_TIMESTAMP)
    ON CONFLICT(file_path) DO UPDATE SET last_opened =
    CURRENT_TIMESTAMP
)");
query.bindValue(":path", path);
query.exec();
```

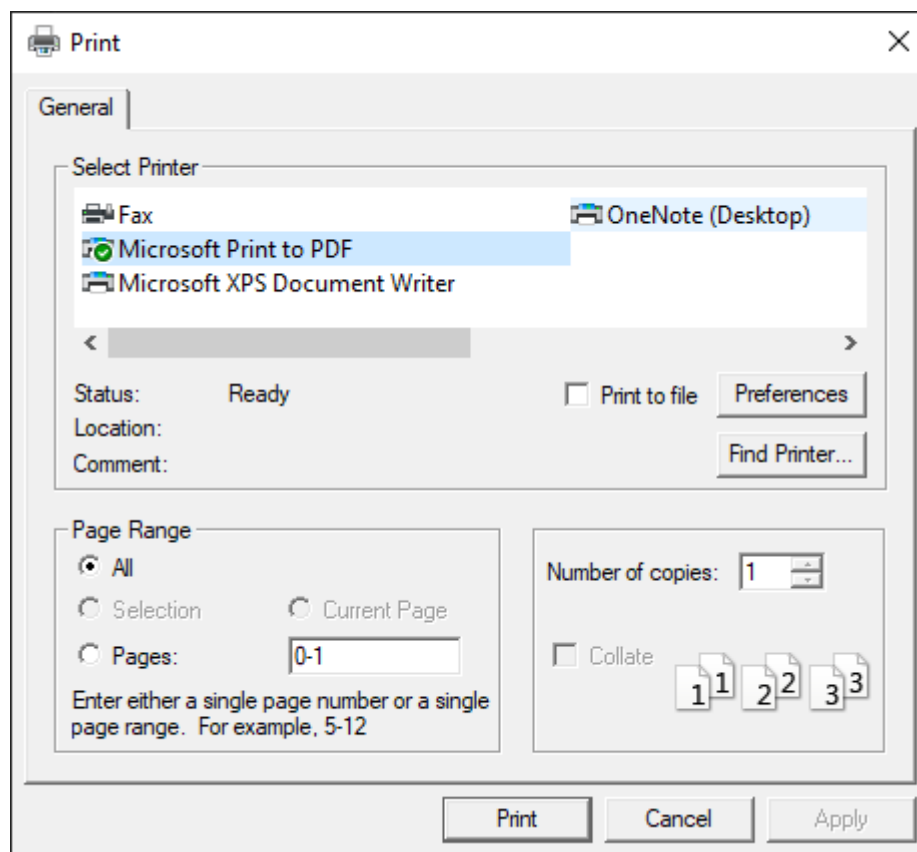
Przedstawione rozwiązanie pokazuje, że zakres Qt wykracza poza samą bibliotekę graficzną i pełni rolę kompletnego środowiska programistycznego. Odróżnia to Qt od bibliotek skupionych wyłącznie na warstwie graficznej.

5.2.6 Drukowanie

Kolejną funkcją wykraczającą poza podstawową prezentację obrazów jest możliwość wydruku. Realizuje ją moduł Qt Print Support, dostarczający komponenty obsługujące współpracę z drukarkami niezależnie od platformy systemowej.

Centralnym elementem tego mechanizmu jest klasa `QPrinter`. Stanowi ona implementację interfejsu urządzenia rysującego (`QPaintDevice`), co oznacza, że z punktu widzenia kodu drukarka traktowana jest tak samo jak inne urządzenia wyjściowe, na przykład obszar okna czy mapa bitowa w pamięci. Obiekt `QPrinter` przechowuje parametry zadania wydruku, takie jak rozdzielczość, rozmiar i orientacja strony czy marginesy, a operacje rysowania kierowane na niego zamieniane są na zawartość kolejnych stron dokumentu. Dzięki temu obraz przeznaczony do wydruku przygotowywany jest przy użyciu tej samej abstrakcji rysującej, którą Qt używa również do rysowania na ekranie. Obraz skalowany jest do rozmiaru strony z zachowaniem oryginalnych proporcji, co zapobiega jego zniekształceniu.

Qt nie realizuje drukowania samodzielnie. Biblioteka nie komunikuje się bezpośrednio z urządzeniem, lecz przygotowuje zawartość strony i przekazuje ją do natywnego podsystemu drukowania danego systemu operacyjnego, który odpowiada za właściwą obsługę sterownika i wysłanie zadania do drukarki. Wykorzystanie natywnych mechanizmów sprawia, że proces drukowania jest dla użytkownika znajomy i spójny z innymi aplikacjami w systemie. Natywne okno drukowania systemu Windows pokazuje Rysunek 9.

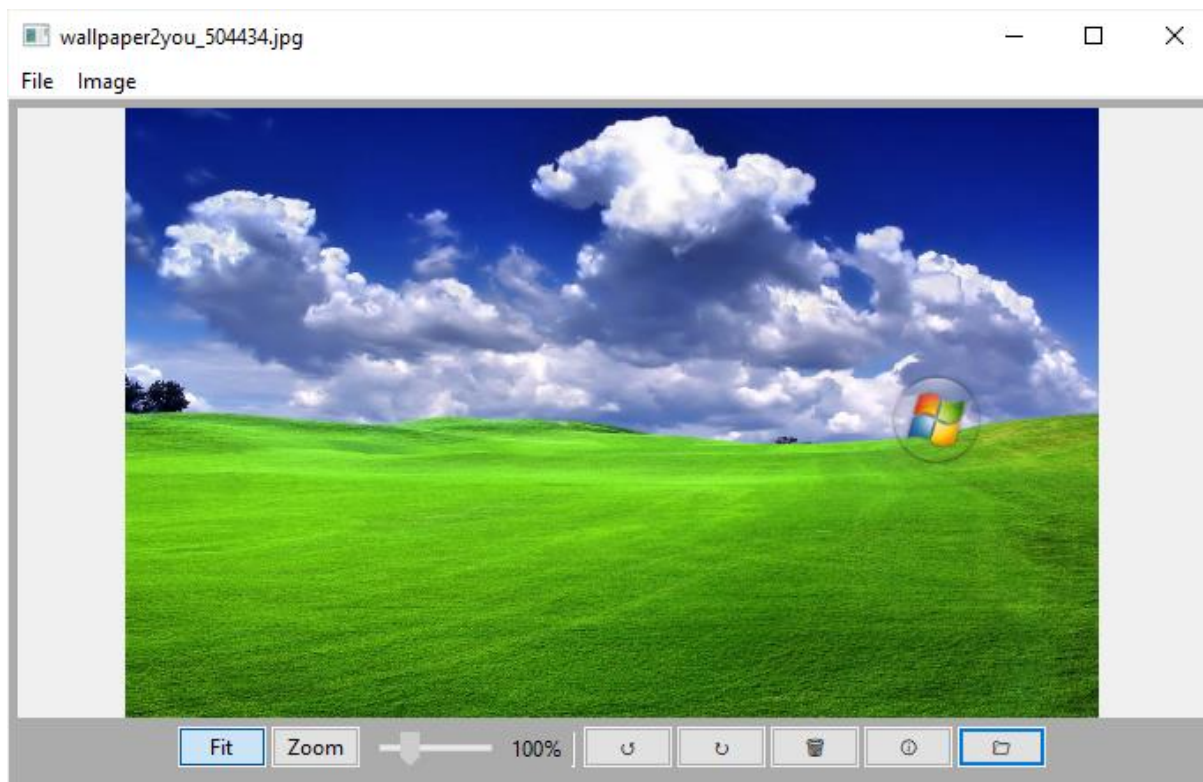


Rysunek 9. Natywne okno drukowania systemu Windows

Podobnie jak w przypadku obsługi bazy danych, funkcja drukowania pokazuje, że Qt udostępnia rozbudowane możliwości w ramach jednego, spójnego interfejsu programistycznego, pełniąc rolę warstwy pośredniczącej pomiędzy aplikacją a usługami systemu operacyjnego.

5.3. Implementacja wxWidgets

Niniejszy podrozdział omawia realizację prototypu w oparciu o bibliotekę wxWidgets. Przedstawiono w nim sposób budowy okna głównego oraz kluczowe rozwiązania odróżniające tę implementację od innych bibliotek. Rysunek 10 przedstawia okno główne prototypu zrealizowanego w bibliotece wxWidgets.



Rysunek 10. Okno główne prototypu wxWidgets uruchomionego w systemie Windows

5.3.1 Okno główne

Okno główne aplikacji reprezentuje klasa `MainFrame`, dziedzicząca po `wxFrame`. W odróżnieniu od wersji opartej na Qt, gdzie układ kontrolki opisano w osobnym pliku projektu interfejsu, w wxWidgets cały interfejs budowany jest w C++, w konstruktorze klasy. Poszczególne kontrolki tworzone są bezpośrednio w kodzie, a następnie dodawane do okna i wiązane z odpowiednimi metodami obsługi zdarzeń. Takie podejście ściślej łączy opis wyglądu z logiką aplikacji, lecz daje pełną kontrolę nad sposobem konstruowania interfejsu bez konieczności korzystania z dodatkowych narzędzi.

Rozmieszczeniem kontrolki zarządzają obiekty rozmieszczające (*sizers*), pełniące rolę analogiczną do menedżerów układu z biblioteki Qt. Samodzielnie wyznaczają one położenie i rozmiar elementów oraz dostosowują je do zmian wielkości okna, dzięki czemu interfejs zachowuje poprawny wygląd niezależnie od rozdzielczości ekranu czy ręcznego skalowania. Listing 9 przedstawia użycie obiektu rozmieszczającego. Główny obszar okna tworzy pionowy obiekt `wxBoxSizer`, w którym umieszczono panel prezentujący obraz oraz poziomy układ z przyciskami sterującymi.

Listing 9. Użycie obiektów rozmieszczających w wxWidgets

```
wxBoxSizer *buttonSizer = new wxBoxSizer(wxHORIZONTAL);
buttonSizer->Add(buttonFit, 0, wxRIGHT, 3);
buttonSizer->Add(buttonZoom, 0, wxRIGHT, 3);
buttonSizer->Add(sliderZoom, 0, wxRIGHT | wxALIGN_CENTER_VERTICAL,
3);
buttonSizer->Add(labelZoom, 0, wxRIGHT | wxALIGN_CENTER_VERTICAL,
5);
```

Pasek narzędzi zawiera komplet operacji dostępnych dla wyświetlanego obrazu. Znajdują się na nim dwa przyciski przełączane (`wxToggleButton`) odpowiadające trybom skalowania, suwak powiększenia (`wxSlider`) wraz z etykietą procentową, a także przyciski obrotu w lewo i w prawo, usunięcia pliku, wyświetlenia informacji o obrazie oraz otwarcia nowego pliku.

Obsługę zdarzeń powiązano z kontrolkami za pomocą metody `Bind`, wiążącej konkretny typ zdarzenia, na przykład naciśnięcie przycisku, z metodą klasy `MainFrame`. Operacje rzadziej używane udostępniono w pasku menu, tworzonym również w kodzie. Składa się on z dwóch menu: `File`, zawierającego polecenia drukowania, zapisu pod nową nazwą, otwarcia pliku w eksploratorze systemowym oraz ostatnio otwarte pliki, a także `Image` z korekcją jasności i kontrastu. Część poleceń otrzymała skróty klawiszowe, na przykład `Ctrl+P` dla drukowania. Do menu `File` dołączono również podmenu *Recent Files* z listą ostatnio otwieranych plików.

Sam obraz prezentuje obiekt autorskiej klasy `ImagePanel`, tworzony dynamicznie w konstruktorze okna i dodawany do głównego układu. Rozwiązanie to oddziela kontrolkę odpowiadającą za wyświetlanie i skalowanie obrazu od pozostałych elementów interfejsu, co porządkuje strukturę kodu i ułatwia zarządzanie sposobem prezentacji obrazu.

5.3.2 Przetwarzanie i wyświetlanie obrazów

W prototypie wczytany obraz przechowywany jest w obiekcie `wxImage`, czyli w niezależnej od platformy reprezentacji umieszczonej w pamięci operacyjnej, udostępniającej bezpośredni dostęp do poszczególnych składowych barwnych. To na niej wykonywane są wszystkie operacje przetwarzania zastosowane w prototypie: korekcja jasności i kontrastu, obrót oraz skalowanie. Wynik przetwarzania zamieniany jest następnie na obiekt `wxBitmap`, opakowujący natywny format obrazu danej platformy, zoptymalizowany pod kątem szybkiego wyświetlania. Z tego powodu sam mechanizm wyświetlania operuje wyłącznie na obiekcie `wxBitmap`. Jak wskazuje dokumentacja, taki podział ułatwia korzystanie ze wspólnego kodu dla operacji niezależnych od platformy oraz z kodu zależnego od platformy tam, gdzie ważna jest wydajność lub zgodność [27]. Rozwiązanie to jest odpowiednikiem podziału na klasy `QImage` oraz `QPixmap` z wersji opartej na Qt. Wszystkie operacje wykonywane są na kopii obrazu znajdującej się w pamięci, bez modyfikowania pliku źródłowego na dysku.

Za prezentację obrazu odpowiada autorska klasa `ImagePanel`, dziedzicząca po `wxScrolledWindow`, dzięki czemu zyskuje gotową obsługę przewijania wykorzystywaną w trybie powiększenia. W przeciwieństwie do standardowych kontroltek, panel rysowany jest samodzielnie: jego zawartość wyznacza metoda obsługująca zdarzenie rysowania (`OnPaint`), wywoływana automatycznie przy każdej zmianie wyglądu okna. Rysowanie odbywa się na buforowanym kontekście graficznym (`wxAutoBufferedPaintDC`), co eliminuje migotanie obrazu podczas przerysowywania. Klasa przechowuje oryginalny obraz oraz udostępnia dwa tryby skalowania określone wyliczeniem (*enum*) `ScaleMode`: dopasowanie do okna oraz powiększenie o zadany procent. Listing 10 przedstawia ten proces. Wybór trybu następuje za pomocą przycisków paska narzędzi, a właściwe przeliczenie rozmiaru obrazu realizuje metoda `UpdateImage`.

Listing 10. Funkcja rysująca obraz w wariancie wxWidgets

