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

Dane odgrywają niezwykle istotną rolę w funkcjonowaniu współczesnego społeczeństwa, będąc fundamentem dla wielu procesów i decyzji. Dzięki danym można analizować i rozumieć zmiany w otaczającym świecie, identyfikować wzorce i trendy oraz przewidywać przyszłe wydarzenia.

Przechowywanie danych jest niezbędne nie tylko ze względu na konieczność zachowania informacji historycznych, ale także po to, aby zapewnić dostępność potrzebnych informacji w odpowiednim czasie oraz miejscu. Organizacje różnego rodzaju gromadzą ogromne ilości informacji, które są kluczowe dla ich funkcjonowania i rozwoju. Systematyczne przechowywanie tych danych pozwala na śledzenie zmian w czasie, analizę trendów oraz identyfikację ważnych wzorców i relacji, co z kolei umożliwia podejmowanie trafniejszych decyzji biznesowych. Wykorzystując dane, można personalizować doświadczenia użytkowników, dostosowując produkty i usługi do ich potrzeb i preferencji.

W kontekście poniższej pracy, zbiór 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.

Zapewnienie bezpieczeństwa i poufności gromadzonych informacji jest również ważnym aspektem przechowywania danych. W obliczu coraz częstszych przypadków naruszeń danych i cyberataków, odpowiednie zabezpieczenie przechowywanych danych staje się priorytetem. Dlatego też organizacje, ale także osoby prywatne muszą inwestować w zaawansowane systemy ochrony danych i stosować odpowiednie procedury zarządzania dostępem do informacji.

Cel i zakres projektu

Projekt ma na celu przedstawienie możliwości systemu ułatwiającego użytkownikowi proces zbierania i segregowania informacji. Poniższa praca skupi się na zaprezentowaniu konceptu aplikacji oraz szczegółowym opisie implementacji modułu obsługi artykułów.

1 Zagadnienia teoretyczne

Zaprojektowany system ma uprościć proces gromadzenia informacji dla użytkownika prywatnego. Zgromadzone dane mogą być przechowywane w pliku w systemie plików lub przy użyciu takich narzędzi jak bazy danych. Rozważając rodzaj przechowywania danych należy również wziąć pod uwagę aspekt fizyczny.

Projekt wykorzystuje architekture typu klient-serwer, jednakże dzięki wykorzystaniu pakietu *Electron* języka Javascript, jest możliwe przekształcenie aplikacji webowej na natywną. Biblioteka Electron jest projektem umożliwiającym tworzenie aplikacji natywnych używając technologii web. Pełny opis tej biblioteki jest zawarty w podrozdziale 1.4.

1.1 Sposoby przechowywania informacji

Istnieje wiele sposobów magazynowania danych. Wybór odpowiedniego sposobu zależy od indywidualnych potrzeb użytkownika oraz od ilości informacji. Na poziomie organizacji dzielą się one na serwery lokalne (*ang. on-premises servers*) oraz przechowywanie w chmurze (*ang. cloud storage*). Poniżej zostały opisane obie metody.

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. Oto przegląd zalet i wad korzystania z serwerów lokalnych do przechowywania danych:

- zalety:

    1. Bezpośrednia kontrola - Organizacje mają pełną kontrolę nad swoim sprzętem i oprogramowaniem, co pozwala na większą elastyczność zarządzania w celu spełnienia określonych potrzeb biznesowych i wymogów bezpieczeństwa.
    2. Bezpieczeństwo - Serwery lokalne zapewniają wyższy poziom bezpieczeństwa i prywatności danych, ponieważ mogą one wdrażać własne środki bezpieczeństwa i protokoły dostosowane do ich konkretnych potrzeb. Może to obejmować fizyczne środki bezpieczeństwa, takie jak ograniczony dostęp do serwerowni i budynków.

3. Zgodność z wymaganiami - W przypadku branż o surowych wymogach regulacyjnych, takich jak opieka zdrowotna lub finanse, serwery lokalne oferują lepszą kontrolę zgodności z nimi. Właściciele mogą zapewnić, że praktyki przetwarzania danych są zgodne z odpowiednimi regulacjami i przepisami.

4. Personalizacja - Oferują one elastyczność w zakresie dostosowywania konfiguracji sprzętowych lub konfiguracji sieci w celu optymalizacji wydajności i spełnienia określonych wymagań dotyczących obciążenia.

