НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

«На правах рукопису»
УДК 004.912

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
«___»_____2022 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Програмне забезпечення автоматизованого збору надвеликих масивів текстових даних»

Виконав:
студент ІІ курсу, групи ІП-13мп
Кувічка Максим Євгенович _____

Керівник:
Доцент кафедри ІПІ, к.т.н., доц.,
Олійник Юрій Олександрович _____

Рецензент:
доцент кафедри ІСТ, к.т.н., доц.,
Ткач Михайло Мартинович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2022 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення інформаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Едуард ЖАРІКОВ

«___»_____2022р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Кувічці Максиму Євгеновичу

1. Тема дисертації «Програмне забезпечення автоматизованого збору надвеликих масивів текстових даних», науковий керівник дисертації Олійник Юрій Олександрович, к.т.н., доцент, затверджені наказом по університету від «09» листопада 2022 р. № 4115-с

2. Термін подання студентом дисертації «9» грудня 2022 р.

3. Об’єкт дослідження – математичне, інформаційне та програмне забезпечення збору надвеликих масивів текстових даних.

4. Предмет дослідження – методи збору надвеликих масивів текстових даних.

5. Перелік завдань, які потрібно розробити – порівняльний аналіз наявних рішень для збору надвеликих масивів текстових даних; формулювання технічних особливостей збору надвеликих масивів текстових даних; розробка уніфікованої структури надвеликих текстових даних, зібраних з різних джерел; розробка програмного забезпечення для збору надвеликих масивів текстових даних; реалізація модульної архітектури в програмному рішенні; оцінка ефективності запропонованого рішення.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу – 3 плакати.

7. Орієнтовний перелік публікацій – одна публікація.

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9.Дата видачі завдання «1» вересня 2022 р.

Календарний план

з/п	Назва етапів виконання магістерської дисертації	Термін виконання	Примітка
1	Збір та аналіз інформації	1.09.2022	
2	Формулювання вимог	3.09.2022	
3	Огляд та аналіз моделей та алгоритмів	19.09.2022	
4	Аналіз формату вхідних, вихідних даних	24.09.2022	
5	Проектування архітектури	5.10.2022	
6	Розробка програмного забезпечення	24.10.2022	
7	Виконання експериментальних досліджень	3.11.2022	
8	Оформлення пояснювальної записки	20.11.2022	
9	Подання дисертації на попередній захист	22.11.2022	
10	Подання дисертації на захист	9.12.2022	

Студент Максим КУВІЧКА

Науковий керівник Юрій ОЛІЙНИК

РЕФЕРАТ

Розмір пояснювальної записки – 107 аркушів, містить 20 ілюстрацій, 28 таблиць, 3 додатки, 21 посиланя на джерела.

Актуальність теми. З кожним роком даних стає все більше, вони можуть принести користь в будь-якій сфері нашого життя за умови правильної обробки. Тема роботи є актуальною, оскільки на сьогодні універсального засобу для збору надвеликих масивів текстових даних з різних джерел не існує.

Метою роботи є створення уніфікації структури та формату надвеликих масивів текстових даних за рахунок використання архітектурних рішень, які дозволяють користувачам розширювати його для власних цілей з мінімальними зусиллями. Для досягнення цієї мети необхідно вирішити такі **задачі**:

- порівняльний аналіз наявних рішень для збору надвеликих масивів текстових даних;
- формулювання технічних особливостей збору надвеликих масивів текстових даних;
- розробка уніфікованої структури надвеликих текстових даних, зібраних з різних джерел;
- розробка програмного забезпечення для збору надвеликих масивів текстових даних;
- реалізація модульної архітектури в програмному рішенні;
- оцінка ефективності запропонованого рішення.

Об'єктом дослідження роботи є математичне, інформаційне та програмне забезпечення збору надвеликих масивів текстових даних.

Предметом дослідження є методи збору надвеликих масивів текстових даних.

Науковою новизною роботи є створення уніфікованого структури даних для джерел великих текстових даних різної природи, що включає зберігання мітки часу та джерела даних, а також декларування строгої структури.

Практичне значення отриманих результатів полягає у можливості використання запропонованої уніфікованої структури для інтеграції між різними системами збору надвеликих масивів текстових даних.

Зв'язок роботи з науковими програмами, планами, темами: дисертаційна робота виконувалась на кафедрі інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського» в рамках теми «Методи та технології високопродуктивних обчислень та обробки надвеликих масивів даних». Державний реєстраційний номер 0117U000924.

Апробація: Основні положення роботи доповідались і обговорювались на III Всеукраїнській науково-практичній конференції молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології (Soft-Tech-2022)».

Публікації. Наукові положення дисертації опубліковані в:

1) Кувічка М.Є. Уніфікація структури надвеликих масивів текстових даних, зібраних з різних джерел / М.Є. Кувічка, Ю.О. Олійник // Матеріали III Всеукраїнської науково-практичної конференції молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології» (SoftTech-2022 осінь) – м. Київ: НТУУ «КПІ ім. Ігоря Сікорського», 23-25 листопада 2022 р.

Ключові слова: ВЕЛИКІ ДАНІ, ЗБІР ДАНИХ, СТРУКТУРИЗАЦІЯ ДАНИХ, ВЕБСКРАПІНГ.

ABSTRACT

Explanatory note size – 107 pages, contains 20 illustrations, 28 tables, 3 applications, 21 references.

Topicality. Every year, the amount of data is increasing, it can be useful in any area of our life, provided it is properly processed. The topic of the work is relevant, because today there is no universal tool for collecting extremely large arrays of text data from various sources.

The goal of the work is to unify the structure and format of super-large arrays of text data through the use of architectural solutions that allow users to expand it for their own purposes with minimal effort. To achieve this goal, it is necessary to solve the following problems:

- perform the comparative analysis of available solutions for collecting super-large arrays of text data;
- formulation of the technical features of the collection of extremely large arrays of text data;
- development of a unified structure of super-large text data collected from various sources;
- development of software for collecting extremely large arrays of text data;
- implementation of modular architecture in a software solution;
- evaluation of the effectiveness of the proposed solution.

The object of research of the work is mathematical, informational and software for collecting super-large arrays of text data.

The subject of research is methods of collecting extremely large arrays of textual data.

The scientific novelty of the work is the creation of a unified data structure for the sources of large text data of various nature, which includes the storage of the time stamp and data source, as well as the declaration of a strict structure.

The practical significance of the obtained results lies in the possibility of using the proposed unified structure for integration between different systems for collecting extremely large arrays of text data.

Relationship with working with scientific programs, plans, topics. The work was performed at the Department of Informatics and Software Engineering of the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute" in the framework of the topic "Methods and technologies of high-performance computing and big data processing". State registration number 0117U000924.

Approbation. The main provisions of the work were reported and discussed at the III All-Ukrainian scientific and practical conference of young scientists and students "Software engineering and advanced information technologies (Soft-Tech-2022)".

Publications. The scientific provisions of the dissertation were published in:

1) Kuvichka M.Y. Unification of the structure of super-large arrays of text data collected from various sources / M.Y. Kuvichka, Yu.O. Oliinyk // Materials of the III All-Ukrainian scientific and practical conference of young scientists and students "Software engineering and advanced information technologies" (SoftTech-2022 autumn) - Kyiv: NTUU "KPI them. Igor Sikorsky", November 23-25, 2022.

Keywords: BIG DATA, DATA COLLECTION, DATA STRUCTURING, WEBSCRAPING.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Big Data – надвеликі масиви даних.

API (Application Programming Interface) – програмний інтерфейс.

DOM (Document Object Model) – специфікація прикладного програмного інтерфейсу для роботи зі структурованими документами.

JSON (JavaScript Object Notation) – це текстовий формат обміну даними між комп'ютерами.

HTTP (HyperText Transfer Protocol) – протокол передачі даних, що використовується в комп'ютерних мережах.

HTML (HyperText Markup Language) – це мова тегів, засобами якої здійснюється розмічання веб-сторінок для мережі Інтернет.

Parsing – процес аналізу вхідної послідовності символів, з метою розбору граматичної структури згідно із заданою формальною граматикою.

ВСТУП

Наразі соціальні мережі та глобальні онлайн-сервіси розвиваються вкрай стрімко. Даних стало так багато, що ми стали називати їх великими та стали краще розуміти, як їх обробляти і з якими цілями можна використати. Вже сьогодні вони допомагають налагоджувати бізнес, розвивати сільське господарство, прогнозувати поведінку фондового ринку, покращувати сферу розваг і навіть ставати президентом. Далі тільки більше: за прогнозами, вже у 2024 році буде генеруватися близько 150 зетабайтів даних і ми можемо отримати з них користь за умови правильного підходу. Проте універсального і простого рішення для цього не існує і досі. Останнім часом з’явилась велика кількість платформ та технологій для зручного та оптимізованого процесу обробки та аналізу такої кількості інформації, але саме для їх збору жодного доступного і універсального рішення наразі не було реалізовано.

Метою роботи є створення уніфікації структури та формату надвеликих масивів текстових даних за рахунок використання архітектурних рішень, які дозволяють користувачам розширювати його для власних цілей з мінімальними зусиллями.

Об'єктом дослідження роботи є математичне, інформаційне та програмне забезпечення збору надвеликих масивів текстових даних.

Предметом дослідження є методи збору надвеликих масивів текстових даних.

Планується за допомогою платформи Node.JS розробити програмне забезпечення з модульною архітектурою, яке зможе збирати дані з заданих джерел як RSS, Telegram-каналів або веб-сторінки, структурувати ці дані та видати їх на один або декілька варіантів виводу: сформувати потік текстових даних, записати їх в файл або в платформу потокової передачі подій Apache Kafka.

Завдання, що вирішуються:

- порівняльний аналіз наявних рішень для збору надвеликих масивів текстових даних;
- формулювання технічних особливостей збору надвеликих масивів текстових даних;
- розробка уніфікованої структури надвеликих текстових даних, зібраних з різних джерел;
- розробка програмного забезпечення для збору надвеликих масивів текстових даних;
- реалізація модульної архітектури в програмному рішенні;
- оцінка ефективності запропонованого рішення.

1 АНАЛІЗ НАЯВНИХ РІШЕНЬ ЗБОРУ НАДВЕЛИКИХ МАСИВІВ ТЕКСТОВИХ ДАНИХ

1.1 Опис предметної області

З останніми технологічними досягненнями кількість даних зростає з кожним днем. Наприклад, ресурси з новинами, сенсорні та соціальні мережі генерують незліченні потоки даних. Загалом, великі дані створюються з кількох джерел у різних форматах з величезною швидкістю[1] і тому їх важко зберігати та обробляти за допомогою традиційного програмного забезпечення. Для цього було розроблено спеціальні програмні засоби, які здатні зберігати значущу інформацію в різних форматах. Щоб задовольнити вимоги користувачів, а також аналізувати та зберігати складні дані, було доступно кілька аналітичних інфраструктур, які допомагають користувачам аналізувати складні структуровані та неструктуровані дані[2]. Для роботи з великими даними було запропоновано та розроблено спеціальні програми, моделі, технології, та програмне забезпечення, основна мета яких — зберігати надійні та точні результати для великих даних[3]. Крім того, для Big Data рішень потрібні найсучасніші технології ефективно зберігати й обробляти великі обсяги даних за дуже обмежений час роботи.

Великі дані включають інформацію, створену людьми та пристроями. Дані, керовані пристроями, здебільшого організовані та впорядковані, але набагато більший інтерес представляють дані, створені людиною, які існують у різних форматах і потребують більш досконалих інструментів для належної обробки та керування.

Збір великих даних зосереджений на таких типах даних:

- мережеві дані. Цей тип даних збирається у всіх видах мереж, включаючи соціальні мережі, інформаційні та технологічні мережі, Інтернет та мобільні мережі тощо;
- дані в реальному часі. Вони створюються на потокових онлайн-сервісах, таких як YouTube, Twitch або Netflix;

- транзакційні дані. Вони збираються, коли користувач здійснює онлайн-покупку (інформація про продукт, час покупки, способи оплати тощо);
- географічні дані. Дані про місцезнаходження гаджетів, людей, транспортних засобів, будівель та інших об’єктів, які постійно обмінюються даними з супутниками;
- дані природної мови. Ці дані збираються в основному за допомогою голосового пошуку, який можна виконати на різних пристроях, що мають доступ до Інтернету;
- дані часових рядів. Цей тип даних пов’язаний зі спостереженням за тенденціями та явищами, що відбуваються в даний момент і протягом певного періоду часу, наприклад, глобальні рівні температури, рівні смертності, рівні забруднення тощо;
- пов’язані дані. Вони базуються на веб-технологіях HTTP, RDF, SPARQL і URI і призначені для забезпечення семантичних зв’язків між різними базами даних, щоб комп’ютери могли правильно читати та виконувати семантичні запити.

Існує три різні типи Big Data платформ: інструменти інтерактивного аналізу, інструменти потокової обробки (stream processing) та інструменти пакетної обробки (batch processing)[4]. Інструменти інтерактивного аналізу використовуються для обробки даних в інтерактивних середовищах і взаємодії з даними в реальному часі. Apache Drill і Dremel від Google є фреймворками для зберігання даних у реальному часі. Інструменти потокової обробки використовуються для зберігання інформації в безперервному потоці[5]. Основними платформами для зберігання потокової інформації є S4 і Storm. Інфраструктура Hadoop використовується для зберігання інформації пакетами. Методи великих даних використовуються в різних дисциплінах, таких як обробка сигналів, статистика, візуалізація, аналіз соціальних мереж, нейронні мережі та збір даних[6].

Великі дані характеризуються трьома V: volume (обсяг), velocity (швидкість) і variety (різноманітність). Ці характеристики були введені компанією Gartner для визначення різноманітних проблем у великих даних[7].

Завдяки архітектурі нового покоління дані тепер зберігаються в різних форматах, отже, три V можна розширити до п’яти V, а саме: volume (обсяг), velocity (швидкість), variety (різноманітність), value (цінність) і veracity (правдивість)[8, 13]. На рисунку 1.1 [21] показано п’ять “V” великих даних.

Рисунок 1.1 – П’ять “V” великих даних

- *Обсяг*: дані генеруються кількома джерелами (сенсорами, соціальними мережами, смартфонами тощо) і постійно розширюються. Інтернет виробляє глобальні дані у великій кількості. У 2012 році щодня створювалося приблизно 2,5 екзабайта даних. Згідно зі звітом International Data Cooperation, обсяг даних у 2013 році подвоївся, досягнувши 4,4 зетабайт. У 2020 році обсяг даних сягнув 40 зетабайт.

- *Швидкість* по суті, вимірює, наскільки швидко надходять дані. Деякі дані надходять у режимі реального часу, тоді як інші надсилаються уривками, надходячи до нас пакетами. І оскільки не всі платформи сприймають вхідні дані з однаковою швидкістю, важливо не узагальнювати, не робити поспішних висновків або кореляцій, не отримавши всіх даних.

- *Різноманітність*: дані генеруються в різних форматах за допомогою соціальних мереж, смартфонів або датчиків. Ці інструменти генерують дані у формі журналів даних, зображень, відео, аудіо, документів і тексту. Дані також можуть бути структурованими, напівструктурованими та неструктурованими[9].

- *Цінність*: цінність є важливою характеристикою великих даних. Це стосується того, як можна обробляти дані та перетворювати їх на корисну інформацію[10].

- *Правдивість*: правдивість стосується якості, правильності та достовірності даних. Тому дотримання правдивості даних є обов'язковим[11, 12]. Наприклад, великі обсяги даних створюють плутанину, тоді як невеликі обсяги даних можуть передавати неповну або половинчасту інформацію.

Обчислення великих даних можна загалом розділити на два типи на основі вимог до обробки, а саме пакетне обчислення великих даних і потокове обчислення великих даних. Пакетної обробки великих даних недостатньо, коли йдеться про аналіз сценаріїв застосунків у реальному часі. Більшість даних, створених у потоці даних у реальному часі, потребують аналізу даних у реальному часі. Крім того, вихідні дані мають бути згенеровані з низькою затримкою, а будь-які вхідні дані мають бути відображені у щойно створеному виводі протягом секунд. Це вимагає аналізу потоку великих даних [14]. Попит на потокову обробку зростає. Причина полягає не лише в тому, що потрібно обробити величезний обсяг даних, але й у тому, що дані мають бути швидко оброблені, щоб організації та підприємства могли реагувати на зміни умов у реальному часі [15].

1.2 Аналіз наявних рішень

Сам по собі процес збору інформації не несе особливої користі, бо для того, щоб мати змогу проаналізувати отримані дані, їх треба відфільтрувати, трансформувати, агрегувати, представити у вигляді звіту, діаграми або графіка. Для цього програмне забезпечення для збору даних повинно інтегруватись з

системами обробки надвеликих масивів даних, найпопулярнішими з яких є: Hadoop, GridGain, MapReduce, HPCC, Apache Storm, Samza, Apache Spark Streaming, Flink, Kafka Streams та IBM Streams. Більшість з цих платформ підтримує доволі обмежений перелік вхідних форматів даних: файлова система, Kafka, web sockets connection.

Тому якщо система збору даних матиме лише один формат вихідних даних, для обробки декількох джерел доведеться розгортати декілька одночасних екземплярів цього ПЗ, що є значно більш ресурсозатратно і гірше вплине на масштабованість цілісного програмного продукту.

1.2.1 Підхід нечіткого агента для розумного вилучення даних у середовищах Big Data

Основна мета цієї статті [16] полягала в тому, щоб запропонувати підхід нечіткого агента для вилучення розумних даних у розподілених великомасштабних середовищах. Особливу увагу було приділено загальності запропонованого підходу, завдяки якому користувач має повну свободу налаштування різних параметрів і адаптації їх до сфери застосування. Основною метою мультиагентної системи було усунення неповних, невизначених, надлишкових і нерелевантних даних на основі прийнятого порогу релевантності, визначеного користувачем. У Wooldridge (2009) агенти визначаються як суб’єкти, здатні виконувати автономні дії в середовищі, в якому вони знаходяться, для досягнення конкретних цілей. Виходячи з цього, багатоагентні технології визначені як група агентів, які співпрацюють один з одним для досягнення глобальної мети, одночасно досягаючи локальних цілей (Weiss and Multiagent, 1999).

