POLITECHNIKA ŚLĄSKA

WYDZIAŁ INŻYNIERII MATERIAŁOWEJ

Kierunek: Informatyka Przemysłowa  
Profil praktyczny  
Rodzaj studiów: stacjonarne I stopnia

Projekt inżynierski  
Krzysztof MOCHOCKI

projekt i implementacja  
Komparatora Baz DANYCH  
z PublikacjAMI NaukowyMI

Design and implementation of  
comparator for databases   
with science publications

|  |  |
| --- | --- |
| Kierujący pracą | Recenzent |
| Dr. inż. Adrian SMAGÓR | Dr. inż. Łukasz MALIŃSKI |

Katowice, czerwiec 2021 r.

# Spis treści

[1. Spis treści 3](#_Toc73817642)

[2. Wstęp 5](#_Toc73817643)

[2.1. Inspiracja 5](#_Toc73817644)

[2.2. Cel 5](#_Toc73817645)

[3. Zakres pracy 6](#_Toc73817646)

[3.1. Przegląd literaturowo - technologiczny 6](#_Toc73817647)

[3.1.1. Podobne rozwiązania 6](#_Toc73817648)

[3.1.2. Aktualny model synchronizacji 6](#_Toc73817649)

[3.1.3. Dostępne źródła wiedzy w alternatywnych serwisach 8](#_Toc73817650)

[3.2. Przegląd wybranych technologii 10](#_Toc73817651)

[3.2.1. System operacyjny 10](#_Toc73817652)

[3.2.2. Język programowania 11](#_Toc73817653)

[3.2.3. Styl 11](#_Toc73817654)

[3.2.4. System kontroli wersji 12](#_Toc73817655)

[3.2.5. Dokumentacja 12](#_Toc73817656)

[3.2.6. Statyczna analiza kodu 14](#_Toc73817657)

[3.2.7. System budowy 14](#_Toc73817658)

[3.2.8. Środowisko deweloperskie 15](#_Toc73817659)

[3.2.9. Wykorzystane biblioteki 15](#_Toc73817660)

[3.3. Część projektowa 18](#_Toc73817661)

[3.3.1. Założeńia projektowe 18](#_Toc73817662)

[3.4. Realizacja 20](#_Toc73817663)

[3.4.1. Interfejs użytkownika 20](#_Toc73817664)

[3.5. Pobór danych 26](#_Toc73817665)

[3.5.1. Baza DOROBEK 26](#_Toc73817666)

[3.5.2. Bazy ORCID oraz SCOPUS 30](#_Toc73817667)

[3.6. Unifikacja 33](#_Toc73817668)

[3.6.1. Reprezentacja obiektowa 34](#_Toc73817669)

[3.6.2. Generowanie obiektów wyższej abstrakcji 35](#_Toc73817670)

[3.7. Dopasowywanie 40](#_Toc73817671)

[3.8. Nadzorowanie przebiegu 42](#_Toc73817672)

[3.8.1. Zrównoleglenie przetwarzania danych 43](#_Toc73817673)

[3.8.2. Generowanie raportu 45](#_Toc73817674)

[3.9. Omówienie serializacji 46](#_Toc73817675)

[3.9.1. Zasada działania 46](#_Toc73817676)

[3.9.2. Problematyka 49](#_Toc73817677)

[3.9.3. Wady i zaley 49](#_Toc73817678)

[3.10. Narzędzia i produkty uboczne 51](#_Toc73817679)

[3.10.1. Generowanie bibliotek 51](#_Toc73817680)

[3.10.2. Asysta z systemu kontroli budowy 52](#_Toc73817681)

[4. Podsumowanie 54](#_Toc73817682)

[4.1. Perspektywy rozwoju 54](#_Toc73817683)

[4.2. Wnioski 54](#_Toc73817684)

[5. Bibliografia 55](#_Toc73817685)

[6. Spis ilustracji 58](#_Toc73817686)

# Wstęp

## Inspiracja

Inspiracją dla niniejszego projektu inżynierskiego było dostrzeżenie problemu dotykającego pracowników naukowych Politechniki Śląskiej, którzy, aby móc dzielić się wynikami swoich prac z resztą świata nauki, zmuszeni są eksponować swój dorobek w różnych agregatach publikacji naukowych. Niestety w związku z różną datą pojawienia się agregatorów, oraz natłokiem innych zadań, ciężko jest dopilnować, aby wszystkie serwisy posiadały wszystkie publikacje.

## Cel

Celem projektu jest dostarczenie narzędzia pozwalającego zautomatyzować cały proces wyszukiwania brakujących publikacji wśród znanych i obsługiwanych serwisów.

Wśród obsługiwanych portali, powinny znaleźć się te, które posiadają najwięcej zbiorów, oraz te, które cieszą się sporą popularnością. Dodatkowym kryterium jest możliwość dodania lub edycji danych w przeszukiwanym przez aplikację portalu. Przy założonych kryteriach wyłaniają się 4 portale:

* DOROBEK (1)
* ORCID (2)
* SCOPUS (3)
* Web Of Science (4)

Docelowo program, ma generować prosty w interpretacji raport w powszechnie znanym formacie, pozwalający na jednoznaczne wskazanie, zbiorów publikacji, które należy uzupełnić w poszczególnym portalu internetowym. Formatem spełniającym te warunki oraz dodatkowo zapewniający przenaszalność i wiele narzędzi do jego obsługi, jest otwarty format opracowany przez firmę Microsoft dla arkuszy kalkulacyjnych: xlsx (5). Biorąc pod uwagę możliwość pojawienia się nowych podobnych serwisów internetowych, projekt aplikacji powinien, również zapewniać łatwą rozszerzalność o nowe strony.

# Zakres pracy

## Przegląd literaturowo - technologiczny

### Podobne rozwiązania

W domenie publicznej niestety nie udało się znaleźć podobnych rozwiązań, co może być zrozumiałe z uwagi na charakter korzystania z tego typu serwisów. Wnioskując na podstawie dokumentacji, dostarczonej przez wyżej wymienione portale, programowalny interfejs jest udostępniany do integracji z wewnętrznymi zasobami ośrodków naukowych.

Zasoby tego typu, jeżeli będą otwarte, to nie będą skoncentrowane na twórczości pracowników Politechniki Śląskiej, lecz na swoich naukowcach. Fakt ten uniemożliwia korzystanie z potencjalnych portali, jako narzędzia do diagnozowania luk we wspieranych portalach, dla obcych badaczy.

### Aktualny model synchronizacji

W momencie rozpoczęcia prac brakowało narzędzi pozwalających na sprawną synchronizację publikacji naukowych pracowników Politechniki Śląskiej   
w internetowych bazach publikacji, takich jak: DOROBEK, SCOPUS, czy ORCID, ponieważ narzędzia, które miały im to umożliwić nie zapewniały wystarczających możliwości.

Dla pracowników Politechniki Śląskiej, punktem wyjścia ręcznego porównania spisów swoich publikacji w internetowych bazach, jest baza DOROBEK. Baza ta powinna zawierać wszystkie publikacje, pracowników Politechniki Śląskiej. Aby możliwe było w prosty i systematyczny sposób porównywać dane pomiędzy portalami potrzebna jest funkcjonalność prostego zbierania danych, czego baza DOROBEK nie zapewnia.

Naturalnym podejściem osoby niepotrafiącej programować, byłoby ręczne kopiowanie tytułów i wklejane ich do innych wyszukiwarek, celem weryfikacji ich istnienia. Temu podejściu nie sprzyja jednak mocno dynamiczny wygląd strony, który uniemożliwia sprawne zaznaczanie, interesującego ciągu znaków kursorem.

Alternatywą dla pierwszego sposobu mogłoby być skorzystanie z generacji pliku csv (6), który jednak jest mocno niekompletny, między innymi przez brakujące numery ORCID, znacznie ułatwiające wyszukiwanie, a dane zarówno w nim, jak i na stronie są nierzadko zapisane w nieprzystępnej formie.

Często widywaną praktyką jest wpisywanie tytułu w języku polskim   
i nowożytnym po przecinku, średniku lub kropce, co generuje kolejny problem. Problem powodujący niejednoznaczność w dalszych poszukiwaniach, ponieważ pojawiają się dwie różne wartości dla jednego nie tablicowego pola. Strona, oprócz pliku w formacie csv oferuje również generowanie pliku w formacie rtf, który względem poprzedniego pliku ma przewagę w postaci kompletnych danych.

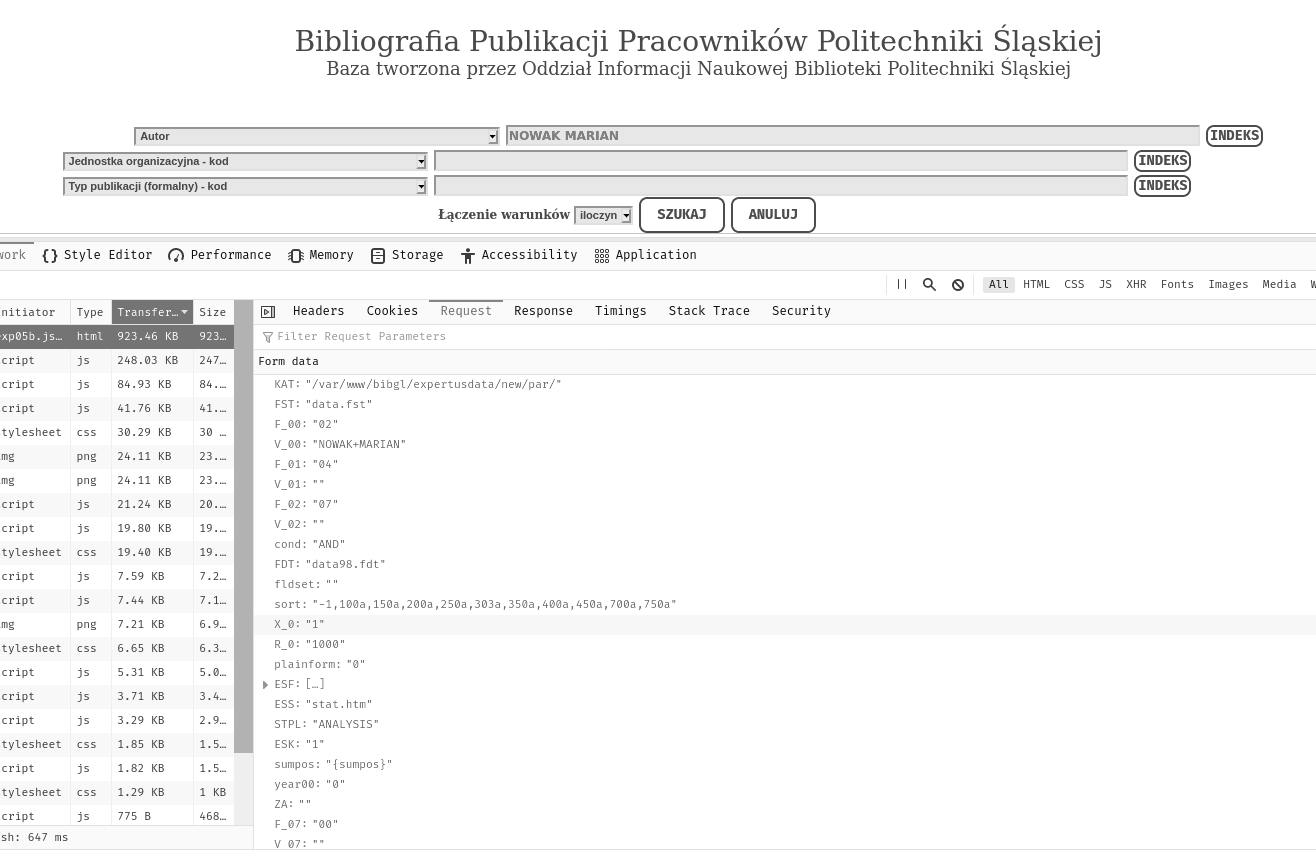
Największe trudności związane z automatycznym poborem danych, z bazy DOROBEK jest brak możliwości zdalnego generowania plików csv i rtf, ponieważ są one generowane przez skrypty klienckie, co wymusza przetwarzanie gołego kodu strony w języku HTML wraz ze wszystkimi stylami i definicjami skryptów klienckich.

Domyślną postawą podczas komunikacji strony internetowej wraz z bazą danych, jest korzystanie z programowalnego interfejsu aplikacji   
(ang. API – Application Programmable Interface), za pomocą protokołu REST, czy GRPC. Można przypuszczać, że w przypadku tego serwisu, również zachodzi taka wymiana, co potwierdza szybka analiza ruchu sieciowego strony. Skorzystanie jednak z wyłonionego zapytania, poprzez wyszukanie największego typu metody POST (Rysunek 1), również nie napawa optymizmem, ponieważ zapytanie to, zwraca kawałek strony internetowej wygenerowanej po stronie serwera.

Ponadto zapytanie zbudowane przez stronę, wygląda na pozór na zaszyfrowane (Rysunek 1), ponieważ w rzeczywistości, są to trudno identyfikowalne nazwy wprowadzonych pól. Stosowanie takiej praktyki sprawia, że wykorzystanie tego zapytania przez zewnętrzny serwis wymaga zarówno wcześniejszego zgadywania przez programistę za pomocą prób i błędów znaczenia poszczególnych pól formularza, a także żmudnego i wymagającego sporych zasobów przetwarzania dostarczonych danych w formacie HTML.

Wielokrotne próby komunikacji z administratorem strony zakończyły się kategoryczną odmową dostępu do jakiegokolwiek punktu dostępowego dla osób trzecich, co wymusza na osobie zainteresowanej automatyzacją procesu weryfikacji spójności danych w innych serwisach na skorzystanie z przetwarzanie danych uzyskanych ze źródła przeznaczonego dla użytkownika przeglądarki internetowej.

Kolejne serwisy, między innymi ORCID, oraz SCOPUS zapewniają bardzo obszerne dokumentacje, z przykładami oraz dokładnymi objaśnieniami,   
w przeciwieństwie do bazy DOROBEK są przyjaźnie nastawione do użytkownika   
i oferują zwięzły panel użytkownik.

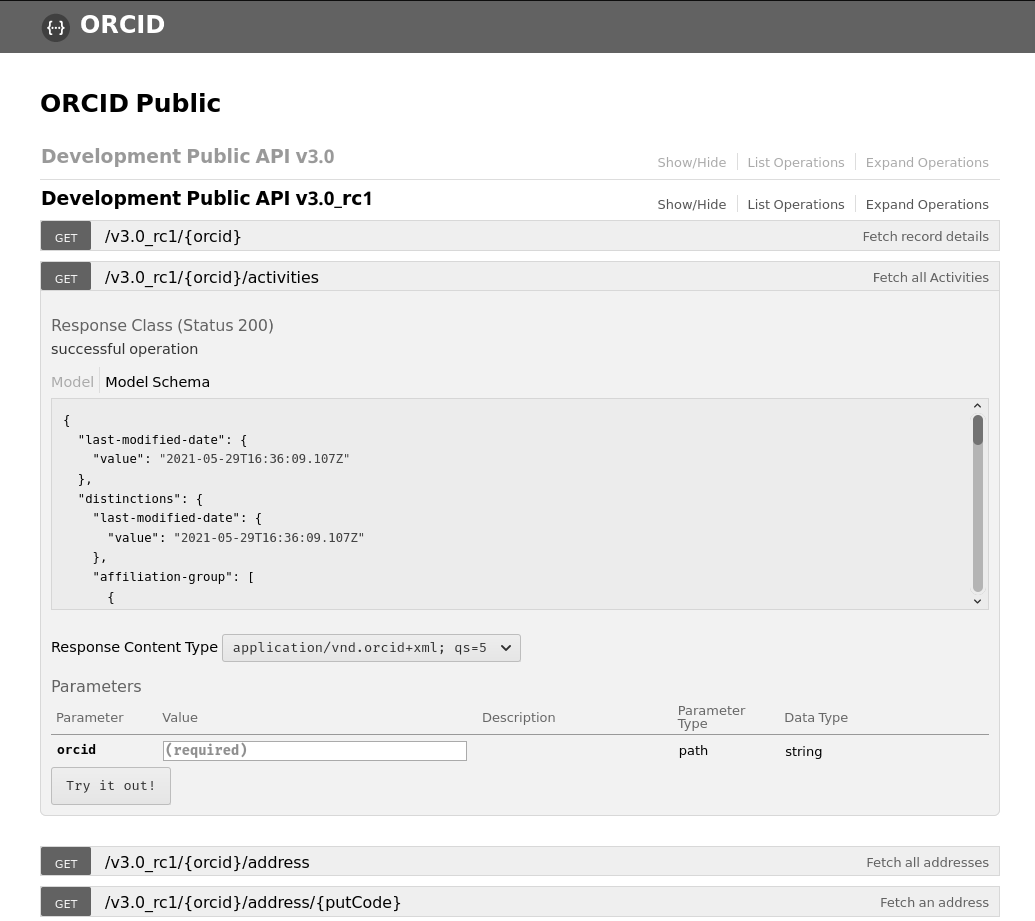


Rysunek 1 Zapytanie na jednym polu - autorze, wygenerowane przez serwis (skala szarości)

### Dostępne źródła wiedzy w alternatywnych serwisach

#### ORCID

Serwis ORCID zapewnia programiście szerokie wsparcie w zakresie obsługi dostarczonych przez serwis narzędzi. Przekonać można się o tym, logując się na stronie, a następnie przechodząc do zakładki narzędzie deweloperskich   
(ang. Developer Tools), która znajduje się w poręcznym miejscu przy ikonie użytkownika w prawym górnym rogu. Po przejściu na witrynę deweloperską, strona od razu przedstawia przydatne hiperłącza, kierujące zainteresowanego do obszernej dokumentacji wystawionego interfejsu sieciowego. Istnieje możliwość, zarejestrowania swojej aplikacji. Daje to możliwość wykorzystania z jednego   
z dostępnych protokołów podwyższających bezpieczeństwo komunikacji: OAuth lub OAuth 2.0 (6). Skorzystanie z bezpieczniejszego połączenia jest wymagane przez serwis, aby móc przeprowadzić proces logowanie i wykonywać operacje na koncie poza stroną internetową.

