



Kierunek studiów: Informatyka

Jakub Kubiak (375197) - Lider

Karol Szatkowski (410380)

Jan Słuchocki (375253) - nieaktywny

Katalogowanie plików

File cataloging

Praca inżynierska

napisana pod kierunkiem

dra hab. Michała Hanćkowiaka

*Wyrażam zgodę na złożenie niniejszej pracy
w Dziekanacie Wydziału Matematyki i Informatyki*

..... *podpis promotora*



Data

Poznań, dnia

OŚWIADCZENIE

My, niżej podpisani Jakub Kubiak i Karol Szatkowski, studenci Wydziału Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu oświadczamy, że przedkładaną pracę dyplomową pt: "Katalogowanie plików" napisaliśmy samodzielnie. Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystaliśmy z pomocy innych osób, a w szczególności nie zlecaliśmy opracowania rozprawy lub jej części innym osobom, ani nie odpisywaliśmy tej rozprawy lub jej części od innych osób. Oświadczam również, że egzemplarz pracy dyplomowej w wersji drukowanej jest całkowicie zgodny z egzemplarzem pracy dyplomowej w wersji elektronicznej. Jednocześnie przyjmujemy do wiadomości, że przypisanie sobie, w pracy dyplomowej, autorstwa istotnego fragmentu lub innych elementów cudzego utworu lub ustalenia naukowego stanowi podstawę stwierdzenia nieważności postępowania w sprawie nadania tytułu zawodowego.

[]* - wyrażam zgodę na udostępnienie mojej pracy w czytelni Archiwum UAM

[]* - wyrażam zgodę na udostępnienie mojej pracy w zakresie koniecznym do ochrony mojego prawa do autorstwa lub praw osób trzecich

*Należy wpisać TAK w przypadku wyrażenia zgody na udostępnianie pracy w czytelni Archiwum UAM, NIE w przypadku braku zgody. Niewypełnienie pola oznacza brak zgody na udostępnianie pracy.

.....
.....
.....



Wstęp.....	4
1. Architektura projektu.....	5
2. System Windows®.....	9
3. Strona użytkownika.....	10
4. Baza danych.....	12
5. Analiza danych plików.....	16
6. Pliki multimedialne.....	17
7. Pliki graficzne.....	19
8. Plik konfiguracyjny.....	20
9. Logowanie do instancji programu.....	21
10. Szyfrowanie i deszyfrowanie bazy danych.....	22
11. Komunikacja między programami.....	24
12. Zestawienie zrealizowanych funkcjonalności.....	27
13. Opis wdrożenia.....	33
14. Kryteria akceptacji.....	34

Wstęp

Wprowadzanie porządku do zbiorów danych jest częstym problemem stojącym przed użytkownikami. Wszechobecny zwyczaj zapisywania wszystkich plików do jednego miejsca przy zapisie danych do folderów domyślnych wprowadza bardzo prędko ogromny bałagan. Zazwyczaj próby wprowadzenia porządku zaczynają się gdy uniemożliwia on już sprawne działanie, ale wtedy dla użytkownika, którego jedynymi narzędziami jest eksplorator systemowy i programy do otwierania plików, staje się to nie lada zadaniem.

Niniejszy projekt pozwala użytkownikom urządzeń pracujących w systemie Windows katalogować pliki w szybki, logiczny i efektywny sposób, odszukiwać je według ich metadanych, zawartości i cech oraz przysyłać je między innymi swoimi urządzeniami zasilanymi systemem operacyjnym z rodziny Windows, oraz dzielić się wybranymi danymi z innymi użytkownikami.

Główne założenia projektu

- Użytkownik może dodawać do katalogu dowolną ilość plików znajdujących się na jego urządzeniu,
- Użytkownik ma wybór, jaki rodzaj danych pliku ma być brany pod uwagę podczas procesu porządkowania otrzymanych danych,
- Aplikacja pozwala ekstrahować oraz analizować metadane i zawartość (w całości bądź częściowo) pliku w celu odpowiedniego automatycznego uporządkowania pliku,

- Użytkownik po procesie porządkowania ma możliwość udostępniania plików między swoimi urządzeniami, jak i urządzeniami innych użytkowników za pośrednictwem sieci Internet,
- ~~Aplikacja posiada ułatwienia dostępu w postaci funkcjonalności przetwarzania mowy na tekst (STT) – ze względu na odejście od zespołu osoby wdrażającej tą funkcjonalność została ona wykluczona z końcowej wersji programu.~~

1. Architektura projektu

Program został napisany w przeważającej większości w języku C# jako *klasyczna aplikacja desktopowa*, używając *Windows® Forms* jako rozwiązania zapewniającego interakcję programu z użytkownikiem. Jego porcje są napisane w języku SQL dla bazodanowego *Firebird*, mają one za zadanie wykonanie odpowiednich operacji na utworzonym przez użytkownika w czasie działania programu katalogu i na katalogach obiegowych innych użytkowników. Podczas implementacji funkcjonalności programu nacisk położony został na wykorzystanie udostępnianych w języku C# standardowych bibliotek i komponentów, jednak wiele z funkcjonalności związanych z porządkowaniem danych korzysta z zewnętrznych bibliotek. Gdzie było to możliwe zewnętrzne biblioteki są pełnoprawnymi bibliotekami C#, jednak w trzech przypadkach wymagane jest użycie bibliotek, które napisane są w innym języku. Gdzie było to możliwe, do instalacji i konfiguracji zewnętrznych bibliotek korzystaliśmy z wbudowanego w Visual Studio menedżera pakietów NuGet.

Program składa się zasadniczo z następujących części:

1. *Komponent katalogujący* – część zajmująca się ekstrakcją metadanych z plików rzeczywistych, tworzeniem odpowiadających im plików wirtualnych zawierających te metadane, jak i stworzeniem katalogu wirtualnego i dodaniem do niego w odpowiedni sposób tak utworzonych plików wirtualnych [reprezentowana przez formularz główny i formularz katalogowania danych],
2. *Komponent prezentacyjny* – zapewnia użytkownikowi podgląd katalogów wirtualnych i obiegowych obecnych w programie, pozwala na intuicyjną manipulację ich zawartością i na wywoływanie dla wybranych przez użytkownika folderów i plików wirtualnych pozostałych komponentów programu [zaimplementowana w formularzu głównym],
3. *Komponent ekstrakcji* – część odpowiedzialna za ekstrakcję tekstu z plików multimedialnych (dokładniej z filmów o rozszerzeniu .avi i .mp4) [reprezentowana przez dwa formularze ekstrakcji tekstu],
4. *Komponent porównujący* – w jego skład wchodzi wszystkie części pozwalające na porównywanie plików po cechach wydedukowanych z ich zawartości. Zaliczamy do niego część pozwalającą na uporządkowanie plików dźwiękowych po różnicy podobieństw w ich zawartości (dla plików .mp3 i .wav) [reprezentowana przez formularz porządkowania plików dźwiękowych], część pozwalającą na porównywanie obrazów na podstawie wyliczonej wartości Perceptual Hash [reprezentowana przez formularz porządkowania plików graficznych],

5. *Komponent przeszukujący* - część umożliwiająca przeszukiwanie plików wirtualnym w katalogu lokalnym pod kątem obecności i wartości obecnych w ich metadanych. W połączeniu z komponentem porównawczym i komponentem ekstrakcji stanowi potężne narzędzie do manipulacji katalogiem [reprezentowana przez formularz przeszukiwarki katalogu],
6. *Komponent sieciowy* - część mająca za zadanie umożliwić dwustronną komunikację między instancjami programu celem udostępnienia do podglądu zawartości swoich katalogów (za pośrednictwem katalogów obiegowych), jak i pobrania udostępnionych w nim plików. [wywoływana z formularza głównego].

