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

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

До захисту допущено

Завідувач кафедри

_____ Віталій РОМАНКЕВИЧ
(підпис) (ініціали, прізвище)

"___" _____ 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Спеціалізовані комп'ютерні системи»

спеціальності **123 «Комп'ютерна інженерія»**

на тему: Система інтелектуального розбору новинних повідомлень

Виконав :

студент IV курсу, групи КВ-61
(шифр групи)

Коровій Олександр Сергійович _____ _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доц. каф. СПіСКС, к. т. н., доцент Замятін Д.С._____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з нормоконтролю, доц.каф.СПСКС, к.т.н. Клятченко Я.М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2020 року

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

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Спеціалізовані комп'ютерні системи»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ
(підпис) (ініціали, прізвище)
«____» _____2020 р.

ЗАВДАННЯ
на дипломний проєкт студента

Коровія Олександра Сергійовича

1. Тема проєкту «Система інтелектуального розбору новинних повідомлень», керівник проєкту: Замятін Денис Станіславович доц. каф. СПіСКС, к. т. н., затверджені наказом по університету від «___» _____ 2020 р. №_____

2. Термін подання студентом проєкту: «18» травня 2020 р

3. Вихідні дані до проєкту:
   - програмне забезпечення, для розбору новинних повідомленнь;

4. Зміст пояснювальної записки:
   - аналіз існуючих рішень для розбору новинних повідомлень;
   - аналіз технологій та алгоритмів реалізації новинних повідомлень;
   - опис алгоритму та архітектурних рішень розробленої системи;

– тестування та рекомендації для подальшого вдосконалення розробленої системи

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо):

– розташування модулів та файлів (схема структурна);
– загальна схема проекту (схема структурна);
– алгоритм побудови шаблону (схема алгоритму);
– структурна діаграма шаблону (схема структурна).

6. Консультанти розділів проєкту:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормконтроль	Клятченко Я.М., к.т.н., доцент каф. СПіСКС		

7. Дата видачі завдання "31" жовтня 2019 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення літератури за тематикою проєкту	18.11.2019	
2	Розроблення та узгодження технічного завдання	01.12.2019	
3	Аналіз існуючих рішень	16.12.2019	
4	Підготовка матеріалів першого розділу проєкту	11.01.2020	
5	Розроблення програмного забезпечення	09.02.2020	
6	Відлагодження програмного продукту	24.02.2020	
7	Підготовка матеріалів другого розділу проєкту	24.03.2020	
8	Розроблення інтерфейсу програмного забезпечення	16.04.2020	
9	Підготовка графічної частини	04.05.2020	
10	Оформлення документації дипломного проєкту	14.05.2020	

Студент _____ Олександр КОРОВІЙ
(підпис)

Керівник проєкту _____ Денис ЗАМЯТІН
(підпис)

АНОТАЦІЯ

Дипломний проєкт включає пояснювальну записку (57 с., 25 рис., 4 додатки).

Об’єкт розробки – система інтелектуального розбору новинних повідомлень.

Система допомагає автоматично будувати шаблон для новинного ресурсу, що звільняє користувача від написання індивідуального веб-скрапера для кожного веб-сайту. Також з використання єдиного веб-скрапера система дозволяє зменшити використання ресурсів комп’ютера.

Розроблена система дозволяє:

- автоматично створювати шаблон для заданого користувачем новинного веб-сайту;
- додавання шаблону створеного користувачем;
- обробляти велику кількість веб-сайтів в режимі реального часу;
- масштабування на декількох комп’ютерах;
- структуризація вихідних даних.

Розробка велася з використанням мови програмування Python та сучасних бібліотек. Використання асинхронного програмування дозволило підвищити швидкодію всієї системи.

Ключові слова: ІНТЕРНЕТ, АСИНХРОННІСТЬ, PYTHON, ВЕБ-СКРАПІНГ, НОВИННІ РЕСУРСИ.

ABSTRACT

The diploma project includes an explanatory note (57 pages, 25 figures, 4 appendices).

The object of development is a system of intelligent analysis of news messages.

The system helps to automatically build a template for a news resource, which frees the user from writing an individual web scraper for each website. Also, by using a single web scrapper, the system allows you to reduce the use of computer resources.

The developed system allows:

• automatically create a template for a user-defined news website;

• adding a user-created template;

• handle a large number of websites in real time;

• scaling on multiple computers;

• structuring of source data.

The development was carried out using the Python programming language and modern libraries. The use of asynchronous programming has increased the speed of the entire system.

Keywords: INTERNET, ASYNCHRONOUSNESS, PYTHON, WEB SCRAPING, NEWS RESOURCES

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045440.002 ТЗ	Система інтелектуального	4		
			розбору новинних			
			повідомлень.			
			Технічне завдання			
	А4	ІАЛЦ.045440.003 ТП	Система інтелектуального	2		
			розбору новинних			
			повідомлень.			
			Відомість технічного			
			проекту			
	А4	ІАЛЦ.045440.004 ПЗ	Система інтелектуального	57		
			розбору новинних			
			повідомлень.			
			Пояснювальна записка			
	А4	ІАЛЦ.045440.005 Д1	Система інтелектуального	1		
			розбору новинних			
			повідомлень.			
			Розташування модулів та			
			файлів.			
			Схема структурна			

Змін	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045440.001 ОА		
Розробив		Коровій О. С.			Система інтелектуального розбору новинних повідомлень. Опис альбому	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Замятін Д.С,					
Н. контроль		Клятченко Я.М.				КПІ ім. Ігоря Сікорського ФПМ КВ-61	
Затвердив		Романкевич В.О					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045440.006 Д2	Система інтелектуального	1		
			розбору новинних			
			повідомлень в режимі			
			реально часу.			
			Загальна схема проекту.			
			Схема структурна			
	А4	ІАЛЦ.045440.007 Д3	Система інтелектуального	1		
			розбору новинних			
			повідомлень.			
			Алгоритм побудови			
			шаблону.			
			Схема алгоритму			
	А4	ІАЛЦ.045440.008 Д4	Система інтелектуального	1		
			розбору новинних			
			повідомлень.			
			Структурна діаграма			
			шаблону.			
			Схема структурна			
		Диск CD-ROM	Текст пояснювальної	1		
			записки.			
			Графічний матеріал			

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Система інтелектуального розбору новинних повідомлень»

Галузь застосування: моніторинг засобів масової інформації та великі дані.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проекту є створення системи розбору новинних повідомлень в режимі реального часу.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації в періодичних виданнях та електронні статті у мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

1.1 Вимоги до програмного продукту, що розробляється

- забір текстових даних з різних новинних веб-ресурсів;
- забір часу публікації новинного повідомлення;
- забір заголовку новинного повідомлення;
- масштабування системи;

- наявність графічного інтерфейсу;
- наявність зручної системи управління;

5.2 Вимоги до апаратного забезпечення

- оперативна пам’ять: 4 Гб;
- наявність доступу до глобальної мережі Internet.

5.3 Вимоги до програмного та апаратного забезпечення користувача

- операційна система Linux, MacOS;
- наявність доступу до мережі Internet.

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Видача завдання на дипломне проєктування	31.10.2019
2.	Вивчення літератури за тематикою роботи	18.11.2019
3.	Розроблення та узгодження технічного завдання	01.12.2019
4.	Розроблення структури додатку	16.01.2020
5.	Розроблення дизайну та графічних елементів	02.02.2020
6.	Програмна реалізація додатку	15.03.2020
7.	Тестування додатку	10.04.2020
8.	Підготовка матеріалів текстової частини проекту	18.04.2020
9.	Підготовка матеріалів графічної частини проекту	04.05.2020
10.	Оформлення технічної документації проекту	14.05.2020

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045440.004 ПЗ	Система	57		
			розбору новинних			
			повідомлень.			
			Пояснювальна записка			
	А4	ІАЛЦ.045440.005 Д1	Система	1		
			розбору новинних			
			повідомлень.			
			Розташування модулів та			
			файлів.			
	А4	ІАЛЦ.045440.006 Д2	Система	1		
			розбору новинних			
			повідомлень в режимі			
			реально часу.			
			Загальна схема проекту.			
			Схема структурна			

Змін	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045440.003 ТП		
Розробив		Коровій О. С.			Система інтелектуального розбору новинних повідомлень. **Відомість технічного проєкту**	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Замятін Д.С,					
Н. контроль		Клятченко Я.М.				КПІ ім. Ігоря Сікорського ФПМ КВ-61	
Затвердив		Романкевич В.О					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045440.007 Д3	Система інтелектуального	1		
			розбору новинних			
			повідомлень.			
			Алгоритм побудови			
			шаблону.			
			Схема алгоритму			
	А4	ІАЛЦ.045440.008 Д4	Система інтелектуального	1		
			розбору новинних			
			повідомлень.			
			Структурна діаграма			
			шаблону.			
			Схема структурна			
		Диск CD-ROM	Текст пояснювальної	1		
			записки.			
			Графічний матеріал			

					ІАЛЦ.045440.003 ТП	Арк.
						2
Змін.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

Додаток 1. Копії графічних матеріалів:

- ІАЛЦ.045440.005 Д1. Розташування модулів та файлів. Схема структурна;
- ІАЛЦ.045440.006 Д2. Загальна схема проекту. Схема структурна;
- ІАЛЦ.045440.007 Д3. Алгоритм побудови шаблону. Схема алгоритму;
- ІАЛЦ.045440.008 Д4. Структурна діаграма шаблону. Схема структурна.

Додаток 2. Лістинг програмного коду.

Додаток 3. Презентація.

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

AJAX – Asynchronous JavaScript And XML - надсилання у фоновому режимі запитів на веб-сервер;

API – Application Programming Interface – прикладний програмний інтерфейс;

CSV - comma-separated values - файловий формат для представлення таблиць;

CSS – Cascading Style Sheets - каскадні таблиці стилів;

DDOS - (Distributed) Denial-of-service attack - атака на відмову в обслуговуванні, розподілена атака на відмову в обслуговуванні;

DOM - Document Object Model - об'єктна модель документа - прикладний програмний інтерфейс для роботи зі структурованими документами (як правило, документами XML/HTML);

HTML – HyperText Markup Language – мова розмітки гіпертекстових документів;

HTTP – HyperText Transfer Protocol – протокол передачі даних;

HTTPS – HyperText Transfer Protocol Save – безпечний протокол передачі даних;

ISO – International Organization for Standardization – Міжнародна організація зі стандартизації;

JSON – JavaScript Object Notation — текстовий формат обміну даними;

JSON-LD – JavaScript Object Notation for Linked Data - метод кодування пов'язаних даних за допомогою JSON;

NLP – Nature Language Processing - обробка природньої мови;

RAM – Random Access Memory - оперативна пам'ять;

RDFa – Resource Description Framework in Attributes - фреймоворк опису ресурсу в атрибутах;

RSS – Really Simple Syndication – формат, що використовується для публікації та постачання інформації;

URL – Uniform Resource Locator – стандартизована адреса певного ресурсу;

W3C - World Wide Web Consortium - консорціум всесвітньої павутини;

WWW – World Wide Web – всесвітня мережа;

XHTML – Extensible HyperText Markup Language – розширювана мова розмітки гіпертексту;

XLST - Extensible Stylesheet Language Transformations - мова для програмування переробки XML-документів;

XML - Extensible Markup Language - розширювана мова розмітки;

XPath - XML Path Language - мова запитів до елемента XML-документа;

ВСТУП

В сучасному світі, де кожну секунду генеруються сотні терабайт даних, дуже важко не заплутатися. Ця інформація дуже корисна в наш час. Її використовують скрізь від аналітичних компанії до науковців. В наш час дані - це серцевина маркетингових досліджень, бізнес-стратегій, науки про дані, тощо.

Постійно потрібно отримувати доступ до великої кількісті даних та аналізувати їх, для того, щоб розробляти нові програми, які будуть використовувати ці дані, проводити дослідження та отримувати сховані шаблони в даних.

Кожну хвилину в українському інтернеті, з'являється тисячі новинних повідомлень. На даний момент людина фізично не встигає відслідковувати кожен ресурс. Багато хто маніпулює новинами, часто виникають фейкові новини, що заплутує користувачів новинних ресурсів та вводить їх в оману. Тому вчасний забір та аналіз подібних новин, допоможе уникнути неприємних ситуацій.