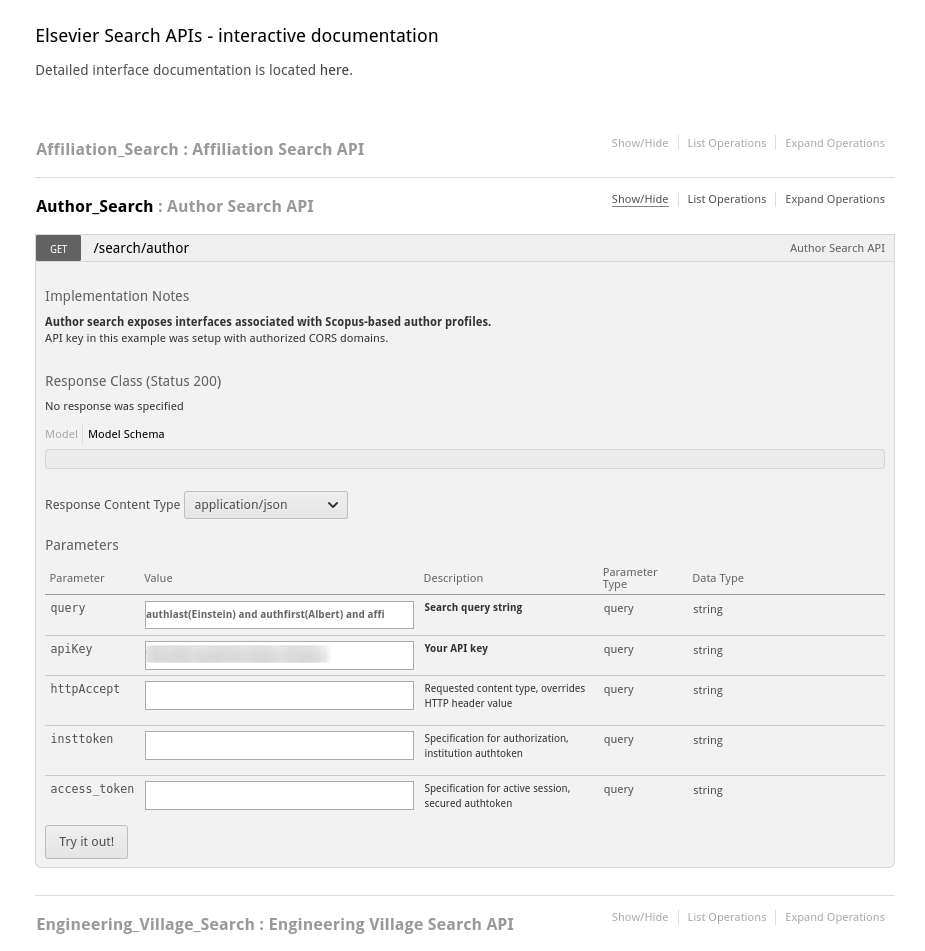


Rysunek 2 Podgląd specyfikacji technicznej dostarczonej przez serwis ORCID (skala szarości)

Dzięki wykorzystaniu otwartego i sprawdzonego standardu tworzenia dokumentacji Swagger (7), (Rysunek 2) dokumentacja interfejsu sieciowego czyta się szybko z uwagi na znany format. Dodatkowo twórcy umożliwili przetestowanie punktów końcowych interfejsu, co umożliwia programiście zaplanowanie sposobu przetwarzania przychodzących danych. Kolejną istotną rzeczą jest wersjonowanie punktów końcowych. Daje to programiście spore poczucie bezpieczeństwa, że model przychodzących danych lub wygląd zapytania, nie zostanie nagle zmieniony. Polityka serwisu polega na dodaniu kolejnej wersji interfejsu sieciowego w przypadku zmiany czegokolwiek w budowie zapytań lub w modelu przychodzących danych.

#### SCOPUS

Serwis SCOPUS za wyjątkiem wersjonowania dostarcza te same równie przyjemną interaktywną dokumentację, co serwis ORCID. Jednakże w porównaniu do poprzedniego zapewnia konkretne przypadki użycia swojego interfejsu   
z wykorzystaniem różnych języków, między innymiPython, czy *JavaScript*. Ponadto interaktywna dokumentacja generuje od razu polecenia cURL, które można wkleić do linii poleceń w systemie GNU/Linux i szybko sprawdzić sposób działania dostępnych punktów końcowych.



Rysunek 3 Podgląd specyfikacji technicznej dostarczonej przez serwis SCOPUS (skala szarości)

## Przegląd wybranych technologii

### System operacyjny

Manjaro (8) – dystrybucja systemu operacyjnego GNU/Linux bazująca na dystrybucji Arch Linux (9), który słynie ze ścisłego trzymania się reguły KISS   
(ang. Keep It Simple, Stupid). Pośrednio wynika z tej zasady, jak wygląda zarządzanie pakietami w tej dystrybucji. Mimo istnienia menadżera pakietów pacman, spora część oprogramowania wymaga ręcznego budowania przez użytkownika na swoim urządzeniu. Wymóg ten miał spory wpływ na wybór zależności do samego programu, ponieważ wybór bibliotek dedykowanych pod aktualny system jest bardzo mały. Wymusza to na programiście wybór bibliotek   
i technologii, które są możliwe do zastosowania również na nietypowych systemach oraz cechujące się wysoką przenaszalnością. Implikuje to poprawienie przenaszalności całego projektu na różne platformy.

### Język programowania

C++ (10) – elastyczny, szybki i wydajny język programowania, dający programiście dużo możliwości oraz stawiający sporo wyzwań podczas pisania w nim. Sam, spadkobierca języka C, jest dość stary i posiada wiele wersji, które, szczególnie te współczesne, mocno się od siebie różnią, stąd konieczność na wstępie jej sprecyzowania. Niniejszy projekt używa najnowszej wersji – 20, oznaczanej, jako C++20, lub C++2a w przypadku wydań eksperymentalnych

Jednakże wersja języka nie jest jedyną rzeczą konieczną do sprecyzowania przy jego omawianiu. Konieczny jest jeszcze wybór kompilatora, który będzie zamieniał go na język maszynowy. Na wybranej platformie, dostępne są dwa popularne kompilatory: gcc oraz clang. Ostatecznym czynnikiem mającym wpływ na wybór kompilatora było wcześniejsze doświadczenia twórcy w pracy z gcc.

Jednym z wielu argumentów za wyborem tego języka jest jego wyżej wspomniana elastyczność. Jest to jeden z niewielu języków dający możliwość korzystania z wielu paradygmatów. Program tutaj opisywany, nie skupia się na kurczowym trzymaniu się jednego z nich, lecz oscyluje wokół trzech: obiektowego, funkcyjnego oraz [szablonowego] meta-programowania.

### Styl

*Notacja wężowa* (ang. sneak case) (11) – zaraz obok wielbłądziej (12) jeden   
z najbardziej rozpoznawalnych stylów używanych w kodzie. Wyborem stojącym za skorzystaniem z tej notacji jest chęć dostosowania się do tej, dostarczonej z biblioteki standardowej. Jedynym odstępstwem od jej ścisłego przestrzegania, są nazwy argumentów w szablonach, gdzie również w bibliotece standardowej można spotkać się z notacją wielbłądzią.

Oprócz samej notacji warto jest zadbać o jednolite formatowanie w całym projekcie, co poprawia czytelność i zwiększa wrażenie spójności wszystkich jego elementów. Celem zapewnienia zgodnego formatowania, należało zdecydować się na dedykowane, oprogramowanie. Najpopularniejszym, otwartym, rozwiązaniem jest program z pakietu clang o nazwie *clang-tidy* (13). Konfiguracja jest bardzo prosta, oparta na pojedynczym pliku tekstowym. Po uruchomieniu, program rekurencyjnie wyszukuje pliki według zadanego wyrażenia regularnego, a następnie podmienia na sformatowane.

### System kontroli wersji

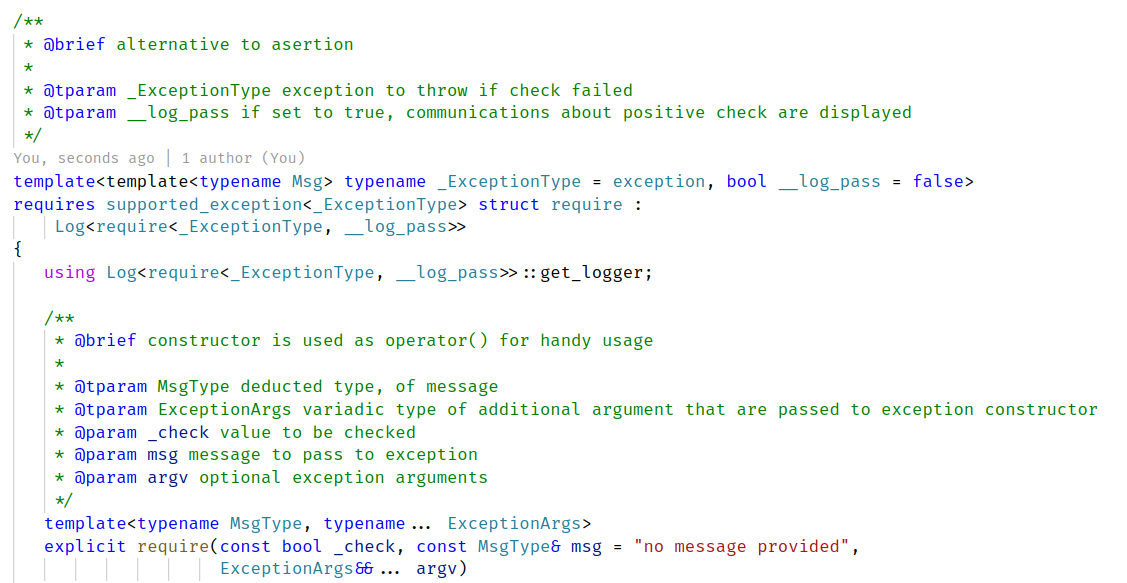
git (14) – popularny program, służący do kontroli wersji. Przy bardziej złożonych projektach jest to narzędzie obowiązkowe, z uwagi na łatwą możliwość powrotu do poprzednich wersji, możliwość łatwej współpracy z wieloma osobami, a także łatwą kontrolę zależności w postaci podmodułów. W połączeniu z serwisem GitHub, służącym, jako zdalne repozytorium, zapewnia bezpieczny ekosystem.

### Dokumentacja

Doxygen (15) (16) – narzędzie do automatycznego generowania dokumentacji w formie strony internetowej, plików w formacie *man* lub *rtf*. Korzyścią tego rozwiązania jest brak konieczności ręcznego tworzenia dokumentacji w osobnych plikach. Cała treść dokumentacji jest zapisana w formie komentarzy w całym kodzie źródłowym. Dodatkowo, wbudowane programy, do wspomagania programisty, takie jak IntelliSense korzystają z tych komentarzy, aby móc wyświetlić pomocne informacje, podczas tworzenia nowego kodu.

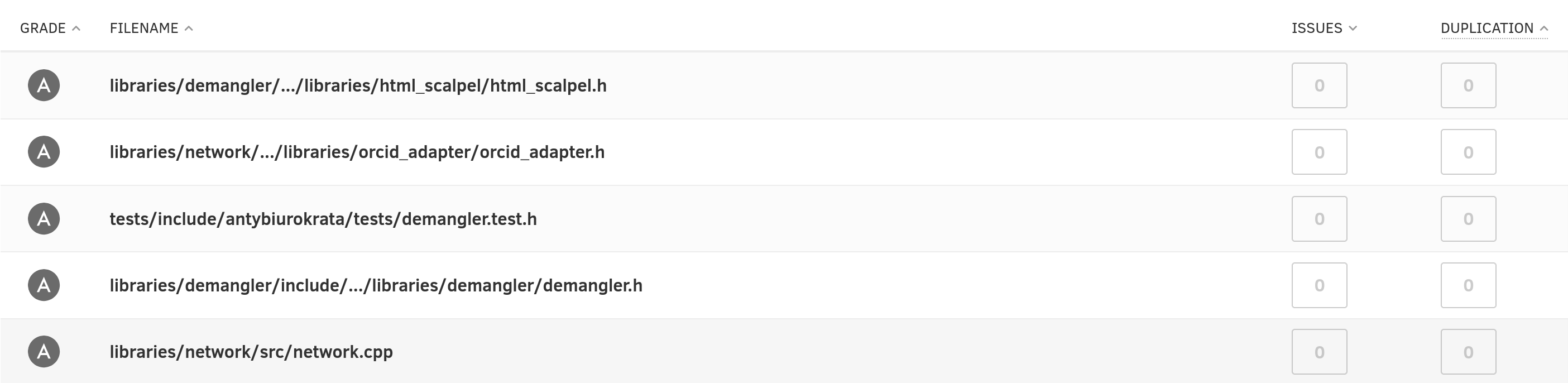


Rysunek 4 Fragment wygenerowanej strony internetowej



Rysunek 5 Fragment kodu z komentarzami do wygenerowania dokumentacji

### Statyczna analiza kodu



Rysunek 6 Raport statycznej analizy kodu (skala szarości)

codacy (16) – serwis internetowy świadczący usługi oceny kodu pod kątem użytych wzorców, bezpieczeństwa, redundancji kodu, czy stosowania niedozwolonych praktyk programistycznych. Automatycznie wykrywa użyte języki, dokonuje ich statycznej analizy, a następnie generuje raport, informujący o wykrytych błędach.

Dodatkowo serwis daje możliwość skorzystania z generowanej etykiety, możliwej do umieszczenia w pliku README.md, która zachęca innych użytkowników serwisu *GitHub* do uczestnictwa w dobrze napisanym kodzie.

### System budowy

cmake (17) – otwarte, wieloplatformowe narzędzie do kontroli przebiegu budowy projektu oraz planowania testów. Oprogramowanie wykonuje instrukcje na podstawie przygotowanych plików, napisanych w dedykowanym języku skryptowym, za pomocą, którego można definiować cele do zbudowania w postaci bibliotek oraz plików wykonywalnych.

Dostarczony język daje również możliwość tworzenia niestandardowych celów budowy, co daje spore możliwości w zakresie tworzenia narzędzi wspomagających proces budowy, czy proces wytwarzania kodu.

W trakcie tworzenia niniejszej aplikacji została wydana nowsza wersja programu cmake – 3.20. Aplikacja natomiast używa wersji 3.19.

### Środowisko deweloperskie

Visual Studio Code (18) – otwarty, wieloplatformowy edytor tekstowy zawierający zestaw narzędzi wspierających programistę. Jego mocnymi stronami jest ilość wtyczek dostępnych bezpośrednio z edytora, tworzonych przez społeczność. Dzięki nim edytor zapewnia wsparcie nawet najbardziej egzotycznym językom, zarówno   
w zakresie podpowiadania składni, debugowania, kontroli wersji czy budowania   
i uruchamiania.

### Wykorzystane biblioteki

#### Wspierająco-narzędziowa

boost (19) – otwarta, wieloplatformowa biblioteka narzędziowa, lustrzana do biblioteki standardowej, jednocześnie rozszerzająca ją o wiele funkcjonalności.   
W przypadku większych projektów jest to, niemalże obowiązkowa pozycja na liście bibliotek.

Argumentem stojącym za wykorzystaniem tej biblioteki jest również powszechność jej użycia w różnych projektach, co sprawia, że najczęściej znajduje się już zainstalowana w systemie. Wadą tej biblioteki, jest wymagający próg wejścia, zadany przez autorów. Przy domyślnych ustawieniach biblioteka jest kompilowana   
w całości, co przy słabszych maszynach, może okazać dość długim procesem.

#### Testowa

boost-ext/μt (20) – eksperymentalna część wcześniej wspomnianej biblioteki boost, zawierająca pakiet narzędzi do tworzenia testów, wraz z raportowaniem. Biblioteka jest stworzona w zgodnie ze standardem C++17, dając możliwość pisania testów korzystając z dobrodziejstw najnowszych rozwiązań biblioteki standardowej.

Biblioteka posiada możliwość organizacji kodu, dzięki zaimplementowanym zestawom (ang. suites). Wewnątrz jednego zestawu można umieścić różne konfiguracje aplikacji, a następnie napisać różne przypadki testowe (ang. test cases), celem przetestowania założonych ustawień. Ciekawe jest podejście autorów, do sposobu deklaracji testów, które polega na używaniu literałów ciągów znakowych.

W samych testach, jako sprawdzeń, używa się bibliotecznych funkcji. Są one statycznie rozwiązywalne, z uwagi na oparcie o szablony, przeciwieństwie do biblioteki matki, która to używa makr z języka C. Takie rozwiązanie powoduje wykrycie części błędów, jeszcze na etapie kompilacji.