Biblioteki zewnętrzne wykorzystane do implementacji funkcjonalności programu, to:

- *TikaOnDotNet* – jest to wrapper biblioteki *Tika* (stworzonej przez firmę Apache) implementującej wykrywanie i ekstrakcję metadanych różnych typów plików. Udostępnia on jej funkcjonalności dla programów napisanych w językach platformy .NET. Autorstwo modułu należy przypisywać jest Kevin’owi Millerowi. *Wrapper* ten udostępniony został na licencji *Apache* (kompatybilnej z licencją GPL w wersji trzeciej).
- *IKVM* – zapewnia możliwość stworzenia i wywoływania programów na maszynie wirtualnej Javy dla programów napisanych w językach platformy .NET. Jest ona niezbędna do poprawnego działania kodu stanowiącego podłoże *wrappera TikaOnDotNet*. Stworzona przez Jeroen’a Frijters na licencji GPL w wersji 2.

- *FirebirdSql.Data.FirebirdClient* – biblioteka udostępniana przez twórców silnika bazodanowego *Firebird*, jej zadaniem jest pośredniczenie w komunikacji programu z silnikiem bazodanowym. Jest licencjonowana na zasadach Mozilla Public Licence V.1.1, kompatybilnej z GPL w wersji drugiej.
- *Firebird Embedded* – kolejna z bibliotek udostępniana przez twórców silnika bazodanowego *Firebird*, ma ona za cel zapewnić programowi wbudowane funkcjonalności silnika bazodanowego bez konieczności jego oddzielnej instalacji i konfiguracji. Jest licencjonowana także na zasadach Mozilla Public Licence V.1.1, kompatybilnej z GPL w wersji drugiej.
- *AForge* i *AForge.Math* – są to biblioteki służące do implementacji widzenia maszynowego i sztucznej inteligencji (przez sieci neuronowe), tutaj korzystamy z jej podstawowego komponentu zawierającego narzędzia pomocne przy przetwarzaniu obrazu oraz obliczaniu podobieństwa cosinusowego, na podstawie zadanych na wejściu wektorów. Udostępniane są one na licencji LGPL w wersji trzeciej.
- *MediaToolkit* – pozwala na operacje przetwarzania i konwertowania plików audio i multimedialnych wewnątrz programu. Napisana przez autora o pseudonimie AydinAdn, licencjonowana na zasadach licencji MIT.
- *NAudio* – biblioteka umożliwiająca bezpośredni dostęp do parametrów plików *.wav oraz *.mp3, a także przeprowadzania operacji na plikach o podanych rozszerzeniach (przykładem dla plików *.wav reprezentującego kanały dla trzydziestodwu- oraz szesnastobitowego wejścia kanału miksującego, a dającego wyjście stereo), a dla *.mp3 – odczyt (nieskompresowanych!) danych tegoż pliku) Omawiana biblioteka została stworzona przez Mark’a Heath na licencji Microsoft Public License.

- Tesseract – zapewnia możliwość OCR (Optical Character Recognition), potrzebne do ekstrakcji tekstu z plików multimedialnych. Stworzona przez Charles’a Weld, rozwijana aktywnie przez Google, udostępniona na licencji Apache w wersji drugiej.
- Shipwreck.Phash – jest to wrapper otwartoźródłowej biblioteki Phash, napisany przez osobę o pseudonimie pgrho (na portalu Github). Umożliwia ona liczenie i porównywanie wartości Phash dla plików graficznych. Zarówno biblioteka źródłowa, jak i jej wrapper są udostępniane na licencji GPL w wersji 3.

2. System Windows®

Rodzina systemów Windows®, rozwijanych przez firmę Microsoft, jeszcze w 2016 roku stanowiła 50% systemów operacyjnych wykorzystywanych w komputerach osobistych wśród użytkowników serwisu Stack Overflow¹, oraz niecały 1% systemów zasilających urządzenia mobilne². Biorąc pod uwagę obecność produktu firmy Microsoft na rynku różnego typu urządzeń, sensownym jest stworzenie aplikacji do działania pod kontrolą właśnie tego systemu. Zastosowane biblioteki umożliwiają kompatybilność z każdym systemem rodziny Windows®, który posiada możliwość zainstalowania pakietu .NET w wersji 4.6.1.

¹ <https://insights.stackoverflow.com/survey/2016#technology-desktop-operating-system>

² <http://www.gartner.com/newsroom/id/3516317>

3. Strona użytkownika

Aplikacja komunikuje się z użytkownikiem za pomocą interfejsu okienkowego. Jego budowa została uproszczona w maksymalnym stopniu: Program wizualnie podzielony jest na zakładki, każda z nich skrywa inną funkcjonalność programu.

Pierwsza z nich pozwala na przetestowanie lokalnego katalogu – czy jest on obecny, czy możemy go załadować, i czy jest on poprawnie zbudowany. Po poprawnym zakończeniu testu umożliwiane są dalsze funkcjonalności programu. Następną dostępną opcją jest przeprowadzenie katalogowania – procesu, w którym z wskazanego folderu na dysku twardym pobierze listę wszystkich plików w nim obecnych, wyłuska te, które są obsługiwane przez program i wyekstrahuje z każdego z nich obecne w nim metadane.

Ekstrakcja metadanych dba też, aby powtórzenia danego pliku nie spowodowały wygenerowania wielu plików wirtualnych w katalogu – każde z powtórzeń jest odnotowywane i dodawane jako nowa ścieżka do pliku rzeczywistego w katalogu wirtualnym. W razie późniejszego stwierdzenia nieobecności pliku rzeczywistego pod ścieżką wskazaną przez plik wirtualny, wybierana jest ścieżka z puli ścieżek alternatywnych. Dopiero gdy plik nie ma już pozostałych ścieżek rzeczywistych ulega on skasowaniu z katalogu wirtualnego. Ma to za zadanie trzymanie porządku w katalogu i eliminację plików wirtualnych dla których pliki rzeczywiste już nie istnieją.