- wady:

1. Koszty początkowe oraz utrzymania - Konfiguracja serwerów lokalnych wymaga znacznych inwestycji w sprzęt, licencje, konfigurację sieciowy i strukturalną. Może to obejmować koszty związane z zakupem serwerów, macierzy dyskowych, przełączników sieciowych czy systemów chłodzenia. Wymusza to również odpowiedzialność za bieżącą konserwację, zarządzanie oprogramowaniem serwerowym, co może wymagać dedykowanego personelu i zasobów IT. Obejmuje to zadania takie jak naprawy sprzętu fizycznego lub aktualizacje oprogramowania.

2. Ograniczenia skalowalności - Należy zakupić dodatkowy sprzęt i rozszerzyć infrastrukturę, aby sprostać rosnącym potrzebom w zakresie przechowywania i przetwarzania danych.

3. Ograniczona redundancja geograficzna - serwery mogą nie mieć redundancji geograficznej, co oznacza, że dane są przechowywane w jednej lokalizacji i są podatne na zagrożenia, takie jak klęski żywiołowe, pożary lub inne nieprzewidziane zdarzenia, które mogą spowodować utratę danych.

Usługi **przechowywania w chmurze**, takie jak Amazon Web Services (AWS), zapewniają możliwość przechowywania danych i zarządzania nimi w zdalnych centrach danych utrzymywanych przez zewnętrznych dostawców usług. Poniżej przedstawiono przegląd wad i zalet korzystania z usług przechowywania danych w chmurze:

- zalety:

    1. Skalowalność: Usługi przechowywania danych w chmurze oferują praktycznie nieograniczoną skalowalność, umożliwiając organizacjom łatwe zwiększanie lub zmniejszanie pojemności pamięci masowej w zależności od zapotrzebowania. Ta elastyczność umożliwia firmom szybkie dostosowanie się do zmieniających się wymagań dotyczących pamięci masowej bez konieczności inwestowania z góry w sprzęt.
    2. Opłacalność: Pamięć w chmurze w większości przypadków działa w modelu cenowym pay-as-you-go, w którym organizacje płacą tylko za przestrzeń dyskową i zasoby, które wykorzystują. Może to prowadzić do oszczędności kosztów w porównaniu z tradycyjnymi lokalnymi rozwiązaniami.
    3. Dostępność: Zapewniają wszechobecny dostęp do danych z dowolnego miejsca z połączeniem internetowym. Umożliwia to zdalny dostęp, współpracę i udostępnianie danych między rozproszonymi zespołami, poprawiając produktywność oraz rozszerzając działalność firmy globalnie.
    4. Wysoka dostępność i niezawodność: Usługi przechowywania danych w chmurze zazwyczaj oferują wbudowane funkcje redundancji i wysokiej dostępności, zapewniając replikację danych w wielu centrach danych i ochronę przed awariami sprzętu lub innymi zakłóceniami. Skutkuje to zwiększoną niezawodnością danych i minimalnymi przestojami.
    5. Bezpieczeństwo: Wiodący dostawcy rozwiązań chmurowych wdrażają solidne środki bezpieczeństwa w celu ochrony danych, w tym szyfrowanie, kontrolę dostępu i certyfikaty zgodności.

- wady:

    1. Zależność: Zależność od dostawcy usług w zakresie dostępności, bezpieczeństwa i wydajności danych. Wszelkie zakłócenia lub przerwy w świadczeniu usług po stronie dostawcy mogą mieć wpływ na dostęp do danych i operacje biznesowe.
    2. Zarządzanie danymi i zgodność z przepisami: Organizacje muszą upewnić się, że dane przechowywane w chmurze są zgodne z odpowiednimi

przepisami i standardami.

3. Koszty transferu danych: Podczas gdy przechowywanie danych w chmurze może być opłacalne, organizacje mogą ponieść dodatkowe koszty transferu danych, zwłaszcza w przypadku przenoszenia dużych ilości danych do i z chmury. Koszty mogą znacznie wzrosnąć w przypadku obciążeń wymagających dużej przepustowości.
4. Problemy ze zmianą miejsca przechowywania: Migracja danych między dostawcami usług w chmurze lub powrót do infrastruktury lokalnej mogą być złożone i kosztowne. Organizacje ryzykują uzależnienie od dostawcy, gdy mocno inwestują w ekosystem określonego dystrybutora rozwiązań, co w dłuższej perspektywie ogranicza ich elastyczność.
5. Obawy dotyczące bezpieczeństwa danych i prywatności: Pomimo solidnych środków bezpieczeństwa wdrożonych przez dostawców usług w chmurze, organizacje mogą nadal mieć obawy dotyczące bezpieczeństwa i prywatności swoich danych przechowywanych w chmurze.