Зараз майже кожен підписується на певну стрічку новин, яка відповідає певній тематиці. Для того щоб створити персоналізовану стрічку потрібно структурувати новини, а це на даний момент досить важко. Тому що виникають проблеми з тим, що дані в новиних інтернет ресурсах не структуровані, і їх важко використовувати для подальшого аналізу. Кожний вебсайт - різний, має власний стиль відображення, дизайн, категорію і т.д. Тому для того щоб забрати текст новин з вебсайту використовується скрапінг.

Вебскрапінг - це автоматизований метод, який використовується для отримання великої кількості даних з вебсайтів. За допомогою вебскрапінгу ви можете витягти дані з будь-якого вебсайту, незалежно від того, наскільки вони великі, на вашому комп'ютері. Більше того, вебсайти можуть мати дані, які ви

не можете скопіювати та вставити. Вебскрапінг може допомогти вам отримати будь-які потрібні вам дані.

Дані зазвичай на вебсайтах неструктуровані, тому для структурування великих даних затрачається багато людських зусиль. Вебскрапінг допомагає збирати ці неструктуровані дані, перетворювати та зберігати їх у структурованому вигляді. Ви зможете зберегти дані у такому форматі, як CSV. Тоді ви зможете отримати, проаналізувати та використовувати дані так, як вам потрібно, наприклад, для побудови персоналізованої стрічки новин.

Таким чином, вебскрапінг спрощує процес отримання даних з веб-сторінки, заощаджує час на вирішення проблем із завантаженням або копіюванням будь-яких даних вручну, автоматизує весь процес та створює простий доступ до отриманого результату, який можна сформувати у будь-якому форматі, що цікавить користувача.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ДЛЯ РОЗБОРУ НОВИННИХ ПОВІДОМЛЕНЬ

1.1 Аналіз особливостей веб-парсингу

Ідея скрапінгу (парсингу) базується на тому щоб дістати потрібну інформацію з DOM-дерева (Document Object Model) HTML (HyperText Markup Language — мова розмітки гіпертексту), для прикладу на рисунку 1.1, зображено DOM-дерево вебсайту ITC.ua. Тому парсинг в першу чергу це розбір HTML і отримання інформації [1].

Рисунок 1.1 – Приклад HTML дерева для новини, вебсайт itc.ua

Скрапінг включає в себе витяг певних даних на цільовій веб-сторінці - наприклад, вилучення тексту новину, збір коментарів під новиною та кількість переглядів. На відміну від цього, краулінг - це те, що роблять пошукові системи. Краулер сканує та індексує весь вебсайт разом зі своїми внутрішніми

посиланнями. Він виконує навігацію по веб-сторінкам без будь-якої певної мети.

Скрапінг використовується для отримання контенту. Багато хто зациклюються на тому, що скрапінг - це саме злодійство контенту, хоча це абсолютно не так. Парсинг - це всього лише автоматизований збір інформації, і не більше того. Наприклад, парсинг фотографій, особливо з "водяними знаками" - це чистої води злодійство контенту і порушення авторських прав. Тому таким зазвичай не займаються (ми в своїй роботі обмежуємося збором тексту, не більше того).

Коли ви збираєте і аналізуєте дані з відкритих джерел - це абсолютно законно. Але публікація отриманого контенту як свого власного, без вказівки джерела також не є етичним. Ви повинні пам'ятати, що спам, плагіат або будь-шахрайське використання даних заборонено законом.

Потрібно памятати, не можна парсити особисті дані, для яких потрібні ім'я користувача і пароль від веб-сервісів, тому що це може бути особиста інформація.

Слід дотримуватися умов користування сервісу, які можуть прямо заборонити використання парсерів.

Не можна копіювати дані, захищені авторським правом (тим більше їх використовувати).

Людина може бути притягнута до відповідальності за декількома законами. Парсинг особистої контактної інформації без дозволу власника і її продаж третім особам для отримання прибутку є незаконною. Наприклад, хтось отримав за допомогою парсинга деяку конфіденційну інформацію і продав її третій особі, не звертаючи уваги на письмову відмову власника сайту. Ця особа може бути притягнута до відповідальності відповідно до закону про порушення прав власності, порушенням Закону про захист авторських прав у цифрову епоху (DMCA), порушенням Закону про комп'ютерне шахрайство і

зловживання (CFAA) і в зв'язку з незаконним привласненням (застосовно до США) [2].

Це не означає, що ви можете вільно парсити блоги і канали в глобальних соціальних мережах, таких як: Twitter, Facebook, Instagram, і YouTube. Вони доброзичливі по відношенню до парсерів, які слідують положенням файлу robots.txt. Даний файл це стандарт для пошукових роботів та парсерів, це спосіб комунікації різних вебсайтів з пошуковими роботами. Стандарт визначає, як повідомити пошукового робота та парсер про те, які частини вебсайту не повинні бути проіндексовані [3].

Перш ніж аналізувати глобальну соцмережу Facebook, вам обов'язково необхідно, звернутися до представників компанії, вказати причину та отримати письмовий дозвіл. Це дозволить уникнути відповідальності в майбутньому.

Скрапінг - не те ж саме, що API (Application Programming Interface). Тому що API - це як канал для відправки вашого запиту даних на веб-сервер і відповіді на нього. Наприклад, компанія може відкрити доступ до API, щоб дозволити іншим системам взаємодіяти з її даними; при цьому якість і кількість доступних даних через API, як правило, нижча, ніж можна отримати, використовуючи парсинг. API повертає дані в форматі JSON (JavaScript Object Notation) по протоколу HTTP/HTTPS (HyperText Transfer Protocol). Однак це не означає, що за своїм запитом ви можете отримати будь-які дані. Також деякі компанії вимагають плату за API, для того щоб отримати структуровані дані. Парсинг в Інтернеті може візуалізувати процес, оскільки він дозволяє вам взаємодіяти з вебсайтами. Крім того, парсинг надає більш актуальну інформацію, ніж через API, і значно простіше налаштовується з практичної точки зору.

В парсингу веб-сторінок на разі є багато проблем одна із проблем це - фахівцям без технічних навичок, буде важко і не просто отримувати дані. Тобто якщо ви не маєте навичок в програмуванні, то для вас буде важко

написати парсер для вебсайту. Вам потрібно скористатися послугами приватних компаній, які надають послуги парсингу веб-сторінок за плату, або особисто копіювати та структурувати дані.

Також при парсингу можна зіткнутися з такою проблемою як DDOS ((Distributed) Denial-of-service attack) атака, на сайт який парситься. При великій кількості запитів, що надходять на веб-сервер, де розміщується вебсайт, сервер може не встигати обробляти інформацію, тому що запити, що надійшли, обробляються в порядку черги. Через велику кількість запитів сервер може перезавантажутися, що зробить його не доступним для інших користувачів. В такому випадку парсинг може кваліфікуватися як DDOS або хакерська атака. Принцип DDOS атаки, ґрунтується на здійсненні великої кількості запитів на веб-ресурс, блокує його роботу, змушуючи перевантажуватися [4].

На даний час вебсайти дуже швидко розвиваються, стають все красивішими, більш функціональними, все робиться для того щоб привернути аудиторію до свого ресурсу. Звідси і виникає наступна проблема при парсингу веб-ресурсів. Що парсер який працював сьогодні, завтра не зможе працювати, тому що вебсайт змінив дизайн, чи вніс не великі корегування в DOM-дерево. І це призводить до того що в деяких випадках потрібно переписувати існуючий парсер, або навіть і створювати новий, останнє використовують найчастіше.

Тому досвідчені інженери зі створення парсерів, з самого початку розробляють своїх роботів таким чином, щоб вони були більш гнучкими до змін вебсайту і т.д., що робить їх набагато надійнішими.

Часто виникають проблеми з якістю даних, яка забирається з вебсайту, тут потрібно спостерігати за роботою парсеру, як саме він парсить сайт, виявляти і усувати першочергові проблеми з якістю витягнутих даних.

З цієї сторони API має перевагу над парсингом, тому що має фіксовану структуру, яка змінюється не дуже часто, але як було сказано раніше API може не надавати актуальну інформацію і т.д.

1.2. Аналіз існуючих рішень

На даний час існують багато рішень для вебскрапінгу. Багато компаній надають послуги парсингу будь-яких вебсайтів, при дотримані всіх рекомендацій. Вони можуть надавати дані одноразово, наприклад якщо потрібні структуровані дані за певний період, або в режимі реального часу. Наприклад парсити новині ресурси, для того щоб оперативно отримувати новини. Багато компаній надають вже готові аналітичні звіти по зібраних даних, тобто клієнту не потрібно займатися обробкою даних, він одразу отримує агреговані та проаналізовані дані, які він може використовувати у своїх цілях.

За досить великий проміжок часу, веб спільнота розрослася до дуже великих масштабів, на даний час зявилося дуже багато бібліотек та фреймворків для різних мов програмування. Кожна з них має як свої плюси так і мінуси, можу бути легким або важким у засвоєні, має різну швидкодію. В кожної з них є свої прихильники. Більшість компаній використовують той чи інший відкритий фреймворк чи бібліотеку, але іноді займаються написанням власних. Тому часто можна зустріти фреймоворк який виклала та чи інша компанія для розвитку Open Source. Багато з них не є ідеальними, тому компанія хоче залучитися підтримкою спільноти для покращення і розвитку свого продукту.

1.3. Засоби вилучення та структурування новинних повідомлень

Scrapy - це фреймворк написаний на мові Python, для сканування вебсайтів, вилучення та структурування даних, які можуть бути широко застосовуватися різних сферах, таких як обробка даних, обробка інформації чи історичний архів [5].

Незважаючи на те, що Scrapy спочатку був розроблений для вебскрапінгу, він також може бути використаний для вилучення даних за

допомогою API (наприклад, веб-служб Amazon Associates Web Services) або як веб-сканер загального призначення [5].

Scrapy - це фреймворк з відкритим кодом, розповсюджуться по відкритій ліцензії, тобто можна використовувати вихідний код фреймоврку у своїх цілях. Приклад написаного готового за допомогою фрейморка Scrapy, парсера для вебсайту http://quotes.toscrape.com, який забирає зі сторінки текст та автора, наведено на рисунку 1.2 [5].

```python
import scrapy


class QuotesSpider(scrapy.Spider):
    name = 'quotes'
    start_urls = [
        'http://quotes.toscrape.com/tag/humor/',
    ]

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'author': quote.xpath('span/small/text()').get(),
                'text': quote.css('span.text::text').get(),
            }

        next_page = response.css('li.next a::attr("href")').get()
        if next_page is not None:
            yield response.follow(next_page, self.parse)
```

Рисунок 1.2. Приклад коду написаний мовою програмування Python з використання фреймворку Scrapy

Scrapy надає безліч потужних функцій для полегшення та ефективності вебскрапінгу, такі як:

1. Вбудована підтримка вибору та вилучення даних з джерел HTML/XML, за допомогою розширених селекторів CSS та виразів XPath з допоміжними методами які використовують регулярні вирази [5].

2. Інтерактивна консоль для випробування та налагодження CSS і XPath шляхів, дуже корисна при написанні або налагодженні ваших веб-скраперів або кроулерів [5].
3. Вбудована підтримка для генерації експорту даних у декількох структурованих форматах (JSON, CSV, XML) та зберігання їх у декількох пакетах (FTP, S3, локальна файлова система). Також за допомогою стороніх бібліотек можна налагодити зберігання в базу даних [5].
4. Надійна підтримка різних кодуваннь тексту та автоматичне виявлення для роботи з іноземними, нестандартними деклараціями кодування. Також присутнє виправлення кодування, при його пошкоджені [5].
5. Підтримка широкого асортименту вбудованих розширень для отримання даних:
    1. файли cookie та обробка сеансів;
    2. HTTP/HTTPS-функції, такі як стиснення, аутентифікація, кешування, шифрування;
    3. підтримка та зчитування файлів robots.txt;
    4. обмеження глибини для краулерів, для того щоб не перезавантажувати парсер, великою кількістю посиланнь;
6. Консоль Telnet для підключення до Scrapy парсера, щоб спостерігати за його роботою, та відлагоджувати [5].
7. Веб-павуки (краулери) для багаторазового використання та сканування сайтів із мапи сайту (Sitemap) та каналів XML/CSV заданих користувачем [5].
8. Медіаконвеєр для автоматичного завантаження зображень, відео, аудіо, які пов’язані з елементами на сторінці, та багато іншого [5].

Але scrapy також має свої недоліки:

1. Він досить великий і складний, тому для парсингу невеликих даних фреймоворк не бажано застосовувати.
2. Потребує корегування програмного, коду якщо сайт змінив дизайн, або вніс зміни в DOM-дерево, тому коли у вас багато парсерів, це накладає незручності у обслуговувані та підтримці програмного коду парсера чи кроулера. З Часом це буде займати все більше і більше часу у інженерів.
3. Документація scrapy важка для початківців. Тому інженеру без досвіду важко буде розібратися в архітектурі scrapy. Із-за складної архітектури scrapy важче відлагоджувати.