#### Sieciowa

drogoncpp (21) – otwarta, sieciowa biblioteka napisana w nowoczesnym standardzie języka C++, dedykowana do pisania aplikacji serwerowych, jednakże równie dobrze sprawdza się, jako biblioteka kliencka.

Posiada pełne wsparcie dla rozwiązań wielowątkowych, a także posiada wsparcie dla korutyn. Bliskie trzymanie się biblioteki standardowej sprawia, że integracja z dostępnym interfejsem aplikacji jest bezproblemowa, a użytek intuicyjny. Użytkowanie biblioteki poprawia również dość szeroka dokumentacja, a także dostarczone przez społeczność, seria przykładów użycia

Wybierając tą bibliotekę konieczne jest również pobranie jej zależności,   
w postaci biblioteki o nazwie tantor. Biblioteka ta zawiera szereg kolekcji zsynchronizowanych oraz wiele narzędzi ułatwiających programowanie wielowątkowe. Instalacja jednak nie jest problematyczna, ponieważ występuje, jako pod moduł biblioteki drogoncpp i jest dociągana automatycznie.

Dodatkowo decydując się na zainstalowanie w systemie biblioteki drogoncpp, zostaje dodane narzędzie umożliwiające szybkie tworzenie projektów serwerowych. Narzędzie nazywa się drogon-cli. Okazało się ono bardzo pomocne w momencie tworzenia konfiguracji cmake. Był to istotny problem z uwagi na brak wpisów w dokumentacji biblioteki, jak należy podejść do tego zagadnienia. Rozwiązaniem okazało się wygenerowanie szablonowego projektu, a następnie użycie gotowych fragmentów plików wsadowych programu cmake.

#### Wspierająca proces logowania

rang (22) – otwarta, nagłówkowa biblioteka oferująca międzyplatformowe, strumieniowe wsparcie dla kolorów w konsoli. Jest to dość niedoceniony aspekt logowania, jednakże zważają na ludzką wrażliwość na kolory, jest to pomocna funkcjonalność, redukująca szansę pominięcia ważnego błędu, który pojawił się   
w logach. Dodatkowo biblioteka jest zaprojektowana, by ściśle współpracować ze strumieniami z biblioteki standardowej, co dodatkowo zwiększa komfort pracy.

#### Graficzna

Qt (23) – rozbudowana, platforma programistyczna (ang. framework (24)) graficzna, dedykowana dla języków C++ oraz Python. Zawiera dwojaki sposób opisu interfejsu użytkownika. Pierwszy, wykorzystany w tym projekcie, z wykorzystaniem formatu xml, który jest generowany za pomocą dołączonego edytora graficznego. Drugi, bardziej zaawansowany, wykorzystuje autorski język QML, który dobrze sprawdza się w przypadku rozwiązań wbudowanych, oraz mobilnych.

Do wersji 5.4 biblioteka oferowała również wsparcie dla wewnętrznego systemu kontroli budowy – qmake, jednakże wraz z najnowszą wersją 6.0 wsparcie to zostało zarzucone. Aktualnie rekomendowanym podejściem jest korzystanie   
z obecnego w tym projekcie nadzorcy budowy – cmake.

#### Obsługująca rozszerzenie XLSX

QtXlsxWriter (25) – otwarta, rozbudowana biblioteka, oferująca obsługę plików w formacie xlsx. Niestety została stworzona w momencie pełnego wsparcia dla systemu budowy qmake, który aktualnie jest zarzucany kosztem cmake   
w najnowszej wersji platformy graficznej Qt. Rozwiązanie dostarczył jeden   
z członków społeczności. Użytkownik *dand-oss* stworzył, jeszcze nierozstrzygniętą, prośbę dołączenia jego zmian. Są one znaczące dla tego projektu, ponieważ dodają pełne wsparcie dla systemu budowy cmake. Jest to również powód, dla którego ta biblioteka, jako pod moduł, nie korzysta z oficjalnego repozytorium autora biblioteki.

## Część projektowa

### Założeńia projektowe

#### Skalowalność

Jednym z głównych problemów, ujednolicenia stanu wszystkich baz jest ich mnogość. Ilość aktualnych źródeł jest spora, z uwagi na bardzo dynamiczny rozwój nauki w ostatnich dekadach można bezpiecznie założyć, że będą powstawać nowe źródła danych. Zarówno masowe (na przykład ORCID), czy mocno specjalistyczne (na przykład International Journal of Minerals, Metallurgy and Materials).

Prawidłowość ta, wymusza na aplikacjach chcących utrzymać wszystkie źródła w takim samym stanie, konieczność ciągłej rozbudowy. Stworzenie aplikacji, która będzie odpowiadać na takie wyzwanie, wymaga zaprojektowania oraz zaimplementowania odpowiedniej architektury.

Cechą aplikacji, która definiuje zdolność do rozszerzania, nazywa się skalowalnością.

#### Przyjazna rozwojowi

Rozwiązanie architektoniczne problemu skalowalności to podstawa pod przyszły rozwój aplikacji, bardzo ważny, lecz wciąż to tylko baza. Żeby rozwój przyszłych funkcjonalności aplikacji był sprawny i nie wymagał pełnej wtórnej analizy całego kodu, potrzebne jest stworzenie odpowiedniego środowiska w samym kodzie.

Czynnikiem, który ma zdecydowany wpływ na atmosferę w kodzie źródłowym jest utrzymana dokumentacja. Raport z Norweskiej Politechniki w Trondheim (26), wskazuje, że osoby posiadające wyłącznie kod źródłowy potrzebowały o 21,5% więcej czasu, aby zrozumieć analizowany program, niż ich koledzy, którzy dodatkowo posiadali dokumentację. Ponadto, jak wskazuje raport, zrozumienie źródeł w grupie z dostępną dokumentacją było statystycznie znaczące.

Kolejne przytoczone badanie (27), tym razem z osiemnastej konferencji na temat odwrotnej inżynierii z 2011 roku (ang. 2011 18th Working Conference on Reverse Engineering) potwierdzają również pozytywny wpływ diagramów UML na efektywność podczas pracy z wzorcami projektowymi.

#### Współczesny

Język C++ w wersji C++20 dodał sporo nowych rozwiązań językowych, które usprawniają proces wytwarzania kodu, oraz wpływają na ilość kodu generowanego podczas kompilacji. Dzieje się tak dzięki prężnemu rozwojowi meta programowania, w ramach biblioteki standardowej, jako bardzo popularnej metody wytwarzania oprogramowania, w ramach języka C++ w ciągu ostatnich paru lat. Jest to ważny krok w rozwoju języka, ponieważ uproszczenie programowania w paradygmacie meta programowania potrafi o wiele zwiększyć czytelność kodu oraz znacząco go uprościć.

Przed zmianami dokonanymi w standardach od C++11 do C++20, podejście do tego paradygmatu było dość mocno sceptyczne, szczególnie z uwagi na spore trudności w tworzeniu i rozwijaniu kodu. Dobrze określił to Herb Sutter, członek komisji standaryzacyjnej języka C++, w wywiadzie z 2011 roku dla kanału 9 (28), na prośbę o skomentowanie wyjścia, cztero-stronnicowego błędu z procesu kompilacji, odrzekł, że jest ono „barokowe”. Na domiar złego, współczesne narzędzia wspomagające programistów, często miewają problemy z pełnieniem swojej roli   
w podpowiadaniu składni, przy sporym zagnieżdżeniu szablonów, co dodatkowo utrudnia rozwój oprogramowania opartego o ten sposób pisania kodu.

Standard C++20 jest pod tym kątem rewolucyjny, ponieważ razem z nim   
w języku pojawiła się konstrukcja konceptów (29), które w wypadku błędu jasno wskazują na miejsce i powód błędu inscenizacji szablonu. W związku z między innymi, powyższymi zmianami, które są kluczowe w optymalizacji kodu, istnieje konieczność trzymania się najnowszych wersji języka C++.

Trend ten podzielają również autorzy bibliotek, co sprawia, że znalezienie biblioteki opartej o najnowsze rozwiązania zawarte w języku nie jest, aż tak wymagające. Spore zaangażowanie, ze strony obszernej społeczności sprawia, że niejednokrotnie była konieczność aktualizacji wersji biblioteki z uwagi na nowe wydanie.

Dodatkowo, z uwagi na wyżej wymienione możliwości, kod został pozbawiony wszelkich makr, uznanych w obliczu nowych możliwości języka C++, jako zaszłości języka C, utrudniające utrzymanie kodu oraz zmniejszające jego czytelność.

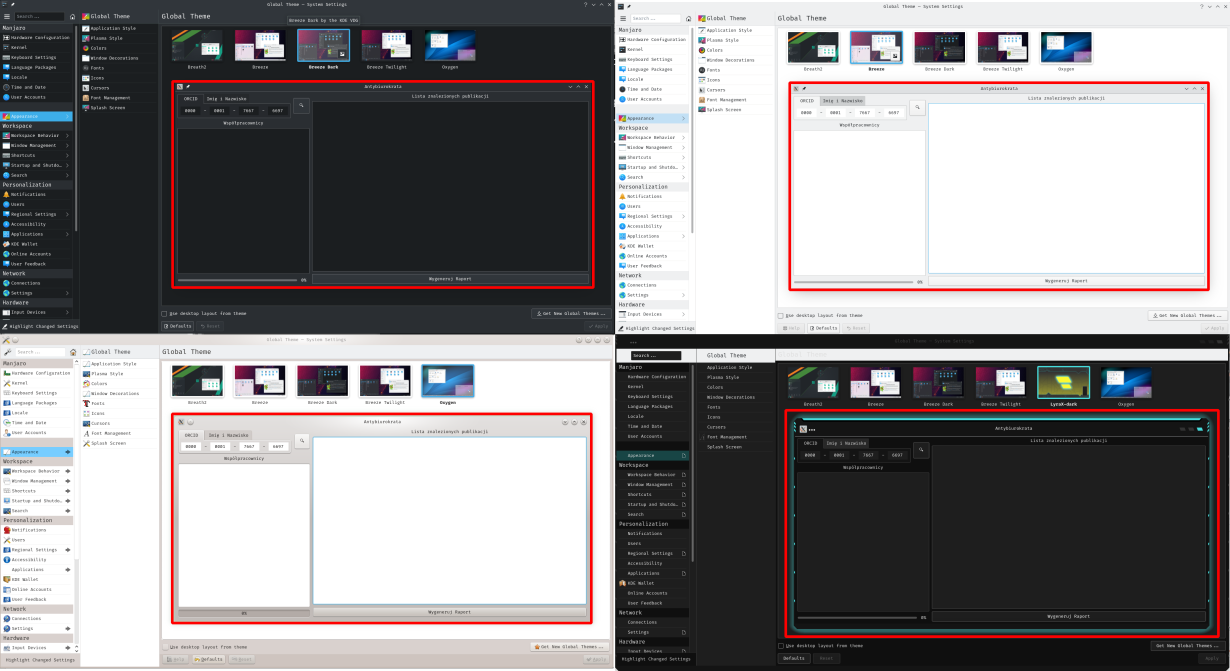
#### Przystępne wyjście

Zrozumiałe i przejrzyste dla użytkownika wyjście programu jest jedną   
z kluczowych kwestii. Szczególnie, że jednym z celów tej aplikacji, jest redukcja czasu potrzebnego na zrównanie wielu baz do tego samego stanu. Wyjście nie powinno być w formacie nieprzyjaznym użytkownikowi, lub formacie wymagającym płatnych programów do jego obsługi. Zmniejszyłoby to grono chętnych do skorzystania z aplikacji. Dodatkową cechą formatu powinna być jego przejrzystość, wynikająca z celu, jaki ma on zaspokajać. W tym przypadku, będzie to możliwość filtrowania, pod względem obecności danej publikacji w wybranej bazie. Możliwość posortowania danych, będzie miała niemały wpływ na przejrzystość raportu.

## Realizacja

### Interfejs użytkownika

Graficzna część aplikacji to zespół dwóch okien. Pierwsze główne, oraz drugie służące, jako dialogowe, do wyświetlania szczegółów. Dzięki wykorzystaniu środowiska Qt, interfejs graficzny, w przypadku uruchomieniu na systemie operacyjnym GNU/Linux z nakładką graficzną KDE, dostosowuje się stylem do wybranego przez użytkownika motywu systemu. Daje to wrażenie integracji   
z systemem oraz w przypadku wybrania motywu ciemnego nie jest to zagrożeniem dla oczu użytkownika. Działa również z zewnętrznymi motywami (Rysunek 7).



Rysunek 7 Porównanie wyglądu w zależności od wybranego motywu systemu

Standardem dzisiejszych czasów jest responsywność interfejsu użytkownika, wymusza to zrezygnowanie ze sztywnych rozmiarów poszczególnych elementów interfejsu, na rzecz dozwolonych przedziałów. Celem zachowania spójności   
w organizacji elementów graficznych, konieczne jest zastosowanie kontenerów oraz dystansów ograniczających względne położenie poszczególnych części interfejsu. Środowisko Qt zapewnia sporo różnych kontenerów, usprawniający cały proces, jednakże działanie części z dostarczonych nie jest deterministyczne, co wymusza projektowanie interfejsu w oparciu o serię prób i błędów.

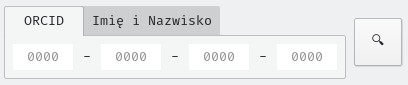
Interfejs użytkownika został zaprojektowany z zachodnio-europejskim modelem postrzegania kierunku ciągłości. Przepływ (ang. flow) obsługi aplikacji jest ułożony w kolejności z lewej do prawej, oraz z góry na dół. Model taki, został wybrany po zapoznaniu się ze wskazówkami firmy Microsoft na temat tworzenia interfejsu użytkownika (30). Stosując się również do tych zaleceń, została ułożona kolejność tabulacji, która umożliwia obsługę niemalże całego interfejsu wyłącznie za pomocą klawiatury. Wynikiem zastosowania się do wskazówek jest uformowanie się dwupodziału w aplikacji, oraz czterech stref, gdzie każda posiada konkretną funkcją   
i jest naturalnym przedłużeniem poprzedniej.

#### Panel wyszukiwania

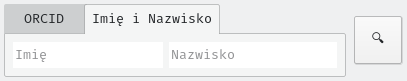
Pierwsza ćwiartka umiejscowiona w lewej górnej części głównego okna programu. Oprócz przycisku umiejscowionego po prawej stronie, posiada dwa tryby widoku, oba z polami na wejście od użytkownika. Domyślny widok (Rysunek 8), oczekuje od użytkownika zadanie numeru ORCID, który ma zostać poddany komparacji przez algorytm wewnętrzny aplikacji.

Wejście składa się z czterech pól, których uzupełnianie powoduje przechodzenie do kolejnych komórek. Mechanizm smukłego przechodzenia, nie jest funkcjonalnością oferowaną przez środowisko Qt i wymagała ręcznego zaimplementowania. Ta sama kwestia dotyczy walidacji wprowadzonych danych. Mimo istnienia w graficznym edytorze interfejsu użytkownika opcji, wymuszających na użytkowniku podanie wyłącznie liczb, pole mimo to przyjmuje dowolne znaki. Funkcjonalność automatycznego przechodzenia do kolejnych pól wejściowych identyfikatora ORCID, jest sprzężona z modelem weryfikacji i automatycznej korekty danych wejściowych. Implementacja ciągłości czterech pól była również konieczna, aby użytkownik miał możliwość wklejenia numeru ORCID. Brak takiej funkcji, spotkałoby się najprawdopodobniej z irytacją i koniecznością ręcznego przepisania danych ze strony wprowadzającego dane.

Drugi widok umożliwia wyszukanie osoby za pomocą imienia i nazwiska. Nie posiada tak zaawansowanej walidacji, jak poprzedni widok, ponieważ w tym wypadku wystarczyło wymuszenie polskiej lokalizacji w polach wejściowych. Jedyną restrykcją w przypadku tych pól jest ograniczenie długości imienia, do nieco dłuższego, niż najdłuższe trzynastoliterowe polskie imię: Wierzchosława.



Rysunek 8 Domyślny widok panelu wyszukiwania



Rysunek 9 Alternatywny widok panelu wyszukiwania

Ustawienie domyślnej formy wyszukiwania, jako identyfikator ORCID, nie jest przypadkowe i jest związane z tworzeniem zapytań do wspieranych baz.   
W przypadku podania imienia i nazwiska, wątki odpowiedzialne za odpytanie zewnętrznych serwisów muszą poczekać na wątek odpytujący bazę dorobek, który to podane imię i nazwisko tłumaczy na numer ORCID. Po jego wyłuskaniu kolejne wątki rozpoczynają pracę. W przypadku podania identyfikatora, taki problem nie występuję.

Ostatnim elementem tej części interfejsu jest przycisk do rozpoczęcia procesu szukania. Zawiera on pojedynczy symbol lewostronnej lupy, oznaczony numerem kodowym U+1F50D. Zastosowanie ikony, jako opisu przycisku ma dwie motywacje. Pierwszą z nich jest łatwość w utrzymaniu proporcji przycisku, w zamierzonej formie kwadratu. Kolejnym aspektem jest ułatwienie przyszłego, potencjalnego procesu lokalizacji, poprzez zmniejszenie pól posiadających napisy. Dodatkowo przycisk, posiada skrót klawiszowy – SHIFT + ENTER.

