AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ GEOLOGII, GEOFIZYKI I OCHRONY ŚRODOWISKA

KATEDRA GEOINFORMATYKI I INFORMATYKI STOSOWANEJ

# Projekt dyplomowy

## Aplikacja do zarządzania zbiorami danych

## Dataset management application

Autor: Monika Hertel

Kierunek studiów: Inżynieria i Analiza Danych

Opiekun pracy: dr Paweł Oleksik

Kraków, 2024

# Spis treści

# Wstęp

W

W kontekście poniższej pracy, zbiorem danych jest określany jako plik zawierający w sobie wszelkie informacje, tj. artykuły naukowe, dane tabelaryczne, grafiki czy notatki. Każdy z tych przykładów przechowuje wiedzę możliwą do interpretacji przez człowieka. Do przeprowadzenia analizy danych, często potrzebna jest znajomość teorii kryjącej się za wartościami w zestawie.

## Cel i zakres projektu

Projekt ma na celu przedstawienie możliwości systemu ułatwiającego użytkownikowi proces zbierania i segregowania informacji. Poniższa praca skupi się na ogólnym koncepcie aplikacji oraz szczegółowym opisie implementacji modułu tworzącego streszczenia.

# 1 Zagadnienia teoretyczne

## 1.1 Problem tworzenia streszczeń

Streszczenie w każdej pracy naukowej jest jej ważną częścią. Osoby poszukujące informacji na dany temat nie są w stanie przetworzyć ilości dostępnych informacji. Ma ono na celu przekazać kluczowe informacje o czytanym dokumencie, aby czytelnik mógł ocenić czy dany artykuł zawiera informacje mu przydatne.

W kontekście uczenia maszynowego kondensowanie treści, w celu stworzenia kontekstu, opiera się o obliczanie poziomu istotności dla każdego zdania. [1]. Automatyzacja tego procesu dąży do ułatwienia autorom tworzenia prac, poprzez skrócenie czasu, którego wymaga kreacja abstraktu. Systemy ATS (ang. *Automatic Text Summarization*) są jednym z cięższych wyzwań sztucznej inteligencji, dotyczących przetwarzania języka naturalnego [2]. Podejścia do tworzenia tych systemów, możemy podzielić na ekstraktywane, abstrakcyjne i hybrydowe.

### 1.1.1 Metody ekstraktywne a abstrakcyjne

Większość badań nad systemami ATS skupia się na użyciu metod ekstraktywnych, starając się przy tym uzyskać zwięzłe i kompletne streszczenia. Podejście ekstraktywne polega na wybraniu najważniejszych zdań z całego dokumentu, a długość uzyskanego wyniku zależy od wartości stopnia kompresji [3].

Streszczenia stworzone z użyciem metod abstraktywnych zawierają zdania, które nie znajdują się w orginalnym tekście. System musi w pewnym stopniu "rozumieć" tekst, aby móc poprawnie zinterpretować dokument. Abstraktywność wymaga implementacji bardziej złożonych algorytmów przetwarzania języka naturalnego.

Istnieje również podział zadania podsumowania na nienadzorowane i nadzorowanie. Nienadzorowane tworzą streszczenia tylko na podstawie danych wejściowych, czyli jedynie zawartości wprowadanego dokumentu. Do optymalnego wyboru zdań, takie systemy implementują metody natury heurystycznej. Z tego powodu są one odpowiednie do przetwarzania danych na bierząco.

Sposób nadzorowany wymaga przeprowadzenia fazy trenowania modelu, która wymaga wprowadzenia opisanego zbioru treningowego. Takie zbiory powinny posiadać docelowe postacie streszczeń otrzymane z pełnego tekstu dokumentu. Taki

proces jest trudny do przeprowadzenia na większej ilości tekstów.

### 1.1.2 Ocena poprawności abstraktu

Zazwyczaj ingerencja człowieka jest wymagana przy ewaluacji wytworzonego streszczenia. Treść jest sprawdzana pod kątem poprawności gramatycznej, składni czy całościowej spójności. Taka ewaluacja wymaga dużego nakładu pracy, a przy większych projektach jest praktycznie niemożliwe. Dlatego możliwość zautomatyzowania tego procesu jest wręcz wymagana.

Jednymi z pierwszych metod ewaluacji automatycznej były metryki takie jak podobieństwo cosinusowe (*ang. cosine similarity*), *unit overlap* czy miara najdłuższego wspólnego podzdania (*ang. longest common subsequence*). Te elementy zostały później skondensowane w zbiór kryteriów opisanych akronimem *ROUGE* (*ang. Recall-Oriented Understudy of Gisting Evaluation*) [4]. Poniższy segment przedstawia poszczególne komponenty zestawu *ROUGE*, to jest kryteria *ROUGE-N*, *ROUGE-L* oraz *ROUGE-S*.

*ROUGE-N* mierzy poziom pokrycia n-gramów, czyli ciągłych sekwencji $n$ słów, pomiędzy wytworzonym tekstem a tekstem źródłowym. Najczęstszym przypadkiem użycia tego kryterium jest ewaluacja poprawności gramatycznej. Miary *ROUGE-1* oraz *ROUGE-2* należą do miar *ROUGE-N* i w kolejności kożystają z unigramów i bigramów. Tymczasem *ROUGE-L* jest wyznaczana na podstawie porównania najdłuższych wspólnych sekwencji słów w zdaniach, występujących w streszczeniu wygenerowanym i wzorcowym. *ROUGE-S* działa na tej samej zasadzie co *ROUGE-2* lecz uwzględnia w swoim działaniu struktury skip-bigram, który jest rozszerzeniem definicji bigramów o możliwość zawierania w sobie maksymalnie jednego przedimka.