Druga zakładka kryje w sobie funkcjonalności przeglądarki katalogów. Pozwala ona, zgodnie z nazwą, podglądać zawartość katalogu lokalnego i, jeżeli są one obecne, także katalogów obiegowych od innych użytkowników. Funkcjonalności udostępniane użytkownikowi zależą od typu przeglądanych katalogów – dla katalogu lokalnego udostępniamy pełen zakres porządkowań danych –

porównywanie plików dźwiękowych pod kątem podobieństwa, porównywanie plików graficznych pod kątem wartości Phash, przeszukiwanie katalogu celem znalezienia plików o konkretnych wartościach lub przedziałach wartości metadanych; jak i pełną możliwość manipulacji plikami wirtualnymi (przenoszenie, zmiana nazwy, kopiowanie, wklejanie, usuwanie). Dla katalogów obiegowych dajemy opcję ściągnięcia pliku do swojego programu. Dla obu typów możliwe jest wyświetlenie metadanych dla danego pliku wirtualnego. Podczas rozwoju przeglądarki katalogów położono nacisk na naśladowanie funkcjonalności dostępnych w systemowym Eksploratorze. Nieznaczne różnice wynikają z trudności w implementacji niektórych z jego cech lub ich znikomym znaczeniu dla przeglądarki katalogów. Wszystkie opcje porządkowań dostępne są z poziomu przycisku Inne albo z ostatniej pozycji listy wywoływanej prawym przyciskiem myszy, o nazwie „Opcje specjalne”.

Opcje sieciowe wywoływane są z poziomu przeglądarki katalogu, jednak dostępne stają się jedynie podczas wyświetlania wszystkich katalogów (pobieranie katalogu obiegowego) i podczas przeglądania konkretnego katalogu obiegowego (opcja ściągnięcia pliku).

Wygląd i struktura aplikacji ma pozwalać nawet osobom o umiejętnościach jego obsługi poniżej zaawansowanych.

4. Baza danych

Baza danych służy w programie do przechowywania informacji o plikach poddanych procesowi katalogowania. O ile w pierwszym semestrze do jej poprawnego działania wymagana była instalacja i poprawnego skonfigurowanie pełnego silnika Firebird w wersji co najmniej 3, to teraz, przez zastosowanie silnika embedded, wymaganie to zostało zniesione. Dużą zaletą samego silnika Firebird jest możliwość stworzenia bazy jako oddzielnego pliku, co pozwala na jej zabezpieczenie w momencie gdy program nie jest uruchomiony (przez szyfrowanie pliku przechowywanego na dysku).

Struktura utworzonej bazy jest łatwo edytowalna z poziomu kodu programu (jakakolwiek jej zmiana wiąże się jednak z koniecznością ponownego wygenerowania katalogu, co wiąże się z utratą jego teraźniejszej zawartości) i definiowana w jednym miejscu. Składa się ona z sześciu tabel:

1. `virtual_folder` – przechowuje foldery wirtualne występujące w katalogu,
2. `metadata_text` – przechowuje wszystkie skatalogowane wirtualne pliki tekstowe (o rozszerzeniach `.txt`, `.csv`, `.tsv`, `.fb2`, `.doc` gdy są one wykryte jako plik tekstowy),
3. `metadata_document` – przechowuje wszystkie skatalogowane wirtualne pliki dokumentów (a więc pliki o rozszerzeniach `.docx`, `.odt`, `.ods`, `.odp`, `.xls`, `.xlsx`, `.pdf`, `.ppt`, `.pptx`, jak i `.doc` gdy są one wykryte jako dokument),
4. `metadata_complex` – przechowuje skatalogowane wirtualne pliki, dla których ilość metadanych nie jest z góry znana (pliki `.htm/.html` i `.xml`),
5. `metadata_image` – przechowuje skatalogowane wirtualne pliki zawierające obrazy (o rozszerzeniach `.jpg`, `.jpeg`, `.tiff`, `.bmp`),

6. metadata_multimedia – przechowuje skatalogowane wirtualne pliki multimedialne (mianowicie pliki .mp4, .avi, .mp3, .wav).

Struktura każdej z tabel jest specyficzna dla metadanych, na których zależy nam najbardziej dla danego typu plików. Uniwersalne między nimi są jednak:

- Nazwa pliku gdy był on katalogowany
- Nazwa pliku w katalogu
- Rozszerzenie pliku
- Pełna ścieżka fizyczna do pliku
- Alternatywne ścieżki fizyczne do pliku (w wypadku znalezienia powtórzeń)
- Rozmiar pliku (w bajtach)
- Data stworzenia pliku
- Data ostatniej modyfikacji pliku
- Data skatalogowania pliku

Z czego dla:

- Plików tekstowych przechowujemy dodatkowo informację o zastosowanym kodowaniu.
- Dokumentów przechowujemy informacje o: języku dokumentu, jego tytule, temacie, opisie, słowach kluczowych, komentarzach, instytucji publikującej, firmie, autorze, ilości: stron/slajdów, tabel, obiektów (na przykład wykresów), obrazów, słów i znaków; jak i informacje o aplikacji, w której został on napisany.
- Plików o nieznanej liczbie metadanych – pierwotne plany zakładały próbę przeparsowania tych plików i wyłuskania pewnej liczby metadanych, jednak ze względu na brak możliwości przeprowadzenia tej operacji przez ekstraktory dostępne w Tika jedyne informacje jakie

zawieramy to tytuł nadany plikowi, jego typ i zastosowane w nim kodowanie.

- Obrazów – przechowujemy w bazie informacje o typie obrazu, dane zawarte w polu komentarza, rozmiary poziomy i pionowy obrazu w pikselach, zastosowaną kompresję i obliczaną w trakcie ekstrakcji wartość Phash.
- Plików multimedialnych – ze względu na zawarcie w tej kategorii plików dźwiękowych i filmów przechowywane jest dla nich wiele metadanych. Należą do nich typ pliku, numer ścieżki, album, czas trwania, autor, rok wydania, gatunek, częstotliwość próbowania, zastosowane kompresory audio/wideo, wielkość klatki w pikselach, ilość klatek na sekundę, pole komentarza i wyekstrahowany tekst.