1.4. Інструментарій для забору новин з емуляцією веб-браузера.

Selenium WebDriver - це набір інструментів для автоматизації веб-браузерів. Перш за все Selenium призначений для автоматизації тестування веб-додатків. Він дозволяє розробнику писати тести на різних мовах програмування, таких як Java, Python та інші. Також, так як це автоматизований браузер, він знайшов своє місце у парсингу веб-сторінок та надає можливості для отримання даних зі веб-сторінки [6].

Selenium WebDriver має підтримку багатьох браузерів, а саме:

- Google ChromeDriver
- Microsoft Edge Driver
- Mozilla GeckoDriver
- Apple SafariDriver

Програма спілкується з браузером через універсальний веб-драйвер який надсилає зазначену дію, яку повинен зробити браузер в рамках поточної сесії. Гарними прикладами таких команд можуть бути команди: парсинг тексту сторінки/елемента, знаходження елементів по заданому шляху, перехід по посиланнях, натискання кнопок на сторінці вебсайту [6].

Тобто selenium webdriver повністю автоматизує роботу з браузером, тому що може виконувати різні команди, і може емулювати поведінку користувача, через це він є дуже корисним для написання тестів та парсингу сторінок [6].

Ключовими особливостями selenium webdriver є те, що він спрощує роботу з основними концепціями Javascript, DOM та дуже легко опрацьовувати запити AJAX/PJAX (Asynchronous JavaScript And XML). Тому що на деяких сайтах певна інформація завантажується за допомогою Javascript, і в такому випадку потрібно виконувати код Javascript. Тут тоді приходить на допомогу selnium webdriver, так як він може спілкуватися з браузером і виконувати код Javascript в середині браузера. Це набагато спрощує роботу з ним [6].

Selenium webdriver має доволі простий інтерфейс для взаємодії з ним, що спрощує роботу для початківців. Приклад використання можна глянути на рисунку 1.3.

```
from selenium import webdriver

driver = webdriver.Firefox()
driver.get("http://www.google.com")

elem = driver.find_element_by_name("q")
elem.send_keys("Hello WebDriver!")
elem.submit()

print(driver.title)
```

Рисунок 1.3. Приклад взаємодії із браузером Firefox, за допомогою мови програмування Python

Але selenium web driver має свої недоліки. Найбільший з них це ресурси, навідміну від звичайних парсерів, потрібно запускати повноціний браузер, що доволі об'ємний по ресурсах компютера. Такі браузери займають багато RAM (Random Access Memory) пам'яті, та процесорного часу. Швидкодія значно менша в порівняні з іншими, це повязано з роботою веб-браузера який, на виконананя тієї чи іншої команди, витрачає значно більше ресурсів. Ресурси комп'ютера не дозволять запустити декілька парсерів одночасно, з використанням selenium webdriver. Також у випадку зміни дизайну вебсайту чи внесення змін у HTML вебсайту, парсер на основі selenium webdriver перестане працювати, тому на підтримку актуальної версії також потрібно витрачати ресурси та час. Доцільно використовути його лише при умові, що потрібно виконувати Javascript та об'єм даних не великий.

1.5. Бібліотека для парсингу новин newspaper

Newspaper - це бібліотека написана на мові програмування Python, для вилучення тексту новин. Вона базується на описаних шаблонах для найбільш відомих сайтів. Має багато корисних вбудованих інструментів [7]:

- ідентифікація URL-адреси (Uniform Resource Locator) новини, тобто перевіряє чи відноситься даний URL до новинного повідомлення;
- витяг тексту новини з HTML;
- вилучення зображеннь з HTML;
- отримання ключових слів з тексту, за допомогою NLP (Nature Language Processing) методів;
- витяг автора тексту, якщо він вказаний;
- працює з більш ніж 10 мовами (англійська, китайська, німецька, арабська, ...).

Але також має власні недоліки:

- невелика швидкодія;

- багато новин не може розпізнати, тому що шаблони прописані на основі топ сайтів, тому з багатьма сайтами не може працювати;
- некоректно розпізнається час публікації новин;

1.6. Висновки до розділу

Проаналізувавши різні інструменти та фреймворки, можна дійди до висновку, що більшість з них не пристосовані до змін на вебсайтах, коли змінюється дизайн вебсайту, парсер потрібно переписувати, або мають не велику кількість розпізнавання новин, важко підтримувати, коли потрібно парсити велику кількість вебсайтів. Мають не велику швидкодію та вимагають велику кількість ресурсів,

2. АНАЛІЗ ТЕХНОЛОГІЙ ТА АЛГОРИТМІВ РЕАЛІЗАЦІЇ СИСТЕМИ РОЗБОРУ НОВИННИХ ПОВІДОМЛЕНЬ

Розробка системи інтелектуального розбору новинних повідомлень в режимі реального часу - має на меті полегшити парсинг вебсайтів які містять новини, або новинні повідомлення. Дане програмне забезпечення полегшить роботу з вебскрапінгом даних вебсайтів, тому що не потребує підтримки сотень парсерів для кожного вебсайту окремо. Парсинг цих сайтів відбуватиметься в реальному часі, це також накладає додаткові складнощі, тому що потрібно вчасно адаптуватися під зміни вебсайту.

Програмне забезпечення базуватиметься на існуючих інструментах, які є досить швидкими та не вимагають великих ресурсів комп'ютера. Також буде реалізовуватиме всі можливості даних інструментів, для збільшення швидкодії.

2.1 Аналіз технологій, фреймворків та бібліотек

На сьогодні існує безліч мов програмування та бібліотек для кожної з них. Одними з найпопулярніших є мови програмування JAVA, C/C++, Python (за даними рейтингу TIOBE). Кожна з них унікальна і має свої плюси та мінуси.

Для розробки програмного забезпечення системи інтелектуального розбору новинних повідомлень в режимі реального часу, ми будемо використовувати мову програмування Python, а саме версію Python 3.8.

2.1.1 Мова програмування Python

Мова Python – це інтерпретована мова програмування високого рівня, має Сі-подібний синтаксис, та дуже легка у використані. Підтримує багато парадигм програмування [8]:

- об'єктно-орієнтована;
- процедурна;
- функціональна.

Розроблена Гвідо ван Россумом в 1990 році. Програмний код на ній пишеться в декілька разів швидше ніж на інших мовах, це дає розробникам та інженерам швидше розробляти програмне забезпечення, та тестувати його, що збільшує швидкість впровадження програм від теоретичного задуму в практичний [8].

Python – не є строго типізованою мовою, він підтримує динамічну типізацію, та сам слідкує за типом даних в процесі виконання програми. Має мабуть одну з найбільших спільнот та колекцій бібліотек, фреймворків, яка підтримується великою кількістю людей [8].

Python завоював велику популярність серед науковців, веб-програмістів, та інших розробників завдяки своїми бібліотекам, які дуже зручні у використані та є досить потужними.

В даному проекті ми будемо використовувати такі бібліотеки Python:

- BeautifulSoup;
- Lxml;
- Pandas;
- Asyncio;
- FastAPI.

2.1.2 Бібліотека для парсингу HTML

BeautifulSoup - це бібліотека мови Python, з відкритим кодом, для парсингу даних з файлів, або вебсайтів формату HTML і XML. Вона працює з вбудованим аналізатором, щоб забезпечити зручні способи навігації, пошуку, отримання даних, зміни та розбору дерева HTML і XML. Зазвичай це економить інженерам та розробника години, а то навіть і дні роботи [9].

Одною з привілегій бібліотеки beautifulsoup - це підтримка декількох різновидів аналізаторів: HTML-parser, HTML5lib, html.parser. Також бібліотека досить швидка, легка в користуванні, завдяки аналізатору HTML5lib, має повну підтримку HTML5. Потрібно відміти те, у даної

бібліотеки є детально описана документацію та досить активна спільнота, яка підтримується багатьма розробниками по всьому світу. Це пришвидшує роботу з нею та зручність її використання [9].

2.1.3 Інструментарій lxml

Інструментарій lxml є зв'язуванням бібліотек написаною мовою C libxml2 та libxslt з Python. Він унікальний тим, що він скомпільований та поєднує в собі швидкість та повноту розбору дерева XML/HTML даних бібліотек з простотою використання в Python. Тобто ми отримуємо швидкість роботи, як у C та простоту використані як у Python. Бібліотека lxml працює з усіма версіями Python від 2.7 до 3.8 [10].

Lxml надає великі можливості в розборі дерева HTML/XML:

- парсинг дерева відбувається за допомогою функції які написані мовою C;
- підтримка мови запитів для вибору вузлів - XPath, підтримуються як і абсолютні так і відносні XPath локатори;
- підтримка трасформацій XLST (eXtensible Stylesheet Language Transformations);
- рекурсивний обхід по вузлах дерева, отримання даних з атрибутів елемента та інші операції з вузлами;
- повна сумісність з кодуванням Unicode та UTF-8.

Потрібно відзначити дуже гарну документацію lxml, яка має безліч прикладів для використання, можна завантажити в будь-якому форматі, що полегшує роботу для новачків. Висока швидкодія та мінімальне використання ресурсів комп’ютера надало перевагу lxml, над іншим подібним інструментарієм.

2.1.4 Бібліотека для роботи із таблицями

Pandas – це мабуть одна з найпопулярніших бібліотек мови Python для роботи з таблицями. Вона розповсюджується відкрито, за умовами ліцензії BSD. Перша версія з'явилася в 2008-му році, за цей час вона зібрала велику спільноту навколо себе [11].

Бібліотека націлена на роботу з великими таблицями, дозволяє робити різні маніпуляції з даними, числовими таблицями та часовими рядами. Надає велику кількість інструментів для аналізу даних, побудови графіків, роботи з структурованими наборами даних. Так як бібліотека має досить простий інтерфейс, можливості роботи з різними форматами файлів і таблиць, з'єднання наборів даних, вона завоювала популярність не тільки серед науковців, які працюють з великими об'ємом даних, а також серед аналітиків, фінансистів та розробників [11].

Для збільшення швидкодії критичні секції програмного коду бібліотеки Pandas було написано мовою програмування C. Це дало приріст у швидкодії та виконанні багатьох операції, а також було збережено повну сумісність з мовою Python.

Також потрібно відмітити підтримку статистики. Pandas може автоматично рахувати статистичні результати, групувати по категоріям на основі даних з таблиці. Pandas працює не тільки з числовими даними, а також надає широкі можливості для текстових даних: отримання корисної інформації з них, очищення, виділення ключових слів та інші [11].

2.1.5 Бібліотека асинхронного програмування

Asyncio – це бібліотека для написання асинхронного або конкурентного програмного коду на мові Python. Данна бібліотека особливо корисна при використанні асинхронного підключення до веб-серверів чи вебсайтів, баз даних, читання та запис даних на диск. Тобто там, де очікується відповідь від зовнішнього ресурсу. Тому asyncio дозволяє виконувати іншу корисну роботу,

замість того щоб очікувати відповідь, а коли відповідь прийде, asyncio повертається в точку виклику і продовжує виконувати подальші дії. Завдяки цьому реалізується принцип асинхронного виконання [12].

Асинхронність в запитах до зовнішніх ресурсів дозволяє пришвидшити роботу в декілька разів, наприклад ми можемо одночасно робити декілька запитів до різних вебсайтів і в процесі отримання даних від них, обробляти ці данні [12].

Asyncio та Python дозволяють вам досягнути швидкості роботи із веб-сервером чи базою даних, як наприклад у Go чи Java.

Asyncio оперує такими поняття як:

- цикл подій (EventLoop) керує виконанням різних завдань: зареєстровує надходження завдання, запускає та зупиняє в певний момент;
- корутини (corutines) - спеціальні функції, або задачі, які можуть очікувати виконанання, та мають можливість передавати управління циклу подій, коли вони очікують на відповідь від зовнішнього ресурсу, запускаються вони лише разом з циклом подій;
- футури (futures) – це об'єкти, що зберігають стан корутини або завдання, на даний момент – це може бути вже готовий результат виконаної задачі, або очікування на виконання, або помилки які виникли в процесі виконання завдання.

Ви можете розділити ваш код на корутини, процедури вказати місця де ви очікуєте в самій функції, тоді цикл подій буде знати, коли він може перемикатися на виконання інших задач [12].