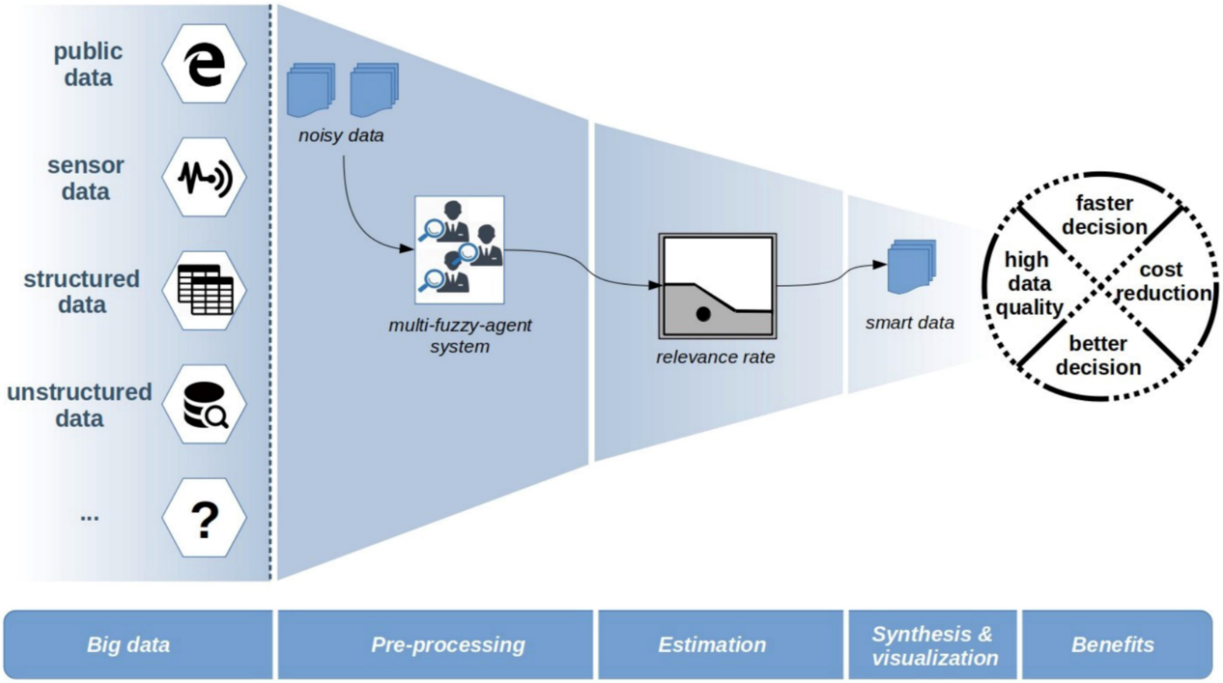


Рисунок 1.2 – Архітектура запропонованого рішення з нечітким агентом

Роль систем з кількома нечіткими агентами в основному може проявлятися на етапі попередньої обробки, де вони будуть інтегровані в обчислювальні вузли Big Data. Як показано на рисунку 1.2 [16], ці агенти застосовують нечітку логіку для визначення рівня релевантності потокових даних. Таким чином, якщо результат дефазифікації перевищує відсоток релевантності, попередньо вказаний користувачем, дані вважатимуться релевантними, а якщо не перевищує - нерелевантними і ними буде знехтувано. Основні нововведення мультифазіагентних систем:

- *краща якість даних*: можливість оцінити рівень релевантності даних і відфільтрувати нерелевантні призводить до підвищення якості даних;
- *точна методологія вирішення проблем*: нечіткі агенти сприймають дані в неточних термінах, а потім відповідають точними діями. Це може вирішити проблему неповноти та невизначеності даних;
- *нижча вартість*: нечітка логіка дозволяє недорогим мікроконтролерам виконувати більш складні функції, які традиційно виконуються потужнішими машинами;

- *універсальність*: користувачі можуть налаштовувати допустимий поріг релевантності в залежності від доменної області. Таким чином, цей підхід є загальним і не залежить від конкретної області.

Запропонована у статті мультиагентна система становить мережу нечітких агентів, розподілених на різних вузлах Big Data. Мережа розділена на кластери нечітких агентів-сусідів.

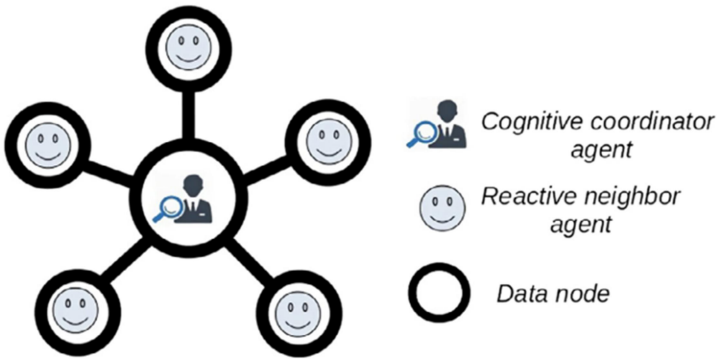


Рисунок 1.3 – Кластер нечітких агентів

Як показано на рисунку 1.3 [16], кожен вузол з обличчям пов’язаний з агентом-координатором, який створений для покращення координації між агентами, для легкої передачі даних кінцевому користувачеві, і особливо для усунення надлишкової координації між вузлами. Агенти-сусіди виступають у цьому випадку як реактивні агенти. Їх основна роль обмежена передачею отриманих даних агенту-координатору для обробки. Такі сервери є масштабованими, тобто кількість вузлів постійно змінюється, що призводить до періодичних змін у структурі мережі нечітких агентів. Тому деякі реактивні агенти в одній структурі можуть стати агентами-координаторами в іншій і навпаки. Агент-координатор застосовує нечітку логіку кожного разу, коли він отримує дані від своїх сусідів, таким чином релевантність даних буде оцінена, а нерелевантні дані будуть видалені.

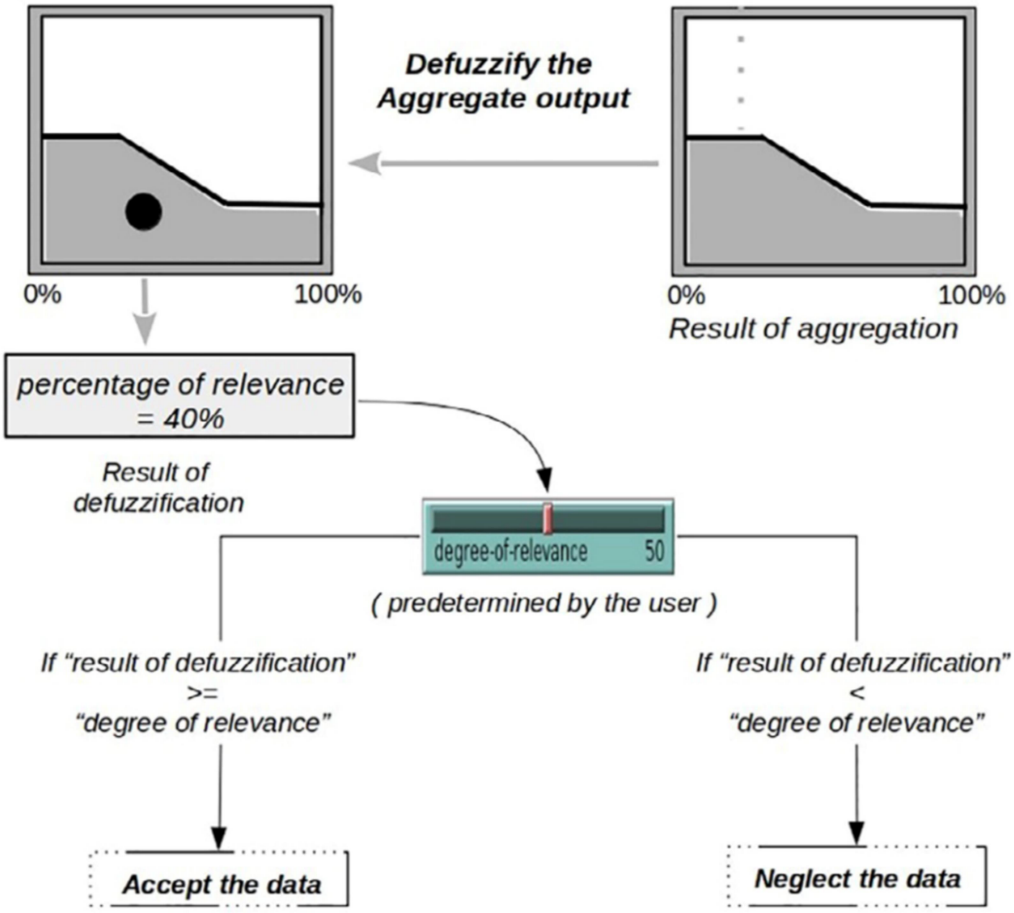


Рисунок 1.4 – Модель визначення релевантності даних

Нечітка логіка, застосована агентом-координатором складається з таких кроків:

- реалізація нечітких концепцій і нечітких вхідних даних: нечіткі концепції представлені параметрами, які дозволяють агентам вимірювати релевантність даних. Ці параметри різні для різних програми, і саме користувачі обирають які типи параметрів додати ґрунтуючись на своїх знаннях у цій сфері. Фундаментальним параметром, який необхідно визначити в усіх типах додатків, є ступінь подібності даних. Він допомагає виміряти релевантність даних відповідно до рівня їхньої схожості, що означає, що зайві дані вважатимуться нерелевантними та будуть пропущені. Відповідно до контексту програми можна додати інші вторинні параметри, щоб допомогти виміряти релевантність даних;

- застосування нечітких операцій: визначивши різні поняття та вхідні дані, різні параметри будуть об’єднані у форматі правил IF-THEN з використанням нечітких операцій (AND/OR);
- застосування методів імплікації та агрегації: як і будь-яка система логічного висновку, яка використовує нечітку логіку, після застосування правил настає черга методів імплікації та агрегації. Підтримуються два методи імплікації: min (мінімум) і prod (продукт). Крім того, підтримуються три методи агрегації: max (максимальний), probor (ймовірнісний або) і sum (просто сума вихідного набору кожного правила);
- дефазифікація сукупного виходу: ми підходимо до останнього етапу, де нечіткий агент порівнює результат дефазифікації з визначеним користувачем порогом релевантності, щоб вирішити, чи є інформація релевантною чи ні. Отже, якщо відсоток релевантності в результаті цієї нечіткої логіки дорівнює або перевищує відсоток, попередньо визначений користувачем, інформація буде класифікована як релевантна; інакше інформація буде класифікована як нерелевантна і не зберігатиметься (рисунок 1.4). Існує п’ять підтримуваних методів дефазифікації: FOM (перший з максимумів), LOM (останній з максимумів), MOM (середина максимумів), MeOM (середнє значення максимумів) і COG (центр ваги).

Щоб продемонструвати роботу запропонованого підходу, автори вибрали мережу керування даними з багатьма датчиками, де була запропонована широкомасштабна мережа бездротових датчиків на основі кількох нечітких агентів. Основною метою мультиагентної системи було усунення неповних, невизначених, надлишкових і нерелевантних даних на основі прийнятого порогу релевантності, визначеного користувачем. Результати моделювання показали, що вилучення інтелектуальних даних може значно допомогти в подовженні терміну служби мережі, але воно також показало, що чим більше підвищується прийнятий поріг релевантності у відсотках, тим більше збільшується термін служби мережі і тим більше точність видобуті дані зменшуються. Насправді, що стосується різних методів збору даних, не існує

єдиного методу, застосовного до всіх областей Big Data, і дослідники даних повинні враховувати конкретну проблему, особливість, контекст програми, продуктивність вимоги та інші фактори наборів даних для вибору належної стратегії.

Переваги:

- забезпечує гнучке до налаштування відсіювання нерелевантних даних;
- може легко масштабуватись.

Недоліки:

- працює лише з одним джерелом даних - з мережею керування даними з багатьма датчиками;
- для коректної роботи систему треба налаштовувати під кожну доменну область окремо.

1.2.2 Htwitt

Htwitt – це платформа на основі Hadoop для аналізу та візуалізації потокових даних Twitter.

В роботі [17] представлено фреймворк HTwitt, створений на основі екосистеми Hadoop, який складається з алгоритму MapReduce і набору методів машинного навчання, вбудованих у платформу аналітики Big Data, для ефективного вирішення таких проблем:

- традиційні методи обробки даних є неадекватними при роботі з великими даними;
- попередня обробка даних вимагає значних ручних зусиль;
- перед класифікацією необхідні знання предметної області;
- семантичне пояснення ігнорується.

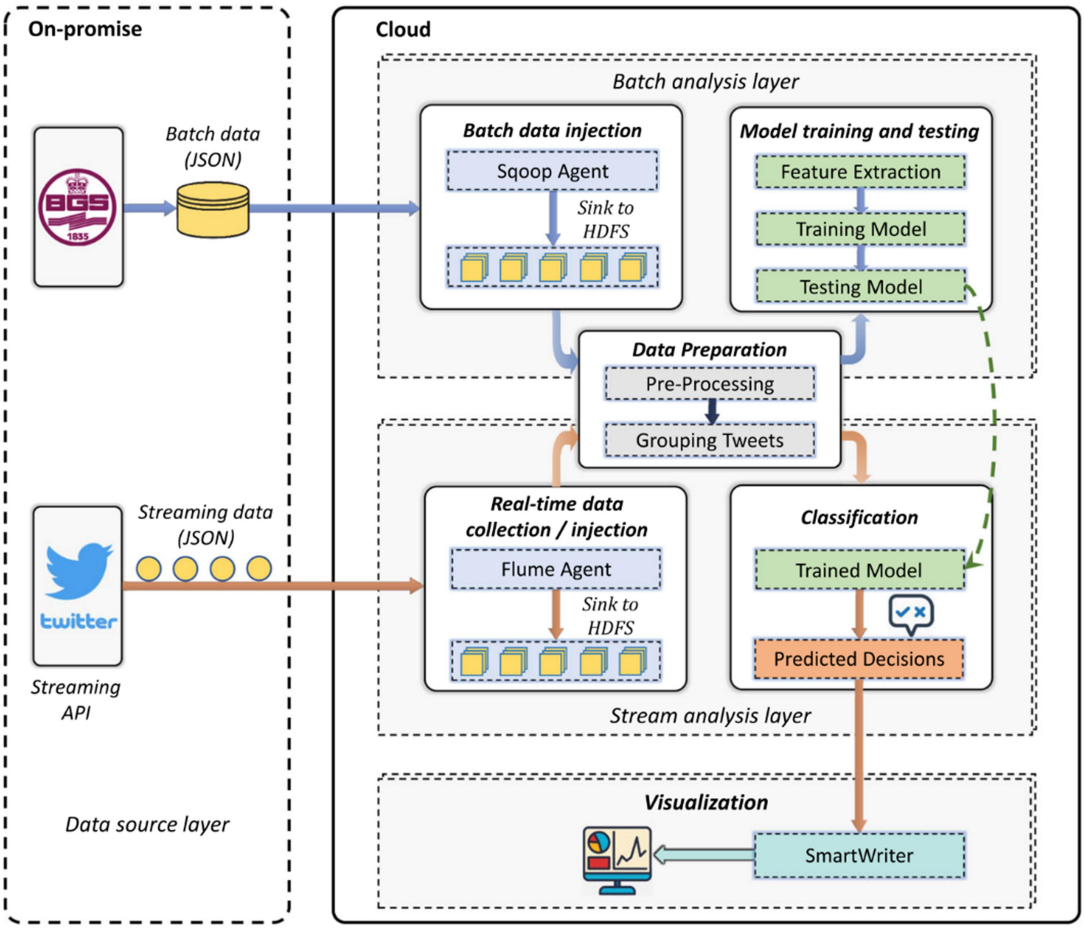


Рисунок 1.5 – Архітектура запропонованого рішення HTwitt

У цій роботі ці проблеми пропонується подолати за допомогою різних алгоритмів у поєднанні з класифікатором Байєса для забезпечення надійності та високоточних рекомендацій у віртуалізації та хмарних середовищах. Ці функції відрізняють HTwitt від інших ефективним і практичним дизайном для класифікації тексту в аналізі великих даних. Основний внесок статті полягає в тому, щоб запропонувати структуру для побудови систем раннього попередження про зсуви шляхом точного визначення корисних твітів і візуалізації їх разом з обробленою інформацією. Продемонстровано результати експериментів, які кількісно визначають рівні перенавчання на етапі навчання моделі з використанням різних розмірів наборів даних реального світу на етапах машинного навчання. Результати демонструють, що запропонована система забезпечує високоякісні результати з оцінкою майже 95% і відповідає вимогам системи класифікації на основі Hadoop.

У рамках HTwitt дані спочатку аналізуються та переносяться у формат CSV, а потім класифікуються в модулі підготовки даних за допомогою алгоритму MapReduce на основі місця, звідки було опубліковано твіти, а також дати й часу публікації. Ці результати використовуються для навчання та тестування моделі для класифікації твітів, пов'язаних зі стихійним лихом, у системі навчання та тестування моделі з використанням алгоритму машинного навчання Naïve Bayes для ефективної класифікації. Ця структура запропонована, щоб відрізнити твіти, пов'язані зі стихійними лихами, від твітів, які включають певні омофони, але не пов'язані з природними явищами.

В цій роботі описано модуль збору/впровадження даних у реальному часі, щоб отримувати потокові дані з Twitter на основі конкретних ключових слів за допомогою Twitter Streaming API і одночасно зберігати їх у розподіленій файловій системі Hadoop (HDFS). Однак Twitter має обмеження на потоковий API, який дозволяє користувачам завантажувати лише обмежену кількість твітів у визначений час. З цієї причини потрібен ефективний інструмент для отримання надвеликих масивів даних із Twitter. Тому було вирішено використовувати Apache Flume, розповсюджений, надійний і налаштовуваний інструмент, щоб збирати, агрегувати та переміщувати величезні обсяги даних. Щоб візуалізувати результат передбачуваних рішень після класифікації твітів, модуль візуалізації дозволяє користувачам бачити точне розташування твітів.

Дані, отримані для цієї роботи, знаходяться у складному вкладеному форматі JSON, який включає текст користувача разом із приблизно 74 різними атрибутами, такими як хештеги, координати, об'єкти, ім'я, ім'я для відображення, ім'я користувача, URL-адреси, ідентифікатор твіта тощо, які необхідно ретельно опрацювати, щоб отримати необхідну інформацію. Дуже важливо знати структуру твітів, щоб виконати попередню обробку даних. Попередня обробка — це важливий крок, який робить текст більш сприйнятливим, видаляючи шум, безглузді фрази та непотрібні повторення, щоб алгоритми машинного навчання могли підвищити свою ефективність. Twitter Streaming API надає твіти у форматі JSON (JavaScript Object Notation).

Однак це викликає певні труднощі, наприклад розмір і складність даних. Такі складні та великі обсяги даних потребують Big Data технологій для обробки та підготовки даних до навчання та тестування. Для досягнення цієї мети створено компонент підготовки даних, який складається з попередньої обробки та групування твітів. На етапі попередньої обробки необхідні елементи, такі як код країни, координати, дата створення, ідентифікатор користувача та текст, ретельно аналізуються за допомогою бібліотеки Gson4, яка використовується для десеріалізації JSON.

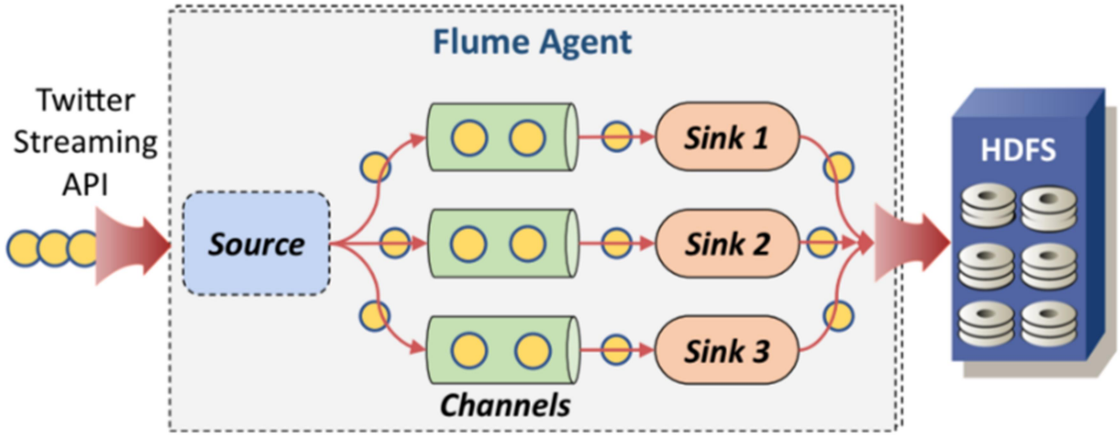


Рисунок 1.6 – Високорівнева архітектура Apache Flume

Дані в режимі реального часу збираються з Twitter за допомогою Apache Flume через Twitter Streaming API. Агент Apache Flume складається з трьох основних компонентів: Source (Джерело), Sink (Приймач) і Channel (Канал), як показано на рисунку 1.6 [17]. Джерело передає події (твіти), що надходять з Twitter Streaming API, у Канали, які буферизують і зберігають події, тоді як Приймачі отримують події з Каналів та зберігають їх у HDFS. Twitter є джерелом, тоді як HDFS використовується як сховище. Конфігурація авторизації в Twitter реалізована в частині конфігурації Джерела, а конфігурація HDFS задається в конфігурації Приймача. У властивостях джерела вказані конкретні ключові слова для збору твітів, які необхідні для досягнення мети цього дослідження. У нашому випадку ключовим пунктом для збору колекції твітів, пов’язаних саме зі стихійними лихами, є визначення ключових слів у файлі конфігурації. У розділі Flume Source ключові слова

вказано для властивості TwitterAgent.sources.Twitter.keywords, щоб розрізняти пов’язані та непов’язані твіти під час операції збору. Це дозволяє вибирати твіти, які містять лише певні ключові слова, перш ніж зберігати події в Каналах. Ми встановлюємо ємність для Каналів як 10 000, що вказує на максимальну кількість подій, які канали прийматимуть із Джерела, використовуючи основну пам’ять. Для Приймачів ми визначаємо місце, де події, що надходять із каналів, зберігатимуться в HDFS, і встановлюємо розмір пакета 1000, а формат файлу даних — DataStream.