## 1.2 Ekstrakcja tekstu plików PDF

Pliki PDF są powszechnie przyjętym standardem przechowywania informacji. Format PDF jest oparty o strukturę binarnego formatu plików, zoptymalizowanego pod kątem wysokiej wydajności odczytu wizualnego. Zawierają w sobie informacje o strukturze dokumentu, takie jak zawartość tekstowa, grafiki czy użyta czcionka. Są one zoptymalizowane pod drukowanie. Niezaszyfrowane PDF mogą być w całości

reprezentowane z użyciem wartości bitowych, odpowiadających części zbioru znaków zdefiniowanego w *ANSI X3.4-1986*, symbole kontrolne oraz puste znaki. Wizualnie jednak nie są one ograniczone do zbioru znaków ASCII.

Format PDF separuje informacje dotyczące samego znaku a jego wizualną reprezentacją. Jest to rozróżnienie na znak pisarski i glif, gdzie grafem jest jednostką tekstu a glif, jednostką graficzną. Glif informuje o położeniu znaków na stronie dokumentu, jego czcionce i innych elementach wyglądu.

Otrzymanie zawartości pliku może być wykonane za pomocą wydobycia elementów PDF z strumienia pliku lub z użyciem analizy obrazów, na przykład technologi optycznego rozpoznawania znaków OCR (*ang. Optical Character Recogintion*). Podstawowym zadaniem systemu OCR jest konwersja dokumentów w dane możliwe do edytowania czy wyszukiwania. Techniki oparte o analizę obrazów są bezpośrednio zależne od jakości wprowadzonego elementu. Idealną sytuacją dla wykożystania metod OCR jest kiedy posiadany plik zawiera w sobie jedynie tekst i jest obrazem binarnym [5]. Dodatkowym atutem takich systemów jest możliwość wykrycia pisma i konwersja na tekst.

Ekstrahowanie danych z strumienia pliku, wiąże się z kilkoma problemami. Biorąc pod uwagę możliwość że plik pdf może posiadać różne kodowanie takie jak *UTF-8*, *ASCII* czy *Unicode*, możemy doświadczyć utraty informacji spowodowanej schematem kodowania pliku. Automatyczna ekstrakcja zawartości polega na selekcji znaków znajdujących się pomiędzy zdefiniowanymi słowami kluczowymi. Pliki PDF są przystosowane do drukowania, z tego powodu reprezentacja tekstu w strumieniu może się znacząco różnić od tej na stronie. Ponieważ pozycje znaków na poszczególnych stronach są absolutne, przedstawienie w strumieniu bierze pod uwagę spacje między elementami.

Niezależne od sposobu pobierania informacji, nie jest możliwe zagwarantowanie ich poprawności względem dokumentu wejściowego. Brak formalnej definicji struktury, przynajmniej jeżeli chodzi o artykuły naukowe, uniemożliwia stworzenie uniwersalnego algorytmu ekstrakcji.

# 2 Implementacja

## 2.1 Projekt systemu

Aplikacja została zaprojektowana z myślą o użytkownikach będących w posiadaniu dużej ilości plików z danymi. Głównym celem jest automatyzacja procesu segregowania i przetwarzania informacji. System etykietuje otrzymane pliki poprzez ekstrakcje słów kluczowych i daje możliwość wyszukiwania plików w bardziej optymalny sposób. Otrzymane dokumenty są kategoryzowane ze względu na słowa klucze. Po otrzymaniu elementów o podobnej strukturze i zawartości, są one łączone ze sobą. Informacja o takim połączeniu jest umieszczana w bazie danych, tak aby przy odczycie jednego z elementów, drugi był sugerowany jako następny plik do wyświetlenia.

