МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Інформаційна технологія екстракції метаданих для аналізу документів»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Москаленко В.В.**

Студента групи ІН – 61 **Кудрявцев А.М.**

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2020 г.

ЗАВДАННЯ

до випускної роботи

Студента четвергого курсу, групи ІН-61 спеціальності "Інформатика" денної форми навчання Кудрявцев Антон Михайлович.

Тема: "**Інформаційна технологія екстракції метаданих для аналізу документів**"

Затверджена наказом по СумДУ

№_____ от _____2020 г.

Зміст пояснювальної записки: 1) аналіз проблеми обробки неструктурованих даних; 2) формалізована постановка задачі дослідження; 3) опис інформаційної технології екстракції метаданих; 4) опис програмної реалізації; 6) аналіз результатів.

Дата видачі завдання "_____"_____2020 г.

Керівник випускної роботи _____ Москаленко В.В.

Завдання прийняв до виконання _____ Кудрявцев А.М.

РЕФЕРАТ

Записка: 58 стор., 15 рис., 3 табл., 3 додатки, 36 джерел

Об'єкт дослідження — інформаційна технологія екстракції метаданих для аналізу документів.

Мета роботи — розробити систему та моделі екстрактору метаданих з документа. Модель повинна вміти аналізувати та виділяти метадані з тексту та зображень. Система повинна дозволяти користувачу завантажувати документи та виконувати пошук.

Результати — проаналізовано та розроблено модель для екстракції метаданих з документа. Метадані отримуються з моделей аналізу тексту та зображень. Було розроблено програмну реалізацію, яка дозволяє виконувати пошук по обробленим документам та завантажувати документи. Отримані результати свідчать про придатність для практичного використання.

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ, МАШИННЕ НАВЧАННЯ, МЕТАДАНІ, МОДЕЛІ МАШИННОГО НАВЧАННЯ, WORD, ELASTICSEARCH, KIBANA, KOTLIN, PYTHON, NODEJS, RABBITMQ

ЗМІСТ

ВСТУП

У сучасномі світі відбувається накоплення даних. Дані − представлення фактів, концепцій чи інструкцій у формалізованому порядку. Вони повинні бути придатними для використанні у спілкування, інтерпретації або для обробки людиною або електронною машиною. Накопичення необхідне для розвитку бізнесу та науки. У сучасному суспільству діяльність людини характеризується найбільшим зростанням кількості поток. Накопичення даних відбувається практично в кожній сфері бізнесу та науки. Проблема обробки та зберігання приділяється велика увага. Ця проблема обумовлена наступними процесами:

1) зростання збереженої та циркулюючої інформації в суспільстві;
2) розвиток комунікаційних процесів;
3) інформаційні технології проникли в кожну сферу.

Зростання обсягу потоків інформації найбільше спостерігається в торгівлі, промисловості та освітній сферах.

Зростання кількості інформації в промисловості зумовлене ускладненням продукції, збільшенням обсягу масового виробництва. Необхідно зберігати інформацію про використання матеріалів, контактів, продаж, технологічного обладнання, зовнішні та внутрішніх зв'язки економічних об'єктів. Саме завдяки збору даних бізнес має якісну інформацію, необхідну для прийняття обґрунтованих рішень для подальшого аналізу, вивчення та дослідження. Без збору даних компанії не можуть прийняти обґрунтоване рішення, використовуючи застарілі методи для прийняття своїх рішень. Збір даних натомість дозволяє їм бути в курсі тенденцій, дати відповіді на проблеми та проаналізувати нову інформацію з найкращим ефектом. Збір даних важливий, для аналізу споживачів та загального цільового ринку бізнесу. Дані в усіх формах і формах − це ключ до розуміння ваших клієнтів. За допомогою правильних даних ви зможете можна визначити коло ваших клієнтів. Створення особистості покупця є важливим для зусиль бізнесу зв'язатись із ринком, оскільки це вплине

на весь контент, рекламу та події, щоб залучити потенційних замовників. Дані також допоможуть зрозуміти продуктивність продукції та проаналізувати, які товари користується попитом, а яка − втрачають значення. Це допоможе у керуванні запасами − збільшити пропозицію одних та зменшити інших. Це може допомогти заощадити багато грошей на операціях та врешті-решт допоможе збільшити прибуток та зменшити виробничі витрати.

Зі зростанням кількості інформації, а зокрема кількості неструктурованих даних, виникає необхідність їх аналізувати та покращити пошук за цими даними.

1 АНАЛІЗ ПРОБЛЕМИ ОБРОБКИ НЕСТРУКТУРОВАНИХ ДАНИХ

1.1 Сучасний стан та тенденції розвитку технологій аналізу неструктурованих даних

Інформація − це будь які дані, відомості, повідомлення, які підлягають зберіганню, обробці та передачі. Інформація постійно збирається та передається в мережу із різноманітних пристроїв: планшетів, смартфонів, комп'ютерів, а незабаром потрапляти навіть через телевізори, датчики, годинники, автоматичне обладнання, та ще через велику кількість різноманітних девайсів. Щодня люди створюють величезні обсяги інформації: спілкування через соцмережі, онлайн-покупки, пошук інформації, завантаження медіа-контенту, все це створює гігабайти даних, які зберігаються, обробляються та передаються за допомогою віддалених серверів [1–2].

Дані поділяються на два типи:

1) традиційні дані;
2) великі дані.

Термін «традиційні дані» не є науковим, він використовується лише для того, щоб розмежувати дані, які зберігаються в базах даних та містять структуровані таблиці, наповнені цифровою, текстовою та іншою інформацією, від великих даних (big data).

Традиційними даними, наприклад, є інформація про користувачів та клієнтів сайту: ПІБ, контактна інформація, адреса, кількість відвідувань, або запитів, зроблених на сайті.

Великі дані або big data − це теж дані, проте величезних розмірів, які зростають в геометричній прогресії. Такі дані є настільки великими швидкими та складними, що їх важко або навіть неможливо ефективно обробляти за допомогою традиційних методів [3-4].

Існують п'ять ключових умов, які вказують на належність даних до категорії великих даних, так звані «п'ять V»:

Volume (об’єм) − дані накопичуються у базі даних і мають настільки великий обсяг, з яким традиційні способи збереження та обробки неефективні. Для роботи із такою інформацією необхідні нові інструменти та переосмислений, зовсім новий підхід.

Velocity (швидкість) − дана характеристика описує швидкість накопичення даних, що постійно зростає та швидкість обробки даних. За два останні роки було зібрано більше 90 відсотків всієї інформації, якою оперує людство. Попит на технології, що виконують обробку великих обсягів даних у реальному часі постійно збільшується.

Variety (різноманітність) − можливість одночасної обробки структурованої та неструктурованої інформації.

Veracity (достовірність) − якість зафіксованих даних може бути різною і впливати на точний аналіз, тому виокремлення достовірних даних є важливою задачею під час збору великих обсягів інформації.

Variability (мінливість) − процеси обробки та управління даними можуть ускладнюватись невідповідністю інформації [2].

У свою чергу big data поділяється на три основні види:

1) структуровані дані;
2) неструктуровані дані;
3) слабоструктуровані (напівструктуровані) дані.

Структуровані дані складаються з чітко визначених типів даних, структура яких робить їх легкими для пошуку, часто такі дані розміщуються у реляційних базах даних. Структуровані дані відповідають табличному формату з відношеннями між рядками і стовпцями. Типовими прикладами структурованих даних є файли Excel та бази даних SQL.

Структуровані дані вважаються найбільш традиційною формою зберігання даних, оскільки перші версії систем управління базами даних могли зберігати, обробляти і отримувати доступ до структурованих даних [5–6].

Слабоструктуровані дані включають в себе обидва типи структурованих та неструктурованих даних, які важко категоризувати, і які мають певні властивості, зокрема теги, які підлягають аналізу.

Неструктуровані дані − це інформація, яка має внутрішню структуру, але не має наперед заданої моделі даних, або не є організованою заздалегідь визначеним чином. По суті, неструктуровані дані − це все інше, що не входить у поняття структурованих даних. Вони можуть бути текстовими або не текстовими, створеними людиною, або машиною, можуть зберігатись у нереляційні бази даних, наприклад такі, як NoSQL.

Типові неструктуровані дані, створені людиною, включають в себе:

1) текстові файли: обробка тексту, електронні таблиці, презентації, електронна пошта, журнали логів;

2) електронна пошта: електронна пошта має деяку внутрішню структуру завдяки своїм метаданим, і іноді її можна назвати напівструктурованою. Однак поле для набору повідомлень неструктуроване, і традиційні інструменти аналітики не можуть його розібрати;

3) соціальні медіа: дані з Facebook, Twitter, LinkedIn;

4) веб-сайти: YouTube, Instagram, сайти для обміну фотографіями;

5) мобільні дані: текстові повідомлення, геолокації;

6) зв’язок: чати, записи телефонних дзвінків, програмне забезпечення для комунікації та співпраці;

7) медіа: цифрові фотографії, аудіо- та відео-файли;

8) додатки для бізнесу: документи MS Office, програми для підвищення продуктивності.

Типові машиногенеровані неструктуровані дані включають в себе:

1) знімки із супутників: дані про погоду, форми рельєфу суходолу, військові пересування;

2) наукові дані: дані про розвідку місць родовищ нафти і газу, дослідження космосу, сейсмічні знімки, атмосферні дані;

3) цифрове спостереження: фото- та відео-спостереження;

4) дані датчиків: дорожнього руху, погоди, океанографічні датчики, радіоактивності.

Можливості зберігання та обробки неструктурованих даних значно зросли за останні роки, з'являється багато нових інструментів і технологій, які здатні зберігати спеціалізовані типи неструктурованих даних. Наприклад, MongoDB оптимізовано для зберігання документів. Apache Giraph оптимізований для зберігання зв'язків між вузлами. Elasticsearch оптимізований для повнотекстового пошуку.

Одним із найпопулярніших у світі та використовуваних інструментів збереження та обробки неструктурованих даних є текстовий процесор MS Word. Даний програмний продукт використовується майже в усіх сферах людської діяльності для роботи із різного роду текстовими документами. Проте можливості текстового процесора не обмежуються лише роботою із текстом, у файли MS Word можна вставляти також ілюстрації, посилання, об'єкти, експрес-блоки, тощо.

Кожна організація, установа, підприємство щодня має обробляти велику кількість документів: приймати та створювати нові, переглядати та редагувати існуючі, мати доступ до документів із електронних архівів. Працівникам кожного разу доводиться вручну переглядати документи, щоб знайти потрібні, орієнтуватись можна лише не назву, та, можливо, дату створення. Даний процес є зовсім не оптимізованим, оскільки працівник може знайти та завантажити необхідний документ, а потім загубити його поміж десятків та сотень інших документів на своєму пристрої.

Саме для оптимізації пошуку необхідних документів можуть використовуватись методи аналізу неструктурованих даних, які містяться, зокрема, у Word-файлах. Аналіз структури документа може допомогти присвоїти йому певні метадані, які коротко та ємко відображають його зміст, що значно

пришвидшить пошук та в цілому прискорить процес роботи із текстовими документами.

Можливість аналізу неструктурованих даних особливо актуальна в контексті великих даних, оскільки значна частина даних в організаціях є неструктурованою або слабоструктуровною. Можливість отримувати цінність з неструктурованих даних є одним з основних рушійних факторів швидкого зростання big data [5-6].