Аналіз даних із Twitter має певні труднощі щодо збору та обробки даних через складність даних у форматі JSON. Крім того, це вимагає використання ефективних методів машинного навчання разом із інструментами великих даних для вилучення корисної та корисної інформації з великої кількості даних. Іншою проблемою є збір даних із Twitter через Streaming API через деякі обмеження, встановлені Twitter. Головне завдання полягає в тому, щоб розрізнити, чи твіти пов’язані зі стихійним лихом чи ні через слова-омоніми. Враховуючи ці проблеми, оцінка показує, що класифікація за допомогою Apache Mahout разом із алгоритмом Naïve Bayes є дуже ефективною, а запропонована структура HTwitt є багатообіцяючою, маючи точність 94,28%.

Переваги:

- забезпечує ефективний процес обробки отриманих даних за рахунок аналізу не тільки синтаксичного, а й семантичного за допомогою алгоритму Naïve Bayes та використання нейронних мереж;
- організовує потік даних попри обмеження з боку джерела даних.

Недоліки:

- працює лише з одним джерелом даних, сильно прив’язане до його специфіки, формату даних, доступності і стабільності роботи;
- не забезпечує жодної відмовостійкості модуля збору даних.

1.3 Постановка задачі

Метою роботи є створення уніфікації структури та формату надвеликих масивів текстових даних за рахунок використання архітектурних рішень, які дозволяють користувачам розширювати його для власних цілей з мінімальними зусиллями. Для досягнення цієї мети необхідно вирішити наступні задачі:

- порівняльний аналіз наявних рішень для збору надвеликих масивів текстових даних;
- формулювання технічних особливостей збору надвеликих масивів текстових даних;
- розробка уніфікованої структури надвеликих текстових даних, зібраних з різних джерел;
- розробка програмного забезпечення для збору надвеликих масивів текстових даних;
- реалізація модульної архітектури в програмному рішенні;
- оцінка ефективності запропонованого рішення.

Створений продукт повинен відповідати таким вимогам:

- забезпечувати збір надвеликих масивів текстових даних з декількох джерел (щонайменше двох типів: RSS-стрічок, Telegram-каналів);
- забезпечувати вивід зібраних надвеликих масивів текстових даних в декілька варіантів виводу (щонайменше трьох типів: в файлову систему, в Kafka, в ElasticSearch, в Web Sockets);
- підтримувати модульну архітектуру. Для підключення додаткового формату вводу необхідно лише створити модуль під’єднання до нового ресурсу;
- приводити зібрані даних з усіх джерел до одного уніфікованого формату;
- бути стійким до відмов (зберігати працездатність після відмови одного з компонентів);
- load balancing;

- бути здатним до конфігурації джерел для збору даних без перезапуску всієї системи.

Це програмне забезпечення можна використати для простої організації збору надвеликих масивів текстових даних з обраних джерел і зручного під’єднання до вже наявної системи обробки потоків таких даних.

Висновки до розділу

У першому розділі проведено аналіз наявних рішень для збору надвеликих масивів текстових даних та предметної області, проаналізовані основні сучасні розробки, визначено переваги та недоліки цих рішень. В результаті проведеного аналізу сформульована постановка завдань, визначено призначення, цілі та вимоги до програмного забезпечення.

2 МЕТОДИ ЗБОРУ НАДВЕЛИКИХ МАСИВІВ ТЕКСТОВИХ ДАНИХ

2.1 Загальні відомості

Існує два терміни, які в загальному описують процес збору даних – це вилучення даних (Data Extraction) та вилучення інформації (Information Extraction). Хоч на перший погляд вони і здаються однаковими, проте між ними є суттєва різниця, яка є визначною для всієї подальшої роботи та вибору методів для вирішення задач цієї роботи.

2.1.1 Дані та інформація

Зазвичай терміни «дані» та «інформація» використовуються як синоніми. Однак між ними є тонка різниця.

Дані визначаються як набір дискретних значень, які передають інформацію, що описує кількість, якість, факт або інші основні одиниці значення або просто послідовності символів, які можна далі інтерпретувати. Дані можуть надходити у формі тексту, спостережень, цифр, зображень, чисел, графіків або символів. Наприклад, дані можуть містити індивідуальні ціни, вагу, адреси, вік, імена, температури, дати або відстані. Дані є необробленою формою знань і самі по собі не мають жодного значення чи мети. Іншими словами, необхідно спочатку інтерпретувати дані, описати контекст, щоб вони набули значення.

Інформація визначається як знання, отримане шляхом навчання, спілкування або дослідження. По суті, інформація є результатом аналізу та інтерпретації фрагментів даних. У той час як дані – це окремі цифри, числа чи графіки, інформація – це сприйняття цих фрагментів знань. Наприклад, набір даних може містити показники температури в певному місці за кілька років. Без будь-якого додаткового контексту ці температури не мають значення. Однак, проаналізувавши та впорядкувавши цю інформацію, можна визначити сезонні температурні моделі або навіть ширші кліматичні тенденції. Лише тоді, коли дані організовані та скомпільовані в корисний спосіб, вони можуть надати інформацію, яка буде корисною для інших.

2.1.2 Вилучення даних та вилучення інформації

Вилучення даних — це дія або процес отримання даних із (зазвичай неструктурованих або погано структурованих) джерел даних для подальшої обробки даних або зберігання даних. Таким чином, вилучення для проміжної системи зазвичай супроводжується перетворенням даних і, можливо, додаванням метаданих перед їх передаванням на наступний етап робочого процесу. Типові джерела неструктурованих даних включають веб-сторінки, електронні листи, документи, PDF-файли, сканований текст, файли буфера, оголошення тощо. Отримання даних з цих неструктурованих джерел перетворилося на значну технічну проблему, оскільки, оскільки історично вилучення даних доводилося мати справу зі змінами у форматах фізичного апаратного забезпечення, більшість поточних витягів даних стосується вилучення даних із цих неструктурованих джерел даних і з різних форматів програмного забезпечення.

Структуризація неструктурованих даних досягається декількома шляхами:

- співставлення текстових шаблонів, таких як регулярні вирази, з метою ідентифікації дрібної або масштабної структури. Наприклад, записи у звіті та пов’язані з ними дані з колонтитулів;
- використання підходу на основі таблиць для ідентифікації загальних розділів у межах конкретної предметної області. Наприклад, у файлі резюме, можна ідентифікувати навички, попередній досвід роботи, кваліфікацію тощо, використовуючи стандартний набір загальновживаних заголовків, специфічний для мови, якою складено це резюме. Так, дані про освіту можна знайти в розділі з назвою «Освіта», «Кваліфікація», «Навчання»;
- використання текстової аналітики, щоб спробувати зрозуміти сенс тексту і зв’язати його з іншою інформацією.

Вилучення інформації — це завдання автоматичного вилучення структурованої інформації з неструктурованих або напівструктурованих документів в форматі, зручному для обробкою комп’ютером, та інших

електронних джерел. У більшості випадків ця діяльність стосується обробки текстів людською мовою за допомогою обробки природної мови (NLP) або створення автоматичних анотацій та вилучення вмісту із зображень, аудіо, відео, документів.

Застосування вилучення інформації до тексту пов’язане з проблемою спрощення тексту, щоб створити структуроване уявлення про інформацію, присутню у тексті. Загальна мета полягає в тому, щоб створити більш легкий для машинного читання текст для обробки речень.

Найбільш широко поширеними є такі стандартні методи:

а) регулярні вирази або вкладена група регулярних виразів;

б) використання класифікаторів:

1) генеративні. Наприклад, наївний класифікатор Байєса;

2) дискримінаційні. Наприклад, моделі максимальної ентропії, як мультиноміальна логістична регресія;

в) моделі послідовності:

1) рекурентна нейронна мережа;

2) прихована модель Маркова;

3) умовна модель Маркова;

4) модель Маркова з максимальною ентропією;

5) умовні випадкові поля (Conditional random fields) зазвичай використовуються для таких різноманітних завдань, як вилучення інформації з наукових статей або вилучення навігаційних інструкцій.

Загалом, вилучення інформації досягається за рахунок сильної прив’язки до конкретного джерела даних та чіткого розуміння його контексту. Але оскільки метою роботи є розробка саме універсального рішення для збору даних, для вирішення задачі такого роду, необхідно залучити методи вилучення даних, а не інформації.

2.2 Методи збору надвеликих масивів текстових даних

До найбільш розповсюджених джерел великих текстових даних належать сховища даних, файлові системи, соціальні мережі, інтернет-ресурси, наукові та медичні реєстри. Серед них особливо можна виділити соціальні мережі та інтернет-ресурси, бо ці джерела даних є:

- відкритими. Доступ можна отримати після загальнодоступної реєстрації, але жодного спеціального права доступу зазвичай не вимагається;
- оновлюваними. Нові дані в цих джерелах генеруються щосекунди, бо створюються великою кількістю користувачів;
- різнорідними. Сховища даних та різного роду реєстри зазвичай є сильно прив'язаними до своєї предметної області і робота з якими є скоріше вилученням інформації чи пошуком конкретної інформації, аніж збором даних. На відміну від цих джерел, інтернет-ресурси та соціальні мережі генерують величезну кількість різної інформації, серед якої можна знайти дані з будь-якої предметної області.

Для збору даних з інтернет-ресурсів застосовують метод web data extraction (або web scraping). Програмне забезпечення для цього методу має мати прямий доступ до мережі Інтернет через протокол HTTP або веб-браузер. Хоча веб-скрапінг може виконуватися користувачем програмного забезпечення вручну, цей термін зазвичай стосується автоматизованих процесів, реалізованих за допомогою бота або веб-сканера. Це форма копіювання, за якої певні дані збираються та копіюються з Інтернету, як правило, у центральну локальну базу даних або електронну таблицю, для подальшого пошуку чи аналізу. Web data extraction передбачає отримання веб-сторінки та вилучення даних з неї. Вміст сторінки можна парсити, проводити пошук та переформатовувати, а її дані скопіювати в електронну таблицю або завантажити в базу даних. Веб-скрапери зазвичай беруть щось зі сторінки, щоб використати це з іншою метою в іншому місці.

Веб-сторінки створюються з використанням мов текстової розмітки HTML та XHTML і часто містять велику кількість корисних даних у текстовій

формі. Однак більшість веб-сторінок розроблено для кінцевих користувачів, а не для автоматизованого використання. У результаті були розроблені спеціалізовані інструменти та програмне забезпечення для полегшення копіювання веб-сторінок. Існують методи, які деякі веб-сайти використовують для запобігання сканування веб-сторінок, наприклад виявлення та заборона роботам сканувати їхні сторінки. У відповідь на це існують системи веб-збирання, які покладаються на використання методів синтаксичного аналізу DOM, комп'ютерного зору та обробки природної мови для імітації перегляду веб-сторінок людиною, щоб уможливити збір вмісту веб-сторінки для аналізу в автономному режимі.

Станом на сьогодні, існують такі методи проведення збору даних з веб-сторінок:

- ручний збір. Найпростіша форма збору даних з веб-сайтів — це ручне копіювання даних із веб-сторінки в текстовий файл або електронну таблицю. Іноді навіть найкраща технологія веб-збирання не може замінити самостійне дослідження людини, а іноді це може бути єдиним працездатним рішенням, коли веб-сайти для збирання явно встановлюють бар'єри для запобігання машинній автоматизації;
- співставлення текстового шаблону. Простий, але дієвий підхід до отримання даних з веб-сторінок, який базується на використання UNIX-утиліти grep або засобах зіставлення регулярних виразів;
- програмування HTTP. Статичні та динамічні веб-сторінки можна отримати, надсилаючи HTTP-запити на віддалений веб-сервер за допомогою програмування сокетів;
- парсинг HTML. Багато веб-сайтів мають великі набори сторінок, які генеруються динамічно з основного структурованого джерела, наприклад бази даних. Дані однієї категорії зазвичай кодуються на схожих сторінках за допомогою загального сценарію або шаблону. У інтелектуальному аналізі даних програма, яка виявляє такі шаблони в певному джерелі інформації, витягує його вміст і переводить його в реляційну форму, називається

оболонкою (wrapper). Алгоритми генерації оболонки припускають, що вхідні сторінки індукційної системи оболонки відповідають загальному шаблону і що їх можна легко ідентифікувати за допомогою загальної схеми URL. Крім того, деякі напівструктуровані мови запитів даних, такі як XQuery та HTQL, можна використовувати для аналізу HTML-сторінок, а також для отримання та перетворення вмісту сторінки;

- парсинг DOM. За допомогою інтеграції повноцінного веб-браузера, програми можуть отримувати динамічний вміст, згенерований скриптами на клієнтській стороні. Ці браузерні елементи керування також парсять веб-сторінки в дерево DOM, на основі якого програми можуть отримувати частини сторінок. Такі мови, як Xpath, можна використовувати для аналізу результуючого дерева DOM;
- вертикальне агрегування. Є кілька компаній, які розробили спеціальні вертикальні збиральні платформи. Ці платформи створюють і контролюють безліч «ботів» для певних галузей без «людини в циклі» (без прямої участі людини) і без роботи, пов’язаної з конкретним цільовим сайтом. Підготовка передбачає створення бази знань для всієї вертикалі, а потім платформа автоматично створює ботів. Надійність платформи вимірюється якістю інформації, яку вона отримує (зазвичай кількістю полів), і її масштабованістю (як швидко вона може масштабуватися до сотень або тисяч сайтів). Ця масштабованість здебільшого використовується для націлювання на довгий хвіст сайтів, з яких звичайні агрегатори вважають складним або надто трудомістким збір вмісту;
- розпізнавання семантичної анотації. Сторінки, з яких необхідно зібрати дані, можуть містити метадані або семантичну розмітку й анотації, які можна використовувати для пошуку конкретних фрагментів даних. Якщо анотації вбудовані в сторінки, цю техніку можна розглядати як окремий випадок парсингу DOM. В іншому випадку анотації, організовані в семантичний рівень, зберігаються та керуються окремо від веб-сторінок, тому

веб-скрапери можуть отримати схему даних та інструкції з цього рівня перед тим, як збирати дані зі сторінки;

- аналіз веб-сторінки за допомогою комп'ютерного зору. Існують рішення з використанням машинного навчання та комп’ютерного зору (Computer Vision), які намагаються ідентифікувати та витягти інформацію з веб-сторінок шляхом візуальної інтерпретації сторінок, як це може зробити людина.

2.3 Проблема уніфікації великих текстових даних

Різні джерела великих даних надають дані в різних форматах і різної структури, проте найбільш ефективна обробка потребує однаково структурованих даних, що приводить нас до проблеми уніфікації такої структури. Це включає в себе як декларування формату серіалізації даних, так і строга нотація структури, в якій ці дані надаватимуться. Під час проведення аналізу було розглянуто найбільш популярні формати серіалізації даних, такі як: JSON, XML, CSV, Protobuf.

2.3.1 XML

XML (Extensible Markup Language) – це текстова мова розмітки, яка походить від стандартної узагальненої мови розмітки (SGML). Вона використовує символи розмітки, які в XML називають тегами, для визначення даних. На відміну від тегів HTML, які визначені завчасно та використовуються для опису відображення даних, теги XML описують дані та використовуються для зберігання й упорядкування даних, але не зазначають спосіб їх відображення.

Є три важливі характеристики XML, які роблять його популярним і корисним у різноманітних системах:

- XML є розширюваним. XML дозволяє вам створювати власні теги, які підходять для вашої програми, тобто краще описують дані для певної конкретної прикладної задачі;

- XML зберігає дані, але не представляє їх. XML дозволяє зберігати дані незалежно від того, як вони будуть використані надалі;
- XML є загальнодоступним стандартом. XML був розроблений організацією під назвою World Wide Web Consortium (W3C) і публічно доступний як відкритий та безкоштовний стандарт.

До переваг XML можна віднести:

- дозволяє передавати документи між системами та програмами. За допомогою XML можливо швидко обмінюватися даними між різними платформами;
- XML відокремлює дані від HTML;
- XML спрощує процес зміни платформи;
- дозволяє створювати визначені користувачем теги.

Проте цей формат має і недоліки, а саме:

- для XML потрібна спеціальна програма обробки. Для виокремлення даних з XML необхідно отримати XML-документ, використати XML DOM для перегляду документа, дістати значення та зберегти їх у змінних;
- синтаксис XML дуже схожий на інші альтернативні «текстові» формати передачі даних, що іноді викликає плутанину;
- немає підтримки внутрішньої типізації даних. В XML всі дані представлені у вигляді рядка;
- синтаксис XML іноді надлишково ускладнений.

Приклад вигляду XML-документа, який описує сутність «студента» представлений на рисунку 2.1.

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
        <student>
                <id>01</id>
                <name>Maksym</name>
                <lastname>Kuvichka</lastname>
        </student>
</root>
```

Рисунок 2.1 – Приклад вигляду XML-документа

2.3.2 CSV

CSV (Comma separated values – значення, розділені комою) добре підходить для зберігання табличних даних у форматі, зручному для читання та обробки. Він не підходить для зберігання структур ключ-значення або хеш-таблиць, як структури даних, що відрізняє його від інших розглянутих форматів.

CSV не стандартизований. Нотація RFC 4180 була спробою стандартизації формату, однак терміном "CSV" можуть називатись файли, які розмежовані іншим символом, аніж кома, такими як пробіл, знак табуляції або крапка з комою. Насправді раніше цей формат називався DSV (Delimiter separated values – значення, розділені обмежувачем), хоча саме термін CSV поширився значно більше в наші дні. А формат даних між роздільниками повністю визначається користувачем (наприклад, відсутній спосіб визначення дати).

Формат CSV передбачає необов'язковий рядок із заголовками полів, який записаний як перший рядок у файлі. Якщо він присутній, він містить назви полів для кожного значення в записі. Цей заголовок дуже корисний для орієнтації в даних і, за рекомендацією, він завжди має бути присутнім.

Формат CSV широко підтримується: бібліотеки для обробки CSV доступні для всіх популярних мов програмування. Популярна бібліотека «pandas» для обробки даних в мові програмування Python може зчитувати дані з CSV прямо в таблицю даних (називається Dataframe) за допомогою простої функції pd.read_csv. CSV – це ледь не єдиний формат серіалізації з усіх, який має якісну підтримку в програмах для роботи з електронними таблицями, такими як Microsoft Excel, Google Sheets або LibreOffice Calc оскільки CSV має табличну структуру.

CSV досить компактний. Однак людині може бути важко розрізнити стовпчики, особливо коли в файлі зберігається велика кількість записів або стовпців. Оскільки рядок з заголовками полів лише один і знаходиться він на початку файлу, користувачу потрібно постійно гортати файл вгору для

перегляду цих заголовків, а також, оскільки довжина значення в кожній клітинці довільна, навігація по горизонталі також викликає складнощі, бо щоб прочитати яке значення знаходиться в певній колонці, необхідно рахувати коми в цьому рядку.

Приклад вигляду CSV-документа, який описує сутність «студента» представлений на рисунку 2.2.