#### Lista współpracowników



Rysunek 10 Panel współpracowników z przykładowymi danymi (skala szarości)

Imiona i nazwiska wraz identyfikatorami ORCID, są wyświetlane w trakcie procesu wyszukiwania, jako efekt uboczny procesu przetwarzania danych z bazy DOROBEK. Pierwotne założenia zakresu prac nie zakładały istnienia tej części interfejsu, jednakże pomysł okazał się na tyle przyjazny dla użytkownika, że znalazł się w ostatecznej wersji aplikacji.

Zakończenie wyszukiwania objawia się w dwojaki sposób. Pierwszym z nich jest pasek postępu, który wskazuję postęp procesu poboru i przetwarzania danych. Drugim indykatorem zakończenia przetwarzania jest zmiana barw w interfejsie spowodowana ponownym aktywowaniem kluczowych elementów interfejsu po zakończeniu obróbki danych. Pasek postępu jest używany przez dwie kluczowe funkcjonalności programu, czyli przez część związaną z przetwarzaniem, oraz raportowaniem. Ciężko dostrzec aktualizacje paska postępu w przypadku używania go przez część raportową i jest to spowodowane bardzo szybkim procesem generowania raportu, nawet na sporej ilości danych.

Pozycje na liście oznaczonej, jako „Współpracownicy” są posortowane według ilości wspólnych publikacji z wyszukiwanym podmiotem, znajdującym się zawsze na górze listy. Wybieranie dowolnej pozycji spowoduje wyświetlenie, na prawym panelu (opisanym w kolejnym podrozdziale) listy wspólnych publikacji. Tego typu rozwiązanie umożliwia dodatkową weryfikację poprawności danych ze strony użytkownika.

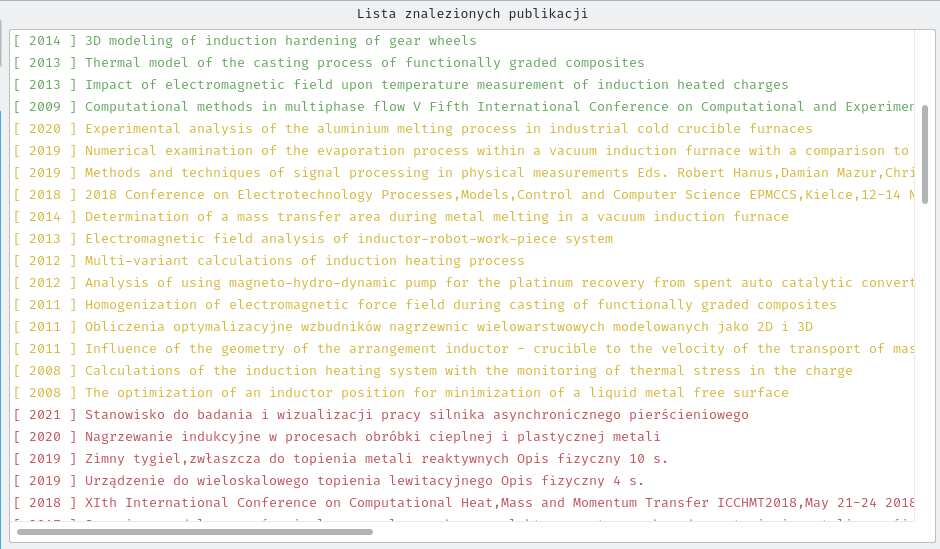
Podwójne kliknięcie w pozycję na tej liście sprawi, że identyfikator wybranej osoby, zostanie przepisany do panelu szukania. Ważnym jest, żeby zaznaczyć, że proces wyszukiwania nie nastąpi, do momentu ponownego wyboru przycisku szukaj. Natychmiastowe rozpoczęcie wyszukiwania, w przypadku przypadkowego podwójnego kliknięcia, mogłoby wywołać konsternację u obsługującego, co nie jest pożądanym odczuciem przy korzystaniu z aplikacji.

#### Lista publikacji

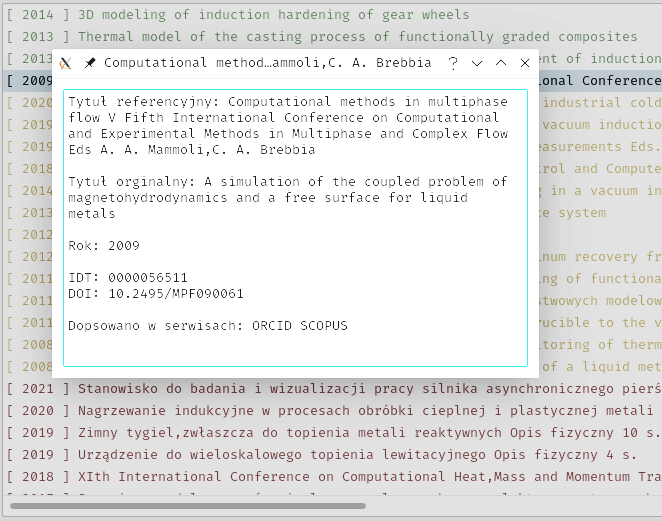
Największym panelem aplikacji, jest pole zawierające wynik działania aplikacji. Są to kolejne publikacje posortowane za pomocą trzech zaznaczonych kluczy. Najważniejszy czynnik wpływający na kolejność to ilość serwisów, w jakich udało się dopasować daną pozycję. Informację tą uwidacznia kolor pozycji. Zielony wskazuje, że publikacja została odnaleziona we wszystkich wspieranych bazach. Pomarańczowy informuje o częściowym dopasowaniu, natomiast czerwony ma za zadanie wyróżnić pozycje bez żadnego dopasowania. Kolejne dwa czynniki wpływające na kolejność to rok zapisany w kwadratowych nawiasach oraz tytuł publikacji.

Przedstawione informacje nie są jednak kompletne, ponieważ wciąż brakuje informacji, w których serwisach prace nie zostały odnalezione. Jeżeli użytkownik potrzebuje takiej informacji, ma dwie możliwości. Pierwszą z nich jest podwójne kliknięcie, które wyświetli okienko ze szczegółami, lub może wygenerować raport.

#### Generowanie raportu



Rysunek 11 Panel publikacji z przykładowymi danymi



Rysunek 12 Okienko dialogowe, ze szczegółami przeszukiwania (rozjaśnione)

Ostatni element interfejsu użytkownika, znajduje się pod listą publikacji, oraz składa się wyłącznie z przycisku o długości równej szerokości elementu powyżej. Jedynym jego celem jest wyzwolenie akcji generowania raportu. Podobnie jak przycisk wyszukiwania, również posiada przypisany skrót klawiszowy:   
CTRL + ENTER

## Pobór danych

### Baza DOROBEK

Problematyka poboru danych z bazy DOROBEK, została po części wyjaśniona w rozdziale poświęconym przeglądowi literaturowym w podrozdziale omawiającym aktualny model synchronizacji (3.1.2). W tym rozdziale nacisk zostanie położony na stworzone rozwiązania, umożliwiające wydobycie interesujących danych.

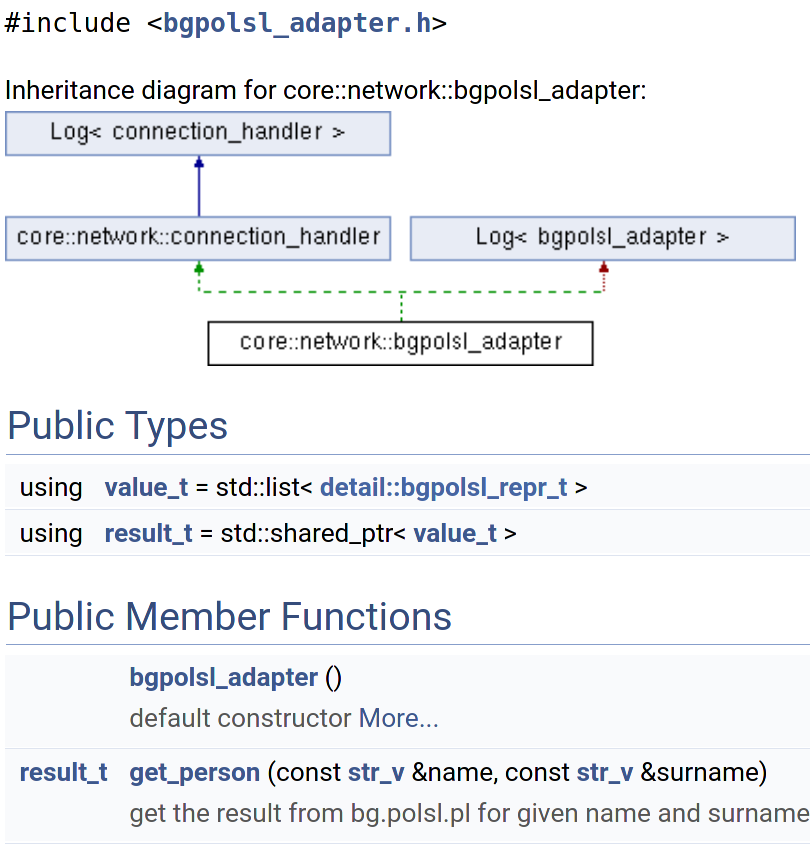
#### Przygotowanie zapytania

Pierwszym krokiem umożliwiającym stworzenie zapytania jest zapoznanie się z dostępną dokumentacją, bądź jak w tym wypadku analiza ruchu, w kontekście wprowadzonych w formularzu danych (Rysunek 1). Konstrukcja aplikacji daje dwa możliwe kryteria wyszukiwania. Pierwszym jest identyfikator ORCID, drugim zaś, para ciągów znakowych imię i nazwisko. Przeglądając dostępne pola, umożliwiające filtrowanie na stronie bazy DOROBEK, okazuje się, że dostępne są obie możliwości, jednakże wyniki zwracane przez stronę w przypadku filtrowania po identyfikatorze ORCID są skromniejsze niż przy wyszukiwaniu tej samej osoby, za pomocą imienia   
i nazwiska. Potwierdza to próba wyszukiwania za pomocą klucza: ADRIAN SMAGÓR, zwracająca 33 wyniki, oraz próba wyszukania za pomocą odpowiadającego identyfikatora ORCID: 0000-0002-1994-3266. Druga próba zamiast spodziewanych 33 wyników, zwraca zaledwie 27. Kierując się większą ilością danych, para imię i nazwisko zostaje wybrana, jako klucz w przeszukiwaniu bazy DOROBEK.

Analiza ruchu sieciowego strony, wskazuje, że polem w żądaniu, pobierającym rekordy, odpowiedzialnym za przechowywanie imienia i nazwiska jest to, nazwane V\_00. Dodatkowo zostaje uwidoczniona różnica, pomiędzy wprowadzonym ciągiem znaków: ADRIAN SMAGÓR, a odczytanym z żądania: SMAG%C3%93R+ADRIAN. Wynika to z faktu zastępowania znaków, nieznajdujących się w zakresie tablicy ASCII (31), odpowiednimi znacznikami, aby zawartość ciała żądania, została poprawnie zinterpretowana, przez wszystkie pośredniczące programy, na serwerze kończąc. W tym przypadku jest to zamiana litery „Ó” na „%C3%93”, co jest liczbą kodową litery, w systemie UTF-8 zapisaną systemem szesnastkowym (31). Dodatkowo podmiana znaków specjalnych zabezpiecza system, przed wstrzykiwaniem fragmentów kodu języka SQL (ang. SQL Injection) (32), oraz wstrzykiwaniem skryptów klienckich (ang. Cross Site Scripting) (33), które mogą narazić serwer, lub klientów na wykonanie niepożądanego kodu.

Przed przejściem do kolejnego etapu, jest sprawdzenie, jak serwer będzie reagował, gdy zapytanie nie nadchodzi za pośrednictwem aplikacji klienckiej, jaką jest aplikacja webowa bazy DOROBEK. Jednym z możliwych scenariuszy, jest odmowa dostępu, na przykład, przez brak ważnego uwierzytelnienia za pomocą protokołu OAuth. Wysłanie żądania za pomocą narzędzia cURL, odrzuca tą wątpliwość, poprzez wyświetlenie odpowiedzi z serwera, zawierającą poprawną odpowiedź.

Generowanie zapytania po stronie aplikacji wymaga jednak stworzenia mechanizmu, który będzie odpowiednio preparował wprowadzone imiona i nazwiska, aby spełniały normy bezpieczeństwa. Zadanie zamiany łańcuchów na różne formaty, należy do klasy demangler. Zawiera ona szereg metod, umożliwiających unifikację, konwersję i rewersję wejściowych łańcuchów znaków. Klasa ta jest wspierana, przez klasę depolonizator, która zawiera, jako pole statyczne, definicję mapy translacji, pomiędzy różnymi zapisami.



Rysunek 13 Interfejs klasy komunikującej się z bazą dorobek



Rysunek 14 Interfejs klasy *demangler*

#### Przetwarzanie danych

Przetwarzanie źródła strony internetowej, można wykonać na jeden z dwóch sposobów. Pierwszy, bardziej elegancki, to zastosowanie odpowiedniego parsera, umożliwiającego semantyczne przetworzenie tekstu, co pozwala w do pewnego stopnia uniezależnić się od zmian wprowadzanych na stronie. Zaletą, jest również możliwość tworzenia złożonej logiki oraz delegację walidacji danych do zewnętrznego kodu parsującego.

Alternatywą jest wyszukiwanie, przez dopasowywanie wzorów, na przykład poprzez wykorzystanie wyrażeń regularnych, lub szukając charakterystycznych fragmentów w tekście. Zdecydowaną zaletą tego rozwiązania, jest brak wymogu, ze strony źródła, aby było poprawnie zapisane, co mogłoby doprowadzić do błędu   
w analizatorze formatu. Dodatkowo, brak konieczności analizy oraz tworzenia rozległych struktur drzewiastych w pamięci, czyni taki proces szybszym. Wadą tego podejścia jest duża wrażliwość mechanizmu przetwarzającego na zmiany w źródle strony.

Podczas procesu tworzenia kodu, przetestowanych zostało kilka bibliotek napisanych w języku C++, jednakże żadna nie spełniła jednocześnie dwóch wymagań: wysokiej wydajności, zrozumiałego interfejsu. Dzięki pierwszej, wykonanie kodu, będzie mieścić się w rozsądnych ramach czasowych, szczególnie przy większej liczbie publikacji. Druga zapewnia łatwość w utrzymaniu kodu oraz nie utrudnia procesu diagnozowania błędów. Leksykalne przetwarzanie kodu HTML, XML, lub innego opartego na znacznikach, nie jest wydajne na tle innych formatów. Dobrze oddaje to cytat wybitnego programisty, twórcy jądra systemu GNU/Linux, Linusa Torvaldsa (35), jednakże z uwagi na jego wulgarność, nie zostanie tutaj przytoczona.

Konsekwencją nieudanych prób, było stworzenie dedykowanego interpretera kodu strony, bazy DORORBEK, opierającego się na słowach kluczowych. Składa się on z trzech warstw. Każda z nich działa w oparciu o wynik poprzedniej. Pierwsza jest najbardziej prymitywna, jednakże redukuje znacząco ilość kodu do przetworzenia. Korzystając z wyrażenia regularnego wyszukuje linijki, zawierające potrzebne dane.

Jeżeli aktualnie przetwarzana linijka, pasuje do używanego wyrażenia regularnego, jest ona przekazywana do funkcjihtml\_scalpel. Jej zadaniem jest odfiltrowanie znaczników, wraz z ich zawartością, pozostawiając wyłącznie tekst pomiędzy nimi. Na tym etapie największa część przetwarzanego kodu jest odfiltrowywana. Znalezione słowa są kumulowane w wektorze, który celem uniknięcia kopiowania, jest przekazywany do funkcji wyłuskującej, jako referencja.

Na samym początku przychodząca linijka jest konwertowana do typu u16str, który zapewnia poprawne kodowanie polskich znaków diakrytycznych. Typ ten jest definiowany, jako zmienna łańcuchowa z biblioteki standardowej, która zamiast podstawowych jednobajtowych zmiennych znakowych używa dwubajtowych. Jest to najbardziej kosztowna operacja w ciele funkcji, jednakże gwarantuje poprawne przetworzenie odczytywanych danych. Następnie w pętli, następuje iterowanie po zmiennej tekstowej. Za pomocą serii przełączników, następuje decyzja, czy dany znak powinien zostać uwzględniony. Wykrywając określone znaki, między innymi spacje i przecinki, ciąg znaków jest dzielony na słowa, które są przesuwane do wektora, korzystając z funkcji bibliotecznej std::move.

Przygotowany w ten sposób zestaw słów, jest przekazywany do trzeciej warstwy, reprezentowanej przez konstruktor struktury bgpolsl\_repr\_t (Rysunek 15). Zawiera ona zbiór instrukcji w postaci tablicy asocjacyjnej. Jej kluczem jest oczekiwane słowo, natomiast wartość jest reprezentowana przez wskaźnik na zmienną, do której kolejne słowa mają trafić. Taka organizacja ma dwie zalety, oraz jedną poważną wadę.



Rysunek 15 Konstruktor struktury *bgpolsl\_repr\_t*

Pierwsza zaleta dotyczy sporej elastyczności kodu. Jeżeli zajdzie potrzeba wyłuskania dodatkowych danych, zmiana będzie polegała na modyfikacji istniejących wartości w mapie lub dodaniu nowych. Definiowanie ignorowanych pól poprzez przypisanie ich punktowi zapisu wartość pustą, jest czytelne i zrozumiałe.