Існує чимало технік та методів аналізу великих даних. McKinsey – міжнародна консалтингова компанія в сфері розв'язування задач, що пов'язані зі стратегічним управлінням, виокремлює 11 різних технік та методик аналізу, які можуть бути застосовані до big data:

1) data mining (глибинний аналіз даних) – методи виявлення у вхідних даних корисних знань для прийняття рішень, які є практичними, проте нетривіальними або раніше невідомими;

2) crowdsourcing – збір інформації із великого, неозначеного кола джерел та окремих особистостей, який відбувається без вступу даних особистостей у трудові стосунки;

3) data fusion and integration (інтеграція даних) – набір технік для інтеграції різного роду даних з великої кількості джерел для проведення глибинного аналізу;

4) machine learning (машинне навчання) – використання моделей, що побудовані на базі статистичного аналізу для отримання прогнозів та класифікації даних;

5) neural networks (штучні нейронні мережі) – евристичні алгоритми пошуку, що використовуються для того, щоб розв'язувати задачі оптимізації і моделювання шляхом варіації, комбінування та випадкового підбору необхідних параметрів, використовуючи певні механізми;

6) pattern recognition (Розпізнавання образів) – ідентифікація певних явищ, предметів, сигналів, процесів, які можна охарактеризувати кінцевим набором певних властивостей та ознак;

7) predictive analytics (Прогнозна аналітика) – методи аналізу даних, виконання яких спрямоване на прогнозування майбутньої поведінки об’єктів та суб’єктів дослідження;

8) simulation (імітаційне моделювання) – дозволяє будувати моделі таким чином, щоб процеси описувались так, як би вони відбувались у реальному світі;

9) spatial analysis (просторовий аналіз) – клас методів, які використовують географічну, геометричну, топологічну інформацію, що отримується із даних;

10) statistical analysis (статистичний аналіз) – метод дослідження, при якому порівнюється контрольна група елементів із набором тестових груп зі зміненими кількома показниками. Допомагає з’ясувати, які зі змін мають позитивний вплив на цільовий показник;

11) visualization of analytical data (візуалізація аналітичних даних) – методи подання інформації за допомогою малюнків, діаграм, анімації та інтерактивних можливостей [7–8].

Використовуючи вищеперераховані техніки можна значно оптимізувати багато процесів у різних сферах: навчання, бізнесу, медицини, науки, можна навіть передбачати та запобігати злочинам або відкривати нові космічні об’єкти.

Технології аналізу неструктурованих даних знаходяться ще тільки на початку свого розвитку, проте мають великий потенціал, адже кількість даних зростає з кожним роком і с кожним роком швидкість зростання кількості даних буде тільки збільшуватись, а за умови правильної обробки та аналізу можуть бути корисні у багатьох сферах людського життя.

1.2 Системи та методи екстракції метаданих документів

Документи надають цінні інформацію, а їх зростаюча популярність в електронній формі призвела до використання різних аналітичних методів. Класифікація документа відноситься до процесу призначення однієї або декількох міток для документа. Основне питання класифікації документів пов'язане з класифікацією контенту, що надає зміст документа. Наприклад, класифікація веб-документів як мистецтв, освіти, науки тощо або класифікація новин за їх темою. Загалом, можна розглядати різні властивості документа в класифікації документів та поєднувати їх, такі як тип документа, автори, посилання на інші документи, вміст тощо. Методи машинного навчання, застосовані до класифікації документів, засновані на загальних методах класифікації. Сьогодні методи та системи екстракції даних працюють лише з текстовим контентом документа [9].

Для аналізу документів, як правило, застосовують загальний аналіз тексту, без аналізу зображень. Аналіз тексту − це приклад машинного навчання у формі обробки природних мов (NLP). Класифікуючи текст, необхідно призначити документу один або декілька класів або категорій, що полегшує управління та сортування [10]. NLP займається побудовою обчислювальних алгоритмів для автоматичного аналізу та представлення людської мови. Системи, що базуються на NLP, давали широкий спектр застосувань, таких як потужна пошукова система Google, а останнім часом голосовий помічник Amazon на ім'я Alexa. NLP також корисний для навчання комп'ютерів виконувати складні завдання, пов'язані з природною мовою, такі як машинний переклад та створення діалогу.

Тривалий час більшість методів, що використовуються для вивчення проблем NLP, використовували неглибокі моделі машинного навчання та трудомісткі функції. Однак, з недавньою популярністю та успіхом вбудовування слів (маломірні, розподілені уявлення), нейронні моделі домоглися найкращих результатів у різних завданнях, пов'язаних з мовою, порівняно з традиційними моделями машинного навчання, такими як SVM або логістична регресія. На рисунку 1.1 зображена робота NLP.

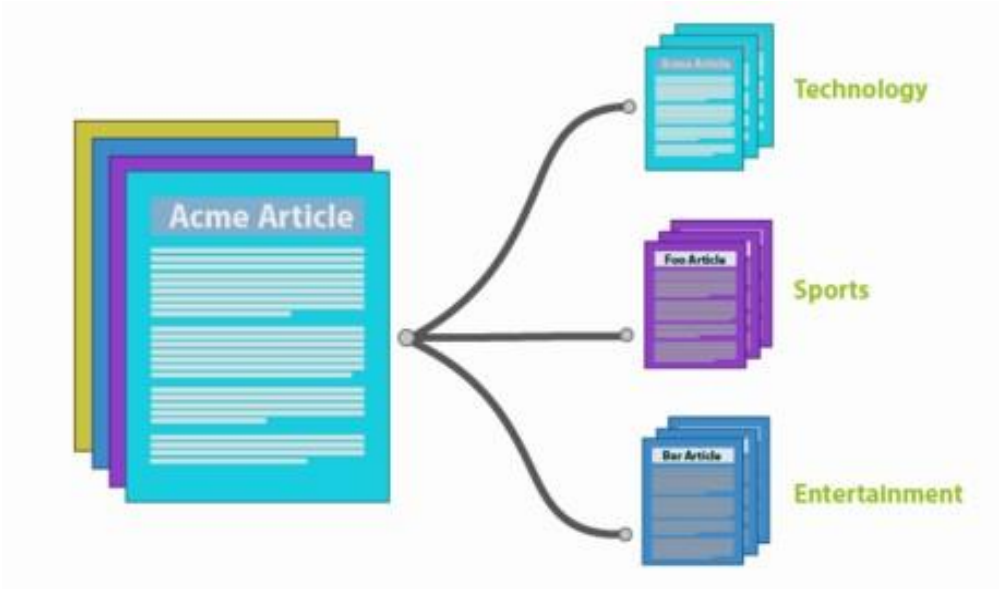

Рисунок 1.1 – Класифікації документів за допомогою NLP

З рисунку 1.1 видно, що NLP може працювати тільки з заздалегідь заданим списком категорій. Інформативність та обсяг метаданих залежить від кількості застосованих попередньо підготовлених словників та навчених моделей аналізу даних. Але їх кількість обмежена кількістю доступних наборів даних, що накопичуються в різних прикладних областях. При цьому під час розробки та оптимізації моделей часто виникають проблеми науково-методологічного характеру, що призводить до не високої ефективності алгоритму.

Monkeylearn − сервіс-фреймворк, який дозволяє створювати власні класифікатори тексту NLP:

1) емоційного забарвлення тексту;
2) теги тексту
3) тема тексту.

Monkeylearn дозволяє використовуються не тільки для аналізу документів, а й для коментарів, відгуків та систем підтримки [11].

Також сервіс дозволяє аналізувати стилі, шрифти тексту. Недоліком цієї системи є відсутність можливості аналізу зображень, що значно звужує можливості для аналізу та класифікації документа.

1.3 Формалізована постановка задачі

Нехай дано вибіркову множину неструктурованих даних {Xk, k=1,N}, кожний елемент якої може мати формат зображення, тексту чи документу Office. Також дано структурований вектор параметрів g, що описує модель екстракції метаданих. Необхідно обрати такі значення параметрів g, які максимізують значення критерію ефективності пошуку інформації в базі неструктурованих даних. Для вирішення поставленої задачі необхідно вирішити ряд завдань :

1) сформувати вибірку неструктурованих даних;
2) розробити інтерфейс доступу до даних;
3) розробити інтерфейс завантаження документів;
4) розробити модель екстрактору метаданих з тексту для пошуку інформації в базі неструктурованих даних;
5) розробити модель екстрактору метаданих з зображень для пошуку інформації в базі неструктурованих даних;
6) виконати програмну реалізацію екстрактора метаданих з неструктурованих даних;
7) виконати валідацію розробленого екстрактора метаданих.

2 ОПИС ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ЕКСТРАКЦІЇ МЕТАДАНИХ

2.1 Модель екстракції метаданих з документа Microsoft Word

Для розробки системи спочатку необхідно розробити її архітектуру. Було обрано сучасну мікросервісну архітектуру, яка дозволить нам будувати легко масштабовані та гнучкі системи, яка має ряд переваг перед старими монолітними архітектурами [12].

Мікросервіси – це сучасний підхід до побудови програмного забезпечення, згідно з яким код програми постачається невеликими, керованими елементами, незалежні один від одного. Невеликий масштаб та відносна ізоляція надають переваги, такі як легше технічне обслуговування, підвищення швидкості розробки, краще масштабування бізнесу тощо [13]. Мікросервісна архітектура зображена на рисунку 2.1.

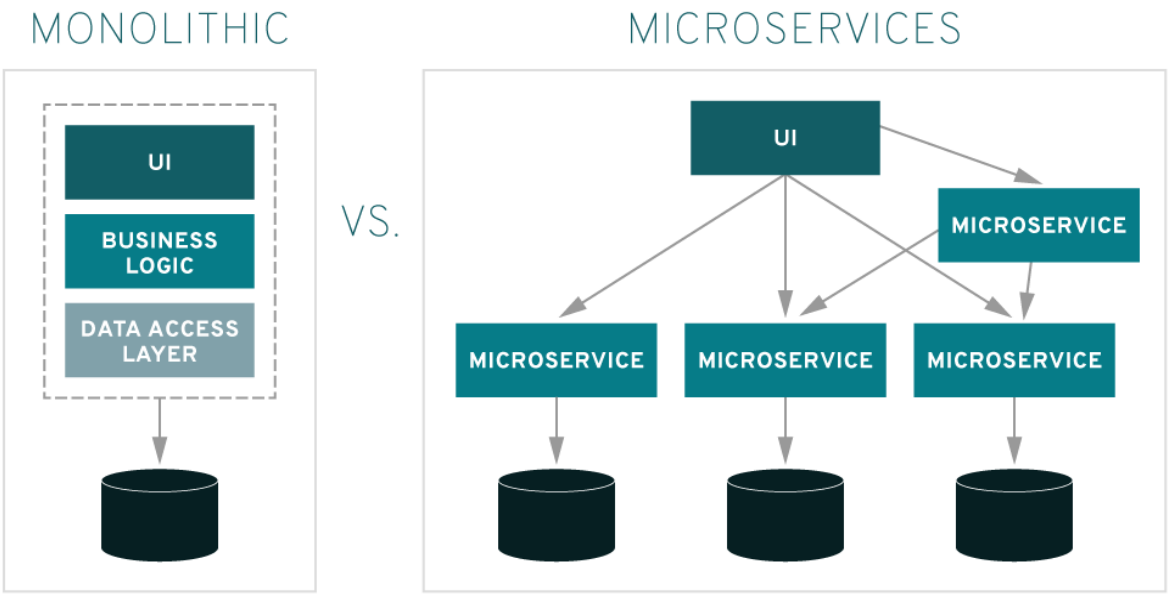

Рисунок 2.1 – Мікросервісна архітектура

З аналізу архітектури зображеної на рисунку 2.1 можна зробити оцінку та висновки про переваги та недоліки використання мікросервісної архітектури.

