Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: Розробка системи збору та аналізу новин із вебджерел

Виконав: студент 4 курсу
групи 2ПІ-21Б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Бондар В.О.
(прізвище та ініціали)

Керівник: д.т.н., проф. каф. ПЗ, Ліщинська Л.Б.
(прізвище та ініціали)

Рецензент: к.т.н., доцент, каф. ОТ, Черняк О.І.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ О.Н. Романюк
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
« 21 » березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Бондар Владиславу Олександровичу

1. Тема роботи – розробка системи збору та аналізу новин із вебджерел. Керівник роботи: Ліщинська Людмила Броніславівна, д.т.н., професор, затверджені наказом вищого навчального закладу від 20 березня 2025 р. № 97.

2. Строк подання студентом роботи 2 червня 2025

3. Вихідні дані до роботи: базові методи зафарбовування – новинні стрічки у форматі RSS або JSON, отримані з відкритих джерел; інформація про категорії новин; параметри фільтрації новин; структура та формат даних для зберігання новин у локальній базі; користувацькі дії; вихідні дані – структурований список новин, відфільтрований та представлений через інтерфейс вебзастосунку.

4. Зміст текстової частини: аналіз сучасних засобів збору новин та їх функціональних можливостей; визначення вимог до вебзастосунку для збору й обробки новин; обґрунтування вибору технологій розробки; архітектура вебзастосунку та принципи взаємодії між модулями; розробка алгоритму збору новин із RSS-джерел; реалізація фільтрації новин за категоріями та ключовими словами; розробка структури локальної бази даних для збереження новин; проектування інтерфейсу користувача з урахуванням UX/UI-принципів; тестування програмного забезпечення на відповідність функціональним вимогам;

5. Перелік ілюстративного матеріалу: приклади сучасних вебзастосунків для моніторингу новин; діаграми системної архітектури; блок-схема алгоритму роботи застосунку; макети інтерфейсу користувача; скріншоти роботи програмного модуля; результати тестування.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	д.т.н., професор, Л.Б. Ліщинська	24.03.25	01.06.25

7. Дата видачі завдання. 24 березня 2025 р

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області та формування вимог до системи	25.03.25 – 05.04.25	вик
2	Розробка архітектури та вибір інструментів реалізації	06.04.25 – 15.04.25	вик
3	Реалізація модуля збору новин з веб-джерел	16.04.25 – 23.04.25	вик
4	Розробка модуля фільтрації, категоризації та інтерфейсу перегляду	24.04.25 – 10.05.25	вик
5	Тестування програми	10.05.25 – 20.05.25	вик
6	Оформлення матеріалів до захисту БКР	20.05.25 - 30.05.25	вик

Студент _____ (підпис) **В.О. Бонда** (ініціали та прізви

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) **Л.Б. Ліщинськ** (ініціали та прізви

Анотація

УДК 004.912.032.26

Бондар В.О. Розробка системи збору та аналізу новин із вебджерел : бакалаврська кваліфікаційна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 87 с.

На укр. мові. Бібліогр. : 22 назв ; рис. : 26; табл. 4.

У бакалаврській кваліфікаційній роботі розроблено програмні модулі вебсистеми збору та аналізу новин із використанням технології RSS. Основною метою є створення інструменту, який дозволяє користувачам додавати власні джерела новин, автоматично отримувати та фільтрувати актуальні повідомлення за заданими параметрами (категорії, ключові слова, дати тощо). Робота включає реалізацію зручного інтерфейсу для взаємодії користувача з системою, а також модулів для обробки RSS-стрічок і виведення структурованого списку новин.

Розробка системи здійснювалася з використанням сучасного JavaScript-фреймворку Vue.js для створення адаптивного та динамічного інтерфейсу. Програмні рішення, розроблені в межах роботи, можуть бути використані для моніторингу новин у різних тематиках, створення персоналізованих стрічок новин або аналітичних інструментів для ЗМІ та дослідницьких центрів.

Ключові слова: JavaScript, Vue, RSS, веб-застосунок, збір даних, аналіз інформації.

‘

Annotation

UDC 004.912.032.26

Bondar V.O. Development of a system for collecting and analyzing news from web sources: bachelor's qualification work in the specialty 121 Software Engineering, educational program - Software Engineering. Vinnytsia: VNTU, 2025. 87 p.

In Ukrainian language. Bibliography: 22 titles; fig. : 26; tab. 4.

This project develops software modules for a web-based news collection and analysis system using RSS technology. The main goal is to create a tool that allows users to add their own news sources, automatically receive and filter current messages by specified parameters (categories, keywords, dates, etc.). The project includes the implementation of a convenient interface for user interaction with the system, as well as modules for processing RSS feeds and displaying a structured list of news.

The system was developed using the modern JavaScript framework Vue.js to create an adaptive and dynamic interface. Software solutions developed within the project can be used to monitor news in various topics, create personalized news feeds or analytical tools for media and research centers.

Keywords: JavaScript, Vue, RSS, web application, data collection, information analysis.

‘

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У сучасному інформаційному суспільстві кількість цифрових новинних ресурсів стрімко зростає. Щоденно в мережі Інтернет публікується величезний обсяг новинного контенту, який містить важливу соціальну, економічну, політичну та культурну інформацію. Ці дані становлять цінність для журналістів, аналітиків, маркетологів, державних структур та пересічних користувачів, однак їх обробка вручну є надзвичайно складною та ресурсомісткою.

У зв’язку з цим актуальним є створення програмних засобів, що автоматизують процес збору, структурування, аналізу та візуалізації новин з різних веб-джерел. Такий підхід дозволяє виявляти тенденції, агрегувати новини за темами, визначати інформаційні сплески, виявляти дезінформацію або формування штучного інформаційного тла.

Сучасні інформаційні системи повинні володіти можливостями ефективного збору структурованих та неструктурованих даних із різнорідних джерел, їх подальшого оброблення, класифікації та надання у зручному для користувача вигляді. Особливої актуальності набувають системи, здатні працювати в реальному часі, адаптуватися до змін у структурі сайтів і підтримувати багатомовну обробку тексту.

У рамках цього дослідження розглядається побудова автоматизованої системи, яка дає змогу збирати новини з веб-джерел (зокрема, через RSS-канали) [1], здійснювати їх первинний аналіз та надавати можливості для фільтрації й категоризації контенту. Така система може знайти застосування у сфері журналістики, інформаційної аналітики, безпеки, моніторингу репутації тощо.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення

Мета та завдання дослідження. Метою бакалаврської кваліфікаційної роботи є спрощення отримання актуальної інформації та управління нею,

шляхом розробки вебзастосунку для автоматичного збору, фільтрації та візуалізації новин із різних джерел.

Для досягнення мети потрібно виконати низку завдань:

– визначити вимоги до функціональності вебсистеми;

– розробити алгоритми збору та обробки RSS-новин;

– реалізувати інтерфейс для додавання та редагування новинних джерел;

– забезпечити можливість фільтрації новин за ключовими параметрами;

– протестувати систему на надійність та зручність використання.

Об'єктом дослідження є процес отримання та управління інформацією для обробки інформаційного контенту з відкритих новинних ресурсів.

Предметом дослідження є методи і алгоритми збору, обробки та фільтрації новинних даних.

Методи дослідження. У процесі виконання роботи використовувалися: методи аналізу та синтезу програмного забезпечення для визначення функціональних та нефункціональних вимог до системи, методи інженерії вимог для збору, формалізації та уточнення потреб користувачів, методи розробки веб-додатків для створення клієнтської та серверної частин системи збору та аналізу новин, інструменти парсингу для автоматизованого збору інформації з вебджерел через RSS-стрічки, основи аналізу текстів та фільтрації даних за ключовими словами для попередньої обробки та класифікації зібраних новин

Новизна отриманих результатів.

1. Запропоновано комплексний підхід до збору новин, який об'єднує можливості роботи з RSS та прямим HTML-парсингом, що дозволило розширити кількість підтримуваних джерел, забезпечити гнучкість системи при роботі з різними форматами вебресурсів та підвищити повноту отримуваної інформації. Такий підхід також мінімізує залежність від структури конкретного сайту та дозволяє адаптувати систему до змін на стороні джерел новин

2. Реалізовано механізм категоризації новин за заданими критеріями, за рахунок чого забезпечується автоматичне визначення тематики новинних повідомлень на основі ключових слів, тегів або метаданих джерел, що дозволяє

автоматизувати тематичне сортування, підвищити релевантність відображуваного контенту та покращити загальну зручність користування системою.