Kolejną zaletą jest umiejscowienie obciążających tekstowych porównań  
w strukturze, zoptymalizowanej pod kątem przeszukiwań. Dzięki złożoności logarytmicznej, przeszukiwany zbiór z każdym porównaniem się zmniejsza.  
W przeciwieństwie do wektora, nie ma konieczności przeszukiwania przy każdym słowie całej struktury.

Wadą tego rozwiązania, jest zwiększające się prawdopodobieństwo wystąpienia błędnego przetwarzania z każdym kolejnym kluczem. Jeżeli w wartości dowolnego pola, na przykład w tytule, znajdzie się wyraz obecny już, jako klucz, może dojść do błędnego wyłuskania tytułu, w tym wypadku ucięcia jego części, oraz potencjalnego wpisania jego reszty, do błędnie rozpoznanego pola. Jest to poważne zagrożenie, jednakże stosunkowo proste w diagnozie z uwagi na charakterystyczne przesunięcie wszystkich danych.

Tak przygotowany obiekt, dodawany jest do kolekcji, który jako zbiór wstępnie zorganizowanych danych, jest przekazywany do warstw o wyższej abstrakcji, celem umożliwienia porównania danych z różnych źródeł. Wyższe warstwy zostaną omówione w podrozdziale 3.6 *Unifikacja*.

Zaprojektowany przepływ danych umożliwia zrównoleglenie procesu przetwarzania, jednakże koszt stworzenia nowego wątku, może przewyższać potencjalne zyski czasowe. Potencjalnym rozwiązaniem tego problemu, mogłoby być przetwarzanie wielowątkowe, gdzie każdy z wątków otrzymuje zakres linijek do przetworzenia. Pytanie, czy synchronizowanie docelowej kolekcji, podczas umieszczania w niej wyników prac wątków, nie pozbawiłoby takiej modernizacji sensu, bez testów, pytanie pozostaje bez odpowiedzi.

### Bazy ORCID oraz SCOPUS

Tytułowe bazy posiadają mocno zbliżony schemat danych przychodzących, co pozwala omówić je jednocześnie nie pomijając kluczowych kwestii. Oba źródła danych są na tyle zbliżone, że korzystają z wielu tych samych klas oraz funkcji.

#### Obsługa zapytań

Portal ORCID, zapewnia wysoki poziom dokumentacji oraz przykładów użycia. Skutkiem tego jest bardzo krótki kod zajmujący się generowaniem żądania, zajmujący zaledwie trzy linijki. Serwis podczas zapytania zwraca komplet danych, powodując wyczerpanie tematu obsługi sieci w ramach bazy ORCID.

Klasa odpowiedzialna, za obsługę połączenia z serwisem ORCID, posiada względem pozostałych adapterów jedną więcej metodę do poboru informacji   
o zadanej za pomocą argumentów osobie. Metoda get\_name\_and\_surname i jest używana w momencie, gdy użytkownik chce skorzystać z wyszukiwania za pomocą identyfikatora ORCID. Mimo, że wyszukanie za pomocą numeru wymaga dodatkowego zapytania, jest szybsze z powodu możliwości uruchomienia wszystkich wątków jednocześnie, ponieważ za wyjątkiem adaptera obsługującego bazę DOROBEK, wszystkie adaptery korzystają z identyfikatora ORCID.

Przykład serwisu SCOPUS, jest bardziej złożony. Generowanie żądania utrudniają limity narzucone przez portal, który zwraca maksymalnie dwadzieścia pięć rekordów, co w przypadku bardziej płodnych twórczo pracowników Politechniki Śląskiej wymaga wielokrotnego odpytywania się bazy. Mechanizm stronicowania   
(ang. pagination) jest zrobiony na zasadzie podpowiedzi doklejanych do danych zwrotnych. Oznacza to, brak konieczności wyliczania zakresu kolejnej części danych, ponieważ wyliczone wartości, są obecne w ciele odpowiedzi z serwisu.

Ograniczenia czasowe nie pozwoliły na wdrożenie dedykowanego rozwiązania, które znajduje się w języku C++ od najnowszego, obowiązującego   
w tym projekcie, standardu – korutyn. Pozwoliłoby to na łatwiejsze uwspółbieżnienie procesu oczekiwania na dane oraz przetwarzanie już pobranych.

#### Przetwarzanie danych

Pod względem przetwarzania, oba źródła danych są identyczne i można opisać je za pomocą algorytmu:

1. Korzystając z funkcji bibliotecznej, przetwórz przychodzące dane w formacie JSON;
2. Korzystając z opublikowanego na stronie dokumentacji serwisu, schematu pliku JSON, stwórz obiekt tymczasowy i wyłuskuj kolejne pola;
   1. Jeżeli pole jest zagnieżdżone, przed każdym wyłuskaniem sprawdź, czy pole nie jest puste (nie ma wartości null);
   2. Jeżeli spodziewanym typem elementu jest tablica, sprawdź czy jej rozmiar jest większy lub równy jedności;
   3. W momencie otrzymania typu prostego na spodziewanym poziomie, dokonaj przypisania do odpowiedniego pola   
      w tymczasowym obiekcie
   4. Po uzupełnieniu wszystkich wymaganych pól w obiekcie, dodaj go do kolekcji

Konsekwencją schematu pliku JSON, co za tym idzie również konstrukcji algorytmu, jest tworzenie przez kod, charakterystycznych wielokrotnie zagnieżdżonych instrukcji sterujących (Rysunek 16)



Rysunek 16 Przykładowy fragment wyłuskania wartości z danych w formacie *JSON*

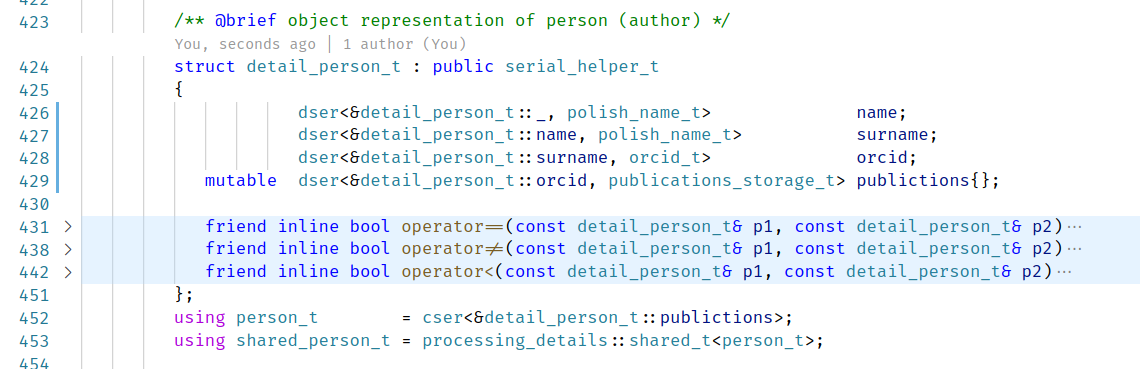
## Unifikacja

Wstępne przetworzenie nadchodzących danych, omówione w poprzednim rozdziale, jest dopiero początkiem w obróbce danych, a zarazem pierwszym poziomem w przepływie (ang. flow) aplikacji. Kolejnym etapem, koniecznym, aby móc w prosty sposób porównywać dane.

Podstawowym rozwiązaniem, zastosowanym również tutaj, jest stworzenie klasy lub struktury, która będzie obiektową reprezentacją autora oraz publikacji. Sposób ten został rozwinięty o możliwość dowolnej serializacji oraz deserializacji takiej struktury. Zabieg ma na celu uprościć proces logowania, a także umożliwić przyszły rozwój takich funkcjonalności jak obiektowe podejście do bazy danych, szerokie wsparcie różnych form raportów, czy możliwość strumieniowania obiektów. Problematyka oraz złożoność implementacji została omówiona w rozdziale 3.9.



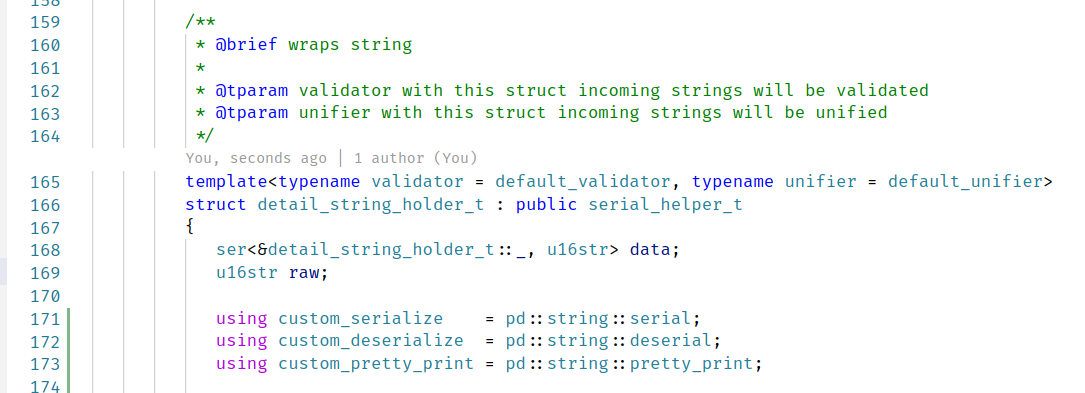
Rysunek 17 Obiektowa reprezentacja publikacji



Rysunek 18 Obiektowa reprezentacja osoby autora

### Reprezentacja obiektowa

Podczas omawiania drugiego poziomu, najważniejsze są dwie klasy, obie zaprezentowane na powyższych zrzutach ekranu. Jednakże faktyczny model reprezentacji obiektowej składa się z wielu mniejszych klas, których zadanie jest wysoce wyspecjalizowane.



Rysunek 19 Fragment klasy - obiektowej postaci zweryfikowanego i jednolitego tekstu

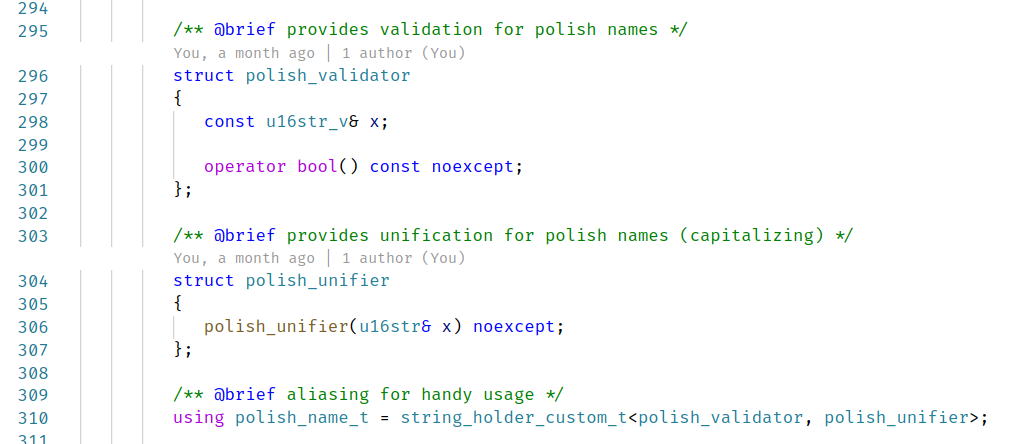
Przykładem takiej klasy jest detail\_string\_holder\_t, która odpowiada za weryfikowanie, oraz unifikowanie przechowywanych łańcuchów znakowych. Wymagane parametry szablonu to dwie klasy.

Pierwsza z nich, o nazwie validator, wymaga od typu dwóch metod. Pierwsza to konstruktor jednoargumentowy, który przyjmuje stałą referencję do łańcucha znaków, który ma zostać poddany weryfikacji. Druga wymagana metoda to operator konwersji na typ logiczny – bool. Użycie tego typu sprowadza się do stworzenia tymczasowego obiektu, na który natychmiast jest wymuszona konwersja do zmiennej logicznej. Otrzymany wynik decyduje o wystąpieniu wyjątku podczas przetwarzania łańcucha znaków, który docelowo ma zostać przechowywany.

Kolejny parametr to unifier, czy unifikator. Wymogi odnośnie tego typu to wyłącznie jednoargumentowy konstruktor, przyjmujący referencje na zweryfikowany łańcuch znaków. Klasa ma za zadanie przetworzyć łańcuch w sposób umożliwiający porównanie szybsze i pewniejsze porównanie.

Dodatkowo struktura przechowuje przetworzoną wartość w oryginale w polu  
o nazwie raw. Pozorna redundancja danych jest w rzeczywistości optymalizacją czasu przetwarzania, dokonana kosztem pamięci operacyjnej. Z uwagi na nierzadkie porównania dwóch ciągów znaków, w strukturach uporządkowanych, lub w trakcie dokonywania dopasowania różnych obiektów, lepiej jest zapisać przetworzoną formę, a w przypadku konieczności odczytu, używać wyliczonej postaci. Konieczność przechowania oryginału wiąże się z wymogiem reprezentacji przechowywanych danych użytkownikowi w interfejsie użytkownika.

Przykładem specjalizacji, tej struktury, jest typ polish\_name\_t (Rysunek 20), który w ramach parametrów szablonu, przekazuje własne narzędzia walidacji  
i ujednolicenia łańcuchów znakowych



Rysunek 20 Deklaracja typu *polish\_name\_t*

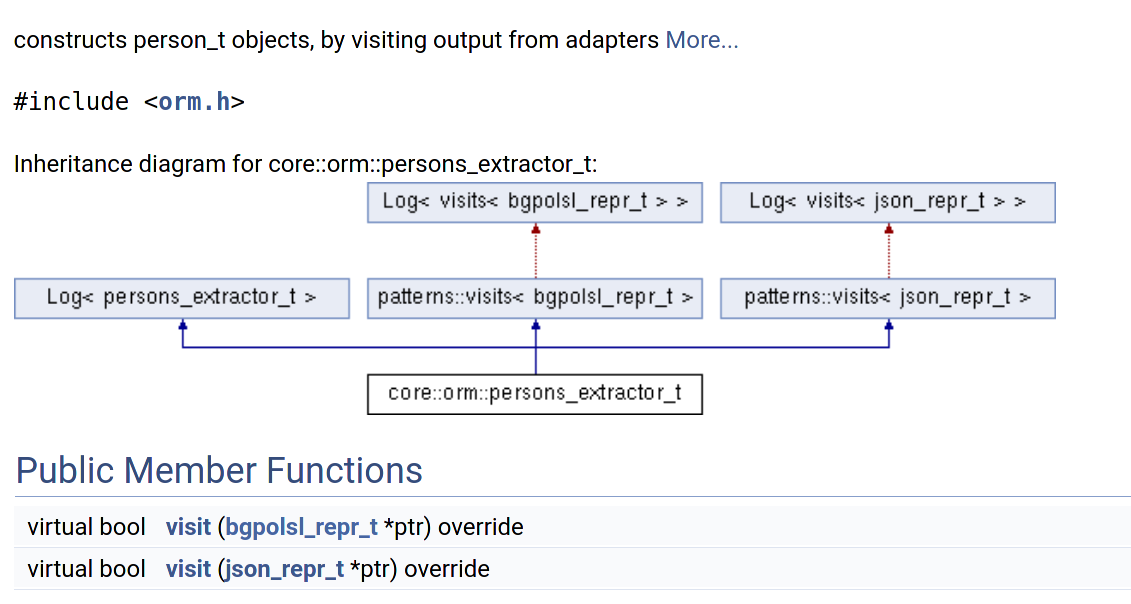
Rozwiązanie oparte o szablony, pozwala na delegację ról do pojedynczych klas lub struktur, oraz znacząco redukuje ilość pisanego kodu. Jeżeli będzie potrzeba zmiany sposobu unifikacji polskich imion, lub dostosowanie poziomu weryfikacji, wystarczy tego dokonać w jednym miejscu. W tym przypadku rolę kontenera pełni string\_holder\_t, rolę weryfikatora polish\_validator, a unifikatora polish\_unifier.

Klasa od przechowywania łańcuchów znaków jest w projekcie wykorzystana również do znakowej reprezentacji typów enumerowanych, przechowywania tytułów, czy reprezentacji identyfikatorów publikacji, takich jak DOI.

### Generowanie obiektów wyższej abstrakcji

Przejściem między poziomem pierwszym, gdzie znajdują się świeżo przetworzone dane z serwisów, do poziomu umożliwiającego ich porównanie, zostało wykonane przy pomocy autorskiej modyfikacji wzorca projektowego wizytator (37) do wzorca nazwanego roboczo: zaproszony wizytator.

Różnica polega na częściowo statycznym rozwiązaniu zależności, a także jawnej deklaracji wspieranych klas, przez wizytatora. Zapewnia to przejrzystość   
w czytaniu kodu, daje możliwość kompilatorowi na zwiększenie poziomu optymalizacji, a także zaznaczenie dodatkowej zależności na diagramie klas w dokumentacji automatycznej (Rysunek 21).



Rysunek 21 Interfejs klasy konwertującej dane na obiektową postać osoby (autora)

Wykorzystanie wzorca projektowego wizytator jest spowodowane, chęcią wydzielenia funkcjonalności poza klasę, w tym wypadku person\_t. Powoduje to, delegację funkcjonalności i odciążenie klasy z części zadań. Dodatkowo umożliwia stworzenie logicznego podziału i przejrzystości w zależnościach pomiędzy bibliotekami. Klasa odpowiedzialna za obiektową reprezentację osoby, nie ma potrzeby dostępu do szczegółów pobierania danych. Rozwój aplikacji, poprzez dodawanie kolejnych interfejsów sieciowych powodowałby znaczący rozrost klasy, której zadanie nie polega na konwersji pomiędzy warstwami, co nie jest pożądane.