Мікросервісна архітектура дозволяє вирішити наступні проблеми побудови сучасних систем:

1) монолітність системи – мікросервісна архітектура дозволяє будувати систему модулями, що дозволяє легко додавати нові функції та можливості до системи(наприклад аналіз додавання до аналізу документа нової його частини);

2) масштабованість – мікросервісна архітектура дозволить вирішити проблему з навантаженням на систему(нейронні мережі вимогливі до ресурсів). Систему, яка створена на мікросервісній архітектурі, можна розгорнути на декількох серверах;

3) заміна старих компонентів – мікросервісна архітектура дозволяє змінювати компоненти, вимагаючи зберігати лише старий формат взаємодії з іншими сервісами;

4) зручність розробки – мікросервісна архітектура дозволяє розробляти систему на різних мовах програмування, що дозволяє вирішувати кожну задачу своєю мовою програмування, яка підходить для її вирішення;

5) низька відмово стійкість – при відмові або одного з мікросервісів система буде працювати.

Мікросервісна архітектура має недоліки, які варто врахувати при побудові системи:

1) накладні розходи на комунікацію між мікросервісами – комунікація між сервісами по протоколу http;

2) дублювання коду – буде дублюватись код на мікросервісах для підключення до бази даних;

3) складна відкладка комунікації між мікросервісами.

Запрована модель, яка дозволяє використовувати переваги мікросервісної архітектури та компенсувати її недоліки. Архітектура системи аналізу документів зображена на рисунку 2.2.

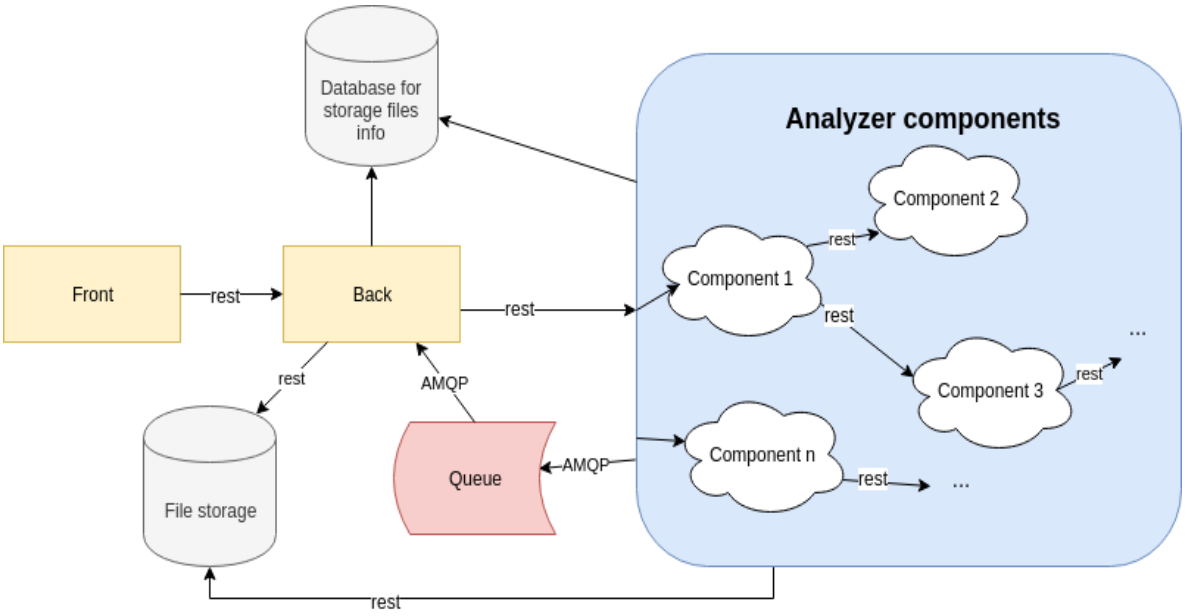

Рисунок 2.2 – Діаграма компонентів системи аналізу документів

Мікросервіс «Front» взаємодіє з користувачем і надає йому веб інтерфейс. В цьому інтерфейсі користувачі можуть авторизуватися, завантажувати документи та виконувати пошук по ним. Виділення веб частини в окремий мікросервіс дозволяє вирішити проблему монолітності серверної частини та відображення.

Мікросервіс «File storage» забезпечує доступ до файлів по протоколу http та відповідає за їх зберігання. Використання http створення для доступу до файлів дозволяє вирішити проблему – мати зручний доступ до файлу з будь-якого мікросервісну та мати зручний доступ користувачу до файлу. Всі файли зберігаються на диску без використання баз даних. Це було використано для простоти доступу до даних і зменшення на використання дискового простору на службову інформацію, яка буде створюватись при додаванні файлу в базу даних.

Для зберігання даних нами було обрано наступний підхід – використовувати дві різні системи керування базами даних, які застосовуються для двох різних задач:

1) задача контролю доступу до даних – доступ користувачів з використанням певної ієрархії(за необхідністю) до своїх даних;

2) задача зберігання інформації про файл без необхідності розробляти повну схему бази даних. Ця задача є актуальною оскільки важко розробити систему модель, яка може описати всі дані нам необхідно зберігати. Для цього було обрано нереляційну базу даних.

«Інтелектуальною» частиною системи є набір мікро-сервісів «Analyzer components». Ми можемо будувати достатньо складні та гнучкі системи аналізу використовуючи модель запропоновану на діаграмі. Кожний сервіс-аналізатор може використовувати інші сервіси-аналізатори. Система аналізує документи, отже необхідно створити сервіс «Аналізатор документу». В кожному документі є зображення, текст, таблиці, стилі. Система інтелектуальних сервісів буде мати архітектуру зображену на рисунку 2.3.

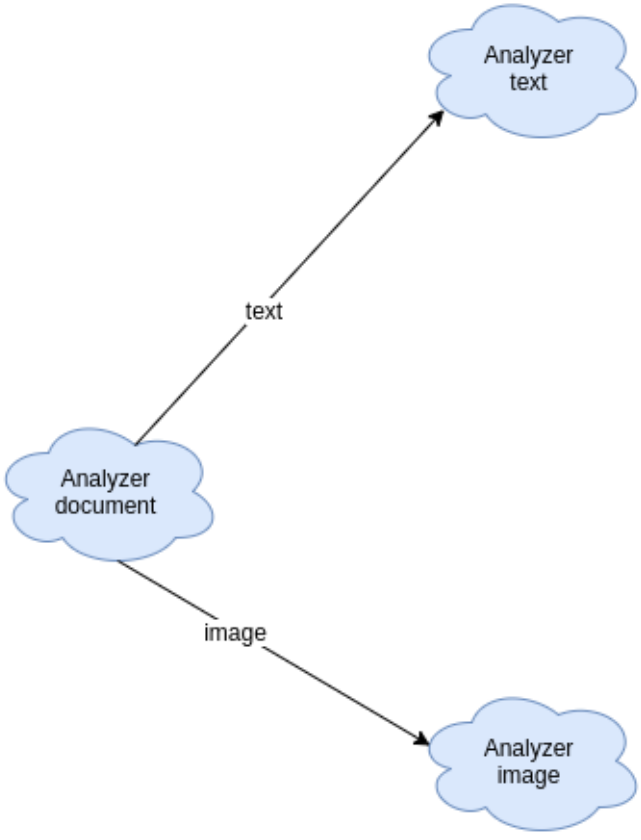

Рисунок 2.3 –Діаграма «інтелектуальної» системи

Розроблений підхід дозволяє повторно використовувати мікро-сервіси для аналізу нових об’єктів. Наприклад, необхідно аналізувати презентації, а саме в них проаналізувати текст та картинки. Ми можемо не переписувати логіку, а використовувати логіку, яку ми розробили для документів. Приклад системи аналізу, яку можна побудувати зображений на рисунку 2.4.

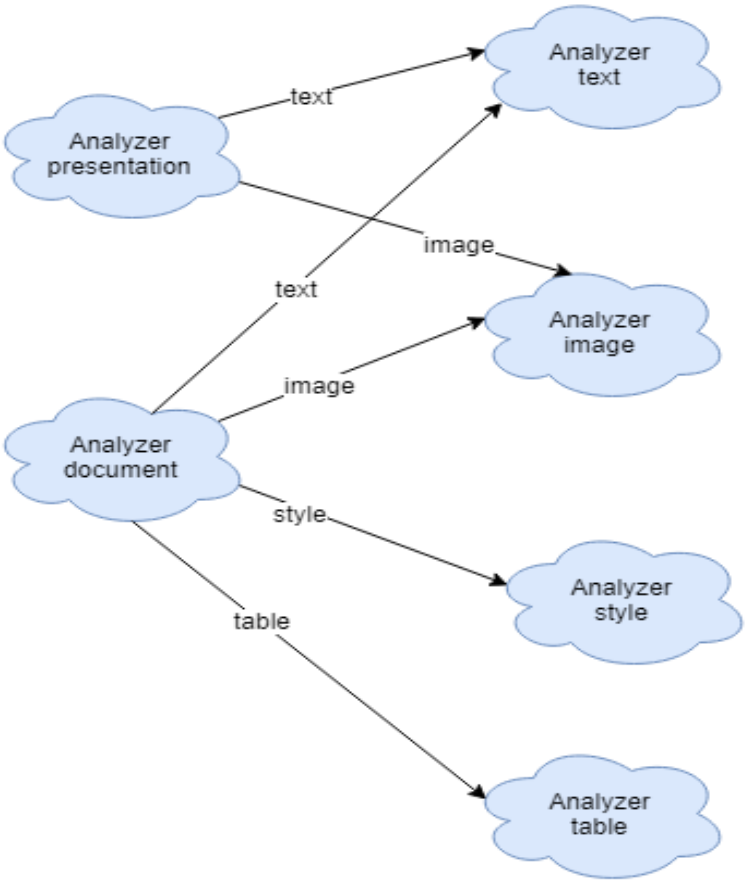

Рисунок 2.4 – Приклад діаграми «інтелектуальної» системи з аналізом презентацій

З рисунку 2.4 можно побачити, що сервіси-аналізатори взаємодіють між собою за допомогою протоколу http. Кожен мікросервіс аналізу має єдиний API – для виконання аналізу взаємодії – *POST /analyzer*. Результати обробки та ідентифікатор документа кладуться в чергу, яка має ім’я таке ж саме як і ім’я

модуля. На рисунку 2.5 зображена схема роботи відправки результатів аналізу та класифікації в чергу.