Практична цінність отриманих результатів полягає в тому, що в результаті роботи було розроблено програмний засіб для автоматизованого збору та аналізу новин з веб-джерел. Система дозволяє здійснювати парсинг новинних сайтів, агрегувати отриману інформацію, проводити її попередню обробку та фільтрацію за заданими критеріями. Розроблений програмний продукт може бути використаний у сфері журналістики, маркетингових досліджень, моніторингу інформаційного простору, а також у навчальному процесі при вивченні сучасних інформаційних технологій та обробки даних.

Особистий внесок здобувача. Усі етапи дослідження, аналізу, проєктування архітектури, програмування окремих модулів та інтеграції системи виконані автором самостійно.

Апробація результатів роботи. Описані у дослідженні положення доповідались на конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)».

Публікації. Основні результати досліджень опубліковано в матеріалах конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)». [2].

1 АНАЛІЗ РОЗВИТКУ ПРОГРАМНИХ МОДУЛІВ ВЕБСИСТЕМИ ЗБОРУ ТА АНАЛІЗУ НОВИН

1.1 Аналіз стану програмних модулів вебсистеми збору та аналізу новин

На сучасному етапі розвитку інформаційних технологій особливої актуальності набувають системи, що автоматизують процес збору, обробки та аналізу великого обсягу новинного контенту. Постійне зростання кількості новинних ресурсів та швидкість оновлення інформації вимагають ефективних інструментів для фільтрації, класифікації та персоналізації даних. Саме тому вебсистеми збору новин, які використовують модульний підхід до реалізації функціоналу, стають усе більш поширеними.

Традиційні новинні агрегатори часто мають обмежений функціонал: вони дозволяють лише перегляд RSS-стрічок без можливості їх глибокого аналізу, категоризації або індивідуального налаштування. Більш сучасні рішення включають модулі для тематичної класифікації новин, виявлення ключових слів, а також базові засоби фільтрації за датою або джерелом. Проте більшість таких систем є або закритими, або платними, що обмежує їх використання в освітніх чи дослідницьких проєктах.

Розвиток обчислювальних технологій, вебфреймворків та бібліотек (таких як Vue.js [3], Node.js [4], Express [5], Axios [6] тощо) відкриває нові можливості для створення ефективних вебрішень, які можуть поєднувати різні програмні модулі в межах однієї системи. Це дозволяє не лише збирати новинні дані з численних джерел, але й забезпечувати їхню інтерактивну обробку та зручну візуалізацію для кінцевого користувача.

Ключовими модулями сучасних систем збору новин є:

– модуль парсингу RSS-стрічок;

– модуль збереження та обробки даних;

– модуль фільтрації та категоризації;

– графічний інтерфейс користувача.

Зокрема, можливість додавання власних джерел новин користувачем, персоналізована фільтрація за темами або ключовими словами, а також доступ до історії переглядів – це ті функції, що сьогодні очікуються навіть від простих новинних сервісів. У відкритому доступі існує небагато проєктів, що поєднують всі ці функції, залишаючись водночас безкоштовними та придатними для адаптації під конкретні завдання.

Отже, аналіз сучасного стану програмних модулів вебсистем збору новин свідчить про необхідність створення відкритих, масштабованих і зручних рішень, що дозволяють ефективно обробляти динамічні новинні потоки. Модульний підхід до розробки таких систем забезпечує гнучкість, можливість поступового розширення функціоналу, а також кращу адаптацію під потреби різних груп користувачів – від звичайних читачів до дослідників інформаційного середовища.

1.2 Порівняльний аналіз аналогів

У сучасному інформаційному середовищі існує велика кількість вебсистем, які здійснюють автоматизований або напівавтоматизований збір, аналіз та візуалізацію новин з різноманітних джерел, таких як інформаційні портали, блоги, соціальні мережі, RSS-канали тощо. Вони виконують важливу функцію – оперативне інформування користувачів про події, що відбуваються у світі, у таких сферах як політика, економіка, технології, наука, культура та інші. Такі системи часто використовують алгоритми машинного навчання, природної обробки мови та рекомендаційні моделі для персоналізації стрічки новин.

Однак попри їхню популярність та широкі можливості, більшість із них є закритими комерційними платформами, які не дозволяють користувачам налаштовувати джерела інформації або структуру збору даних під свої потреби.

Розглянемо кілька найбільш відомих аналогів та порівняємо їх функціональні можливості з розробленим програмним модулем. Є такі аналоги вебсистем збору та аналізу новин:

– Google News – це популярний новинний агрегатор, який автоматично збирає заголовки та короткі описи статей із понад 50 000 різноманітних джерел новин з усього світу [7]. Платформа аналізує нові публікації у режимі реального часу та групує їх відповідно до актуальних тем, подій, а також географічного розташування користувача. Завдяки цьому система може формувати персоналізовану стрічку новин, що відповідає інтересам конкретного читача.

Інтерфейс платформи є зручним та інтуїтивно зрозумілим, що забезпечує швидкий доступ до найважливіших новин, аналітики та репортажів. Приклад роботи системи та вигляд персоналізованої стрічки новин наведено на рисунку 1.1.

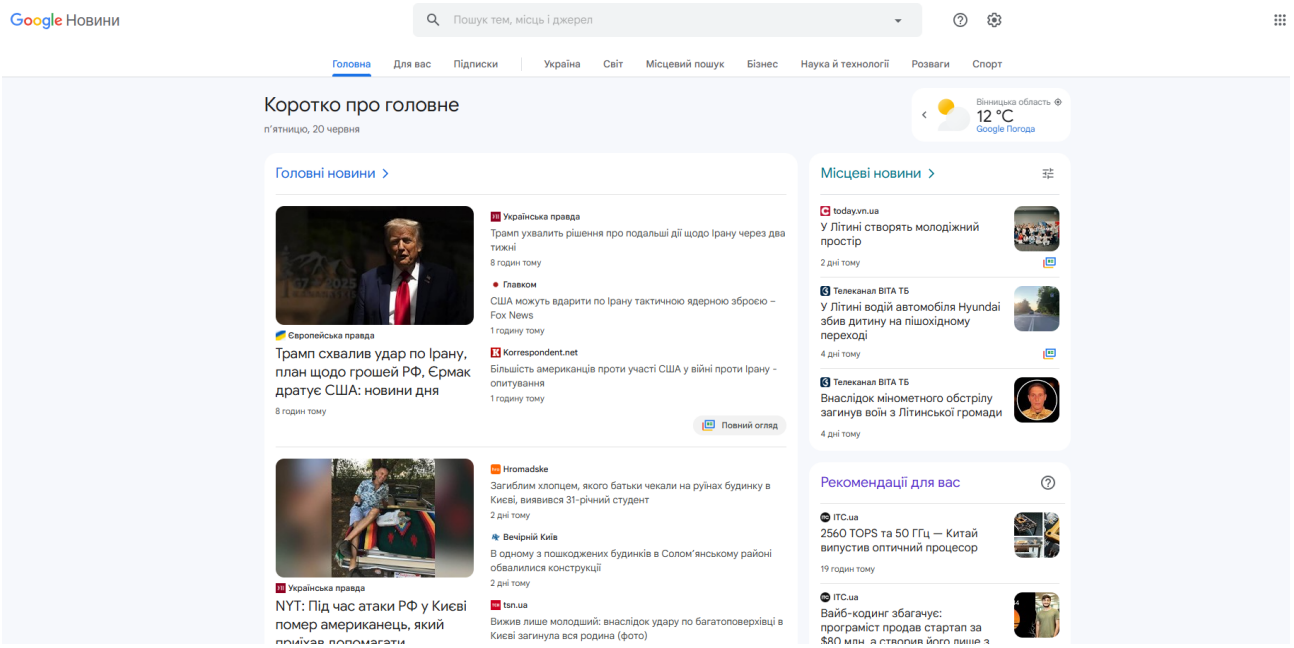

Рисунок 1.1 – Google News

– Feedly – це популярний RSS-агрегатор, який надає користувачам можливість підписуватися на улюблені новинні ресурси, блоги, тематичні сайти та інші інформаційні стрічки [8]. Платформа дозволяє об'єднувати новини з

різних джерел в одну зручну стрічку для подальшого читання, що значно спрощує процес моніторингу великої кількості інформаційних ресурсів.

Крім того, Feedly підтримує синхронізацію між різними пристроями, можливість створювати власні інформаційні колекції, ділитися матеріалами з іншими користувачами та інтегруватися з різними сервісами, такими як Evernote, OneNote, Slack тощо. Усе це робить Feedly ефективним інструментом для професіоналів у галузі маркетингу, журналістики, IT та дослідницької діяльності.