```
void ImagePanel::OnPaint(wxPaintEvent &event)
{
    wxAutoBufferedPaintDC dc(this);
    DoPrepareDC(dc);
    dc.SetBackground(wxBrush(GetBackgroundColour()));
    dc.Clear();

    if(scaledBitmap.IsOk())
    {
        if(scaleMode == ScaleMode::Fit)
        {
            const int x = (GetClientSize().GetWidth() -
scaledBitmap.GetWidth()) / 2;
            const int y = (GetClientSize().GetHeight() -
scaledBitmap.GetHeight()) / 2;
            dc.DrawBitmap(scaledBitmap, x, y, false);
        }
        else
        {
            int vw, vh;
            GetVirtualSize(&vw, &vh);
            const int x = (vw - scaledBitmap.GetWidth()) / 2;
            const int y = (vh - scaledBitmap.GetHeight()) / 2;
            dc.DrawBitmap(scaledBitmap, x > 0 ? x : 0, y > 0 ?
y : 0, false);
        }
    }
}
```

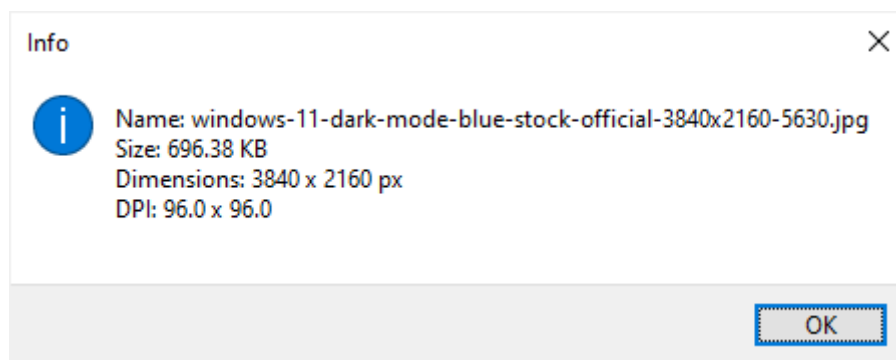
W trybie dopasowania obraz skalowany jest do rozmiaru obszaru roboczego panelu z zachowaniem proporcji, wyznaczanych jako mniejszy ze współczynników skalowania w poziomie i w pionie, co zapobiega zniekształceniu obrazu. Skalowanie wykonuje metoda `Scale` z opcją `wxIMAGE_QUALITY_BILINEAR`, zapewniającą wygładzenie obrazu. Tak przygotowany obraz wyśrodkowywany jest w obszarze panelu. Aby obraz dostosowywał się do zmian wielkości okna, panel reaguje na zdarzenie zmiany rozmiaru (`OnResize`), ponownie dopasowując obraz w trybie dopasowania. W trybie powiększenia rozmiar obrazu wyznaczany jest na podstawie wartości procentowej pobranej z suwaka. Panelowi nadawany jest wówczas rozmiar wirtualny (*virtual size*) odpowiadający powiększonemu obrazowi, co uruchamia mechanizm przewijania odziedziczony po `wxScrolledWindow` i pozwala oglądać fragmenty obrazu wykraczające poza widoczny obszar.

Obrót obrazu realizują metody wywołujące funkcję `Rotate90` na obiekcie `wxImage`, zmieniającą orientację o wielokrotność dziewięćdziesięciu stopni, po czym następuje ponowne przeliczenie i przerysowanie zawartości panelu.

Podobnie do implementacji Qt, parametry jasności i kontrastu ustawiane są w osobnym oknie dialogowym, otwieranym z menu `Image`. Okno to zawiera dwa suwaki o zakresie od -100 do +100, których przesunięcie natychmiast aktualizuje podgląd obrazu, dzięki powiązaniu zdarzenia suwaka z wyrażeniem lambda modyfikującym ustawienia panelu. Zatwierdzenie okna utrwała wprowadzone

ustawienia, natomiast jego anulowanie przywraca wartości sprzed otwarcia i odtwarza pierwotny wygląd obrazu.

Informacje o wyświetlanym obrazie, takie jak nazwa pliku, rozmiar, wymiary oraz rozdzielczość, prezentowane są w prostym oknie komunikatu (`wxMessageBox`). W odróżnieniu od wersji opartej na Qt, okno to nie zawiera histogramu rozkładu składowych barwnych. Standardowa biblioteka `wxWidgets` nie dostarcza bowiem wbudowanego modułu wykresów, odpowiadającego wykorzystanemu wcześniej modułowi Qt Charts. Wizualizacja histogramu wymagałaby sięgnięcia po zewnętrzną bibliotekę lub samodzielnego rysowania wykresu, dlatego w tej wersji prototypu ograniczono się do tekstowej prezentacji metadanych. Różnica ta dobrze ilustruje odmienny zakres obu bibliotek. `wxWidgets` głównie skupia się na modułach związanych z interfejsem użytkownika, podczas gdy Qt znacznie wykracza poza ten obszar. Rysunek 11 przedstawia okno komunikatu `wxWidgets` z metadanymi obrazu.



Rysunek 11. Standardowe okno komunikatu `wxWidgets` prezentujące metadane obrazu

5.3.3 Baza danych SQLite3

W odróżnieniu od biblioteki Qt, która dostarcza kompletny moduł bazodanowy, biblioteka `wxWidgets` koncentruje się na warstwie interfejsu graficznego i nie udostępnia własnego mechanizmu obsługi baz danych. Z tego powodu w prototypie wykorzystano bezpośrednio bibliotekę SQLite3 wraz z jej interfejsem programistycznym w języku C, dołączoną jako zewnętrzna zależność za pośrednictwem menedżera pakietów Conan.

Funkcjonalność oparta na bazie danych jest identyczna jak w przypadku modułu Qt i polega na zapamiętywaniu ostatnio otwieranych plików graficznych. Po ponownym uruchomieniu aplikacji użytkownik uzyskuje dostęp do listy wcześniej przeglądanych obrazów, dostępnej z poziomu podmenu w głównym menu programu. Dzięki zastosowaniu tego samego mechanizmu składowania w obu wariantach prototypu możliwe staje się porównanie nie samych możliwości bazy danych, lecz sposobu jej integracji z każdą z bibliotek.

Plik bazy danych umieszczany jest w systemowym katalogu danych aplikacji, którego ścieżkę uzyskuje się przy użyciu `wxStandardPaths`. Rozwiązanie to zapewnia zgodność z konwencjami obu wspieranych systemów operacyjnych, podobnie jak w wariantcie wykorzystującym Qt. Podczas inicjalizacji tworzona jest tabela `recent_files`, przechowująca ścieżkę pliku oraz znacznik czasu ostatniego otwarcia, przy czym kolumna ze ścieżką oznaczona jest jako unikalna. Każde otwarcie obrazu powoduje zapisanie jego ścieżki wraz ze znacznikiem czasu, jednak ponowne otwarcie znanego już pliku jedynie aktualizuje ten znacznik, zamiast tworzyć duplikat. Przy budowie podmenu pobieranych jest dziesięć najnowszych pozycji, uporządkowanych malejąco według czasu otwarcia, a każdej z nich przypisywana jest osobna akcja umożliwiająca ponowne wczytanie obrazu.

Istotną różnicą w stosunku do Qt jest poziom abstrakcji, na którym programista wykonuje operacje bazodanowe. Korzystając z biblioteki SQLite3 bezpośrednio, należy ręcznie zarządzać kolejnymi etapami obsługi zapytań: przygotowaniem instrukcji za pomocą funkcji

sqlite3_prepare_v2, dowiązaniem parametrów (sqlite3_bind_text), wykonaniem zapytania (sqlite3_step) oraz zwolnieniem zasobów (sqlite3_finalize). Listing 11 przedstawia ten proces. Konieczne jest również samodzielne zwolnienie połączenia z bazą przy zamykaniu aplikacji. Podejście to daje pełną kontrolę nad operacjami, lecz wymaga większej ilości kodu i staranności w zarządzaniu zasobami. Przedstawione rozwiązanie potwierdza, że wxWidgets, w przeciwieństwie do Qt, nie pełni roli wszechstronnego ekosystemu, lecz biblioteki interfejsu graficznego.

Listing 11. Usunięcie rekordu z użyciem biblioteki SQLite3

```
std::string sql = "DELETE FROM recent_files WHERE file_path = ?";
sqlite3_stmt *stmt;
if(sqlite3_prepare_v2(db, sql.c_str(), -1, &stmt, nullptr) ==
    SQLITE_OK)
{
    sqlite3_bind_text(stmt, 1, filePath.utf8_str(), -1,
        SQLITE_TRANSIENT);
    sqlite3_step(stmt);
    sqlite3_finalize(stmt);
}
```

5.3.4 Drukowanie

Podobnie jak w wariantcie wykorzystującym Qt, prototyp oparty na wxWidgets udostępnia funkcję wydruku przeglądanego obrazu. Obsługa drukarek wbudowana jest tu w samą bibliotekę, bez konieczności dołączania osobnego modułu, a odpowiednie klasy dostępne są po włączeniu nagłówka `wx/print.h`.

Centralnym elementem tego mechanizmu jest klasa `wxPrintout`, reprezentująca dokument przeznaczony do wydruku. W prototypie utworzono jej klasę pochodną `ImagePrintout`, która przesłania (*override*) metodę `OnPrintPage` odpowiedzialną za narysowanie pojedynczej strony. Rysowanie odbywa się za pośrednictwem kontekstu urządzenia `wxDC`, czyli tej samej abstrakcji, której wxWidgets używa do rysowania w obszarze okna. Z punktu widzenia kodu drukarka traktowana jest zatem identycznie jak inne urządzenia wyjściowe, dzięki czemu zawartość strony przygotowywana jest tym samym mechanizmem rysującym, co prezentacja w interfejsie. Obraz skalowany jest do rozmiaru strony z zachowaniem oryginalnych proporcji, co zapobiega jego zniekształceniu, a następnie wyśrodkowywany na arkuszu.

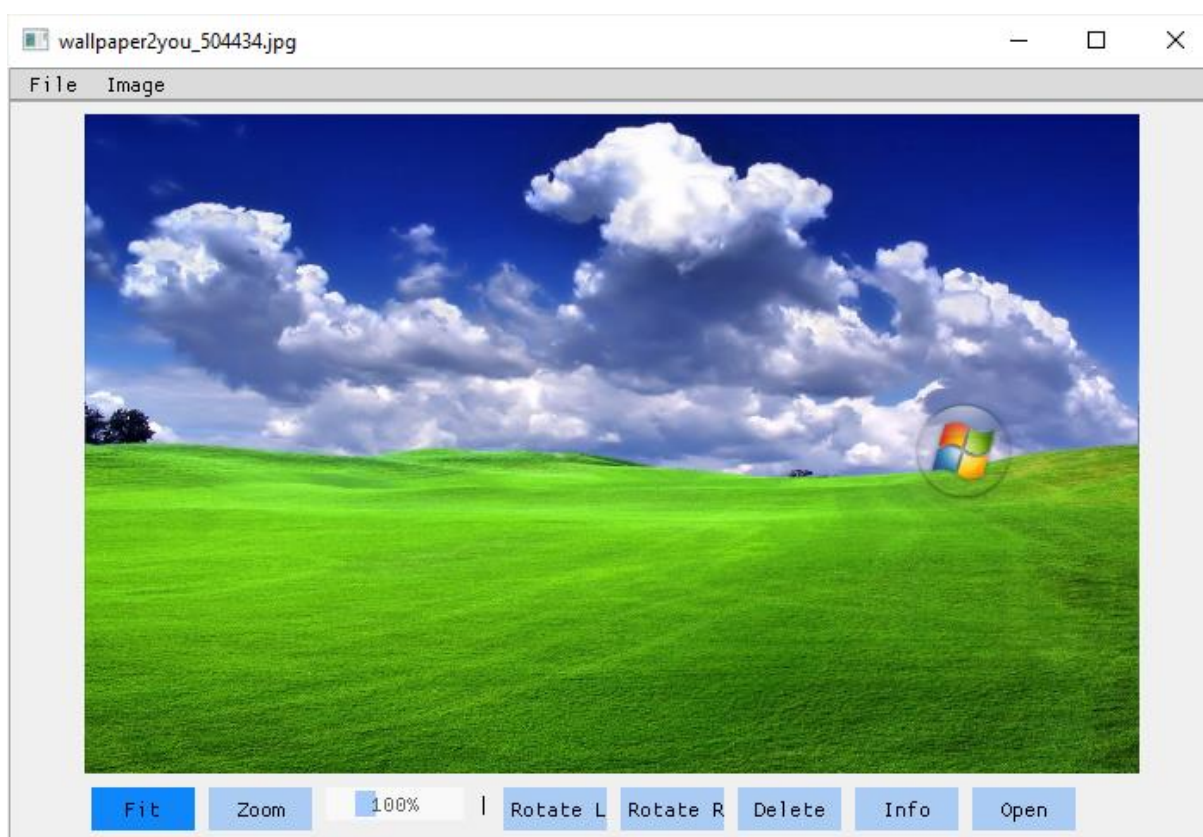
Samym procesem wydruku steruje klasa `wxPrinter`, której obiekt po wywołaniu metody `Print` wyświetla systemowe okno dialogowe drukowania i kieruje gotowy dokument do wybranej drukarki. Pojawienie się natywnego okna nie stanowi zaskoczenia, ponieważ jednym z podstawowych założeń biblioteki wxWidgets jest korzystanie z natywnych widżetów wszędzie tam, gdzie jest to możliwe. Dzięki temu okno dialogowe wydruku wygląda i zachowuje się tak samo jak w innych aplikacjach danego systemu operacyjnego, co zapewnia użytkownikowi znajome i spójne środowisko pracy.

Tak jak w przypadku Qt, biblioteka wxWidgets nie komunikuje się bezpośrednio z urządzeniem, lecz przygotowuje zawartość strony i przekazuje ją do natywnego podsystemu drukowania danego systemu operacyjnego, który odpowiada za obsługę sterownika oraz wysłanie zadania do drukarki. Różnica polega na poziomie, na którym programista definiuje zadanie wydruku: w wxWidgets wymagane jest utworzenie własnej klasy pochodnej od `wxPrintout` i ręczne rozmieszczenie treści na stronie za pomocą kontekstu urządzenia, podczas gdy Qt opiera ten sam proces na klasie `QPrinter`.

realizującej interfejs urządzenia rysującego. Oba rozwiązania prowadzą do zbliżonego efektu, korzystając z natywnych mechanizmów systemu, lecz odzwierciedlają odmienny sposób organizacji kodu w obu bibliotekach.

5.4. Implementacja Dear ImGui

Implementacja prototypu oparta na bibliotece Dear ImGui różni się zasadniczo od pozostałych wariantów. Wynika to z odmiennej natury tej biblioteki, działającej w trybie natychmiastowym (*Immediate Mode*) i nieobejmującej własnej warstwy zarządzania oknami systemowymi ani komunikacji ze sprzętem graficznym. Poniższe podrozdziały omawiają kolejno sposób uruchomienia środowiska graficznego, budowę interfejsu oraz realizację poszczególnych funkcji prototypu w tym wariantcie. Rysunek 12 przedstawia okno główne prototypu zrealizowanego w bibliotece Dear ImGui.



Rysunek 12. Okno główne prototypu Dear ImGui uruchomionego w systemie Windows

5.4.1 Inicjalizacja z SDL2 i OpenGL

W przeciwieństwie do pozostałych bibliotek, które stanowią kompletne środowiska zarządzające cyklem życia okna i jego zawartością, Dear ImGui nie tworzy okna systemowego, nie odbiera zdarzeń wejścia i nie komunikuje się z kartą graficzną. Aby przekształcić generowane przez bibliotekę polecenia rysowania w widoczny obraz, konieczne jest powiązanie biblioteki z dwoma zewnętrznymi komponentami: modułem platformowym, odpowiedzialnym za utworzenie okna oraz obsługę klawiatury i myszy, a także modułem renderującym, kierującym przygotowane polecenia do interfejsu graficznego. W prototypie rolę modułu platformowego pełni biblioteka SDL2, natomiast renderowanie realizowane jest przy użyciu OpenGL.

Potrzeba samodzielnego przygotowania tego środowiska wynika bezpośrednio z przeznaczenia biblioteki Dear ImGui. Powstała ona z myślą o zastosowaniach takich jak narzędzia deweloperskie, edytory czy interfejsy diagnostyczne osadzone w istniejących silnikach graficznych, na przykład w grach komputerowych. W typowym scenariuszu okno aplikacji oraz kontekst renderowania są już utworzone i obsługiwane przez silnik, a biblioteka jedynie dołącza swój interfejs. W takim przypadku opisywany etap inicjalizacji w praktyce nie istnieje, gdyż całość infrastruktury graficznej zapewnia środowisko, w którym ImGui pracuje. W prototypie będącym samodzielną aplikacją infrastrukturę tę trzeba zbudować ręcznie, co dobrze ilustruje minimalistyczne założenia biblioteki.

Proces uruchomienia podzielono na dwa etapy. Pierwszy z nich, realizowany w metodzie `InitSDL`, inicjalizuje bibliotekę SDL2, ustala wersję interfejsu OpenGL oraz języka cieniowania GLSL odpowiednią dla danej platformy, a następnie tworzy okno z dołączonym kontekstem renderowania. Włączana jest przy tym, za pomocą `SDL_GL_SetSwapInterval(1)` synchronizacja pionowa VSync ograniczająca liczbę klatek do częstotliwości odświeżania ekranu. Drugi etap, zawarty w metodzie `InitImGui`, tworzy kontekst samej biblioteki, konfiguruje jej parametry oraz styl wizualny, a następnie inicjalizuje wspomniane moduły powiązania za pomocą funkcji `ImGui_ImplSDL2_InitForOpenGL` oraz `ImGui_ImplOpenGL3_Init`. Dopiero po pomyślnym zakończeniu obu etapów możliwe staje się rozpoczęcie głównej pętli aplikacji, w której odbywa się przetwarzanie zdarzeń oraz budowanie i wyświetlanie interfejsu.

Przedstawiony sposób inicjalizacji, a także ogólna struktura prototypu, wzorowane są na oficjalnych przykładach dostarczanych wraz z biblioteką. Dear ImGui udostępnia bowiem gotowy przykład dla każdej z obsługiwanych kombinacji modułu platformowego i renderującego, co upraszcza poprawne przygotowanie środowiska graficznego.

5.4.2 Okno główne

Okno główne w wariantcie opartym na Dear ImGui budowane jest w sposób zasadniczo odmienny od pozostałych bibliotek. Wynika to z natychmiastowego trybu pracy: interfejs nie istnieje jako trwały zbiór obiektów kontrolek, lecz odtwarzany jest od nowa w każdej klatce. Kod głównego okna interfejsu zawarto w metodzie `DrawMainWindow`, wywoływanej cyklicznie w głównej pętli aplikacji. Nie występuje tu ani osobny plik projektu interfejsu, jak w Qt, ani trwałe obiekty kontrolek tworzone w konstruktorze, jak w `wxWidgets`. Zamiast tego każde wywołanie funkcji biblioteki, na przykład `ImGui::Button`, jednocześnie wyświetla dany element i zwraca informację o jego stanie w bieżącej klatce, co pokazuje Listing 12.

Listing 12. Element rysowany i zwracający stan w jednym wywołaniu