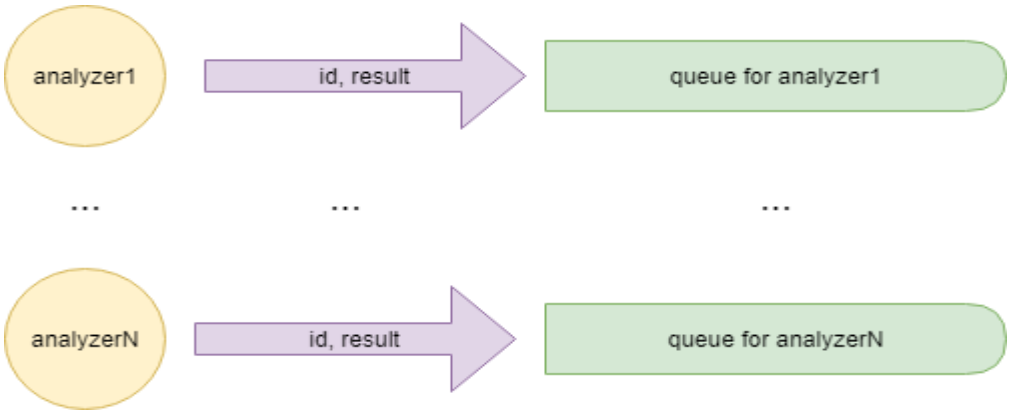

Рисунок 2.5 – Діаграма відправки результатів в чергу

«Інтелектуальна» частина системи може не давати нам результат обробки в реальному часі – час аналізу може бути достатньо довгим, щоб чекати результат. З аналізу рисунку 2.5 видно, як мікросервіс «Queue» вирішує цю проблему. Система не буде чекати поки буде повністю проаналізовано документ – після аналізу однієї частини (наприклад одного зображення) результат буде відправлено в чергу, що дозволяє показувати процесс обробки в реальному часі та не чекати поки буде повністю оброблений документ. Мікросервіс «Queue» використовує протокол AMQP (Advanced Message Queuing Protocol). Протокол дозволяє обмінюватись повідомленнями різним підсистемам. Системи можуть надсилати повідомлення в чергу, а також отримувати оновлення з цих черг. Цей протокол дозволяє значно швидше передавати повідомленні ніж http. Також черга дозволяє забезпечувати відмовостійкість[14].

Основною компонентою є мікросервіс «Back», яка виконує функцію rest-сервера. REST – це архітектурний підхід, який використовує прості HTTP-запити для комунікації замість складніших варіантів, таких як CORBA, COM +, RPC, SOAP. Використання REST означає, що виклики будуть описуватися на основі повідомлень і залежать від стандарту HTTP для опису цих повідомлень.

Використання протоколу HTTP означає, що REST – це простий механізм запиту / відповіді[15]. Кожен запит повертає відповідь(рис. 2.6).

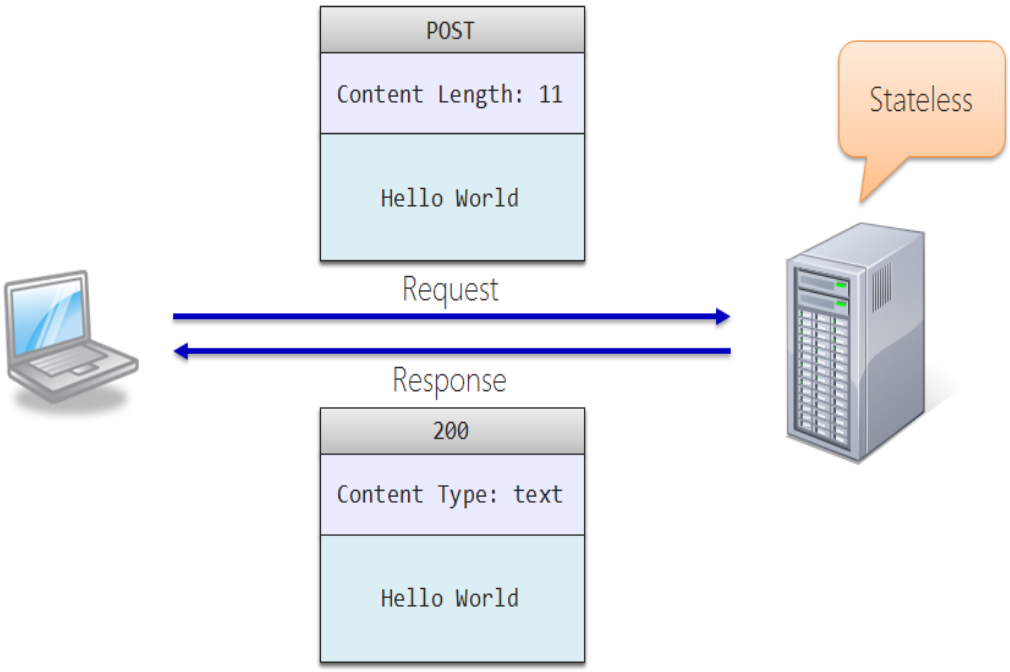

Рисунок 2.6 – Діаграма роботи REST-сервісу

«Back» надає API додавання документу та пошуку по ним. Також він виконує іншу важливу функцію – читає з відповідних черг результати обробки та заносить їх в базу.

2.2 Модель і метод екстракції метаданих з зображень

Класифікація та аналіз зображення – це визначення до якого класу або класів відноситься зображення. Класифікація зображень відноситься до комп'ютерному зорі, який може класифікувати зображення відповідно до його візуального змісту. Наприклад, алгоритм класифікації зображень може бути розроблений таким чином, щоб визначати, чи містить зображення фігуру людини чи ні. У зображення існують потенційно n кількість класів, в яких дане зображення можна класифікувати. Перевірка та класифікація зображень вручну може бути неможливим завданням, особливо коли необхідно проаналізувати

великий масив зображень (скажімо, 10 000), і тому буде дуже корисно, якщо можно автоматизувати весь цей процес за допомогою комп'ютерного зору[16].

Типи класифікації зображень:

1) двійкова класифікація – допомагає визначити наявність об’єкту на зображенні;

2) багато-класова класифікація – допомагає визначити, які об’єкти є на зображенні.

На рисунку 2.8 показана діаграма роботи багатакласової класифікації зображення.

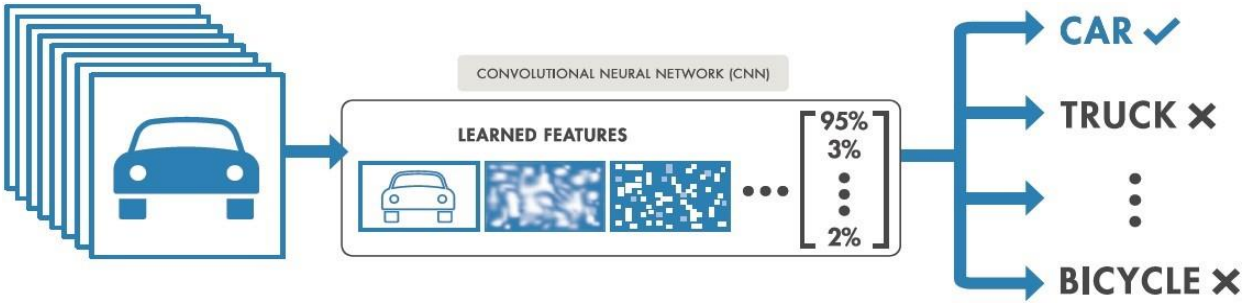

Рисунок 2.8 – Схема екстракції мета-даних зображення на основі класифікатора

З рисунку 2.8 видно, що для аналізу та отримання метаданих зображення з документа Word необхідна багатокласова класифікація.

Існує велика кількість архітектур і мікроархітектур нейронних мереж для розпізнавання зображень, серед яких найбільшого поширення набули VGG-16, VGG-19, ResNet-50, Google Net та інші. Виходи моделі формуватимуть набір тих класів, які були присутні у навчальному наборі. Серед доступних в інтернеті набули поширення такі навчальні набори зображень як ImageNet, LSUN, MS COCO, COIL100, Google’s Open Images, Indoor Scene Recognition та інші. Для аналізу та класифікації було використано алгоритм машинного навчання. Було використано нейронну мережу VGG16. VGG16 – модель згорткової нейронної

мережі(рис. 2.6). Вона використовується для розпізнавання об’єктів на зображеннях. Модель досягає точності класифікації зображення на тестовій вибірці 92.7%[17].

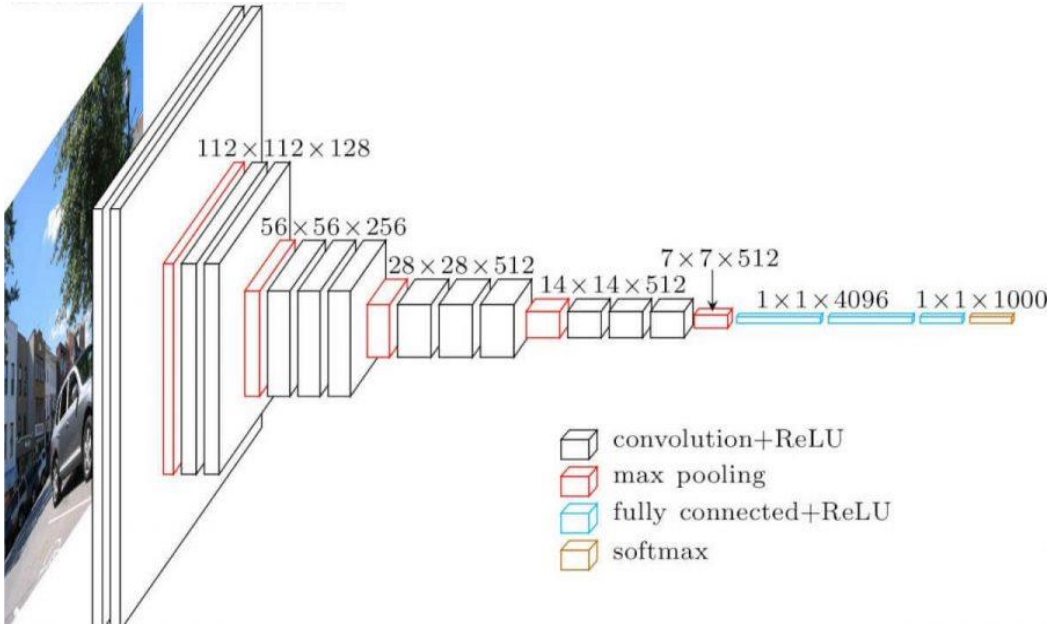

Рисунок 2.9 – Структура нейронної мережі VGG16

Згорткові нейронні мережі забезпечують часткову стійкість до змін масштабу, зсувів, поворотам, зміні ракурсу і іншим спотворень. Згорткові нейронні мережі об'єднують три архітектурних ідеї, для забезпечення інваріантності до зміни масштабу, повороту зрушення і просторовим спотворень.

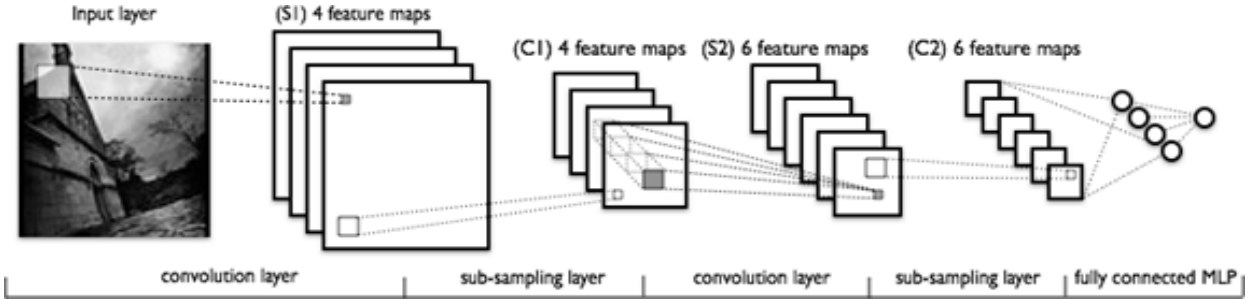

Рисунок 2.10 – Приклад найпростішої згорткової нейронної мережі

Мережа VGG має два недоліки:

1) повільна швидкість навчання(навчалася протягом декількох тижнів при використанні відеокарт NVIDIA TITAN BLACK);

2) архітектура мережі займає багато пам’яті(533 мб), що може стати причиною проблем зі швидкістю роботи додатку, який використовує модель[18].