Wybór miejsca przechowywania danych zależy od takich czynników, jak wymagania bezpieczeństwa, potrzeby skalowalności czy ograniczenia budżetowe. Często korzystne dla organizacji jest przyjęcie podejścia hybrydowego lub wielochmurowego, wykorzystującego mocne strony różnych opcji przechowywania danych. Umożliwia to zaspokojenie różnorodnych potrzeb biznesowych przy jednoczesnym złagodzeniu wpływu niedoskonałości różnych rozwiązań.

1.2 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 cech czytanej 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 [1]. Skracając czas potrzebny na napisanie abstraktu, automatyzacja ma na celu ułatwienie autorom pisania artykułów. Systemy ATS (ang. *Automatic Text Sum-*

marization) są jednym z cięższych wyzwań sztucznej inteligencji, dotyczących przetwarzania języka naturalnego [2]. Podejścia do tworzenia tych systemów, możemy podzielić na ekstraktywane, abstrakcyjne i hybrydowe.

1.2.1 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 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 wprowadanego 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 [2].

1.2.2 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.

- *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.

1.3 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 [5].

Format PDF separuje informacje dotyczące samego znaku a jego wizualną reprezentacją. Jest to rozróżnienie na znak pisarski i glif, gdzie grafem jest jednost-

ką 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 [6]. 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 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.

1.4 Pakiet Electron

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

Izolacja wątku

W Electronie separacja wątków odnosi się do możliwości izolacji różnych części aplikacji w celu zapobiegania interferencji oraz zapewnienia stabilności i bezpieczeństwa. 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. Separacja ta 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 deweloperzy mogą zaimplementować dwukierunkową komunikację między procesem głównym a procesami renderującymi, umożliwiając im koordynację działań, udostępnianie danych i synchronizację stanu w różnych częściach aplikacji.

2 Projekt systemu

System w założeniu działa następująco, użytkownik dodaje plik a serwer zajmuje się resztą, po stronie serwera jest wykrywany rodzaj pliku i wykonywane są specyficzne dla niego operacje

3 Implementacja

3.1 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.

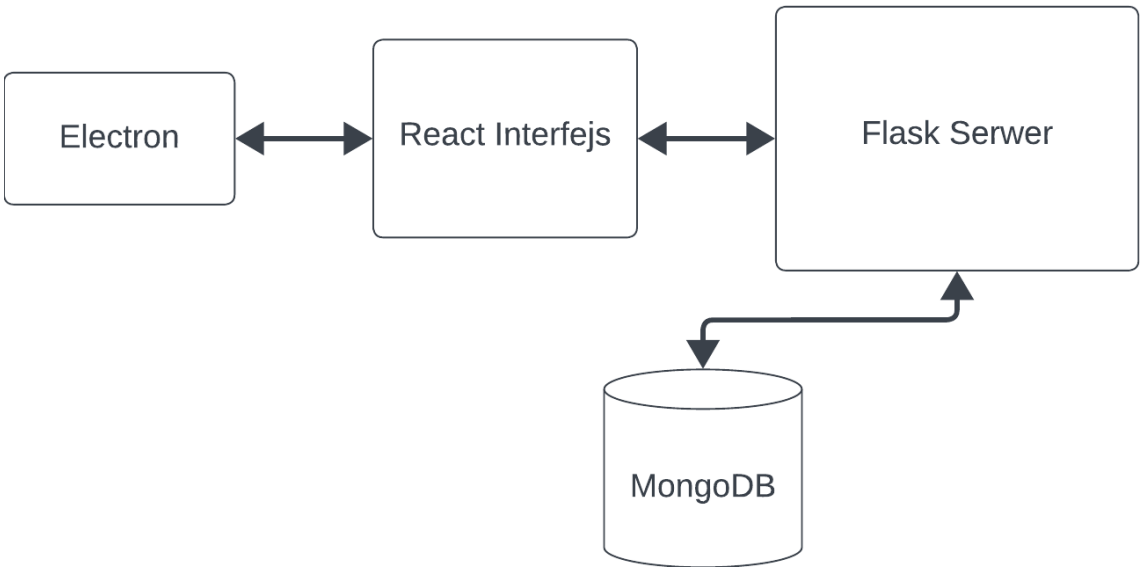
3.2 Architektura systemu

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



Rysunek 1: Diagram łączenia komponentów

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.