```
if(ImGui::Button("OK", ImVec2(120, 0)))
{
    showInfoPopup = false;
    ImGui::CloseCurrentPopup();
}
```

Sam obszar roboczy to pojedyncze okno biblioteki, tworzone wywołaniem `ImGui::Begin` i rozciągnięte na całą powierzchnię okna systemowego poniżej paska menu. Okno ma zestaw znaczników, dzięki którym pozbawione jest własnej ramki i paska tytułu, sprawiając wrażenie jednolitego obszaru aplikacji. Jego położenie i rozmiar wyznaczone są ręcznie na podstawie aktualnych wymiarów okna systemowego oraz wysokości paska menu.

W przeciwieństwie do Qt oraz `wxWidgets`, biblioteka Dear ImGui nie udostępnia menedżerów układu ani obiektów rozmieszczających. Położenie elementów ustalone jest na zasadzie kursora

przesuwającego się w dół okna, a kolejne kontrolki umieszczane są jedna pod drugą, o ile nie zażąda się ich ułożenia w wierszu za pomocą funkcji `ImGui::SameLine`. Wyśrodkowanie obrazu czy paska przycisków wymaga zatem samodzielnego obliczenia odpowiednich przesunięć i ustawienia pozycji kursora funkcjami z rodziny `SetCursorPos`. Sam obraz prezentowany jest w osadzonym obszarze `ImGui::BeginChild`, który w trybie powiększenia uzupełniany jest o pasek przewijania, a rysowanie tekstury obrazu odbywa się bezpośrednio na liście poleceń rysowania za pomocą metody `AddImageQuad`. Rozwiązanie to pozwala swobodnie skalować i obracać obraz przy zachowaniu jego proporcji.

Pasek narzędzi umieszczony w dolnej części okna zawiera komplet operacji dostępnych dla wyświetlanego obrazu: dwa przyciski wyboru trybu skalowania, suwak powiększenia, a także przyciski obrotu w lewo i w prawo, usunięcia pliku, wyświetlenia informacji o obrazie oraz otwarcia nowego pliku. Ponieważ tryb natychmiastowy nie przechowuje stanu kontrolki pomiędzy klatkami, informacja o aktywnym trybie skalowania utrzymywana jest w osobnym polu aplikacji, a wyróżnienie wciśniętego przycisku realizowane jest przez ręczną zmianę jego koloru. Operacje rzadziej używane udostępniono w pasku menu tworzonym wywołaniem `ImGui::BeginMainMenuBar`, złożonym z menu `File`, zawierającego polecenia zapisu pod nową nazwą oraz otwarcia pliku w eksploratorze systemowym, a także `Image` z korekcją jasności i kontrastu.

Etykiety przycisków opisano krótkimi nazwami w języku angielskim, zamiast graficznych symboli. Wynika to z ograniczeń domyślnej czcionki dołączonej do biblioteki, która nie zawiera znaków tego typu. Wyświetlenie ikon wymagałoby dostarczenia i wczytania własnego zestawu czcionek. Stanowi to dobrą ilustrację minimalistycznego charakteru biblioteki `Dear ImGui`, która domyślnie dostarcza jedynie podstawowy zestaw funkcji GUI.

5.4.3 Wczytywanie obrazów biblioteką STB

W odróżnieniu od pozostałych porównywanych bibliotek, `Dear ImGui` nie udostępnia żadnego mechanizmu odczytu plików graficznych. Operuje jedynie na gotowych teksturach procesora graficznego (GPU). Dokumentacja biblioteki jednoznacznie wskazuje, że wczytanie pliku graficznego do tekstury procesora graficznego pozostaje poza jej zakresem i zależy od używanego interfejsu programistycznego grafiki [28]. Funkcje rysujące, takie jak `ImGui::Image` oraz `ImGuiDrawList::AddImage`, przyjmują wyłącznie uchwyt gotowej tekstury `ImTextureID`, a nie ścieżkę pliku ani surowe dane pikseli. Z tego powodu kod aplikacji jest odpowiedzialny za odczyt i zdekodowanie obrazu, a biblioteka odpowiada jedynie za wyświetlenie przygotowanej wcześniej tekstury.

Poradnik dołączony do dokumentacji `Dear ImGui` opisuje wczytywanie obrazów jako proces dwuetapowy: najpierw plik jest zdekodowany z dysku do pamięci operacyjnej przy użyciu biblioteki pomocniczej, a następnie uzyskane dane są przesyłane do pamięci karty graficznej funkcjami interfejsu graficznego [28]. W przedstawionych w nim przykładach do zdekodowania plików wykorzystywana jest biblioteka STB, co stanowiło dodatkowy argument za zastosowaniem jej również w niniejszym prototypie.

Tę lukę wypełnia zatem biblioteka STB, a dokładniej `stb_image` oraz `stb_image_write`. Biblioteka STB jest dostarczana w postaci pojedynczych plików nagłówkowych (*single-header*), bez zależności zewnętrznych, co dobrze odpowiada lekkiej i minimalistycznej filozofii samego `Dear ImGui`. W prototypie biblioteka została dołączona poprzez menedżer pakietów Conan, a jej implementacja jest włączana do jednostki kompilacji przez zdefiniowanie makr `STB_IMAGE_IMPLEMENTATION` oraz `STB_IMAGE_WRITE_IMPLEMENTATION`.

Wczytanie pliku graficznego sprowadza się do wywołania funkcji `stbi_load`, która dekoduje obraz do ciągłego bufora pikseli. Ostatni argument wymusza format czterokanałowy RGBA, dzięki czemu dalsze przetwarzanie oraz przesyłanie danych do tekstury przebiega jednolicie, niezależnie od pierwotnej liczby kanałów w pliku. Otrzymany bufor jest zachowywany w polu `originalData`, co

umożliwia późniejszą regulację jasności i kontrastu bez ponownego odczytu z dysku. Następnie dane są przesyłane do tekstury OpenGL funkcją `glTexImage2D`, a pamięć zwrócona przez bibliotekę STB jest zwalniana wywołaniem `stbi_image_free`. Uzyskany identyfikator tekstury `ImTextureID` jest przekazywany do Dear ImGui podczas rysowania. Cały proces wczytania pliku i utworzenia tekstury OpenGL przedstawia Listing 13.

Listing 13. Wczytanie obrazu biblioteką STB i utworzenie tekstury OpenGL

```
void ImGuiBackend::Image::LoadFile(const char *path)
{
    int channels;
    unsigned char *data = stbi_load(path, &width, &height,
&channels, 4);
    if(!data)
        return;

    filePath = path;
    rotation = 0;

    originalData.assign(data, data + width * height * 4);

    if(textureId)
        glDeleteTextures(1, &textureId);

    glGenTextures(1, &textureId);
    glBindTexture(GL_TEXTURE_2D, textureId);

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);

    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0,
GL_RGBA, GL_UNSIGNED_BYTE, data);
    stbi_image_free(data);
}
```

`stb_image_write` odpowiada za operację odwrotną, czyli zapis obrazu na dysk. W zależności od rozszerzenia pliku docelowego prototyp wywołuje funkcję `stbi_write_jpg`, `stbi_write_bmp` albo `stbi_write_png`.

Konieczność użycia dodatkowej biblioteki stanowi istotną różnicę w stosunku do pozostałych zestawów narzędzi. W przypadku Qt, `wxWidgets` oraz `GTK` odczyt i wyświetlenie obrazu realizują wbudowane klasy, podczas gdy w Dear ImGui programista odpowiada za pełny łańcuch przetwarzania.

5.4.4 Obracanie obrazów współrzędnymi UV

W przeciwieństwie do pozostałych porównywanych bibliotek, które obracają obraz wbudowanymi metodami obiektu obrazu, na przykład `wxImage::Rotate90` w `wxWidgets`, Dear ImGui nie udostępnia żadnej takiej funkcji. Z tego powodu obrót trzeba zrealizować na niższym poziomie, przez bezpośrednią manipulację współrzędnymi UV.

Współrzędne UV określają, który punkt tekstury jest pobrany dla danego wierzchołka rysowanego wielokąta. Wyrażane są w znormalizowanym zakresie od 0 do 1, niezależnie od

rozdzielczości obrazu w pikselach, gdzie para (0, 0) odpowiada lewemu górnemu narożnikowi tekstury, a (1, 1) prawemu dolnemu [28]. Współrzędne ekranowe decydują więc o tym, gdzie wielokąt jest narysowany, natomiast współrzędne UV o tym, który fragment tekstury zostanie na nim odwzorowany.

Prototyp używa metody `ImDrawList::AddImageQuad`, której podaje się cztery narożniki prostokąta na ekranie oraz cztery współrzędne UV mówiące, który róg tekstury trafia do którego narożnika. Narożniki ekranowe podawane są zawsze w tej samej kolejności, a obrót obrazu uzyskuje się przez przestawienie czterech współrzędnych UV, dzięki czemu na poszczególne rogi prostokąta trafiają inne fragmenty tekstury. Listing 14 przedstawia ten proces. Same wymiary prostokąta zmieniają się przy obrocie o wielokrotność kąta prostego, ponieważ kod zamienia wówczas szerokość i wysokość obrazu, co zachowuje właściwe proporcje.

Listing 14. Kod źródłowy obrotu przez współrzędne UV