4.1 Weryfikowanie stanu bazy danych

Z uwagi na fakt, że poprawne działanie programu bardzo mocno zależy od danych w bazie danych przed każdym użyciem programu wymagane jest od użytkownika sprawdzenie stanu bazy danych. Najpierw program sprawdza czy w folderze pod którym powinna się znajdować baza rzeczywiście znajduje się plik CATALOG.FDB – jeżeli go nie ma przechodzi do regeneracji bazy danych z parametrami bazy, które zawarte są w programie. Jeżeli z kolei znajdzie bazę danych to przechodzi do sprawdzenia jej wewnętrznej struktury. Dopiero gdy baza przejdzie pomyślnie to sprawdzenie staje się możliwy podgląd jej zawartości i dodawanie do niej nowych danych. W przypadku negatywnego wyniku sprawdzenia użytkownik informowany jest o tym fakcie, w teraźniejszej implementacji naprawa tej sytuacji wymaga niestety ręcznego usunięcia bazy

danych gdy program jest wyłączony i przegenerowania jej na nowo, z utratą wcześniejszej zawartości.

```
private FbConnectionStringBuilder database_connection_string_builder { get; set; }  
database_connection_string_builder = new FbConnectionStringBuilder();  
database_connection_string_builder.DataSource = "localhost";  
database_connection_string_builder.UserID = [redacted];  
database_connection_string_builder.Password = [redacted];  
database_connection_string_builder.Database = database_path;  
database_connection_string_builder.Charset = "UTF8";  
database_connection_string_builder.ServerType = FbServerType.Default;
```

Ilustracja 1: Przygotowanie danych połączenia z bazą

4.2 Wirtualizacja w bazie danych

Celem zasadniczym przeprowadzenia wirtualizacji danych skatalogowanych do danych w katalogu jest odłączenie informacji o nich od wymogów lokalnego systemu plików.

W przypadku korzystania z systemowego eksploratora wyszukujemy zazwyczaj pliki po informacjach, jakie o nich mamy (na przykład po nazwie czy typie, rzadziej po konkretnym rozszerzeniu czy innych metadanych), jednak każde takie poszukiwanie wiąże się z wielokrotnym kosztem wykonywania kilkukrotnie takiej operacji, jak i przemieszczenia pliku znalezionego w miejsce w którym chcemy go mieć (szukałem i znalazłem, to teraz skopiuję/wytnę tak, żebym nie musiał szukać). Zaletą posiadania zwirtualizowanych informacji o plikach jest fakt, że nie musimy przesuwać na dysku ani jednego pliku, żeby uzyskać pożądaną przez nas porządek – możemy przesuwać nowe pliki do folderu katalogowanego kiedy tylko chcemy i porządkować je zaraz po katalogowaniu w zasadzie dowolnym porządku.

Są one dostępne tak, jak byłyby dostępne z eksploratora systemowego, ale mają też dodatkową zaletę że przy kopiowaniu do nowego foldera wirtualnego nie

kopiuujemy całego pliku, a jedynie jego wirtualną reprezentację. Nie marnujemy miejsca ani uwagi na stworzenie i trzymanie w naszej pamięci miejsca gdzie umieściliśmy nową kopię, możemy mieć ich w katalogu bardzo wiele a mimo to nie spowolnić w znaczący sposób działania programu.

5. Analiza danych plików

Program w procesie ekstrakcji metadanych z plików w czasie katalogowania wykonuje analizę tych metadanych. W różnych plikach pola w których zakodowane są metadane często mają różne nazwy, stąd konieczność analizowania ekstrahowanych danych w locie celem przyporządkowania najczęściej powtarzającej się, niepustej zawartości pola. W programie osiągnięte jest to przez zastosowanie zbiorów porządkujących – są to stworzone na podstawie obserwacji wyników ekstrakcji metadanych zbiory, które zawierają możliwe nazwy pól, w których znajdziemy jeden konkretny typ metadanej (na przykład informacje o ilości stron dla dokumentów mogą wystąpić w procesie ekstrakcji pod pięcioma różnymi nazwami).

Osobna analiza metadanych zachodzi w momencie wpisania ich do bazy danych, jest ona przeprowadzana przez silnik bazodanowy i ma na celu nie dopuszczenie do wpisania do bazy nieprawidłowo sformatowanych danych. Jeżeli dana wartość jest niezgodna z przyjętą definicją pola przeznaczonego na jej przechowywanie w tabeli bazy danych, to jest ona zastępowana pustą informacją.

6. Pliki multimedialne

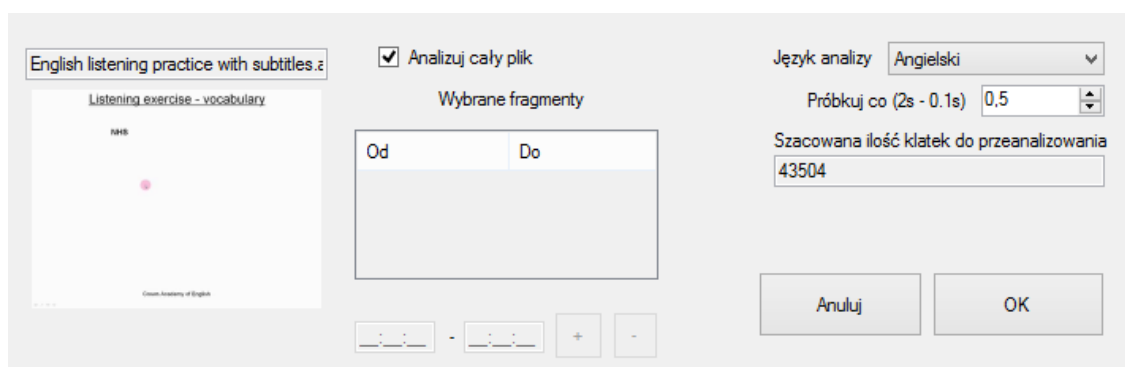
6.1 Porównywanie plików audio (*.wav, *.mp3)

Aplikacja potrafi dokonywać analizy spektrogramu pliku dźwiękowego. Funkcjonalność ogranicza się tutaj wyłącznie do pliku z rozszerzeniem *.wav. Jednakże, dzięki implementacji konwertera pliku o wyżej wymienionym rozszerzeniu do *.mp3, problem wyłączności znika, i pliki *.mp3 stają się zdadne do dalszych analiz. Tutaj należy mieć jeszcze na uwadze niepotrzebnie pozostawione pliki *.wav na dysku – to jednak też nie problem, gdyż przy pomocy modułu obsługującego pliki, istnieje możliwość usunięcia niechcianych pozostałości w zasobach maszyny. Porównanie spektrogramów plików dźwiękowych pozwoli wykryć duplikaty, nawet gdy mamy do czynienia z kopiami, które powstały zaszumionym środowisku – wówczas proces „odszumiana” takich plików nie zawsze jest skuteczny, a próby „obejścia” tego problemu odbijają się echem na jakości pliku audio. Analiza spektrogramu pozwala zredukować entropię danych, przyspieszając pracę poprzez zredukowanie obszaru wyszukiwania w katalogu wirtualnym.

