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

Abstrakt

Celem pracy inżnierskiej jest przedstawienie projektu aplikacji wspomagającej gromadzenie i segregację plików z danymi ze względu na ich zawartość. Zostaną opisane możliwe implementacje dotyczące obsługi zbiorów danych. Praca skupia się na elementach realizacji związanej z plikami tekstowymi, a w szczególności na formacie PDF. System wykorzystuje architekturę webową lecz tworzy własny interfejs graficzny.

Słowa klucze: Przechowywanie danych, streszczenia, aplikacja natywna, analiza tekstu.

Spis treści

Wstęp

Niniejsza praca ma na celu prezentacje projektu aplikacji, która umożliwiłaby użytkownikowi gromadzenie i zarządzanie zbiorami danych. Przedstawia elementy implementacji i potencjalne rozszerzenia aplikacji.

Aplikacja ułatwiłaby użytkownikom importowanie i przechowywanie plików zawierających informacje. Pliki te, określane zbiorczo jako zbiór danych, obejmują szeroki zakres typów treści, między innymi artykuły naukowe, dane tabelaryczne czy pliki tekstowe. System, po rozpoznaniu formatu danych, przetwarzał i analizowałby ich zawartość, aby wspomóc proces klasyfikacji. Został zaprojektowany jako proste rozwiązanie programowe do indywidualnego użytku z możliwością wyposażenia w rozszerzenia, które wykorzystują techniki sztucznej inteligencji (AI).

Praca zaprezentuje kluczowe komponenty aplikacji, takie jak moduł gromadzenia danych, mechanizm kategoryzacji czy wyszukiwania. Zostanią przedstawione schematy działania tych funkcjonalności.

1 Zagadnienia dotyczące projektu

Zaprojektowany system jest przeznaczony do gromadzenia i klasyfikowania danych dla indywidualnego użytkownika. Zgromadzone dane mogą być przechowywane w różnych formach (pliki w systemie plików, rekordy w bazach danych), jak i w różnych miejscach (lokalne nośniki danych, zasoby w chmurze). Ze względu na przewidywaną skalę tej wersji projektu, zdecydowano się na składowanie danych w nierelacynej bazie danych zarządzanej silnikiem MongoDB. Omówienie różnych aspektów przechowywania danych znajduje się w rozdziale 1.1.

Zamysłem aplikacji jest wspomaganie klasyfikacji gromadzonych danych. Aby było to możliwe należy wydobyć informacje z pliku, a następnie odpowiednio je przetworzyć. Sposób wydobywania właściwych informacji i ich przekształcania zależy od rodzaju danych (dokumenty tekstowe, dane tabelaryczne, grafiki). Szczególnym przypadkiem są pliki PDF, gdzie proces ekstrakcji zawartości jest utrudniony w związku z kodowaniem i formatem pliku, co zostanie omówione w dziale 1.2.

W pracy skupiono się na przedstawieniu elementów przetwarzania plików tekstowych. Dla nich zdecydowano się na wykorzystanie informacji pod kątem kreacji streszczeń zawartości, jak i generacji słów kluczowych. Etykiety służyłyby do kategoryzacji plików względem tematu. W punkcie 1.3 opisano problem tworzenia abstraktów, metody oraz sposoby sprawdzania poprawności.

Projekt zakłada wykorzystanie architektury webowej, która jest specyficzną realizacją architektury typu klient-serwer. Zazwyczaj klientem jest przeglądarka, a komunikacja z serwerem odbywa się poprzez protokół HTTP. W realizacji, dzięki wykorzystaniu pakietu Electron, finalna aplikacja nie wymaga użycia zewnętrznej przeglądarki, dzięki czemu zachowuje się ona jak aplikacja natywna. Opis architektury webowej oraz pakietu Elektron został umieszczony w punkcie 1.4.

1.1 Sposoby przechowywania informacji

Istnieje wiele sposobów magazynowania danych. Wybór odpowiedniego sposobu zależy od indywidualnych potrzeb użytkownika oraz ilości informacji. Na poziomie organizacji dzielą się one na serwery lokalne (*ang. on-premise*) oraz przechowywanie w chmurze (*ang. cloud storage*). Na niższym poziomie architektury, dane mogą być przechowywane z wykorzystaniem baz danych czy plików w systemie plików.

Bazy danych

Bazy danych dzielą się na bazy SQL (*ang. Structured Query Language*) oraz NoSQL (*ang. Not only SQL*). Najważniejszym czynnikiem rozróżniającym te bazy jest model przechowywania danych. Bazy danych SQL charakteryzują się ustrukturyzowanym modelem danych oraz porządkują informacje w tabele z predefiniowanymi schematami. Wykorzystywane są w sytuacjach gdzie wymagane jest zagwarantowanie spójności danych oraz zastosowanie złożonych zapytań czy transakcji, jak np. w systemach finansowych.

Bazy NoSQL oferują elastyczny model przechowywania, który obsługuje różne struktury danych. Obsługują różnorodne języki zapytań lub interfejsy API dostosowane do określonych typów baz danych, zaspokajając indywidualne potrzeby w zakresie zarządzania danymi. Bazy danych powyższego rodzaju wyróżniają się w scenariuszach, w których elastyczność, skalowalność i wydajność są najważniejsze, znajdując zastosowanie w aplikacjach internetowych, analityce danych w czasie rzeczywistym (np. monitorowanie stanu systemu) i środowiskach zajmujących się dużymi ilościami nieustrukturyzowanych lub częściowo ustrukturyzowanych danych [1].

Na potrzeby projektu wykorzystano bazę typu NoSQL, a dokładniej MongoDB. Elementem stojącym za tą decyzją była niezdefiniowana struktura rekordów powstałych w wyniku działań systemu.

MongoDB przechowuje dane w dokumentach, które składnią przypominają JSON, umożliwiając dynamiczny schemat i łatwą integrację z systemami o architekturze klient-serwer [2].

Serwery lokalne a chmura

Gdy rozmiar bazy danych przekracza dedykowany rozmiar pamięci masowej prywatnych urządzeń, takich jak komputery osobiste lub dyski zewnętrzne, lub gdy jej obecność zaczyna ograniczać funkcjonalność poszczególnych systemów komputerowych, należy rozważyć przeniesienie danych na serwer lokalny lub do infrastruktury opartej na chmurze.

Pojęcie **serwerów on-premises** odnosi się do fizycznego sprzętu i infrastruktury, które znajdują się pod kontrolą organizacji lub osoby fizycznej. Podejście to obejmuje konfigurowanie i utrzymywanie serwerów na miejscu w celu przechowywania danych i zarządzania nimi, bez angażowania usługodawców zewnętrznych [3].

Usługi **przechowywania w chmurze**, takie jak Amazon Web Services (AWS) czy Azure Cloud, zapewniają możliwość przechowywania danych i zarządzania nimi w zdalnych centrach danych. Możliwe jest zapewnienie redundancji zasobów poprzez przechowywanie ich w więcej niż jednym miejscu. Chroni to przed utratą danych w wypadku nieprzewidywalnych zdarzeń (awarie, klęski żywiołowe).