```
ImVec2 uvs[4];
switch(image.rotation % 4)
{
case 0: // 0
    uvs[0] = ImVec2(0, 0);
    uvs[1] = ImVec2(1, 0);
    uvs[2] = ImVec2(1, 1);
    uvs[3] = ImVec2(0, 1);
    break;
case 1: // 90
    uvs[0] = ImVec2(0, 1);
    uvs[1] = ImVec2(0, 0);
    uvs[2] = ImVec2(1, 0);
    uvs[3] = ImVec2(1, 1);
    break;
case 2: // 180
    uvs[0] = ImVec2(1, 1);
    uvs[1] = ImVec2(0, 1);
    uvs[2] = ImVec2(0, 0);
    uvs[3] = ImVec2(1, 0);
    break;
case 3: // 270
    uvs[0] = ImVec2(1, 0);
    uvs[1] = ImVec2(1, 1);
    uvs[2] = ImVec2(0, 1);
    uvs[3] = ImVec2(0, 0);
    break;
}
```

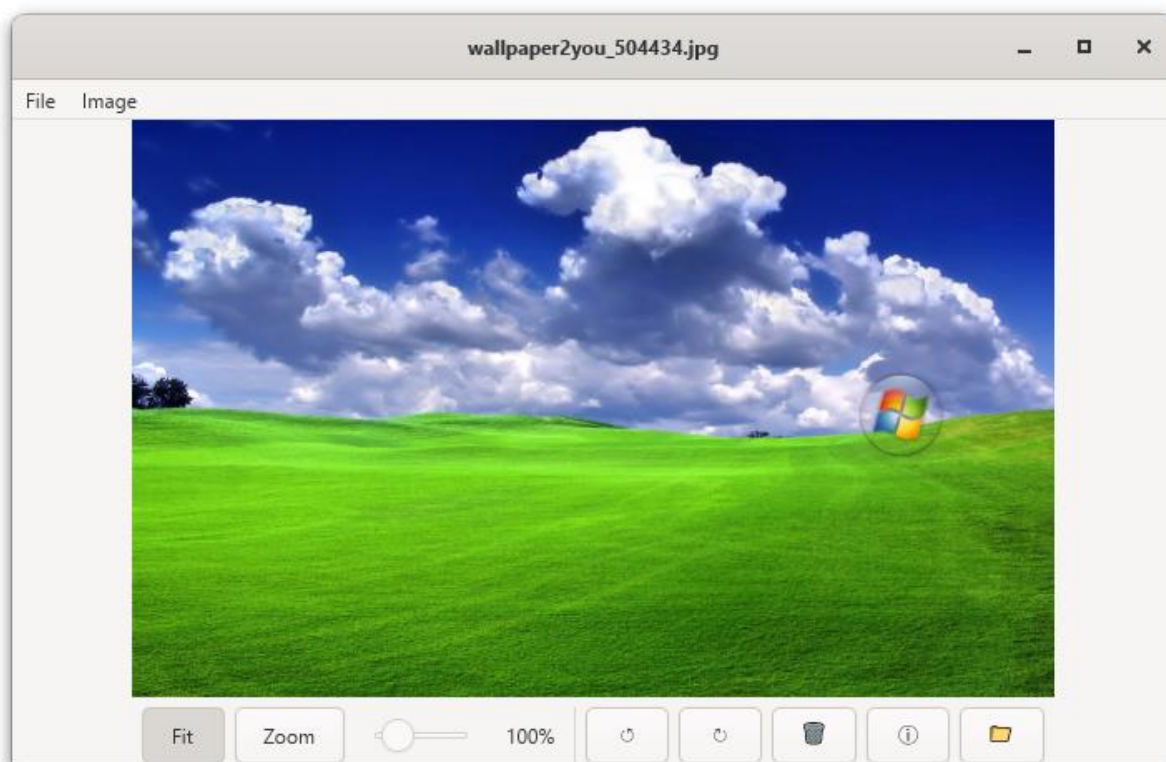
5.5. Implementacja GTK

Niniejszy podrozdział omawia realizację prototypu w oparciu o bibliotekę GTK w wersji 4. Przedstawiono w nim sposób budowy okna głównego oraz kluczowe rozwiązania wynikające z faktu, że GTK jest biblioteką napisaną w języku C.

Logikę aplikacji rozdzielono pomiędzy dwie klasy: `AppWindow`, odpowiadającą za okno główne oraz obsługę poszczególnych funkcji, a także pomocniczą klasę `ImageWidget`, przechowującą stan wczytanego obrazu oraz jego parametry, takie jak obrót, jasność i kontrast.

Punktem wejścia modułu jest obiekt `GtkApplication`, utworzony z flagą `G_APPLICATION_HANDLES_OPEN`, dzięki czemu aplikacja potrafi przyjąć ścieżkę do pliku przekazaną przy uruchomieniu. Z sygnałami `activate` oraz `open` powiązano funkcje zwrotne odpowiadające za wyświetlenie okna oraz otwarcie wskazanego obrazu.

Cechą charakterystyczną kodu opartego na GTK jest sposób przechowywania kontroltek. Wszystkie widżety, niezależnie od ich właściwego typu, trzymane są w polach wskaźnikowych typu `GtkWidget`, na przykład obszar przewijania, suwak czy przyciski. Wynika to z hierarchii typów biblioteki, w której poszczególne kontrolki, takie jak `GtkScrolledWindow`, `GtkScale` czy `GtkToggleButton`, wywodzą się ze wspólnego typu bazowego `GtkWidget`. Operacje wspólne dla wszystkich widżetów, na przykład ustawianie rozszerzania czy dodawanie do pojemnika, wykonuje się wprost na takim wskaźniku. Gdy potrzebna jest funkcja właściwa konkretnemu typowi, wskaźnik rzutuje się na ten typ za pomocą makr, na przykład `GTK_SCROLLED_WINDOW` lub `GTK_TOGGLE_BUTTON`, które jednocześnie sprawdzają poprawność typu w czasie wykonania. Rysunek 13 przedstawia okno główne prototypu zrealizowanego w bibliotece GTK.



Rysunek 13. Okno główne prototypu GTK uruchomionego w systemie Windows

5.5.1 Okno główne

Okno główne aplikacji reprezentuje klasa `AppWindow`, opakowująca widżet okna aplikacji `GtkApplicationWindow` tworzony funkcją `gtk_application_window_new`. Podobnie jak w wersji opartej na `wxWidgets`, cały interfejs budowany jest w C++, w konstruktorze klasy. Kontrolki tworzone są bezpośrednio w kodzie, a następnie dodawane do okna i wiązane z odpowiednimi funkcjami

obsługi zdarzeń. Takie podejście daje pełną kontrolę nad sposobem konstruowania interfejsu bez konieczności korzystania z dodatkowych narzędzi.

Ponieważ GTK jest biblioteką napisaną w języku C, obsługę zdarzeń realizuje się odmiennie niż w bibliotekach C++. Kontrolki łączy się z logiką aplikacji za pomocą funkcji `g_signal_connect`, wiążącej nazwany sygnał, na przykład `clicked` dla przycisku, z wolną funkcją zwrotną. Aby zachować obiektową strukturę kodu, każda taka funkcja jest statyczną metodą klasy `AppWindow`, która odbiera wskaźnik na instancję okna przekazany jako parametr `userData` i przekierowuje wywołanie do odpowiedniej metody. Dzięki temu właściwa logika obsługi zdarzeń pozostaje zamknięta w metodach klasy.

Rozmieszczeniem kontrolki zarządzają pojemniki tworzone przez `gtk_box_new`, pełniące rolę analogiczną do menedżerów układu z biblioteki Qt. Samodzielnie wyznaczają one położenie i rozmiar elementów oraz dostosowują je do zmian wielkości okna, dzięki czemu interfejs zachowuje poprawny wygląd niezależnie od rozdzielczości ekranu czy ręcznego skalowania. Głównym elementem okna jest pionowy pojemnik, w którym umieszczono obszar przewijania `scrolledWindow`, w którym wyświetlany jest obraz, oraz poziomy pojemnik z przyciskami sterującymi.

Pasek narzędzi zawiera komplet operacji dostępnych dla wyświetlanego obrazu. Znajdują się na nim dwa przyciski przełączane odpowiadające trybom skalowania, suwak powiększenia wraz z etykietą procentową, a także przyciski obrotu w lewo i w prawo, usunięcia pliku, wyświetlenia informacji o obrazie oraz otwarcia nowego pliku. Aby spośród dwóch trybów skalowania aktywny pozostawał zawsze tylko jeden, w funkcjach obsługujących oba przyciski czasowo blokowane są ich sygnały podczas wzajemnego przełączania stanu.

Podobnie jak w pozostałych implementacjach, operacje rzadziej używane udostępniono w pasku menu. Strukturę menu opisuje obiekt `GMenu`, a poszczególne polecenia reprezentują obiekty `GSimpleAction` powiązane z funkcjami obsługi i zarejestrowane w oknie. Pasek składa się z dwóch menu: `File`, zawierającego polecenia zapisu pod nową nazwą oraz otwarcia pliku w eksploratorze systemowym, a także `Image` z korekcją jasności i kontrastu. Poleceniu zapisu przypisano skrót klawiszowy `Ctrl+Shift+S` za pomocą funkcji `gtk_application_set_accels_for_action`.

5.5.2 Sygnały GObject

Powiązanie kontrolki z logiką aplikacji opiera się w GTK na mechanizmie sygnałów, pochodzącym z systemu obiektowego `GObject`, na którym zbudowano całą bibliotekę. Sygnał to nazwane zdarzenie emitowane przez obiekt, łączące zdarzenia specyficzne dla aplikacji z dowolną liczbą odbiorców. Mechanizm ten pełni rolę zbliżoną do sygnałów i slotów z biblioteki Qt, lecz działa na poziomie języka C i nie korzysta z mechanizmów obiektowych C++.

Połączenie sygnału z funkcją obsługi realizuje funkcja `g_signal_connect`, która rejestruje funkcję zwrotną wywoływaną wyłącznie wtedy, gdy sygnał jest emitowany na konkretnej instancji obiektu. Jako ostatni argument przekazuje się wskaźnik na dowolne dane użytkownika, który biblioteka przekazuje następnie do funkcji obsługi jako jej ostatni parametr, bez zmiany jego postaci. Pozwala to udostępnić funkcji zwrotnej potrzebny kontekst. W prototypie wykorzystano ten parametr do przeniesienia wskaźnika na instancję okna. Każda funkcja obsługi jest statyczną metodą klasy `AppWindow`, która rzutuje otrzymany wskaźnik `userData` z powrotem na typ `AppWindow` i przekierowuje wywołanie do właściwej metody. Przykład takiego połączenia wraz z odpowiadającą mu metodą zwrotną przedstawia Listing 15. Dzięki temu, mimo że biblioteka operuje na wolnych funkcjach języka C, cała logika obsługi zdarzeń pozostaje zamknięta w obiektowej strukturze klasy. W ten sposób powiązано między innymi sygnały `clicked` przycisków, `toggled` przycisków przełączanych oraz response okien dialogowych.

Listing 15. Połączenie sygnału GTK z metodą klasy przez statyczną funkcję zwrrotną

```
g_signal_connect(buttonRotateLeft, "clicked",
G_CALLBACK(RotateLeftCallback), this);