Завдяки цьому можна писати дуже швидкий та асинхронний код.

2.1.6 Фреймворк FastAPI

FastAPI – це фреймворк для створення Rest API. Написаний на мові Python. Працює в двох режимах в синхронному та асинхронному – завдяки бібліотеці asyncio. Це досить сучасний фреймворк з підтримкою типів [13] [14].

Основними перевагами є:

- швидкість, завдяки використанню типів та бібліотеки asyncio, веб-сервер на основі FastAPI може обробляти близько 60 тисяч запитів за хвилину;
- швидкість написання програмного коду, завдяки спрощенню деяких вбудованих функцій;
- зменшення кількості помилок при написанні коду, завдяки використанню типізації;
- інтуїтивний, завдяки інтеграції з великою кількістю середовищ розробки, його легко відлагоджувати;
- легкий, фреймоворк спроектований так, щоб його було легко вивчати та використовувати, в цьому допомагає проста документація з великою кількістю прикладів;
- короткий, мінімізація кількості написаних рядків коду, тому що багато функції вже є вбудовані у фреймворк;
- стандартизований, має вбудовану підтримку стандарту OpenAPI.

FastAPI – чудовий інструмент, має високу швидкодію, лаконічність при написані API. А підтримка валідації запитів, дозволить уникнути помилок при динамічній типізації. Автоматично згенерована документація за стандартом OpenAPI, дозволить користувачам легко взаємодіяти з вашим сервісом [13].

2.2 Аналіз використаних додаткових алгоритмів та інструментів

Для успішного вебскрапінгу новинних повідомлень в режимі реального часу, нам потрібно використати декілька інших додаткових алгоритмів та інструментів, які полегшать нам подальшу розробку. Ми будемо використовувати такі алгоритми та інструменти:

- алгоритм ROBULA+ для створення надійних локаторів XPath
- алгоритм розпізнавання посиланнь (URL), чи це посиланням відноситься до новинного повідомлення, чи ні;
- інструменти для отримання відкритих даних та діставання із них дати та часу публікації новинного повідомлення, та інших метаданих;
- регулярні вирази.

2.2.1 Алгоритм ROBULA+ для створення надійних локаторів XPath

Алгоритм ROBULA+ був створений чотирьома науковцями: Мауріціо Леотта, Андреа Стокко, Філіппо Рікка, Паоло Тонелла з університету Генуї. Алгоритм використовують, для того, щоб зменшити кількість зламаних XPath локаторів, під час автоматизованого веб-тестування. Тобто під час тестування веб-додатків, коли змінюється дизайн, або DOM-дерево, XPath локатор також змінюється, тому це стало причиною розробки даного алгоритму, щоб зменшити ручне втручання в програмний код. Для того щоб одразу генерувати надійні локатори на базі XPath локаторів, який під час зміни веб-програми не буде зламуватися, у нових випусках. ROBULA+ знижує крихкість локаторів в середньому на 90% [15].

Ідея полягає в тому що ми будемо робити декілька перетворень, крок за кроком, поки XPath локатор не буде вказувати на один, потрібний нам, вузол DOM-дерева і буде найменшу довжину. Наприклад в нас є XPath локатор

«/html/body/div[1]/section/div/div[2]/main/article/div[1]/div[2]», який вказує на елемент на рисунку 2.1, він є абсолютним. Але при будь-які зміні одного із вузлів, наприклад назва вузла *main* зміниться *div* цей XPath локатор більше працювати не буде, і його потрібно змінювати.

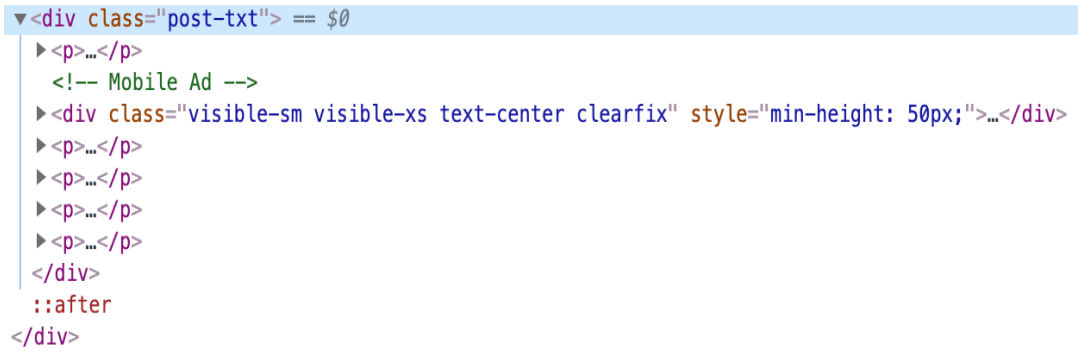

Рисунок 2.1 – Виділений вузол DOM-дерева, на який вказує XPath локатор «/html/body/div[1]/section/div/div[2]/main/article/div[1]/div[2]»

Але якщо цей шлях перетворити за допомогою алгоритма ROBULA+ на більш надійний він матиме наступний вигляд «//div[@class=’post-txt’]». Тобто можна спостерігати, що XPath локатор став більш коротшим та зробив прив’язку до атрибута DOM-дерева, а не до абсолютного шляху, цей локатор зламається лише при умові, якщо буде змінено значення атрибуту, або такий ж самий атрибут буде присвоєний іншому вузлу. Тому він значно надійніший порівняно з абсолютним локатором [15].

Алгоритм ROBULA+ базується на таких перетвореннях:

- **transfConvertStar** - замінює початковий символ * на ім'я тегу вузла DOM-дерева;
- **transfAddID** - додавання атрибута id до вузла;
- **transfAddText** - додавання атрибута text() до вузла, якщо даний вузол містить текст, не великої довжини;
- **transfAddAtribbute** - для кожного наявного атрибута елемента DOM-дерева генерується XPath локатор;

- **transfAddAttributeSe**t – додавання пари атрибутів до локатора, , наприклад «//div[@class='txt' and @name='article']», при умові якщо кількість атрибутів більше ніж один;
- **transfAddPosition** – додавання позиції, тобто якщо на одному рівні декілька вузлів, для полегшення ми можемо вказати позицію цього вузла, наприклад «//div[5]»
- **transfAddLevel** – додаваня вузла верхнього рівня, якщо не можливо визначити елемент, вище зазначеними способами.

Повний алгоритм, а точніше псевдокод алгоритму, можна переглянути на рисунку 2.2.

```
 1. XPath ROBULA+(XPath abs, Document d)
 2. {
 3.   Element e = eval(abs, d);
 4.   List<XPath> xpList = ["//*"];
 5.   while (true)
 6.   {
 7.      XPath xp = xpList.removeFirst();
 8.      List<XPath> temp = [];
 9.      temp.append(transfConvertStar(xp));
10.      temp.append(transfAddID(xp));
11.      temp.append(transfAddText(xp));
12.      temp.appendAll(transfAddAttribute(xp));
13.      temp.appendAll(transfAddAttributeSet(xp));
14.      temp.append(transfAddPosition(xp));
15.      temp.append(transfAddLevel(xp));
16.      for (XPath x : temp)
17.      {
18.         if (uniquelyLocate(x, e, d)) return x;
19.         else xpList.append(x);
20.      }
21.   }
22. }

List<XPath> eval(XPath abs, Document d):
       returns the element in d selected by the XPath locator abs

Boolean uniquelyLocate(XPath x, Element e, Document d):
       TRUE iff eval(x, d) contains only e
```

Рисунок 2.2 – Псевдокод алгоритму ROBULA+

В роботі я використовуватиму алгоритм ROBULA+ для того, щоб перетворювати абсолютний XPath локатор у більш надійний який буде адаптований до змін у DOM-дереві веб-програми. На данний час це мабуть один із найкращих алгоритмів для покращення XPath. Але іноді він може не спрацювати, тому що точність у алгоритму близько 90%. Тоді використовувати абсолютний локатор.

2.2.2 Алгоритм розпізнавання посилань на новинні повідомлення

Даний алгоритм потрібен для того, щоб розпізнавати посилання, які відносяться до посилань на новинне повідомлення. Він досить простий, та базується на простих правилах та аналізі великої кількості веб-посилання, на новинні повідомлення чи статті. Це евристичний алгоритм, який має свої позитивні сторони, а також і недоліки.

Алгоритм виконує перевірки в декілька етапів:

- переконуємося чи URL відповідає стандарту, тобто чи вказана схема передачі даних (http or https), чи вказаний вірно прописаний доменне ім'я вебсайту із вказання домену верхнього рівня і т.д.;
- перевіряємо, чи це посилання не є посилання на медіа, тобто картинку, відео, документ тощо;
- перевіряємо чи вебсайт містить дату в форматі YYYY/MM/DD, YYYY – це рік публікації новини, та число знаходиться в діапазоні від 1990 – 2099. MM – це число місяця, яке знаходиться в діапазоні від 0 до 12, DD – число дня, діапазон від 1 – 31. Новинні вебсайти, дуже люблять використовувати цей шаблон, тому що це дозволяє структурувати новини за датою публікації, а для нас корисно, тому що ми можемо розпізнавати чи цей URL являє собою новинне повідомлення. Для прикладу можна навести вебсайт *cnn.com*, де використовується даний шаблон: https://cnn.com /news/2020/05/9/7251072/;
- наступним кроком ми відділяєм схему передачі даних та домене ім'я від посилання, результат розбиваємо на підрядки по символу «/»,
- кожен підрядок перевіряємо чи він належить до таких найбільш популярно вживаних слів, для посилань на новинні повідомлення, такі як: «news», «article», «post», «press»;

- кожен підрядок перевіряємо чи містить такі символи, як «_» (нижнє підкреслення) та «-» (тире), тому що багато сайтів використовують дані символи як заміну пробілів, для того щоб вказати в URL назву статті чи повідомлення, для швидкої ідентифікації. На прикладі даного посилання: «https://www.nytimes.com/us/glynn-county-police-ahmaud-arbery.html», можна помітити що підрядок «glynn-county-police-ahmaud-arbery.html» розділений через «-» (тире).

Одним із найбільших недоліків – це неможливість розпізнати новинне повідомлення, якщо воно не відповідає цим правилам. Тому його потрібно розвивати та доповнювати новими правилами.

В роботі ми його будемо використовувати, для розпізнавання та аналізу посилання на статтю чи новину. Він досить простий в написані та дає досить високу точність розпізнавання новин.

2.2.3 Аналіз інструментів для отримання відкритих даних

Відкриті дані – це колекція спільних словників, які використовуються для розмітки своїх веб-сторінок, щоб їх зрозуміли основні пошукові двигуни: Google Search, Microsoft Bing Search, Yandex Search, Yahoo Search тощо. Дані словники описані на вебсайті schema.org [16].

Однією із причин виникнення таких словників - те що звичайний HTML-тег не несе в собі ніякої корисної інформації про текст, який обгорнутий ним. Це ускладнює пошуковим двигунам інтелектуальний пошук на веб-сторінці, тому сторінка може не попадати в результат. При використані таких словників, збільшується розуміння про публікацію, користувачам пошукових двигунів, видається більш релевантна та детальна інформація. Для прикладу за допомогою додаткових атрибутів, для HTML тегів, із словників можна вказати на веб-сторінці назву книги, автора, вартість, кількість сторінок тощо. Тоді при індексації сторінки, пошуковий двигун запам’ятає ці параметри та буде знати

додаткову інформацію, для того щоб відображати її користувачам. Користувач, коли буде шукати книгу, одразу зможе побачити основну інформацію про книгу, і зможе перейти на сайт для більш детального ознайомлення.

Виділяють такі основні види відкритих словників: Microdata, RDFa, JSON-LD.

Microdata – це спосіб семантичної розмітки, використовуючи стандартні елементи мови розмітки HTML. Описує власний словник атрибутів HTML елемента, де атрибут вказує на семантичне значення. Ця розмітка більш пристосована до машин, для того щоб полегшити їм роботу, в порівняні з іншими. Приклад розмітки зображено на рисунку 2.3 [16]. Він оперує основними атрибутами:

- itemtype містить значення цього атрибуту містить посилання на словник, що описує елемент;
- itemprop містить значення, що описують властивості елемента із словника, тобто HTML тег містить дані, які описують елемент що міститься в атрибуті itemtype;
- datetime – це атрибут, що описує дату у форматі ISO 8601.

```
<section itemscope itemtype="http://schema.org/Person">
    Hello, my name is
    <span itemprop="name">John Doe</span>,
    I am a
    <span itemprop="jobTitle">graduate research assistant</span>
    at the
    <span itemprop="affiliation">University of Dreams</span>.
    My friends call me
    <span itemprop="additionalName">Johnny</span>.
    You can visit my homepage at
    <a href="http://www.JohnnyD.com" itemprop="url">www.JohnnyD.com</a>.
    <section itemprop="address" itemscope itemtype="http://schema.org/PostalAddress">
        I live at
        <span itemprop="streetAddress">1234 Peach Drive</span>,
        <span itemprop="addressLocality">Warner Robins</span>,
        <span itemprop="addressRegion">Georgia</span>.
    </section>
</section>
```

Рисунок 2.3 – HTML код, що містить атрибути розмітки Microdata

RDFa (Resource Description Framework in Attributes) – формат опису метаданих рекомендований консорціумом W3C (World Wide Web Consortium).