3.2.1 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

3.2.2 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.

```python
from flask import Flask

app = Flask(__name__)

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

Rysunek 3: Przykładowa implementacja funkcji serwera

3.3 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, że 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*.

3.4 Schematy poszczególnych funkcjonalności

3.4.1 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,

```
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()*

- *top* - ilość elementów wyjściowych

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.

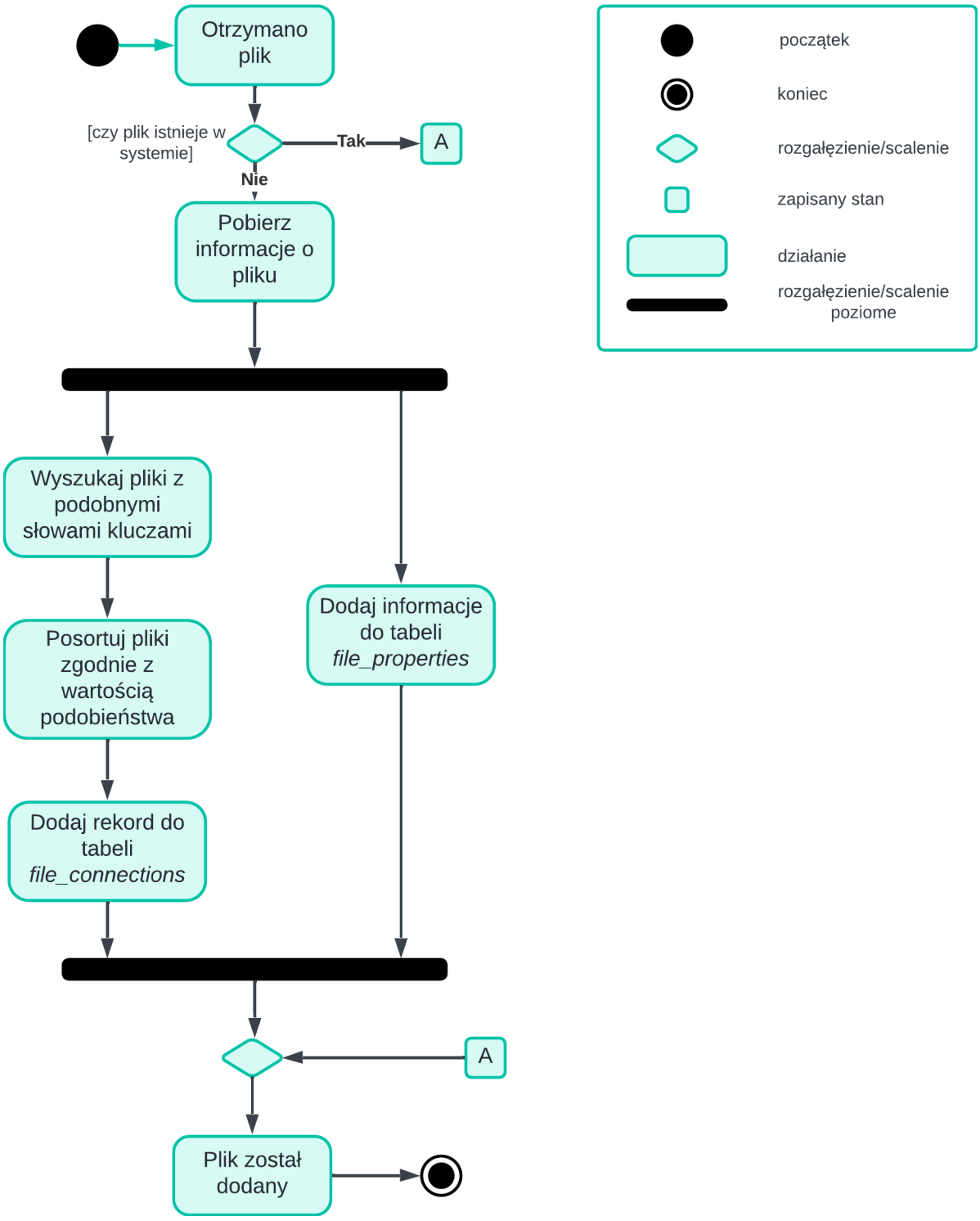
Model TextRank zastosowany w projekcie, jest algorytmem rankingowym opartym na grafach [7]. Operuje on na zasadach "głosowania" i "rekomendacji". Jest on używany do tworzenia streszczeń metodą ekstraktywną i nienadzorowaną. Dla przedstawionego systemu, nie jest możliwe wytworzenie odpowiedniego zbioru 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.