#### Przetwarzanie danych z bazy DOROBEK

Rozpoczęcie procesu przetwarzania odbywa się poprzez odwiedzenie, przez klasę publications\_extractor\_t, struktury wyjściowej z adaptera bazy DOROBEK – bgpolsl\_repr\_t. Przed uzupełnieniem dowolnego pola, zapewniana jest podstawowa weryfikacja, sprawdzająca obecność jakichkolwiek danych, oraz zakończenie przetwarzania, w przypadku braku kluczowych.

Pierwszym elementem uzupełnianym przez wizytator, są identyfikatory.  
W przypadku bazy DOROBEK, obsługiwane są następujące wyróżniki: IDT, DOI,   
E-ISSN oraz P-ISSN. Każdy z nich jest unifikowany podczas dodawania do obiektu reprezentującej publikację. Jeżeli po wykryciu wszystkich możliwych zbiór jest pusty, następuje wyjście z funkcji, ponieważ jest to podstawowy (ale nie jedyny!) czynnik determinujący dopasowanie publikacji względem innych źródeł danych.

Kolejnym przetwarzanym polem jest data, która po sprawdzeniu, czy pole nie jest puste, wywołuje funkcje biblioteczną std::stoi. Brak wsparcia dla dwubajtowych ciągów znakowych, ze strony biblioteki standardowej było zaskoczeniem, oraz wymusiło dodatkową konwersje do standardowego jednobajtowego kontenera tekstu. Jest to dobre miejsce do rozpoczęcia procesu optymalizacji, zastępując koniunkcję tekstowej konwersji oraz wywołania funkcji bibliotecznej, na dedykowany konwerter dwubajtowych łańcuchów znakowych do liczb całkowitych.

Pola przetwarzane, jako trzecie w kolejności, to tytuły. Ponieważ obiektowa reprezentacja przewiduje dwa rodzaje tytułów, jest konieczność dopasowania, który tytuł zostanie wpisany do odpowiedniego pola. Pierwszym polem obiekcie bardziej abstrakcyjnym, jest tytuł oryginalny, który ma priorytet podczas komparacji dwóch publikacji, oraz polski, gdy tytuł występuje w dwóch językach.

Optymistycznym przypadkiem rozróżnienia, jest zbiór publikacji, posiadających dwa osobne wpisy, na tytuł oryginalny i w języku polskim. Taki przypadek pozostawia tylko problem detekcji języka, ponieważ nie ma gwarancji, zastosowanego języka dla danego pola w obiekcie wyjściowym adapteru bazy DOROBEK. Decyzyjność w tej sprawie jest niedopracowana i bazuje na detekcji polskich znaków w tekście, co jest mocnym nadużyciem, ale z uwagi na brak lepszego rozwiązania, częściowo spełnia swoje zadanie. Założenie takich kryteriów sprawia, że zdania nie zawierające polskich znaków, nie zostaną poprawnie rozpoznane, na przykład: Ala ma kota.

Istnieje jednak niemały zbiór prac, gdzie dwa tytuły są wpisane do tego samego pola w obiekcie wynikowym z wstępnego przetwarzania. Niestety z uwagi na tę, niechlubną praktykę konkatenacji tytułów w języku polskim i nowożytnym przy użyciu kropki, przecinka, średnika lub w najgorszym przypadku spacji, jako separatora, nie jest możliwe poprawne rozdzielenie dwóch tytułów, jeżeli znajdą się w tym samym polu. Lata zaniedbań w weryfikacji danych oraz trzymaniu spójności wpisów, mają swoją konsekwencję w braku możliwości reprezentacji tych publikacji  
w sposób prawidłowy za pomocą podejścia obiektowego. Warto zaznaczyć, że mimo niepoprawnych danych, wciąż jest możliwość wykazania zbieżności między odpowiadającymi sobie publikacjami z różnych źródeł. Presja czasowa, niestety uniemożliwiła implementacje opisanego później, w niniejszej pracy, rozwiązania.

Przed zakończeniem przetwarzania publikacji następuje użycie jeszcze jednego, wcześniej wspomnianego (Rysunek 21) wizytatora. Użycie polega na wpisaniu przetworzonej publikacji, do pola wizytatora, oraz użyciu go na przychodzących danych.



Rysunek 22 Metoda wyłuskująca kolejnych autorów z zadanej publikacji

Klasa wyłuskująca autorów, korzysta z autorskiego iteratora (38) zakresów tekstowych. Motywacją do implementacji takiej konstrukcji, było rozczarowanie nowym modułem biblioteki standardowej dla operacji na zakresach – ranges. Potrzeba podzielenia tekstu na słowa, innymi słowy, wskazania zakresów, pomiędzy spacjami, jest zaspokajana przez bibliotekę zakresów, jednakże oferowany dostęp do wytyczonych danych uniemożliwia wydajnościowe przetwarzanie. Dzieje się tak, ponieważ wbudowane iteratory pozwalają wyłącznie na dostęp do kolejnych liter, zamiast generować widoki, za pomocą bardzo szybkich widoków tekstowych  
(ang. string view). Powoduje to konieczność przepisania zakresu do tymczasowej zmiennej tekstowej, a następnie wykorzystanie zakresu, co jest marnotrawstwem zasobów. Napisana klasa pozwala dowolnie dzielić tekst, zarówno na słowa, jak również na linie, czy dowolny inny znak, co zostało zaprezentowane w linii 17 i 20 na powyższym zrzucie ekranu (Rysunek 22). Klasa pozwala na dostęp do kolejnych zakresów w pętli zakresowej, co zdecydowanie poprawia czytelność kodu.

Dodanie autora do unikalnego zbioru, następuje, jeżeli zostaną poprawnie rozpoznane trzy łańcuchy znaków: imię, nazwisko oraz identyfikator ORCID.  
W każdym innym wypadku następuje przejście do przetwarzania kolejnego autora. Przy poprawnym rozpoznaniu autora, zostaje dodana jeszcze referencja do aktualnie przetwarzanej publikacji, jako wspólnego wskaźnika, co omija duplikację danych oraz niweluje problem własności obiektu. Stworzenie tej zależności jest pomocą dla graficznego interfejsu użytkownika.

Zakończenie przetwarzanie w ekstraktorze publikacji kończy dodanie obecnej do zbioru już wcześniej poprawnie przetworzonych prac.

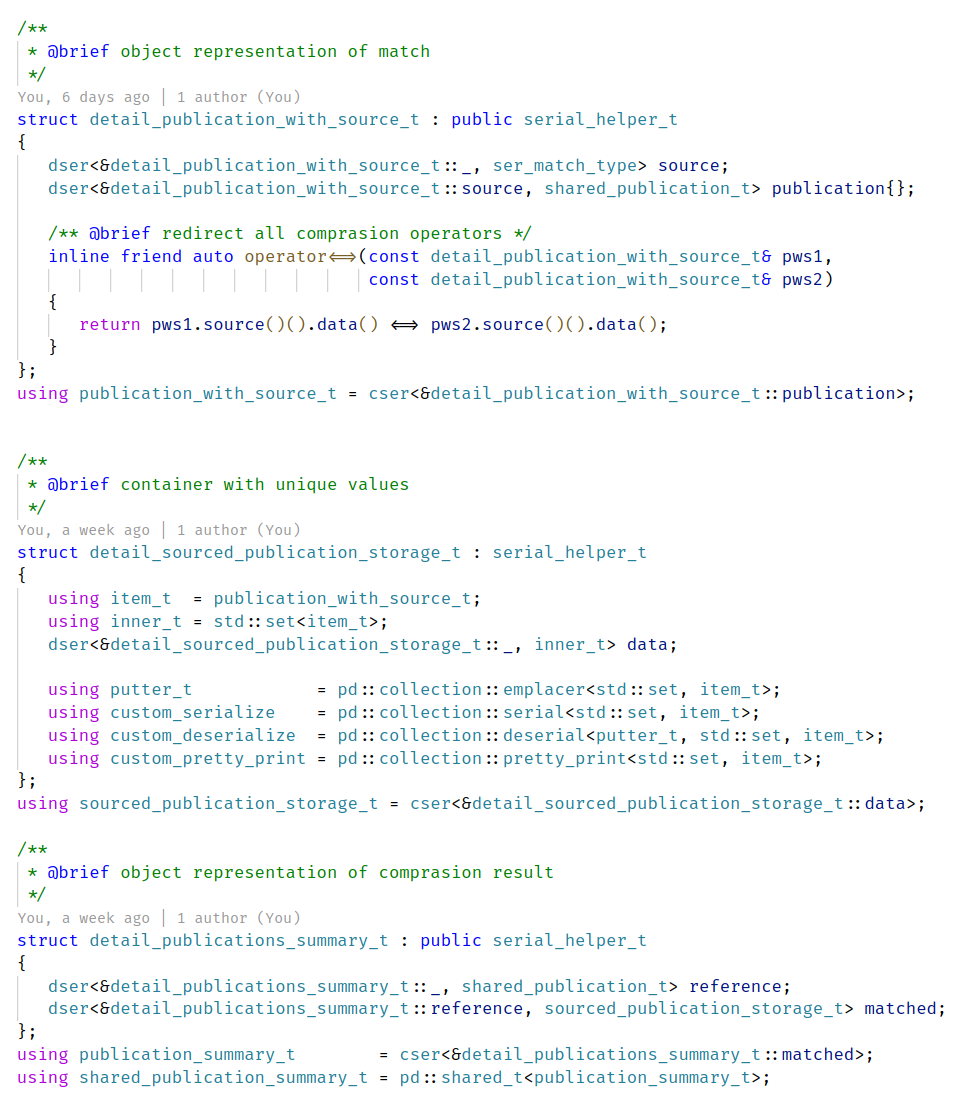
#### Przetwarzanie danych z pozostałych źródeł

Obiektem wynikowym z przetwarzania danych pochodzących z baz ORCID oraz SCOPUS, są obiekty typu json\_repr\_t, dzięki czemu możliwe jest przetworzenie wyniku z obu źródeł za pomocą tej samej funkcji. Obniża to ilość pisanego,   
jak i generowanego kodu oraz ułatwia utrzymanie kodu w przyszłości.

Przetwarzanie rozpoczyna się od sprawdzenia roku, jeżeli jest niepusty następuje wcześniej wspomniana nieoptymalna konwersja do standardowego łańcucha znaków, a następnie konwersja na liczbę. W kolejnych krokach zostają przepisanie tytuły oraz przepisane zostają numery identyfikacyjne. Jeżeli identyfikatory nie występują, lub tytuł w przetwarzanym obiekcie jest pusty, metoda kończy przebieg. Aktywowanie ekstraktora autorów na tym obiekcie, jest podyktowane, wyłącznie weryfikacją danych, co zostanie wyjaśnione lepiej   
w kolejnym rozdziale. Ostatecznie publikacja zostaje dodana do zbioru znalezionych publikacji.

## Dopasowywanie

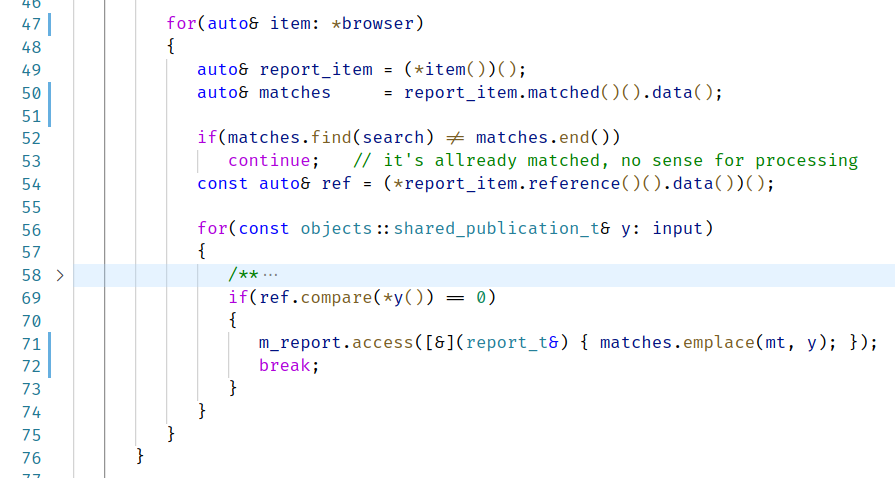
Kolejnym poziomem przetworzenia danych, jest stworzenie głębszych zależności pomiędzy przetworzonymi danymi. Zapis relacji, występujący pomiędzy publikacjami jest zdefiniowany przez trzy struktury: detail\_publication\_with\_source\_t, detail\_sourced\_publication\_storage\_t, oraz detail\_publications\_summary\_t.



Rysunek 23 Deklaracje struktur opisujące zależności pomiędzy publikacjami

Klasa, która zawiera definicję dopasowania publikacji, oraz generuje obiekty struktur przedstawionych na powyższym rysunku (Rysunek 23) nazywa się summary. Metodą odpowiedzialną za przetwarzanie, która posiada wsparcie dla wielowątkowości, jest process\_impl.

Metoda jest zaprojektowana do wywołania w osobnym wątku, oraz wymaga ustawienia zmiennych wewnętrznych, czego nie gwarantuje utworzenie klasy. Opóźnienie pełnej inicjalizacji, jest konieczne, aby umożliwić przekazanie do wielu wątków instancji tej samej klasy. Implikuje to również konieczność potwierdzenia, ustawienia wszystkich pól w klasie, za co odpowiedzialna jest zmienna logiczna   
o charakterze atomowym. Wykorzystane zostaje tutaj podejście oparte o rygiel pętlowy (ang. spin lock) (39), w kombinacji z oddaniem czasu procesora, co jest łatwiejsze w implementacji, oraz szybsze, niż tradycyjne podejście wykorzystujące muteks (40).



Rysunek 24 Pętla tworząca zależności, pomiędzy publikacjami

Linia 40 zawiera rozpoczęcie pętli po zsynchronizowanej kolekcji, zawierającej instancje struktury detail\_publications\_summary\_t. Zaraz po sformułowaniu aliasów w dwóch pierwszych wersach, następuje sprawdzenie, czy dopasowanie już wcześniej nie zostało dokonane. Jest to zabezpieczenie, na wypadek zmiany sposobu organizacji wątków lub sposobu przetwarzanie, aktualnie można uznać ten fragment kodu za niepotrzebny.

Po zweryfikowaniu unikalności, następuje iteracja po publikacjach do przypisania, w przypadku wyznaczenia zgodności następuje utworzenie relacji, poprzez dodanie współdzielonego wskaźnika na publikację do kolekcji oraz przerwanie iteracji, z kwestii optymalizacyjnych. Linia 71 zawiera blokadę na iterowanej kolekcji browser, celem modyfikacji obecnego elementu.

Klasa generująca raport używa również wzorca projektowego obserwator za pomocą, którego informuje inne zainteresowane instancje o zakończeniu przetwarzania. Emisja sygnału znajduje się w destruktorze klasy wraz, z którym zostaje rozpropagowany współdzielony wskaźnik do gotowego obiektu reprezentującego zależności pomiędzy publikacjami.

## Nadzorowanie przebiegu

Konstrukcja poprzednich struktur wymusza ścisłą kontrolę życia obiektów, szczególnie podczas korzystania z klasy summary, która informację, o zakończeniu zadanie wysyła dopiero w destruktorze. Ponadto, sposób przetwarzania danych umożliwia zrównoleglenie części procesu.



Rysunek 25 Kolejność startu nowych wątków.

### Zrównoleglenie przetwarzania danych

Podczas pobierania i dopasowywania, korzysta się w sumie z czterech wątków. Pierwszy, jest tworzony z uwagi na rozdzielenie obsługi graficznego interfejsu użytkownika, od czasochłonnego procesu ewaluacji danych. Kolejne trzy są tworzone w pierwszym. Każdy z nich posiada na własność jeden adapter oraz komplet wizytatorów, do ekstrakcji obiektów z przychodzących danych. Ma to dwie zalety. Pierwszą z nich jest klarowny podział własności, oraz użytku obiektów. Dodatkowo dzięki ujednoliceniu interfejsów adapterów, możliwe jest stworzenie ogólnej formy przetwarzania danych dla dwóch z trzech wątków, które zostały zdefiniowane, jako funktory (Rysunek 26).



Rysunek 26 Deklaracja uniwersalnego funktora

Zgeneralizowanie problemu, do szablonu struktury wykonawczej, umożliwia czytelne tworzenie nowych wątków, co zostało zaprezentowane w liniach 270 i 279 na wcześniejszym zrzucie ekranu (Rysunek 25). Przy zachowaniu spójności   
w interfejsach kolejnych adapterów, podczas dodawania obsługi dla kolejnych serwisów, da to możliwość ponownego wykorzystania tego szablonu. Trzymając się zaleceń standardu, zostało dodane wsparcie tokena zatrzymania (ang. stop token), co zapewnia dodatkową responsywność. Bierze się z ona z braku konieczności oczekiwania na zamknięcie programu w przypadku, gdy życzy sobie tego użytkownik. Daje to dodatkowo możliwość w przyszłości zaimplementowania anulowania aktualnego przetwarzania, co jest szczególnie przydatne, gdy użytkownik się pomylił i nie chce czekać na zakończenie zadania.