Він подібний по опису до Microdata, але оперує більшою кількістю атрибутів, та більшим описом. Також як і Microdata посилається на словники shema.org, але також може містити посилання на словники www.w3.com. Приклад розмітки зображено на рисунку 2.4 [17].

```
<div xmlns:dc="http://purl.org/dc/elements/1.1/"
  about="http://www.example.com/books/wikinomics">
  <span property="dc:title">Wikinomics</span>
  <span property="dc:creator">Don Tapscott</span>
  <span property="dc:date">2006-10-01</span>
</div>
```

Рисунок 2.4 – HTML код, що містить атрибути розмітки RDFa

JSON-LD (JavaScript Object Notation for Linked Data) – це один із способів представлення даних які зв'язані між собою. Він використовує спосіб представлення даних – JSON. Зазвичай на сайті його вкладають в тег <script application="json-ld">, із спеціальним атрибутом *application* та значенням атрибута *"json-ld"*. Пошукові двигуни та роботи, отримують всі значення вже готовому і структурованому вигляді, що відрізняється від Microdata та RDFa. Тому, останнім часом, JSON-LD витісняє інші способи розмітки даних, із-за своєї зручності та простоти використання, як для пошукових двигунів так і для веб-розробників, тому що всі потрібні дані потрібно розмістити лише в одному місці. JSON-LD рекомендується до використання консорціумом W3c. Приклад розмітки метаданих зображено на рисунку 2.5 [18].

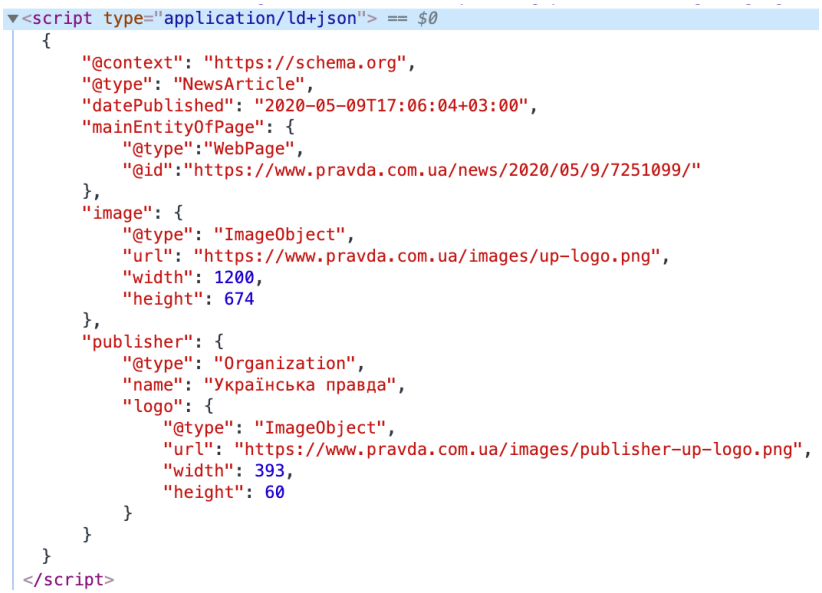

Рисунок 2.5 – Приклад розмітки JSON-LD на веб-сторінці.

На даний момент, більшість вебсайтів використовують всі три види розмітки метаданих, для того щоб охопити максимальну кількість пошукових двигунів, а тим самим і користувачів. У своїй роботі, я буду використовувати метадані для того, щоб ідентифікувати час публікації, автора, ключові слова, та інші метадані новинного повідомлення, це збереже час розробникам, та збільшить функціональність системи. Що свою чергу зробить її більш універсальною та стійкою до змін вебсайту [18].

2.2.4 Регулярні вирази

Регулярні вирази (або мова регулярних виразів) – спеціальна мова для опису шаблонів рядків. Дуже сильний і потужний інструмент для пошуку підрядків в рядку за заданим шаблоном. Має свій набір символів та метасимволів, що описують мову регулярних виразів. Застосовую у різних сферах, від пошуку в звичайних текстах, до лексичного аналізатора в компіляторах. В основі регулярних виразів знаходяться: теорія формальних мов, теорія автоматів, а також класифікація формальних граматик та мов Хомського. По суті регулярний вираз це скінчений автомат. Популярність серед користувачі регулярні вирази отримали після того, як Кеннет Томпсон вбудував їх в текстовий редактор операційної системи UNIX [19].

Мова регулярних запитів мала та досить обмежена у своїй функціональності, і не може застосовуватися скрізь. Що накладає деякі обмеження на використання регулярних виразів [19].

Також іноді виникає проблема в тому що регулярний вираз може бути занадто складним, і прочитати його та зрозуміти підсилу лише тому, хто його написав. В такому випадку слід відмовитися від використання регулярних виразів, та скористатися іншими інструментами, інакше виникне проблема з підтримкою та відлагодженням [19].

Для прикладу давайте розглянемо простий доволі регулярний вираз «\d{4}-\d{2}-\d{2}», даний шаблон відповідає тексту «2020-05-09», тобто цей регулярний вираз гарно підходить для того щоб шукати в тексті дати. Скінчений автомат для даного виразу зображено на рисунку 2.6. «\d» - означає знайти будь-яку цифру і він також відповідає наступному шаблону «[0-9]». «{4}» – цей вираз біля «\d» відповідає повторити згадування 4 рази лівостороннього виразу, тобто його можна замінити наступним виразом «\d\d\d\d». Звідси виходить «\d\d\d\d» рівносильний «\d{4}», що означає доволі простий вираз – «знайди мені чотири підряд цифри». Такі регулярні вирази доволі прості та зрозумілі.

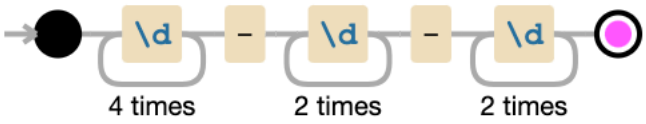

Рисунок 2.6 – Скінчений автомат для регулярного виразу «\d{4}-\d{2}-\d{2}»

2.2.5 Висновки до розділу

Отже в даному розділі ми розглянули інструменти та додаткові алгоритми, які будуть корисні під час проектування та розробки системи. Ці інструменти є дуже корисним та можуть розглядатися не тільки для парсингу новинних повідомлень, а також в розробці інших програм в різних сферах.

3. ОПИС АЛГОРИТМУ ТА АРХІТЕКТУРНИХ РІШЕНЬ РОЗРОБЛЕНОЇ СИСТЕМИ

Для того щоб система стабільно працювала та не виникало проблем під час її роботи, потрібно продумати архітектуру розробленої системи. Також потрібно продумати гарно реалізацію алгоритму та написати достатню кількість тестів для перевірки дієздатності програми.

Отже для того щоб парсити велику кількість вебсайтів та щоб парсер автоматично адаптувався під зміни вебсайта, а також для того щоб мінімізувати втручання розробником в його роботу, користуючись правилом «розділяй та володій», ми розіб'єм нашу задачу парсингу новинних вебсайтів на дві підзадачі, та створимо розподілену систему:

- перша – це побудова системи взаємодії з використанням REST API, а саме OpenAPI, для взаємодії з ядром системи – алгоритмом створення шаблону, для новинного вебсайту;
- друга – це побудова система яка використовуючи API, буде отримувати шаблон певного вебсайту, та витягувати дані з вебсайту, користуючись шаблоном для цього вебсайту.

Така стратегія надає нам декілька переваг:

- якщо вебсайт змінився то системі достатньо перебудувати шаблон, або користувач за допомогою API, може сам створити шаблон, якщо система створила хибний або не робочий шаблон;
- динамічно додавати або видаляти, новинні вебсайти, не перезавантажуючи систему;
- збільшувати кількість парсерів, які використовують шаблон, для отримання даних, тобто, коли в нас збільшується кількість вебсайтів, і парсер не встигатиме їх всіх обходити в режимі реального часу, ми можемо підключати ще один парсер, який користуючись шаблонами через API, може паралельно виконувати

роботу, їх кількість може бути необмежена, завдяки цьому можна вийти на новий рівень швидкості отримання даних;

- через API можна перевіряти роботу та точність створення шаблонів, а також перевіряти чи взагалі є можливість парсити даний вебсайт, тому що на деяких вебсайтах є захист, і тому потрібно до цього ставитися з порозумінням, що власник вебсайту не хоче, щоб його інформація була агрегована;
- API та парсер можуть працювати на різних серверах чи комп'ютерах, для обміну даних використовувати глобальну мережу Internet, це дозволяє будувати повністю розподілену систему, що підвищує стійкість системи;
- API з використанням стандарту OpenAPI, надає універсальний стандарт, що полегшить розширення функціональності в майбутньому, та не зламає функціоналу який вже реалізований;

Також таке архітектурне рішення має свої недоліки:

- якщо API не буде відповідати на запити від парсера, парсер буде очікувати відповідь, і не зможе збирати дані;
- для додаваня, видалення та перевірки ресурсів, потрібне втручання людини;
- при побудові розподілений архітектури потрібно використовувати декілька серверів чи комп'ютерів, що є досить затратним.

3.1 Опис алгоритму для створення шаблону новинного вебсайту

3.1.1 Отримання тексту новинного повідомлення

У HTML є два види відображення тегів, які називаються блокові елементи та вбудовані елементи. Елементи блоку, завжди починається з

нового рядка та відповідають за те щоб розділяти сторінку на умовні блоки, це потрібно для зручності роботи зі сторінкою веб-розробників. Вбудовані елементи відповідаю за редагування рядків в середині блоків, це може бути наприклад робота із текстом, і т.д. Вбудовані елементи завжди знаходяться в середині блокових елементів [20].

До блокових елементів відносяться такі теги: <address>, <article>, <aside>, <blockquote>, <canvas>, <dd>, , <dl>, <dt>, <fieldset>, <figcaption>, <figure>, <footer>, <form>, <h1>-<h6>, <header>, <hr>, <li>, <main>, <nav>, <noscript>, <ol>, <p>, <pre>, <section>, <table>, <tfoot>, <ul>, <video> [20].

До вбудованих елементів відносяться наступні теги: <a>, <abbr>, <acronym>, <b>, <bdo>, <big>,
, <button>, <cite>, <code>, <dfn>, , <i>, <img>, <input>, <kbd>, <label>, <map>, <object>, <output>, <q>, <samp>, <script>, <select>, <small>, , , <sub>, <sup>, <textarea>, <time>, <tt>, <var> [20].

Далі ми будемо використовувати цю особливість для побудови шаблону. Розглянемо принцип побудови шаблону для новинного повідомлення, який реалізовано в ядрі системи.

Для початку потрібно отримати HTML новини використовуючи пряме посилання на новину. Після того як HTML код був отриманий від веб-сервера новинного ресурсу. Далі ми виконуємо наступні кроки:

- парсимо HTML за допомогою Python бібліотеки BeautifulSoup (див 2.1.2), та отримує DOM-дерево;
- проводимо очищення DOM-дерева від не потрібних тегів, які відповідають за дизайн та вигляд вебсайту, такі як: <script>, <style>, <css>, <noindex>
- виділяємо з дерева блоки що містять текстову інформацію, та вбудовані елементи;

- знаходимо абсолютний XPath локатор для кожного блока, який вказує на блок;
- знаходимо батьківський абсолютний XPath локатор, кожного блока, це об’єднує групу блоків в один, тому що більшість вебсайтів розділяють текст на декілька блоків (на абзаци);
- застосовуємо алгоритм ROBULA+ (див 2.2.1) до кожного блоку з абсолютним XPath локатором, знаходимо більш надійний XPath локатор;
- групуємо блоки з однаковим батьківським XPath локатором, і робимо один блок;
- рахуємо об’єм тексту, тобто кількість символів для кожного блоку;
- сортуємо за об’ємом тексту;
- обираємо той блок який містить найбільший об’єм тексту;
- результуючий XPath локатор буде вказувати на знаходження тексту новинного повідомлення;

Даний підхід дозволяє знайти знаходження тексту новинного повідомлення без ручного втручання та побудувати надійний XPath локатор, навіть, іноді, кращий ніж людина. Текст буде отриманий набагато кращий, ніж якби ми збирали весь текст зі сторінки. Цей метод дозволяє відфільтровувати зі сторінки весь не потрібний текст: сторонні посилання, різні коментарі, назви категорій вебсайту, різні бокові меню тощо. Також це допомагає нам позбавитися від втручання людини, і використовувати даний інструмент для збору новин без написання окремого парсера для окремого вебсайту. Для прикладу можна глянути на результат роботи алгоритму на рисунку 3.1. Як можна побачити текс зберігає всі відступи та посилання, тим самим покращується його сприйняття.