## 2.2 Architektura systemu

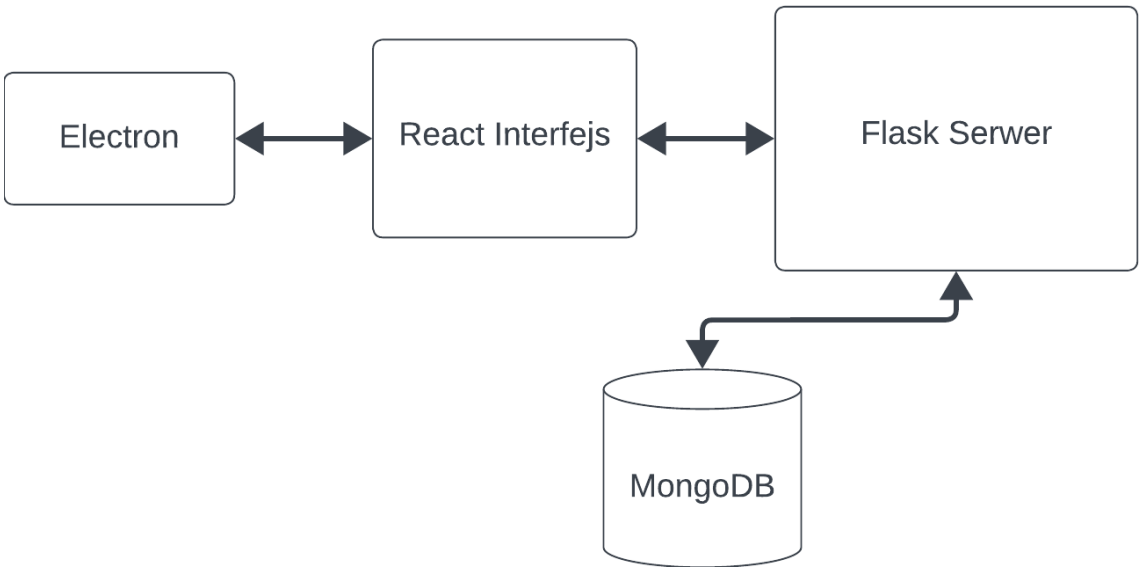
Aplikacja została zbudowana z myślą o zapewnieniu interfejsu użytkownika przy użyciu JavaScript React, serwera HTTP opartego na Flask Python, oraz bazy danych MongoDB, która jest bazą typu NoSQL. Zintegrowana całość jest uruchamiana z wykorzystaniem pakietu Electron, co umożliwia stworzenie aplikacji natywnej, zachowując przy tym funkcje przeglądarki oraz pozwalając na dostęp do zasobów systemowych.

Zakres implementacji obejmuje stworzenie serwera działajacego w oparciu o protokół HTTP. Serwer zwraca odpowiedz w postaci pliku JSON, który następnie jest przetwarzany przez stronę React, czego wynik jest wyświetlany klientowi.

System składa się z 4 głównych komponentów:

1. serwer HTTP - zbudowany z pomocą języka Python oraz biblioteki Flask,
2. baza danych NoSQL
3. interfejs użytkownika - generowany z użyciem frameworku Javascript React,
4. konwerter na aplikację natywną Electron.

Poniższe sekcje przedstawiają opis poszczególnych elementów służących do implementacji systemu.




Rysunek 1: Diagram łączenia komponentów

### 2.2.1 Javascript React

Jest to biblioteka pozwalająca na budowanie interaktywnych interfejsów użytkownika. Główną koncepcją React są komponenty, czyli samodzielne, hermetyczne jednostku interfejsu. Pozwala to na reakcje na zmiany wywołane przez użytkownika lub sam system i przy tym automatyczne aktualizowanie tych stron. React używa składni JSX (*Javascript XML*), który jest rozszerzeniem składni Javascript. Integruje ona kod Javascript z deklaratywnym opisem struktury interfejsu.

React posiada specyficzne dla siebie struktury jakimi są "*hooks*". Są to funkcje pozwalające na korzystanie z React, bez potrzeby tworzenia klas. Posiada on kilka wbudowanych funkcji tego rodzaju. Jedną z nich jest funkcja "*useState()*, która deklaruje zmienną stanu z wnętrza komponentu funkcyjnego. Jest to sposób na przechowanie wartości między wywołaniami funkcji.

Komunikacja z serwerem aplikacji odbywa się odbywa się z użyciem klienta HTTP *Axios*, który jest mechanizmem opartym o obietnice (*ang. promise*), co pozwala na asynchroniczną relację z punktem końcowym (*ang. endpoint*). W systemie, po otrzymaniu odpowiedzi od serwera, ustawiany jest stan, czego przykład wykorzystania jest przedstawiony poniżej.

```
const [Message, setMessage] = useState();
const handleUpload = () => {
  const fileInput = document.getElementById("fileInput");
  const method = document.getElementById("method").value;
  const filePath = { path: fileInput.files[0]?.path, method: method };
  if (filePath.path) {
    axios
      .post(`${SERVER_URL}/upload`, filePath, {
        headers: { "Content-Type": "application/json" },
      })
      .then((response) => {
        setMessage(response.data.message);
      })
      .catch((err) => {
        console.error(err);
      });
  }
};
```

Rysunek 2: Przykład wykorzystania *Axios*; Implementacja dodania pliku

### 2.2.2 Python Flask

Flask jest to elastyczny framework, pozwalający na tworzenie aplikacji internetowych przy minimalnej ilości kodu. Dostarcza on jedynie niezbędne narzędzia, stawiając tym na dużą swobodę implementacji innych funkcji. Funkcja serwera jest uruchamiana w sytuacji gdy na podany adres URL zostanie wysłany komunikat. Informacja o aktywującym adresie jest przekazywana z użyciem dekoratora *.route()*. Rysunek X przedstawia przykład takiego wywołania.

Jedną z możliwość tej biblioteki jest generowanie plików strony jako wynik końcowy działania funkcji. Kożystamy w takiej sytuacji z metody *render_template()*, która otrzymuje na wejściu uprzednio utworzony szablon HTML.

Implementacja serwera z pomocą pakietu Flask umożliwia dostęp do szerokiego spektrum bibliotek języka Python, dotyczących sztucznej inteligencji i przetwarzania danych.