3.4.2 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,



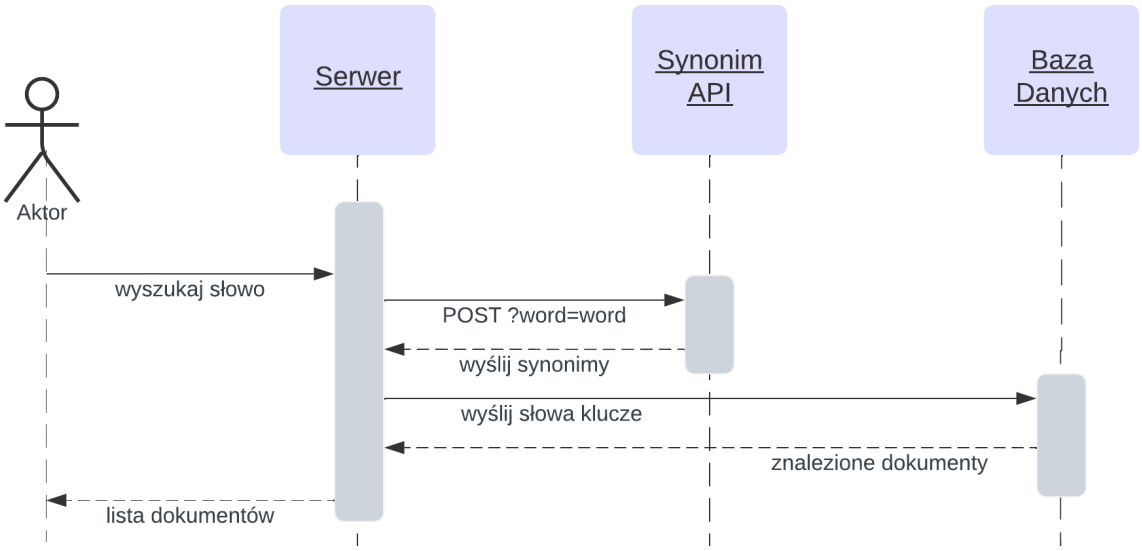
Rysunek 5: Diagram aktywności przesyłania pliku PDF

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.