Рисунок 3.1 - Зліва новинний вебсайт, а справа отриманий готовий текст за допомогою алгоритму.

3.1.2 Отримання заголовку новинного повідомлення

Наступним кроком нам потрібно отримати заголовок новини. Це одна із простих та досить тривіальних задач, тому що заголовок новин знаходиться в окремому HTML тегу «title». Він зазвичай один і знаходить на початку HTML документу, це спрощує нам роботу, тому що всі вебсайти вказують заголовок таким способом. На рисунку 3.2 наведено приклад html коду який містить заголовок новини.

```
▼<title>
    "В Google Lens теперь можно скопировать рукописный текст и отправить его на ПК – ITC.ua"
  </title>
```

Рисунок 3.2 - Приклад оформлення тегу «title» новинного вебсайту

3.1.3 Отримання часу публікації новинного повідомлення

В нас залишається лише одне з не вирішене питань – це час публікації новинного повідомлення. Це мабуть одна з найбільших проблем, тому що важко забирати час зі сторінки, коли на кожній сторінці відображається час по різному деякі вказують повну дату публікації, хтось вказує скорочену дату без конкретної години та хвилини. Також на кожному вебсайті час публікації розміщується по-різному, це в свою чергу накладає додаткові проблеми при парсингу новинного вебсайту.

Тут є два способи вирішити дану проблему, перший це – прописувати все вручну, тобто описувати XPath локатор вузла де знаходиться текст, і вказувати формат тексту, або використовувати відкриті дані(див. 2.2.3), якими на даний час успішно користуються пошукові роботи типу Google, Yahoo тощо. Тобто потрібно створити правила, по яким ми будемо отримувати час з метаданих, тому що на даний час не існує одного єдиного стандарту відкритих даних, і більшість сайтів можуть використовувати один стандарт, або два, а то і навіть три одночасно.

Останнім часом популярність завойовує JSON-LD. Тому що має досить великий об’єм параметрів, та зручність у використанні. Візьмемо для прикладу JSON-LD новинного вебсайта та глянемо які данні нам можуть бути доступні (Рисунок 3.3). Але потрібно пам’ятати, що використання чужих даних без вказівки автора або джерела, карається законом та не є етичним (Див 1.1).

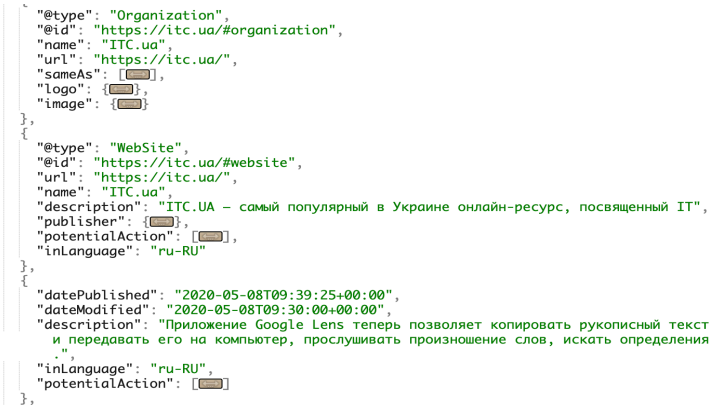

Рисунок 3.3 – Приклад JSON-LD даних новинного ресурсу

Отже, якщо глянути на дані JSON-LD (Див. Рисунок 3.3), ми можемо отримувати велику кількість інформації, а саме:

- час публікації новинного повідомлення (datePublished);
- останній час модифікації новинного повідомлення (dateModified);
- логотип вебсайту (logo);
- автора публікації;
- повну назву ресурсу;
- короткий опис новинного повідомлення;
- мову на якій написано.

Варто зазначити те, що час публікації та час модифікації вказаний форматі ISO8601, що є досить зручним у використанні і підтримується майже всіма мовами програмування.

Також іноді вебсайти додають ключові слова у JSON-LD для того щоб полегшити роботу пошуковим роботам. Тому за свою зручність ми будемо орієнтуватися на використання даних із JSON-LD [18].

RDFa також надає досить зручний формат отримання часу публікації, назву вебсайту тощо. Давайте розглянемо які дані можна отримати з RDFa на рисунку 3.4 показано дані які можна отримати із RDFa.

```
<meta name="description" content="Приложение Google Lens теперь позволяет копировать рукописный текст и
передавать его на компьютер, прослушивать произношение слов, искать определения.">
<meta name="robots" content="index, follow">
<meta name="googlebot" content="index, follow, max-snippet:-1, max-image-preview:large, max-video-preview:
-1">
<meta property="og:locale" content="ru_RU">
<meta property="og:type" content="article">
<meta property="og:title" content="В Google Lens теперь можно скопировать рукописный текст и отправить его на
ПК - ITC.ua">
<meta property="og:description" content="Приложение Google Lens теперь позволяет копировать рукописный текст
и передавать его на компьютер, прослушивать произношение слов, искать определения.">
<meta property="og:url" content="https://itc.ua/news/v-google-lens-teper-mozhno-skopirovat-rukopisnyj-tekst-
i-otpravit-ego-na-pk/">
<meta property="og:site_name" content="ITC.ua">
<meta property="article:publisher" content="https://www.facebook.com/ITC.UA">
<meta property="article:author" content="https://www.facebook.com/vadim.karpus">
<meta property="article:published_time" content="2020-05-08T09:39:25+00:00">
<meta property="article:modified_time" content="2020-05-08T09:30:00+00:00">
<meta property="og:image" content="https://i0.wp.com/itc.ua/wp-content/uploads/2020/05/dsc04857_2.0.jpg?
fit=2050%2C1369&quality=100&strip=all&ssl=1">
<meta property="og:image:width" content="2050">
<meta property="og:image:height" content="1369">
```

Рисунок 3.4 – Приклад RDFa даних новинного ресурсу

Даний приклад (Рисунок 3.4) містить такі ж самі данні як і JSON-LD, але які є вбудованим у відповідні HTML теги, і мають трохи інший формат відображення, описаний у вигляді атрибутів. Час публікації описаний через атрибут *property=”article:published_time”,* але іноді значення атрибуту *property* може змінювати, а сам час публікації вказаний в полі *content*. Формат часу також відповідає стандарту ISO8601. Але для того щоб отримати ці данні – потрібно додатково обходити DOM-дерево. Тому це менш зручний спосіб, порівняно з JSON-LD, де можна отримати дані за певним ключом майже миттєво. Але якщо у вебсайта немає JSON-LD, тоді ми проходимо по DOM-дерево, і можемо отримати час публікації новини.

У формату відкритих даних Microdata є досить зручна перевага над RDFa, тому що Microdata використовують спеціальний атрибут *datetime* для того щоб вказати час публікації, значенням якого є час у форматі стандарту ISO8601. Для прикладу на рисунку 3.5 зображено як вказується час на веб-сторінці. Через Microdata також відображають інші метадані: назва вебсайта, автора публікації, короткий опис тощо [16][17]. (Див 2.2.3)

```
<span class="published" datetime="2020-05-08T12:39:25+03:00">
08.05.2020 в 12:39 </span>
```

Рисунок 3.5 – HTML тег з атрибутом datetime.

Вебсайти заради того, щоб попадати на перші сторінки видачі пошукових систем вказують велику кількість метаданих, які є відкритими та можна використовувати. Відкриті дані допомагають полегшити нам велику кількість роботи по парсингу новинних вебсайтів, тому що вони є готові та розмічені за правилами. Але є такі вебсайти які не використовують жодний із перелічених вище типів відкритих даних. Тому потрібно передбачити

використання ручного редагування, та можливість вказати XPath локатор де знаходиться час публікації на сторінці, та формат відображення часу.

3.2 Опис реалізації розробленої системи

Так як я використав розподілену систему, тоді опишемо окремо структуру API для створення шаблону, та окремо структуру парсеру, який працює на основі шаблонів.

3.2.1 Опис реалізації API для створення шаблонів

API реалізує наступні основні функції:

- створення шаблона для доданого ресурсу;
- зберігання шаблону в БД (База Даних);
- зберігання ресурсу в БД;
- надає ресурси для парсера;
- надає шаблон для певного ресурсу для парсера;
- виконує перевірку на наявність часу публікації в новині;
- виконує перевірку XPath локатора, для отримання часу публікації

API було побудовано з використанням бібліотеки FastAPI, яка має підтримку стандарту OpenAPI.

Розглянемо наступну структуру модулів та файлів API, яка зображена на рисунку 3.6.

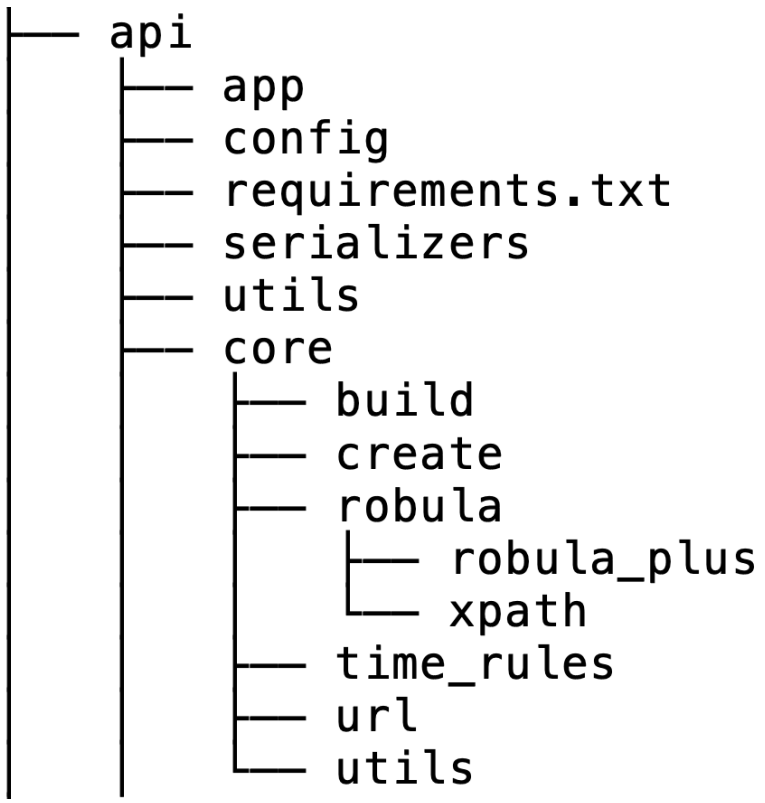

Рисунок 3.6 – Структура модулів та файлів API

API складається з наступних файлів та модулів:

- файл **app** – в ньому описано основні методи API для взаємодії, а саме: додавання та видалення ресурсів, створення, корегування та видалення шаблону, зберігання шаблонів в БД, надання шаблонів по запитам;
- файл **config** – містить змінні для конфігурації API;
- файл **requirements.txt** – містить список залежних бібліотек, які потрібно інсталювати перед використання API;
- файл **serializers** – містить описані моделі, ресурсів та шаблонів;
- файл **utils** – містить допоміжні функції;
- модуль **core** – основне ядро системи, який реалізує побудову шаблону для доданого ресурсу.

Розглянемо модуль **core** більш детальніше. Він містить наступні файли та модулі:

- файл **build** – реалізує методи для побудови текстового шаблону, тобто знаходить XPath локатор для тексту;
- файл **time_rules** – реалізує правила по яким можна знаходити час публікації за допомогою відкритих даних;
- файл **url** – реалізує метод розпізнавання URL(посилання) на новинний ресурс;
- файл **utils** – реалізує допоміжні функції: збирання всіх посилань зі сторінки, нормалізація посилань тощо;
- модуль **robula** – реалізує методи для роботи алгоритму ROBULA+ (Див. 2.2.1).