6.2 Wyciąganie danych z pliku video- napisy (*.mp4, *.avi)

Jeżeli aplikacja wykryje, że pośród plików do skatalogowania znajduje się multimedialny plik video, będzie mieć możliwość dokonania ekstrakcji napisów znajdujących się wewnątrz filmu. Ekstrakcja ta oparta jest o narzędzie *Tesseract OCR*. *Tesseract* analizuje obrazy pod kątem występowania w nich znaków z alfabetu danego języka. By umożliwić mu działanie na obrazie zawartym w filmie, aplikacja zapisuje pojedyncze klatki z pliku, na których przeprowadzane jest rozpoznawanie obrazu.

Pozyskiwanie klatek z pliku video, oraz metadane związane z pozyskiwaniem ich uzyskiwane są dzięki bibliotece *MediaToolkit*. Napisy ekstrahowane są z przeanalizowanych fragmentów pliku według parametrów ustalonych przez użytkownika. Wyekstrahowane dane wykorzystywane są do późniejszego katalogowania i wyszukiwania pliku.



Ilustracja 2: Okno wyboru opcji ekstrakcji napisów z pliku video

7. Pliki graficzne

Nową funkcją jest także analiza porównawcza wybranych przez użytkownika plików graficznych. Spośród selekcji użytkownik wybiera jeden plik, który zostanie przyrównany do pozostałych – od tego momentu program, na podstawie dystansu Hamminga między wartościami Phash dla każdego z nich, stwierdzi wzajemny stopień podobieństwa. Pierwszy i najwyższy stopień podobieństwa to kopie wybranego obrazu – zawierają one obrazy, dla których wartość Phash była identyczna. Zawierać się w nich powinny kopie obrazu w innych formatach lub zeskalewane w stosunku do obrazu badanego. Następnie przechodzimy przez trzy pozostałe stopnie podobieństwa do obrazu źródłowego – bardzo podobne, podobne i niepodobne.

Wartości Phash obrazów z tych kategorii coraz bardziej oddalają się od tej w wskazanym obrazie, jednak ze względu na ograniczony jego rozmiar (64 bitowa liczba całkowita) i na złożony proces wyliczania jego wartości, podobieństwa pomiędzy obrazami mogą być dla użytkownika niejasne. Stąd jest to bardziej narzędzie do znajdowania kopii obrazu w różnych formatach i rozmiarach niż pełnoprawny analizator zawartości.

8. Plik konfiguracyjny

Niektóre dane programu różnią się w zależności od użytkownika. Zakodowane na twardo w programie dane nie wystarczą by stworzyć program, w którym elementy (takie jak salt hasła) różnią się między instancjami i mogą zmieniać w trakcie działania programu, oraz muszą być zachowywane pomiędzy poszczególnymi jego uruchomieniami. Dlatego zdecydowaliśmy się wcielić do struktury programu plik konfiguracyjny (lub inaczej *config*).

W pliku config programu zapisywane są informacje takie, jak:

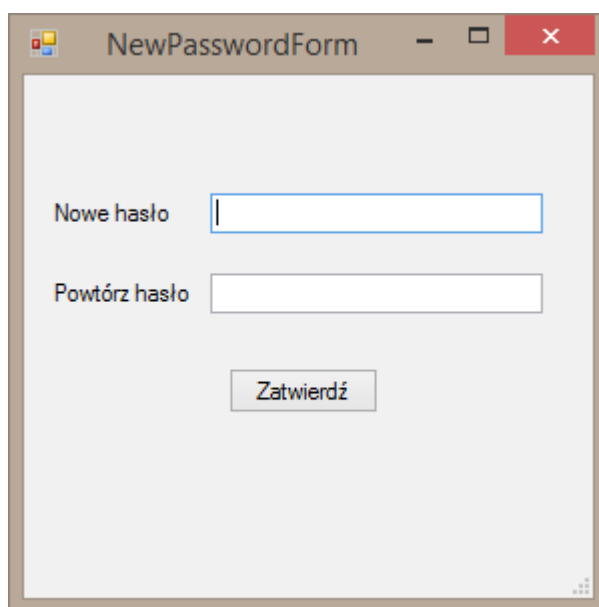
- *digest hash* oraz *salt* hasła logowania do instancji programu
- *salt* oraz *wektor inicjalizacyjny* wykorzystywane do szyfrowania bazy danych programu
- lokalizacja silnika bazy danych
- lokalizacja lokalnej bazy danych, oraz baz danych otrzymanych od innych użytkowników
- lokalizacja programu
- maksymalny czas oczekiwania na dochodzące pakiety TCP w komunikacji sieciowej

Do zarządzania wartościami zawartymi w bazie danych stworzona została statyczna klasa *ConfigManager*. Klasa ta pozwala na odczyt oraz modyfikację zawartości pliku konfiguracyjnego, odszukując elementy jego zawartości, interpretując ją na zasadzie połączenia klucz-wartość.

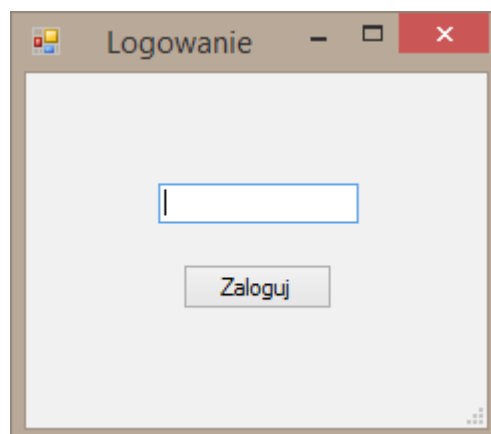
Sam plik konfiguracyjny zapisany jest w formacie XML, który umożliwia łatwe znalezienie szukanej zawartości. *ConfigManager* wykorzystuje klasy *Configuration* oraz *ConfigurationManager* zawarte w namespace *System.Configuration*.

9. Logowanie do instancji programu

Aby zapewnić bezpieczeństwo danych przechowywanych w danych programu, takich, jak katalog użytkownika, oraz by nie dopuścić do dostępu do sieci przez osoby niepożądane, program wspiera funkcjonalność logowania. Przy pierwszym uruchomieniu programu, użytkownik proszony jest o wprowadzenie hasła logowania do instancji programu.



Hasło jest później wykorzystywane do szyfrowania i deszyfrowania bazy danych zawierającej katalog programu. Przy każdym kolejnym uruchomieniu programu użytkownik zostanie poproszony o podanie hasła. Jeżeli użytkownik nie jest w stanie podać poprawnego hasła, nie będzie w stanie dostać się do danych zapisanych w bazie danych programu. Korzystanie z programu nie będzie możliwe do czasu usunięcia pliku bazy danych oraz danych logowania z pliku konfiguracyjnego.