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

Rysunek 3: Przykładowa implementacja funkcji serwera

### 2.2.3 Electron

Biblioteka Electron jest projektem umożliwiającym tworzenie aplikacji natywnych używając technologii web. Standard tworzenia aplikacji z pomocą tej biblioteki jest oparty o komunikację międzyprocesową (*ang. IPC, Inter-process communication*), która jest wynikiem implementacji izolacji kontekstu. Izolacja kontekstu odpowiada za separację procesu renderowania strony od głównego procesu aplikacji w środowisku Node.js. Zwiększa to bezpieczeństwo aplikacji, negując możliwość dostępu do użytych API (*ang. Application Programming Interface*) od strony klienta.

System posiada jeden główny proces *"main"*, którego celem jest tworzenie aplikacji i kierowanie podprocesami, oraz wiele procesów odpowiedzialnych za renderowanie zawartości strony zwanych procesami *"renderer"*. Biblioteka umożliwia również dodanie skryptów, które działają przed renderowaniem strony (ang. *preload scripts*). Otrzymują one dodatkowe możliwości w porównaniu plików renderujących, poprzez posiadanie uprawnień do korzystania z Node.js APIs. Mogą one bezpiecznie udostępnić części API dla procesów strony.

## 2.3 Obsługiwane rodzaje plików

System jest przystosowany do obsługi plików o rozszerzeniach takich jak *PDF*, *CSV* i *TXT*. Każdy rodzaj pliku ma osobne funkcjonalności opisane poniżej. Do plików przypisujemy tag opisujący rodzaj pliku, lecz jest on odrębną jednostką od słów kluczowych generowanych na podstawie zawartości pliku.

### Pliki PDF

Proces obsługi plików PDF jest zależny od wielu czynników. Najprostrzymi plikami do analizy są artykuły posiadające zakładki (*ang. outlines*) o ujednoliconej strukturze, zawierające jedynie tekst. Słowo *outlines* jest zwrotem specyficznym dla plików PDF i są potrzebne do ustalenia tytułu danego pliku. Wynika to z faktu, żę nie zawsze nazwa pliku koresponduje tytułowi zawartości.

System wykorzystuje potokowanie zawartości w celu ekstrakcji tekstu, obrazów oraz tabel. Oznacza to, że aplikacja nie "widzi" dokumentu, lecz pobiera informacje z kodu źródłowego pliku. Działanie aplikacji jest oparte o sam tekst. Na

potrzeby aplikacji, takie fragmenty jak tabele czy grafiki są jedynie przeszkodą ,ponieważ nie powinne one wpływać na wynik algorytmu streszczania. Część elementów tabelarycznych jest niepoprawnie podpisywana jako tekst, co może przyczynić się do generacji niezrozumiałych streszczeń.

#### Pliki tabelaryczne

Podstawowym zachowaniem aplikacji w obliczu plików zawierających dane tabelaryczne jest generacja opisu zgodnie z następującymi krokami. Dla każdej kolumny pobierana jest jej nazwa oraz rodzaj danych zawartych w niej. Ze względu na specyfikę pliku, generacja słów kluczy może przebiec na dwa sposoby: tagami zostają nazwy kolumn lub użytkownik samodzielnie je dodaje.

Oba podejścia mają swoje wady. Nazwy kolumn nie są wymagane podczas kreacji tabel, więc tworzenie tagów w takiej sytuacji nie jest optymalne. Drugie podejście wymaga od użytkownika większego wkładu w ten proces. Potrzebna do tego jest pewna znajomość zestawu danych, który chcemy wprowadzić do systemu aby odpowiednio przypisać słowa klucze do plików.

#### Pliki tekstowe TXT

Zawartość plików tekstowych nie jest uwarunkowana żadnymi normami, co czyni obsługę tych plików niemożliwą do standaryzacji. Podczas dodawania do aplikacji, wymagają one od użytkownika określenia rodzaju zawartości. Opcjami, które użytkownik ma do wyboru są: tekst lub dane tabelaryczne.

W pierwszym przypadku program sprawdza długość tekstu i na tej podstawie decyduje o następnych krokach. Domyślną długością graniczną jest 200 słów, lecz może ona być ustawiona manualnie przez klienta. Po przekroczeniu tego progu, system procesuje plik w sposób podobny do obsługi plików *PDF*.

## 2.4 Schematy poszczególnych funkcjonalności

### 2.4.1 Dodawanie plików PDF

Zachowanie systemu przy dodaniu plików tekstowych.

1. Gdy użytkownik wybierze plik, aplikacja przesyła ścieżkę pliku do serwera,

2. System ekstraktuje z pliku całą zawartość z pomocą biblioteki języka Python *PdfMiner*,

3. Z zawartości zostają wyciągnięte słowa klucze. Tworzenie tzw. tagów odbywa się z pomocą biblioteki *yake* oraz metody *KeywordExtractor()*. Przyjmuje ona następujące parametry:

   - *lan* - język danego tekstu,
   - *n* - maksymalna ilość słów w jednym tagu,
   - *deduplim* - szansa na powtóżenie się słów w różnych słowach kluczach,
   - *top* - ilość elementów wyjściowych

```
lang = "en"
max_ngram_size = 3
deduplication_treshold = 0.5
num_keywords = 6
kw_extract = yake.KeywordExtractor(lan=lang,n=max_ngram_size,
                                   dedupLim=deduplication_treshold,
                                   top=num_keywords, features=None)

keywords = kw_extract.extract_keywords(res)
tags = [kw[0].lower() for kw in keywords]
```