void AppWindow::RotateLeftCallback(GtkButton *, gpointer userData)
{
    static_cast<AppWindow *>(userData)->OnRotateLeft();
}
```

W przypadku najprostszych zdarzeń, niewymagających dostępu do stanu okna, zastosowano alternatywnie wyrażenia lambda bez przechwytywania, konwertowane na wskaźniki funkcji. Rozwiązanie to wykorzystano na przykład przy zamykaniu okna informacji o pliku.

5.5.3 Rysowanie z Cairo i GdkPixbuf

Prezentacja obrazu w implementacji GTK różni się zasadniczo od pozostałych modułów graficznych. Zamiast gotowej kontrolki wyświetlającej mapę bitową zastosowano obszar rysowania `GtkDrawingArea`, czyli pustą kontrolkę, której zawartość aplikacja wyznacza samodzielnie poprzez funkcję rysującą. Funkcję tę przypisuje się wywołaniem `gtk_drawing_area_set_draw_func`, a biblioteka wywołuje ją zawsze, gdy obszar wymaga odświeżenia. Rysowanie odbywa się przy użyciu biblioteki `Cairo`, która jest zintegrowana bezpośrednio w GDK. Odświeżenie obszaru następuje samoczynnie przy pierwszym wyświetleniu kontrolki, a w pozostałych przypadkach wymuszane jest funkcją `gtk_widget_queue_draw`, wywoływana po każdej operacji zmieniającej obraz, takiej jak obrót, zmiana powiększenia czy korekcja jasności.

Dane obrazu przechowuje obiekt `GdkPixbuf`, stanowiący mapę bitową umieszczoną w pamięci wraz z informacją o szerokości, wysokości, liczbie kanałów i obecności kanału przezroczystości. Oryginalny obraz wczytywany jest z pliku do pola `originalPixbuf` klasy `ImageWidget` i pozostaje niezmienny przez cały czas pracy, dzięki czemu kolejne przekształcenia można nakładać wielokrotnie bez utraty jakości. Każde rysowanie operuje na osobnej kopii.

Gotową mapę bitową rysuje się na obszarze za pośrednictwem biblioteki `Cairo`. Funkcja `gdk_cairo_set_source_pixbuf` ustawia gotowy obraz jako wzorzec źródłowy kontekstu rysowania w wyliczonym, wyśrodkowanym położeniu, a wywołanie `cairo_paint` nanosi go na powierzchnię. Po zakończeniu rysowania pomocnicze mapy bitowe są zwalniane, co zapobiega wyciekom pamięci. W trybie powiększenia dodatkowo aktualizowany jest żądany rozmiar obszaru rysowania, dzięki czemu obraz większy od okna można przewijać w obrębie obszaru przewijania.

6. Testy i pomiary

Niniejszy rozdział przedstawia wyniki testów i pomiarów przeprowadzonych dla czterech implementacji prototypu, zrealizowanych w bibliotekach Qt, wxWidgets, Dear ImGui oraz GTK. Celem pomiarów jest dostarczenie obiektywnych danych uzupełniających jakościową ocenę poszczególnych bibliotek i umożliwiającą ich porównanie.

Kolejne podrozdziały omawiają najpierw środowisko, w którym przeprowadzono testy, a następnie wyniki w poszczególnych obszarach: wydajność renderowania, zużycie pamięci operacyjnej i procesora, rozmiar plików binarnych wraz z zależnościami oraz czas kompilacji. Dalsza część rozdziału dotyczy aspektów jakościowych: przenoszalności między systemami operacyjnymi, wygody programowania oraz jakości dokumentacji. Rozdział zamyka zestawienie wyników oraz rekomendacje doboru biblioteki w zależności od charakteru projektu.

6.1. Środowisko testowe

Wiarygodne porównanie wymaga, aby wszystkie pomiary ilościowe wykonano w jednym, niezmiennym środowisku. Dzięki temu zaobserwowane różnice wynikają z właściwości samych bibliotek, a nie z odmiennej konfiguracji sprzętu czy oprogramowania. Wszystkie cztery warianty prototypu uruchamiano na tej samej maszynie, kompilowano tym samym zestawem narzędzi i mierzono w identycznych warunkach. Dodatkowym czynnikiem zapewniającym porównywalność jest wspólna logika aplikacji.

Pomiary ilościowe, obejmujące wydajność, zużycie zasobów, rozmiar plików binarnych oraz czas kompilacji przeprowadzono w systemie Windows. Przenoszalność między systemami operacyjnymi zweryfikowano sprawdzając możliwość zbudowania i uruchomienia każdego wariantu w systemie Linux. Weryfikacja ta nie obejmowała pomiarów wydajności. Ocena wygody programowania oraz jakości dokumentacji ma charakter jakościowy i nie zależy od środowiska sprzętowego.

Testy przeprowadzono na laptopie wyposażonym w procesor Intel Core i5-10200H oraz 32 GB pamięci operacyjnej. Maszyna dysponuje dwoma układami graficznymi: zintegrowanym Intel UHD Graphics oraz dedykowanym NVIDIA GeForce GTX 1650. Obecność dedykowanego procesora graficznego ma znaczenie zwłaszcza dla wariantu Dear ImGui, który renderuje interfejs bezpośrednio przez OpenGL.

Środowisko programowe stanowił 64-bitowy system Windows 10 (wersja 22H2, kompilacja 19045). Kod kompilowano kompilatorem MSVC dostarczonym z pakietem Visual Studio 2022 (zestaw narzędzi v143), zgodnie ze standardem C++20. Procesem budowania zarządzały CMake w wersji 3.25.2 oraz menedżer pakietów Conan w wersji 1.66.0. Wszystkie warianty zbudowano w konfiguracji Release, z włączonymi optymalizacjami kompilatora, dla architektury x86_64.

Zestawienie kluczowych parametrów środowiska testowego przedstawia Tabela 2. Szczegółowy sposób pomiaru w każdym z obszarów opisano w odpowiednich podrozdziałach.

Tabela 2. Specyfikacja środowiska testowego

ELEMENT	WARTOŚĆ
PROCESOR	Intel Core i5-10200H (4 rdzenie / 8 wątków, 2,40 GHz)
PAMIĘĆ OPERACYJNA	32 GB
KARTA GRAFICZNA	NVIDIA GeForce GTX 1650 / Intel UHD Graphics
SYSTEM OPERACYJNY	Windows 10 64-bit (wersja 22H2, kompilacja 19045)
KOMPILATOR	MSVC

6.2. Testy wydajności renderowania

Renderowanie obrazu jest podstawowym zadaniem przeglądarki zdjęć. Badanie objęło dwie operacje: wczytanie i zdekodowanie pliku do wewnętrznej reprezentacji biblioteki oraz pojedynczy cykl renderowania klatki. Rzetelny pomiar samego renderowania jest jednak trudny i słabo porównywalny w bibliotekach działających w trybie zachowanym. Interfejs jest w nich przerysowywany jedynie na żądanie, a mierzony czas obejmuje wyłącznie skopiowanie zbuforowanej mapy bitowej, a nie pełne złożenie klatki, jak w trybie natychmiastowym modułu Dear ImGui. Z tego względu wyniki te potraktowano jako orientacyjne. Wszystkie pomiary wykonano zegarem wysokiej rozdzielczości `std::chrono::high_resolution_clock` umieszczonym bezpośrednio w kodzie każdego z modułów.

6.2.1 Czas cyklu renderowania

Czas cyklu renderowania określa koszt narysowania pojedynczej klatki obrazu, który jest już wczytany i przeskalowany do rozmiaru wyświetlania. Aby zapewnić powtarzalność i porównywalność pomiaru, tryb skalowania ustawiono na powiększenie (*Zoom*) o stałej wartości 30%. Dzięki temu rozmiar wyświetlanego obrazu pozostawał niezależny od wielkości okna aplikacji i był jednakowy w trakcie wszystkich pomiarów. Z tego powodu dla każdego backendu podano jedną uśrednioną wartość, obliczoną z serii po 100 klatek. Zakres mierzonej operacji różni się między bibliotekami, gdyż wynika z odmiennej architektury rysowania każdej z nich:

- w Qt pomiar obejmuje przerysowanie etykiety `QLabel::paintEvent`, czyli skopiowanie na ekran gotowej, przeskalowanej i zbuforowanej mapy bitowej;
- w wxWidgets pomiar obejmuje pełną procedurę `OnPaint`: utworzenie podwójnego bufora `wxAutoBufferedPaintDC`, wyczyszczenie tła oraz narysowanie gotowej bitmapy;
- w GTK pomiar obejmuje wyłącznie złożenie zbuforowanego obrazu `GdkPixbuf` za pomocą biblioteki Cairo;
- w Dear ImGui pomiar obejmuje renderowanie całej klatki interfejsu na karcie graficznej, a nie samego obrazu.

Tabela 3 zawiera uśrednione czasy cyklu renderowania dla wszystkich czterech modułów.

Tabela 3. Średni czas pojedynczego cyklu renderowania klatki obrazu

BACKEND	ŚREDNI CZAS RENDEROWANIA KLATKI OBRAZU
	[μ s]
QT	42,9
DEAR IMGUI	75,9
GTK	217,7
WXWIDGETS	393,4

Najkrótszy czas renderowania osiągnęło Qt. Wynika to z faktu, że pomiar dotyczy jedynie skopiowania gotowej, wcześniej przeskalowanej mapy bitowej. Samo skalowanie odbywa się poza cyklem rysowania, a kopiowanie bitmapy przez mechanizm rysowania Qt jest operacją bardzo szybką. Moduł Dear ImGui uzyskał drugi wynik, mimo że jako jedyny renderuje w trybie natychmiastowym i przy każdej klatce odbudowuje oraz wysyła na kartę graficzną dane całego interfejsu, a nie samego obrazu. Mierzony czas obejmuje zatem większy zakres pracy niż w pozostałych bibliotekach, a niski wynik świadczy o wydajności sprzętowego renderowania przez OpenGL oraz o niewielkim narzucie lekkiego interfejsu.

GTK i wxWidgets osiągnęły czasy o rząd wielkości dłuższe niż Qt. W przypadku GTK koszt wynika z budowania obrazu na procesorze przez bibliotekę Cairo. Najdłuższy czas zmierzono dla wxWidgets, jednak wynika to z tego, że pomiar obejmuje całą procedurę malowania okna, w tym utworzenie podwójnego bufora oraz wyczyszczenie tła. Te dodatkowe operacje częściowo wyjaśniają wyższy wynik. Mimo różnic w zakresie pomiaru wszystkie zmierzone wartości pozostają na poziomie mikrosekund, czyli wielokrotnie poniżej progu pojedynczej klatki przy odświeżaniu 60 Hz (około 16,7 ms). Oznacza to, że dla typowego zastosowania przeglądarki zdjęć czas renderowania w żadnej z bibliotek nie wpływa zauważalnie na działanie aplikacji, a zaobserwowane różnice mają znaczenie głównie porównawcze.

6.2.2 Czas wczytywania plików

Pomiar czasu wczytywania objął operację dekodowania pliku JPEG i umieszczenia jego zawartości w wewnętrznym obiekcie obrazu danej biblioteki. W przypadku Qt jest to wywołanie `QPixmap::load`, w wxWidgets `wxImage::LoadFile`, a w GTK `gdk_pixbuf_new_from_file`. W module opartym na Dear ImGui surowe piksele zwracane przez STB nie są bezpośrednio rysowalne, z tego powodu zmierzono pełną funkcję `LoadFile`, obejmującą dekodowanie funkcją `stbi_load` oraz przesłanie obrazu do tekstury `glTexImage2D`. Pomiar przeprowadzono dla trzech plików testowych o rozdzielczościach 800x600, 1920x1080 oraz 3840x2160 pikseli, każdy z nich wczytano trzykrotnie, a wynik uśredniono. Tabela 4 przedstawia uśrednione czasy wczytywania obrazu dla trzech rozdzielczości testowych.

Tabela 4. Średni czas wczytania obrazu

ROZDZIELCZOŚĆ	QT	WXWIDGETS	DEAR IMGUI	GTK
	[ms]	[ms]	[ms]	[ms]
800x600	4,12	3,73	2,54	1,63
1920x1080	14,22	14,09	9,21	4,99
3840x2160	53,28	53,31	32,26	16,63

Wyniki układają się w wyraźny ranking, jednakowy dla wszystkich trzech rozdzielczości. Najszybszy okazał się moduł GTK, następnie Dear ImGui, a najwolniejsze i praktycznie nierozróżnialne były Qt oraz wxWidgets. Różnice wynikają przede wszystkim z biblioteki dekodującej format JPEG. Qt oraz wxWidgets korzystają z biblioteki libjpeg, która stanowi domyślną konfigurację pakietów w menedżerze Conan. Stos GTK budowany w systemie Windows narzędziem gvsbuild korzysta natomiast z biblioteki libjpeg-turbo, czyli zoptymalizowanej implementacji libjpeg. We wszystkich bibliotekach czas wczytywania rośnie w przybliżeniu proporcjonalnie do liczby pikseli.

Wyniki niniejszego podrozdziału należy traktować jako orientacyjne, ponieważ w większym stopniu odzwierciedlają konfigurację i metodę pomiaru niż rzeczywiste możliwości samych bibliotek. Różnica w czasie wczytywania wynika z domyślnego doboru dekodera JPEG, który można zmienić w każdej z bibliotek. Z kolei czas cyklu renderowania jest trudny do rzetelnego porównania, gdyż biblioteki działające w trybie zachowanym przerysowują obraz jedynie na żądanie, a zakres mierzonej operacji różni się między nimi. Dla typowego zastosowania przeglądarki zdjęć wydajność renderowania nie stanowi zatem czynnika istotnie różnicującego badane biblioteki.

6.3. Pomiary zużycia pamięci i procesora

Niniejszy podrozdział obejmuje dwa wskaźniki obciążenia zasobów: zużycie procesora w stanie spoczynku aplikacji oraz zajętość pamięci operacyjnej.

6.3.1 Zużycie procesora

Zużycie procesora zmierzono w stanie spoczynku, przy otwartym oknie aplikacji z wczytanym obrazem, bez interakcji użytkownika. Wykorzystano skrypt zbierający dziesięć próbek w odstępach jednosekundowych. Dla każdej próbki obliczono przyrost czasu procesora zużytego przez proces, podzielony przez czas rzeczywisty. Przyjęta jednostka odpowiada wykorzystaniu jednego rdzenia logicznego, gdzie 100% oznacza pełne obciążenie jednego z ośmiu wątków sprzętowych. Tabela 5 przedstawia uśrednione wyniki.

Tabela 5. Zużycie procesora w stanie spoczynku

BACKEND	ŚREDNIA [%]	MEDIANA [%]
QT	0	0
WXWIDGETS	0	0
GTK	0	0
DEAR IMGUI	9	9,3

Wyniki ujawniają zasadniczą różnicę wynikającą z modelu renderowania. Qt, wxWidgets oraz GTK działają w trybie zachowanym i przerysowują interfejs wyłącznie w reakcji na zdarzenia, dlatego w spoczynku nie wykonują żadnej pracy graficznej, a ich zużycie procesora jest nierozróżnialne od zera. Moduł oparty na Dear ImGui działa w trybie natychmiastowym i odbudowuje całą klatkę w pętli, niezależnie od tego, czy zawartość okna uległa zmianie. W efekcie nawet bezczynna aplikacja stale obciąża procesor. W zastosowaniu takim jak przeglądarka zdjęć, gdzie aplikacja przez większość czasu pozostaje w spoczynku, stałe zużycie procesora przez moduł Dear ImGui jest zauważalną wadą.

6.3.2 Zużycie pamięci

Zużycie pamięci operacyjnej zmierzono w stanie spoczynku, bezpośrednio po uruchomieniu aplikacji, przy otwartym oknie bez wczytanego obrazu i bez interakcji użytkownika. Dla działającego procesu odczytano dwa wskaźniki opisujące jego zapotrzebowanie na pamięć z różnych perspektyw: zbiór roboczy (*Working Set*) oraz pamięć prywatną (*Private Bytes*). Każdy z wariantów uruchomiono kilkakrotnie, a podane wartości stanowią średnią z pomiarów.

Zbiór roboczy oznacza ilość fizycznej pamięci operacyjnej aktualnie przydzielonej procesowi, do której procesor ma natychmiastowy dostęp bez sięgania do pliku stronicowania na dysku. Obejmuje on zarówno pamięć prywatną procesu, jak i strony współdzielone, na przykład kod bibliotek dynamicznych, a także strony kodu mapowane z pliku wykonywalnego [34].

Pamięć prywatna oznacza natomiast ilość pamięci wirtualnej zatwierdzonej wyłącznie na potrzeby danego procesu, niewspółdzielonej z żadnym innym procesem [35]. Nie obejmuje ona kodu bibliotek ani stron mapowanych z plików, dlatego lepiej oddaje własne zapotrzebowanie aplikacji na pamięć, niezależne od tego, ile innych procesów korzysta z tych samych bibliotek współdzielonych.