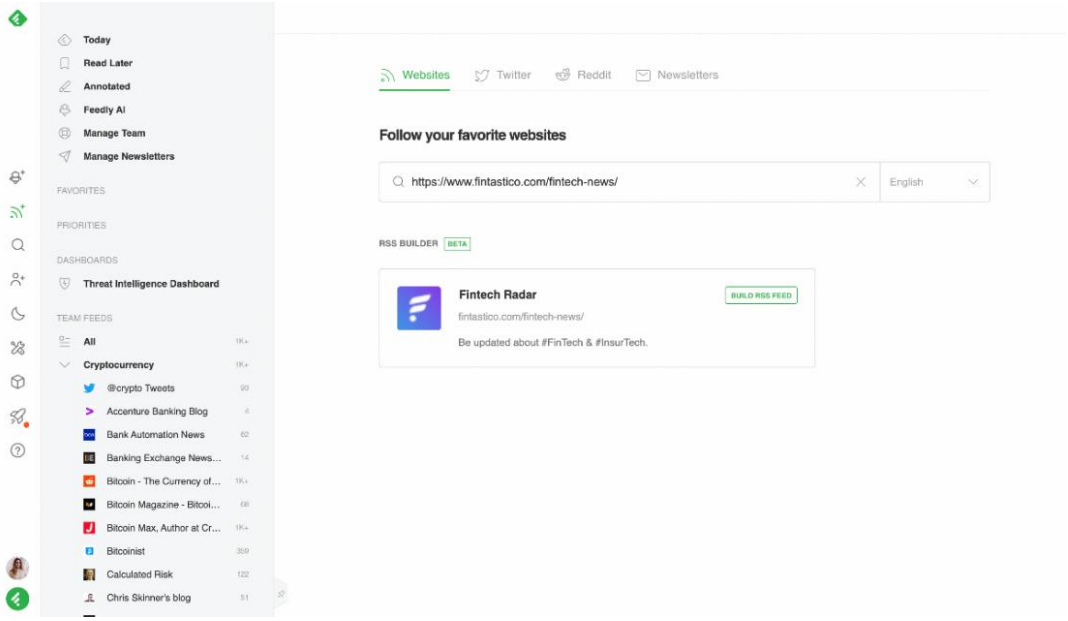

Рисунок 1.2 – Feedly

Flipboard – це інтерактивна платформа для агрегування новин, яка перетворює інформаційні стрічки, статті з новинних порталів та публікації із соціальних медіа на зручний візуальний журнал [9]. Особливістю сервісу є його унікальний інтерфейс, що імітує перегортання сторінок традиційного журналу, що робить читання новин більш привабливим та зручним для користувача.

Користувачі можуть налаштовувати власні тематичні добірки, створюючи персональні "журнали" з улюблених джерел або окремих статей. Такі журнали можуть охоплювати різні сфери – від політики та науки до культури, технологій,

моди чи спорту. Крім того, Flipboard дозволяє підписуватись на добірки інших користувачів або офіційні канали нових видань.

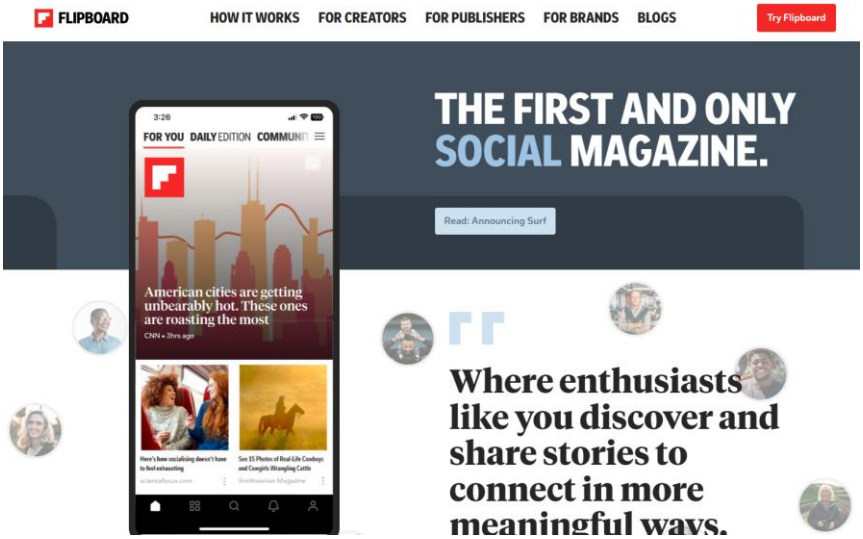

Рисунок 1.3 – Flipboard

– Particle – це інноваційний мобільний додаток, що використовує передові технології штучного інтелекту для автоматизованого збору, організації та узагальнення новинної інформації [10]. Головна особливість цієї платформи полягає у здатності системи не лише збирати новини з численних джерел, а й аналізувати їх зміст для подальшої класифікації за тематичними групами.

Алгоритми штучного інтелекту автоматично визначають основні події та сюжети, формують єдині тематичні блоки зі схожих за змістом статей і створюють короткі, змістовні резюме для кожної новинної теми. Завдяки цьому користувачі можуть за лічені хвилини ознайомитися з головними новинами дня без необхідності переглядати десятки окремих матеріалів.

Додаток також надає можливість налаштування персоналізованих стрічок новин, що враховують інтереси та переваги конкретного користувача. Система пропонує матеріали з урахуванням його попередніх уподобань, активності в додатку та актуальних світових подій.

Крім того, Particle підтримує функцію візуалізації інформації у вигляді графіків, хронологічних стрічок та інфографіки, що допомагає краще зрозуміти взаємозв’язок між різними новинними подіями та тенденціями.

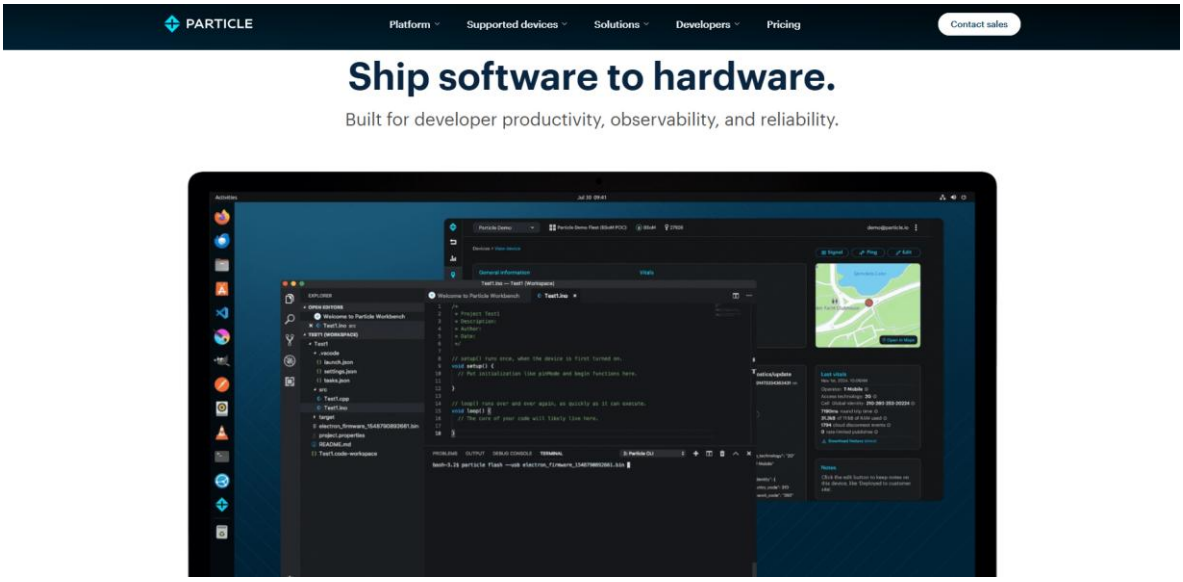

Рисунок 1.4 – Particle

SilkData News Analyzer – це аналітична платформа для збору, обробки та порівняння новинних матеріалів з різних джерел, яка дозволяє виявляти як спільні риси, так і унікальні аспекти кожної публікації [11]. Основною метою цієї системи є автоматизований аналіз великого обсягу текстової інформації для формування стислого та інформативного викладу новинної події.