Rysunek 4: Przykład użycia *KeywordExtractor()*

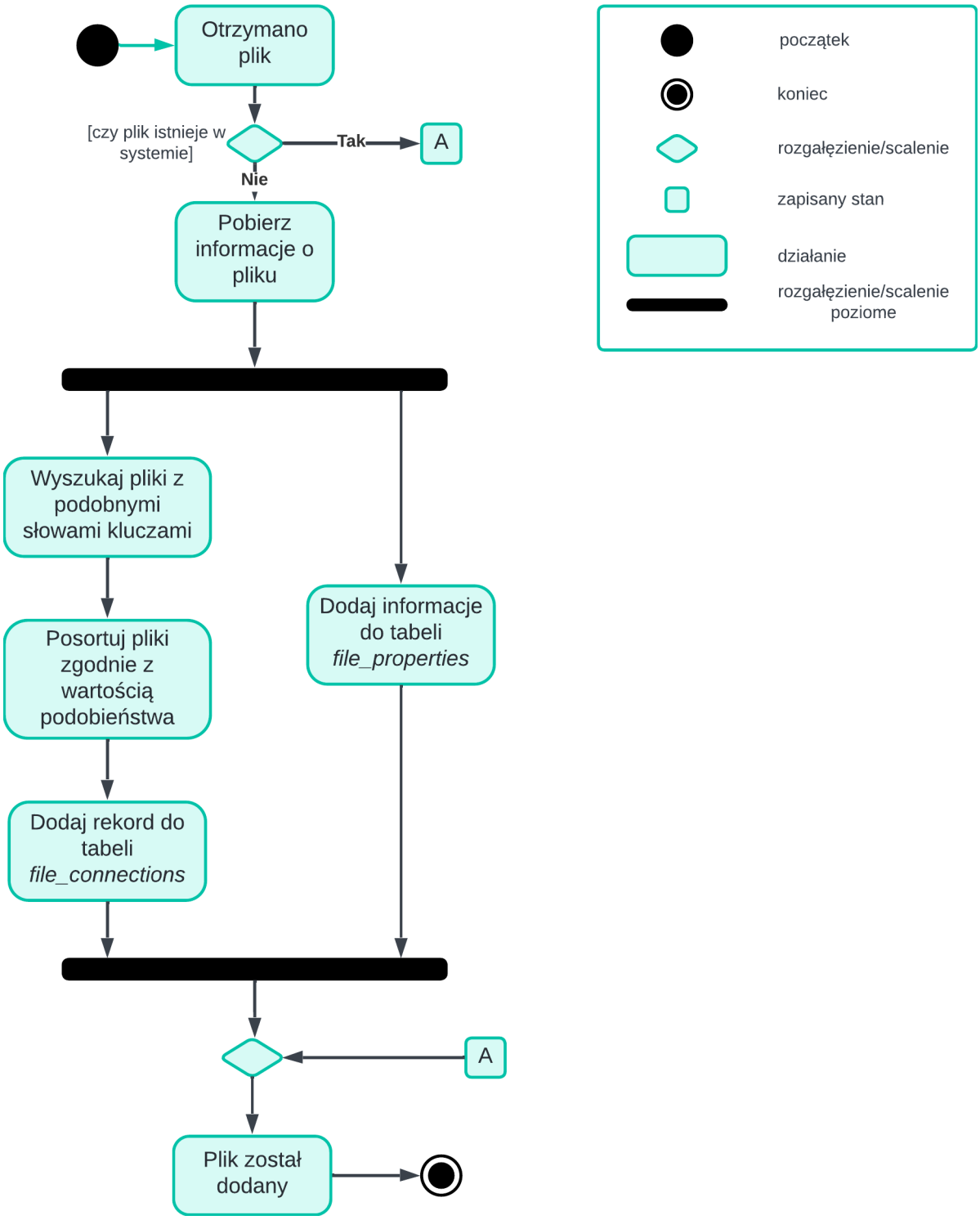
4. w tym samym czasie z pomocą biblioteki *sumy* oraz funkcji *TextRankSummarizer*, jest tworzone streszczenie z użyciem algorytmu *TextRank*,

5. jeżeli plik pdf posiada zakładki (ang. *outlines*) to metoda *.get_outlines()* ekstraktuje je i z wyniku możemy otrzymać tytuł dokumentu i nadać plikowi taką nazwę. a jeżeli ich nie ma to tytułem pliku zostaje bazowa nazwa pliku,

6. ostatecznie wszystkie informacje są zbierane i przesyłane do bazy danych.

Model TextRank zastosowany w projekcie, jest algorytmem rankingowym opartym na grafach [6]. Operuje on na zasadach "głosowania" i "rekomendacji". Jest on używany do tworzenia streszczeń metodą ekstraktywną i nienadzorowaną. Dla przedstawionego systemu, nie jest możliwe wytworzenie odpowiedniego zbioru treningowego, aby móc zastosować metody nadzorowane.

Poniższy diagram przedstawia proces dodawania pliku PDF wraz z uwzględnieniem implementacji asocjacji plików o podobnej tematyce, opisanej w punkcie 2.4.3.