Zrównoleglenie przetwarzania wiąże się z powstaniem problemu zebrania wyników, z wszystkich wątków, oraz przekazania ich do wyświetlenia na ekranie  
w graficznym interfejsie użytkownika. Rozwiązaniem tej kwestii okazał się wcześniej wspomniany wzorzec projektowy obserwator, który w ramach platformy programistycznej Qt, nazywany jest mechanizmem sygnałów i slotów. Zastosowanie go umożliwia informowanie wielu różnych instancji o zaistniałych wydarzeniach, bez konieczności tworzenia nielogicznych dependencji. Klasa generująca sygnał, nie musi wiedzieć o istnieniu klasy, która zarządza wyświetlaniem tej informacji na ekranie. Użycie takiego schematu dodatkowo poprawia czytanie kodu.



Rysunek 27 Konstruktor klasy głównego okna

Wspomnianą czytelność widać na powyższym zrzucie ekranu (Rysunek 27). Zaznaczone nazwy pól w liniach 31-35, są to obiekty klasy observable, która jest interfejsem na implementację sygnałów z biblioteki boost. Dokładnie widać elementy obsłużone przez główne okno aplikacji, wśród tych oferowanych przez klasę zawierającą opisany wyżej model wielowątkowego przetwarzania – engine.

Wydzielenie osobnej klasy do zarządzania wątkami oraz przebiegiem synchronizacji, ma dać możliwość w przyszłości implementacji innych interfejsów użytkownika. Przykładowe implementacje, to interfejs sieciowy, lub interfejs konsolowy.

### Generowanie raportu

Wybranym formatem reprezentującym raport jest xlsx. Generowanie raportu jest obsługiwane przez klasę generator, która również interfejs wyjściowy bazuje na wzorcu projektowym obserwator. Dodatkowo cała klasa spełnia wymagania stawiane przez funktor, poprzez implementację operatora okrągłych nawiasów. Jest to spowodowane, chęcią ułatwienia uruchamiania procesu generowania pliku  
w osobnym wątku.

Przy instancjonowaniu klasy generującej plik opracowany przez firmę Microsoft, należy podać trzy parametry. Pierwszym jest struktura reprezentujące zależności między publikacjami, natomiast pozostałe dwa to wskaźniki na funkcje, które mają zostać wywołana podczas ogłaszania postępu, oraz po zakończeniu generowania pliku.

Metoda implementująca generację nazywa się process\_impl. Rozpoczyna się od ustawiana używanych styli w arkuszu, co w wyniku zapewni jego stylistyczną spójność. Następnie zostaje stworzone indeksowanie, pomiędzy nagłówkami,   
a numerami kolumn, za pomocą dynamicznych tablic ciągłej alokacji – wektorze, oraz tablic asocjacyjnych pochodzących z biblioteki standardowej.

Zapis do arkusza rozpoczyna się od nagłówków, które są przechowywane  
w wektorze. Na końcu tego oraz wszystkich kolejnych wierszy jest dokładana pojedyncza czarna komórka, co ma służyć, jako graficzne zakończenie wiersza. Spowodowane jest to wadą biblioteki, która posiada bardzo ograniczony zasób działających operacji na zakresach komórek, sprawiając, że kolorowane są całe wiersze, zamiast wyróżnionych zakresów.

W dalszej kolejności następuje iteracja po kolejnych relacjach pomiędzy publikacjami, oraz ich zapis w generowanym arkuszu. Całość zwieńczona jest zapisem na dysk, ze sprawdzeniem powodzenia za pomocą asercji. W trakcie każdej pętli dochodzi niestety do sporej ilości konwersji, co jest spowodowane używaniem, jako łańcuchów znaków klasy QString, przez bibliotekę obsługującą rozszerzenie xlsx.

W funkcji nadrzędnej progress, dostępnej publicznie, następuje wysłanie sygnałów o pełnym zakończeniu generacji pliku.

## Omówienie serializacji

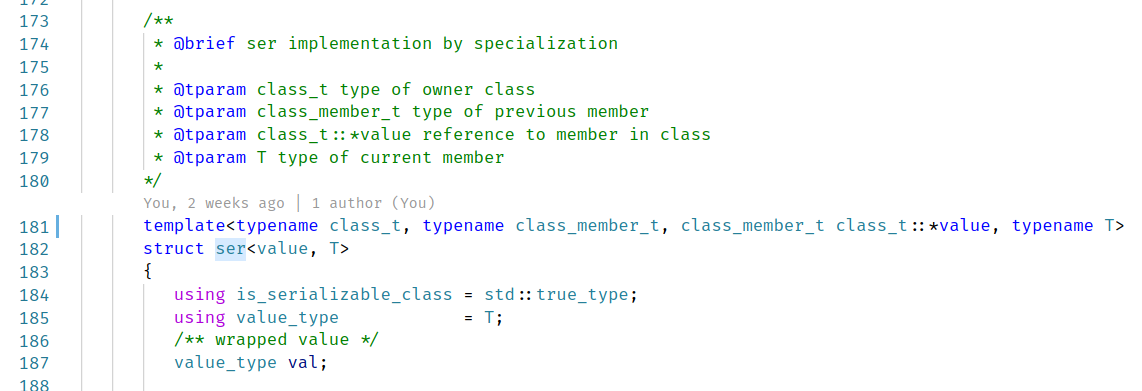
### Zasada działania

System serializacji składa się z szeregu klas i struktur, uzupełniających się w zastosowaniu oraz funkcjonalności. Kluczowymi z punktu widzenia użytkownika klasy są dwie, ser oraz cser. Obie, pełnią funkcje prostego kontenera na dowolny typ, oraz świadczą szereg operatorów, ułatwiających interakcje z posiadanym elementem.

Przekierowane są konstruktory, operatory przypisania, operatory porównania, a także zostały zaimplementowane operatory konwersji. Oprócz wymienionych występują dodatkowo operatory okrągłych nawiasów, które pełnią rolę funkcji dostępowych, ponieważ umożliwiają zarówno zapis jak i odczyt referencyjny wewnętrznego obiektu.

Kluczową, z punktu widzenia pełnionej funkcji, jest implementacja interfejsu akceptora, z wzorca projektowego wizytator. Metody accept w przypadku wspomnianych wcześniej dwóch klas różnią się znacząco swoją rolą, lecz nie aż tak bardzo implementacją. Obie klasy zawierają dwie implementacje metody accept, co jest spowodowane kwestiami optymalizacyjnymi, jako że jedna z metod jest oznaczona, jako stała, co umożliwia kompilatorowi lepsze ułożenie stosów wywołań.

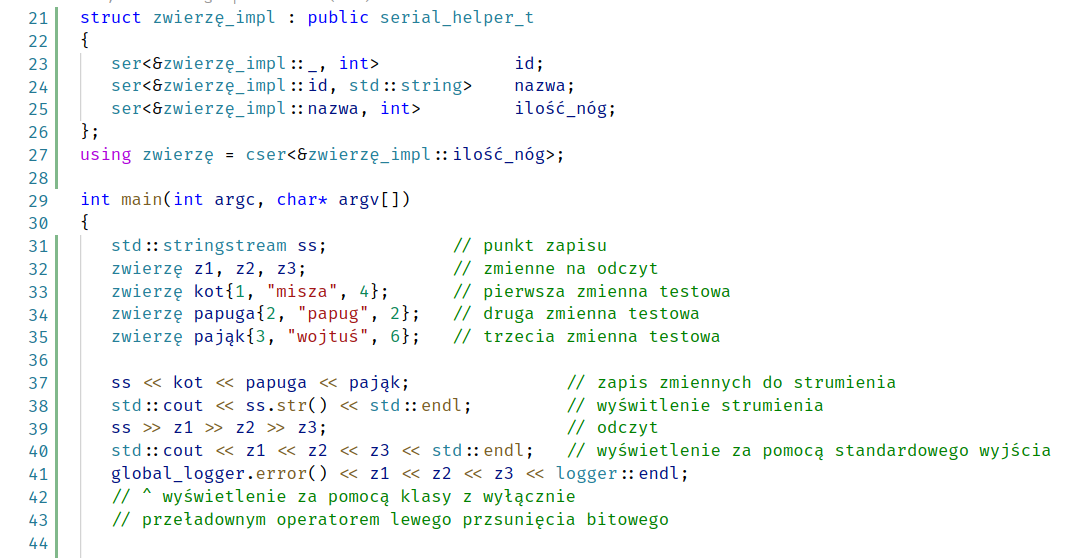
Przed omówieniem metod akceptujących, należy omówić parametry szablonu, jako że to one stanowią kluczową część implementacji. Obie klasy są zadeklarowane za pomocą przeciążenia swojej pustej wersji, jest to spowodowane, zarówno przez wymogi stawiane przez język, jak również chęcią czytelnego zapisu, podczas korzystania z struktur.



Rysunek 28 Początek deklaracji struktury *ser*

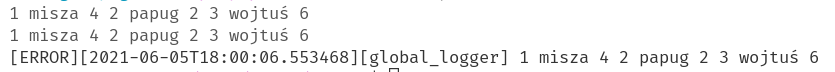
Linia 181 na powyższym zrzucie ekranu (Rysunek 28), zawiera cztery typy. Ostatni z nich to typ enkapsulowany przez omawianą klasę. Pozostałe trzy, są to niezbędne typy pozwalające na stworzenie statycznego wskaźnika na poprzednie pole klasy. Deklaracja klasy oparta, o połączonych za pomocą statycznych wskaźników kolejnych pól w klasie, imituje kontener połączonej listy (ang. linked list).

Takie rozwiązanie umożliwia jednostronne, rekurencyjne, swobodne, statyczne iterowanie po zadeklarowanej w ten sposób klasie, za pomocą wizytatorów. Przykładem zastosowania takiego mechanizmu jest serializacja oraz deserializacja strumieniowa.



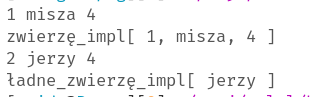
Rysunek 29 Przykład użycia struktur *ser* oraz *cser*

Przykład z powyższego zrzutu ekranu (Rysunek 29), prezentuje użycie serializacji oraz deserializacji. Nie ma potrzeby pisania dodatkowych funkcji, które obsłużą proces wkładania, oraz wyciągania danych ze strumienia. Rozwiązanie to daje sporą elastyczność w zakresie tworzenia sposobów własnych modyfikatorów. Kolejny przedstawiony przykład, zaprezentuje możliwość nadpisania domyślnej reprezentacji oraz tworzenia nowych modyfikatorów. Kolejny przedstawiony przykład, zaprezentuje możliwość nadpisania domyślnej reprezentacji oraz tworzenia nowych modyfikatorów.

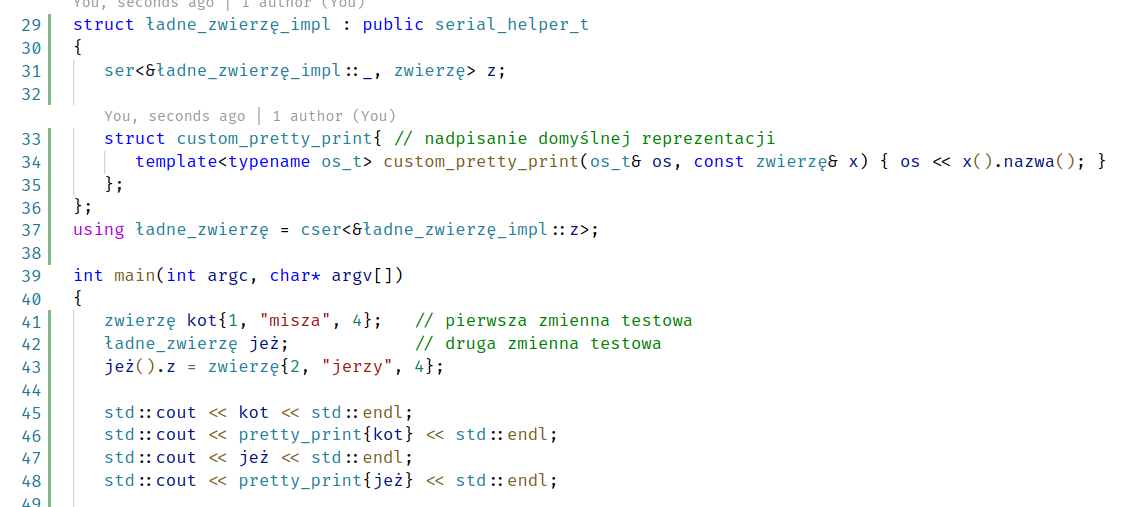


Rysunek 30 Wyjście z powyższego przykładu (Rysunek 29)

Dopisanie struktury, w linii 34, o nazwie ustalonej przez modyfikator pretty\_print, używanego w obecnym projekcie, jako deweloperskie narzędzie do drukowania reprezentacji obiektów wraz z typami, umożliwiło zmianę wyświetlanej wartości, z wszystkiego, na wyłącznie jedno pole.



Rysunek 31 wyjście z powyższego przykładu (Rysunek 30)



Rysunek 32 Przykład nadpisania domyślnego wywołania

Zadaniem metody accept, w przypadku struktury ser, jest przekazanie otrzymanego wizytatora do kolejnego pola, które jest wskazane za pomocą wbudowanego w typ, statycznego wskaźnika, zapisanie wyniku z wizytatora, do jego instancji, a następnie wywołanie go dla siebie. Wymagania, jakie musi spełnić wizytator są określone za pomocą konceptu, co w przypadku niespełnienia wymagań przez wizytator, prowadzi do klarownego błędu, jeszcze podczas kompilacji, co jest zaletą, oferowaną przez najnowszy standard języka.

Zadanie metody akceptującej w kontenerze cser, jest inne. W przeciwieństwie do metody z struktury ser, wpisuje do pustego wskaźnika, adres przechowywanej przez siebie klasy (reprezentuje wskaźnik this), a następnie przekazuje wizytator do ostatniego pola w przechowywanej klasie. Referencje do niego, podobnie jak klasa ser, posiada dzięki statycznemu wskaźnikowi, zawartego w szablonie.

### Problematyka

Największym problemem podczas implementacji, było zapewnienie odpowiedniego przekierowania konstruktorów, oraz operatorów, umożliwiających wygodne korzystanie z klasy enkapsulowanej klasy. Złożoność tkwi w możliwości przychodzących do operatorów oraz konstruktorów danych. Mogą pojawić się, bowiem dane zapakowane zarówno w strukturę ser jak również strukturę cser. Nie można dopuścić do sytuacji, gdzie zostanie podjęta próba przypisania opakowanego typu, do typu podstawowego, co doprowadzi do błędu kompilacji.

Kolejnym problemem, podczas tworzenia kodu wymienionych klas, były ciężkie w zrozumieniu błędy, często niemieszczące się na obróconym pionowo ekranie. Doprowadzało to do absurdalnych sytuacji, gdzie stos błędnego instancjonowania, przewyższał wielokrotnie długość całej implementacji. Wymusiło to opracowywanie rozwiązania w osobnym projekcie, a w drugiej kolejności wklejenie rozwiązania do niniejszego projektu.

Projekt przechodził bardzo burzliwy proces ewolucji, który początkowo nie zakładał innych funkcjonalności, niż serializacja, deserializacja oraz drukowanie wraz z typami (41). Przejście na kolejną warstwę abstrakcji, umożliwiło tworzenie nowych zastosowań, możliwe, że nawet w pełni statycznych operacji iterowanych po polach struktury. Proces ewolucji kodu, można zaobserwować, przez nazewnictwo użyte dla struktur. Początkowo, gdy struktury te miały wyłącznie służyć do serializacji, nazwa ser, jako skrótowiec od angielskiej nazwy serializacji, miał jak najbardziej sens.   
To samo tyczy się nazwy cser, której wytłumaczenie to serializacja klasy w języku angielskim. Aktualnie nazwa powinna odnosić się do możliwości iteracji po typach.

Ostatni problem dotyczy samych modyfikatorów. Podczas implementacji modyfikatora pretty\_print, zachodził częsty problem błędnego przejścia na modyfikator zwykłej serializacji, co wiązało się z nieczytelnym, wyjściem. Spowodowało to konieczność wprowadzenia oznaczania typów (ang. type tagging). Zostało to zaimplementowane, za pomocą prostych struktur, zawierających wyłącznie referencje na oznaczany typ, co powoduje wymuszenie określonej ścieżki podczas instancjonowania typów przez kompilator.

### Wady i zaley

Definitywną zaletą skorzystania z opisanego mechanizmu jest brak konieczności dedykowanego pisania mechanizmów serializacji oraz deserializacji, dla każdego z typów. Umożliwia to ograniczenie pisanego kodu, oraz z uwagi na statyczne rozwiązywanie, redukcję generowanego kodu, jeżeli kompilator nie wykryje użycia dla danych modyfikatorów. Dodatkowo z uwagi na statyczność tego kodu oraz implementacja z wykorzystanie szablonowego meta-programowania umożliwia kompilatorowi bardzo głębokie optymalizacje.

Konstrukcja oparta o prosty kontener, uzależnia możliwość korzystania z kopii binarnej od przechowywanego typu, co w przypadku rozwiązań z sporą ilością wykonywanych kopii jest kluczowe i pozwala wysoce przyspieszyć mechanizmy kopiujące. Zrezygnowanie z mechanizmu kompilacji pozwala również na binarne kopiowanie pomiędzy typami o tych samych wartościach, co zdarza się wykorzystywać w przypadku niektórych implementacjach na niższym poziomie.