Wybór miejsca przechowywania danych zależy od takich czynników, jak wymagania bezpieczeństwa, potrzeby skalowalności czy ograniczenia budżetowe. Często korzystne jest przyjęcie podejścia hybrydowego lub wielochmurowego, wykorzystującego mocne strony różnych opcji przechowywania danych [4].

1.2 Ekstrakcja tekstu plików PDF

Pliki PDF są powszechnie przyjętym standardem przechowywania informacji. Format PDF jest oparty o binarną strukturę pliku, zoptymalizowanego pod kątem wysokiej wydajności odczytu wizualnego oraz wydruku. Zawierają w sobie informacje o strukturze dokumentu, takie jak zawartość tekstowa, grafiki czy użyta czcionka. Format PDF separuje informacje dotyczące samego znaku od jego wizualnej reprezentacji. Jest to rozróżnienie na znak pisarski (grafem) 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.

Niezaszyfrowane PDF mogą być w całości reprezentowane z użyciem znaków ze zbioru zdefiniowanego w *ANSI X3.4-1986*, tj. znaków kodowanych na 7 bi-

tach. Wizualnie jednak nie są one ograniczone do zbioru ASCII [5]. W przypadku liter nie łacińskich czy znaków matematycznych, są one zapisywane w postaci 2-bitowej wartości odpowiadającej znakowi w kodowaniu *Unicode* np. < 0020 >. Glif takiego fragmentu posiada również wartość *toUnicode*, a w przypadku jej braku, informacja o znaczeniu będzie niedostępna.

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 wykorzystania metod OCR jest kiedy posiadany plik zawiera w sobie jedynie tekst i jest obrazem binarnym [6]. Dodatkowym atutem takich systemów jest możliwość wykrycia pisma i jego konwersja na tekst.

Ekstrahowanie danych ze strumienia pliku, wiąże się z kilkoma problemami. Biorąc pod uwagę, że plik PDF może posiadać różne schematy kodowania takie jak *UTF-8*, *ASCII* czy *Unicode*, możemy doświadczyć utraty informacji. 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 koordynaty elementów.

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.

1.3 Problem tworzenia streszczeń

Streszczenie artykułu naukowego jest jego kluczowym elementem. Ilość informacji dostępnych dla każdego, kto szuka wiedzy na dany temat, może być przytłaczająca. Jego celem jest przekazanie najważniejszych elementów treści, aby czytelnik mógł określić, czy informacje są dla niego istotne.

W kontekście uczenia maszynowego, kondensacja treści w celu stworzenia abstraktu, najczęściej opiera się na obliczaniu poziomu istotności dla każdego zdania [7]. Skracając czas potrzebny na napisanie abstraktu, automatyzacja ma na celu ułatwienie autorom pisania artykułów. Systemy ATS (ang. *Automatic Text Summarization*) są jednym z cięższych wyzwań sztucznej inteligencji, dotyczących przetwarzania języka naturalnego [8]. Podejścia do tworzenia tych systemów, możemy podzielić na ekstraktywane, abstrakcyjne i hybrydowe.

1.3.1 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 [9].

Streszczenia stworzone z użyciem metod abstrakcyjnych wymagają głębszej analizy tekstu wejściowego. Generują one podsumowanie poprzez "zrozumienie" głównych pojęć w dokumencie wejściowym. Dzieje się to poprzez implementacje złożonych algorytmów przetwarzania języka naturalnego (*ang. NLP, Natural Language Processing*). Następnie dokonywane jest parafrazowanie tekstu w celu wyrażenia tych pojęć z użyciem słów, które nie należą do oryginalnego tekstu. W praktyce rozwój wspomnianych systemów wymaga kompleksowych zbiorów danych.

Istnieje również podział technik tworzenia streszczeń na nienadzorowane i nadzorowanie. Nienadzorowane tworzą streszczenia tylko na podstawie danych wejściowych, czyli jedynie zawartości wprowadzanego dokumentu. Próbują one odkryć ukrytą strukturę w nieoznakowanych danych. Techniki te są zatem odpowiednie dla wszelkich nowo zaobserwowanych danych nie wymagających modyfikacji.

Sposób nadzorowany wymaga trenowania modelu, jak i 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 [8].

1.3.2 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 automatyzacja 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*) [10]. 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 korzystają z unigramów i bigramów.

- *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.

1.4 Architektura klient-server

Architektura klient-serwer to model projektowania systemów rozproszonych, w którym zadania lub procesy obliczeniowe są podzielone między klientów i serwery. W tej architekturze klientami są zazwyczaj urządzenia użytkowników końcowych (takie jak komputery, smartfony lub tablety), które żądają usług lub zasobów, pod-

czas gdy serwery są scentralizowanymi systemami, które je zapewniają [11]. Architektura webowa, jako typ struktury klient-serwer, jest najczęściej wykorzystywana w wytwarzaniu oprogramowania internetowego z użyciem takich narzędzi jak języki opisowe HTML i CSS, czy języka Javascript. Takie aplikacje działają w przeglądarkach internetowych, lecz dzięki takim szkieletom aplikacji jak Electron, istnieje możliwość konwersji na program z własnym interfejsem graficznym, oddzielonym od przeglądarki. Opis tego projektu znajduje się w dalszej części rozdziału.

Transfer informacji między elementami aplikacji może odbywa się z pomocą protokołu komunikacyjnego jak HTTP/HTTPS (*ang. Hypertext transfer protocol*). Jest to bezstanowy protokół, co oznacza, że każde zapytanie jest przetwarzane niezależne od poprzedniego. Jest on zgodny z modelem zapytanie-odpowiedź (*ang. request-response*), gdzie klient wysyła proźbę a serwer odpowiada. Odpowiedź zawiera w sobie kod statusu, wymagane zasoby lub treść błędu.

1.4.1 Pakiet Electron

Electron jest oparty na Chromium, projekcie stojącym za przeglądarką Google Chrome. Oznacza to, że aplikacje wykorzystują ten sam silnik renderujący i obsługę standardów internetowych, co Chrome. Electron osadza instancję Chromium do renderowania treści internetowych w oknach aplikacji [12].

Standard tworzenia aplikacji z pomocą tej biblioteki jest oparty o komunikację międzyprocesową (*ang. IPC, Inter-process communication*), która jest wynikiem implementacji izolacji wątku.

Izolacja wątku

Separacja wątków odnosi się do możliwości izolacji różnych części aplikacji w celu zapobiegania interferencji oraz zapewnienia stabilności aplikacji. Separacja ta jest osiągana poprzez wykorzystanie wielu procesów, w tym procesu głównego i procesów renderujących.

- Proces główny: zarządza cyklem życia aplikacji i interakcjami z systemem operacyjnym. Działa we własnym, odizolowanym wątku, oddzielnie od procesów renderujących.

- Procesy renderujące: obsługują renderowanie i wyświetlanie treści internetowych w poszczególnych oknach aplikacji. Każdy proces renderujący działa niezależnie, odizolowany od innych procesów renderujących i procesu głównego.