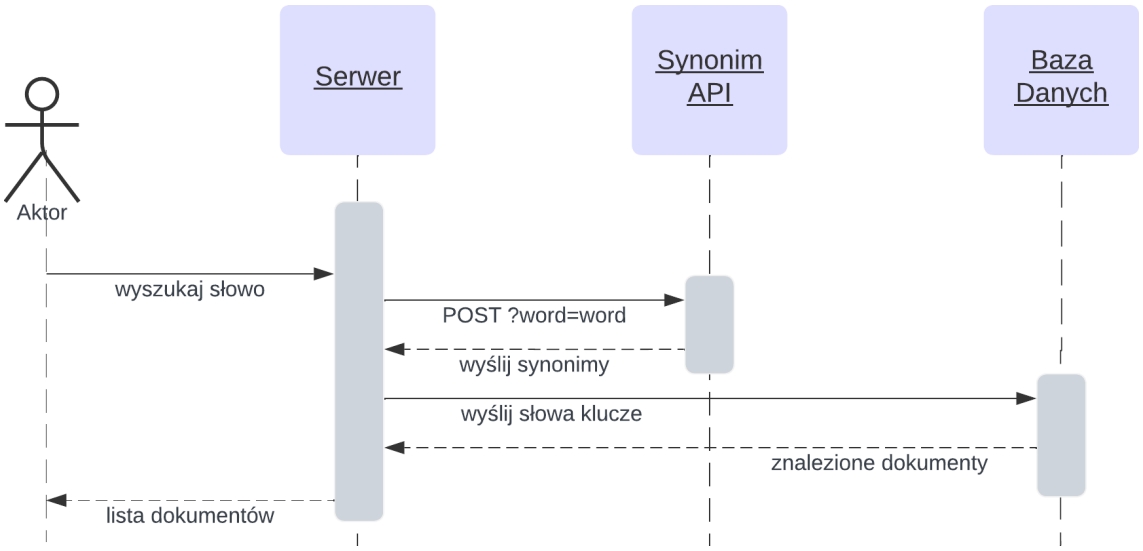



Rysunek 5: Diagram aktywności przesyłania pliku PDF

### 2.4.2 Wyszukiwanie i wyświetlanie plików

Użytkownik ma możliwość wysukiwania plików po słowach kluczach lub tytule pliku. Tutaj przydatne jest użycie API generującego synonimy dla wyszukiwanego słowa. System zachowuje się w poniżej opisany sposób.

1. aplikacja przesyła komunikat z wyszukiwanym słowem,
2. serwer używając API synonimów pobiera 5 najbliższych słów do słowa szukanego,
3. serwer przesyła osobne komunikaty do tabeli *file_properties* w bazie danych, zawierające osobno słowo klucz oraz synonimy
4. baza zwraca informacje dotyczące znalezionych dokumentów oraz słowo klucz,
5. serwer wysyła użytkownikowi posortowaną listę plików.




Rysunek 6: Schemat sekwencyjny wyszukiwania pliku

```
@app.route("/search_view", methods=["GET", "POST"])
def search():
    word = request.json["word"].lower()
    querystring = {"entry":word}
    response = requests.get(url, headers=headers, params=querystring)

    syn = response.json()["associations_array"][:5]
    syn = [word] + syn
    temp = {}
    for i in syn:
        for x in table_in.find({"keywords":i}, {"_id":0, "main title": 1, "keywords":1,"summary":1}):
            temp = {**temp,**{i:x}}
    res = {"word":word,"all":temp}

    return json.dumps(res,indent=4)
```

Rysunek 7: Implementacja wyszukiwania pliku

### 2.4.3 Asocjacja plików o podobnej tematyce

Poniższa sekcja przedstawia proces łączenia dokumentów na podstawie wygenerowanych słów kluczowych. System z każdym dodanym plikiem przeszukuję przestrzeń dokumentów i aktualizuje już utworzone rekordy, w sytuacji dopasowania. Schemat przedstawia tworzenie połączeń dla nowo dodanego pliku.

1. dla każdego słowa kluczowego, jest pobrana lista ID plików z tabeli *file_properties*, pod warunkiem dopasowania przynajmniej jednego tagu,

2. następnie obliczany jest współczynnik podobieństwa, którego schemat postępowania jest przedstawiony poniżej:

   (a) wyznaczana jest wartość dla poszczególnych słów kluczy

   - jeżeli wysłane słowo kluczowe zgadza się fragmentarycznie z otrzymanym tagiem (np. “zaburzenia” a “zaburzenia odżywiania”), to zapisywana jest wartość $\frac{ilosc(dopasowan)}{ilosc\_slow(wejscie)+ilosc\_slow(pobrane)}$,
   - jeżeli słowa są identyczne, zapisywana jest wartość 1,