Платформа застосовує сучасні алгоритми обробки природної мови (NLP) та методи добування даних для автоматичного визначення ключових слів, основних понять, фактів і висновків, що містяться у статтях. Завдяки цьому SilkData News Analyzer може скорочувати первинний текст до компактних резюме, що містять тільки найважливішу інформацію, без втрати змісту.

Ще однією важливою функцією системи є можливість порівняння публікацій з різних інформаційних джерел для виявлення розбіжностей у викладі фактів або акцентів на певних аспектах події. Це дозволяє користувачам отримати більш об'єктивну картину подій та уникнути інформаційних маніпуляцій або однобокого висвітлення теми.

Крім того, SilkData News Analyzer надає інструменти для візуалізації результатів аналізу у вигляді графіків або таблиць, що спрощує сприйняття великих масивів даних.

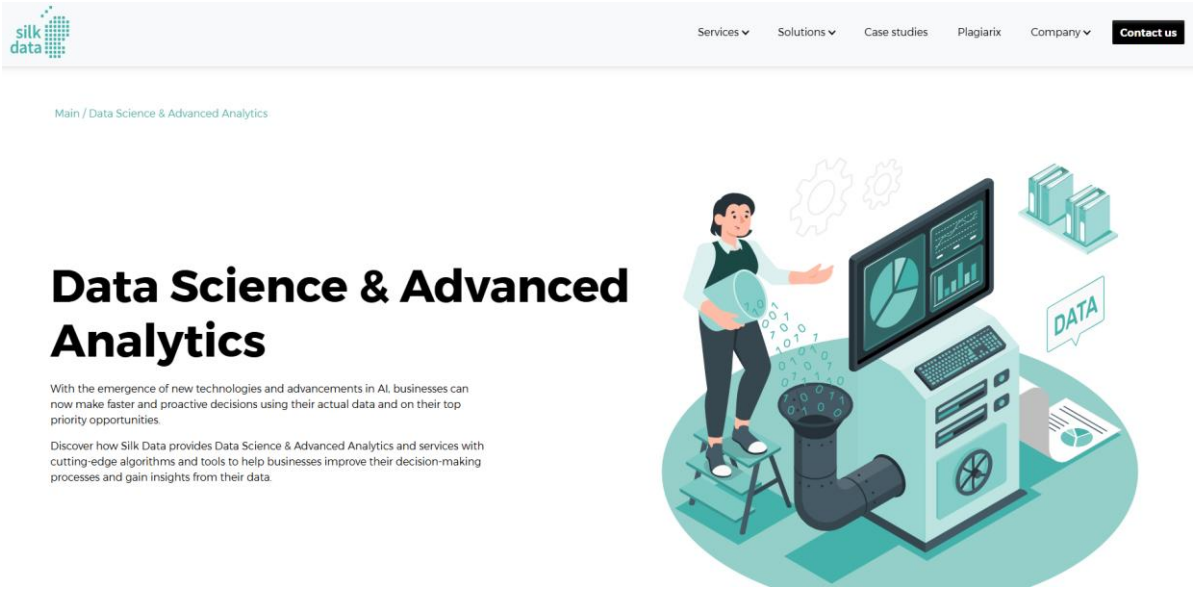

Рисунок 1.5 – SilkData News Analyzer

Результати порівняння аналогів зведено в табл. 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерій	Google News	Feedly	Flipboard	Particle	SilkData News Analyzer	Власний застосунок
Візуалізація даних	-	-	+	+	+	+
Порівняння новин з різних джерел	-	-	-	+	+	+
Підтримка української мови	+	+	+	-	-	+

Продовження таблиці 1.1

Критерій	Google News	Feedly	Flipboard	Particle	SilkData News Analyzer	Власний застосунок
Швидкість роботи	+	-	+	+	-	+
Підтримка RSS-джерел	-	+	-	-	-	+

З аналізу видно, що власний модуль:

– має ширші можливості налаштування новинної стрічки;

– дозволяє користувачам додавати власні RSS-джерела;

– підтримує автоматичну та ручну фільтрацію новин за категоріями;

– візуалізує статистику для аналітичного огляду;

– орієнтований на український інформаційний простір;

– використовує Vue для оптимізації продуктивності.

Ці переваги роблять систему більш гнучкою, локалізованою та адаптованою до сучасних вимог користувачів у порівнянні з існуючими аналогами.

1.3 Аналіз методів розв’язання задачі

Розробка програмного модуля збору та аналізу новин вимагає комплексного підходу до вирішення завдань, пов’язаних з агрегуванням новинного контенту, його фільтрацією, категоризацією, аналізом змісту та зручним представленням інформації користувачу. Для цього існує кілька методів, які застосовуються у подібних системах.

1. Метод парсингу RSS-стрічок

Найбільш поширеним підходом до збору новин є використання RSS-протоколу. Цей метод передбачає парсинг XML-даних [12] зі стрічок новин

сайтів, що дозволяє ефективно отримувати новини у структурованому вигляді. До переваг цього методу належить:

– висока швидкість обробки;

– мінімальне навантаження на сервери джерел;

– простота реалізації.

Недоліками є:

– обмежений обсяг інформації (іноді заголовок та короткий опис);

– відсутність RSS на деяких сучасних ресурсах;

– складність уніфікації даних із різних форматів стрічок.

2. Використання веб-скрейпінгу.

Альтернативним методом є веб-скрейпінг, який передбачає аналіз HTML-структури вебсайтів та виділення потрібних блоків [13] (текст новини, зображення, дата тощо). Цей метод дозволяє збирати новини навіть з сайтів, що не мають RSS, і отримувати повний текст статей. Проте:

– такі рішення менш стабільні (зміна верстки сайту ламає логіку);

– потребують складнішої підтримки та обробки винятків;

– можуть створювати навантаження на сайти та порушувати правила використання.

3. Класифікація новин за допомогою алгоритмів NLP [14].

Для автоматичної категоризації новинних статей використовуються методи обробки природної мови (NLP), такі як:

– TF-IDF [15] + KMeans [16] (простий та швидкий метод кластеризації новин);

– Naive Bayes [17] / SVM [18] (класичні ML-алгоритми для класифікації тексту).

Застосування NLP дозволяє:

– автоматично присвоювати категорії;

– виявляти тональність тексту (позитивна, нейтральна, негативна);

– визначати ключові слова та тематику.

Недоліки:

– потреба у великій навчальній вибірці;

– залежність якості результату від мови, стилю, довжини тексту;

– складність у реалізації без використання готових бібліотек.

4. Візуалізація та фільтрація

Для зручності користувача важливим є інтерфейс для фільтрації, сортування та перегляду новин. Можливі підходи:

– звичайне сортування за датою, джерелом, категорією;

– візуалізація статистики: графіки кількості новин по категоріях, динаміка за часом;

– відображення трендів (частотність тем у певний період).

Такі елементи підвищують інформативність системи та дозволяють користувачу швидко орієнтуватися в інформаційному потоці.

Для розробки власного модуля було вирішено поєднати кілька методів:

– парсинг RSS-стрічок, з можливістю додавання джерел користувачем;

– аналіз тексту за допомогою NLP, з реалізацією автоматичної категоризації;

– гнучка система фільтрів та візуалізація даних;

– реалізація на Vue.js для швидкої та адаптивної роботи інтерфейсу.

Такий підхід дозволяє створити сучасний, точний та гнучкий інструмент для роботи з новинами, який перевершує більшість існуючих аналогів у можливостях кастомізації, локалізації та ефективного аналізу інформації.

1.4 Постановка задачі дослідження

Після детального аналізу існуючих систем збору та аналізу новинних даних із веб-джерел, їх функціональних можливостей, переваг та недоліків, а також визначення доцільності створення власного програмного засобу, було

сформовано перелік основних завдань, необхідних для реалізації системи, яка відповідатиме сучасним вимогам до автоматизації збору, обробки та аналітики інформації.

Насамперед, необхідно провести дослідження існуючих методів збору даних з відкритих веб-джерел, таких як парсинг HTML-сторінок, використання RSS-каналів, API новинних сервісів. Це дозволить обґрунтувати вибір оптимального інструментарію для побудови системи.

Наступним етапом є розробка механізмів фільтрації та попередньої обробки новинної інформації для виділення релевантних даних відповідно до заданих критеріїв (ключові слова, тематичні категорії, часові рамки тощо). Для цього планується застосувати регулярні вирази, алгоритми обробки текстів природної мови (NLP) та інші методи інтелектуальної обробки даних.