Separacja wątku zapewnia, że zmiany dokonane w jednej części aplikacji nie wpływają na inne części, zwiększając stabilność i bezpieczeństwo. Izolacja umożliwia również efektywne zarządzanie zasobami i skalowalność, ponieważ każdy proces może być zarządzany i optymalizowany niezależnie.

Komunikacja międzyprocesowa (IPC)

Pomimo separacji między procesami, Electron zapewnia mechanizmy komunikacji między nimi poprzez IPC (*ang. Inter-Process Communication*). Pozwala to różnym częściom aplikacji na efektywną wymianę danych, wyzwalanie akcji i synchronizację stanu.

- IPC Main: Electron zapewnia moduł *ipcMain* w głównym procesie, pozwalając mu nasłuchiwać i obsługiwać zdarzenia IPC wysyłane z procesów renderujących.

- IPC Renderer: W procesach renderujących, moduł *ipcRenderer* umożliwia wysyłanie zdarzeń IPC do głównego procesu i odbieranie odpowiedzi.

Dzięki IPC możliwa jest implementacja dwukierunkowej komunikacji między procesem głównym a procesami renderującymi, umożliwiając im koordynację działań, udostępnianie danych i synchronizację stanu w różnych częściach aplikacji.

2 Projekt

Celem pracy było zaprojektowanie i wykonanie programu użytkowego wspomagającego gromadzenie informacji oraz późniejsze przeszukiwanie utworzonych kolekcji. Wśród jego planowanych funkcjonalności należy wymienić:

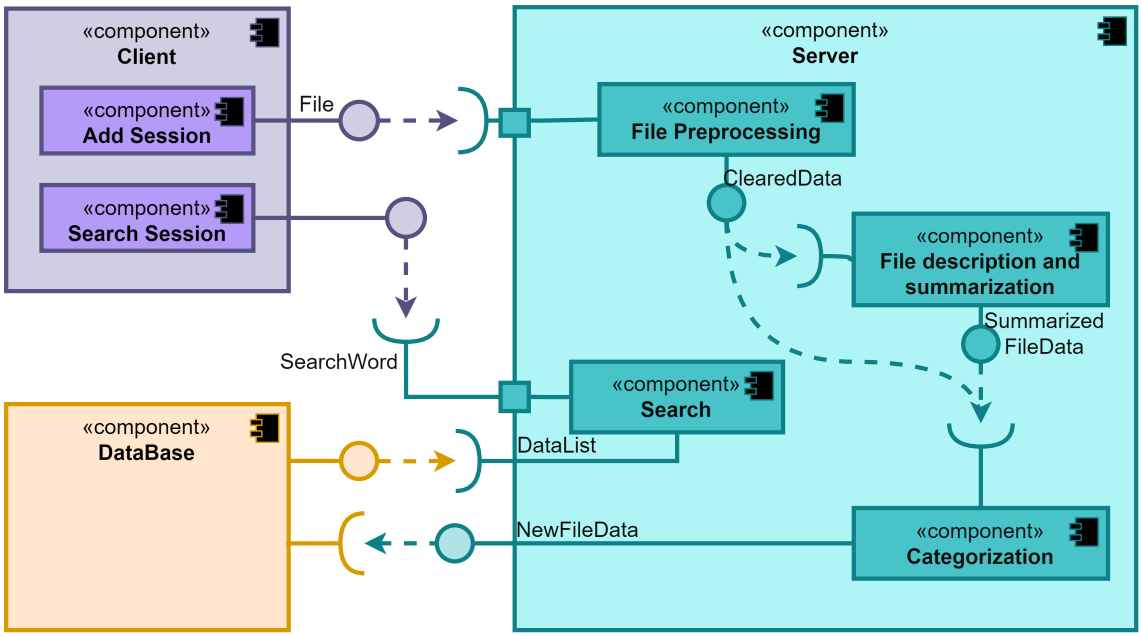
- możliwość gromadzenia danych różnych rodzajów (dokumenty tekstowe i hipertekstowe, dane tabelaryczne, grafiki i inne),
- automatyczną organizację i kategoryzację plików ze względu na zawartość,
- automatyczne generowanie abstraktów z dokumentów tekstowych tak, by zapewnić użytkownikowi szybki wgląd w treść bez konieczności zapoznania się z całą zawartością.

Docelowym odbiorcą produktu ma być indywidualny użytkownik, korzystający z lokalnych zasobów. Dlatego założeniem projektowym było, aby unikać używania usług zewnętrznych, np. chmurowych.

2.1 Opis projektu

Ze względu na rozbudowany charakter narzędzia, zostało ono zaprojektowane jako system współpracujących elementów. Są to:

- moduł przetwarzania plików - najważniejszy moduł, decydujący jakie akcje zostaną podjęte dla danego pliku,
- moduł organizacji danych - segreguje dokumenty oraz tworzy połączenia między podobnymi plikami,
- wyszukiwarka - pobiera informacje z systemu związane z frazą wejściową i przedstawia użytkownikowi,
- moduł podsumowujący - niezależnie od rodzaju pliku tworzy skrótowy opis zawartości.



Rysunek 1: Diagram komponentów projektu.

2.2 Wymagania funkcjonalne modułów

Moduł organizacji danych automatycznie kategoryzuje i organizuje dane użytkownika, aby ułatwić ich efektywne wyszukiwanie i zarządzanie.

- System tagowania: użytkownicy mogą samodzielnie przypisywać etykiety do każdego dokumentu lub pozostawić te wygenerowane przez system, umożliwiając im kategoryzowanie i łączenie plików w oparciu o wspólne motywy lub tematy.

- Struktura folderów: aplikacja zapewnia hierarchiczną strukturę folderów, która pozwala użytkownikom organizować swoje dane w zagnieżdżone kategorie, zapewniając uporządkowany i intuicyjny sposób poruszania się po zebranych informacjach.

- Automatyzacja: wykorzystując zaawansowane algorytmy, aplikacja analizuje zawartość elementów, aby automatycznie kategoryzować je w oparciu o cechy, takie jak słowa kluczowe, tematy lub podobieństwo treści.

Wyszukiwarka umożliwia użytkownikom szybkie i skuteczne znajdowanie określonych informacji w systemie danych.

- Funkcja wyszukiwania pełnotekstowego: użytkownicy mogą przeszukiwać wszyst-

kie elementy danych w aplikacji za pomocą słów kluczowych lub fraz, pobierając odpowiednie wyniki na podstawie pasujących treści.

- Zaawansowane filtry: aplikacja oferuje szereg filtrów wyszukiwania, takich jak zakres dat, typ pliku lub tag, umożliwiając użytkownikom zawężenie wyników wyszukiwania i znalezienie dokładnie tego, czego szukają.

- Sugestie i autouzupełnianie: gdy użytkownicy wpisują swoje zapytania, aplikacja zapewnia sugestie i opcje autouzupełniania, aby przyspieszyć proces wyszukiwania i poprawić dokładność.