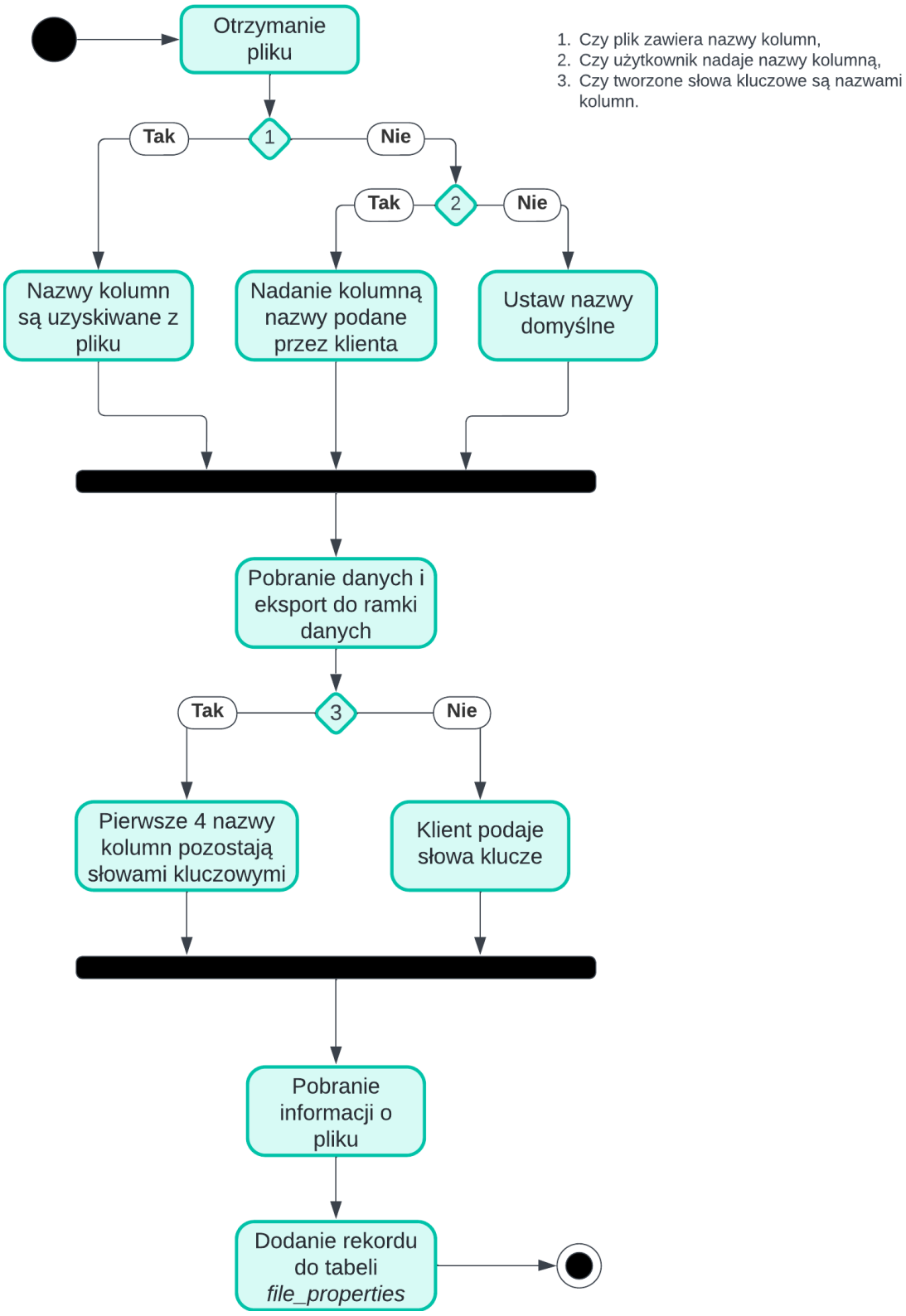
   (b) wartości dla wszystkich słów kluczy są sumowane i dzielone przez liczbę tagów.

3. do ID pliku przypisana jest wartość podobieństwa, i po posortowaniu od największej wartości, ID są dodawane do jednego rekordu w tabeli *file_connections*

   - jeżeli wartość podobieństwa nie przekracza 0.5, to plik nie jest brany pod uwagę

### 2.4.4 Obsługa plików zawierających dane tabelaryczne

1. użytkownik przy dodaniu pliku podaje następujące informacje:
    - czy plik zawiera informacje o nazwach kolumn, jeżeli nie to czy chce on nadać te nazwy,
    - czy słowa kluczowe będą nadane manualnie lub czy powinny być pobrane z nazw kolumn,
2. odpowiednio wczytywana na server do postaci ramki danych (*ang. dataframe*) modułu *pandas*,
3. sprawdzany i zapisywany jest typ wartości w każdej kolumnie oraz obliczony procent ilości wartości brakujących,
4. dodanie rekordu do bazy danych.




Rysunek 8: Diagram aktywności przetwarzania plików tabelarycznych

# 3 Możliwości rozwoju i wykorzystania aplikacji

Wspomniane funkcjonalności to jedynie początek istnienia tego systemu. Stworzenie dodatkowych zdolności programu jest uproszczone, poprzez oparcie systemu o

odpowiedzi w postaci komunikatu http zamiast stosowania szablonów, jak to jest zwykle przy standardowym projektowaniu z użyciem Python Flask. Baza danych wykorzystana w projekcie, MongoDB, jest bazą typu NoSQL, która umożliwia przechowywanie informacji w nie ustrukturyzowany sposób. Dzięki czemu tabele można wypełnić dowolnymi wartościami a sama aplikacja wyświetla wyszystkie elementy tabeli dla wyszukiwanego obiektu.