10. Szyfrowanie i deszyfrowanie bazy danych

W lokalnej bazie danych programu przechowywane są dane dotyczące lokacji i zawartości plików znajdujących się na dysku użytkownika. Aby ukryć te dane przed dostępem nieautoryzowanych osób trzecich, baza danych jest szyfrowana. Niestety, mobilna baza danych Firebird nie wspiera szyfrowania rekordów, w związku z czym szyfrowanie musi odbyć się w inny sposób. Plik bazy danych pozostaje odszyfrowany jedynie wtedy, gdy jest w użyciu przez instancję programu.

Nazwa	Data modyfikacji	Typ
 externals	2018-01-05 23:16	Folder plików
 catalog.fdb	2018-01-07 14:31	Plik FDB

W chwili zamknięcia plik bazy danych zostaje zaszyfrowany, by potem zostać odszyfrowany dopiero w momencie ponownego uruchomienia programu i podania prawidłowego hasła logowania.

Nazwa	Data modyfikacji	Typ
 externals	2018-01-05 23:16	Folder plików
 encrypted_catalog	2018-01-07 14:30	Plik

Baza danych jest szyfrowana i deszyfrowana algorytmem AES, którego klucz generowany jest na podstawie hasła wpisywanego na początku działania programu, oraz wektora inicjalizacji, który generowany jest przy pierwszym szyfrowaniu bazy danych, oraz zapisywany w pliku konfiguracyjnym. W tym celu program wykorzystuje klasę *RijndaelManaged*, oraz strumienie *FileStream* i *CryptoStream*. *RijndaelManaged* generuje klucz na podstawie zapewnionych mu danych (hasło programu i wektor inicjalizacji), oraz odczytuje plik zaszyfrowany/zdeszyfrowany, przetwarzając odczytane dane i zapisując odpowiednio jako zdeszyfrowane/zaszyfrowane.

11. Komunikacja między programami

Komunikacja sieciowa między instancjami programu odbywa się w wątkach osobnych niż wątek interface'u użytkownika. Jest tak z kilku powodów. Po pierwsze, gdyby przesył danych odbywał się w wątku głównym programu, program mógłby na czas nieokreślony zostać zatrzymany na czas przesyłania lub otrzymywania dużej ilości danych. W tym przypadku niemożliwe byłoby nie tylko korzystanie w międzyczasie z programu, ale i zatrzymanie transferu danych w czasie ich przesyłania. Program implementujący funkcjonalność przesyłania plików, zwłaszcza o dużych rozmiarach, powinien zezwalać na anulowanie ich przesyłania. Dodatkowo, umieszczenie każdego połączenia sieciowego w osobnym wątku pozwala na jednoczesne utrzymywanie kilku połączeń. W ten sposób każda instancja programu jest w stanie jednocześnie spełniać rolę serwera przesyłającego dane, jak i klienta żądającego danych od innych użytkowników.

Program przygotowany jest do nawiązania komunikacji dzięki wykorzystaniu dwóch klas dostępnych w języku C#: `TcpListener`, oraz `TcpClient`. Każda instancja programu spełnia rolę zarówno serwera, jak i klienta w komunikacji z inną instancją.

Rolę serwera spełnia od początku działania programu. Po zalogowaniu do instancji programu uruchamiany jest wątek serwera, w którym `TcpListener` nasłuchuje na nadchodzące połączenia. Po wykryciu próby nawiązania połączenia, tworzy obiekt klasy `TcpClient` odpowiadający klientowi nawiązującemu połączenie. Ten obiekt przekazywany jest do kolejnego wątku, w którym odbywa się wymiana danych między stronami przez strumień sieciowy.

Nasłuchiwanie jest czynnością blokującą, w związku z czym nie może być przeprowadzane w tym samym wątku, w którym przeprowadzane są inne operacje. Wątek nasłuchiwania trwa aż do wyłączenia programu, kiedy to otrzymuje sygnał przerwania, nakazujący zakończyć nasłuchiwanie i opuścić wątek.

Rolę klienta instancja programu spełnia każdorazowo gdy użytkownik wyśle do innego użytkownika zapytanie o przesłanie danych. Tworzony jest wtedy obiekt klasy `TcpClient`, który służy do komunikacji z instancją programu, od której dane mają być pozyskane. Wątek, w którym odbywa się komunikacja tworzony jest, gdy połączenie zostaje nawiązane, oraz kończy się wraz z zakończeniem procesu komunikacji.

Przesyłanie danych między instancjami programu odbywa się z wykorzystaniem protokołu TCP. Na tym protokole zbudowany jest pseudo-protokół transferu danych. Protokół ten wykorzystywany jest zarówno do przesyłu danych komunikacyjnych między instancjami programu użytkowników, jak i do transferu plików. Pakiety danych posiadają stałą strukturę, która pozwala zapewnić jednoznaczność interpretacji komunikatu, oraz wyznacza granice między poszczególnymi pakietami danych przesyłanych w strumieniu.

Pakiet danych zawsze posiada następującą budowę:

- Bajt inicjalizacji – 1 bajt, stała wartość,
- Bajt zapytania – 1 bajt, mówi klientowi, jak interpretować nadchodzący pakiet danych,
- Rozmiar nadchodzącego pakietu- 4 bajty, otrzymane dane interpretowane są jako liczba całkowita wskazująca, ile bajtów będzie miał pakiet danych,

a zatem ile bajtów ze strumienia komunikacji program powinien odczytać i zinterpretować jako należące do tego samego pakietu,

- Bajt separacji – 1 bajt, stała wartość,
- Dane dodatkowe pakietu – Rozmiar może wahać się między 0 a 4096 bajtami. Ich ilość określona jest przez 4 bajty rozmiaru, a interpretacja zależy od wartości bajtu zapytania. Dzięki przesłanej wcześniej informacji o rozmiarze tego pakietu, mamy pewność, że ewentualne informacje oczekujące w strumieniu nie należące do bieżącego pakietu nie zostaną odczytane jako jego część.

Bajty: inicjalizacji i separacji służą jedynie poszukiwaniu możliwych błędów występujących w czasie przesyłu, bądź wykrywaniu hałasu. Porównując ich wartość z założoną dla nich stałą wartością można, każdorazowo, z prawdopodobieństwem $255/256$ zidentyfikować taki pakiet danych jako niepoprawną próbę nawiązania komunikacji.