Tabela 6 przedstawia uśrednione zużycie pamięci dla wszystkich czterech modułów.

Tabela 6. Zużycie pamięci operacyjnej w stanie spoczynku

BACKEND	ZBIÓR ROBOCZY [MB]	PAMIĘĆ PRYWATNA [MB]
WXWIDGETS	15,0	3,4
QT	27,5	8,9
DEAR IMGUI	59,4	81,7
GTK	127,7	140,4

Wyniki układają się w jednakowy ranking dla obu wskaźników. Najmniejsze zużycie pamięci osiągnął moduł wxWidgets, a w dalszej kolejności Qt, Dear ImGui oraz GTK. Niskie wartości dla wxWidgets wynikają z jego architektury, opartej na lekkiej nakładce na natywne kontrolki systemu Windows. Moduł Qt zajął drugie miejsce. Zwraca przy tym uwagę duża różnica między jego zbiorem roboczym a niewielką pamięcią prywatną. Różnica ta wynika ze statycznego linkowania. Kod biblioteki wkompiłowany w plik wykonywalny jest mapowany z tego pliku do pamięci, przez co powiększa zbiór roboczy, lecz nie jest zaliczany do pamięci prywatnej procesu.

Moduł Dear ImGui zużył w czasie działania więcej pamięci niż Qt oraz wxWidgets. Jego pamięć prywatna przewyższa nawet zbiór roboczy, co oznacza, że część zatwierdzonej pamięci nie pozostaje stale rezydentna w pamięci fizycznej. Wysokie zużycie wynika z renderowania interfejsu przez kartę graficzną. Sterownik graficzny oraz kontekst renderowania alokują znaczną ilość pamięci, niezależnie od prostoty samego interfejsu.

Największe zużycie pamięci, w obu wskaźnikach, odnotował moduł GTK. Wynika to przede wszystkim z charakteru tej biblioteki, stanowiącej kompletny zestaw narzędzi do budowy graficznego interfejsu. W odróżnieniu od pozostałych wariantów GTK korzysta z linkowania dynamicznego i ładuje wiele bibliotek współdzielonych, co powiększa jego zbiór roboczy. Na wysoką pamięć prywatną składają się natomiast własne struktury biblioteki.

Zestawienie ujawnia wyraźny podział badanych bibliotek pod względem gospodarowania pamięcią. Warianty oparte na natywnych kontrolkach systemu, czyli wxWidgets, oraz na statycznie wkompiłowanej bibliotece, zajmują najmniej pamięci. Z kolei moduły dostarczające własną, rozbudowaną warstwę uruchomieniową, czyli Dear ImGui i GTK, zużywają jej dużo więcej. Dla przeglądarki zdjęć uruchamianej na współczesnym komputerze osobistym żadna z odnotowanych wartości nie stanowi istotnego ograniczenia, jednak zaobserwowane różnice mają znaczenie

porównawcze i mogą być odczuwalne w środowiskach o ograniczonych zasobach lub w bardziej rozbudowanych aplikacjach.

6.4. Rozmiar plików binarnych i zależności

Niniejszy podrozdział przedstawia rozmiar produktu końcowego każdej z czterech implementacji prototypu. Na rozmiar dystrybucji składa się rozmiar pliku wykonywalnego oraz rozmiar bibliotek wymaganych do uruchomienia aplikacji. Ponieważ prototyp nie zawiera dodatkowych zasobów ani rozbudowanej logiki, zmierzony rozmiar odzwierciedla niemal wyłącznie narzut samej biblioteki interfejsu. Rozmiar pliku wykonywalnego odczytano z systemu plików, a zbiór zależności ustalono na podstawie bibliotek faktycznie ładowanych przez działający proces. Uwzględniono wyłącznie biblioteki niesystemowe podlegające dystrybucji, pomijając składniki systemu Windows oraz bibliotekę uruchomieniową kompilatora.

O rozkładzie tego rozmiaru decyduje sposób linkowania. Warianty Qt, wxWidgets i Dear ImGui linkowane są statycznie, natomiast wariant GTK korzysta z linkowania dynamicznego. Pakiet GTK pozyskiwany jest przez menedżer Conan, który na platformie Windows buduje bibliotekę przy użyciu narzędzia gvsbuild [29], dostarczającego GTK wyłącznie w postaci bibliotek współdzielonych. Z tego powodu linkowanie dynamiczne było praktyczną koniecznością. Zmierzone wartości przedstawia Tabela 7.

Tabela 7. Rozmiar plików binarnych i zależności poszczególnych wariantów

BIBLIOTEKA	LINKOWANIE	PLIK WYKONYWALNY [MB]	ŁĄCZNY ROZMIAR DYSTRYBUCJI [MB]
DEAR IMGUI	statyczne	2,19	2,19
WXWIDGETS	statyczne	5,55	5,55
QT	statyczne	17,81	17,81
GTK	dynamiczne	0,23	27,08

Najmniejszy produkt końcowy daje Dear ImGui: samodzielny plik o rozmiarze 2,19 MB bez zależności zewnętrznych. wxWidgets oraz Qt również tworzą pojedyncze pliki niewymagające dodatkowych bibliotek, odpowiednio 5,55 MB i 17,81 MB, ponieważ cały kod biblioteki zawarty jest w pliku wykonywalnym. Wariant GTK reprezentuje model przeciwny: plik wykonywalny zajmuje jedynie 0,23 MB, lecz jego uruchomienie wymaga 27 bibliotek współdzielonych o łącznym rozmiarze 26,85 MB, co daje największą i najbardziej rozproszoną dystrybucję, równą 27,08 MB. Wynika to z architektury GTK, która renderuje interfejs we własnym silniku graficznym i przenosi cały zestaw bibliotek rysujących oraz przetwarzających tekst, między innymi cairo, pango, harfbuzz i freetype.

Linkowanie statyczne wariantów Qt, wxWidgets i Dear ImGui upraszcza wdrożenie do skopiowania jednego pliku, kosztem jego rozmiaru, natomiast linkowanie dynamiczne GTK utrzymuje znikomy plik wykonywalny, lecz przenosi ciężar na komplet bibliotek towarzyszących. Z perspektywy prostej aplikacji desktopowej najkorzystniej wypada Dear ImGui, przy czym Qt zachowuje prostotę wdrożenia mimo największego pliku, a GTK wnosi najbardziej złożoną dystrybucję. Spośród bibliotek trybu zachowanego najlepiej wypada wxWidgets, łączący niewielki rozmiar dystrybucji z prostotą wdrożenia.

6.5. Czas kompilacji

Niniejszy podrozdział przedstawia czas kompilacji samego prototypu dla każdego z czterech wariantów. Pomiar obejmuje wyłącznie budowanie kodu aplikacji, to znaczy kompilację plików źródłowych wybranego backendu oraz pliku wykonywalnego wraz z ich linkowaniem. Biblioteki GUI i pozostałe zależności zostały wcześniej zbudowane i umieszczone w pamięci podręcznej menedżera Conan, wobec czego ich kompilacja nie wchodzi w skład mierzonego czasu. Budowanie bibliotek od źródeł to jednorazowy koszt przygotowania środowiska, silnie zależny od konfiguracji i łączy sieciowego, podczas gdy przedmiotem porównania jest wpływ samej biblioteki na czas budowania korzystającej z niej aplikacji.

Każdy wariant kompilowano w trybie *Release*, za każdym razem od zera, z usunięciem wyników wcześniejszej kompilacji. Etap konfiguracji systemu budowania wykonywano poza pomiarem, mierząc wyłącznie czas kompilacji i linkowania. Budowanie prowadzono równolegle, z wykorzystaniem ośmiu wątków, jednakowo dla wszystkich wariantów. Podane wartości są medianami z sześciu przebiegów, a niewielki rozrzut wyników potwierdza powtarzalność pomiaru. Zestawienie przedstawia Tabela 8.

Tabela 8. Czas kompilacji prototypu

BIBLIOTEKA	CZAS KOMPILACJI
	[s]
GTK	7,4
DEAR IMGUI	9,9
WXWIDGETS	12,5
QT	14,2

Najkrótszy czas kompilacji osiąga wariant GTK, a najdłuższy Qt, pomiędzy nimi są Dear ImGui oraz wxWidgets. Różnice są dla aplikacji tej wielkości niewielkie, lecz powtarzalne, a w rozbudowanym interfejsie graficznym mogłyby okazać się istotnie większe, gdyż generowanie kodu i złożoność nagłówków narastają wraz z wielkością projektu. Na zaobserwowane wyniki składa się kilka czynników związanych z charakterem poszczególnych bibliotek:

- najdłuższy czas wariantu Qt wynika z dodatkowego etapu generowania kodu przez kompilator metaobiektów oraz narzędzie uic, a także z linkowania ze statycznie dołączaną, obszerną biblioteką,
- wxWidgets osiąga czas krótszy, lecz nadal wysoki, na co wpływa statyczne linkowanie biblioteki wraz z SQLite,
- w przypadku Dear ImGui, mimo niewielkiego rozmiaru samej biblioteki, kompilacji podlegają także pliki wiążące modułu platformowego i renderującego (imgui_impl_sdl2 oraz imgui_impl_opengl3), a wynik obejmuje statyczne linkowanie z bibliotekami imgui i SDL2,
- najkrótszy czas wariantu GTK, mimo dużej liczby nagłówków, wynika z linkowania dynamicznego. Aplikacja łączona jest jedynie z bibliotekami importu, podczas gdy właściwy kod biblioteki pozostaje w osobnych plikach DLL, co skraca etap linkowania.

Linkowanie dynamiczne, które w zestawieniu rozmiaru dystrybucji wypadło dla GTK najmniej korzystnie, w przypadku czasu kompilacji działa na jego korzyść, odciażając etap linkowania. Statycznie linkowane warianty Qt, wxWidgets i Dear ImGui ponoszą natomiast koszt dołączania kodu biblioteki do pliku wykonywalnego. Należy przy tym podkreślić, że przedstawione wartości dotyczą wyłącznie kompilacji kodu aplikacji. Pełne przygotowanie środowiska, obejmujące zbudowanie

bibliotek od źródeł, jest znacznie bardziej czasochłonne, szczególnie dla rozbudowanego frameworka Qt, i stanowi jednorazowy koszt ponoszony przed rozpoczęciem właściwej pracy nad aplikacją.

6.6. Przenoszalność

Niniejszy podrozdział ocenia przenoszalność czterech wariantów prototypu między systemami Windows i Linux na podstawie możliwości ich zbudowania i uruchomienia na obu platformach oraz zakresu niezbędnych zmian w konfiguracji i kodzie źródłowym.

Wszystkie cztery biblioteki są oficjalnie wspierane na obu docelowych platformach. Prototyp w wariantach Qt, wxWidgets oraz Dear ImGui buduje się na systemach Windows i Linux w porównywalny sposób, a różnice konfiguracyjne ograniczają się do kilku warunków w pliku Conan oraz w plikach CMake, opisanych w 5.1.1. Wyjątkiem jest GTK: na Linuksie, gdzie jest fundamentem środowiska GNOME, biblioteka jest dostępna natywnie i instaluje się bez trudności. Natomiast na Windows wymaga zbudowania całego stosu biblioteki narzędziem gvsbuild i dystrybucji w postaci wielu bibliotek współdzielonych. Proces ten obsługuje wprawdzie Conan, zdejmując z programisty ręczną pracę, jednak sam proces pozostaje skomplikowany i bardziej podatny na problemy niż w pozostałych wariantach, co jest słabością GTK pod względem przenoszalności na platformę Windows.

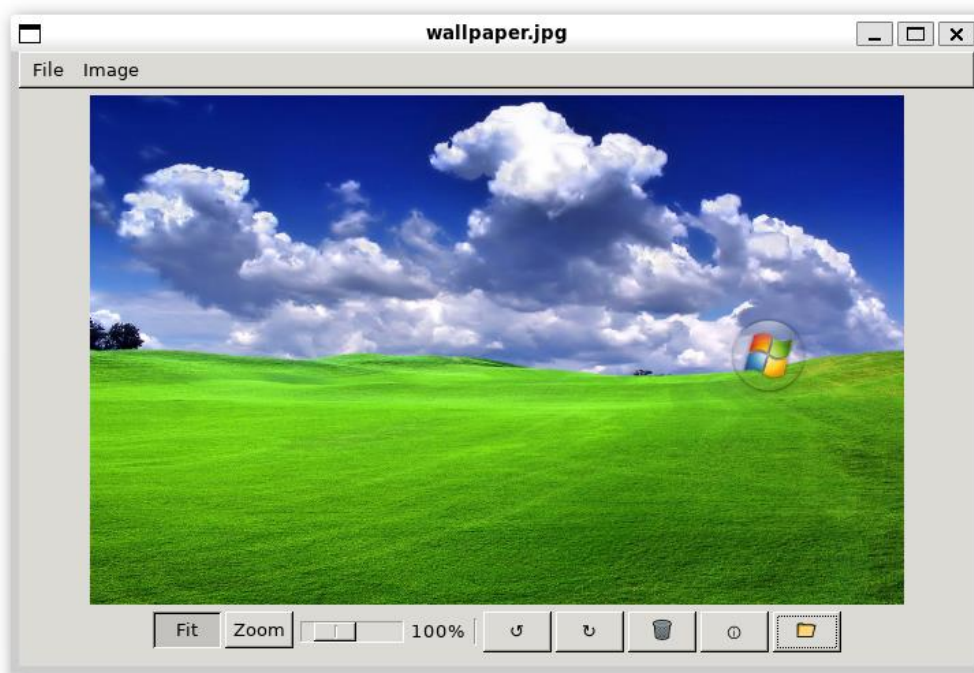
Drugim wymiarem jest ilość kodu zależnego od platformy, jaką programista musi napisać przy integracji z systemem operacyjnym. W prototypie ujawnia się ona głównie w funkcji otwierającej katalog pliku w systemowym menedżerze plików. Jej zakres różni się między bibliotekami:

- wxWidgets nie zawiera żadnych rozgałęzień zależnych od platformy. Katalog otwiera przenośna funkcja `wxLaunchDefaultApplication`,
- Qt wymaga jednego rozgałęzienia, importu wtyczki `QWindowsIntegrationPlugin` przy linkowaniu statycznym na Windows. Katalog otwiera przenośna metoda `QDesktopServices::openUrl`,
- GTK wymaga jednego rozgałęzienia do otwarcia menedżera plików, polecenia `explorer` na Windows, a na pozostałych platformach funkcji `g_app_info_launch_default_for_uri`,
- Dear ImGui wymaga najwięcej kodu zależnego od platformy, gdyż nie dostarcza abstrakcji integracji z systemem. W przygotowaniu środowiska graficznego pomagają oficjalne przykłady biblioteki dla każdej konfiguracji, jednak otwarcie katalogu pliku trzeba było zrealizować osobno dla każdego systemu.

Trzecim wymiarem jest spójność wizualna interfejsu między platformami. Zrzuty okien głównych w rozdziale 5 wykonano w systemie Windows. Sposób zachowania wyglądu w innym systemie różni się pomiędzy bibliotekami:

- wxWidgets, korzystając z natywnych kontrolek systemu, dostosowuje wygląd aplikacji do stylu danej platformy, dzięki czemu na każdej z nich wygląda naturalnie. Różnicę tę przedstawia Rysunek 14, ukazujący prototyp uruchomiony w systemie Linux, którego wygląd odbiega od wersji dla systemu Windows z Rysunek 10;
- Qt rysuje interfejs we własnym silniku graficznym, lecz dzięki stylom platformowym zbliża go do wyglądu natywnego, przez co w systemie Linux wygląda odmiennie niż w systemie Windows, co przedstawia Rysunek 15;
- GTK rysuje interfejs własnym silnikiem, prezentując wygląd naturalny w środowisku GNOME, lecz całkowicie odbiegający od natywnego w systemie Windows, a okno wygląda identycznie w obu systemach. Rysunek 13 odpowiada również jego wyglądowi w systemie Linux;

- Dear ImGui rysuje interfejs własnym silnikiem, niezależnie od systemu, wobec czego jego wygląd pozostaje identyczny na obu platformach, tak jak na Rysunek 12.



Rysunek 14. Okno główne prototypu wxWidgets uruchomionego w systemie Linux



Rysunek 15. Okno główne prototypu Qt uruchomionego w systemie Linux

Podsumowując, wszystkie cztery warianty budują i uruchamiają się na obu platformach bez większych zmian kodu realizującego logikę aplikacji. Najwyższą przenoszalność w praktyce zapewnia wxWidgets, którego kod źródłowy nie zawiera żadnych rozgałęzień zależnych od platformy, oraz Qt. Dear ImGui buduje się w obu systemach, lecz przenosi na programistę największą liczbę zależnych od platformy fragmentów kodu. GTK oferuje przenośny kod źródłowy, jego słabością pozostaje jednak głównie przeznaczenie biblioteki na system Linux.

6.7. Ocena wygody programowania

Niniejszy podrozdział ocenia wygodę programowania przy użyciu każdej z czterech bibliotek na podstawie obserwacji zebranych podczas implementacji prototypu. Ocena obejmuje projekt interfejsu programistycznego, ilość kodu pomocniczego (*boilerplate*) wymaganego do realizacji typowych zadań oraz dostępność narzędzi wspierających pracę programisty. Ma ona charakter subiektywny i dotyczy wyłącznie scenariusza tworzenia aplikacji desktopowej o zakresie porównywalnym z prototypem.

Spośród porównywanych bibliotek najlepiej pod tym względem wypadło Qt. Biblioteka ta jest popularna i dobrze udokumentowana, dzięki czemu rozwiązania większości napotkanych problemów można szybko odnaleźć w oficjalnej dokumentacji lub zasobach społeczności. Klasy i metody stosują konwencje nazewnictwa zgodne ze standardem C++, co czyni API czytelnym i przewidywalnym. Dedykowane narzędzie Qt Designer umożliwia wizualne projektowanie układu interfejsu, co skraca czas budowania okien w porównaniu z podejściem czysto programistycznym. Moduły dodatkowe, takie jak Qt SQL czy Qt Charts, pozwalają realizować wymagania wykraczające poza interfejs graficzny bez sięgania po zewnętrzne zależności.

wxWidgets pod względem wygody jest zbliżone do Qt. API jest przejrzyste, kompilacja przebiega bez komplikacji, a konfiguracja środowiska jest prostsza, gdyż biblioteka nie wymaga obsługi