Moduł podsumowujący generuje zwięzłe streszczenia dokumentów, zapewniając użytkownikom szybki wgląd w główne punkty artykułów. Realizuje też opisywanie innych rodzajów plików.

- Algorytmy podsumowujące tekst: aplikacja wykorzystuje zaawansowane algorytmy do podsumowywania tekstu, które mogą obejmować metody ekstrakcyjne, które wybierają ważne zdania lub metody abstrakcyjne, które generują nową treść podsumowania.

- Automatyczne podsumowanie: użytkownicy mogą wybrać opcje dla automatycznego generowania podsumowań dla dokumentów w aplikacji, takie jak ilość zdań lub poziom szczegółowości podsumowań.

Przetwarzanie plików wykrywa rodzaj pliku dodanego przez klienta i wyznacza akcje do przeprowadzenia zgodnie z typem.

2.3 Przykładowe schematy funkcjonalności

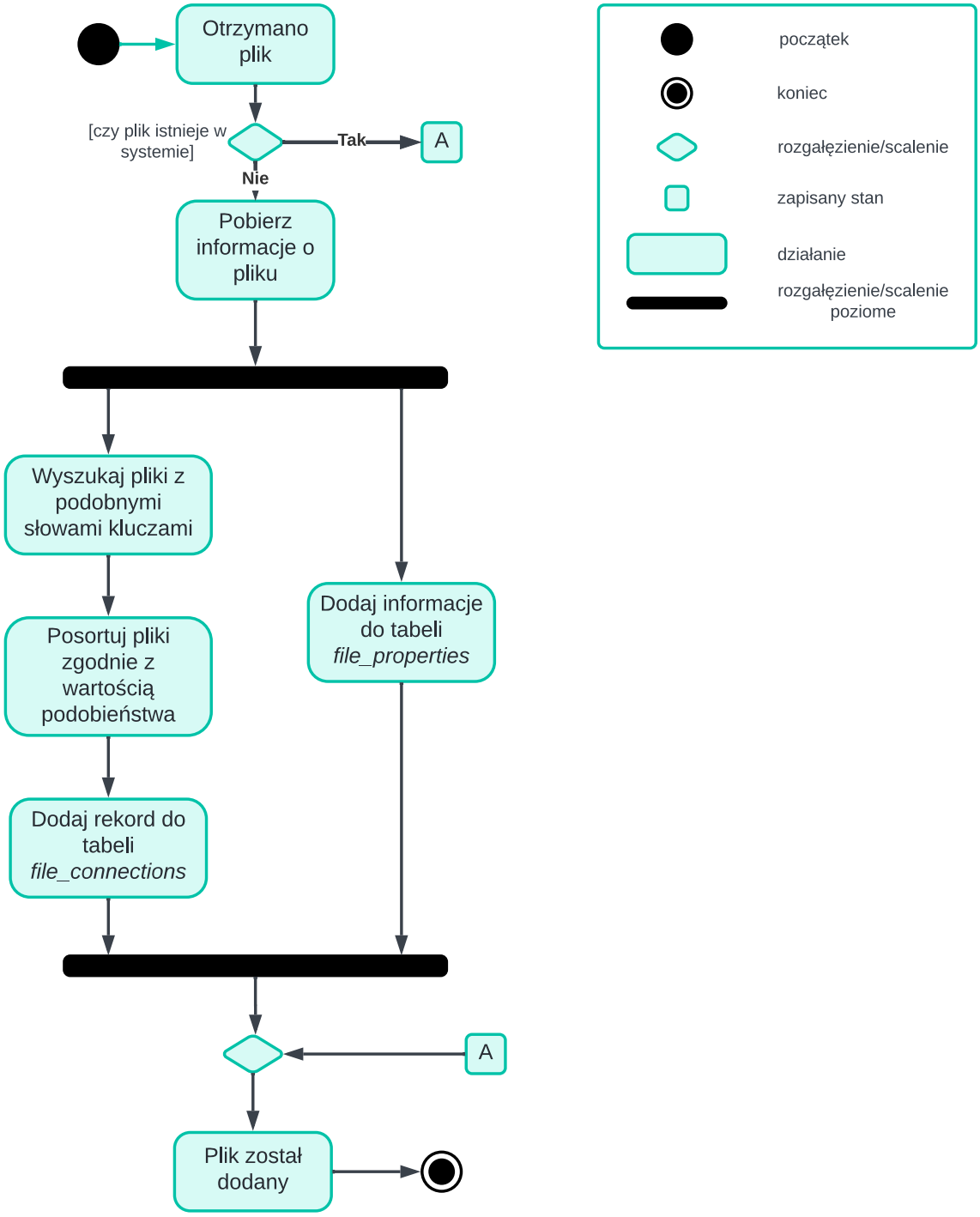
Dodawanie plików PDF

Zachowanie systemu przy dodaniu plików tekstowych.

1. Gdy użytkownik wybierze plik, aplikacja przesyła ścieżkę pliku do serwera.
2. System ekstraktuje z pliku całą zawartość.
3. Z tekstu wytwarzane są słowa klucze.
4. W tym samym czasie jest tworzone streszczenie z użyciem algorytmu *TextRank*.
5. Jeżeli plik PDF posiada zakładki (ang. *outlines*) to możemy otrzymać tytuł dokumentu i nadać plikowi taką nazwę. W przypadku ich braku 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 [13]. 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.

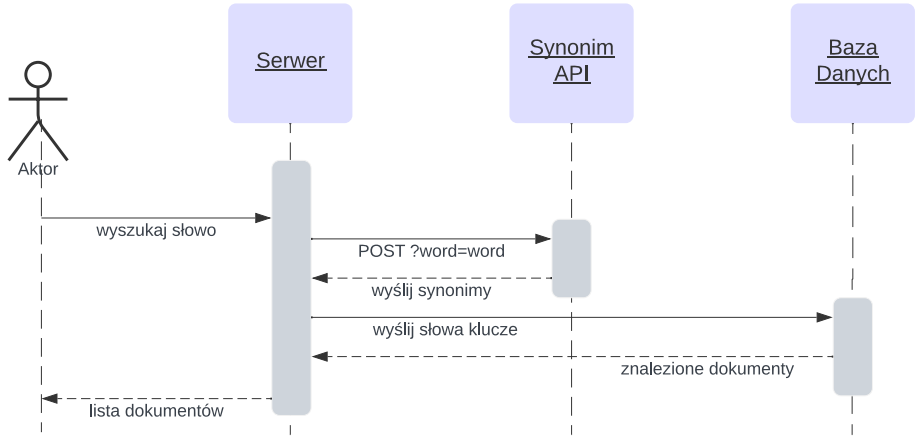


Rysunek 2: Diagram aktywności przesyłania pliku PDF.

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 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 3: Schemat sekwencyjny wyszukiwania pliku.