Завдяки використанню FastAPI, ми отримуємо готову графічну оболонку з якою можна користувач може взаємодіяти через браузер (Рисунок 3.7) [12].

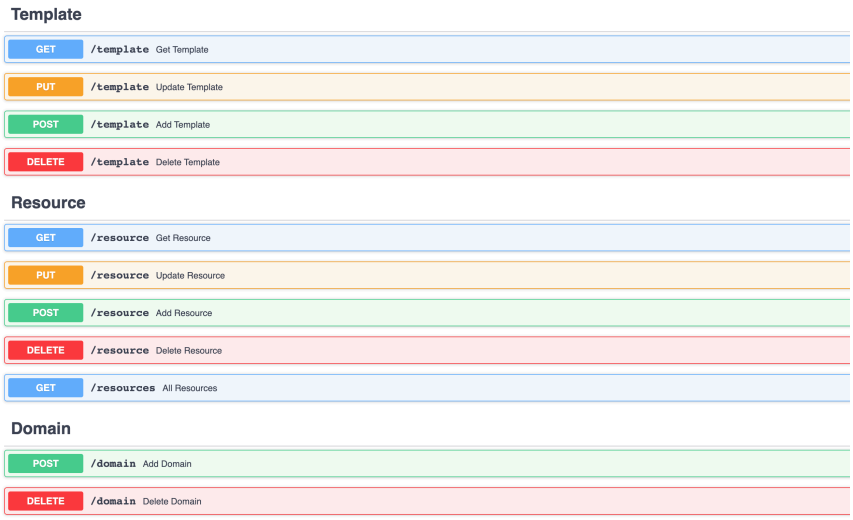

Рисунок 3.7 – Графічна оболонка, яка згенерована FastAPI.

Результуючий шаблон будемо зберігати у вигляді JSON в БД. База даних може використовуватися будь-яка, тому що шаблон має дуже простий вигляд (Рисунок 3.8).

```
{
    "text": {
        "path": "//div[@class='text']"
        "type": "xpath"
    },
    "time": {
        "path": "rule_1",
        "type": "open_data"
    },
    "domain": "https://www.cnn.com/"
}
```

Рисунок 3.8 – Приклад результуючого шаблону у форматі JSON

3.2.2 Опис реалізації та структури парсера, який працює на основі шаблонів

Парсер реалізує та виконує наступні функції:

- отримання списку вебсайтів з API, які потрібно обійти;
- отримання шаблону для вебсайту з API;
- обхід всіх сайтів та отримання посилань на новини;
- парсинг новинного повідомлення відповідно до шаблону;
- вивантаження результату парсингу в БД;
- кешування посилань, які вже було оброблено.

Структура файлів та модулів парсера, зображено на рисунку 3.9.

```
parser
├── config
├── main
├── pattern
│   ├── api
│   ├── getter
│   ├── time_rules
│   ├── url
│   └── utils
└── requirements.txt
```

Рисунок 3.9 – Структура файлів та модулів парсера.

Вона досить проста та складається з наступних файлів та модулів:

- файл **config** – в ньому описані основі конфігураційні зміні такі як: адреса API, логін та пароль для доступу, БД тощо;
- файл **main** – реалізує основні функції для асинхронного отримання даних із вебсайтів, з використання бібліотеки Asyncio, та надсилання результату парсингу в БД чи у файл;
- модуль **pattern** – реалізує парсинг даних та взаємодію з API;

В свою чергу модуль **pattern** складається з файлів:

- файл **api** – реалізує взаємодію з API, отримання вебсайтів та шаблонів, через мережу;
- файл **getter** – реалізує методи парсингу HTML за шаблоном, з використанням бібліотеки lxml, очищення тексту та отримання часу публікації;
- файл **time_rules** – реалізує правила по яким можна знаходити час публікації за допомогою відкритих даних;
- файл **url** – реалізує метод розпізнавання URL(посилання) на новинний ресурс;
- файл **utils** – реалізує додаткові функції а також функції кешування посилань, які були відвідані.

Парсер має не перевантажену структуру, це означає, що він матиме високу швидкодію, яку можна в майбутньому легко підтримувати та відлагоджувати. Використання бібліотеки асинхронного програмування – Asyncio, зменшило розмір програмного коду в декілька раз, та збільшилося навантаження на мережу, так як запити надсилаються паралельно.

3.3 Висновки до розділу

В даному розділі ми розглянули сам алгоритм побудови шаблонів, який не потребує втручання людини та може, будувати самостійно шаблон для новинного вебсайту, і лише іноді коли шаблон буде створено не вірно, користувач може самостійно створити його, використовуючи API. Розглянули структуру API та парсера, яка є досить не великою, та простою. Це спрощувати розробку та впровадження нового функціоналу в майбутньому.

4. ТЕСТУВАННЯ ТА РЕКОМЕНДАЦІЇ ДЛЯ ПОДАЛЬШОГО ВДОСКОНАЛЕНЯ РОЗРОБЛЕНОЇ СИСТЕМИ

4.1 Тестування системи

В наш час тестування програмного забезпечення виділяється досить велика увага. І від якісного тестування залежить вся доля розробленого продукту. Тому що лише під час тестування можна встановити відповідність готової програми з технічним завданням, що в процесі тривалого навантаження ніякі системи не відмовляють та будуть працювати стабільно без втручання в їх роботу.

Великі компанії наймають цілі підрозділи людей які займаються тестування програмного забезпечення, для того щоб гарантувати стабільність роботи. QA (Quality Assurance) інженери описують велику кількість тестів для того щоб покрити повністю весь функціонал, та гарантувати, що продукт буде стабільно працювати. Автоматизація тестування дозволяє нам виявляти всі помилки при внесені змін у функціонал системи, що полегшить роботу нам в майбутньому.

До тестування розробленої звернемо велику увагу на точність отриманих шаблонів, тобто кількість вірно створених шаблонів від загальної кількості новинних вебсайтів, та стабільність роботи всієї системи. Проведення автоматичного тестування дозволить нам виявити всі недоліки та переваги програмного забезпечення.

Основними типами тестування буде навантажувальне тестування та тестування точності розробленого програмного забезпечення. Навантажувальне тестування дозволить перевірити дієздатність та швидкодію системи та окремих компонентів під великим навантаженням.

Виділимо наступні пункти для тестування:

- тестування часу для побудови шаблону;

- тестування часу відповіді від API;
- тестування всіх методів API;
- тестування часу на парсинг новин за шаблоном;
- тестування точності створених шаблонів;
- тестування API при великій кількості парсерів;
- інтеграційне тестування.

Тестування даних компонентів дозволяє зробити перевірку на відповідність всім вимогам.

Для тестування всіх методів та роботи функціоналу API, використовуватиметься димне тестування, або коротке тестування, для того щоб перевірити коректність роботи API.

Під час тестування на швидкість парсингу новинного повідомлення за шаблоном, було встановлено, що швидкість залежить лише від того наскільки швидко дані будуть передані від веб-сервера новинного вебсайта, тобто якщо веб-сервер перевантажний запитами від користувачів, то ми можемо дуже довго очікувати на відповідь.

Навантажувальне тестування дозволяє спостерігати швидкість відповіді від API при великій кількості парсерів. Звідси ми можемо вираховувати додаткові затримки які будуть виникати при збільшені кількості парсерів. При запущених 5 парсерах, які одночасно запитували дані від API, навантаження на API по розпізнаванню шаблонів, було мінімальне, отже API може підтримувати велику кількість парсерів одночасно.

Ручне тестування не доцільно використовувати, при тестуванні великого набору новинних вебсайтів, для виміру точності побудови шаблону. Також недоцільно перевіряти вручну кожен вебсайт – це займатиме велику кількість часу. Тому доцільно використовувати автоматичне тестування на великій кількості вебсайтів, як мінімум 100 вебсайтів.

Для того щоб отримати точність побудови шаблонів будемо притримуватися наступних дій:

- додаємо новинні вебсайти через API;
- очікуємо завершення побудови шаблонів для кожного вебсайту;
- запускаємо парсер на доданих вебсайтам, якщо шаблон створено не вірно, то не можливо буде парсити новинні повідомлення з даного вебсайту;
- порівнюємо кількість успішно оброблених персером вебсайтів до загальної кількості, звідси отримаємо приблизну точність роботи алгоритму з побудови шаблонів.

Отже при дотримані вище згаданих дій, при тестування на ста найбільш популярних новинних вебсайтах України, автоматично було створено 100 шаблонів. 89 шаблонів були створені вірно та успішно отримано текст та час новинного повідомлення. 11 інших шаблонів не виділили текст новинного повідомлення, тобто XPath локатор для тексту вказував на неіснуючий вузол в DOM-дереві, що не дало змогу обробити новинні повідомлення для вебсайтів. Тобто за результатами тестування, точність складає 89%. Варто зазначити, що при збільшені кількості вебсайтів точність може змінювати, в залежності від якості вебсайтів, та чи присутні на сайтах засоби проти веб-парсингу.

4.2 Користування системою

Для розгортання та роботи із системою потрібно виконати наступні кроки:

1. Встановити інтерпретатор мови програмування Python версії 3.8.
2. Перейти в папку *api.*
3. Виконати інсталяцію додаткових бібліотек командою «pip install –r requirements.txt».

4. Запустити веб-сервер по створенню шаблонів командою «uvicorn app:app –host=localhost –port=8000»
5. Перейти в папку *parser.*
6. Виконати інсталяцію додаткових бібліотек командою «pip install –r requirements.txt».
7. Перейти в браузер за адресою «http://localhost:8000/docs»
8. В розділі «Domain» натиснути на кнопку «Add domain» (рисунок 4.2).
9. В розгорнутому вікні натиснути клавішу «Try it out» і в полі «domain» написати адресу новинного вебсайту та натиснути кнопку «Execute» (рисунок 4.3).
10. У разі успішності в полі «Responses» з’явиться JSON з категоріями із створеними шаблонами, для кожної категорії.
11. Перейти в папку *parser.*
12. Для запуску парсера потрібно виконати команду «python main.py»
13. Результуючі новини які були спарсені з’являться у файлі «result.json» у форматі JSON.
14. Щоб подивитися на створений шаблон потрібно перейти на розділ «Template» та натиснути на кнопку «Get template» (рисунок 4.4).
15. В розгорнутому вікні натиснути клавішу «Try it out» і в полі «url» написати адресу вебсайту, або адресу категорії новинного ресурсу та натиснути кнопку «Execute» (рисунок 4.4).
16. В полі «Responses» з’явиться JSON шаблону (рисунок 4.5), або повідомлення про те, що шаблон не знайдено (рисунок 4.6).