Ці недоліки не є критичними для вирішення поставленої нами задачі, оскільки час навчання моделі та її розмір не заважає розпізнавати документи Word.

Датасетом для навчання моделі було обрано ImageNet. ImageNet – це набір даних зображень, організований відповідно до ієрархії WordNet. ImageNet містить 21841 класів та 14197122 зображння. Кожна змістовна концепція в WordNet, можливо описувана кількома словами або словосполученнями, називається «набором синонімів» або «син сетом». У WordNet існує понад 100000 синсетів, більшість з них є іменниками (80 000+).

Проект ImageNet створений через зростання популярності в області комп’ютерного-зору – потребою в більшій кількості даних. З часу зародження цифрової епохи та наявності обміну даними в масштабах веб-сайтів, дослідники в цих галузях наполегливо працюють над розробкою все більш складних алгоритмів для індексації, пошуку, організації та анотації мультимедійних даних[19].

2.3 Модель і метод екстракції метаданих з текстових документів

Найбільш поширений спосіб для екстракції метаданих з тексту є використання NLP. NLP займається побудовою обчислювальних алгоритмів для автоматичного аналізу та представлення людської мови(рис 2.11). Системи, що базуються на NLP, давали широкий спектр застосувань, таких як потужна пошукова система Google, а останнім часом голосовий помічник Amazon на ім'я Alexa. NLP також корисний для навчання комп'ютерів виконувати складні

завдання, пов'язані з природною мовою, такі як машинний переклад та створення діалогу[20].

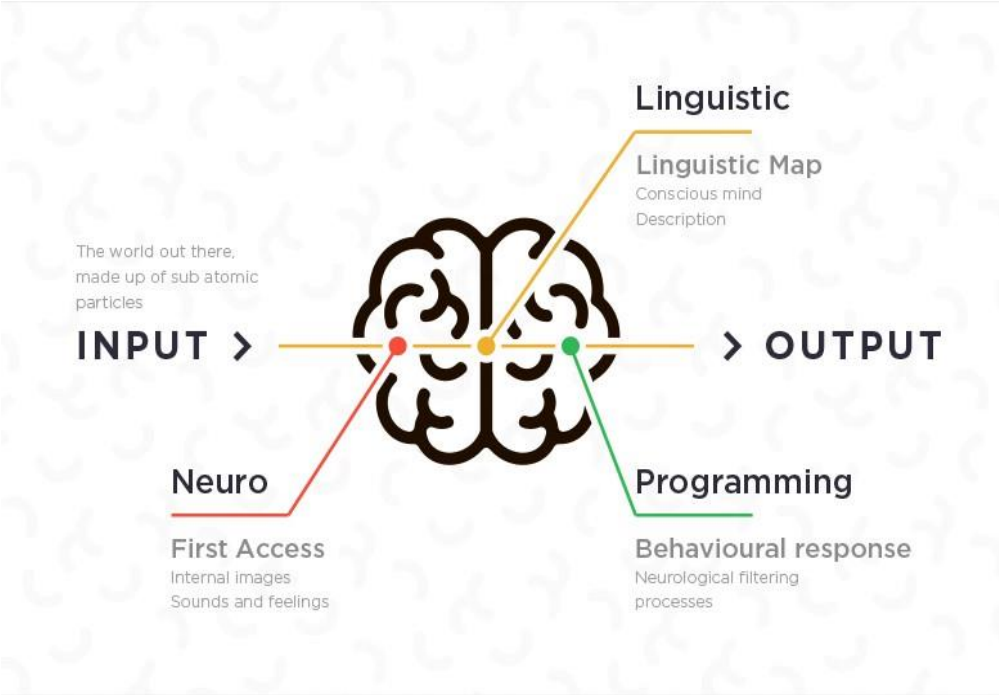

Рисунок 2.11 – Структура процесу обробки в фреймворках NLP

Інформативність та обсяг метаданих залежить від кількості застосованих попередньо підготовлених словників та навчених моделей аналізу даних. Але їх кількість обмежена кількістю доступних наборів даних, що накопичуються в різних прикладних областях. При цьому під час розробки та оптимізації моделей часто виникають проблеми науково-методологічного характеру, що призводить до не високої ефективності NLP .

Розглянемо інший алгоритм для аналізу та кластеризації тексту. TF / IDF – статистичний показник, який використовується переважно для оцінки важливості конкретного слова у контексті всього документа, що знаходиться в обстеженні базу. TF визначає частоту вхідних термінів, а IDF розширюється, як зворотна (інвертована) частота документа[21]. У відповідності з відношенням TF / IDF вказує конкретний слова прямо залежно від кількості його використання у

конкретному тексті та прямо залежно від кількості використання даного слова в декількох інших документах. TF або частота словника – це відношення кількості вхідних даних конкретного терміналу до сумарного набору шару в досліджуваному тексті (документі). Цей показник відображає важливість слова в рамках конкретному тексті. IDF або зворотна (інвертована) частота документа – це інверсія частот, за якою визначене слово фігурує в текстах. Завдяки даному показнику можна знизити важливість найбільш широко використовуваних слів. Щодо кожного терміна в рамках конкретної бази текстових файлів передбачено одне єдине значення IDF. Покажчик TF / IDF буде вищим, якщо конкретне слово з великим частотою використовується в конкретному тексті, а рідко – в інших документах. Показаний TF / IDF використовується для аналізу текстового контента у великих потоках даних. Так, для даного розв'язання прив'язують пошукові алгоритми, визначають релевантність конкретних сторінок Так само даний статистичний показник може визначити близькі різні документи (тексти) інших, що може бути використане в їх групі (кластеризації) [22].

Цей алгоритм підходить для отримання метаданих с документи оскільки він має кілька переваг:

1) не треба формувати датасет та попередньо навчати моделі для класифікації тексту;

2) визначає не тему документа, а визначає його слова, що краще підходить для пошуку.

2.4 Критерії оцінки ефективності пошуку даних на основі метаданих

Для оцінки роботи системи для отримання метаданих за документа взято метрики релевантності (приналежності) документа запиту (рубриці).

Метрики для не впорядкованої більшості документів засновані на бінарної класифікації документів «релевантний / НЕ релевантним» по відношенню до вибраного запиту [23]. Дані метрики грунтуються на матриці класифікації (див. табл.. 2.1):

Таблиця 2.1 Основні категорії документів відповіді системи

Результат	релевантні	не релевантні
знайдено системою	a	b
не знайдено системою	c	d

1) a - кількість документів, знайдених системою і релевантних з точки зору експертів;

2) b - кількість документів, знайдених системою, але не релевантних з

3) точки зору експертів;

4) c - кількість релевантних документів, що не знайдені системою;

5) d - кількість не релевантних документів, що не знайдені системою.

Для розрахунку релевантності необхідно розрахувати:

1) повнота обчислюється як відношення знайдених релевантних документів до загальної кількості релевантних документів

$$r = \frac{a}{a + c}$$

Повнота характеризує здатність системи знаходити потрібні користувачеві документи, але не враховує кількість не релевантних документів, які видаються користувачеві.

2) точність обчислюється як відношення знайдених релевантних документів до загальної кількості знайдених документів;

$$p = \frac{a}{a + b}$$

Точність характеризує здатність системи видавати в списку результатів тільки релевантні документи;

3) акуратність обчислюється як відношення правильно прийнятих системою рішень до загальної кількості рішень

$$accuracy = (a + d)/(a + b + c + d)$$

Оскільки ми припускаємо, що система приймає рішення про приналежність до даної категорії для кожного документа колекції. Таким чином знаменник не залежить від даної категорії. При обчисленні оцінки в якості знаменника використовувалося загальне число документів які оцінювалися хоча б для однієї категорії;

4) помилка обчислюється як відношення неправильно прийнятих системою рішень до загальної кількості рішень

$$error = (b + c)/(a + b + c + d).$$

Одним з важливих питань побудови метрик в текстовому пошуку є метод усереднення результатів[24]. Необхідно знайти загальну кількість документів, що відносяться до категорій (див. таблиця 2.1) і вже на їх основі обчислити шукану метрику.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Опис програмної реалізації

3.1.1 Реалізація бази даних

В якості системи керування базами даних було використано Elasticsearch. Elasticsearch – це пошуковий движок, який може використовуватись як база даних[25]. Elasticsearch - це розподілене, відкрите джерело пошуку та аналізу для всіх типів даних, включаючи текстові, числові, геопросторові, структуровані та неструктуровані. Elasticsearch пошук побудований на Apache Lucene. Написаний на Java. Переваги Elasticsearch:

1) доступ до бази даних через Rest API – зручний доступ до бази даних через http протокол;

2) масштабування. Дані в Elasticsearch можуть зберігатись розподілено – на декількох машинах. Якщо одна машина більше не буде доступною, Elasticsearch збереже все документи, якщо була налаштована реплікація індексів;

3) NoSQL – дозволяє не розробляти повну схему даних та зручно контролювати дані з коду;

4) інтеграція з візуалізатором Kibana. Кібана забезпечує візуалізацію даних з Elasticsearch у режимі реального часу, будувати графіки та виконувати її групування;

5) fulltext пошук – пошук по всьому тексту документа, не вказуючи як шукати. Elasticsearch – це також пошукова платформа в реальному часі, що означає затримку від часу індексації документа до його пошуку, дуже короткою – зазвичай це одна секунда. Як результат, Elasticsearch добре підходить для використання, залежних від часу систем, таких як аналітика безпеки та моніторинг інфраструктури.

Дані в Elasticsearch зберігаються в документах. Документи зберігаються в індексах(див. табл.. 3.1).

Таблиця 3.1 – Відповідність структури реляційних баз даних до бази даних

Реляційна база даних	**Elasticsearch**
таблиця	індекс
запис в таблиці	документ

Elasticsearch має динамічну схему даних в індексі, тобто нам не треба розробляти модель даних.

Загальна структура документа для збереження інформації про документ користувача (див. таблиця 3.2):

Таблиця 3.2 – Загальна структура документа

Поле	**Тип**	**Опис**
id	string	id документа
name	string	ім’я документа
url	string	посилання на документ
about	string	опис документа, який ввів користувач
image_tags	array	результати аналізу зображень документа
text_tags	array	результати аналізу тексту документа
comments	array	список коментарів доданих до документа, інформація про

Поле	**Тип**	**Опис**
		коментар складається з імені користувача, який залишив коментар та самого коментаря
count_charts	number	кількість діаграм в документі
count_comments	number	кількість коментарів в документі
count_hyperlinks	number	кількість посилань в документі
count_paragraph	number	кількість параграфів в документі
count_pictures	number	кількість зображень в документі
count_tables	number	кількість таблиць в документі
has_charts	boolean	наявність діаграм в документі
has_comments	boolean	наявність діаграм в документі
has_hyperlinks	boolean	наявність посилань в документі
has_pictures	boolean	наявність зображень в документі
has_tables	boolean	наявність таблиць в документі
hyperlinks	array	список посилань в документі
time	date	час додавання документа

Індекс для зберігання інформації про документ ─ documents.