Utworzenie interfejsu umożliwia wprowadzenie naturalnych meto dostępu (ang. accessors). Ich użycie w projekcie daje możliwość sprawniejszego debugowania, oraz może poprawić stan hermetyzacji kodu.

Pojawienie się dłuższych nazw typów, wymusza na użytkowniku biblioteki stosowanie aliasów, co ma bezpośredni wpływ na czytelność kodu, oraz intencje jego wprowadzenia.

Opisane tutaj rozwiązanie, jest wolne od makr. Jest to kuszące rozwiązanie, dla projektów celujących w wysoki poziom kodu, a także unikanie przestarzałych rozwiązań.

Naczelną wadą tego podejścia jest redukcja przejrzystości kodu, poprzez utrudniony dostęp do bardziej zagnieżdżonych pól struktur i klas. Wymusza to na użytkowniku klasy stosowanie referencji, jako aliasy do bardziej położonych niżej   
w hierarchii danych w przechowywanych strukturach. Zostało to podkreślone na poniższym zrzucie ekranu (Rysunek 33), gdzie większość inwokacji operatora okrągłych nawiasów to wyłuskanie pola bardziej zagnieżdżonego.

Kolejną wadą jest wydłużony czas kompilacji. Spowodowane jest to potrzebą kompilatora na głębsze przeanalizowanie struktur. Czas ten potrafi być znacząco wydłużony, gdy zostaje wybrany wyższy stopień optymalizacji, wśród udostępnianych przez kompilator.



Rysunek 33 Przykład dostępu do głębiej umieszczonych pól w strukturach

Ostatnią z znaczących wad, jest pojawienie się bardzo długiego błędu, przy niepoprawnym zapisie. Zastosowanie konceptów, znacznie zredukowało ilość miejsc, gdzie takie błędy mogą się pojawić. Warto zwrócić uwagę, że jest to nie jest wyłączny problem, tej biblioteki, lecz wszystkich rozwiązań bazujących na podejściu meta-programowania.

## Narzędzia i produkty uboczne

Rozwijając przedstawiony projekt, powstała paleta różnych narzędzi wspomagających oraz kilka incydentów w innych projektach. Ponieważ udział tych elementów nie pozostał obojętny dla projektu, a nie mają bezpośredniego wpływu na działanie aplikacji, zostały wydzielone do osobnego rozdziału.

### Generowanie bibliotek

Struktura projektu, określona katalogami, ma charakter drzewiasty oraz silnie przewidywalny. Konieczność tworzenia katalogów oraz trzymanie spójności konwencji wymusiło napisanie skryptów automatyzujących proces dodawania kolejnych bibliotek.

Niespełnione aspiracje niniejszego projektu do rangi statusu projektu w pełni międzyplatformowego wymusiły skorzystanie z języka, który również posiada możliwość ewaluacji na wielu platformach. Biorąc pod uwagę łatwość obsługi, skryptowy język Python, został wybrany, jako język wsparcia.

Skrypty zostały umieszczone w dedykowanych katalogach, podkatalogu gen. Pierwszy znajduje się w podkatalogu o nazwie library, kolejny w windows. Nazwy skryptów jasno definiują ich przeznaczenie. Każdy z nich przyjmuje jeden parametr i jest nim odpowiednio nazwa biblioteki i nazwa nowego okna dialogowego.

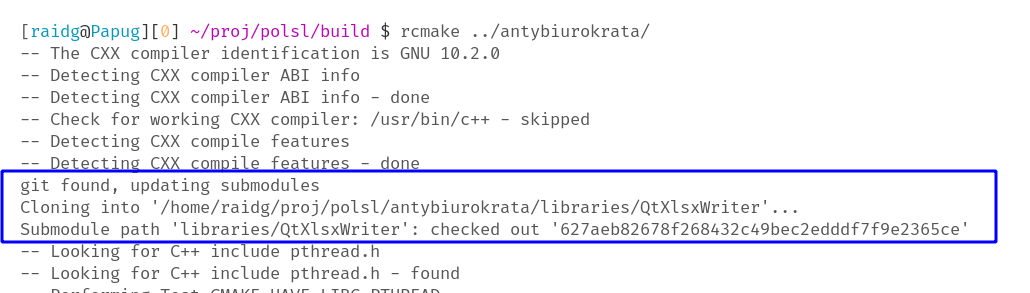
Każdy z tych drobnych programów wykonany z odpowiednim argumentem, tworzy katalog o podanej nazwie w odpowiednim miejscu w projekcie. Następnie generowana jest odpowiednia struktura, a na końcu generowane są pliki, opierając się o dostarczone, wejściowe. Cały proces jest błyskawiczny, a przede wszystkim sprawdzony, ponieważ zdecydowana większość ujętych bibliotek została stworzone przy pomocy opisanych skryptów.

### Asysta z systemu kontroli budowy

Wspomniany wyżej system kontroli wersji cmake, zawiera szereg ciekawych funkcjonalności, umożliwiających wsparcie procesu tworzenia kodu.

#### Pilnowanie aktualności bibliotek

Integracja z programem git, zapewnia, że przed rozpoczęciem skanowania zależności, czy generowania plików MAKEFILE, zostaną pobrane najnowsze wersje podmodułów. Celem demonstracji, po usunięciu jednego z modułów, został on od razu pobrany jeszcze przed rozpoczęciem budowy, umożliwiając pozytywne jej zakończenie, co widać na poniższym przykładzie (Rysunek 34).



Rysunek 34 Pobranie podmodułu przez system kontroli budowy

#### Zapobieganie przypadkowej budowie

#### Generowanie dokumentacji

#### Formatowanie

# Podsumowanie

## Perspektywy rozwoju

## Wnioski

# Bibliografia

1. **SPLENDOR; Biblioteka Główna Politechniki Śląskiej.** Strona Główna Wyszukiwarki Bazy Dorobek. [Online] 29 05 2021. [Zacytowano: 30 06 2021.] https://www.bg.polsl.pl/expertus/new/bib/.

2. **ORCID.** Dokumentacja Interfejsu Sieciowego ORCID. [Online] [Zacytowano: 30 06 2021.] https://pub.orcid.org/v3.0/.

3. **© Elsevier B.V. .** Dokumentacja Interfejsu Sieciowego SCOPUS. [Online] 2021. [Zacytowano: 30 06 2021.] https://dev.elsevier.com/search.html.

4. **Clarivate.** Dokumentacja Interfejsu Sieciowego Web Of Science. [Online] 2021. [Zacytowano: 02 06 2021.] https://developer.clarivate.com/apis/wos.

5. **Microsoft.** Specyfikacja techniczna formatu XLSX. [Online] [Zacytowano: 30 06 2021.] https://support.microsoft.com/pl-pl/office/specyfikacje-i-ograniczenia-programu-excel-1672b34d-7043-467e-8e27-269d656771c3.

6. Specyfikacja plików CSV. [Online] [Zacytowano: 30 06 2021.] https://en.wikipedia.org/wiki/Comma-separated\_values.

7. **Bertocci Vittorio.** Bezpieczeństwo protokołu OAuth 2.0. [Online] 08 01 2019. [Zacytowano: 30 06 2021.] https://auth0.com/blog/oauth2-implicit-grant-and-spa/.

8. **SMARTBEAR.** Strona domowa Swagger. [Online] [Zacytowano: 30 06 2021.] https://swagger.io/.

9. **Müller Philip.** Strona dystrybucji Manjaro. [Online] 2011. [Zacytowano: 30 06 2021.] https://manjaro.org/.

10. **Vinet Judd, Griffin Aaron i Polyák. Levente.** Strona dystrybucji Archlinux. [Online] 2002. [Zacytowano: 30 06 2021.] https://archlinux.org/.

11. Dokumentacja języka C++. *cppreference.* [Online] 11 08 2020. [Zacytowano: 30 06 2021.] https://en.cppreference.com/w/.

12. Specyfikacja notacji wężowej. [Online] 19 05 2021. [Zacytowano: 30 06 2021.] https://en.wikipedia.org/wiki/Snake\_case.

13. Specyfikacja notacji wielbłądziej. [Online] 24 05 2021. [Zacytowano: 30 06 2021.] https://en.wikipedia.org/wiki/Camel\_case.

14. **The Clang Team.** Dokumentacja programu clang-tidy. [Online] 2021. [Zacytowano: 30 06 2021.] https://clang.llvm.org/extra/clang-tidy/.

15. Strona główna programu git. [Online] 2021. [Zacytowano: 30 06 2021.] https://git-scm.com/.

16. **Heesch Dimitri van.** Strona główna narzędzia Doxygen. [Online] 2021. [Zacytowano: 30 06 2021.] https://www.doxygen.nl/index.html.

17. **Codacy.** Strona projektu codacy. [Online] 2021. [Zacytowano: 02 06 2021.] https://www.codacy.com.

18. **Kitware.** Strona projektu cmake. [Online] 2021. [Zacytowano: 02 06 2021.] https://cmake.org/.

19. **Microsoft.** Strona główna projektu Visual Studio Code. [Online] 2021. [Zacytowano: 30 06 2021.] https://github.com/microsoft/vscode.

20. Strona projektu Boost. [Online] 11 12 2020. [Zacytowano: 30 06 2021.] https://www.boost.org/.

21. Repozytorium projektu boost-ext/μt. [Online] 13 05 2021. [Zacytowano: 30 06 2021.] https://github.com/boost-ext/ut.

22. **an-tao.** Repozytorium projektu drogoncpp. [Online] 10 04 2021. [Zacytowano: 31 06 2021.] https://github.com/an-tao/drogon.

23. **Gauniyal Abhinav.** Repozytorium projektu rang. [Online] 24 11 2018. [Zacytowano: 31 06 2021.] https://github.com/agauniyal/rang.

24. **Chambe-Eng Eirik i Nord Haavard.** Strona projektu Qt. [Online] Qt, 2021. [Zacytowano: 02 06 2021.] https://www.qt.io/.

25. **Community Research and Development Information Service.** Strona słownika internetowego Glosbe. *Witryna zamieszczonego przykładu, tłumaczącego słowo framework.* [Online] [Zacytowano: 31 05 2021.] https://app.glosbe.com/tmem/show?id=-2423176738232212855.

26. **Zhang Debao i dand-oss.** Repozytorium projektu QtXlsxWriter. [Online] 09 12 2020. [Zacytowano: 31 06 2021.] https://github.com/dand-oss/QtXlsxWriter.

27. **Tryggeseth Eirik.** *Report from an Experiment: Impact of Documentation on Maintenance.* Department of Computer and Information Science, Norwegian University of Science and Technology. Trondheim : Empirical Software Engineering, Kluwer Academic Publishers, 1997. str. 5, Raport z Eksperymentu.

28. *Does the Documentation of Design Pattern Instances Impact on Source Code Comprehension? Results from Two Controlled Experiments.* **Gravino Carmine, i inni i inni.** Limerick, Ireland : IEEE, 2011. 2011 18th Working Conference on Reverse Engineering. strony 67-76. DOI: 10.1109/WCRE.2011.18.

29. **Sutter Herb.** Herb's last appearance on C9. [os. udziel. wyw.] Charles Channel 9. *Going Deep.* 07 06 2011. https://channel9.msdn.com/Shows/Going+Deep/Conversation-with-Herb-Sutter-Perspectives-on-Modern-C0x11.

30. **Mochocki Krzysztof.** Zastosowanie konceptów w Szablonym Meta-Programowaniu. *Concepts.* [Online] 14 05 2021. [Zacytowano: 1 06 2021.] https://en.wikipedia.org/wiki/Template\_metaprogramming#Concepts.

31. **Microsoft.** Guidelines for Keyboard User Interface Design. [Online] 04 2002. [Zacytowano: 02 06 2021.] https://docs.microsoft.com/en-us/previous-versions/windows/desktop/dnacc/guidelines-for-keyboard-user-interface-design.

32. Tablica kodów ASCII. [Online] [Zacytowano: 02 06 2021.] https://ascii.cl/.

33. Wikipedia. *Strona litery Ó.* [Online] [Zacytowano: 02 06 2021.] https://en.wikipedia.org/wiki/%C3%93.

34. **Hacksplaining.** Hacksplaining. *Protecting Against SQL Injection.* [Online] Hacksplaining. [Zacytowano: 02 06 2021.] https://www.hacksplaining.com/prevention/sql-injection.

35. —. Hacksplaining. *Protecting Your Users Against Cross-site Scripting.* [Online] Hacksplaining. [Zacytowano: 02 06 2021.] https://www.hacksplaining.com/prevention/xss-stored.

36. **Torvalds Linus.** Komentarz Linusa Torvaldsa na temat formatu XML. *.* [Komentarz]. 07 03 2014. https://en.wikiquote.org/wiki/Linus\_Torvalds#2014.

37. Wikipedia. *Strona wzorca projektowego Wizytator.* [Online] [Zacytowano: 03 06 2021.] https://pl.wikipedia.org/wiki/Odwiedzaj%C4%85cy.

38. Wikipedia. *Strona słowa iterator.* [Online] [Zacytowano: 03 06 2021.] https://pl.wikipedia.org/wiki/Iterator.

39. **computerworld.** computerworld. *Strona tłumaczenia słowa spin lock.* [Online] [Zacytowano: 03 06 2021.] https://www.computerworld.pl/slownik/termin/47256/spin-lock.html.

40. **Demin Alexander.** Blok osobisty Alexandra Demina. *Porównanie różnych mechanizmów synchronizacji.* [Online] 05 05 2012. [Zacytowano: 03 06 2021.] https://demin.ws/blog/english/2012/05/05/atomic-spinlock-mutex/.

41. **Mochocki Krzysztof.** GitHub. *Strona zmiany w repozytorium projektu antybiurokrata.* [Online] 10 05 2021. [Zacytowano: 05 06 2021.] https://github.com/raidgar98/antybiurokrata/commit/d517d11e1ddb33004019cae1336f590a6d9dda90.

# Spis ilustracji

[Rysunek 1 Zapytanie na jednym polu - autorze, wygenerowane przez serwis (skala szarości) 8](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817687)

[Rysunek 2 Podgląd specyfikacji technicznej dostarczonej przez serwis ORCID (skala szarości) 9](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817688)

[Rysunek 3 Podgląd specyfikacji technicznej dostarczonej przez serwis SCOPUS (skala szarości) 10](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817689)

[Rysunek 4 Fragment wygenerowanej strony internetowej 13](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817690)

[Rysunek 5 Fragment kodu z komentarzami do wygenerowania dokumentacji 13](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817691)

[Rysunek 6 Raport statycznej analizy kodu (skala szarości) 14](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817692)

[Rysunek 7 Porównanie wyglądu w zależności od wybranego motywu systemu 20](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817693)

[Rysunek 8 Domyślny widok panelu wyszukiwania 22](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817694)

[Rysunek 9 Alternatywny widok panelu wyszukiwania 22](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817695)

[Rysunek 10 Panel współpracowników z przykładowymi danymi (skala szarości) 23](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817696)

[Rysunek 11 Panel publikacji z przykładowymi danymi 25](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817697)

[Rysunek 12 Okienko dialogowe, ze szczegółami przeszukiwania (rozjaśnione) 25](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817698)

[Rysunek 13 Interfejs klasy komunikującej się z bazą dorobek 27](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817699)

[Rysunek 14 Interfejs klasy *demangler* 27](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817700)

[Rysunek 15 Konstruktor struktury *bgpolsl\_repr\_t* 29](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817701)

[Rysunek 16 Przykładowy fragment wyłuskania wartości z danych w formacie *JSON* 32](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817702)

[Rysunek 17 Obiektowa reprezentacja publikacji 33](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817703)

[Rysunek 18 Obiektowa reprezentacja osoby autora 33](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817704)

[Rysunek 19 Fragment klasy - obiektowej postaci zweryfikowanego i jednolitego tekstu 34](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817705)

[Rysunek 20 Deklaracja typu *polish\_name\_t* 35](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817706)

[Rysunek 21 Interfejs klasy konwertującej dane na obiektową postać osoby (autora) 36](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817707)

[Rysunek 22 Metoda wyłuskująca kolejnych autorów z zadanej publikacji 38](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817708)

[Rysunek 23 Deklaracje struktur opisujące zależności pomiędzy publikacjami 40](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817709)

[Rysunek 24 Pętla tworząca zależności, pomiędzy publikacjami 41](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817710)

[Rysunek 25 Kolejność startu nowych wątków. 42](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817711)

[Rysunek 26 Deklaracja uniwersalnego funktora 43](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817712)

[Rysunek 27 Konstruktor klasy głównego okna 44](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817713)

[Rysunek 28 Początek deklaracji struktury *ser* 46](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817714)

[Rysunek 29 Przykład użycia struktur *ser* oraz *cser* 47](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817715)

[Rysunek 30 Wyjście z powyższego przykładu (Rysunek 29) 47](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817716)

[Rysunek 31 wyjście z powyższego przykładu (Rysunek 30) 48](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817717)

[Rysunek 32 Przykład nadpisania domyślnego wywołania 48](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817718)

[Rysunek 33 Przykład dostępu do głębiej umieszczonych pól w strukturach 51](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817719)

[Rysunek 34 Pobranie podmodułu przez system kontroli budowy 52](file:///Z:\home\raidg\proj\polsl\antybiurokrata\engeener_docx\engeneer_project.docx#_Toc73817720)