Важливою складовою системи є модуль аналізу зібраної інформації, який дозволить групувати новини за тематикою, визначати найбільш популярні теми, джерела та динаміку інформаційних потоків. Передбачається використання методів кластеризації, тематичного моделювання та базових статистичних методів аналізу даних.

Програмна реалізація проєкту буде здійснюватися з використанням мови програмування JavaScript із застосуванням фреймворку Vue.js для створення клієнтської частини та Node.js для реалізації серверної логіки. Це забезпечить кросплатформеність, гнучкість та розширюваність системи.

Особлива увага приділятиметься тестуванню системи на різних етапах розробки для перевірки її працездатності, стійкості до збоїв, швидкодії та відповідності вимогам користувача. Буде проведено як модульне, так і інтеграційне тестування всіх компонентів системи.

Реалізація зазначених завдань дозволить створити ефективний інструмент для автоматизованого збору та аналізу новин із веб-джерел, який може бути використаний у журналістиці, PR, бізнес-аналітиці, моніторингу інформаційних ризиків, а також у наукових дослідженнях інформаційних потоків. Крім того, передбачено можливість подальшого розвитку системи, зокрема додавання

модулів прогнозування інформаційних трендів, аналізу тональності новинних матеріалів та ін.

1.5 Висновки

У першому розділі було розглянуто сучасний стан програмних засобів для збору та аналізу новин. Були проаналізовані основні рішення, такі як програми для парсингу новин через RSS-стрічки, а також можливості для фільтрації та категоризації новин. Кожен з розглянутих підходів має свої переваги і недоліки, які були ретельно проаналізовані.

На основі цього аналізу було сформульовано завдання для подальшої розробки власного програмного забезпечення для збору та аналізу новин. Це включає створення алгоритмів для автоматичного парсингу новин з різних джерел, систему фільтрації новин за категоріями та ключовими словами, а також розробку графічного інтерфейсу користувача, який забезпечить зручність та ефективність використання.

Таким чином, було доведено доцільність розробки власного програмного рішення, яке дозволить подолати недоліки існуючих рішень і надасть користувачам потужніші та більш гнучкі інструменти для збору та аналізу новин. На основі отриманої інформації були сформовані задачі, необхідні для реалізації цього програмного продукт.

2. РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ РЕАЛІЗАЦІЇ ДЛЯ ЗБОРУ ТА АНАЛІЗУ ІНФОРМАЦІЇ

2.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Під час розробки вебзастосунку для збору та аналізу новин важливим етапом стало обґрунтування вибору технологій, які дозволяють забезпечити зручність у користуванні, масштабованість, гнучкість інтерфейсу та ефективність обробки даних. Для реалізації програмного забезпечення було обрано стек технологій, що включає Vue.js, Node.js, Express та RSS.

Vue.js – це сучасний JavaScript-фреймворк для створення інтерфейсів користувача. Його головною перевагою є реактивність, тобто автоматичне оновлення частин інтерфейсу у відповідь на зміну даних. Завдяки простому синтаксису та широким можливостям компонентної архітектури, Vue.js ідеально підходить для розробки зручного та динамічного фронтенду новинного вебзастосунку. Він дозволяє реалізувати інтерфейс, що містить функціонал перегляду новин, їх фільтрацію, пошук, а також інтерактивну взаємодію з користувачем у реальному часі.

Node.js – це середовище виконання JavaScript на стороні сервера, яке дозволяє створювати масштабовані серверні додатки. У контексті розробки новинного сервісу Node.js забезпечує ефективне оброблення великої кількості запитів, а також реалізацію логіки додавання джерел, збору RSS-стрічок та обробки запитів користувачів до бази даних.

Express – це мінімалістичний вебфреймворк для Node.js, що забезпечує просту маршрутизацію запитів, обробку REST API [19] та взаємодію між клієнтською та серверною частиною застосунку. Використання Express дає змогу швидко реалізувати API-інтерфейс для обробки новин, джерел та параметрів фільтрації.

RSS (Really Simple Syndication) – це стандартний формат, який використовується для автоматичного збору новин із зовнішніх джерел. Завдяки використанню RSS, вебзастосунок має змогу інтегрувати новини з різних сайтів, зберігаючи актуальність інформації без ручного втручання користувача. Це є ключовим аспектом для реалізації основної функціональності застосунку – агрегування новин із численних джерел.

Вибір цих технологій був обґрунтований також підтримкою спільноти, кількістю готових бібліотек та модулів, швидкістю розробки, а також можливістю масштабування проєкту. Завдяки поєднанню Vue.js (на фронтенді), Node.js та Express (на бекенді) і RSS (для збору новин), вдалося створити цілісну, продуктивну та зручну систему для користувачів.

Таким чином, обраний стек технологій забезпечує повний цикл функціональності вебзастосунку: від збору новин та їх обробки до візуалізації на клієнтському інтерфейсі. Це дозволяє досягнути високого рівня інтерактивності, надійності й зручності у роботі як для кінцевого користувача, так і для розробників при подальшій підтримці й розширенні функціоналу системи.

2.2 Розробка модуля пошуку

Модуль "Пошук новин" є ключовим компонентом вебзастосунку для збору та аналізу новин, оскільки він забезпечує зручний інструмент для користувачів, які хочуть швидко знаходити новини за ключовими словами або фразами. Основна мета модуля полягає у швидкому та точному знаходженні релевантних новин серед великого обсягу інформації, що зберігається в базі даних.

Архітектура модуля.

Модуль реалізований у межах повноцінного клієнт-серверного підходу:

– клієнтська частина (Vue.js) відповідає за інтерфейс введення пошукового запиту та виведення результатів;

– серверна частина (Node.js + Express) обробляє запити від клієнта, виконує пошук у базі даних та надсилає результати у вигляді JSON [20].

Клієнтська частина.

Інтерфейс модуля побудований за допомогою Vue.js. Основні елементи включають:

– поле введення для текстового запиту;

– кнопка пошуку для ініціалізації запит;

– список результатів, який динамічно оновлюється відповідно до отриманих з сервера даних.

```
<!-- Центральна частина: пошук -->
<div class="center-panel">
  <h3>Пошук</h3>
  <input
      v-model="query"
      @keyup.enter="search"
      placeholder="Введіть запит..."
  />
  <button @click="search">Пошук</button>
</div>
```

Рисунок 2.1 – Код інтерфейсу пошуку новин

Для реалізації пошуку використовується двостороннє зв’язування (v-model), а також подія @click для запуску функції пошуку. Результати відображаються у вигляді компонентів-карток із короткою інформацією про кожну новину.

Серверна частина.

У модулі "Пошук новин" реалізовано логіку пошуку безпосередньо по RSS-стрічках джерел у реальному часі. Серверна частина цього модуля розроблена за допомогою Node.js та Express. Запити надходять за маршрутом /search методом GET.

Обробка запиту.

Клієнт надсилає параметри:

– q – текст пошукового запиту (ключові слова);

– sources – перелік джерел (назви), в яких потрібно здійснити пошук (опційно).

Запит q приводиться до нижнього регістру для реалізації нечутливого до регістру пошуку.

Завантаження джерел.

У системі зберігається окремий JSON-файл зі списком усіх доступних RSS-джерел. Він зчитується при кожному запиті.

Якщо користувач вказав конкретні джерела – відбувається фільтрація новин

Таким чином, забезпечується можливість точкового пошуку лише в обраних джерелах.

Пошук у стрічках.

Для кожного вибраного джерела програма:

1. Завантажує RSS-стрічку за допомогою парсера (наприклад, rss-parser).

2. Перебирає всі новини (елементи feed.items).

3. Порівнює заголовок (title) і опис (description) кожної новини з пошуковим запитом.

Якщо знайдено збіг, новина додається до списку результатів. Після завершення пошуку всі знайдені новини, що відповідають запиту, збираються у масив результатів. Кожен елемент містить такі властивості: заголовок новини, опис, посилання на повну версію статті, а також джерело, з якого ця новина була отримана.

Отриманий масив новин повертається клієнту у форматі JSON. Завдяки цьому на стороні користувача забезпечується динамічне оновлення вмісту сторінки без необхідності її перезавантаження.

Рисунок 2.2 – Блок схема роботи модуля пошуку

Обробка помилок.

У випадку недоступності або некоректності RSS-джерела, помилка не зупиняє виконання всього процесу – вона фіксується у консолі.

Повернення результату.

Після обробки всіх джерел сервер повертає масив об’єктів з результатами у форматі JSON.

Кожен об’єкт містить:

– title – заголовок новини;

– description – короткий опис;

– url – посилання на повну новину;

– source – назва джерела.