12. Zestawienie zrealizowanych funkcjonalności

12.1 Semestr pierwszy:

- Interface użytkownika:
 - Szkielet interfejsu użytkownika (wygląd zakładki *Kryteria katalogowania*) – Jan Słuchocki,
 - Stworzenie i implementacja logiki przycisku *Kataloguj* – Jan Słuchocki, (do momentu wywołania okna *Ekstrakcja metadanych* – Jakub Kubiak)
 - Stworzenie i implementacja logiki okna *Ekstrakcja metadanych* – Jakub Kubiak,
 - Stworzenie i implementacja logiki przycisku *Test bazy* – Jakub Kubiak,
 - Stworzenie i implementacja logiki zakładki umożliwiającej testowanie bazy danych – Jan Słuchocki, Jakub Kubiak,
 - Stworzenie szkieletu i implementacja wszystkich kontrolek i elementów graficznych zakładki *Katalog* – Jakub Kubiak,
 - Stworzenie szkieletu i implementacja wszystkich kontrolek i elementów okna *Opcje specjalne* – Jakub Kubiak,
 - Stworzenie szkieletu i implementacja wszystkich kontrolek i elementów okna *Katalogowanie plików dźwiękowych* – Jan Słuchocki
 - Stworzenie szkieletu i implementacja wszystkich kontrolek i elementów okna *Ekstrakcja tekstów* – Karol Szatkowski

- Finalny wygląd zakładki *Kryteria katalogowania* – Jan Słuchocki,
- Finalny wygląd zakładki *Katalog* – Jakub Kubiak,
- Ekstrakcja metadanych:
 - Pierwszy prototyp ekstrakcji metadanych – Jan Słuchocki,
 - Ekstrakcja istotnych z punktu widzenia programu metadanych z wynikowych metadanych, za pomocą zbiorów porządkujących – Jakub Kubiak,
 - Dodawanie przetworzonych przez zbiory porządkujące metadanych do bazy danych – Jakub Kubiak,
- Wirtualny katalog:
 - Stworzenie struktury tabel i rekordów katalogu – Jakub Kubiak
 - Stworzenie reprezentacji struktury tabel i rekordów katalogu wewnątrz programu – Jakub Kubiak,
 - Utworzenie skryptów tworzących i sprawdzających budowę bazy danych, jak i istnienie samej bazy danych – Jakub Kubiak,
 - Obsługa wybierania plików i folderów w reprezentacji graficznej katalogu, jak i przesyłaniu tego wyboru do dalszej analizy – Jakub Kubiak
 - Obsługa kopiowania plików i folderów wirtualnych wewnątrz katalogu i jego reprezentacji graficznej – Jakub Kubiak,
 - Obsługa usuwania plików i folderów wirtualnych wewnątrz katalogu i jego reprezentacji graficznej – Jakub Kubiak,
 - Obsługa zmiany nazwy plików i folderów wirtualnych wewnątrz katalogu i jego reprezentacji graficznej – Jakub Kubiak,

- Obsługa otwierania plików programem skojarzonym z danym rozszerzeniem przez system operacyjny z wewnątrz katalogu –
Jakub Kubiak
- Cała logika ekstrakcja tekstu z plików multimedialnych –
Karol Szatkowski
 - Okno z przyciskiem inicjalizacji procesu ekstrakcji napisów
 - Okno z ustawieniami ekstrakcji
- Cała logika analizy zawartości, spektrogramów, zestawienia czasu trwania pliku, a także podobieństwa, odtwarzania plików audio i nawigacji –
Jan Słuchocki
 - Obsługa klawiatury dla wspomnianej części – skróty klawiaturowe
 - PreviousTrack – aktywuje ścieżkę z indeksem mniejszym o jeden, aniżeli indeks ścieżki obecnie zaznaczonej na liście. Jeżeli aktywowana ścieżka jest ostatnią pozycją na liście, aktywowana jest pierwsza „od góry”
 - PlayPause – odtwarza aktywowaną ścieżkę
 - StopTrack – zatrzymuje odtwarzaną ścieżkę
 - NextTrack – aktywuje ścieżkę z indeksem większym o jeden, aniżeli indeks ścieżki obecnie zaznaczonej na liście. Jeżeli aktywowana ścieżka jest pierwszą pozycją na liście, aktywowana jest pierwsza „od dołu”
- Finalna integracja projektu przed obroną – Jakub Kubiak, po konsultacji i z pomocą Jana Słuchockiego i Karola Szatkowskiego.

12.2 Semestr drugi:

- Interface użytkownika:
 - Usunięcie pozostałości po elementach graficznych używanych do wyświetlania danych służących do debugingu – Jakub Kubiak,
 - Usprawnienie nawigacji w przeglądarce katalogu – Jakub Kubiak,
 - Wprowadzenie okien logowania i tworzenia nowego konta – Karol Szatkowski,
 - Wprowadzenie okna przeszukiwania katalogu pod kątem wartości metadanych plików wirtualnych – Jakub Kubiak,
 - Wprowadzenie okna porównywania obrazów na podstawie podobieństwa funkcji Phash – Jakub Kubiak,
- Przeszukiwanie plików lokalnych bazy danych:
 - Implementacja przeszukiwania określonych typów plików wirtualnych z wybranego zakresu plików i folderów wirtualnych – Jakub Kubiak
 - Implementacja wprowadzenia interesujących użytkownika w wyszukiwaniu pól metadanych, zarówno specyficznych dla danego typu danych, jak i globalnych dla wszystkich z nich – Jakub Kubiak,
 - Implementacja definicji wartości interesującej w wyszukiwaniu dla konkretnej metadanej dbając o typ danych w niej obecnej – Jakub Kubiak,
 - Wprowadzenie możliwości przeszukiwania po przedziałach możliwych wartości zmiennych (dla zmiennych będących datami i liczbami) – Jakub Kubiak,
 - Wprowadzenie możliwości przeszukiwania po frazie występującej w tekście lub dokładnej frazy w tekście (dla zmiennych będących tekstem) – Jakub Kubiak,

- Wprowadzenie możliwości wyszukiwania zmiennej o wartości niezdefiniowanej – Jakub Kubiak,
- Szyfrowanie bazy danych:
 - Implementacja szyfrowania bazy danych w stanie spoczynku programu – Karol Szatkowski,
 - Implementacja deszyfrowania bazy danych po poprawnej weryfikacji użytkownika – Karol Szatkowski,
 - Przechowywanie w bezpieczny sposób danych służących do szyfrowania i deszyfrowania bazy danych – Karol Szatkowski, Jakub Kubiak,
- Logowanie użytkownika:
 - Implementacja tworzenia konta użytkownika w przypadku jego braku – Karol Szatkowski,
 - Implementacja logowania użytkownika już utworzonego – Karol Szatkowski,
 - Przechowywanie w bezpieczny sposób danych służących do weryfikacji użytkownika – Karol Szatkowski (prototyp), Jakub Kubiak i Karol Szatkowski (finalna wersja),
- Plik konfiguracyjny:
 - Stworzenie logiki tworzenia, uzupełniania i czytania pliku konfiguracyjnego – Karol Szatkowski,
 - Inicjalizowanie pliku konfiguracyjnego w programie, wypełnienie go danymi i czytanie roboczych danych – Karol Szatkowski, Jakub Kubiak,
 - Kontrolowanie poprawności danych zawartych w pliku konfiguracyjnym – Karol Szatkowski, Jakub Kubiak,