```
id,name,lastname
1,Maksym,Kuvichka
```

Рисунок 2.2 – Приклад вигляду CSV-документа

2.3.3 Protobuf

Protobuf - це бінарний протокол серіалізації, розроблений компанією Google. Оскільки він серіалізується у двійковій формі, дані в цьому форматі не є зручними для читання людиною. Типи даних, які може містити Protobuf, строго визначені і включають розповсюджені загальні типи, як рядки, цілі та дробові числа, дати.

Двійковий формат Protobuf допомагає йому бути дуже компактним та займати вкрай мало пам’яті. Він також використовує бітове кодування для типів даних, таких як цілі числа, щоб зменшити кількість байтів, коли числа невеликі. Формат швидкий для серіалізації та десеріалізації, підтримується багатьма популярними мовами програмування. Також його використовують сервіси, які найчастіше побудовані за принципом RPC, для обміну даними у мережі Інтернет для зменшення об’ємів трафіку. Недоліком формату є те, що його майже неможливо прочитати людині і він не має вбудованих засобів перевірки коректності даних. Також, при зберіганні великої кількості рядків переваги формату стають менш відчутними, так як він зберігає рядки у первинному вигляді.

2.3.4 JSON

JSON (JavaScript Object Notation) – це розповсюджений формат серіалізації даних, який використовує зрозумілий для людини текст для зберігання та передачі даних у вигляді об’єктів, що містять пари атрибут-значення і масиви та підтримується всіма популярними мовами програмування. JSON використовується для зберігання інформації в організованому та легкодоступному вигляді. Він пропонує зрозумілий для людини набір даних, до якого можна отримати логічний доступ. Структури даних в цьому форматі мають вигляд як у звичайних об'єктів в багатьох мовах програмування, хоча існують винятки. Важливо зазначити, що синтаксис JSON не підтримує коментарів та жоден з форматів дати. Ключі в парах ключ-значення об'єкта JSON завжди повинно бути рядком.

До переваг JSON можна віднести:

- простота використання. JSON API пропонує високорівневий фасад, який допомагає спростити розповсюджені випадки використання;
- продуктивність. JSON досить швидкий, оскільки потребує дуже мало пам’яті, що особливо підходить для великих об’ємів даних;
- доступність. Нотація JSON має відкритий вихідний код і є безкоштовною для використання;
- зручність. JSON-документ є чистим і зрозумілим людині, його можна легко прочитати;
- відсутність зайвих залежностей. Бібліотека JSON не потребує додаткових бібліотек для обробки.

Приклад вигляду JSON-документа, який описує сутність «студента» представлений на рисунку 2.3.

```
{
  "id": 1,
  "name": "Maksym",
  "lastname": "Kuvichka"
}
```

Рисунок 2.3 – Приклад вигляду JSON-документа

2.3.5 Уніфікація структури вихідних даних

Оскільки в цій роботі розглядається саме робота, з великими текстовими даними, необхідно, щоб формат серіалізації займав мало пам’яті, швидко оброблявся та був зручним для читання людиною. Результат аналізу представлений в Таблиці 2.1.

Таблиця 2.1 – Порівняння форматів серіалізації

	Займає мало пам’яті (за умови роботи з текстовими даними)	**Швидко обробляється**	**Зручний для читання**
XML	+	-	+
CSV	+	+	-
Protobuf	-	+	-
JSON	+	+	+

Як видно з Таблиці 2.1, для задачі, яка розглядається в даній роботі, найкраще підходить саме формат JSON, оскільки дані в цьому форматі займають менше пам’яті, ніж дані в інших форматах, більшість мов програмування підтримує його парсинг без використання додаткових бібліотек та виконують цей парсинг доволі швидко, а також JSON-документи зручні для читання людиною.

Не менш важливим кроком для уніфікації структури даних в форматі JSON є строге декларування полів в JSON-документах, щоб система під час обробки великих об’ємів текстових даних робила це ефективно та була позбавлена необхідності проведення додаткових перевірок та неоднозначності. В результаті була запропонована структура, зазначена в Таблиці 2.2.

Таблиця 2.2 – Поля уніфікованої структури вихідних даних в форматі JSON

Ім’я поля	**Тип даних в полі**	**Значення поля береться з джерела даних**	**Значення поля задається програмно**	**Опис**
title	string | null	+	-	Коротке загальне пояснення яка саме інформація знаходиться в цьому фрагменті даних.
link	string	-	+	Посилання на вихідне джерело конкретного фрагменту даних.
body	string	+	-	Повний текст фрагменту даних.

Продовження таблиці 2.2

Ім’я поля	**Тип даних в полі**	**Значення поля береться з джерела даних**	**Значення поля задається програмно**	**Опис**
author	string	-	+	Маркування з якого джерела фрагмент даних був вилучений. Зазвичай значення співпадає з назвою джерела даних.
category	string | null	+	-	Значення поля є довільним, слугує для загальної категоризації або агрегації даних за певним критерієм.

Закінчення таблиці 2.2

Ім’я поля	**Тип даних в полі**	**Значення поля береться з джерела даних**	**Значення поля задається програмно**	**Опис**
date	timestamp	-	+	Мітка часу коли дані були отримані. Необхідно для коректної аналітики та організації потоку текстових даних.

Висновки до розділу

У другому розділі було розглянуто теоретичне підґрунтя збору даних в загальному і виділено, що найбільш популярним методом збору текстових даних є web scraping, який адаптований до роботи з різнорідними інтернет-ресурсами.

Було проведено аналіз наявних форматів серіалізації даних, їх переваг та недоліків та обрано найбільш підходящий для роботи з надвеликими масивами текстових даних, а саме формат JSON. Також було запропоновано та описано уніфіковану структуру даних, яка забезпечує стандартизовану обробку даних, зібраних з різних джерел.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Використані технології

Для розробки програмного забезпечення було використано декілька технологій: для збору даних та парсингу HTML-сторінок – платформа Node.js, для організації потоку текстових даних – Apache Kafka, для попередньої обробки потоку даних – Apache Spark Streaming, для збереження зібраних даних – ElasticSearch, для відображення цих даних – Kibana. Для контейнеризації та оркестрації контейнерів було обрано Docker Compose.

3.1.1 Node.js

Node.js — платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript. Якщо раніше JavaScript застосовувався для обробки даних в браузері користувача, то Node.js надав можливість виконувати JavaScript-скрипти на сервері та відправляти користувачеві результат їхнього виконання. Платформа Node.js перетворила JavaScript на мову загального використання з великою спільнотою розробників.[20]

Node.js має наступні властивості:

- асинхронна одно-поточна модель виконання запитів;
- неблокуючий ввід/вивід;
- система модулів CommonJS;
- рушій JavaScript Google V8.

Для керування модулями використовується пакетний менеджер npm (node package manager).

Для забезпечення обробки великої кількості паралельних запитів у Node.js використовується асинхронна модель запуску коду, заснована на обробці подій в неблокуючому режимі та визначенні обробників зворотніх викликів (callback). Як способи мультиплексування з'єднань підтримується epoll, kqueue, /dev/poll і select. Для мультиплексування з'єднань використовується бібліотека libuv, для створення пулу потоків (thread pool)

задіяна бібліотека libeio, для виконання DNS-запитів у неблокуючому режимі інтегрований c-ares. Всі системні виклики, що спричиняють блокування, виконуються всередині пулу потоків і потім, як і обробники сигналів, передають результат своєї роботи назад через неіменовані канали (pipe).[19]

За своєю суттю Node.js схожий на фреймворки Perl AnyEvent, Ruby Event Machine і Python Twisted, але цикл обробки подій (event loop) у Node.js прихований від розробника і нагадує обробку подій у веб застосунку, що працює в браузері.[19]

3.1.2 Apache Kafka

Apache Kafka — це розподілене сховище подій і платформа для обробки потоків. Це система з відкритим кодом, розроблена Apache Software Foundation, написана мовами програмування Java та Scala. Проект має на меті забезпечити уніфіковану високопродуктивну платформу з низькою затримкою для обробки потоків даних у реальному часі. Kafka може підключатися до зовнішніх систем (для імпорту/експорту даних) через Kafka Connect і надає бібліотеки Kafka Streams для програм обробки потоків. Kafka використовує бінарний протокол на основі TCP, який оптимізований для підвищення ефективності та покладається на абстракцію «набір повідомлень», яка природним чином групує повідомлення разом, щоб зменшити накладні витрати на мережеву передачу. Це «призводить до більших мережевих пакетів, більших послідовних дискових операцій, безперервних блоків пам’яті, що дозволяє Kafka перетворювати бурхливий потік випадкових записів повідомлень у лінійні записи».

Kafka зберігає повідомлення ключ-значення, які надходять від будь-якої кількості процесів, які називаються виробниками. Дані можна розділити на різні «розділи» в межах різних «тем». Усередині розділу повідомлення строго впорядковуються за їх зміщенням (позицією повідомлення в розділі), індексуються та зберігаються разом із міткою часу. Інші процеси, які називаються «споживачами», можуть читати повідомлення з розділів. Для потокової обробки Kafka пропонує Streams API, який дозволяє писати програми

Java, які споживають дані з Kafka та записують результати назад до Kafka. Apache Kafka також працює із зовнішніми системами обробки потоків, такими як Apache Apex, Apache Beam, Apache Flink, Apache Spark, Apache Storm і Apache NiFi.

Kafka працює на кластері з одного або кількох серверів (які називаються брокерами), а розділи всіх тем розподілені між вузлами кластера. Крім того, розділи копіюються на кілька посередників. Ця архітектура дозволяє Kafka доставляти масивні потоки повідомлень у відмовостійкий спосіб і дозволила їй замінити деякі звичайні системи обміну повідомленнями, такі як Java Message Service (JMS), Advanced Message Queuing Protocol (AMQP) тощо. Починаючи з версії 0.11.0.0 Kafka пропонує транзакційні записи, які забезпечують одноразову обробку потоку за допомогою Streams API.

API споживача та виробника відокремлені від основних функцій Kafka за допомогою базового протоколу обміну повідомленнями. Це дозволяє писати сумісні рівні API на будь-якій мові програмування, які є такими ж ефективними, як Java API, що постачаються з Kafka.

3.1.3 Apache Spark Streaming

Spark Streaming — це розширення основного API Spark, яке забезпечує масштабовану, високопродуктивну, відмовостійку потокову обробку потоків даних в реальному часі. Дані можуть надходити з багатьох джерел, таких як Kafka, Kinesis або TCP-сокети, і можуть оброблятися за допомогою складних алгоритмів, виражених за допомогою функцій високого рівня, таких як map, reduce, join і window. Оброблені дані можна надсилати до файлових систем, баз даних і інформаційних панелей. Насправді можна навіть застосувати алгоритми машинного навчання Spark і алгоритми обробки графіків до потоків даних.

Рисунок 3.1 – Архітектура Spark Streaming

Всередині це працює наступним чином: Spark Streaming отримує потоки вхідних даних в реальному часі і розділяє дані на пакети, які потім обробляються механізмом Spark для створення кінцевого потоку результатів у пакетах.

Рисунок 3.2 – Візуалізація потоку даних в Spark Streaming

Spark Streaming забезпечує високорівневу абстракцію під назвою дискретизований потік або DStream, який представляє безперервний потік даних. DStream можна створювати або з вхідних потоків даних із таких джерел, як Kafka та Kinesis, або шляхом застосування високорівневих операцій до інших DStreams. Внутрішньо DStream представлений як послідовність RDD. Стійкий розподілений набір даних (RDD) - це базова абстракція в Spark. Представляє собою незмінний розділений набір елементів, з якими можна працювати паралельно. Цей клас містить основні операції, доступні на всіх RDD, такі як map, filter і persist.

3.1.4 ElasticSearch

ElasticSearch — пошуковий рушій, заснований на бібліотеці Lucene. Він надає розподілену систему повнотекстового пошуку з підтримкою кількох орендарів із веб-інтерфейсом HTTP та документами JSON без схем. ElasticSearch розроблено на Java та має подвійну ліцензію відповідно до

загальнодоступної ліцензії на стороні сервера та ліцензії Elastic, тоді як інші частини підпадають під пропрієтарну (доступну для джерела) ліцензію Elastic. Офіційні клієнти доступні на Java, .NET (C#), PHP, Python, Ruby та багатьох інших мовах. Згідно з рейтингом DB-Engines, ElasticSearch є найпопулярнішою корпоративною пошуковою системою.

ElasticSearch можна використовувати для пошуку будь-якого документа. Він забезпечує масштабований пошук, має пошук майже в реальному часі та підтримує мультиорендність. «ElasticSearch є розподіленим, що означає, що індекси можна розділити на сегменти (shards), і кожен сегмент може мати нуль або більше реплік. Кожен вузол містить один або більше сегментів і діє як координатор для делегування операцій правильному сегменту(ам). Перебалансування та маршрутизація виконуються автоматично». Зв’язані дані часто зберігаються в одному індексі, який складається з одного або кількох основних сегментів і нуля чи кількох реплік. Після створення індексу кількість первинних фрагментів не можна змінити.

ElasticSearch розроблено разом із механізмом збору даних і аналізу журналів Logstash, платформою аналітики та візуалізації Kibana, а також набором легких відправників даних під назвою Beats. Чотири продукти розроблено для використання як інтегроване рішення, яке називається «Elastic Stack». (Раніше «Стек ELK», скорочення від «ElasticSearch, Logstash, Kibana».)

ElasticSearch використовує Lucene і намагається зробити всі свої функції доступними через JSON і Java API. Він підтримує фасетування та просочування (форма проспективного пошуку), що може бути корисним для сповіщення, якщо нові документи відповідають зареєстрованим запитам. Інша функція, «шлюз», забезпечує тривале збереження індексу; наприклад, індекс можна відновити зі шлюзу в разі збою сервера. ElasticSearch підтримує GET-запити у реальному часі, що робить його придатним як сховище даних NoSQL, але йому бракує розподілених транзакцій.

В 2019 році компанія Elastic надала безкоштовний доступ до основних функцій безпеки Elastic Stack, включаючи TLS для зашифрованого зв’язку,

файлову та власну область для створення та керування користувачами, а також контроль доступу на основі ролей для керування доступом користувачів до кластерних API та індекси. Відповідний вихідний код доступний за ліцензією «Elastic License». Крім того, ElasticSearch тепер пропонує SIEM і машинне навчання як частину пропонованих послуг.

Важливою концепцією в побудові ElasticSearch кластера є індекси. Вони описують дві концепції, важливі для організації зберігання та отримання даних, але різні за своєю суттю.

Отже, в першому визначенні сказано, що індекс схожий на базу даних у реляційній базі даних. Він має відображення, яке визначає кілька типів. Тобто, індекс — це певний тип механізму організації даних, який дозволяє користувачеві розділяти дані певним чином.

В другому визначенні «індекса» сказано, що індекс — це логічний простір імен, який відображається на один або кілька основних фрагментів і може мати нуль або більше сегментів-реплік. Ця концепція стосується реплік і сегментів, механізму, який ElasticSearch використовує для розподілу даних по всьому кластеру. Під усіма індексами, типами та документами ElasticSearch має десь зберігати дані. Ця функція зберігається в сегментах, які є основними (Primary) або репліками (Replica). Кожен індекс налаштовано для певної кількості основних і реплік сегментів.

Загалом, індекси організовують дані логічно, але вони також організовують дані фізично через базові фрагменти.

Також сховище ElasticSearch в поєднанні з плагіном Kibana зарекомендували себе як зручне та корисне рішення для зберігання великих масивів даних в роботі [18].

3.1.5 Kibana

Kibana — це плагін з відкритим кодом для візуалізації даних з ElasticSearch. Він забезпечує можливості візуалізації змісту проіндексованого

кластером ElasticSearch. Користувачі можуть створювати гістограми, лінійні і точкові діаграми, або діаграми і карти на великих обсягах даних.

Поєднання компонент ElasticSearch, Logstash, і платформи Kibana, називають «Elastic stack» (колишня назва — англ. ELK stack). Воно доступне як окремий продукт або послуга. Logstash забезпечує входовий потік для зберігання і пошуку в Elastic, а платформи Kibana отримує дані для візуалізації, зокрема дані візуалізуються на інформаційних дошках.

3.1.6 Docker Compose

Docker Compose — це інструмент для визначення та запуску багатоконтейнерних Docker-програм. Docker Compose використовує YAML-файл для налаштування сервісів програми. Потім за допомогою однієї команди створюються та запускаються всі сервіси з конфігурації.

Compose працює в усіх середовищах: production, staging, development, testing, а також у робочих процесах CI. Він також містить команди для керування всім життєвим циклом програми:

- запуск, зупинка та відновлення сервісів;
- перегляд статусів запущених сервісів;
- вивід логів запущених сервісів;
- виконання одноразової команди в сервісі.

Основні функції Compose, які роблять його ефективним:

- можливість мати кілька ізольованих середовищ на одному хості;
- збереження даних volume-ів після створення контейнерів;
- перестворення лише тих контейнерів, які були змінені;
- підтримка змінних та переміщення композиції між середовищами.

3.2 Архітектура програмного забезпечення

Архітектура програмного забезпечення представлена на рисунку 3.3.

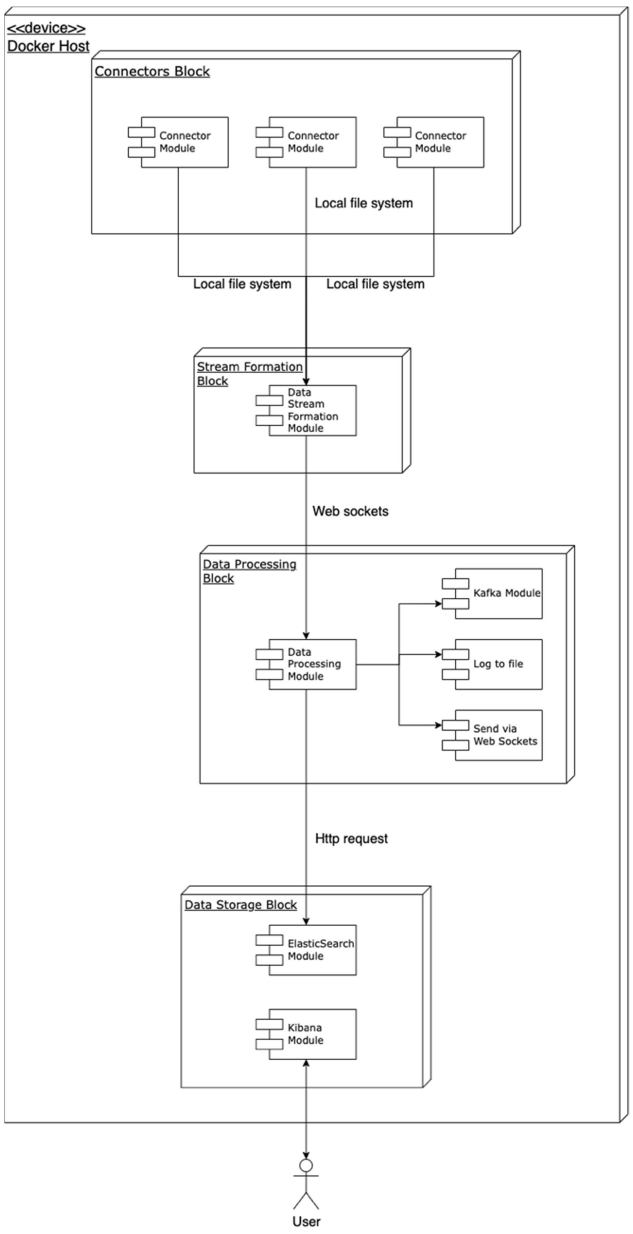


Рисунок 3.3 — Архітектура програмного забезпечення збору надвеликих масивів текстових даних

Source – зовнішнє джерело даних (RSS-стрічка, новинний ресурс, телеграм-канал тощо).

Connector – модуль, який відповідає за збір даних з зовнішнього джерела. Кожен Connector має свої власні налаштування, в яких вказаний перелік джерел, з яких він збирає дані, періодичність збору цих даних та певні, специфічні для цих джерел налаштування. Наприклад, для RSS-стрічок – це посилання на стрічку та CSS-selector, який допомагає дістати текст статті з відповідної HTML-сторінки; для Talagram-каналу – це дані для входу в спеціально створений акаунт, який має бути підписаний на ці канали, та

унікальні ідентифікатори каналів, з який проводиться збір даних. Кожен Connector має одне або декілька джерел, з яких відбувається збір даних. Цей модуль реалізований за допомогою Node.js, бо ця платформа гарно оптимізована під велику кількість асинхронних запитів і не потребує великої кількості ресурсів для роботи. Після збору даних, кожен Connector модуль вказує з якого типу джерела взятий кожен фрагмент даних та потім надсилає дані на вхід до Data Stream Formation модуля. Методи модуля Connector наведені в Таблиці 3.1.

Таблиця 3.1 – Опис методів модуля Connector

Назва метода	**Аргументи метода**	**Значення, що повертається**	**Опис метода**
main	-	-	Точка входу в модуль Connector. Запускає роботу основного функціоналу модуля – збір даних з обраних джерел з заданим інтервалом. В залежності від типів обраних джерел (RSS-стрічка, Telegram-канал, тощо), викликає інші методи для безпосередньо отримання даних.
getEnv	envVariable: string	envVariableValue: any	Метод для доступу до змінних середовища. Приймає в аргументи ім’я змінної середовища і повертає відповідне значення.

Продовження таблиці 3.1

Назва метода	**Аргументи метода**	**Значення, що повертається**	**Опис метода**
getTgUser	-	user: Promise<object>	Метод для отримання інформації про спеціально зареєстрованого користувача в Telegram. Використовується для отримання списку каналів, на які цей користувач підписаний, для збору даних з них.
tgApiCall	method: string; params: object; options: object	data: Promise<any>	Метод, який дозволяє робити запит на Telegram API та правильно обробляти помилки при запитах. Приймає аргументи такі аргументи: - method – назва сутності, до якої нам треба отримати доступ; назва точки доступу на Telegram API; - params – параметри, які передаються разом з запитом; - options – налаштування для запиту.

Продовження таблиці 3.1

Назва метода	**Аргументи метода**	**Значення, що повертається**	**Опис метода**
sendCode	phone: string	request: Promise<object>	Метод для відправки зареєстрованому в Telegram користувачу повідомлення з кодом підтвердження авторизації. Використовується для логіну в Telegram для проходження другого фактора авторизації.
signIn	code: string; phone: string; phone_code_hash: string	authResult: Promise<object>	Метод для запиту на авторизацію в Telegram. Приймає такі аргументи: - code – код підтвердження, який прийшов в повідомленні спеціально зареєстрованому в Telegram користувачу; - phone – номер телефону спеціально зареєстрованого в Telegram користувача; - phone_code_hash – системне значення для перевірки правильності коду підтвердження.

Продовження таблиці 3.1

Назва метода	**Аргументи метода**	**Значення, що повертається**	**Опис метода**
getPassword	-	passConfig: Promise<object>	Метод для отримання інформації про алгоритм шифрування паролю користувача та його налаштувань.
checkPassword	passConfig: object	authResult: Promise<object>	Метод для перевірки правильності введеного паролю від відповідного Telegram акаунта.
login	-	-	Метод для обробки логіки авторизації користувача в Telegram. Надсилає повідомлення з кодом підтвердження, перевіряє другий фактор авторизації та відповідає за обробку помилок в процесі авторизації.
getChannelData	channelConfig: object	channelData: Promise<object>	Метод для отримання даних з обраного Telegram-каналу. Отримує записи в цьому каналі, їх тексти та метадані.
logout	-	authResult: Promise<object>	Метод для деавторизації користувача в Telegram.

Продовження таблиці 3.1

Назва метода	**Аргументи метода**	**Значення, що повертається**	**Опис метода**
fetchTgdata	-	postsData: Promise<object[]>	Метод відповідає за отримування даних з обраних Telegram-каналів. Обробляє авторизацію спеціально зареєстрованого в Telegram користувача, отримує список обраних для збору даних каналів та отримує дані з цих каналів.
fetchRssData	-	postsData: Promise<object[]>	Метод відповідає за отримування даних з обраних RSS-стрічок та відповідних веб-ресурсів. Виконує отримання обраних джерел для збору даних, отримання тіла RSS-стрічки в форматі XML-файлу, який має метадані веб-ресурсу та масив його статей. Кожен запис про статтю містить метадані про відповідну статтю та посилання на веб-сторінку з повним текстом статті.

Закінчення таблиці 3.1

Назва метода	**Аргументи метода**	**Значення, що повертається**	**Опис метода**
getArticleBody	url: string; cssSelector: string	articleBody: Promise<object>	Метод для парсингу веб-сторінки, посилання на яку містилось в RSS-стрічці. Ця веб сторінка містить повний текст повідомлення, яке нам необхідно отримати, тому метод отримує в аргументи посилання на цю сторінку та відповідний CSS-selector для правильного парсингу і отримання лише необхідних нам даних. Використовує бібліотеку cheerio, яка аналізує розмітку веб-сторінки та надає API для маніпулювання отриманою структурою даних – для примінення CSS-selector.

Кожен модуль Connector має власний файл з налаштуваннями джерел, з яких він збиратиме дані. Цей файл задається в форматі JSON (JavaScript Object Notation) – текстовому форматі обміну даними між комп'ютерами. За рахунок своєї лаконічності в порівнянні з XML, формат JSON може бути більш придатним для серіалізації складних структур. Якщо говорити про веб-застосунки, у такому ключі він доречний у задачах обміну даними як між браузером і сервером (AJAX), так і між самими серверами (програмні HTTP-

інтерфейси). Формат JSON так само добре підходить для зберігання складних динамічних структур у реляційних базах даних або файловому кеші. Файл з налаштуваннями містить окремі поля для різних видів джерел: поле rssSources відповідає за опис джерел типу RSS-стрічок, а tgSources – за опис джерел типу Telegram-каналів. В полі tgSources задається масив унікальних ідентифікаторів каналів, з яких збиратимуться дані, в той час як в полі rssSources зберігається масив об’єктів, кожен з яких містить назву джерела, посилання на його RSS-стрічку та спеціальний CSS-selector для коректного збору оригінального тексту статті при парсингу її веб-сторінки. Приклад файлу налаштувань джерел для модуля Connector показано на рисунку 3.4.

Рисунок 3.4 – Приклад файлу з налаштуваннями джерел для збору даних для модуля Connector

RSS-стрічка – це родина XML-форматів, що використовується для публікації та постачання інформації, що часто змінюється, наприклад, нових записів в блозі, заголовків новин, анонсів статей, зображень, аудіо і відео матеріалів. Документ в стандарті RSS складається з повного або часткового тексту і метаданих. Наприклад, теги першого рівня <title>, <link>, <description>

- метадані джерела даних, а кожен тег <item> містить інформацію про статтю на цьому веб-ресурсі. Так, ми можемо отримати заголовок статті з тегу <title>, посилання на веб-сторінку зі статтею з тегу <link>, дату публікації статті з тегу <pubDate>, тощо. Через певний інтервал часу, різний для різних ресурсів, ця стрічка оновиться і матиме вже новий список статей для збору.