3.1.2 Реалізація системи

Для реалізації компонентів «Back» та «File storage» було обрано NodeJS. NodeJS – це кросплатформна платформа, побудовано на движку V8, для виконання JavaScript, яка працює на серверах[26]. NodeJS використовує npm[27]. Основні переваги використання NodeJS:

1) швидкість роботи – може працювати значно швидше, ніж сервери, які написані на таких мови програмування: Java, Python, Ruby, що досягається завдяки неблокуючій архітектурі;

2) швидкість розробки – дозволяє швидко розробляти невеликі серверні додатки, що є зручним при написанні мікросервісної системи;

3) наявність великої кількості бібліотек;

4) зручна робота з об’єктами – дозволяє не створювати класи, а просто працювати з об’єктами;

5) дозволяє розробляти кросплатформенні додатки;

6) простота розгортання та встановлення;

7) JavaScript – дозволяє писати асинхронний код, що дає можливість не створювати затримок для виконання складних операцій.

В «Back» та «File storage» використані наступні фреймворки:

1) @hapi/hapi, express, expressfileupload – для створення rest api сервера;

2) amqplib – для роботи з rabbitmq;

3) elasticsearch для роботи з elasticsearch.

Була взята готова реалізація компоненти, яка виконує функції сервісу «Queue» – RabbitMQ. RabbitMQ дозволяє взаємодіяти різним компонентам за допомогою протокола AMQP. Також важливу роль у виборі цього рішення зіграла – легкість та простота налаштування. RabbitMQ кросплатформена система. Його основна ціль – надсилати та отримувати повідомлення. В RabbitMQ є три ролі:

1) Producer – відправляє повідомлення;

2) Queue – черга де зберігаються повідомлення. Черга немає обмежень на кількість повідомлень. Будь-яка кількість программ може кидати в неї повідомлення та читати з неї;

3) Consumer – читає повідомлення.

Producer, Consumer та Queue не обов'язково повинні знаходитися на одній фізичні машині. Також RabbitMQ дозволяє персистентно зберігати свої черги, що дозволяє не втратити дані. RabbitMQ має багато інструментів для моніторингу, налаштування черг. Також RabbitMQ маштабується, що дозволяє будувати складні та навантажені системи. RabbitMQ проста система для розгортання та підтримки та не вимоглива до ресурсів сервера[28].

Для реалізації «Front» було взято модифікацію Kibana від компанії Amazon – OpenDistro Kibana. Kibana – інструмент для візуалізації даних з відкритим вихідним кодом[29]. Створений на NodeJS. Kibana має набір інструментів:

1) Discover – дозволяє виконувати пошук по документам з заданням фільтрів та часового проміжку;

2) DevTools – дозволяє виконувати ручні маніпуляції над документами за допомогою відповідної мови запитів: видаляти, додавати, редагувати дані в базі даних;

3) Visualization – дозволяє переглядати статистику, створювати діаграми та агрегувати дані;

4) Dashboard – дозволяє групувати візуалізації, запити, графіки та таблиці на одному полотні;

5) Management – дозволяє виконувати налаштування для Kibana .

Було обрано OpenDistro Kibana оскільки компанія Amazon додала в Kibana функціонал, який доступний тільки в платній версії Kibana(Xpack):

1) авторизація та аутентифікація ;

2) підтримка ролей – дозволяє контролювати доступ користувачів до даних по групам;

3) підтримка роботи з elasticsearch з авторизацією;

4) створювати нотифікації і повідомлення по заданим умовам.

Kibana підтримує створення плагінів[30]. Оскільки система потребує можливості завантаження документів, необхідно розробити відповідний плагін, який дозволить користувачам виконати завантаження. Для розроблення плагінів на Kibana можно використовувати будь-який фреймворк для написання браузерної частини додатку. Було обрано фреймворк ReactJS. Веб-компоненти React дозволяють брати фрагменти HTML-коду, оформляти їх у вигляді самостійних компонентів, і, замість того, щоб додавати на сторінку ці фрагменти, включати до складу сторінок щось на зразок особливих HTML-тегів, що вказують на них[31]. У нашому випадку, замість того, щоб додавати на сторінку сорок рядків HTML-розмітки, досить включити в її склад компонент, що містить цю розмітку.ReactJS підтримується компанією Facebook. Його вибір був зумовлений:

1) React дає вам мову шаблонів і деякі callback-функції для відтворення HTML;

2) поєднання JavaScript і HTML в JSX робить компоненти простими для розуміння;

3) розробка UI на основі окремих компонентів;

4) React-компоненти можна повторно перевикористовувати;

5) можна рендерити React на сервері.

Для зручного конфігурування кожної компоненти системи було використано Docker. Docker – це платформа, яка призначена для розробки, розгортання і запуску додатків в контейнерах[32]. Використання Docker вирішує декілька проблем:

1) ізолювання ресурсів – дозволяє запобігти втручання процесів роботи однієї компоненти в процеси роботи іншої, наприклад блокування віртуальної машини або ізолювання диску;

2) масштабування – в Docker є образи та контейнери. Образ – це опис(шаблон) для додатку. Контейнер – це запущений екземпляр образу додатку ;

3) ізолювання середовища – дозволяє використовувати різні версії платформ та самі платформи на різних контейнера;

4) зручного та автоматизованого розгортання системи – достатньо виконати команди git pull та docker-compose build для розортання останьої версії на сервері;

5) відтворення середовища для запуску сервісу: операційна система, версія бібліотек.

Для розробки сервісу для аналізу документів було обрано об’єктно орієнтовану мову програмування Kotlin. Kotlin має всі переваги мови програмування Java оскільки компілюється в Java-байткод. Java розробляється та підтримується компанією Oracle. Java має C-подібний синтаксис[33]. Основні переваги Java:

1) простота коду та його ефективність;

2) об’єктно-орієнтовано мова програмування – дозволяє розробляти великі системи;

3) garbage-collector;

4) велика кількість бібліотек.

Вибір Kotlin зумовлений більш новим синтаксисом та можливістю контролювати NullPointerException[34]. Основною причиною вибору Java/Kotlin стала бібліотека Apache Poi. Apache POI представляє єдиний API, який використовується для роботи з файлами MS Office у Java-додатках. Apache POI включає в себе класи та методи для читання та запису інформації у документах MS Office[35].

3.1.3 Реалізація екстракції метаданих

Для реалізації екстракції метаданих було обрано Python. Python – мова програмування орієнтовано на підвищення продуктивності та читаємості коду[36]. Python має особливості, які стали причинами вибором його, як основного інструменту для розробки сервісів аналізаторів:

1) зрозумілість синтаксису;
2) набір бібліотек для машинного навчання (Tensorflow, Keras) ;
3) бібліотека для роботи з масивами та для виконання математичних операцій – numpy;
4) менеджер для встановлення додаткових бібліотек для спрощення розробки – pip;
5) інтерпретована мова програмування – дозволяє швидше розробляти програми в порівнянні з мовами програмування, які компілюються, що допомагає економити час;
6) бібліотека для розробки веб сервісів – Flask, яка дозволяє швидко розробляти невеликі веб сервіси, які працюють на протоколі http і виконують функцію rest api сервіса.

Мікросервіс обробки зображень виконує функцію аналізу та класифікації зображень, які на нього буду надсилати мікросервіс аналізу документів Word. Ця інформація є необхідною для більш детального аналізу змісту документа, оскільки на зображені може бути інформація, яка може бути корисною для здійснення пошуку по документам в базі даних. Його основна задача визначити та класифікувати отримані зображення та надіслати результати класифікації на RabbitMQ.

Для запуску аналізу та класифікації зображення з документа було створено http-сервер на фреймворке Flask. Flask дозволяє швидко розробляти прості веб сервери на мові програмування Python. Для запуску мікросервіса було взято вільний порт 1490. Алгоритм роботи мікросервісу класифікації зображень з документа зображений на рисунку 3.1.

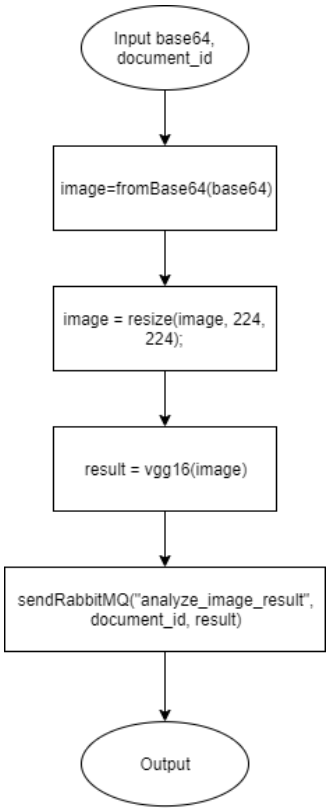

Рисунок 3.1 – Блок-схема роботи алгоритму аналізу зображення з документа

Мікросервіс аналізу зображень отримує на вхід зображення в форматі base64. Формат base64 зручно використовувати для передачі даних між компонентами через протокол http. Після отримання base64 тексту конвертуємо його в зображення. Для того щоб використовувати нейронну мережу VGG16 необхідно виконати певні перетворення над зображенням. Необхідно змінити розмір зображення на 214x214, оскільки на вхідний шар нейронної мережі необхідно подавати зображення такого розміру. Було взято готову реалізацію VGG16 навчену на датасеті ImageNet моделі з бібліотеки Keras. Після отримання результату відправляємо його в чергу «analyze_image_result» в RabbitMQ.

Мікросервіс обробки тексту виконує функцію аналізу тексту документа, які на нього буду надсилати мікросервіс аналізу документів Word. Ця інформація є необхідною для більш детального аналізу змісту документа. Його основна задача визначити, виділити ключові слова з тексту та надіслати оброблені результати на RabbitMQ. Для запуску аналізу було створено http-сервер на фреймворке Flask. Для запуску мікросервіса було взято вільний порт 1491.

Мікросервіс аналізу отримує отримує на вхід текст. Для аналалізу документа Після отримання результату відправляємо його в чергу «analyze_text_result» в RabbitMQ.

3.2 Тестування та оцінювання ефективності системи

Для тестування системи було завантажено файл документ на тему “Футбол”. Для завантаження документа необхідно виконати авторизацію. Сторінка авторизації зображена на рисунку 3.2.