- Bezpieczeństwo programu:
 - Bezpieczne generowanie kluczy i danych wrażliwych, jak i poprawne ich składowanie w programie – Karol Szatkowski,
 - Uzbrojenie programu na próby wczytania uszkodzonych/niepoprawnych plików – Jakub Kubiak,
- Funkcjonalności sieciowe:
 - Katalog obiegowy:
 - Wprowadzenie możliwości oznaczania danych z katalogu lokalnego użytkownika celem ich zawarcia w katalogu obiegowym – Jakub Kubiak,
 - Wprowadzenie możliwości oznaczania danych z katalogu lokalnego użytkownika celem oznaczenia możliwości ich ściągnięcia w katalogu obiegowym – Jakub Kubiak,
 - Wprowadzenie możliwości oznaczania danych z katalogu lokalnego użytkownika celem możliwości ich ściągnięcia bez pytania w katalogu obiegowym – Jakub Kubiak,
 - Poprawne generowanie katalogu obiegowego wraz z obsługą przypadków brzegowych i zachowaniem struktury drzewiastej pomiędzy katalogiem lokalnym a obiegowym – Jakub Kubiak,
 - Stworzenie możliwości przesyłania katalogów obiegowych między programami – Karol Szatkowski (wymagane do jej obsługi funkcjonalności), Jakub Kubiak (kod implementujący),
 - Stworzenie możliwości przeglądania katalogów obiegowych uzyskanych od innych użytkowników – Jakub Kubiak,
 - Wprowadzenie możliwości podglądu metadanych w katalogu obiegowym – Jakub Kubiak,

- Przesyłanie plików:
 - Stworzenie logiki nawiązującej połączenia między stronami – Karol Szatkowski,
 - Stworzenie logiki akceptującej połączenia przychodzące – Karol Szatkowski,
 - Stworzenie indeksu dozwolonych komunikatów pomiędzy stronami, wraz z ich znaczeniem w komunikacji – Karol Szatkowski,
 - Stworzenie części nasłuchującej i synchronizacja jej pracy z resztą programu – Karol Szatkowski,
 - Obsługa logiki przesyłania arbitralnie dużego pliku – jego dzielenie na pakiety i kontrola ich przesyłu – Karol Szatkowski,
 - Wprowadzenie możliwości przesyłania plików pomiędzy programami (w opcjach z pytaniem do posiadacza pliku i bez) – Karol Szatkowski,

13. Opis wdrożenia

Wdrożenie programu osiągnięto przez umieszczenie jego gotowej wersji w formie kodu skompilowanego w programie Visual Studio 2015 wraz z niezbędnymi do działania bibliotekami i instrukcjami instalacji, jak i kodu źródłowego, na portalu GitHub, pod adresem <https://github.com/Qbded/PRI-Repo/commits/master>.

14. Kryteria akceptacji

Zasadniczo kryteria akceptacji projektu stanowi osiągnięcie wszystkich założeń projektowych wymienionych wcześniej w stopniu zadowalającym komisję pod koniec przedmiotu PRI2. W drugim semestrze za dwa wymagające dalszego rozwoju kryteria ustaliliśmy uzupełnienie i rozwinięcie obecnych już funkcjonalności porządkowania danych, jak i stworzenie możliwości dzielenia się plikami między użytkownikami programu.

Wyłączony został z założeń projektowych punkt: Aplikacja posiada ułatwienia dostępu w postaci funkcjonalności przetwarzania mowy na tekst (STT), jako że w przewidzianym czasie jego implementacji osoba, której zadaniem miało być jego wypełnienie odeszła od grupy projektowej. Ze względu na pozostałe siły robocze i niewielką wartość użytkową tego założenia w porównaniu z innymi wymaganiami pozostałymi do implementacji, zostało ono wyłączone z ostatecznych założeń projektowych.

Na osiągnięcie uzupełnienia i rozwinięcia obecnych już funkcji porządkowania danych przeznaczaliśmy połowę dostępnego nam czasu (od początku przedmiotu PRI2 do terminu pierwszego przyrostu), podczas niego program został wzbogacony zarówno o możliwości przeszukiwania katalogu pod kątem zawartości metadanych w plikach wirtualnych, jak i nowego porządkowania na podstawie wartości Phash dla plików graficznych. Dodanie tych funkcjonalności uznaliśmy za wskazane ze względu na odejście członka zespołu odpowiedzialnego za rozwój metody porządkowania plików dźwiękowych – z tylko jedną rozwijaną dalej metodą porządkowania program byłby wysoce wybrakowany w wypełnianiu swojej roli.

W międzyczasie także dodane zostały plik konfiguracyjny, przechowujący dotychczas zawarte w kodzie programu zmienne krytyczne do jego funkcjonowania, jak i szyfrowanie zawartości katalogu, gdy program jest wyłączony. Celem osiągnięcia tego szyfrowania tworzone jest lokalnie konto, na podstawie podanego przez użytkownika hasła. Jest ono przechowywane w możliwie bezpieczny sposób w wcześniej wymienionym pliku konfiguracyjnym.

Funkcjonalności związane z dzieleniem się plikami między użytkownikami programu zajęły czas od pierwszego przyrostu aż do końca roku 2017.

Długotrwała praca koncepcyjna, rozwój i późniejsze porzucenie nierokującego kodu rozwojowego w połowie implementacji, jak i późniejsze próby rozwiązania problemu z ruchem ponad zaporami ogniowymi doprowadziły do szkieletowego stanu implementacji tej konkretnie grupy funkcjonalności. Z tych właśnie względów nie są one zaimplementowane w kryptograficznie bezpieczny sposób. Funkcjonalności sieciowe przez sieć Internet wymagają w wypadku połączeń w sieciach z translacją adresów (wszystkie wypróbowane sieci z routerami i stosujące adresy IPv4) i w przypadku odcięcia możliwości nawiązywania połączeń przychodzących na jakikolwiek port przez firewall albo poprawnego skonfigurowania zapory i wirtualnego serwera na routerze (czego nie jest w stanie zrobić użytkownik, o którym zakładamy że posiada minimalną wiedzę informatyczną), albo zastosowania zewnętrznych rozwiązań tunelujących ruch (co z kolei wymaga instalacji i poprawnej konfiguracji programów przez tego samego użytkownika). Jednak jako że problemy te wynikają z konfiguracji urządzeń w sieci użytkownika, czegoś nad czym nie mamy żadnej kontroli, uznaliśmy że zawarcie informacji o tym ograniczeniu wraz z instrukcjami poprawnej konfiguracji stanowi dostateczne rozwiązanie napotkanego problemu.