```
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0" xmlns:atom="http://www.w3.org/2005/Atom">
<channel>
<atom:link href="https://lb.ua/rss/ukr/rss.xml" rel="self" type="application/rss+xml"/>
<title>LB.ua</title>
<link>https://lb.ua</link>
<description>LB.ua</description>
<language>ru</language>
<webMaster>info@lb.ua</webMaster><pubDate>Fri, 28 Oct 2022 05:09:01 +0300</pubDate><lastBuildDate>Fri, 28 Oct 2022 05:09:01 +0300</lastBuildDate>
<item>
        <guid>https://lb.ua/world/2022/10/28/533987_latviya_vidpravila_ukrainu_dizelni.html</guid>
        <pubDate>Fri, 28 Oct 2022 05:03:00 +0300</pubDate>
        <title>Латвія відправила в Україну дизельні генератори і продукти</title>
        <link>https://lb.ua/world/2022/10/28/533987_latviya_vidpravila_ukrainu_dizelni.html</link>
        <category>Світ</category>
        <author>info@lb.ua (LB.ua)</author>
        <enclosure url="https://i.lb.ua/125/03/635b1546957b5.jpeg" type="image/jpeg" length="453783"/>
        <description>&lt;p&gt;Дизельні генератори будуть працювати у Львові.&lt;/p&gt;</description>
</item>
<item>
        <guid>https://lb.ua/news/2022/10/28/534018_ponedilok_kiieva_priidut_cheski.html</guid>
        <pubDate>Fri, 28 Oct 2022 04:51:00 +0300</pubDate>
        <title>У понеділок до Києва приїдуть чеські міністри, йтиметься про відновлення України та біженців</title>
        <link>https://lb.ua/news/2022/10/28/534018_ponedilok_kiieva_priidut_cheski.html</link>
        <category>Політика</category>
        <author>info@lb.ua (LB.ua)</author>
        <enclosure url="https://i.lb.ua/015/23/635b26350494b.png" type="image/jpeg" length="332038"/>
        <description>&lt;p&gt;До української столиці вирушить приблизно третина уряду Петра Фіали. &lt;/p&gt;</description>
</item>
<item>
        <guid>https://lb.ua/world/2022/10/28/534019_kiievi_lvovi_visadyat_tyulpani_z.html</guid>
        <pubDate>Fri, 28 Oct 2022 04:44:00 +0300</pubDate>
        <title>У Києві та Львові висадять тюльпани з Нідерландів</title>
        <link>https://lb.ua/world/2022/10/28/534019_kiievi_lvovi_visadyat_tyulpani_z.html</link>
        <category>Світ</category>
        <author>info@lb.ua (LB.ua)</author>
        <enclosure url="https://i.lb.ua/014/59/635b28301e053.jpeg" type="image/jpeg" length="86724"/>
        <description>&lt;p&gt;Вантажівка з цибулинами тюльпанів вже вирушила до України.&lt;/p&gt;</description>
</item>
```

Рисунок 3.5 – Приклад вигляду RSS-стрічки

Data Stream Formation – модуль, для формування потоку текстових даних. Він реалізований на платформі Apache Kafka, яка підтримує роботу з декількома вхідними джерелами і формує один суцільний потік текстових даних. Завдяки тому, що Kafka працює на кластері серверів, вона сама відповідає за відмовостійкість своїх вузлів та рівномірний розподіл навантаження між ними.

Data Processing Module – модуль, який відповідає за логіку обробки потоку текстових даних. В цьому модулі відбувається фільтрація даних, їх форматування в залежності від типу джерела, з якого ці дані прийшли, приведення всіх даних до уніфікованого формату тощо. Цей модуль реалізований за допомогою платформи Apache Spark Streaming, перевагою якої є можливість розпаралелити обчислення, що значно пришвидшує обробку потоку надвеликих масивів даних. Також зважаючи на те, що модулі Data

Processing та Data Stream Formation реалізовані за допомогою платформ від Apache, вони досить просто і зручно можуть між собою з’єднуватись.

В залежності від конфігурації користувача, вихідний потік форматованих і уніфікованих текстових даних можна направити на запис в ElasticSearch, відправляти по Web Sockets, записувати в файли в локальне сховище або направити в Kafka для інтеграції з будь-якими популярними платформами для потокової обробки даних. Методи модуля Data Processing Module наведені в Таблиці 3.2.

Таблиця 3.2 – Опис методів модуля Data Processing Module

Назва метода	**Аргументи метода**	**Значення, що повертається**	**Опис метода**
streamJoin	-	-	Метод об’єднує декілька потоків текстових даних в один потік. Складність об’єднання декількох потоків в один в тому, що в кожен момент часу, стан набору даних недоконаний, тому важливо знайти перетин результатів виконання усіх потоків.
formatOutputStream	-	streams[]: Dataset<Row>	Цей метод формує вихідний потік даних, який можна направити, в залежності від конфігурації, в різні вихідні формати.

Закінчення таблиці 3.2

Назва метода	**Аргументи метода**	**Значення, що повертається**	**Опис метода**
sendToWebSockets	-	socketId: string	Метод направляє вихідний потік текстових даних на вивід та надсилає їх через Web Socket-и.
saveToElasticSearch	-	-	Метод направляє вихідний потік текстових даних на вивід та зберігає їх в сховище даних Elasticsearch.
saveToFileSystem	-	-	Метод направляє вихідний потік текстових даних на вивід та зберігає їх в локальну файлову систему.
sendToKafka	-	-	Метод направляє вихідний потік текстових даних на вивід та надсилає їх в сервіс Apache Kafka.

Для коректної роботи ElasticSearch необхідно створити відповідні індекси, які описують структуру даних, що надходить на кластер ElasticSearch. В цьому випадку було створено індекс news, структура якого показана на рисунку 3.6 та описана в Таблиці 3.3.

Name	Type	Format	Searchable	Aggregatable	Excluded	
_id	string		●	●		
_index	string		●	●		
_score	number					
_source	_source					
_type	string		●	●		
author	string		●			
author.keyword	string		●	●		
body	string		●			
body.keyword	string		●	●		
category	string		●			
category.keyword	string		●	●		
date	date		●	●		
ingestion_time	date		●	●		
link	string		●			
link.keyword	string		●	●		
title	string		●			

Рисунок 3.6 – Структура індекса news в ElasticSearch

Таблиця 3.3 – Опис структури індекса news в ElasticSearch

Назва поля	**Тип даних в полі**	**Опис поля**
_id	string	Унікальний ідентифікатор запису.
_index	string	Назва індексу, в якому зберігається цей запис. Для індекса news, значення цього поля буде news.
_type	string	Тип фрагмента даних, який зберігається. Для об’єктів статей, тип дорівнює «_doc».

Продовження таблиці 3.3

Назва поля	**Тип даних в полі**	**Опис поля**
author	string	Автор статті. Поле використовується для відслідковування співвідношення кількості даних, зібраних з кожного джерела.
body	string	Оригінальний текст статті.
category	string	Категорія, в якій написана стаття. Хоч на різних ресурсах і різні категорії статей і немає якогось стандарту категоризації новин, але все ж часто ці категорії повторюються від одного ресурса до іншого, тому ці дані також можна використовувати для аналітики. Наприклад, для відслідковування трендів в новинах, абощо.
date	date	Дата та час написання оригінальної статті.

Закінчення таблиці 3.3

Назва поля	Тип даних в полі	Опис поля
ingestion_time	date	Дата та час додавання цієї статті в ElasticSearch. Це поле використовується для фільтрації даних по датах або по часу.
link	string	Посилання на веб-сторінку або Telegram-канал оригінальної статті.
title	string	Заголовок статті.

Також приклад запису в ElasticSearch представлений на рисунку 3.7.

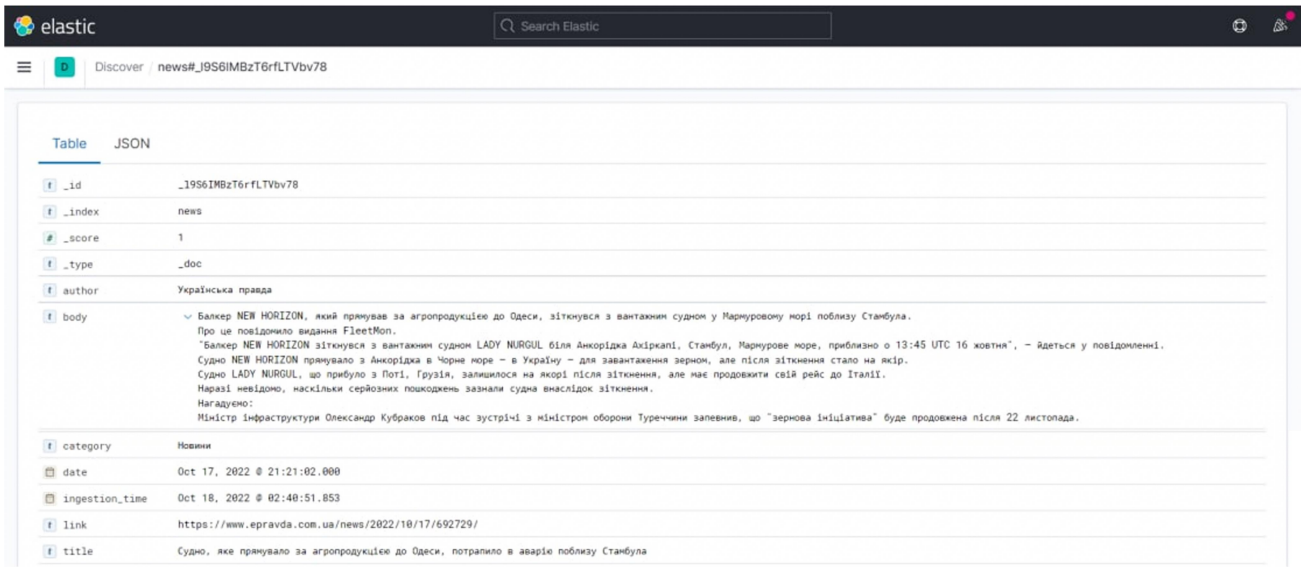


Рисунок 3.7 – Приклад запису в ElasticSearch

До ElasticSearch також налаштована Kibana для візуалізації та аналітики зібраних даних, побудови діаграм, звітів, тощо. На рисунку 3.8. представлений приклад дешборду Kibana з 4 діаграмами (зліва – направо, зверху – вниз):

- загальна кількість записів в сховищі;
- кількість збережених записів в певний момент часу;

- співвідношення кількості записів по джерелам, з яких дані були взяті;
- кількість записів, збережених в сховище, в певний момент часу із співвідношенням записів в залежності від джерела даних.

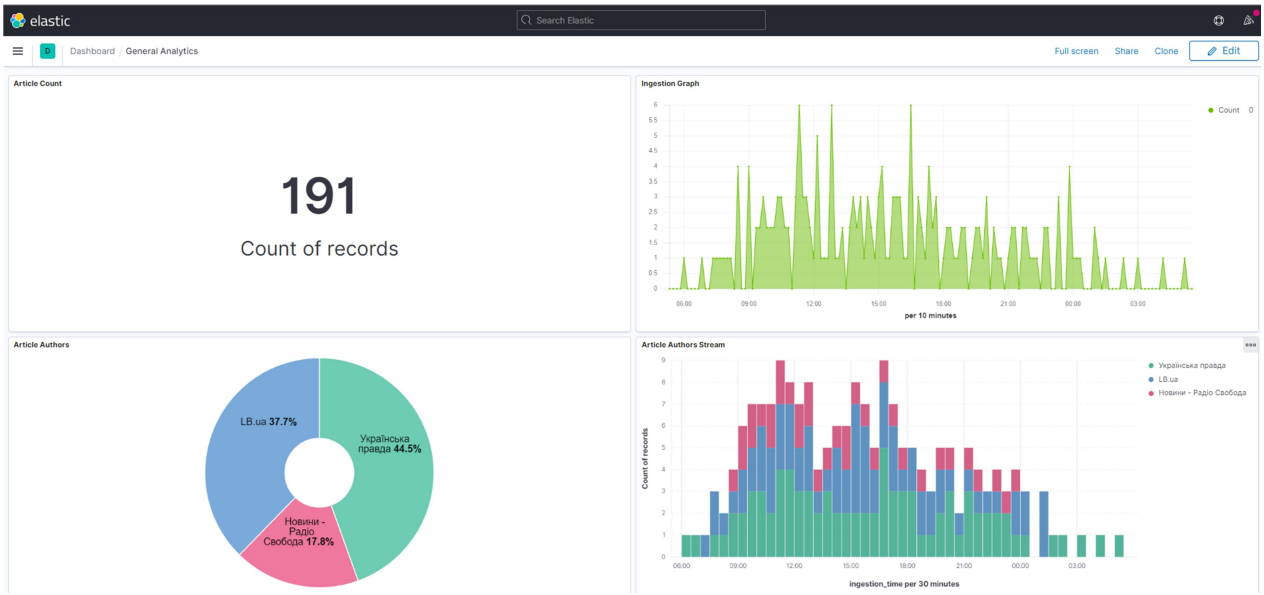


Рисунок 3.8 – Дешборд Kibana

Висновки до розділу

Третій розділ присвячений розробці програмного забезпечення для збору надвеликих масивів текстових даних. Також описано архітектуру програмного рішення для збору надвеликих масивів текстових даних з використанням технологій Node.js, Apache Spark Streaming, Apache Kafka, сховища даних ElasticSearch та платформи Kibana для візуалізації та аналітики. Розгортання програмного забезпечення організовано за допомогою інструмента контейнеризації Docker та засобу для оркестрації Docker Compose. Наведено специфікації функцій модулів Connector та Data Processing.

4 ОЦІНКА ЕФЕКТИВНОСТІ РОБОТИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Порівняння ефективності в різних режимах роботи

Для збору даних модуль Connector може працювати в 2 режимах роботи: Batch та Stream. В режимі Batch спочатку всі дані з усіх джерел, до яких під'єднаний модуль, отримуються та обробляються, а потім віддаються на вивід однією партією, а в режимі Stream кожен окремий запис віддається на вивід як тільки він був отриманий та оброблений.

Для проведення експерименту порівняння ефективності роботи модуля Connector в цих двох режимах було згенеровано однаковий набір вхідних даних та зібрано інформацію про час обробки кожного окремого запису за загалом всіх даних в кожному з режимів. В якості джерел даних були використані RSS-стрічки новинних ресурсів. Також, як вивід з модуля Connector був використаний запис в локальну файлову систему: для режиму Batch повний набір даних записувався в один файл, а в режимі Stream кожен запис записувався в окремий файл. В ході експерименту було проведено заміри часу збору даних в режимах Batch та Stream. На рисунку 4.1 представлено результат роботи в режимі Batch, а на Рис 4.2 – в режимі Stream.