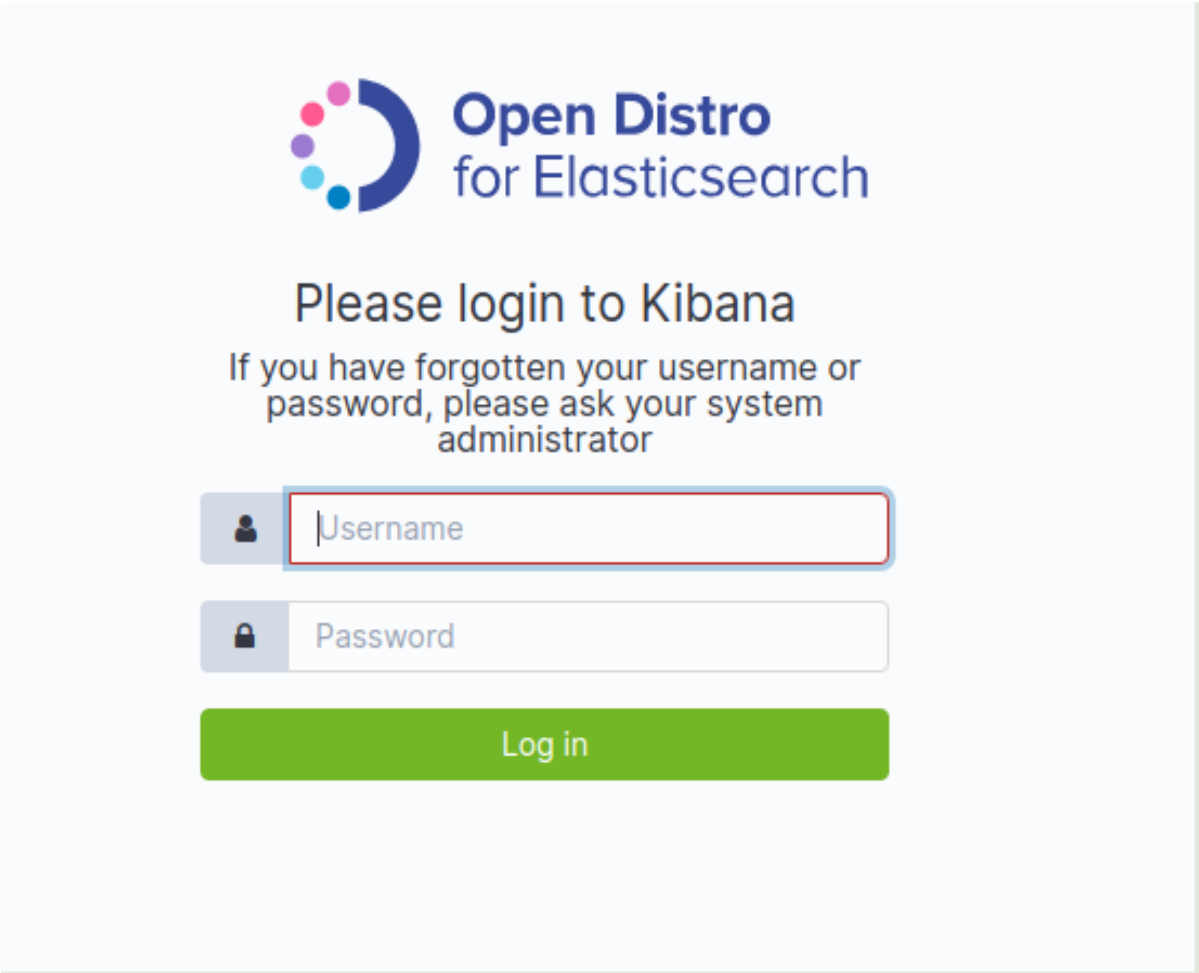

Рисунок 3.2 – Зовнішній вигляд сторінки авторизації

Після натискання кнопки «Log in» з’явиться сторінка для завантаження документа. Далі необхідно виконати завантаження документа в систему для

аналізу та проставлення метаданих. Сторінка завантаження документа зображення на рисунку 3.3.

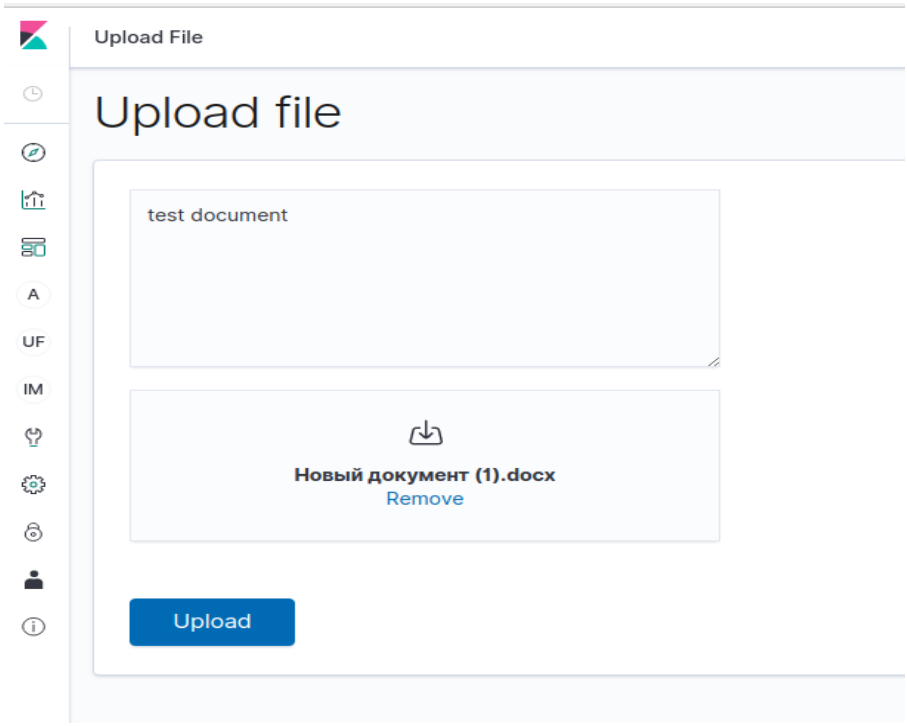

Рисунок 3.3 – Зовнішній вигляд сторінки завантаження документа

Після натискання кнопки «Upload» можно виконати перехід на сторінку пошуку. Для перевірки роботи системи виконали пошук. Сторінка пошуку зображена на рисунку 3.4.

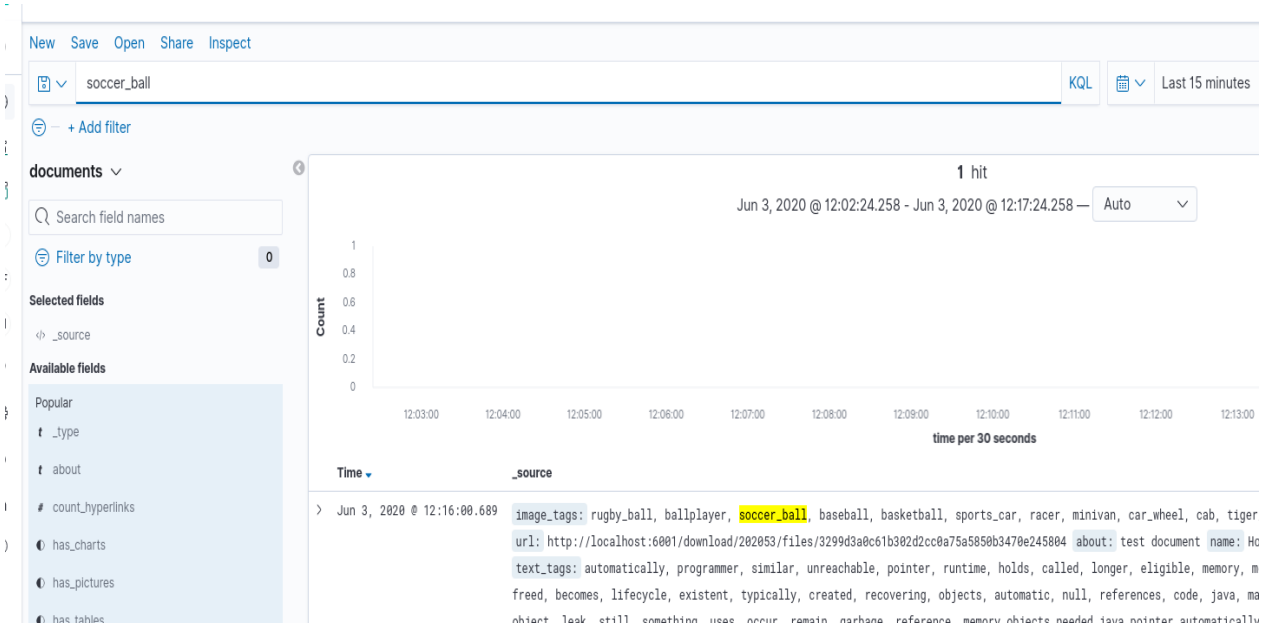

Рисунок 3.4 – Зовнішній вигляд сторінки пошуку документа

З рисунку 3.4 система змогла виконати пошук. Екстракція метаданих дозволила правильно виконати пошук.

Було пораховано критерії оцінки ефективності пошуку даних на основі метаданих. Для розрахунку було згенеровано по 100 документів на 10 тематик та завантажені до системи. Після виконання пошуку було отримано наступні результати(див. табл. 3.1).

Таблиця 3.1 Матриця помилок

Результати пошуку	**релевантні**	**не релевантні**
знайдено системою	943	117
не знайдено системою	57	883

Пораховано критерії:

1) повнота – 0.943;
2) точність – 0.889;
3) акуратність – 0.913;
4) помилка – 0.087.

По даним результатам пошуку документів на основі метаданих можна вважати ефективним.

ВИСНОВКИ

Проаналізовано та розроблено модель для екстракції метаданих з неструктурованих документів. Метадані отримуються шляхом застосування попередньо навчених моделей аналізу тексту та зображень. Було розроблено програмну реалізацію, яка дозволяє завантажувати документи на обробку та здійснювати пошук серед оброблених документів. Отримані результати свідчать про придатність розробленої інформаційної системи для практичного використання.

Під час виконання роботи було вирішено ряд завдань :

Вирішено ряд завдань :

1) сформовано вибірку неструктурованих даних;
2) розроблено інтерфейс доступу до даних;
3) розроблено інтерфейс завантаження документів;
4) розроблено модель екстрактору метаданих з тексту для пошуку інформації в базі неструктурованих даних;
5) розроблено модель екстрактору метаданих з зображень для пошуку інформації в базі неструктурованих даних;
6) виконано програмну реалізацію екстрактора метаданих з неструктурованих даних;
7) виконано валідацію розробленого екстрактора метаданих.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Поняття інформації [Електронний ресурс] – Режим доступу до ресурсу: https://dl.sumdu.edu.ua/semipub/htz/399573/index.html?clear=true.

2. Вайгенд А. BIG DATA. Вся технология в одной книге / Андреас Вайгенд., 2018.

3. What is BIG DATA? Introduction, Types, Characteristics & Example [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/what-is-big-data.html.

4. Марр Б. Big Data: Using SMART Big Data, Analytics and Metrics To Make Better Decisions / Бернард Марр., 2015.

5. Structured vs. Unstructured Data [Електронний ресурс] – Режим доступу до ресурсу: https://www.datamation.com/big-data/structured-vs-unstructured-data.html.

6. Structured vs. Unstructured Data [Електронний ресурс] – Режим доступу до ресурсу: Вайгенд А. Structured vs. Unstructured Data [Електронний ресурс] / Андреас Вайгенд – Режим доступу до ресурсу: http://https//www.bigdataframework.org/data-types-structured-vs-unstructured-data/.

7. Big Data [Електронний ресурс] – Режим доступу до ресурсу: https://www.it.ua/knowledge-base/technology-innovation/big-data-bolshie-dannye.

8. ТЕХНОЛОГІЇ BIG DATA: КЛЮЧОВІ ХАРАКТЕРИСТИКИ, ОСОБЛИВОСТІ ТА ПЕРЕВАГИ [Електронний ресурс] – Режим доступу до ресурсу: https://aiconference.com.ua/uk/news/tehnologii-big-data-klyuchevie-harakteristiki-osobennosti-i-preimushchestva-97883.

9. Document Classification [Електронний ресурс] – Режим доступу до ресурсу: https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-30164-8_230#howtocite.

10. Text Analysis 101: Document Classification [Електронний ресурс] – Режим доступу до ресурсу: https://www.kdnuggets.com/2015/01/text-analysis-101-document-classification.html.

11. Document Classification [Електронний ресурс] – Режим доступу до ресурсу: https://monkeylearn.com/blog/document-classification/.