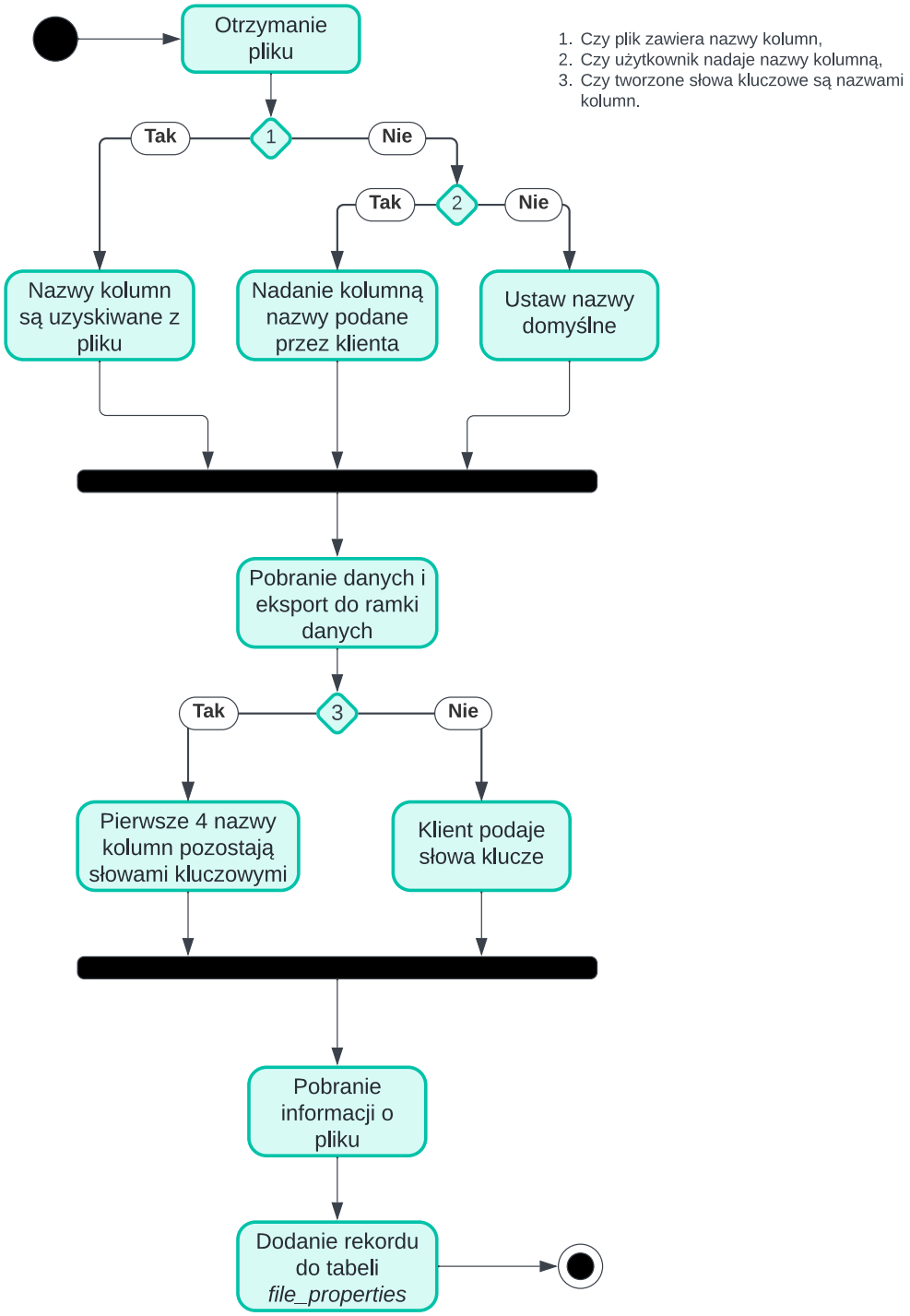
Obsługa plików zawierających dane tabelaryczne

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. Dane są odpowiednio wczytywane na serwer do postaci ramki danych.

3. Sprawdzany i zapisywany jest typ wartości w każdej kolumnie oraz obliczony procent ilości wartości brakujących.

4. Dodanie rekordu do bazy danych.



Rysunek 4: Diagram aktywności przetwarzania plików tabelarycznych.

Asocjacja plików o podobnej tematyce

Poniższa sekcja przedstawia proces łączenia dokumentów na podstawie wygenerowanych słów kluczowych. System, z każdym dodanym plikiem, przeszukuje 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 zawierającej informacje o zawartości plików, 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 snu"), 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 informującej o podobieństwach dokumentów.

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

3 Implementacja

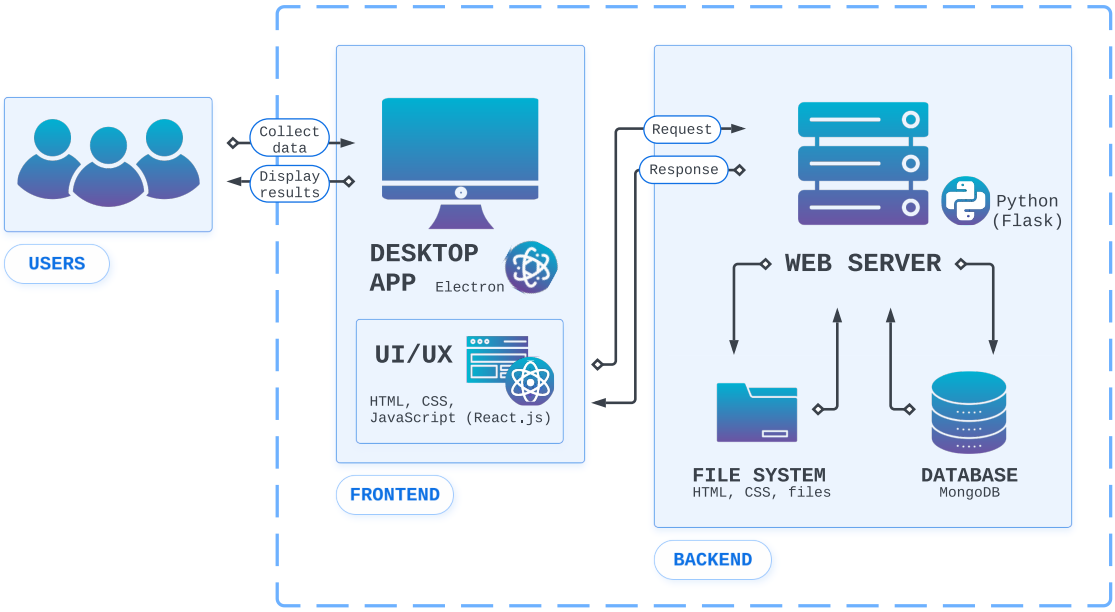
Realizacja miała na celu wykazanie możliwości projektu biorąc pod uwagę następujące aspekty. Po pierwsze, wykonalność aplikacji została oceniona w celu określenia jej praktyczności i potencjału pomyślnego rozwoju. Dodatkowo przeprowadzono analizę złożoności i wymagań dotyczących zasobów związanych z przetwarzaniem różnych typów plików, skupiając się na plikach tekstowych.