dodatkowych wtyczek. Biblioteka skupia się wyłącznie na warstwie interfejsu graficznego i nie oferuje wbudowanego modułu bazodanowego, co oznaczało konieczność korzystania bezpośrednio z interfejsu języka C biblioteki SQLite i zwiększyło narzut zarządzania zasobami w porównaniu z modułem Qt SQL.

Dear ImGui jest proste przy tworzeniu interfejsu narzędziowego. Każdy element opisywany jest jednym wywołaniem funkcji, a dodanie nowej kontrolki nie wymaga dodatkowej konfiguracji. Trudności ujawniają się przy budowie pełnej aplikacji desktopowej, gdyż biblioteka przeznaczona jest do osadzania w istniejącym silniku graficznym, wobec czego wymaga ręcznego przygotowania środowiska graficznego. Biblioteka nie dostarcza żadnych mechanizmów wczytywania plików graficznych ani wbudowanych opcji manipulacji obrazem, wobec czego niezbędna okazała się integracja zewnętrznej biblioteki STB oraz ręczna manipulacja współrzędnymi UV przy obracaniu obrazu.

GTK okazało się najmniej wygodne spośród porównywanych bibliotek. Interfejs języka C wymusza wzorce uważane w C++ za złe praktyki: obsługa zdarzeń opiera się na wolnych funkcjach zamiast metodach klasy, każda funkcja obsługi wymagała ręcznego rzutowania parametru `userData`, a powszechne użycie makr takich jak `GTK_SCROLLLED_WINDOW` do weryfikacji i rzutowania typów jest podejściem nieuznawanym za dobrą praktykę w C++. Wymienione cechy składają się na największy narzut kodu i najmniej płynne doświadczenie programistyczne spośród badanych bibliotek.

Podsumowując, najwygodniejsze okazało się Qt, łączące spójne API z dedykowanymi narzędziami i bogatym ekosystemem modułów. `wxWidgets` jest zbliżone pod względem prostoty użycia, lecz ograniczone do warstwy graficznej. Dear ImGui sprawdza się w swoim docelowym zastosowaniu, lecz jako samodzielna aplikacja desktopowa generuje dodatkowy narzut integracyjny. GTK, ze względu na interfejs języka C, jest najmniej wygodne przy programowaniu w C++.

6.8. Porównanie dokumentacji

Niniejszy podrozdział porównuje jakość dokumentacji czterech bibliotek, między innymi kompletność i aktualność oficjalnej dokumentacji interfejsu programistycznego, dostępność poradników i przykładów kodu oraz zasoby tworzone przez społeczność. Ocena opiera się na materiałach wykorzystywanych podczas implementacji prototypu i ma charakter jakościowy. Liczebność zasobów społeczności traktowana jest przy tym jako pośrednia miara popularności i dojrzałości projektu.

Qt dysponuje najobszerniejszą i najlepiej zorganizowaną dokumentacją spośród porównywanych bibliotek. Oficjalny portal obejmuje pełny opis interfejsu programistycznego, artykuły przeglądowe, poradniki wprowadzające oraz rozbudowany zbiór przykładów, które można uruchomić bezpośrednio w środowisku Qt Creator [30], a opisy klas zawierają fragmenty kodu ilustrujące typowe zastosowania. Dzięki dużej popularności frameworka dokumentację oficjalną uzupełnia bogaty zbiór materiałów tworzonych przez społeczność: poradników, kursów oraz odpowiedzi na forach i w serwisach pytań i odpowiedzi, czego miarą jest ponad 86 tysięcy pytań oznaczonych tagiem `qt` w serwisie Stack Overflow [31]. W praktyce oznacza to, że na większość problemów napotkanych podczas implementacji można szybko odnaleźć gotowe rozwiązanie.

Dokumentacja `wxWidgets` prezentuje dobry, choć mniej rozbudowany niż w przypadku Qt, poziom. Oficjalny opis interfejsu programistycznego generowany jest narzędziem Doxygen z nagłówek interfejsu [32] i obejmuje pełne zestawienie klas, a opisy wielu z nich zawierają fragmenty kodu ilustrujące użycie. Bibliotekę uzupełnia obszerny zestaw przykładów dołączanych do dystrybucji [33] oraz książka poświęcona programowaniu w `wxWidgets` [4]. Społeczność biblioteki jest mniej liczna niż w przypadku Qt, co przekłada się na mniejszą liczbę zewnętrznych poradników, jednak dostępne materiały okazały się wystarczające do realizacji wszystkich wymagań prototypu.

Dear ImGui nie udostępnia tradycyjnej, osobnej dokumentacji interfejsu programistycznego. Rolę tę pełnią obszernie komentarze w plikach nagłówkowych, aktywna pomoc autora na GitHub, krótki poradnik startowy oraz przede wszystkim plik `imgui_demo.cpp` [11], prezentujący w działaniu niemal wszystkie funkcje biblioteki wraz z odpowiadającym im kodem źródłowym. Repozytorium zawiera ponadto gotowe przykłady integracji dla każdej kombinacji modułu platformowego i renderującego. Ze względu na minimalistyczny charakter biblioteki i niewielką liczbę funkcji takie podejście okazało się w praktyce wystarczające, gdyż większość zagadnień można rozwiązać, odszukując odpowiedni fragment w pliku demonstracyjnym.

Dokumentacja GTK okazała się najmniej przejrzysta spośród porównywanych bibliotek. Oficjalny portal zawiera kompletny opis interfejsu programistycznego oraz krótki poradnik wprowadzający [13], ma jednak charakter powierzchowny i nie zawiera przykładów implementacji wykraczających poza podstawowe scenariusze. Dodatkową trudnością jest fakt, że GTK napisane jest w języku C, więc dokumentacja opisuje interfejs w tym języku. Korzystanie z biblioteki z poziomu C++, jak w prototypie, wymaga przenoszenia wzorców z dokumentacji języka C. W połączeniu z mniejszą popularnością biblioteki na platformach innych niż Linux skutkuje to ograniczoną liczbą materiałów społeczności, przez co rozwiązywanie problemów było zauważalnie trudniejsze niż w przypadku Qt.

Podsumowując, najlepiej pod względem dokumentacji wypada Qt, łączące kompletną i przejrzystą dokumentację oficjalną z najliczniejszymi zasobami społeczności. wxWidgets oferuje solidną dokumentację z przykładami kodu, ustępując Qt głównie objętością materiałów i wielkością społeczności. Dear ImGui, mimo braku formalnej dokumentacji, zapewnia wystarczające wsparcie dzięki przykładom i prostocie biblioteki. Najslabiej wypada GTK, którego powierzchowna, zorientowana na język C dokumentacja oraz ograniczone zasoby społecznościowe czynią naukę tej biblioteki najtrudniejszą spośród badanych.

6.9. Zestawienie wyników

Niniejszy podrozdział zbiera wyniki badań opisanych w podrozdziałach 6.2-6.8 i ujmuje je w całościowym porównaniu czterech bibliotek. Część kryteriów oceniono ilościowo, na podstawie pomiarów wykonanych w systemie Windows, a część jakościowo, na podstawie doświadczeń zebranych podczas implementacji prototypu.

Punktowane zestawienie wyników przedstawia Tabela 9. Dla każdego kryterium przypisano poszczególnym bibliotekom od czterech punktów za wynik najlepszy do jednego za najgorszy, a wynikiem nierozróżnialnym przyznano jednakową, wyższą liczbę punktów. Punktacja ma charakter umowny i równo waży wszystkie kryteria, przez co służy wyłącznie porównaniu, a nie obiektywnemu rozstrzygnięciu.

Tabela 9. Zestawienie punktowe wyników porównania

KRYTERIUM	QT	WXWIDGETS	DEAR IMGUI	GTK
WYDAJNOŚĆ RENDEROWANIA	4	1	3	2
CZAS WCZYTYWANIA OBRAZU	2	2	3	4
ZUŻYCIE PROCESORA	4	4	1	4
ZUŻYCIE PAMIĘCI	3	4	2	1

ROZMIAR DYSTRYBUCJI	2	3	4	1
CZAS KOMPILACJI	1	2	3	4
PRZENOSZALNOŚĆ	3	4	2	2
WYGODA PROGRAMOWANIA	4	3	2	1
JAKOŚĆ DOKUMENTACJI	4	3	2	1
SUMA PUNKTÓW	27	26	22	20

Qt uzyskało najlepsze oceny w wygodzie programowania i dokumentacji. Dysponuje także najszerzym zestawem wbudowanych modułów, co pozwoliło zrealizować obsługę bazy danych, drukowanie i histogram bez zewnętrznych zależności. Wadami są największy plik wykonywalny oraz najdłuższy czas kompilacji, przy umiarkowanym zużyciu pamięci.

wxWidgets osiągnęło najbardziej wyrównane wyniki. Zajęło pierwsze miejsce w zużyciu pamięci oraz przenoszalności, a drugie w rozmiarze dystrybucji, wygodzie i dokumentacji, ustępując w dwóch ostatnich jedynie Qt. Węższa funkcjonalność wbudowana wymusiła użycie zewnętrznej biblioteki SQLite. Najwyższy w zestawieniu czas renderowania wynika z szerszego zakresu mierzonej operacji, a nie z rzeczywistej powolności biblioteki.

Dear ImGui ma najmniejszy rozmiar dystrybucji oraz jedno z najkrótszych czasów kompilacji i wczytywania obrazu. Po przeciwnej stronie znalazły się stałe zużycie procesora w spoczynku, wynikające z trybu natychmiastowego oraz wysokie zużycie pamięci związane z renderowaniem na karcie graficznej. Przy samodzielnej aplikacji desktopowej biblioteka wymaga największego nakładu pracy integracyjnej i nie dostarcza gotowych mechanizmów wczytywania obrazów ani funkcji wykraczających poza zakres biblioteki graficznej.

GTK osiągnęło najkrótszy czas wczytywania obrazu oraz najkrótszy czas kompilacji, co wynika odpowiednio z domyślnego użycia zoptymalizowanego dekodera JPEG i z linkowania dynamicznego. To samo linkowanie dynamiczne dało jednak najobszerniejszą i najbardziej rozproszoną dystrybucję. Biblioteka uzyskała najsłabsze wyniki w zużyciu pamięci, wygodzie programowania w języku C++ oraz dokumentacji.

Wyniki nie wskazują pojedynczej biblioteki najlepszej we wszystkich kryteriach. Dobór biblioteki jest kompromisem zależnym od priorytetów konkretnego zastosowania.

6.10. Rekomendacje w zależności od potrzeb projektu

Wnioski z zestawienia wyników prowadzą do praktycznych zaleceń dotyczących wyboru biblioteki. Ponieważ żadne z rozwiązań nie jest najlepsze pod każdym względem, przedstawiono typowe scenariusze zastosowań wraz z sugerowaną biblioteką oraz uzasadnieniem opartym na zebranych pomiarach i obserwacjach.

Do rozbudowanej aplikacji produkcyjnej, łączącej interfejs graficzny z zaawansowanymi funkcjami, takimi jak baza danych czy prezentacja danych na wykresach, najlepiej nadaje się Qt. Biblioteka oferuje najwygodniejsze API, najobszerniejszą dokumentację oraz komplet wbudowanych modułów, dzięki czemu większość wymagań można zrealizować bez zewnętrznych zależności. Większy plik wykonywalny i dłuższy czas kompilacji mają w takich projektach znaczenie drugorzędne wobec prostoty i kompletności narzędzi.

Do lekkiej, wieloplatformowej aplikacji, w której liczą się mały rozmiar dystrybucji, niskie zużycie zasobów oraz wygląd zgodny z systemem operacyjnym, zaleca się wxWidgets. Biblioteka ma najniższe zużycie pamięci oraz jest najbardziej przenoszalna, a dzięki natywnym kontrolkom aplikacja wygląda naturalnie na każdej platformie przy minimalnej ilości kodu zależnego od systemu. Jest to rozwiązanie o najbardziej zrównoważonych wynikach testów, odpowiednie również dla środowisk o ograniczonych zasobach.

Do narzędzia diagnostycznego lub interfejsu osadzonego w aplikacji czasu rzeczywistego, na przykład w grze lub programie do wizualizacji, dobrym wyborem jest Dear ImGui. Tworzenie interfejsu jest tu najszybsze, biblioteka naturalnie integruje się z pętlą renderowania i daje najmniejszy produkt końcowy. Należy jednak uwzględnić stałe zużycie procesora oraz wysokie zużycie pamięci, przez co rozwiązanie to jest mniej odpowiednie dla aplikacji pozostających długo w bezczynności.

Do aplikacji przeznaczonej przede wszystkim na system Linux lub środowisko GNOME właściwym wyborem jest GTK, gdzie biblioteka jest dobrze zintegrowana z systemem i nie wymaga dystrybucji dodatkowych zależności. W projekcie wieloplatformowym z istotnym udziałem systemu Windows należy jednak uwzględnić złożone budowanie i obszerną dystrybucję biblioteki oraz mniejszą wygodę programowania w języku C++.

W praktyce o wyborze najczęściej decyduje jeden dominujący czynnik. Bogactwo funkcji i wygoda programowania wskazują na Qt, oszczędność zasobów i prosta przenoszalność na wxWidgets, minimalny rozmiar oraz integracja z renderowaniem na Dear ImGui, a docelowa platforma Linux na GTK.

7. Podsumowanie i wnioski

Niniejszy rozdział zamyka pracę. Podrozdział 7.1 ocenia przyjęte rozwiązanie, czyli metodę porównania opartą na jednolitej aplikacji testowej oraz architekturę prototypu z wymiennymi backendami, wskazując jego mocne strony i ograniczenia. Podrozdział 7.2 zbiera wnioski płynące z przeprowadzonego porównania, odnosi je do celu sformułowanego w rozdziale 1 oraz nakreśla kierunki dalszego rozwoju.

7.1. Wady i zalety rozwiązania

Rozwiązanie zaproponowane w niniejszej pracy obejmuje metodę porównania bibliotek GUI przez implementację możliwie zbliżonych do siebie aplikacji testowych. Przedmiotem oceny w tym podrozdziale jest samo przyjęte podejście, a nie poszczególne biblioteki, których mocne i słabe strony zestawiono w podrozdziałach 6.9 oraz 6.10. Ocenie podlega zatem trafność i ograniczenia obranego sposobu badania.

Do najważniejszych zalet przyjętego rozwiązania należą:

- porównywalność wyników, wynikająca z realizacji tej samej specyfikacji funkcjonalnej w każdej z bibliotek. Współdzielona logika aplikacji oraz minimalny interfejs `IGui` sprawiają, że zaobserwowane różnice można przypisać właściwościom samych bibliotek, a nie odmiennej implementacji wymagań;
- realistyczny zakres aplikacji testowej, obejmujący dekodowanie i przetwarzanie obrazów, obsługę systemu plików, prezentację metadanych, a także opcjonalnie bazę danych i histogram. Dzięki temu porównanie ujawnia różnice niewidoczne w prostych przykładach, takie jak brak wbudowanych modułów wymuszający użycie zewnętrznych;
- generyczność i rozszerzalność architektury, zbliżonej w działaniu do systemu wtyczek. Dodanie kolejnej biblioteki sprowadza się do utworzenia nowego backendu implementującego interfejs `IGui` oraz wskazania go w konfiguracji budowania, bez zmian we współdzielonym kodzie;
- oparcie wniosków na dwóch niezależnych podstawach: pomiarach ilościowych wykonanych w jednym, kontrolowanym środowisku oraz doświadczeniu zebranych podczas implementacji każdej z czterech wersji aplikacji;
- potwierdzona przenoszalność, gdyż rozwiązanie kompiluje się i działa na systemach Windows oraz Linux bez dużych zmian w kodzie i konfiguracji budowania.

Szczególne znaczenie ma generyczność rozwiązania. Architektura wymiennych backendów nie jest związana z konkretnym zestawem czterech bibliotek ani z funkcją przeglądarki zdjęć, dzięki czemu może posłużyć jako szkielet do porównania innych bibliotek graficznych lub aplikacji innego typu.

Przyjęte podejście ma jednak również wady i ograniczenia, które należy uwzględnić przy interpretacji wyników:

- oparcie badania na jednym typie aplikacji. Przeglądarka zdjęć reprezentuje program o przewadze stanów bezczynnych i statycznej treści, przez co część obserwacji, na przykład niekorzystne dla Dear ImGui stałe zużycie procesora, może nie przenosić się na aplikacje o dynamicznym interfejsie czy narzędzia czasu rzeczywistego;
- niesymetryczne pokrycie funkcji. Dążenie do porównywalności ogranicza wspólną specyfikację do zakresu możliwego do zrealizowania we wszystkich bibliotekach, a funkcje

dostępne tylko w części z nich, takie jak wbudowana baza danych czy histogram, nie poddają się w pełni jednolitemu porównaniu;

- cienka warstwa wspólna. Ujednolicenie cyklu życia i obsługi zdarzeń poszczególnych bibliotek prowadziłoby do sztucznych abstrakcji. Interfejs `IGui` deklaruje wyłącznie metodę `run()`, wobec czego rzeczywiście współdzielone pozostają jedynie specyfikacja, punkt wejścia oraz infrastruktura budowania, a znaczna część kodu każdego backendu jest swoista dla danej biblioteki;
- subiektywność kryteriów jakościowych. Ocena wygody programowania oraz dokumentacji opiera się na doświadczeniu jednego programisty i bywa zależna od wcześniejszej znajomości danej biblioteki, przez co nie jest w pełni powtarzalna;
- ograniczenia samych pomiarów. Czas renderowania jest trudny do porównania między trybem zachowanym a natychmiastowym, a czas wczytywania odzwierciedla jedynie domyślny dobór dekodera formatu JPEG.

Mimo wymienionych ograniczeń przyjęte rozwiązanie spełniło swoje zadanie, dostarczając spójnej i powtarzalnej podstawy do wielokryterialnego porównania czterech bibliotek. Wskazane wady mają charakter świadomie przyjętych kompromisów, wynikających z dążenia do porównywalnych wyników, a ich znajomość pozwala ocenić ważność wniosków.

7.2. Wnioski

Cel pracy, sformułowany w podrozdziale 1.1, został zrealizowany. Powstał prototyp przeglądarki zdjęć zaimplementowany czterokrotnie, w bibliotekach Qt, wxWidgets, Dear ImGui oraz GTK, na podstawie wspólnej specyfikacji funkcjonalnej, a same biblioteki poddano wielokryterialnemu porównaniu. Oceniono wygodę programowania, przenoszalność między systemami Windows i Linux, wydajność oraz zużycie zasobów, rozmiar pliku wykonywalnego wraz z zależnościami, zgodność ze standardem C++ oraz jakość dokumentacji. Wyniki pomiarów ilościowych i obserwacji jakościowych przedstawiono w rozdziale 6, a ich zestawienie oraz rekomendacje doboru biblioteki zawarto w podrozdziałach 6.9 i 6.10.

Najważniejszym wnioskiem płynącym z porównania jest brak biblioteki uniwersalnie najlepszej. Każda z badanych bibliotek ma swoje mocne i słabe strony, a ich obserwowany rozkład nie jest przypadkowy. Wynika on bezpośrednio z architektury każdej z bibliotek. Zależność tę ilustrują następujące obserwacje:

- paradygmat działania, czyli tryb zachowany lub natychmiastowy, tłumaczy zachowanie aplikacji w spoczynku. Dear ImGui, odbudowując całą klatkę w każdym przebiegu pętli, stale obciąża procesor, podczas gdy biblioteki trybu zachowanego przerysowują interfejs wyłącznie na żądanie i w spoczynku nie wykonują pracy;
- sposób renderowania kontroltek, oparty na kontrolkach systemu lub na własnym silniku graficznym, tłumaczy zużycie pamięci, spójność wizualną oraz przenoszalność;
- model linkowania, statyczny lub dynamiczny, tłumaczy rozmiar dystrybucji oraz czas kompilacji. Statycznie linkowane aplikacje tworzą pojedynczy plik wykonywalny kosztem jego rozmiaru i dłuższego linkowania, natomiast dynamicznie linkowane mają znikomy plik wykonywalny i najkrótszy czas kompilacji, lecz przenoszą ciężar na komplet bibliotek współdzielonych;
- zakres biblioteki, od kompletnego frameworka po minimalną bibliotekę, tłumaczy bogactwo funkcji oraz wygodę programowania. Rozbudowane frameworki dostarczają przydatne wbudowane moduły, kosztem większego pliku wykonywalnego i dłuższej kompilacji,

podczas gdy minimalistyczne biblioteki dają najmniejszy produkt końcowy, lecz wymagają większego nakładu pracy;

- język bazowy biblioteki, C lub C++, tłumaczy wygodę programowania oraz zgodność z idiomami języka. GTK jako biblioteka języka C wymusza wzorce nieprzystające do C++, co czyni ją najmniej wygodną spośród badanych, mimo poprawnej współpracy z kompilatorami języka C++.

Wybór biblioteki GUI sprowadza się do wyboru modelu najlepiej odpowiadającego priorytetom konkretnego projektu, a nie do poszukiwania rozwiązania najlepszego w każdej sytuacji. Zalecenia powiązane z typowymi scenariuszami zastosowań zebrano w podrozdziale 6.10. Dostarczają one programiście podstawy do świadomego wyboru, opartej na zmierzonych właściwościach bibliotek, a nie wyłącznie na ich popularności.

Przeprowadzone porównanie nie wyczerpuje tematu i wyznacza dwa główne kierunki dalszych badań. Pierwszym z nich jest poszerzenie zbioru porównywanych narzędzi. Otwarta architektura prototypu, niezwiązana z konkretnym zestawem bibliotek, sprzyja porównaniu kolejnych bibliotek graficznych języka C++. Drugim kierunkiem jest powtórzenie badania dla aplikacji o innym charakterze niż przeglądarka zdjęć, w szczególności programu o dynamicznie zmieniającym się interfejsie, w którym tryb natychmiastowy biblioteki mogłby ujawnić zalety niewidoczne w aplikacji pozostającej głównie w spoczynku.

Prace cytowane

- [1]. Microsoft, *Retained Mode Versus Immediate Mode*, <https://learn.microsoft.com/en-us/windows/win32/learnwin32/retained-mode-versus-immediate-mode>, Dostęp 15.05.2026
- [2]. Qt Group, Dokumentacja *Signals & Slots*, <https://doc.qt.io/qt-6/signalsandslots.html>, Dostęp 15.05.2026
- [3]. O. Cornut, *Dear ImGui, About the ImGui paradigm*, <https://github.com/ocornut/imgui/wiki/About-the-ImGui-paradigm>, Dostęp 15.05.2026
- [4]. J. Smart, K. Hock, S. Csomor, *Cross-Platform GUI Programming with wxWidgets*, Pearson Education, 2006, ISBN 0-13-147381-6
- [5]. Qt Group, Dokumentacja *QStyle Class*, <https://doc.qt.io/qt-6/qstyle.html>, Dostęp 15.05.2026
- [6]. Qt Group, Dokumentacja *The Meta-Object System*, <https://doc.qt.io/qt-6/metaobjects.html>, Dostęp 15.05.2026
- [7]. Qt Group, Dokumentacja *Qt Widgets*, <https://doc.qt.io/qt-6/qtwidgets-index.html>, Dostęp 15.05.2026
- [8]. Qt Group, Dokumentacja *Qt Quick*, <https://doc.qt.io/qt-6/qtquick-index.html>, Dostęp 15.05.2026
- [9]. wxWidgets Team, *wxEvtHandler Class Reference*, https://docs.wxwidgets.org/3.2.5/classwx_evt_handler.html, Dostęp 15.05.2026
- [10]. wxWidgets Team, *Licence*, <https://wxwidgets.org/about/licence/>, Dostęp 15.05.2026
- [11]. O. Cornut, *Dear ImGui, README*, <https://github.com/ocornut/imgui>, Dostęp 15.05.2026
- [12]. GTK Development Team, Dokumentacja *Type System Concepts*, <https://docs.gtk.org/gobject/concepts.html>, Dostęp 15.05.2026
- [13]. GTK Development Team, Dokumentacja *Gtk – 4.0*, <https://docs.gtk.org/gtk4/>, Dostęp 15.05.2026
- [14]. GTK Development Team, Dokumentacja *CssProvider*, <https://docs.gtk.org/gtk4/class.CssProvider.html>, Dostęp 15.05.2026
- [15]. wxWiki, *WxWidgets Compared To Other Toolkits*, https://wiki.wxwidgets.org/WxWidgets_Compared_To_Other_Toolkits, Dostęp 16.05.2026

- [16]. eSOM++ Team, *Qt Vs GTK+ Vs WxWidgets – A Comparative Study*, <https://www.e-consystems.com/blog/system-on-module-som/qt-vs-gtk-vs-wxwidgets-a-comparative-study/>, Dostęp 16.05.2026
- [17]. Baeldung on Linux, *Differences Between GTK+ and Qt Applications*, <https://www.baeldung.com/linux/gui-toolkits-gtk-qt>, Dostęp 16.05.2026
- [18]. Philippe M. Groarke, C++ UI Libraries, https://philippegroarke.com/posts/2018/c++_ui_solutions/, Dostęp 16.05.2026
- [19]. Baeldung on CS, *Static vs. Dynamic Linking*, <https://www.baeldung.com/cs/static-dynamic-linking-differences>, Dostęp 18.05.2026
- [20]. JetBrains, *The State of C++ 2025*, <https://lp.jetbrains.com/the-state-of-cpp-2025/>, Dostęp 20.05.2026
- [21]. SQLite, *About SQLite*, <https://www.sqlite.org/about.html>, Dostęp 21.05.2026
- [22]. Qt Group, Dokumentacja *Qt Charts*, <https://doc.qt.io/qt-6/qtcharts-index.html>, Dostęp 21.05.2026
- [23]. SDL2 Wiki, *What is it*, <https://wiki.libsdl.org/SDL2/FrontPage>, Dostęp 21.05.2026
- [24]. The Khronos Group, OpenGL - The Industry's Foundation for High Performance Graphics, <https://www.khronos.org/opengl/>, Dostęp 22.05.2026
- [25]. Microsoft, *What is Visual Studio?*, <https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide>, Dostęp 22.05.2026
- [26]. Microsoft, *IntelliSense in Visual Studio*, <https://learn.microsoft.com/en-us/visualstudio/ide/using-intellisense>, Dostęp 23.05.2026
- [27]. wxWidgets Team, *wxBitmap Class Reference*, https://docs.wxwidgets.org/3.3/classwx_bitmap.html, Dostęp 06.06.2026
- [28]. O. Cornut, *Image Loading and Displaying Examples*, <https://github.com/ocornut/imgui/wiki/Image-Loading-and-Displaying-Examples>, Dostęp 09.06.2026
- [29]. GTK Development Team, *Setting up GTK for Windows*, <https://www.gtk.org/docs/installations/windows>, Dostęp 16.06.2026
- [30]. Qt Group, Dokumentacja *Qt Examples And Tutorials*, <https://doc.qt.io/qt-6/qtexamplesandtutorials.html>, Dostęp 16.06.2026
- [31]. Stack Overflow, Pytania z tagiem [qt], <https://stackoverflow.com/questions/tagged/qt>, Dostęp 17.06.2026

- [32]. wxWidgets Team, *Adding wxWidgets class documentation*, <https://github.com/wxWidgets/wxWidgets/blob/master/docs/contributing/how-to-add-class-documentation.md>, Dostęp 17.06.2026
- [33]. wxWidgets Team, *Samples Overview*, https://docs.wxwidgets.org/stable/page_samples.html, Dostęp 17.06.2026
- [34]. Microsoft, *Working Set*, <https://learn.microsoft.com/en-us/windows/win32/memory/working-set>, Dostęp 21.06.2026
- [35]. Microsoft, *Process.PrivateMemorySize64 Property*, <https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.process.privatememorysize64>, Dostęp 21.06.2026
- [36]. B. Stroustrup, *Bjarne Stroustrup's FAQ*, https://www.stroustrup.com/bs_faq.html, Dostęp 24.06.2026
- [37]. Cppreference, *RAII*, <https://en.cppreference.com/cpp/language/raii>, Dostęp 24.06.2026
- [38]. Kitware, *cmake-generators(7)*, <https://cmake.org/cmake/help/latest/manual/cmake-generators.7.html>, Dostęp 24.06.2026
- [39]. Conan, *Welcome to Conan C/C++ Package Manager Documentation*, <https://docs.conan.io/1/>, Dostęp 24.06.2026
- [40]. S. T. Barrett, *stb single-file public domain libraries for C/C++*, <https://github.com/nothings/stb>, Dostęp 24.06.2026
- [41]. Qt Group, *Qt Creator Documentation Overview*, <https://doc.qt.io/qtcreator/creator-overview.html>, Dostęp 24.06.2026
- [42]. Qt Group, *Dokumentacja User Interface Compiler (uic)*, <https://doc.qt.io/qt-6/uic.html>, Dostęp 24.06.2026
- [43]. TIOBE Software BV, *TIOBE Index for June 2026*, <https://www.tiobe.com/tiobe-index/>, Dostęp 28.06.2026
- [44]. Qt Group, *Qt Framework*, <https://www.qt.io/development/qt-framework>, Dostęp 28.06.2026
- [45]. GTK Development Team, *GTK*, <https://www.gtk.org/>, Dostęp 28.06.2026
- [46]. GIMP, *The Early Days of GIMP*, <https://docs.gimp.org/2.10/en/gimp-introduction-history-early-days.html>, Dostęp 28.06.2026
- [47]. Conan, *Conan - The open-source C and C++ package manager*, <https://github.com/conan-io/conan>, Dostęp 28.06.2026
- [48]. SQLite, *Home*, <https://sqlite.org/>, Dostęp 28.06.2026

- [49]. Qt Group, *Why Does Qt Use Moc for Signals and Slots?*, <https://doc.qt.io/qt-6/why-moc.html>, Dostęp 28.06.2026
- [50]. Qt Group, *Using the Meta-Object Compiler (moc)*, <https://doc.qt.io/qt-6/moc.html>, Dostęp 29.06.2026

Spis rysunków

Rysunek 1. Wizualizacja trybu zachowanego. Źródło: [1]	8
Rysunek 2. Wizualizacja trybu natychmiastowego. Źródło: [1].....	9
Rysunek 3. Proces budowania i linkowania aplikacji. Źródło: [19]	19
Rysunek 4. Okno Visual Studio 2026	24
Rysunek 5. Qt Designer w Qt Creator otwarty z plikiem prototypu.....	25
Rysunek 6. Okno główne prototypu Qt uruchomionego w systemie Windows	32
Rysunek 7. Okno dialogowe korekcji jasności i kontrastu w Qt	36
Rysunek 8. Okno dialogowe Qt z metadanymi i histogramem RGB obrazu.....	36
Rysunek 9. Natywne okno drukowania systemu Windows	39
Rysunek 10. Okno główne prototypu wxWidgets uruchomionego w systemie Windows	40
Rysunek 11. Standardowe okno komunikatu wxWidgets prezentujące metadane obrazu	43
Rysunek 12. Okno główne prototypu Dear ImGui uruchomionego w systemie Windows	45
Rysunek 13. Okno główne prototypu GTK uruchomionego w systemie Windows	50
Rysunek 14. Okno główne prototypu wxWidgets uruchomionego w systemie Linux	61
Rysunek 15. Okno główne prototypu Qt uruchomionego w systemie Linux	62

Spis tabel

Tabela 1. Zależności backendów	29
Tabela 2. Specyfikacja środowiska testowego	54
Tabela 3. Średni czas pojedynczego cyklu renderowania klatki obrazu.....	55
Tabela 4. Średni czas wczytania obrazu.....	55
Tabela 5. Zużycie procesora w stanie spoczynku	56
Tabela 6. Zużycie pamięci operacyjnej w stanie spoczynku	57
Tabela 7. Rozmiar plików binarnych i zależności poszczególnych wariantów	58
Tabela 8. Czas kompilacji prototypu.....	59
Tabela 9. Zestawienie punktowe wyników porównania	64

Spis listingów

Listing 1. Kod źródłowy interfejsu IGui	26
Listing 2. Kod źródłowy punktu wejścia prototypu	27
Listing 3. Kod źródłowy funkcji konfiguracyjnej pliku Conan	28
Listing 4. Polecenie SQL tworzące tabelę w bazie danych.....	31
Listing 5. Połączenie sygnału przycisku obrotu z metodą	34
Listing 6. Funkcja dopasowująca rozmiar obrazu do okna	35
Listing 7. Kod źródłowy iteracji po każdym pikselu w Qt	37
Listing 8. Dodanie rekordu z użyciem QSqlDatabase	38
Listing 9. Użycie obiektów rozmieszczających w wxWidgets	41
Listing 10. Funkcja rysująca obraz w wariancie wxWidgets.....	42
Listing 11. Usunięcie rekordu z użyciem biblioteki SQLite3	44
Listing 12. Element rysowany i zwracający stan w jednym wywołaniu	46
Listing 13. Wczytanie obrazu biblioteką STB i utworzenie tekstury OpenGL	48
Listing 14. Kod źródłowy obrotu przez współrzędne UV	49
Listing 15. Połączenie sygnału GTK z metodą klasy przez statyczną funkcję zwrrotną	52