12. Newman S. Building Microservices: Designing Fine-Grained Systems / Sam Newman., 2016.

13. Microservices [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/microservices.

14. RabbitMQ [Електронний ресурс] – Режим доступу до ресурсу: https://www.rabbitmq.com/.

15. REST matters [Електронний ресурс] – Режим доступу до ресурсу: https://www.pluralsight.com/blog/tutorials/representational-state-transfer-tips.

16. VGG16 — сверточная сеть для выделения признаков изображений [Електронний ресурс] – Режим доступу до ресурсу: https://neurohive.io/ru/vidy-nejrosetej/vgg16-model/.

17. Сверточная нейронная сеть [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/348000/.

18. Basics of Image Classification Techniques in Machine Learning [Електронний ресурс] – Режим доступу до ресурсу: https://iq.opengenus.org/basics-of-machine-learning-image-classification-techniques/.

19. ImageNet [Електронний ресурс] – Режим доступу до ресурсу: http://imagenet.stanford.edu/index.

20. Natural Language Processing [Електронний ресурс] – Режим доступу до ресурсу: https://blog.bitext.com/natural-language-processing-vs-machine-learning.

21. TF-IDF с примерами кода: просто и понятно [Електронний ресурс] – Режим доступу до ресурсу: http://nlpx.net/archives/57.

22. Отношение TF/IDF [Електронний ресурс] – Режим доступу до ресурсу: https://topodin.com/seo/post/otnoshenie-tfidf.

23. Агеев М., Кураленок И., Некрестьянов И. Официальные метрики РОМИП 2010. “Российский семинар по Оценке Методов Информационного Поиска. Труды РОМИП 2010”. Казань, 2010, с. 172-187.

24. Гулин А., Карпович П., Расковалов Д., Сегалович И. Яндекс на РОМИП'2009. Оптимизация алгоритмов ранжирования методами машинного обучения // Российский семинар по Оценке Методов Информационного Поиска. Труды РОМИП 2009, с.163-168.

25. Шривастава А. Elasticsearch 7 Quick Start Guide / А. Шривастава, Д. Миллер., 2019.

26. NodeJS [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/uk/docs/.

27. What is npm [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@tanya/what-the-heck-is-npm-b8168f61e3b5.

28. RabbitMQ [Електронний ресурс] – Режим доступу до ресурсу: http://onreader.mdl.ru/RabbitMQInDepth/content/index.html.

29. Kibana [Електронний ресурс] – Режим доступу до ресурсу: https://www.elastic.co/what-is/kibana.

30. Kibana development [Електронний ресурс] – Режим доступу до ресурсу: https://www.elastic.co/guide/en/kibana/current/plugin-development.html.

31. ReactJS [Електронний ресурс] – Режим доступу до ресурсу: https://www.c-sharpcorner.com/article/what-and-why-reactjs/.

32. Моуэт Э. Использование Docker / Эдриен Моуэт., 2017.

33. Java [Електронний ресурс] – Режим доступу до ресурсу: https://www.java.com/.

34. Kotlin [Електронний ресурс] – Режим доступу до ресурсу: https://kotlinlang.org/.

35. Apache POI, взаимодействие с Excel [Електронний ресурс] – Режим доступу до ресурсу: http://java-online.ru/java-excel.xhtml.

36. What is Python? Executive Summary [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org/doc/essays/blurb/.

ДОДАТКИ

Додаток А

Код аналізу зображення

```
 import keras
import base64
from PIL import Image
from io import BytesIO
import numpy as np
from flask import Flask, request, jsonify
from keras.applications import vgg16, inception_v3, resnet50, mobilenet
from keras.preprocessing.image import img_to_array
from keras.applications.imagenet_utils import decode_predictions
from threading import Thread
import test_image
import pika
import json


vgg_model = vgg16.VGG16(weights='imagenet')
#inception_model = inception_v3.InceptionV3(weights='imagenet')
#resnet_model = resnet50.ResNet50(weights='imagenet')
#mobilenet_model = mobilenet.MobileNet(weights='imagenet')


def prepareImage(data):
  original = Image.open(BytesIO(base64.b64decode(data)))
  original = original.resize((224, 224), Image.ANTIALIAS)
  numpy_image = img_to_array(original)
```

```
    image_batch = np.expand_dims(numpy_image, axis=0)
    return image_batch

def analyzeImage(image):
    processed_image = vgg16.preprocess_input(image.copy())
    predictions = vgg_model.predict(processed_image)
    labels = []
    for item in decode_predictions(predictions)[0]:
      labels.append(item[1])
    return labels

class Analyze(Thread):
    def __init__(self, id, data):
        Thread.__init__(self)
        self.id = id
        self.data = data

    def run(self):
        image = prepareImage(self.data)
        labels = analyzeImage(image)
        result = { 'id': self.id, 'value': labels }
        connection =
pika.BlockingConnection(pika.ConnectionParameters('rabbitmq'))
        channel = connection.channel()
        channel.queue_declare(queue='analyze_image_result', durable=True)
        channel.basic_publish(exchange='',
```

```
                routing_key='analyze_image_result',
                body=json.dumps(result),
                properties=pika.BasicProperties(
                  delivery_mode = 2
                ))
        connection.close()

image = prepareImage(test_image.data)
labels = analyzeImage(image)

app = Flask('doca_analyze_image')
@app.route('/analyze', methods=["POST"])
def analyze():
    content = request.json
    id = content['id']
    data = content['value']
    result = { 'status': True }
    analyze = Analyze(id, data)
    analyze.start()
    return jsonify(result)


if __name__ == '__main__':
  app.run(port=1490, host='0.0.0.0')
```

Додаток Б

Екстракція зображень та тексту з документа

```
package kudrya.doca.analyze.docx

import com.google.gson.annotations.Expose
import com.google.gson.annotations.SerializedName
import org.apache.poi.xwpf.usermodel.XWPFDocument
import java.io.InputStream
import java.util.*
import kotlin.collections.HashMap

class Docx(@Expose(serialize = true) val id: String, inputStream: InputStream)
{
    @Expose(serialize = false)
    val images: List<String>

    @Expose(serialize = false)
    val texts: List<String>

    @Expose(serialize = true)
    val hyperlinks: List<Map<String, String>>

    @Expose(serialize = true)
    val comments: List<Map<String, String>>

    @Expose(serialize = true)
    @SerializedName(value = "has_tables")
    val hasTables: Boolean
```

```
@Expose(serialize = true)
@SerializedName(value = "has_charts")
val hasCharts: Boolean

@Expose(serialize = true)
@SerializedName(value = "has_pictures")
val hasPictures: Boolean

@Expose(serialize = true)
@SerializedName(value = "has_hyperlinks")
val hasHyperlinks: Boolean

@Expose(serialize = true)
@SerializedName(value = "has_comments")
val hasComments: Boolean

@Expose(serialize = true)
@SerializedName(value = "count_paragraph")
val countParagraph: Int

@Expose(serialize = true)
@SerializedName(value = "count_tables")
val countTables: Int

@Expose(serialize = true)
```

```
@SerializedName(value = "count_charts")
val countCharts: Int

@Expose(serialize = true)
@SerializedName(value = "count_pictures")
val countPictures: Int

@Expose(serialize = true)
@SerializedName(value = "count_hyperlinks")
val countHyperlinks: Int

@Expose(serialize = true)
@SerializedName(value = "count_comments")
val countComments: Int

init {
    val document = XWPFDocument(inputStream)
    hasPictures = document.allPictures.size > 0
    countPictures = document.allPictures.size
    images =  document.allPictures.map {
        String(Base64.getEncoder().encode(it.data))
    }
    countParagraph = document.paragraphs.size
    texts = document.paragraphs.map {
        it.text
    }.filter {
```

```
            it.isNotEmpty()
        }
        hasHyperlinks = document.hyperlinks.isNotEmpty()
        countHyperlinks = document.hyperlinks.size
        hyperlinks = document.hyperlinks.map {
            val res = HashMap<String, String>()
            res["name"] = it.id
            res["url"] = it.url
            res
        }
        hasTables = document.tables.size > 0
        countTables = document.tables.size
        hasCharts = document.charts.size > 0
        countCharts = document.charts.size
        hasComments = document.comments.isNotEmpty()
        countComments = document.comments.size
        comments = document.comments.map {
            val res = HashMap<String, String>()
            res["author"] = it.author
            res["comment"] = it.text
            res
        }
    }
}

package kudrya.doca.analyze.docx
```

```
import com.google.gson.GsonBuilder
import com.rabbitmq.client.ConnectionFactory
import io.javalin.Javalin


fun main() {
    val factory = ConnectionFactory()
    val gson = GsonBuilder()
            .excludeFieldsWithoutExposeAnnotation()
            .create()
    factory.host = "rabbitmq"
    val app: Javalin = Javalin.create().start(7001)
    app.post("/analyze") {
        val file = it.uploadedFile("value")
        val id = it.queryParam("id")
        val docx = Docx(id!!, file!!.content)
        docx.texts.forEach { text ->
            SendToAnalyze("http://analyze_text:1491/analyze", id, text).send()
        }
        docx.images.forEach { image ->
            SendToAnalyze("http://analyze_image:1490/analyze", id, image).send()
        }

        factory.newConnection().use { connection ->
connection.createChannel().use { channel ->
```

```
                channel.queueDeclare("analyze_docx_result", true, false, false, null)
                val text = gson.toJson(docx)
                channel.basicPublish("", "analyze_docx_result", null,
gson.toJson(docx).toByteArray());
            } }
            it.result("{ \"status\": true }")
        }
    }
```

Додаток В

Плагін для завантаження документа в Kibana

```
import React from 'react';
import {
  EuiPage,
  EuiPageHeader,
  EuiTitle,
  EuiPageBody,
  EuiPageContent,
  EuiTextArea,
  EuiPageContentBody,
  EuiButton,
  EuiFilePicker
} from '@elastic/eui';

export class Main extends React.Component {
  constructor(props) {
    super(props);
```

```
  this.state = {};
  this.params = {};
}

render() {
  const { title } = this.props;
  return (
    <EuiPage>
      <EuiPageBody>
        <EuiPageHeader>
          <EuiTitle size="l">
            <h1>Upload file</h1>
          </EuiTitle>
        </EuiPageHeader>
        <EuiPageContent>
          <EuiPageContentBody>
            <EuiTextArea
              placeholder='Information about document'
              onChange={text => {
                this.params.about = text.target.value;
              }
              }
            />
            <br></br>
            <br></br>
            <EuiFilePicker
```

```
        id='id'
        multiple
        initialPromptText="content that appears in the dropzone if no file is attached"
        onChange={files => {
          if (files.length == 1) {
            const file = files[0];
            if (file.type == 'application/vnd.openxmlformats-officedocument.wordprocessingml.document') {
              this.params.file = file;
              this.params.name = file.name;
            }
          }
        }
        }
      />
      <br></br>
      <br></br>
      <br></br>
      <EuiButton
        fill
        onClick={() => {
          console.log(this.params);
          if (this.params.file && this.params.about) {
            const { httpClient } = this.props;
            const formData = new FormData();
```

```
                formData.append('file', this.params.file);
                formData.append('name', this.params.name);
                formData.append('about', this.params.about);
                fetch('../api/upload_file/upload', {
                  method: 'POST',
                  body: formData,
                  headers: {
                    'kbn-xsrf': 'xxx'
                  }
                });

              }
            }}>
            Upload
          </EuiButton>
          </EuiPageContentBody>
        </EuiPageContent>
      </EuiPageBody>
    </EuiPage>
  );
 }
}
```