Co więcej, wysiłki ukierunkowano na identyfikację wymaganych rozszerzeń lub ulepszeń niezbędnych do optymalizacji funkcjonalności aplikacji, ze szczególnym uwzględnieniem algorytmów klasyfikacji czy modułu tworzenia abstraktów, opartych na sztucznej inteligencji. Wreszcie, podjęto planowanie w celu określenia zakresu wdrożenia w przydzielonych ramach czasowych, zapewniając zgodność z celami projektu i wykonalność w ramach ograniczeń dostępnych zasobów.

Do implementacji aplikacji użyto kombinacji technik typowych dla tworzenia aplikacji internetowych jakie zostały określone w rozdziale 1.4. Interfejs użytkownika (UI) został stworzony przy użyciu języka JavaScript, w szczególności wykorzystując bibliotekę React, wraz z językami opisu HTML i CSS do zdefiniowania struktury i stylizacji witryny. Jednak zamiast polegać na tradycyjnej przeglądarce internetowej, cały interfejs użytkownika został zamknięty w aplikacji natywej, dzięki wykorzystaniu szkieletu aplikacji Electron. Opis tego procesu jest zamieszczony w rozdziale 1.4.

Do przechowywania wyodrębnionych danych wybrano nierelacyjną bazę danych, a jako system zarządzania bazą danych wybrano MongoDB. Decyzja ta była podyktowana zróżnicowanym i niejednolitym charakterem danych, gdzie na strukturę każdego rekordu wpływają takie czynniki jak typ pliku.

Aby ustanowić komunikację między aplikacją a bazą danych, zaimplementowano serwer WWW przy użyciu biblioteki Flask języka Python. Działając jako istotny łącznik, serwer ten zarządza wszystkimi podstawowymi funkcjami aplikacji. Nadzoruje on zadania takie jak przetwarzanie plików, ekstrakcja danych, klasyfikacja i segregacja, wraz z wszelkimi niezbędnymi operacjami manipulacji danymi. Takie podejście zapewnia płynną interakcję między komponentami aplikacji, umożliwiając wydajną obsługę i zarządzanie danymi w całym systemie.



Rysunek 5: Diagram struktury aplikacji.

3.1 Opis wykorzystywanych elementów

Javascript to powszechnie używany język programowania do tworzenia stron internetowych, znany ze swojej wszechstronności i kompatybilności z przeglądarkami [14]. Opisywany system wykorzystuje Javascript w środowisku *Node.js*, co ułatwia dostęp do różnorodnych pakietów takich jak *React*, których opis jest przedstawiony poniżej.

- *React* - biblioteka pozwalająca na budowanie interaktywnych interfejsów użytkownika. Główną koncepcją React są komponenty, czyli samodzielne, hermetyczne jednostku interfejsu [15].

- *Axios* - klient HTTP z pomocą którego odbywa się komunikacja z serwerem. Jest to mechanizm oparty o obietnice (*ang. promises*) pozwala na asynchroniczną relacje 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 6: Przykład wykorzystania *Axios*; Implementacja dodania pliku.

Python to język programowania wysokiego poziomu znany ze swojej prostoty i czytelności, Stworzenie serwera w tym języku pozwala na korzystanie z jego rozległego ekosystemu bibliotek czy szkieletów aplikacji, w szczególności narzędzi dotyczących takich tematów, jak przetwarzanie i analiza danych czy sztuczna inteligencja [16]. Poniżej zostały przedstawione bibloteki używane w aplikacji elementy.

- Flask - jest to elastyczny szkielet, 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.

- PdfMiner - biblioteka przeznaczona do wyodrębniania tekstu i metadanych z dokumentów PDF.

- Yake - biblioteka stworzona do identyfikowania kluczowych fraz, które reprezentują główne tematy lub wątki dokumentu.

- Sumy - biblioteka umozliwiająca automatyczne tworzenie streszczeń tekstów. Posiada wiele algorytmów streszczania ekstraktywnego.

3.2 Architektura

Podstawowe funkcjonalności aplikacji są oparte na architekturze klient-serwer. Interfejs po stronie klienta, opracowany przy użyciu standardowych techink internetowych przedstawionych w rozdziale 1.4, komunikuje się serwerem w celu obsługi żądań użytkowników oraz przechowywania i przetwarzania danych.

Nie jest ona jednak aplikacją internetową. Po stronie klienta jest ona zamknięta w *Electronie*, szkielecie (*ang. framework*), umożliwiającym tworzenie wieloplatformowych aplikacji desktopowych przy użyciu technologii internetowych. Osadzając aplikację internetową w Electron, przekształca się ją w samodzielną aplikację desktopową, która może być dystrybuowana i uruchamiana natywnie w różnych systemach operacyjnych (Windows, macOS, Linux). Ta integracja oferuje użytkownikom znane doświadczenie interakcji z aplikacją desktopową przy jednoczesnym zachowaniu korzyści płynących z technik webowych.

3.3 Status realizacji

Ze względu na ograniczenia czasowe, implementacja obejmuje następujące elementy określone w rozdziale 2. Odchylenia od założeń przedstawionych dziale związanym z opisem projektu zostały opisane w rozdziale 3.4, a poniżej wydnieje lista częściowo lub w pełni zawartych elementów w zależności od modułu.