## 3.1 Ekspansja działań dotyczących danych tabelarycznych

Aktualny system pozwala na implementacje metod wstępnego przetwarzania danych (*ang. preprocessing*) z wnętrza aplikacji. Preprocessing oznacza zbiór działań miejących na celu obróbkę i przygotowanie danych do dalszej manipulacji. Do takich działań należą usunięcie wartości brakujących, filtracja czy usuniecie wartości odstajacych.

Celem stworzenia systemu jest automatyzacja procesów związanych z przechowywaniem informacji. możliwość pd.concat

## 3.2 Implementacja abstrakcyjnego tworzenia streszczeń

Abstrakcyjna generacja streszczeń jest procesem wymagającym posiadania zestawu treningowego złożonych z streszczeń i przypisanym im tekstów początkowych. Jest to proces czasochłonny, lecz jakościowo przewyższa streszczenia metodami ekstraktywnymi.

Samodzielna kreacja takiego modelu nie jest optymalna pod względem ilości zasobów potrzebnych do modelowania a zasobów pamięciowych przeznaczonych na działanie całej aplikacji. Istnieje jednak rozwiązanie tego problemu za pomocą wykożystania *API*.

## 3.3 Identyfikacja obrazów

Identyfikacja obrazów mogą przysłużyć się sukcesowi systemu. Dla człowieka proces identyfikacji elementów obrazu dzieje się podświadomie. Program jest w stanie rozpoznać i przypisać grafikę do odpowiedniej kategorii, jedynie w sytuacji gdy jest on zaprogramowany do wykrywania ich. Z punktu widzenia maszyny, obraz jest ustruk-

turyzowaną macierzą liczb, której każdy element zawiera informacje o nasyceniu i kolorze piksela. Aby stworzyć system odpowiedzialny za kategoryzację obrazów, należy przejść przez następujące kroki.

1. Zebranie danych,
2. przygotowanie danych,
3. wybór, trenowanie i testowanie modelu,
4. wprowadzenie modelu do systemu.

Pierwszym i najbardziej kluczowym krokiem przy tworzeniu systemu jest zebranie i opisanie obszernego zestawu grafik, przedstawiających różnorodne obiekty lub sceny rodzajowe. Tak aby nauczyć model rozpoznawania obrazów, skupiając się na podobieństwu, a nie idealnym dopasowaniu.

Celem drugiego etapu jest przekształcenie zebranych danych do odpowiedniej postaci. Proces ten może obejmować działania takie jak normalizowanie wartości pikseli, ujednolicenie rozmiaru obrazów wejściowych czy filtracja. Dopiero po takich modyfikacjach, zbiór jest gotowy do trenowania modelu.

Wybór modelu zależy od złożoności danego problemu oraz wielkość przeznaczonych zasobów Komputery kategoryzują obrazy, porównując układ pikseli wejściowego obrazu ze schematami zapisanymi w systemie. Schematy nie muszą być takie same aby obraz został poprawnie zaklasyfikowany, wystarczy aby były podobne. W celu oszczędności zasobów pamięciowych, do implementacji można wykorzystać dostępne powszechnie narzędzia.

## 3.4 Kategoryzowanie plików względem zawartości

Na obecnym etapie implementacji systemu, wszystkie pliki znajdują się w jednym miejscu oraz jedynym sposobem na odnalezienie plików jest skorzystanie z zaprogramowanej wyszukiwarki plików. System łączy pliki ze względu na wspólne słowa klucze, lecz nie ingeruje w proces sytuowania obiektów w katalogu aplikacji. Pliki o różnorodnej tematyce mieszają się ze sobą do tego stopnia, że nie jest możliwe, aby przy niepoprawnie nazwanych plikach, znaleźć te potrzebne.

Odpowiedzią na ten problem jest system kategoryzacji i podziału plików. Segregacja powinna odbywać się na podstawie słów kluczy, jednakże dalsze badania są wymagane do określenia wymagań warunku podziału. Niepoprawny warunek może spowodować duplikację plików na szeroką skalę, co negowałoby cel tej implementacji.

# Podsumowanie i wnioski

Automatyzacja procesów związanych z

# Spis rysunków



# Literatura

[1] B. Mutlu, E. A. Sezer, and M. A. Akcayol, "Candidate sentence selection for extractive text summarization," *Information Processing & Management*, vol. 57, no. 6, p. 102359, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306457320308542

[2] W. S. El-Kassas, C. R. Salama, A. A. Rafea, and H. K. Mohamed, "Automatic text summarization: A comprehensive survey," *Expert Systems with Applications*, vol. 165, p. 113679, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417420305030

[3] M. Gambhir and V. Gupta, "Recent automatic text summarization techniques: a survey," *Artificial Intelligence Review*, vol. 47, no. 1, pp. 1–66, Jan 2017. [Online]. Available: https://doi.org/10.1007/s10462-016-9475-9

[4] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: https://aclanthology.org/W04-1013

[5] R. Mithe, S. Indalkar, and N. Divekar, "Optical character recognition," *International journal of recent technology and engineering (IJRTE)*, vol. 2, no. 1, pp. 72–75, 2013.

[6] R. Mihalcea and P. Tarau, "TextRank: Bringing order into text," in *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, D. Lin and D. Wu, Eds. Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 404–411. [Online]. Available: https://aclanthology.org/W04-3252