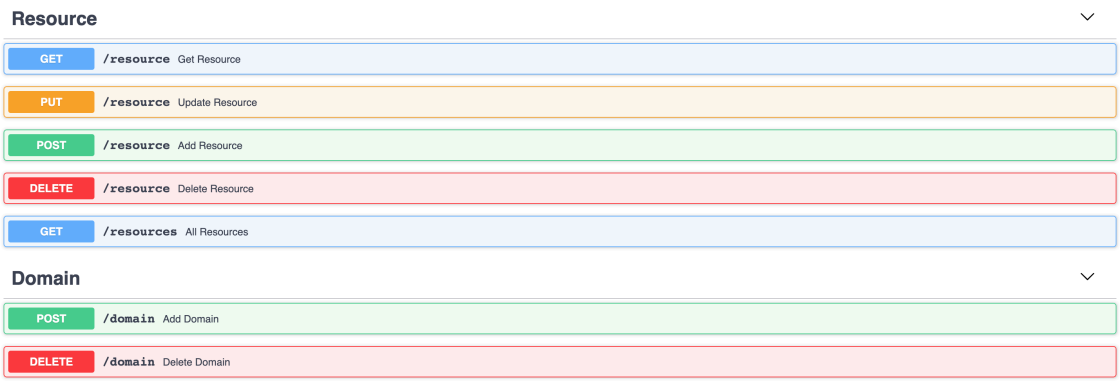

Рисунок 4.2 – Інтерфейс API

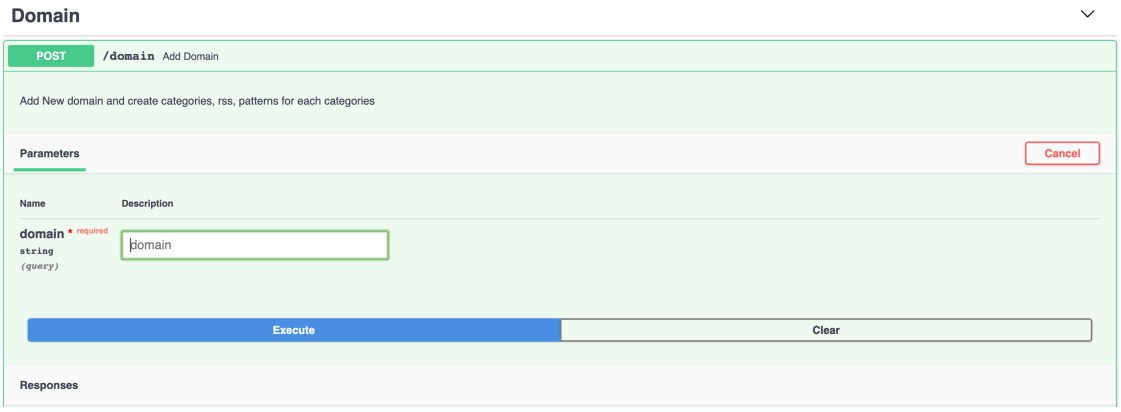

Рисунок 4.3 – Розгорнуте меню «Add Domain»

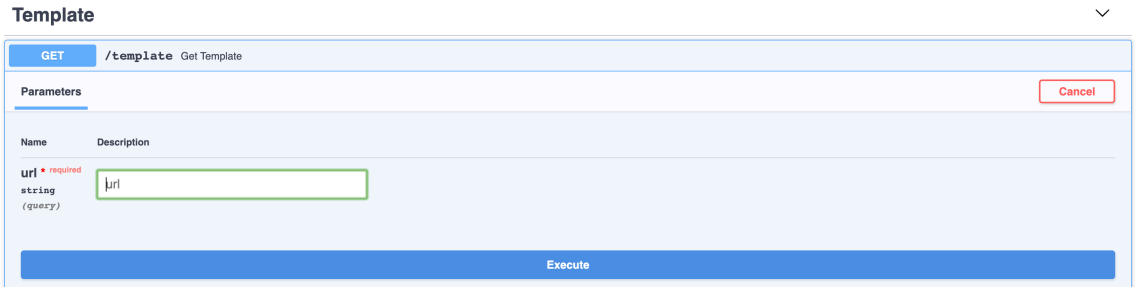

Рисунок 4.4 – Розгорнуте меню «Get Template»

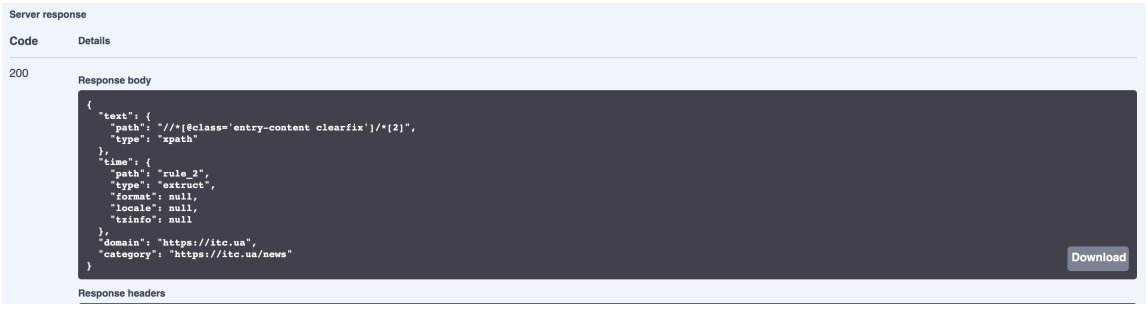

Рисунок 4.5 – Результуючий шаблон у форматі JSON

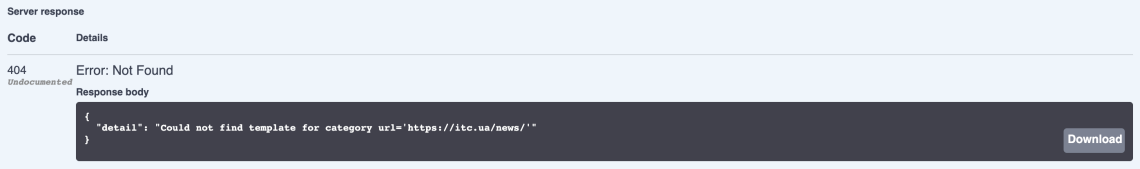

Рисунок 4.6 – Повідомлення про помилку якщо шаблон не було знайдено.

4.3 Рекомендації для подальшого вдосконалення

Данна розроблена система має як і велику кількість переваг, так і недоліків, які можна вдосконалити. Розвиток системи дозволить покращити роботу системи та її точність.

Для того щоб збільшити точність системи можна використати додаткові методи для очистки вхідного HTML, тобто система буде менше помилятися, при створені XPath локатора. Це збільшить точність побудови шаблона, і зменшить кількість ручного вручання в систему. Також можна добавити, декілька нових правил по парсингу, так як це реалізовано в бібліотеці newspaper (Див 1.5).

Також можна додати забір даних із RSS-каналів (Really Simple Syndication), що зменшить витрати часу на пошук та збір посилань із вебсторінки [21].

На даний момент дуже швидко розвиваються пошукові системи, в яких застосовують методи машинного навчання та штучні нейронні мережі для покращення видачі користувачам, і тому для їх більш точної роботи Google, Yandex та інші, стараються просувати новий стандарт відкритих даних JSON-LD, де можна вказувати повний текст новини (рисунок 4.7), що сотні раз полегшує роботу пошуковим роботам, тому що їм не потрібно парсити HTML, але цим користуються дуже мало новинних вебсайтів. Тому

Рисунок 4.7 – Приклад повного тексту новинного повідомлення в JSON-LD, знаходиться за ключем *articleBody*

Так як в проєкті реалізовано додавання новинних джерел, лише в ручному режимі, також можна добавити в парсер автоматичне наповнення джерела, тобто парсер аналізуючи новинні повідомлення знаходить посилання на інші новинні вебсайти, робить запит до API та перевіряє чи є даний ресурс в БД, якщо немає то автоматично додає його для створення шаблону.

Також можна підвищити, швидкодію API за рахунок використання асинхронних запитів, тоді потужність виросте на декілька порядків, так само як це реалізовано в парсері, який робить запити до новинних вебсайтів асинхронно. Асинхронні запити до бази даних, де зберігаються шаблони, допоможуть використати по максимуму всі ресурси БД.

Створення нового графічного інтерфейсу дозволить користувачам більш зручніше взаємодіяти із системою, та управляти доданими ресурсами та шаблонами, графічний інтерфейс який надається фреймворком FastAPI, відповідав мінімальним вимогам, і тому був обраний. Але в подальшому вдосконаленні, можна обрати більш зручніший та функціональний інтерфейс.

ВИСНОВКИ

У ході дипломного проєкту було розроблено програмне забезпечення для розбору новинних повідомлень в режимі реального часу. Також розглянуто проблеми та різні сторони веб-парсингу.

Використання сучасних інструментів дозволило написати одночасно швидкий та невеликий програмний код, який дуже легкий в підтримці. Серед таких інструментів було взято за основу мову програмування Python та додаткові бібліотеки. Данна мова – є дуже легкою в засвоєні, а написання програм на ній займає не великий час. Сучасні бібліотеки Python, критичні секції яких були написані мовою програмування C, використовують швидкість мови C, та легкість взаємодії з ними, як у Python, завдяки спеціалізованому Python API. Що дозволило отримати високу швидкодію та легкість в розробці.

Проаналізувавши уже готові рішення та інструменти які використовуються для скрапінгу новин було виявлено наступні проблеми, під час парсингу новинних повідомлень:

- потрібно писати окремий парсер під кожен вебсайт;
- потрібно підтримувати велику кількість веб-парсерів при збільшені кількості вебсайтів;
- потрібно змінювати веб-парсер, при зміні дизайну вебсайту, що є затратним зі сторони людських ресурсів;
- досить велика кількість ресурсів потрібна для великої кількості парсерів;
- існує проблематика з парсингом часу публікації, із-за різного формату дати на кожному вебсайті.

Реалізована система вирішує більшість вказаних вище проблем, але застосовується лише для парсингу новинних повідомлень, тому його не можна застосовувати, наприклад щоб парсити інтернет магазини, форуми, тощо. Завдяки створенню шаблону для кожного вебсайту нам не потрібно

підтримувати велику кількість парсерів. Шаблон завжди швидко можна перебудувати при зміні вебсайту, або створити вручну, що додає більшої гнучкості системі.

Використання алгоритму ROBULA+ дозволило будувати надійні XPath локатори, що від зміни HTML вебсайту – не зламаються, лише в деяких випадках потрібно буде змінювати локатор. Алгоритм розпізнавання посилання на новинні повідомлення допомагає з фільтрацією посилань, які не стосуються новин, що в свою чергу підвищує швидкодію системи, тому що не потрібно обходити всі посилання.

Як видно з результатів тестування шаблон не завжди створюється вірно, тому іноді його потрібно вручну створювати, але це набагато легше ніж створювати окремий парсер. Точність вірно створеного шаблону складає ~89%. Що є досить високим показником.

Розподілена система API і парсера, дозволяє досить швидко та ефективно масштабуватися, навіть на різних комп’ютерах, головне щоб вони були підключені до глобальної мережі Internet. І додає більшої стійкості, та дозволяє розподіляти ресурси.

Зібрані новини, мають дуже велику цінність та використовуються в багатьох сферах:

- штучного інтелекту, для навчання штучних нейронних мереж;
- пошуку, виділення ключових слів з новини, тощо;
- новинних стрічках, будують спеціалізовані новинні стрічки які відповідають темам, що цікавлять користувачів;
- відслідковування фейкових новин та джерел які їх публікують;

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Вебскрапінг [Електронний ресурс]. – 2020 – Режим доступу: https://scrapinghub.com/what-is-web-scraping
2. ЗАКОН УКРАЇНИ Про авторське право і суміжні права [Електронний ресурс]. – 2020 – Режим доступу: https://zakon.rada.gov.ua/laws/show/3792-12
3. Robots.txt [Електронний ресурс]. – 2020 – Режим доступу: https://support.google.com/webmasters/answer/6062608
4. DDoS-атака [Електронний ресурс]. – 2020 – Режим доступу: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/
5. Scrapy [Електронний ресурс]. – 2020 – Режим доступу: https://docs.scrapy.org/
6. Selenium [Електронний ресурс]. – 2020 – Режим доступу: https://www.selenium.dev/documentation/en/
7. Newspaper [Електронний ресурс]. – 2020 – Режим доступу: https://github.com/codelucas/newspaper
8. Python [Електронний ресурс]. – 2020 – Режим доступу: https://www.python.org/
9. Beautiful Soup Documentation [Електронний ресурс]. – 2020 – Режим доступу: https://www.crummy.com/software/BeautifulSoup/bs4/doc/
10. lxml - XML and HTML with Python [Електронний ресурс]. – 2020 – Режим доступу: https://lxml.de/
11. Pandas documentation [Електронний ресурс]. – 2020 – Режим доступу: https://pandas.pydata.org/docs/
12. Asyncio [Електронний ресурс]. – 2020 – Режим доступу: https://docs.python.org/3/library/asyncio.html
13. FastAPI [Електронний ресурс]. – 2020 – Режим доступу: https://fastapi.tiangolo.com/

14. REST [Електронний ресурс]. – 2020. – Режим доступу: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest
15. Maurizio Leotta, Andrea Stocco, Filippo Ricca, Paolo Tonella. ROBULA+: An Algorithm for Generating Robust XPath Locators for Web Testing. Journal of Software: Evolution and Process, Volume 28, Issue 3, pp.177–204. John Wiley & Sons, 2016.
16. Microdata [Електронний ресурс]. – 2020. – Режим доступу: https://schema.org/docs/gs.html
17. RDFa [Електронний ресурс]. – 2020. – Режим доступу: https://www.w3.org/TR/rdfa-primer/
18. JSON-LD [Електронний ресурс]. – 2020. – Режим доступу: https://json-ld.org/
19. Regular-Expressions [Електронний ресурс]. – 2020. – Режим доступу: https://www.regular-expressions.info/
20. HTML Block and Inline Elements [Електронний ресурс]. – 2020. – Режим доступу: https://www.w3schools.com/html/html_blocks.asp
21. Really Simple Syndication (RSS) [Електронний ресурс]. – 2020. – Режим доступу: https://www.w3.org/wiki/RSS