Таким чином, серверна частина забезпечує ефективний, гнучкий та стійкий до помилок механізм пошуку новин з можливістю фільтрації за джерелами та підтримкою необов'язкових параметрів пошукового запиту.

2.3 Розробка модуля додавання джерел

Модуль "Додавання джерел" дозволяє розширювати список інформаційних ресурсів, з яких буде здійснюватися збір новин. Це важлива функція, яка забезпечує гнучкість та масштабованість системи, дозволяючи користувачеві самостійно додавати нові RSS-стрічки без необхідності змінювати код або конфігурацію вручну.

Реалізація цього модуля здійснена на серверній частині застосунку за допомогою Node.js з використанням фреймворку Express. Додавання джерела виконується за маршрутом POST /sources.

Це робить систему більш динамічною та дозволяє кінцевим користувачам або адміністраторам збагачувати перелік джерел у відповідь на зміни в інформаційному просторі або потреби конкретного застосування.

Покроковий опис.

1. Обробка вхідних даних.

Сервер очікує, що у тілі запиту (req.body) буде передано два обов'язкових параметри: name (назва джерела) та url (посилання на RSS-стрічку). Якщо хоча б

один з параметрів відсутній або некоректний (наприклад, URL не є валідним посиланням), сервер повертає клієнту повідомлення про помилку з HTTP-статусом 400 та відповідним текстом.

2. Завантаження поточного списку джерел.

Використовується функція loadSources(), яка читає локальний JSON-файл (наприклад, sources.json), у якому збережено список усіх наявних джерел. Це дозволяє працювати з актуальним переліком без необхідності взаємодії з базою даних.

3. Додавання нового джерела.

На основі отриманих від клієнта параметрів формується новий об’єкт виду { name, url }, який додається до масиву джерел.

4. Збереження оновленого списку.

Оновлений масив джерел передається у функцію saveSources(sources), яка перезаписує JSON-файл, зберігаючи всі старі та нові елементи. Таким чином забезпечується постійна актуальність списку.

5. Відповідь клієнту.

У разі успішного додавання нового запису сервер повертає клієнту JSON-відповідь з повідомленням про успішне завершення операції (наприклад: { message: 'Джерело додано' }). У разі помилки відповідь містить інформацію про причину відмови.

Окрім базової перевірки наявності обов’язкових параметрів, модуль може бути розширений додатковими механізмами валідації. Наприклад, можлива перевірка коректності формату переданого URL, уникнення дублювання записів (перевірка, чи таке джерело вже існує у списку), а також автоматичне тестування доступності RSS-стрічки перед додаванням до загального переліку.

Таким чином, модуль "Додавання джерел" є не лише інструментом поповнення бази даних джерел, але й фундаментальною частиною загальної архітектури системи збору новин, яка забезпечує її адаптивність до змін у мережевому середовищі та дозволяє користувачеві самостійно контролювати якість і релевантність отримуваної інформації.

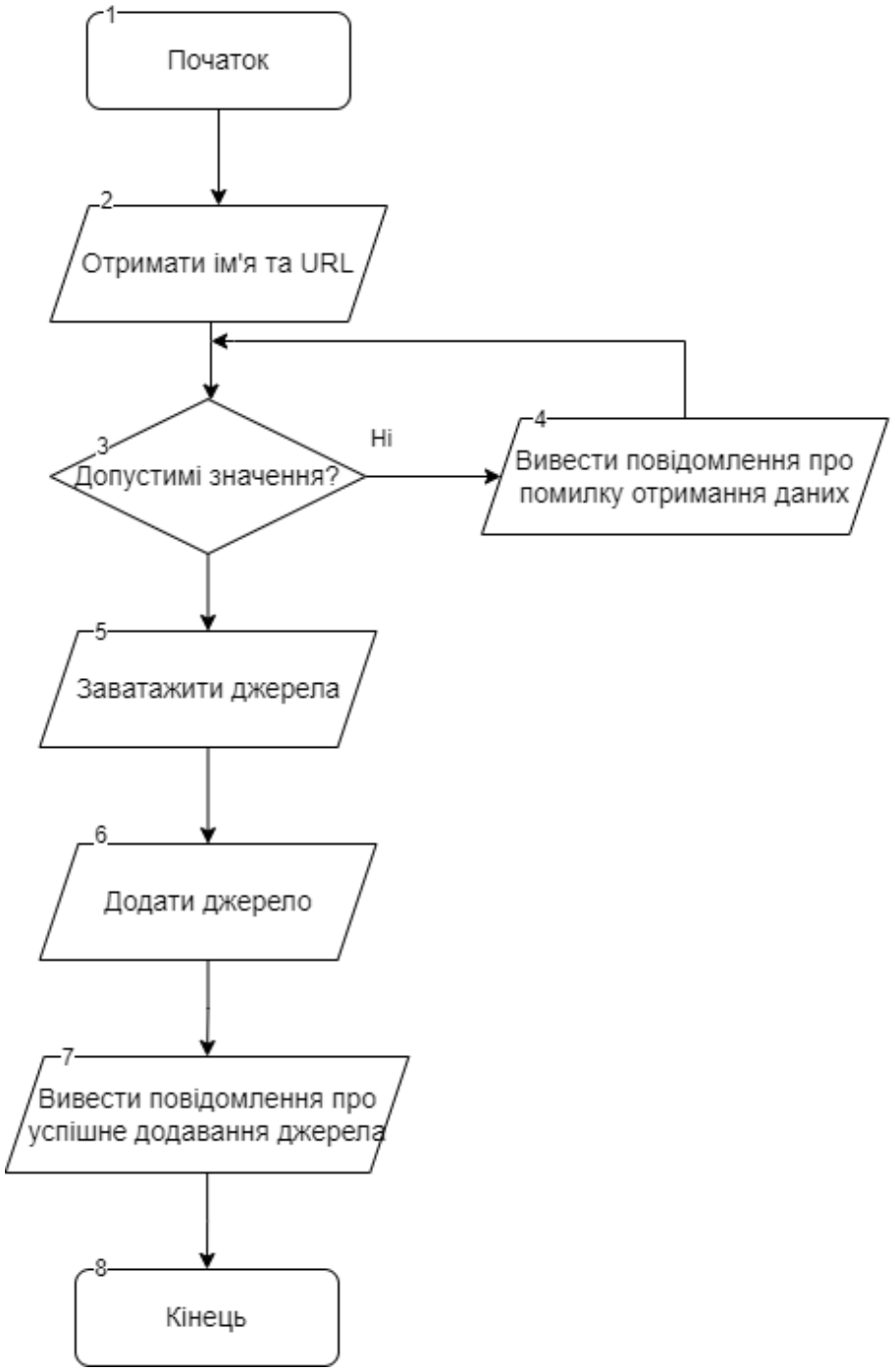

Рисунок 2.3 – Блок-схема модулю програми для додавання джерел

Додатково система може бути розширена механізмом перевірки наявності дублюючих записів, валідацією формату URL та автоматичною перевіркою доступності зазначеної RSS-стрічки для уникнення помилкових чи недійсних джерел. Це дозволить ще більше підвищити якість і надійність функціоналу додавання нових ресурсів.

2.4 Розробка модуля фільтрації новин

Модуль фільтрації новин є ключовим компонентом програмного застосунку для збору та аналізу новин, що дозволяє користувачам обмежити результати пошуку або перегляду новин лише тими джерелами, які їх цікавлять. Це значно покращує зручність користування системою, дозволяє фокусуватись на авторитетних чи тематичних джерелах та зменшує інформаційний шум.

Принцип роботи:

1. Формування списку доступних джерел.

У системі зберігається перелік усіх активних RSS-джерел, які були попередньо додані користувачем або адміністрацією через відповідний модуль додавання джерел. Цей список доступний на клієнтській частині через API.

2. Інтерфейс фільтрації.

У вебінтерфейсі користувач має змогу вибрати джерело з переліку за допомогою випадаючого списку. Це дозволяє сформувати запит лише до обраних джерел.

3. Надсилання запиту на сервер.

При пошуку новин або оновленні списку новин, користувацький запит надсилається до серверу з параметром sources, який містить обрані джерела (у вигляді масиву або рядка з назвами).

4. Фільтрація на сервері.

Серверна частина отримує список джерел та виконує фільтрацію повного списку, залишаючи лише ті, що відповідають зазначеним користувачем. Далі здійснюється аналіз RSS-стрічок лише для цих джерел, з витягуванням заголовків, описів, посилань тощо.