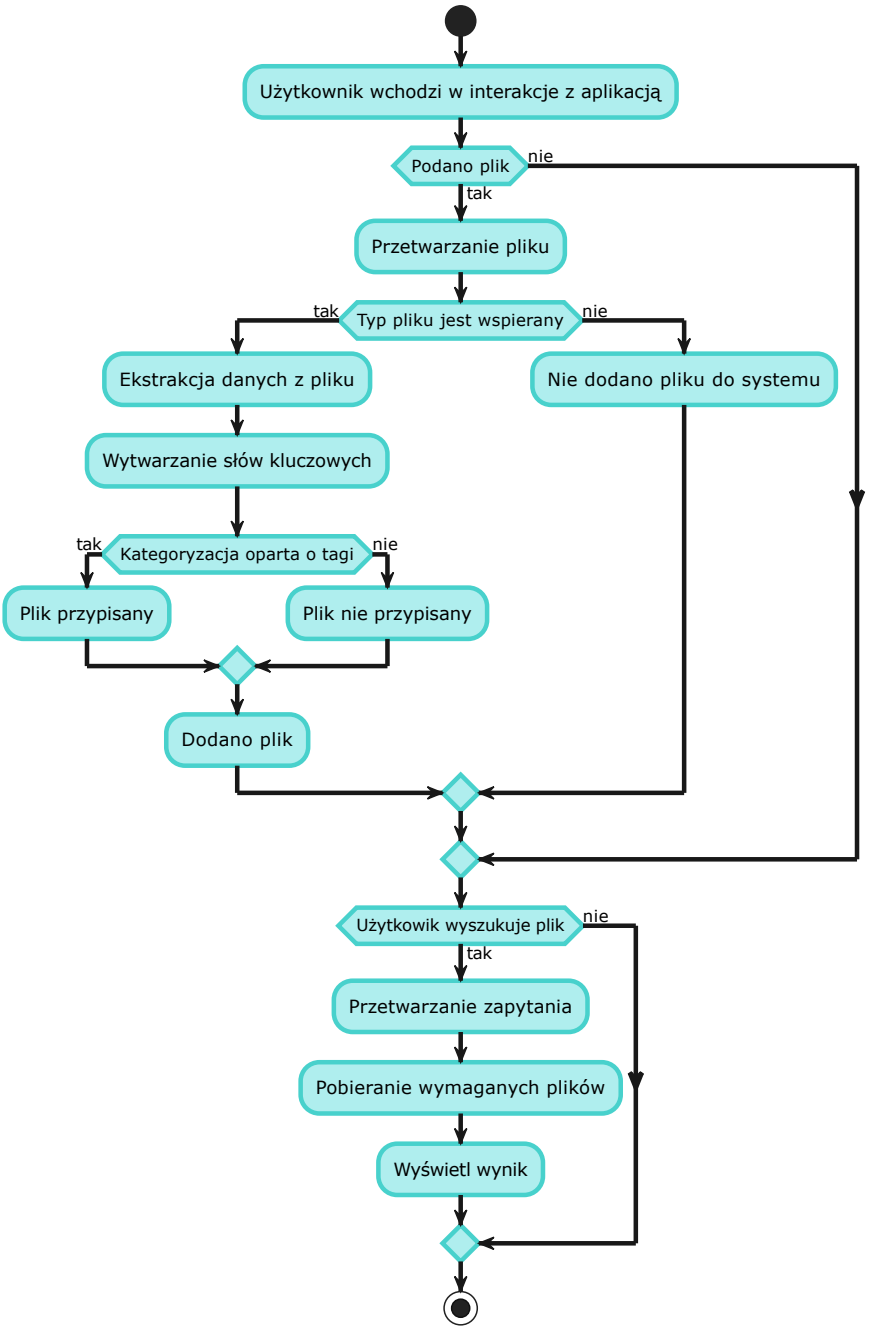
1. Moduł organizacji danych.
    - Zrealizowano: tagowanie plików poprzez generację słów kluczy na podstawie zawartości oraz łączenie plików.
    - Odrzucono: automatyczne wytwarzanie struktury folderów oraz kategoryzacja.
2. Wyszukiwarka.
    - Zrealizowano: wyszukiwanie, prezentacja wyników wyszukiwania oraz możliwość filtrowania wyników.
    - Odrzucono: sugestie i autouzupełnianie.
3. Moduł podsumowujący.
    - Zrealizowano: Tworzenie streszczeń metodą ekstraktywną opartą o algorytm TextRank.

- Odrzucono: Wykorzystanie metod abstrakcyjnych.

4. Przetwarzanie plików.
   - Zrealizowano: Przetwarzanie cześciowe dla plików o rozszerzeniu PDF oraz CSV.

Opis specyfikacji zależnej od rodzaju pliku jest przedstawiony w rozdziale 3.5.

Końcowy diagram aktywności



Rysunek 7: Diagram aktywności końcowej aplikacji.

3.4 Ograniczenia implementacji

Organizacja plików

Podczas tworzenia systemu napotkano problem dotyczący modułu organizacji danych, a dokładniej automatycznego tworzenia struktur katalogów. Proces ten wymaga wzięcia pod uwagę wielu warunków co sprawia, że koszty implementacji są niewspółmierne do oczekiwanych wyników. Przykładowe wymagania zostały wypisanie poniżej.

1. Występowanie niejednoznacznych lub nakładających się słów kluczowych w treści pliku może prowadzić do niepewności w kategoryzacji. Jeśli słowo kluczowe jest powiązane z wieloma kategoriami lub jeśli różne pliki zawierają podobne słowa kluczowe o różnych znaczeniach kontekstowych, może to spowodować błędną klasyfikację plików do nieprawidłowych katalogów.

2. Zmienność w użyciu słów kluczowych, takich jak synonimy, skróty lub różnice w pisowni, może stanowić wyzwanie w dokładnej identyfikacji i wyodrębnianiu odpowiednich słów kluczowych z zawartości pliku.

Zważając na ograniczenia czasowe, została podjęta decyzja o ograniczeniu modułu organizacji danych do implementacji systemu generacji słów kluczowych z zawartości plików oraz łączenia plików ze względu na nie. Schemat działania asocjacji plików został przedstawiony w rozdziale 2.3.

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 systemu nie jest optymalna pod względem ilości zasobów potrzebnych do uczenia modelu oraz zasobów pamięciowych przeznaczonych na działanie całej aplikacji. Istnieje jednak rozwiązanie tego problemu z pomocą dostępnych na rynku *API*. Wiele narzędzi zostało wytrenowane na ogromnych zestawach danych w celu tworzenia takich abstraktów, tak jak oferowany przez *OpenAI* system streszczeń oparty na *GPT-3*.

3.5 Obsługiwane rodzaje plików

System w aktualnym stanie realizacji jest przystosowany do obsługi plików w formatach, między innymi takich jak *PDF* oraz *CSV*. W zależności od struktury pliku, system udostępnia stosowne funkcjonalności, co jest opisane poniżej. Do plików przypisujemy etykietę opisującą rodzaj pliku, lecz jest ona odrębną jednostką od słów kluczowych generowanych na podstawie zawartości pliku.

Pliki PDF

Proces obsługi plików PDF jest zależny od wielu czynników. Najprostszymi 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, że rzadko 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 powinny 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: etykietami zostają nazwy kolumn lub użytkownik samodzielnie je dodaje.

Żadne z wymienionych podejść, nie jest wystarczające. Nazewnictwo kolumn tworzonych tabel nie jest ustandaryzowane, ani wymagane, co może powodować wytworzenie mylących lub bezużytecznych nagłówków. Dlatego w pełni zautomatyzowany algorytm może działać wadliwie. Drugie podejście wymaga od użytkownika większego wkładu w ten proces. Poprawne etykietowanie wprowadzanego pliku, wy-

maga wiedzy dotyczącej zawartości zestawu danych i informacji o pliku. Z wymienionych powodów, w implementacji są stosowane oba sposoby.

3.6 Możliwości rozwoju i wykorzystania aplikacji

Wspomniane funkcjonalności to jedynie początek istnienia tego systemu. Stworzenie dodatkowych zdolności programu jest uproszczone dzięki oparciu 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 nieustrukturyzowany sposób. Dzięki czemu tabele można wypełnić dowolnymi wartościami, a sama aplikacja wyświetla wyszystkie elementy tabeli dla wyszukiwanego obiektu.

3.6.1 Ekspansja działań dotyczących danych tabelarycznych

Aktualny system pozwala na implementacje metod wstępnego przetwarzania danych (*ang. preprocessing*) z wnętrza aplikacji. Preprocessing oznacza zbiór działań mających 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. Dodanie takich możliwości jak łączenie tabel w pojedyńcze pliki czy, w przypadku plików zawierających kilka osobnych arkuszy danych, pozwolenie na rozdzielenie ich na osobne pliki, powinno być celem kolejnych implementacji.