```
Message 9 from source 0: 400.022ms      Message 42 from source 1: 1.376s        Message 53 from source 1: 1.753s        Message 98 from source 1: 2.107s
Message 19 from source 0: 430.217ms     Message 12 from source 1: 1.399s        Message 25 from source 1: 1.807s        Message 79 from source 1: 2.123s
Message 2 from source 2: 636.183ms      Message 15 from source 1: 1.413s        Message 43 from source 1: 1.803s        Message 94 from source 1: 2.123s
Message 0 from source 2: 661.104ms      Message 22 from source 1: 1.418s        Message 81 from source 1: 1.793s        Message 73 from source 1: 2.136s
Message 3 from source 2: 704.673ms      Message 21 from source 1: 1.424s        Message 54 from source 1: 1.812s        Message 77 from source 1: 2.152s
Message 6 from source 2: 726.3ms        Message 24 from source 1: 1.432s        Message 50 from source 1: 1.820s        Message 62 from source 1: 2.189s
Message 8 from source 2: 738.7ms        Message 31 from source 1: 1.443s        Message 47 from source 1: 1.836s        Message 86 from source 1: 2.196s
Message 7 from source 2: 756.957ms      Message 23 from source 1: 1.463s        Message 15 from source 2: 1.810s        Message 64 from source 1: 2.230s
Message 5 from source 2: 803.796ms      Message 29 from source 1: 1.475s        Message 92 from source 1: 1.832s        Message 80 from source 1: 2.230s
Message 9 from source 2: 818.857ms      Message 33 from source 1: 1.480s        Message 41 from source 1: 1.876s        Message 95 from source 1: 2.230s
Message 16 from source 2: 825.012ms     Message 30 from source 1: 1.500s        Message 10 from source 2: 1.851s        Message 96 from source 1: 2.245s
Message 11 from source 2: 850.293ms     Message 14 from source 1: 1.523s        Message 13 from source 2: 1.859s        Message 93 from source 1: 2.273s
Message 17 from source 2: 858.291ms     Message 26 from source 1: 1.524s        Message 18 from source 2: 1.875s        Message 76 from source 1: 2.289s
Message 19 from source 2: 896.321ms     Message 63 from source 1: 1.510s        Message 88 from source 1: 1.904s        Message 89 from source 1: 2.290s
Message 1 from source 2: 951.902ms      Message 36 from source 1: 1.543s        Message 84 from source 1: 1.914s        Message 97 from source 1: 2.292s
Message 4 from source 2: 961.18ms       Message 44 from source 1: 1.548s        Message 65 from source 1: 1.940s        Message 91 from source 1: 2.304s
Message 1 from source 1: 1.042s         Message 46 from source 1: 1.564s        Message 60 from source 1: 1.948s        Message 14 from source 0: 2.415s
Message 3 from source 1: 1.056s         Message 70 from source 1: 1.569s        Message 59 from source 1: 1.955s        Message 1 from source 0: 2.447s
Message 7 from source 1: 1.081s         Message 51 from source 1: 1.598s        Message 55 from source 1: 1.963s        Message 2 from source 0: 2.461s
Message 5 from source 1: 1.108s         Message 48 from source 1: 1.606s        Message 69 from source 1: 1.970s        Message 6 from source 0: 2.478s
Message 12 from source 2: 1.125s        Message 39 from source 1: 1.621s        Message 78 from source 1: 1.972s        Message 11 from source 0: 2.487s
Message 8 from source 1: 1.208s         Message 18 from source 1: 1.641s        Message 87 from source 1: 1.974s        Message 0 from source 0: 2.562s
Message 2 from source 1: 1.229s         Message 34 from source 1: 1.647s        Message 90 from source 1: 1.978s        Message 3 from source 0: 2.540s
Message 27 from source 1: 1.231s        Message 17 from source 1: 1.664s        Message 85 from source 1: 1.999s        Message 5 from source 0: 2.550s
Message 32 from source 1: 1.246s        Message 28 from source 1: 1.665s        Message 75 from source 1: 2.008s        Message 4 from source 0: 2.573s
Message 11 from source 1: 1.286s        Message 38 from source 1: 1.669s        Message 83 from source 1: 2.013s        Message 10 from source 0: 2.584s
Message 10 from source 1: 1.295s        Message 35 from source 1: 1.685s        Message 68 from source 1: 2.033s        Message 7 from source 0: 2.605s
Message 19 from source 1: 1.302s        Message 58 from source 1: 1.680s        Message 66 from source 1: 2.050s        Message 8 from source 0: 2.618s
Message 6 from source 1: 1.315s         Message 37 from source 1: 1.697s        Message 61 from source 1: 2.064s        Message 12 from source 0: 2.639s
Message 13 from source 1: 1.330s        Message 45 from source 1: 1.699s        Message 71 from source 1: 2.063s        Message 16 from source 0: 2.647s
Message 16 from source 1: 1.336s        Message 40 from source 1: 1.718s        Message 67 from source 1: 2.070s        Message 18 from source 0: 2.671s
Message 9 from source 1: 1.348s         Message 56 from source 1: 1.719s        Message 74 from source 1: 2.082s        Message 13 from source 0: 2.688s
Message 4 from source 1: 1.358s         Message 49 from source 1: 1.727s        Message 72 from source 1: 2.089s        Batch mode time: 2.776s
Message 0 from source 1: 1.380s         Message 57 from source 1: 1.729s        Message 99 from source 1: 2.085s
Message 20 from source 1: 1.376s        Message 52 from source 1: 1.748s        Message 82 from source 1: 2.100s
```

Рисунок 4.1 – Результати ефективності роботи модуля Connector в режимі Batch

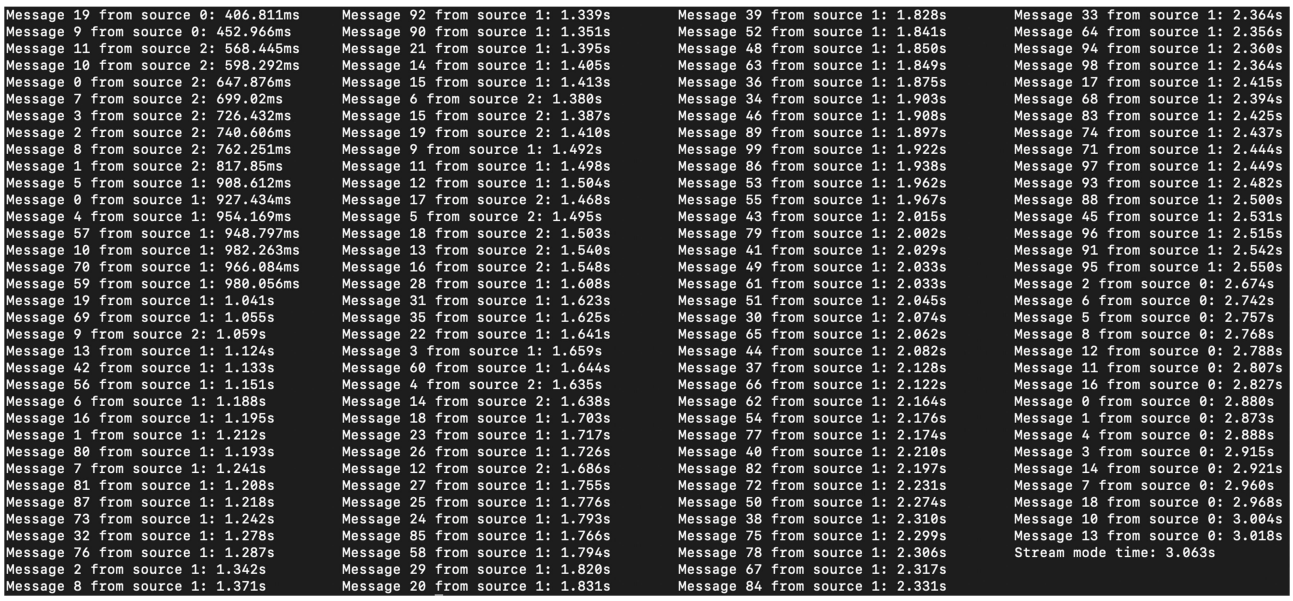


Рисунок 4.2 – Результати ефективності роботи модуля Connector в режимі Stream

Як видно з результатів, в режимі Stream загальний час збору всіх даних більший, ніж в режимі Batch на 10.3%. Це можна пояснити тим фактом, що в режимі Stream відбувається значно більше взаємодії з інтерфейсом виводу даних, в той час як в режимі Batch ця взаємодія відбувається лише один раз – в кінці обробки всіх даних.

Проте, якщо розглядати питання швидкості потрапляння кожного окремого запису на наступний етап обробки, перевага буде на стороні режиму Stream через те, що кожен запис потрапляє у вивід як тільки він був зібраний та підготований, в той час як в режимі Batch кожному запису доведеться чекати обробки всіх інших записів для того, щоб потрапити на наступний етап.

Для прикладу, в режимі Batch запис 9 з джерела 0 був повністю підготований до виводу через 400.022 мс після початку експерименту, проте на наступний етап обробки він потрапив лише через 2.776 с, бо йому довелось чекати закінчення обробки повної партії даних. А в режимі Stream запис 19 з джерела 0 був підготований до виводу через 406.811 мс і одразу був відправлений на вивід модуля Connector.

В результаті цього порівняння було виявлено, що в загальному випадку більш ефективно буде використовувати модуль Connector в режимі Stream для

збору надвеликих масивів текстових даних. Проте, в режимі Batch цей модуль працюватиме ефективніше за умови розгортання цього модуля на машині з файловою системою, яка не підтримує багато одночасних записів в файл або операція запису в файл є більш важкою для опрацювання, ніж сам процес збору даних.

4.2 Порівняння ефективності для різної кількості джерел

Кожен екземпляр модуля Connector може збирати дані з декількох джерел за умови, що ці джерела мають однаковий формат даних. Також важливим критерієм для системи збору надвеликих масивів текстових даних є час збору даних в залежності від навантаження. З цього випливає, що необхідно провести експеримент з визначення часу збору даних в залежності від кількості джерел вхідних даних.

Для цього експерименту було взято один екземпляр модуля Connector та поступово збільшувалось навантаження на нього за рахунок додавання все більшої кількості джерел для обробки. В якості джерел даних було взято від одної до десяти RSS-стрічок різних новинних ресурсів. Спочатку було обрано одне джерело, проведено 20 ітерацій збору даних та обраховано середній час збору даних для такої конфігурації. Потім кількість джерел покроково збільшувалась з інтервалом в одне джерело та проводилась така ж кількість ітерацій збору даних. Результати, отримані в ході цього експерименту представлені в Таблиці 4.1.

Таблиця 4.1 – Результати перевірки на навантаження модуля Connector

Кількість джерел вхідних даних	**Час повного збору даних**
1	0.508 с
2	1.516 с
3	1.824 с
4	2.046 с

Закінчення таблиці 4.1

Кількість джерел вхідних даних	**Час повного збору даних**
5	3.085 с
6	3.589 с
7	3.862 с
8	4.805 с
9	5.040 с
10	6.158 с

Для більшої наглядності отриманого результату, його представлено також у вигляді графіка на Рис 4.3.

Рисунок 4.3 – Графік залежності часу повного збору даних від кількості джерел вхідних даних

Як видно з графіка, час, витрачений на повний збір даних з усіх джерел, зростає лінійно відносно кількості використаних вхідних джерел. Це можна

пояснити тим, що кожен запис кожного джерела обробляється асинхронно, тому кількість вхідних джерел для одного екземпляра модуля Connector обмежена лише обчислювальною потужністю машини, на якій розгорнуто це програмне забезпечення. Також варто зазначити, що навіть для 10 вхідних джерел текстових даних, час обробки доволі малий, що також є гарним результатом для використання цього програмного рішення в сфері Big Data.

Висновки до розділу

У четвертому розділі описано експериментальну оцінку ефективності запропонованого рішення та проведено ряд експериментів збору текстових наборів даних.

Порівняння двох режимів роботи модуля Connector показало, що обидва режими є достатньо ефективними для вирішення задачі збору надвеликих масивів текстових даних. В режимі Batch програмне забезпечення відпрацювало швидше на 10.3%. Режим Stream є більш ефективним в загальному випадку, бо дозволяє більш продуктивно організовувати потік текстових даних, що позитивно впливає на загальний час роботи програмного забезпечення. Режим Batch в свою чергу краще підходить для певних специфічних джерел даних та для випадку розгортання цього програмного забезпечення на машинах зі старими версіями файлової системи.

Експеримент з перевірки навантаження на модуль Connector також дав позитивний результат і показав, що обрана технологія ефективно виконує задачу обробки великої кількості одночасних мережевих запитів та підходить для використання в сфері Big Data. Виявлено, що кількість вхідних джерел для одного екземпляра модуля Connector обмежена лише обчислювальною потужністю машини, на якій розгорнуто це програмне забезпечення. Встановлено, що модуль Connector виконує збір даних з 10 джерел за 6с, що є прийнятним результатом.

5 МАРКЕТИНГОВИЙ АНАЛІЗ СТАРТАП-ПРОЄКТУ

5.1 Опис ідеї проєкту

Спрямуванням стартап-проєкту є розробка програмного забезпечення, що дозволить користувачам легко розв’язувати задачу збору надвеликих масивів текстових даних. В основу програмного забезпечення покладено розроблені методи ефективного збору таких даних. У таблиці 5.1 наведено загальний опис проєкту.

Таблиця 5.1 — Опис ідеї стартап-проєкту

Зміст ідеї	**Напрямки застосування**	**Вигоди для користучача**
Програмне забезпечення автоматизованого збору надвеликих масивів текстових даних	Міграція текстових даних	Висока швидкість обробки масивів текстових даних
	Аналіз текстових даних з різних джерел	Централізація даних з різних джерел в одному місці

Результат проведеного аналізу ідеї проєкту наведено у таблиці 5.2

Таблиця 5.2 — Опис ідеї стартап-проєкту

Техніко-економічні характеристики ідеї	**Продукція конкурентів**			**W (слабка сторона)**	**N (нейтральна сторона)**	**S (сильна сторона)**
	Мій проєкт	Підхід нечіткого агента	HTwitt			
Організація потоку даних	+	-	+			X
Декілька джерел даних	+	-	-			X

Закінчення таблиці 5.2

Техніко-економічні характеристики ідеї	**Продукція конкурентів**			**W (слабка сторона)**	**N (нейтральна сторона)**	**S (сильна сторона)**
	Мій проєкт	Підхід нечіткого агента	HTwitt			
Масштабованість системи	+	+	+		X	
Висока швидкість обробки масивів текстових даних	+	-	+		X	
Дружність до нетехнічних користувачів	-	-	+	X		
Візуалізація результатів запитів	+	+	+		X	

5.2 Технологічний аудит ідеї проєкту

Реалізацію ідеї проєкту планується проводити на основі наявних технологій, які доступні у вигляді відкритого програмного забезпечення. Разом з тим, використання розроблених власних архітектурних рішень дозволить отримати конкурентну перевагу. У таблиці 5.3 розглянуто різні можливі технології реалізації проєкту.

Таблиця 5.3 — Технологічна здійсненність ідеї проєкту

Ідея проєкту	**Технології її реалізації**	**Наявність технологій**	**Доступність технологій**
Міграція, зберігання текстових даних	MongoDB	+	+
	Elasticsearch	+	+
Аналітичні запити до текстових даних	MongoDB	+	+
	Elasticsearch	+	+
Візуалізація результатів запитів	MongoDB Atlas	+	+
	Kibana	+	+
Обрана технологія реалізації ідеї проєкту: Elasticsearch/Kibana			

Висновок: технологічна реалізація продукту – можлива, вибрана технологія Elasticsearch/Kibana.

5.3 Аналіз ринкових можливостей запуску стартап-проєкту

Визначення конкуретного середовища та можливостей, які можна використати під час впровадження проєкту, а також ринкових ризиків та загроз, що можуть завадити реалізації проєкту, дозволить перебачити і спланувати можливі точки зростання проєкту. Врахування стану ринкового середовища дозволить визначити потенційні проблеми для розвитку проєкту. У таблиці 5.4 наведено стан ринку. У таблиці 5.5 наведено опис можливих клієнтів проекту. У таблиці 5.6 наведено аналіз можливих загроз для реалізації проекту зі сторони ринку, а у таблиці 5.7 узагальнено можливості, які сприятимуть швидкому зростанню компанії.

Таблиця 5.4 — Попередня характеристика потенційного ринку

Показники стану ринку	**Характеристика**
Кількість головних гравців, од	3
Загальний обсяг продаж, грн./ум.од	3 000 000 000
Динаміка ринку	зростає
Наявність обмежень для входу	Uptime сервісу 99.99%
Специфічні вимоги до стандартизації та сертифікації	немає
Середня норма рентабельності в галузі або по ринку, %	175%

Висновок: враховуючи кількість головних гравців по ринку, зростаючу динаміку ринку, невелику кількість конкурентів та середню норму рентабельності можна зробити висновок, що на даний момент, ринок для входження стартап-продукту є привабливим.

Таблиця 5.5 — Характеристика потенційних клієнтів стартап-проєкту

Потреба, що формує ринок	**Цільова аудиторія**	**Відмінності у поведінці цільових груп клієнтів**	**Вимоги споживачів до товару**
Потреба у аналізі великих даних	Бізнес в будь-якій галузі	Чим більший бізнес, тим більш йому необхідна така платформа	До продукції: - точність обробки даних - швидкодія - можливість встановлення на власних серверах

Закінчення таблиці 5.5

Потреба, що формує ринок	**Цільова аудиторія**	**Відмінності у поведінці цільових груп клієнтів**	**Вимоги споживачів до товару**
Потреба у зборі надвеликих масивів текстових даних	Розробники систем аналізу великих текстових даних	Великі компанії можуть спробувати створити свої засоби розробки	До постачальника: - технічна підтримка користувачів - впровадження нових можливостей у систему навчання користувачів системи

Таблиця 5.6 — Фактори загроз

Фактор	**Зміст загрози**	**Можлива реакція компанії**
Конкуренти	Наявність конкурентів котрі надають схожі рішення	Зменшення ціни на поставлену послугу; Розробка унікальних характеристик товару; Надання ліцензій на обслуговування
Кошти на розробку та підтримку продукту	Закінчення грошей та недостатнє фінансування	Залучення додаткових інвесторів, мотивація роботи на перспективу; Ітеративна розробка продукту задля покрокового виведення продукту на ринок та отримання відповіді користувачів

Закінчення таблиці 5.6

Фактор	**Зміст загрози**	**Можлива реакція компанії**
Вихід аналогу	Вихід аналогу даного товару може призвести до знецінення та безідейності даного товару	Вихід товару на ринок в коротші строки з не повною, але достатньою, функціональністю для зацікавлення усіх цільових аудиторій; Проведення рекламної компанії

Таблиця 5.7 — Фактори можливостей

Фактор	**Зміст можливості**	**Можлива реакція компанії**
Новий продукт	Вихід на ринок, Зменшення монополії, Надання нових рішень у сфері	Розробка нової функціональності; Вихід нової продукції на ринок; Надання різноманітних типів ліцензій в залежності від потреб користувача \ замовника.
Machine Learning	Компанії хочуть застосувати Machine Learning для більш ефективної попередньої обробки даних	Запровадження підходів Machine Learning у продукті для подальшої автоматизації збору даних
Зворотній зв’язок від користувачів	Можливість отримання необхідної інформації для вдосконалення продукту	Наявність вхідних даних та реакція на них з боку команди розробників задля задоволення потреб та бажань кінцевих користувачів системи кешування даних.

Таблиця 5.8 — Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	**В чому проявляється дана характеристика**	**Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)**
Тип конкуренції: чистий	Наявність великої кількості конкурентів	Ускладнене виявлення конкуретних переваг
Рівень конкурентної боротьби: міжнародний	Розміщення проєктів конкурентів	Підтримка англійської мови та мов локальних користувачів. Підтримка актуальної документації мовами користувачів. Можливість подорожувати для підтримки та запровадження продукту
Галузева ознака: внутрішньогалузева	Визначена спрямованість (інформаційні технології)	Існують інструменти і методи розробки, тестуваннння, впровадження
Конкуренція за видами товарів: між бажанням	Конкуренцію виграє той проект, що більше задовольняє клієнтів	Користувачеві потрібно донести конкурентні переваги
Характер конкурентних переваг: цінова	Користувач імовірно обере дешевший проект	Ціна має бути ринковою