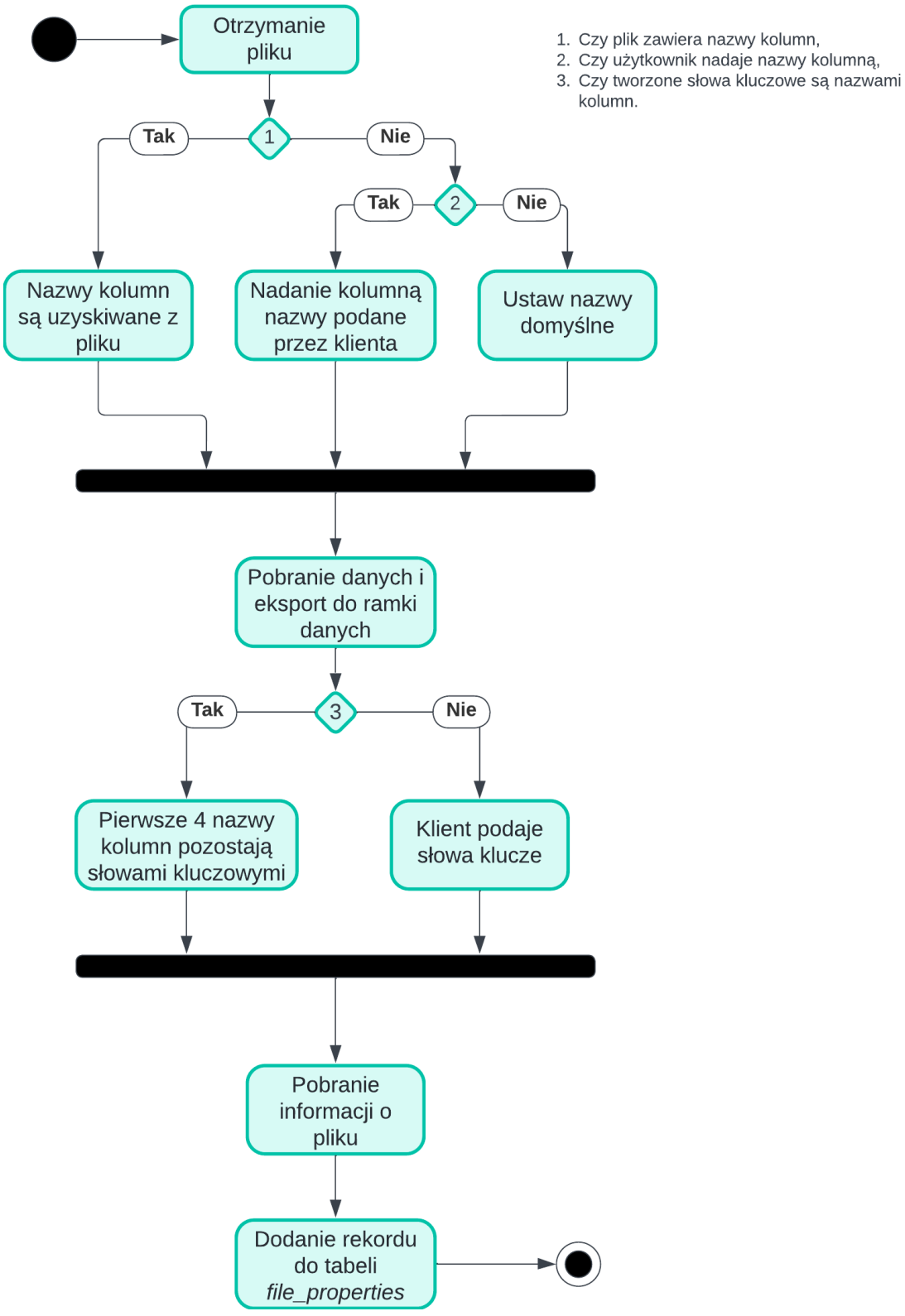
Rysunek 6: Schemat sekwencyjny wyszukiwania pliku

```
@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

3.4.3 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

3.4.4 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 *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

4 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.

4.1 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. Dodanie takich możliwości jak łączenie tabel w pojedyńcze pliki czy w przypadku posiadania przez plik osobnych arkuszy danych pozwolenie na rozdzielenie ich na osobne pliki, powinno być celem kolejnych implementacji.

4.2 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*.

4.3 Identyfikacja obrazów

Identyfikacja obrazów mogą 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. Wybór architektury powinien być dostosowany 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 przewi-

dywanymi 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.

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 projektu zaimplementowano aplikację umożliwiającą komfortowe przechowywanie oraz zmniejszenie ingerencji użytkownika w proces interpretacji plików. Opisano funkcjonalności aplikacji oraz możliwe rozszerzenia aplikacji w dalszym ciągu rozwoju.

Badania nad aplikacją wyeksponowały ciekawy koncept jakim jest praca z pilkami o rozszerzeniu PDF, jakoże proces automatycznego pobierania tekstu jest zależny od wielu czynników. Okazało się, że rozkład tekstu na stronie jak i program, którego użyto do stworzenia pliku może przyczynić się do błędnego eksportu tekstu. Nie jest możliwe uwzględnienie każdego przypadku stuktury pliku aby zapewnić prawidłowy proces ekstrakcji. Należałoby zastanowić się nad poprawą implementacji tego algorytmu lub zastosowanie innych sposobów, takich jak *Optical Character Recognition*. System powinien być usprawniony o implementacje serwera HTTP oraz logowanie, aby zezwolić na przenoszenie danych użytkownika.

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] 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] 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