3.6.2 Identyfikacja obrazów

Identyfikacja obrazów może przysłużyć się sukcesowi systemu. Dla człowieka proces identyfikacji elementów obrazu dzieje się podświadomie, a w przypadku programu, jest on w stanie rozpoznać i przypisać grafikę do odpowiedniej kategorii, jedynie w sytuacji gdy jest on zaprogramowany do wykrywania ich. Komputery kategoryzują obrazy, porównując układ pikseli wejściowego obrazu ze schematami zapisanymi w systemie. Z punktu widzenia maszyny, obraz jest ustrukturyzowaną 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. gromadzenie danych,
2. przygotowanie danych,
3. wybór, trenowanie i testowanie modelu,
4. wprowadzenie modelu do systemu.

Najważniejszym elementem skutecznego systemu rozpoznawania obrazów jest dobrze skonstruowany zbiór danych. Etap gromadzenia danych obejmuje pozyskiwanie różnorodnych obrazów reprezentujących obiekty, sceny lub wzorce, które system nauczy się rozpoznawać. Kompleksowy zbiór danych przyczynia się do zdolności systemu do uogólniania swojego zrozumienia w różnych scenariuszach.

Przygotowanie danych (*ang. Preprocessing*) to proces przekształcania i udoskonalania zebranych danych, aby były odpowiednie do przeprowadzenia treningu systemu. Może to obejmować takie zadania, jak usuwanie niespójności, zmiana rozmiaru obrazów do spójnego formatu i normalizacja wartości pikseli. Skuteczny preprocessing danych wspomaga zdolność modelu do nauki istotnych wzorców z danych.

Kluczową decyzją jest wybór architektury modelu, która powinna być dostosowana do wymagań zadania i zasobów obliczeniowych. Określa ona sposób, w jaki system będzie przetwarzał i interpretował dane wejściowe. Trening modelu to proces uczenia wybranej architektury rozpoznawania wzorców i cech w danych treningowych. Osiąga się to poprzez iteracyjną optymalizację, w której parametry modelu są dostosowywane w celu zminimalizowania różnicy między przewidywanymi a rzeczywistymi wynikami. Trening trwa do momentu, aż model osiągnie optymalny poziom dokładności i uogólnienia, który sprawi, że będzie on biegły w rozpoznawaniu obiektów na obrazach.

Ostatnim etapem, poprzedzającym implementacje modelu, jest testowanie wytrenowanego modelu. Jest ono niezbędne do oceny jego wydajności dla nowych danych. System jest zasilany obrazami, których nie napotkał podczas szkolenia, umożliwiając ocenę jego zdolności do generalizacji. Metryki takie jak dokładność, precyzja i czułość zapewniają wgląd w mocne i słabe strony modelu. Rygorystyczne testowanie pomaga w dostrojeniu modelu w celu uzyskania lepszej wydajności [17].

Powszechnie dostępne, ogromne bazy danych, takie jak *Pascal VOC* czy *ImageNet*, pozwalają na stworzenie i wdrożenie takiego modelu. Zawierają one mnóstwo oznaczonych wzorców opisujących obiekty znajdujące sie na obrazach.

Podsumowanie

W ramach pracy zaprezentowano projekt aplikacji, która umożliwiłaby komfortowe przechowywanie danych oraz zmniejszenie ingerencji użytkownika w proces interpretacji plików. Opisano funkcjonalności aplikacji oraz możliwe rozszerzenia aplikacji w dalszym ciągu rozwoju.

Jednym z intrygujących wyzwań była implementacja ekstrakcji tekstu z plików PDF. Analiza tego procesu skupiła się głównie na wyodrębnianiu tekstu bezpośrednio ze strumienia pliku, podkreślając złożoność i interpretowania struktur dokumentów. Skłoniło to również do rozważenia alternatywnych metod ekstrakcji, takich jak optyczne rozpoznawanie znaków (OCR).

Ponadto, chociaż aplikacja oferuje możliwość wyszukiwania, ważne jest, aby uznać jej potencjalne ograniczenia, szczególnie w przypadkach, gdy błędy ekstrakcji tekstu sprawiają, że słowa kluczowe są nieskuteczne dla dokładnych wyników wyszukiwania.

Architektura aplikacji pozwala jednak na jej przyszłą rozbudowę i ulepszenie. Integracja z zewnętrznymi interfejsami API lub dodanie rozszerzeń możę zwiększyć jej funkcjonalność. Co więcej, wykorzystanie bazy danych NoSQL zapewnia elastyczność w przechowywaniu różnych typów informacji, przyczyniając się do skalowalności aplikacji. Niemniej jednak ważne jest, aby zdać sobie sprawę, że wdrożenie kategoryzacji według tematów w celu tworzenia katalogów w aplikacji stanowi znaczące przedsięwzięcie. Osiągnięcie tej funkcjonalności wymaga znacznych inwestycji w czas i zasoby, ponieważ opiera się na płynnym funkcjonowaniu różnych połączonych ze sobą komponentów w ekosystemie aplikacji.

Spis rysunków

Literatura

[1] Z. Parker, S. Poe, and S. V. Vrbsky, "Comparing nosql mongodb to an sql db," in *Proceedings of the 51st ACM Southeast Conference*, ser. ACMSE '13. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: https://doi.org/10.1145/2498328.2500047

[2] I. MongoDB, "Mongodb." [Online]. Available: https://www.mongodb.com/

[3] D. Fagbuyiro, "On-premise vs cloud solutions: "choosing the best fit for your business"," Feb 2023. [Online]. Available: https://strapi.io/blog/on-premise-vs-cloud-solutions-which-is-the-best-for-your-business

[4] R. Ho, "On premise vs cloud: What's the difference," May 2022. [Online]. Available: https://blog.tessaract.io/on-premise-vs-cloud-whats-the-difference

[5] International Organization for Standardization, "ISO 32000:2008," Geneva, Switzerland, 2008, document management – Portable document format – Part 1: PDF 1.7.

[6] 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.

[7] 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

[8] 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

[9] 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

[10] 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

[11] H. S. Oluwatosin, "Client-server model," *IOSR Journal of Computer Engineering*, vol. 16, no. 1, pp. 67–71, 2014.

[12] I. GitHub, "Electron," GitHub Repository, 2013. [Online]. Available: https://github.com/electron/electron

[13] 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

[14] "Javascript," Web Browser, 1995. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[15] Facebook, "React," 2013. [Online]. Available: https://react.dev/

[16] G. van Rossum and P. S. Foundation, "Python," Python Software Foundation, 1991. [Online]. Available: https://www.python.org/

[17] D. Lu and Q. Weng, "A survey of image classification methods and techniques for improving classification performance," *International Journal of Remote Sensing*, vol. 28, no. 5, pp. 823–870, 2007. [Online]. Available: https://doi.org/10.1080/01431160600746456