5. Повернення результатів.

У відповідь на запит користувач отримує оновлений список новин, що відповідають як пошуковому запиту, так і обраним джерелам.

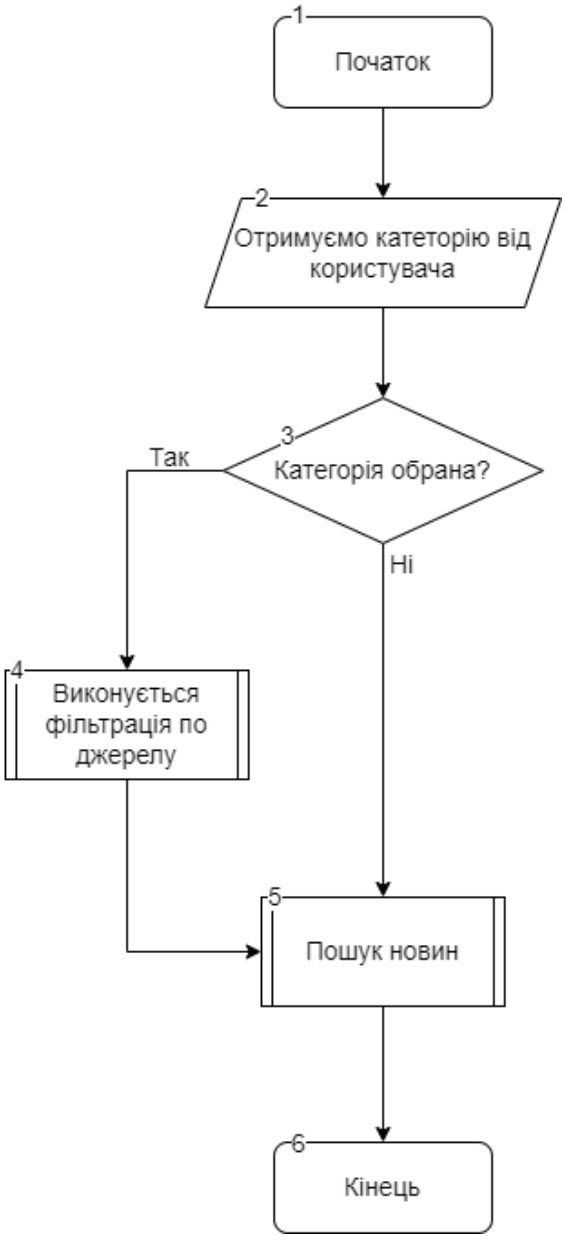

Рисунок 2.4 – Блок-схема алгоритму фільтрації новин

У перспективі можлива інтеграція механізмів машинного навчання, які зможуть автоматично пропонувати релевантні джерела або новини на основі історії переглядів і уподобань користувача, що зробить систему ще зручнішою та ефективнішою для щоденного використання.

Таким чином, модуль фільтрації новин є не лише технічним інструментом для відбору контенту, але й важливим елементом персоналізації та адаптації інформаційного середовища до потреб конкретного користувача.

2.5 Розробка модуля повної інформації про новину

Модуль "Повна інформація про новину" відповідає за витягнення повного тексту новинної статті за вказаним посиланням (рис. 2.5). Його мета – надати користувачу не лише заголовок та короткий опис новини (які зазвичай доступні через RSS), а й повний зміст новинної публікації у зручному та читабельному вигляді, що дозволяє отримати максимально повну картину події без необхідності переходу на сторонні ресурси.

Для реалізації цього функціоналу використовується парсер HTML-сторінок, який здійснює завантаження вмісту вебсторінки за відповідним URL та виконує обробку DOM-структури з метою вилучення основного текстового контенту. Особлива увага приділяється фільтрації технічних елементів сторінки, таких як навігаційні панелі, рекламні блоки, футери, бокові панелі та інші допоміжні частини інтерфейсу, які не належать до змісту новини та можуть відволікати користувача або створювати інформаційний шум. Для цього можуть застосовуватися спеціальні бібліотеки (наприклад, Readability.js) або власноруч розроблені алгоритми, що дозволяють визначати та ізолювати текстову частину статті на основі HTML-тегів, класів елементів, їхніх розмірів або семантичного наповнення.

Отриманий текст передається на клієнтську частину, де відображається у спеціально виділеному блоці повного перегляду новини. Завдяки цьому користувач може ознайомитися з усім змістом статті, не переходячи на сторонні сайти, що покращує зручність та швидкість доступу до інформації. Крім основного тексту, за потреби, може завантажуватися також супутній контент — зображення, відео, цитати, таблиці або графіки, які є частиною основної публікації та доповнюють її зміст.

У перспективі модуль може бути доповнений підтримкою багатомовності, автоматичним визначенням основного змісту навіть у складних або погано структурованих HTML-сторінках, а також функціями машинного перекладу чи автоматичного текстового аналізу для виявлення ключових фактів, понять або

навіть емоційного забарвлення публікацій. Це значно розширить можливості застосування системи, дозволить інтегрувати її у багатонаціональні чи тематично специфічні проєкти, а також сприятиме створенню персоналізованих стрічок новин з урахуванням інтересів і потреб конкретних користувачів.

Крім того, модуль передбачає обробку ситуацій, коли сторінка новини недоступна (наприклад, видалена або переміщена) або не містить очікуваного контенту (наприклад, коли сайт блокує парсинг ботами або використовує складну JavaScript-навігацію).

Завдяки цій функціональності модуль "Повна інформація про новину" відіграє важливу роль у забезпеченні інформативності та самодостатності програмного продукту, дозволяючи користувачу отримувати повноцінні новини без потреби покидати межі застосунку

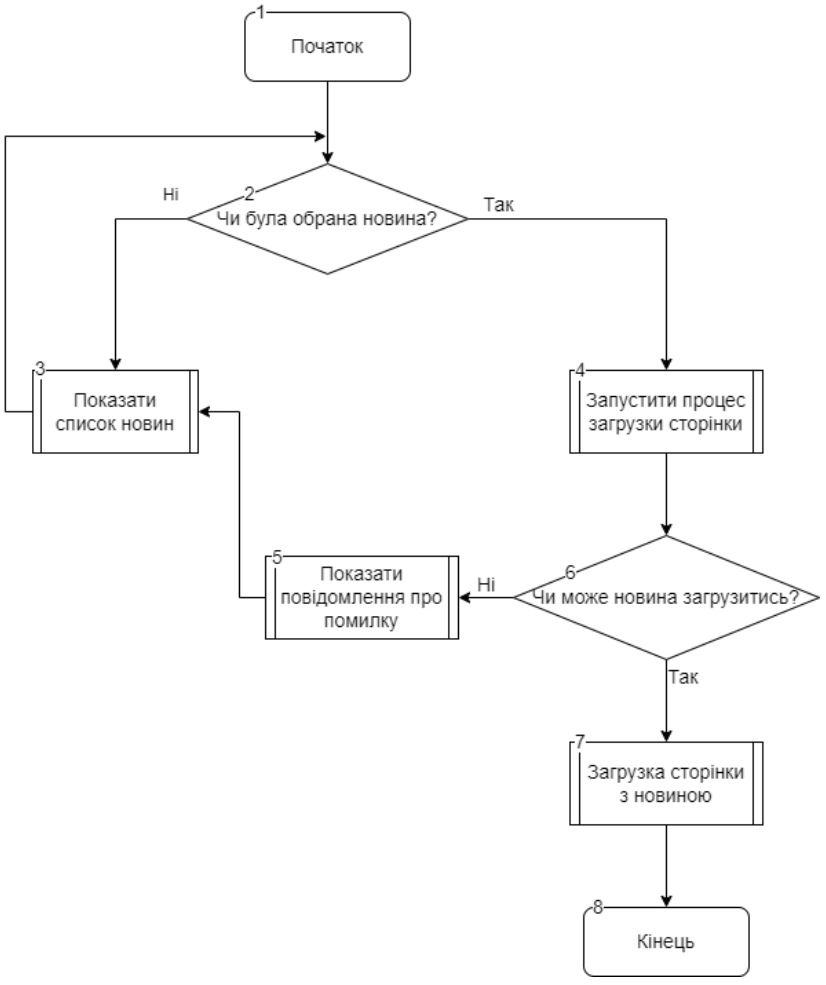

Рисунок 2.5 – Блок схема логіки інтерфейсу для показу новин

Призначення модуля.