Закінчення таблиці 5.8

Особливості конкурентного середовища	**В чому проявляється дана характеристика**	**Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)**
За інтенсивністю: марочна	Унікальне ім’я	Можливість досягти впізнаваності проекту шляхом просування

Таблиця 5.9 — Аналіз конкуренції в галузі за М. Портером

Складові аналізу	**Прямі конкуренти в галузі**	**Потенційні конкуренти**	**Постачальники**	**Клієнти**	**Товари-замінники**
	HTwitt	Підхід з нечіткими агентами	Oracle IBM	Бізнес в будь-якій галузі	MongoDB
Висновки	Середня інтенсивність конкурентної бортьби	Є можливості входу в ринок, які не використовуються.	Постачальники не диктують умови роботи на ринку.	Клієнти диктують умови роботи на ринку. Продукт повинен бути швидким, зручним, щоб затримати увагу користувача на собі.	Необхідно забезпечувати конкурентоспроможну швидкодію.

Отже, з огляду на конкурентну ситуацію на ринку рішень для збору надвеликих масивів текстових даних існує принципова можливість виходу на цей ринок. Аби цей проєкт був конкуретноспроможним, необхідно забезпечити швидкодію продукту, легкість та зручність використання для кінцевих користувачів. Вагомою перевагою також буде стабільність роботи програмного забезпечення.

5.4 Стратегія розвитку продукту

Результати аналізу факторів конкурентоспроможності наведено у таблиці 4.10. За визначеними факторами конкурентоспроможності проаналізовано сильні та слабкі сторони стартап-проекту у таблиці 4.11. SWOT-аналіз стартап-проекту (таблиця 4.12) є основою для розробки альтернативних стратегії впровадження на ринку (таблиця 4.13). У таблицях 4.14-4.18 описано ключові позиції стратегії розвитку продукту.

Таблиця 5.10 — Обґрунтування факторів конкурентоспроможності

Фактор конкурентоспроможності	**Обґрунтування**
Швидкість оброблення даних	Швидкість оброблення даних вища, ніж у конкурентів
Підтримка декількох різних джерел текстових даних	Більше джерел допомагає збирати дані в більшій кількості, на відміну від конкурентів, які строго прив’язані до обраного джерела даних
Формування потоку текстових даних	Гарантує сумісність з більшістю популярних платформ потокової обробки текстових даних

Таблиця 5.11 — Порівняльний аналіз сильних та слабких сторін програмного забезпечення Select&Collect

Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з запропонованим						
		-3	-2	-1	0	+1	+2	+3
Швидкість оброблення даних	18				X		X	
Підтримка декількох різних джерел текстових даних	20							X
Формування потоку текстових даних	16						X	

Таблиця 5.12 — SWOT аналіз стартап-проєкту

Сильні сторони (S):	Слабкі сторони (W):
- Швидкість оброблення даних - Підтримка декількох різних джерел текстових даних - Формування потоку текстових даних	- Слабка попередня обробка даних - Складний процес розгортання
Можливості (O):	**Загрози (T):**
- Популяризація - Вбудовані запити	- Невідомість бренду - Консервативність клієнтів

Таблиця 5.13 — Альтернативи ринкового впровадження стартап-проєкту

Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	**Ймовірність отримання ресурсів**	**Строки реалізації**
Безкоштовне надання певного функціоналу у користування споживачам на обмежений термін	Головний ресурс – люди, наявний	1-2 місяці
Презентація товару на хакатонах й інших ІТ заходах	Ресурс – час та гроші для участі, наявні	3-6 місяців
Продаж прав на технологію більшій компанії що займається цим же напрямом	Ресурс – люди та гроші, наявні	1-6 місяців

5.5 Розроблення ринкової стратегії проєкту

Таблиця 5.14 — Вибір цільових груп потенційних споживачів

Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
Бізнес в будь-якій галузі	Так	Є	Висока	Складно
Розробники платформ аналізу надвеликих масивів текстових даних	Так	Є	Низька	Середня
Які цільові групи обрано: 2				

Відповідно до проведеного аналізу можна зробити висновок, що підходящою цільовою групою для розповсюдження даного програмного продукту є представники бізнесу, що зацікавлені в розширенні своєї аудиторії чи побудові більш довгострокових взаємозв’язків з нею, а також розробники систем обробки потоків текстових даних. Відповідно до стратегії охоплення ринку збуту товару обрано стратегію таргетованого маркетингу, щоб зосередитись на якісній роботі основного функціоналу, саме в якому і зацікавлені користувачі.

Таблиця 5.15 — Визначення базової стратегії розвитку

Обрана альтернатива розвитку проєкту	**Стратегія охоплення ринку**	**Ключові конкурентоспроможні позиції відповідно до обраної альтернативи**	**Базова стратегія розвитку**
Створення платформи, що дозволяє збирати не тільки текстові, а й інші види даних.	Проведення реклами, освітлення унікальної функціональності через інтернет-ресурси та інші канали, контакт напряму з споживачами.	Зниження ступеню замінності товару; Прихильність клієнтів; Відмітний функціонал системи;	Стратегія диференціації

Таблиця 5.16 — Визначення базової стратегії конкурентної поведінки

Чи є проєкт «першопрохідцем» на ринку	**Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?**	**Чи буде компанія копіювати основні характеристики товару конкурента, які?**	**Стратегія конкурентної поведінки**
Ні, оскільки є товари-аналоги, але вони не є настільки універсальними	Так, ціль компанії знайти нових споживачів	Ні, такої задачі немає	Стратегія заняття конкурентної ніші

Таблиця 5.17 — Визначення стратегії позиціонування

Вимоги до товару цільової аудиторії	**Базова стратегія розвитку**	**Ключові конкурентоспроможні позиції власного стартап-проєкту**	**Вибір асоціацій, які мають сформувати комплексну позицію власного проєкту**
Цілодобова доступність	Стратегія спеціалізації	Швидкість оброблення даних	Просто, надійно, таргетовано
Надійний захист даних			
Зручні інструменти роботи		Підтримка декількох різних джерел текстових даних	
Швидкодія		Формування потоку текстових даних	

Відповідно до проведеного аналізу можна зробити висновок, що стартап-компанія вибирає як базову стратегію розвитку – стратегію диференціації, як базову стратегію конкурентної поведінки – стратегію заняття конкурентної ніші.

5.6 Розроблення маркетингової програми стартап-проєкту

Таблиця 5.18 — Визначення ключових переваг концепції потенційного товару

Потреба	**Вигода, яку пропонує товар**	**Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)**
Формування потоку текстових даних.	Швидкість оброблення даних.	Підтримка декількох різних джерел текстових даних.

Таблиця 5.19 — Опис трьох рівнів моделі товару

Рівні товару	**Сутність та складові**		
1. Товар за задумом	Програмне забезпечення автоматизованого збору надвеликих масивів текстових даних		
2. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх/Тл/Е/Ор
	Вартість	М	Вр
	Відповідність моді	М	Тх
	Стабільність роботи	Нм	Тл
	Якість: Стабільність роботи системи		
	Пакування: розповсюдження за моделлю SaaS, On Premise, документація		
	Марка: Select&Collect		

Закінчення таблиці 5.19

Рівні товару	**Сутність та складові**
3. Товар із підкріпленням	До продажу: наявна повна документація, акції на придбання декількох ліцензій, знижки для певних сегментів на покупку товару
	Після продажу: додаткова підтримка спеціалістів налаштування
За рахунок чого потенційний товар буде захищено від копіювання: захист інтелектуальної власності, патент на уніфіковану структуру даних	

Таблиця 5.20 — Визначення меж встановлення ціни

Рівень цін на товари-замінники	**Рівень цін на товари-аналоги**	**Рівень доходів цільової групи споживачів**	**Верхня та нижня межі встановлення ціни на товар/послугу**
260000 грн.	180000 грн.	Не важливо	180000 – 260000 грн.

Таблиця 5.21 — Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	**Функції збуту, які має виконувати постачальник товару**	**Глибина каналу збуту**	**Оптимальна система збуту**
Закупівля товарів та послуг програмного забезпечення	Розгортання програмного забезпечення. Навчання персоналу для використання ПЗ	1	Відділ збуту - користувач

Таблиця 5.22 — Концепція маркетингових комунікацій

Специфіка поведінки цільових клієнтів	**Канали комунікацій, якими користуються цільові клієнти**	**Ключові позиції, обрані для позиціонування**	**Завдання рекламного повідомлення**	**Концепція рекламного звернення**
Збір надвеликих масивів текстових даних	Інтернет-ресурси, конференції	1. Швидкість оброблення дани 2. Підтримка декількох різних джерел текстових дани 3. Формування потоку текстових даних.	1. Донести до споживача наявність та назву продукту 2. Донести конкурентні переваги 3. Переконати у необхідності придбання продукту	Ви обираєте – ми збираємо

Як результат було створено ринкову (маркетингову) програму, що включає в себе визначення ключових переваг концепції потенційного товару, опис моделі товару, визначення меж встановлення ціни, формування системи збуту та концепцію маркетингових комунікацій.

Висновки до розділу

В п’ятому розділі описано стратегії та підходи з розроблення стартап-проєкту, визначено наявність попиту, динаміку та рентабельність роботи ринку,

як висновок було вказано що існує можливість ринкової комерціалізації проєкту. Розглянувши потенційні групи клієнтів, бар’єри входження, стан конкуренції та конкурентоспроможність проєкту було встановлено що проєкт є перспективним. Розглянуто та вибрано альтернативу впровадження стартап-проєкту та доведено доцільність подальшої імплементації проєкту.

ВИСНОВКИ

В магістерській роботі було розроблено та описано програмне забезпечення для збору надвеликих масивів текстових даних. Програмна реалізація була створена за допомогою високорівневих мов програмування JavaScript та Python.

На першому етапі дослідження було проведено огляд предметної області, описано що таке Big Data і специфіку роботи з нею. Було розглянуто та проаналізовано технології, які використовуються для роботи надвеликими масивами текстових даних. Також було проведено аналіз конкурентів, виокремлені їхні сильні та слабкі сторони та розглянуто технології, які використовувались для їхньої реалізації. Виявлено, що наразі доступних та універсальних рішень для організації процесу збору надвеликих масивів текстових даних не існує. В результаті було сформовано список завдань, які необхідно вирішити в рамках цієї роботи.

У другому розділі було розглянуто теоретичне підґрунтя збору даних в загальному і виділено, що найбільш популярним методом збору текстових даних є web scraping, який адаптований до роботи з різнорідними інтернет-ресурсами. Було проведено аналіз наявних форматів серіалізації даних, їх переваг та недоліків та обрано найбільш підходящий для роботи з надвеликими масивами текстових даних, а саме формат JSON. Також було запропоновано та описано уніфіковану структуру даних, яка забезпечує стандартизовану обробку даних, зібраних з різних джерел.

Третій розділ присвячений розробці програмного забезпечення для збору надвеликих масивів текстових даних. Також описано архітектуру програмного рішення для збору надвеликих масивів текстових даних з використанням технологій Node.js, Apache Spark Streaming, Apache Kafka, сховища даних ElasticSearch та платформи Kibana для візуалізації та аналітики. Розгортання програмного забезпечення організовано за допомогою інструмента контейнеризації Docker та засобу для оркестрації Docker Compose.

Для перевірки ефективності розробленого програмного забезпечення було проведено ряд необхідних експериментів, проаналізовано та пояснено їхні результати та виявлено, що використання цього програмного рішення в рамках предметної області Big Data є цілком доречним та ефективним. В режимі Batch програмне забезпечення відпрацювало швидше на 10.3%. Режим Stream є більш ефективним в загальному випадку, в той час як режим Batch в свою чергу краще підходить для певних специфічних джерел даних та для випадку розгортання цього програмного забезпечення на машинах зі старими версіями файлової системи. Виявлено, що кількість вхідних джерел для одного екземпляра модуля Connector обмежена лише обчислювальною потужністю машини, на якій розгорнуто це програмне забезпечення. Встановлено, що модуль Connector виконує збір даних з 10 джерел за 6с, що є прийнятним результатом.

Науковою новизною роботи є створення уніфікованого структури даних для джерел великих текстових даних різної природи, що включає зберігання мітки часу та джерела даних, а також декларування строгої структури.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan, The rise of ‘big data’ on cloud computing: Review and open research issues, Inform. Syst., vol. 47, pp. 98–115, 2015.

2) J. H. Yu and Z. M. Zhou, Components and development in big data system: A survey, J. Electr. Sci. Technol., vol. 17, no. 1, pp. 51–72, 2019.

3) S. Kumar and K. K. Mohbey, A review on big data based parallel and distributed approaches of pattern mining, J. King Saud Univ. – Comput. Inform. Sci., doi: 10.1016/j.jksuci.2019.09.006.

4) Y. N. Liu, N. Li, X. Zhu, and Y. Qi, How wide is the application of genetic big data in biomedicine, Biomed. Pharmacother., vol. 133, p. 111074, 2021.

5) V. Subramaniyaswamy, V. Vijayakumar, R. Logesh, and V. Indragandhi, Unstructured data analysis on big data using map reduce, Procedia Comput. Sci., vol. 50, pp. 456–465, 2015

6) S. Maitrey and C. K. Jha, MapReduce: Simplified data analysis of big data, Procedia Comput. Sci., vol. 57, pp. 563–571, 2015.

7) M. A. Beyer and D. Laney, The importance of ‘big data’: A definition, Stamford, CT, USA: Gartner, G00235055, 2012.

8) L. Rabhi, N. Falih, A. Afraites, and B. Bouikhalene, Big data approach and its applications in various fields: Review, Procedia Comput. Sci., vol. 155, pp. 599–605, 2019.

9) O’Driscoll, J. Daugelaite, and R. D. Sleator, ‘Big data’, Hadoop and cloud computing in genomics, J. Biomed. Inform., vol. 46, no. 5, pp. 774–781, 2013.

10) S. Karimian-Aliabadi, D. Ardagna, R. Entezari-Maleki, E. Gianniti, and A. Movaghar, Analytical composite performance models for Big Data applications, J. Netw. Comput. Appl., vol. 142, pp. 63–75, 2019.

11) H. F. Yu, A priori algorithm optimization based on Spark platform under big data, Microprocess. Microsyst., vol. 80, p. 103528, 2021.

12) M. Muniswamaiah, T. Agerwala, and C. Tappert, Big data in cloud computing review and opportunities, Int. J. Comput. Sci. Inform. Technol., vol. 11, no. 4, pp. 43–57, 2019.

13) K. Sandhu, "Big data with cloud computing: Discussions and challenges," in Big Data Mining and Analytics, vol. 5, no. 1, pp. 32-40, March 2022, doi: 10.26599/BDMA.2021.9020016.

14) Qian ZP, He Y, Su CZ et al. TimeStream: Reliable stream computation in the cloud. In: Proc. 8th ACM European conference in computer system, EuroSys 2013. Prague: ACM Press; 2013. p. 1–4.

15) Kolajo, T., Daramola, O. & Adebiyi, A. Big data stream analysis: a systematic literature review. J Big Data 6, 47 (2019). https://doi.org/10.1186/s40537-019-0210-7.

16) Zakarya Elaggoune, Ramdane Maamri, Imane Boussebough, A fuzzy agent approach for smart data extraction in big data environments, Journal of King Saud University - Computer and Information Sciences, Volume 32, Issue 4, 2020, Pages 465-478, ISSN 1319-1578, https://doi.org/10.1016/j.jksuci.2019.05.009.

17) Demirbaga, U. HTwitt: a hadoop-based platform for analysis and visualization of streaming Twitter data. Neural Comput & Applic (2021). https://doi.org/10.1007/s00521-021-06046-y.

18) Педоренко, О. Р. Математичне та програмне забезпечення обробки надвеликих масивів даних у форматі XML : магістерська дис. : 121 Інженерія програмного забезпечення / Педоренко Олег Русланович . - Київ, 2019. - 75 с.

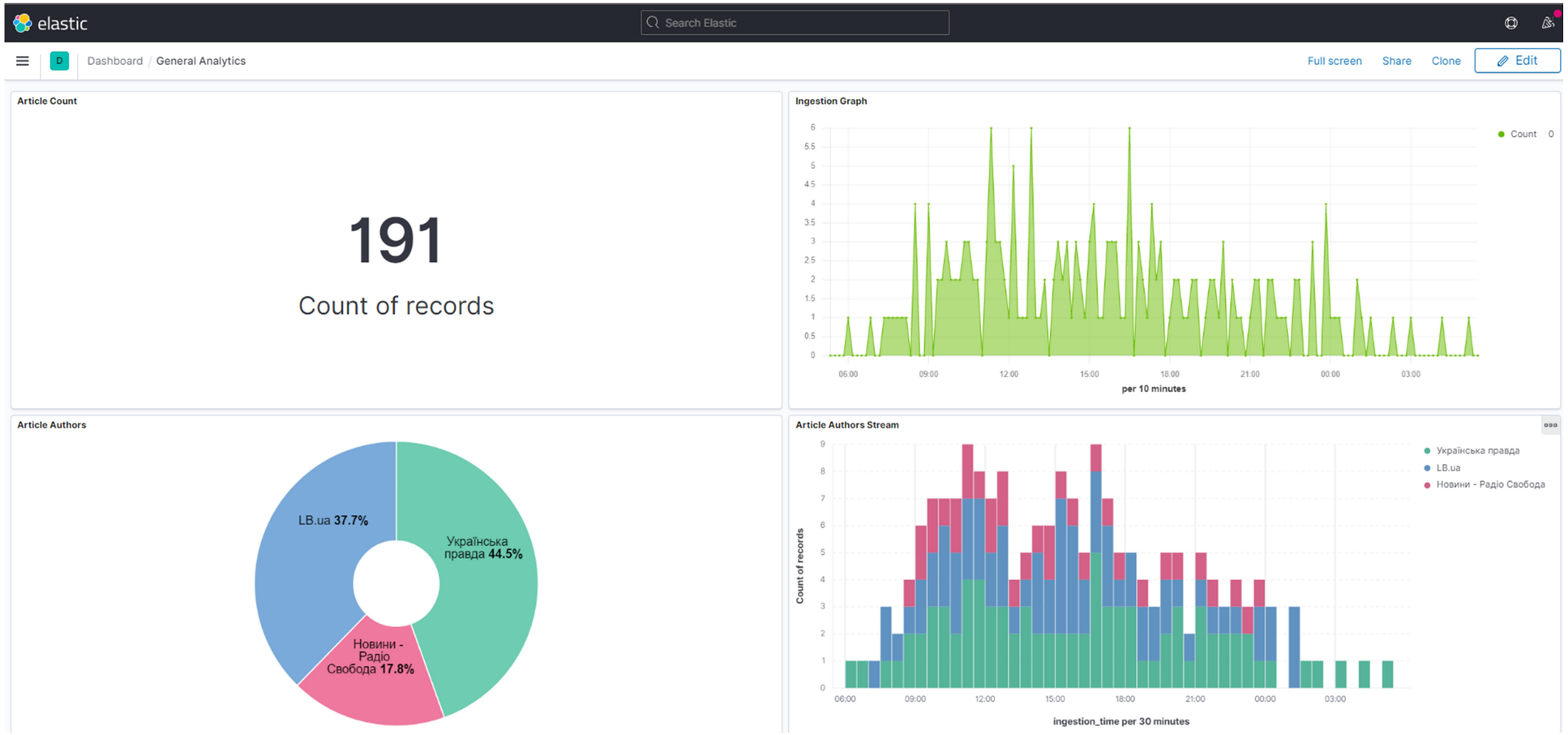
19) ОСОБЛИВОСТІ СТВОРЕННЯ WEB-ОРІЄНТОВАНИХ СИСТЕМ З ДОПОМОГОЮ MEAN СТЕК [Електронний ресурс] – Режим доступу до ресурсу: https://otakoyi.ua/osoblyvosti-stvorennya-web-system-z-dopomohoyu-mean-stek.

20) Ткачук, Є. А. Інформаційна система доставки онлайн замовлень в мережі закладів швидкого харчування : магістерська дис. : 122 Комп’ютерні науки / Ткачук Євгеній Андрійович. - Хмельницький, 2020. - 103 с.

21) Data Science: The 5 V’s of Big Data [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/analytics-vidhya/the-5-vs-of-big-data-2758bfcc51d.

ДОДАТОК А ГРАФІЧНІ МАТЕРІАЛИ

Екранні форми користувацького інтерфейсу



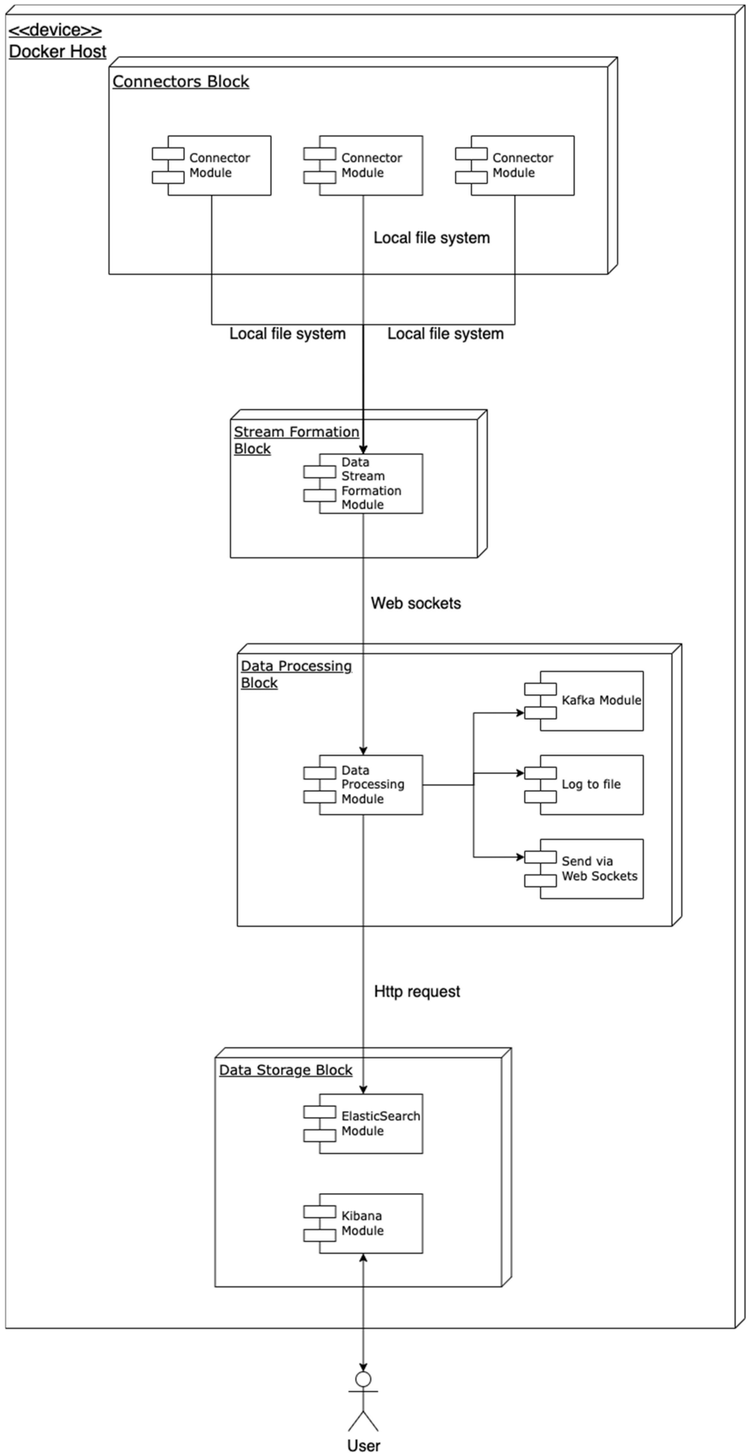
Демонстраційний плакат до магістерської дисертації

Екранні форми користувацького інтерфейсу

Виконав студент гр. ІП-13мп Кувічка М.Є.

Керівник Олійник Ю.О.

Діаграма розгортання програмного забезпечення



Основні компоненти:

- Docker Host – обчислювальна машина, відповідальна за роботу одного або кількох Docker-контейнерів.
- Connectors Block – набір заданої кількості контейнеризованих модулів Connector.
- Stream Formation Block – Docker-контейнер модуля Data Stream Formation.
- Data Processing Block – контейнеризований модуль Data Processing.
- Data Source Block – Docker-контейнер для модулів ElascticSearch та Kibana для зберігання та візуалізації зібраних даних.

Демонстраційний плакат до магістерської дисертації
Діаграма розгортання програмного забезпечення
Виконав студент гр. ІП-13мп Кувічка М.Є.
Керівник Олійник Ю.О.

Результати порівняння часу повного збору даних від кількості джерел вхідних даних

Кількість джерел вхідних даних	Час повного збору даних
1	0.508 с
2	1.516 с
3	1.824 с
4	2.046 с
5	3.085 с
6	3.589 с
7	3.862 с
8	4.805 с
9	5.040 с
10	6.158 с

Демонстраційний плакат до магістерської дисертації

Результати порівняння часу повного збору даних від кількості джерел вхідних даних

Виконав студент гр. ІП-13мп Кувічка М.Є.

Керівник Олійник Ю.О.

ДОДАТОК Б ЛІСТИНГ КОДУ

config.js

```
const dotenv = require('dotenv');

dotenv.config({ path: 'env/.env' });

const envVars = {
  tg: {
    apiId: +process.env.TG_API_ID,
    apiHash: process.env.TG_API_HASH,
    phoneNumber: process.env.TG_PHONE_NUMBER,
    password: process.env.TG_PASSWORD,
    defaultDcId: +process.env.TG_DEFAULT_DC_ID,
  },
  resultPathPrefix: process.env.RESULT_PATH_PREFIX,
};

const getEnv = (variable) => envVars[variable];

module.exports = {
  getEnv,
};
```

rss.js

```
const fs = require('fs');
const { XMLParser } = require('fast-xml-parser');
const fetch = require('node-fetch');
const cheerio = require('cheerio');
const { rssSources } = require('../../sources.json');

const getArticleBody = async (url, sourceSelector) => {
  let response;
  try {
    response = await fetch(url);
  } catch (err) {
    response = null;
  }

  if (!response) return null;

  const html = await response.textConverted();
```

```
  const querySelector = cheerio.load(html);

  return querySelector(sourceSelector).toArray().map((e) => querySelector(e).text()).join('\n');
};

const fetchRssData = async () => {
  const results = [];
  const parser = new XMLParser();

  const sourceParsingPromises = rssSources.map(async (source, sourceIndex) => {
    try {
      const response = await fetch(source.link);
      // const response = fs.readFileSync(source.link);
      const xml = await response.textConverted();
      const feed = parser.parse(xml);

      const sourceItemParsingPromises = feed.rss.channel.item.map(async (item, itemIndex) => {
        // console.time(`Message ${itemIndex} from source ${sourceIndex}`);
        const link = item.link;

        const articleBody = await getArticleBody(link, source.selector);

        if (!articleBody) return;

        try {
          results.push({
            title: item.title,
            link,
            body: articleBody,
            author: item.author ?? null,
            category: item.category,
            date: new Date(item.pubDate),
          });
        } catch (error) {
          console.error(error)
        }

        // console.timeEnd(`Message ${itemIndex} from source ${sourceIndex}`);
      });

      await Promise.all(sourceItemParsingPromises);
    } catch (error) {
      console.error(error);
    }
  });
```

```
  await Promise.all(sourceParsingPromises);

  return results;
};

module.exports = {
  fetchRssData,
};
```

tg.js

```
const MTProto = require('@mtproto/core');
const { sleep } = require('@mtproto/core/src/utils/common');
const prompt = require('prompt');
const path = require('path');

const { getEnv } = require('../../config/config');
const { tgSources } = require('../../sources.json');

const {
  apiId,
  apiHash,
  phoneNumber,
  password,
  defaultDcId,
} = getEnv('tg');

// 1. Create an instance
const mtproto = new MTProto({
  api_id: apiId,
  api_hash: apiHash,

  storageOptions: {
    path: path.resolve(__dirname, './data/1.json'),
  },
});

(async() => { await mtproto.setDefaultDc(defaultDcId); })()

const api = {
  call(method, params, options = {}) {
    return mtproto.call(method, params, options).catch(async error => {
      // console.log(`${method} error:`, error);
```

```
            const { error_code, error_message } = error;

            if (error_code === 420) {
              // console.log({error_message})
              const seconds = +error_message.split('FLOOD_WAIT_')[1];
              const ms = seconds * 1000;

              await sleep(ms);

              return this.call(method, params, options);
            }

            if (error_code === 303) {
              const [type, dcId] = error_message.split('_MIGRATE_');

              // If auth.sendCode call on incorrect DC need change default DC, because call auth.signIn on incorrect DC return PHONE_CODE_EXPIRED error
              if (type === 'PHONE') {
                await mtproto.setDefaultDc(+dcId);
              } else {
                options = {
                  ...options,
                  dcId: +dcId,
                };
              }

              return this.call(method, params, options);
            }

            return Promise.reject(error);
          });
        },
      };

      async function getUser() {
        try {
          const user = await api.call('users.getFullUser', {
            id: {
              _: 'inputUserSelf',
            },
          });

          return user;
        } catch (error) {
```

```
    return null;
  }
}

function sendCode(phone) {
  return api.call('auth.sendCode', {
    phone_number: phone,
    settings: {
      _: 'codeSettings',
    },
  });
}

function signIn({ code, phone, phone_code_hash }) {
  return api.call('auth.signIn', {
    phone_code: code,
    phone_number: phone,
    phone_code_hash: phone_code_hash,
  });
}

function getPassword() {
  return api.call('account.getPassword');
}

function checkPassword({ srp_id, A, M1 }) {
  return api.call('auth.checkPassword', {
    password: {
      _: 'inputCheckPasswordSRP',
      srp_id,
      A,
      M1,
    },
  });
}

const login = async () => {
  const user = await getUser();
  // console.log(user)

  if (!user) {
    const { phone_code_hash } = await sendCode(phoneNumber);

    prompt.start();
```

```
    const {code} = await prompt.get(['code']);

    try {
      const authResult = await signIn({
        code,
        phone: phoneNumber,
        phone_code_hash,
      });

      // console.log(`authResult:`, authResult);
    } catch (error) {
      if (error.error_message !== 'SESSION_PASSWORD_NEEDED') {
        return;
      }

      // 2FA

      const { srp_id, current_algo, srp_B } = await getPassword();
      const { g, p, salt1, salt2 } = current_algo;

      const { A, M1 } = await mtproto.crypto.getSRPParams({
        g,
        p,
        salt1,
        salt2,
        gB: srp_B,
        password,
      });

      const authResult = await checkPassword({ srp_id, A, M1 });

      // console.log(`authResult:`, authResult);
    }
  }
};

const getChannelData = ({id, access_hash}) => api.call('messages.getHistory',
  {
    limit: 10, peer: {
      _: 'inputPeerChannel',
      channel_id: id,
      access_hash,
    }
  });
```

```
const logOut = () => api.call('auth.logOut', {});

const fetchTgData = async() => {
  // console.log('logout', await logOut());
  // console.log('login', await login());
  await login();

  const dialogs = await api.call('messages.getDialogs', {limit: 10, offset_peer: {
    _: 'inputPeerEmpty',
  } });

  const channels = dialogs.chats.filter((dialog) => dialog._ === 'channel' &&
tgSources.includes(dialog.id));
  // console.log(channels)

  const channelsPromises = channels.flatMap(async (channel) => {
    const channelData = await getChannelData(channel);
    return channelData.messages.map((item) => {
      if (!item.message) return;

      const [title, ...body] = item.message.split('\n\n');

      return {
        title,
        link: `https://t.me/${channel.username ? channel.username : `c/${channel.id}`}/${item.id}`,
        body: body.filter(elem => elem).join('\n'),
        author: item.peer_id.channel_id,
        category: null,
        date: new Date(item.date),
      };
    }).filter(e => e);
  });

  const result = await Promise.all(channelsPromises);
  return result.flatMap(e => e);
};

fetchTgData();

module.exports = {
  fetchTgData,
};
```

elastic.js

```
'use strict';

const { Client } = require('@elastic/elasticsearch');

class Elastic {
  client;
  index;

  constructor(url, index) {
    this.client = new Client({node: url});
    this.index = index;
  }

  add(data) {
    return this.client.index({index: this.index, body: data});
  }

  refresh() {
    return this.client.indices.refresh({index: this.index});
  }

  async get(filter) {
    const { body } = await this.client.search({
      index: this.index,
      body: {
        query: filter
      },
    });

    return body;
  }
}

module.exports = Elastic;
```

sources.json

```
{
 "rssSources": [
  {
   "name": "Ukrainian pravda",
   "link": "https://www.pravda.com.ua/rss/view_news/",
   "selector": ".post_text > p, .post__text > p"
  },
  {
```

```
    "name": "LB.ua",
    "link": "https://lb.ua/rss/ukr/news.xml",
    "selector": "div[itemprop=\"articleBody\"] > p"
   },
   {
    "name": "Radio Svoboda",
    "link": "https://www.radiosvoboda.org/api/zrqiteuuir",
    "selector": "#article-content > div > p"
   }
  ],
  "tgSources": ["1449473117", "1134948258"]
}
```

index.js

```
const fs = require('fs');
const path = require('path');
const { v4: uuidv4 } = require('uuid');

const { fetchRssData, fetchTgData } = require('./src/utils');
const { getEnv } = require('./config/config');

const pathPrefix = getEnv('resultPathPrefix');

let counter = 1;

const main = async () => {
 const FIVE_MINUTES = 0.2 * 60 * 1000;

 const pathToDataDir = path.resolve(__dirname, pathPrefix);
 let lastFileIndex = 1;

 setInterval(async () => {
  console.time(`Collection time on ${counter} iteration`);
  let data = [];

  data = data.concat(await fetchRssData());
  data = data.concat(await fetchTgData());

  const dataFileNames = fs.readdirSync(pathToDataDir).sort();
  if (dataFileNames.length > 0) {
   lastFileIndex = +dataFileNames[dataFileNames.length - 1].split('')[0];
  }

  const pathToFile = `${pathToDataDir}/data_${uuidv4()}.json`;
```

```
    await fs.promises.writeFile(pathToFile, JSON.stringify(data));
    console.timeEnd(`Collection time on ${counter} iteration`);
    // console.timeEnd('Batch mode time');
    // console.log('\n');
    counter++;
  }, FIVE_MINUTES)
};

main();
```

ДОДАТОК В РЕЗУЛЬТАТ ПЕРЕВІРКИ НА СПІВПАДІННЯ

UNICHECK

Имя пользователя:
Лісовиченко Олег Іванович

ID проверки:
1013167052

Дата проверки:
03.12.2022 13:02:00 EET

Тип проверки:
Doc vs Internet + Library

Дата отчета:
03.12.2022 13:04:30 EET

ID пользователя:
76913

Название файла: **ІП-13мп_Кувічка_ПЗ**

Количество страниц: 64 **Количество слов:** 10750 **Количество символов:** 79644 **Размер файла:** 7.35 MB **ID файла:** 1012932754

11.3%
Совпадения

Наибольшее совпадение: 1.93% с источником из Библиотеки (ID файла: 1009535872)

1.75% Источники из Интернета	5	Страница 66
11.3% Источники из Библиотеки	283	Страница 66

0% Цитат

Исключение цитат выключено

Исключение списка библиографических ссылок выключено

0% Исключений

Нет исключенных источников