У багатьох випадках RSS-стрічки обмежуються лише короткими анотаціями або уривками з повної статті. Модуль дозволяє:

– завантажити повний вміст новини з сайту-джерела;

– очистити сторінку від зайвих елементів (меню, реклама, коментарі тощо);

– надати користувачу текст у структурованому вигляді для подальшого ознайомлення або аналізу.

Принцип роботи:

1. Отримання URL-адреси новини.

Користувач, натискаючи на кнопку "Детальніше" або аналогічний елемент, передає URL новини як параметр url у GET-запиті до маршруту /article.

2. Завантаження HTML-коду сторінки.

За допомогою функції fetch() сервер здійснює HTTP-запит до вказаного сайту, отримуючи повний HTML-код новинної сторінки.

3. Формування DOM та парсинг контенту.

Використовується бібліотека JSDOM [21], яка створює віртуальне DOM-середовище з HTML-коду. Це дозволяє поводитися з вмістом так, ніби він відкритий у браузері.

4. Виділення основного тексту новини.

Для отримання лише головного тексту статті використовується бібліотека Readability від Mozilla. Вона автоматично виділяє контент, який найімовірніше є основною частиною новини (заголовок, текст, параграфи), і очищує сторінку від зайвих елементів (навігації, банерів, коментарів).

5. Формування відповіді.

У відповідь клієнту надсилається HTML-контент новини, або повідомлення про помилку, якщо витягнути статтю не вдалося.

Сервер виконає завантаження сторінки https://example.com/news-article-123, витягне основний текст новини та надішле його у відповідь.

Обробка помилок.

Якщо URL не доступний, містить некоректний формат або сайт захищено від сканування – користувачу виводиться повідомлення:

Не вдалося витягти текст статті.

У випадку внутрішньої помилки серверу (наприклад, помилка мережі чи парсингу) – повертається відповідь з кодом 500.

Рисунок 2.6 – Блок схема логіки роботи показу повної новини

Завдяки впровадженню цього модуля користувач отримує можливість швидко та зручно ознайомлюватися з повним вмістом новин без необхідності

переходити на сторонні ресурси або відволікатися на рекламу та зайві елементи сайту. Це значно підвищує загальну якість користувацького досвіду та дозволяє зосередитися саме на інформаційному наповненні матеріалу. У перспективі модуль може бути доопрацьований для підтримки багатомовності, інтеграції з системами машинного перекладу та автоматичного узагальнення текстів для ще більшої зручності користувачів.

2.6 Висновки

У другому розділі було здійснено варіантний аналіз та обґрунтовано вибір технологій для реалізації програмного забезпечення, призначеного для збору, обробки та візуалізації новинних даних з різних джерел. Зважаючи на задачі застосунку, було обрано стек технологій на основі мови програмування JavaScript із використанням Node.js на серверній стороні та Vue.js для реалізації клієнтського інтерфейсу. Такий вибір дозволив забезпечити високу швидкодію, гнучкість розширення функціоналу та зручність роботи як для розробників, так і для кінцевих користувачів.

На серверній стороні застосунку реалізовано обробку запитів від клієнта, парсинг RSS-стрічок, збереження та оновлення списку джерел. Для обробки RSS-фідів використано популярну бібліотеку rss-parser, яка забезпечує зручний доступ до структури новин та дозволяє ефективно працювати з великою кількістю джерел інформації.

З метою гнучкого збереження та модифікації джерел новин було реалізовано модуль додавання RSS-джерел, який дозволяє користувачу самостійно розширювати перелік джерел у будь-який момент часу без потреби втручання у код чи конфігурацію системи. Нові джерела зберігаються у локальному JSON-файлі, що спрощує управління даними та дозволяє швидко змінювати їх перелік.

Для отримання новинної інформації з вказаних джерел створено пошуковий модуль, який здійснює завантаження RSS-стрічок, фільтрацію новин за вказаними ключовими словами, перевірку наявності співпадінь у заголовках та описах, формування результатів пошуку та їх повернення клієнтові у зручному форматі. Завдяки цьому користувач може оперативно знаходити актуальну та релевантну інформацію серед великої кількості джерел.

Крім того, було розроблено модуль фільтрації, який дозволяє обмежити результати пошуку новинами лише з вибраних користувачем джерел. Це забезпечує можливість більш точного налаштування параметрів пошуку згідно з інтересами або професійними потребами користувача, наприклад, для отримання новин лише з певної тематики або від конкретного ресурсу.

Окрему увагу приділено модулю отримання повного тексту новини, який автоматично витягує основний зміст новинної статті з вебсторінки за вказаним у стрічці посиланням. Для реалізації цього функціоналу використано бібліотеки JSDOM та Readability, що дозволяють перетворити HTML-структуру сторінки на зручний для читання текстовий блок, очищений від сторонніх елементів — таких як реклама, навігаційні меню, колонтитули та інші несуттєві елементи інтерфейсу. Це значно підвищує якість подання інформації для користувача та зменшує кількість відволікаючих факторів.

Таким чином, у розділі було розглянуто структуру, функціонування та взаємодію основних модулів програмного забезпечення: пошуку, додавання джерел, фільтрації результатів, отримання повного тексту новин. Обрані технологічні рішення дозволяють створити ефективний, масштабований, зручний у використанні та адаптивний до змін у мережевому середовищі вебзастосунок для збору, обробки та аналізу новинного контенту з широкого спектру джерел.

3 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ

3.1 Аналіз вхідних даних системи

Для реалізації програмного продукту для збору та аналізу новин на основі вебсистеми будуть використані мова програмування Vue.js для розробки фронтенду та Node.js для бекенду, а також алгоритми для роботи з RSS-джерелами та фільтрацією новин. Vue.js – це популярний фреймворк для побудови інтерфейсів, що забезпечує гнучкість і простоту інтеграції з іншими технологіями. Node.js дозволяє ефективно обробляти запити з високою швидкістю завдяки асинхронному підходу. Для обробки новин з різних джерел буде використано RSS, що дозволяє здійснювати підключення до новинних каналів, отримувати та обробляти дані.

Розробка програмних засобів для збору та аналізу новин включає кілька етапів. По-перше, потрібно створити модуль для взаємодії користувача з вебсистемою. Цей модуль буде відповідати за інтерфейс користувача та навігацію по функціоналу вебсистеми. Наступним етапом буде розробка модуля для імпорту новин з різних RSS-джерел. Цей модуль буде здійснювати підключення до різних джерел новин та завантажувати відповідні дані. Після цього розробимо модуль для фільтрації новин за категоріями, що дозволить користувачам налаштовувати параметри відображення новин.

Завершальним етапом є створення модуля для відображення та взаємодії з новинами на вебсистемі. Цей модуль дозволить користувачам переглядати новини, а також здійснювати фільтрацію та пошук за різними категоріями.

Розробка цих програмних модулів вимагає глибоких знань у веб-розробці та досвіду роботи з фреймворками на основі JavaScript. Для реалізації цих функцій будуть використані такі інструменти як Vue.js, Node.js, а також бібліотеки для роботи з RSS, зокрема rss-parser.

Отже, розробка програмних модулів для збору та аналізу новин з використанням технологій Vue.js та Node.js є комплексним завданням, яке

включає кілька етапів. Модулі повинні бути ретельно спроектовані та протестовані для забезпечення точності та швидкості збору новин та їх фільтрації. Маючи необхідні інструменти та досвід, можна створити програмне забезпечення, яке задовольнить потреби користувачів і відкриє нові можливості в сфері збору та аналізу новин.

3.2 Розробка інтерфейсу програмного додатку

Розробка графічного інтерфейсу користувача (GUI) є важливою складовою вебсистеми збору та аналізу новин. Основна мета – забезпечити просту, інтуїтивну та зручну навігацію для користувача, який взаємодіє з великою кількістю інформації.

Інтерфейс було реалізовано з використанням технології Vue.js – сучасного JavaScript-фреймворку для створення адаптивних односторінкових додатків. Для стилізації інтерфейсу застосовано CSS у поєднанні з бібліотеками Tailwind CSS та компонентами, побудованими за принципами модульності.

Основним кольором оформлення інтерфейсу обрано чорний, як універсальний фон для текстової інформації, що не перевантажує зір користувача. Для акцентування уваги використовується темно-синій (для заголовків, кнопок, навігації) та світло-сірий (для відокремлення блоків, форм та списків).

Для акцентування уваги на ключових елементах інтерфейсу використано темно-синій колір. Він застосовується для заголовків, кнопок, елементів навігації та активних компонентів, оскільки добре поєднується з чорним фоном і водночас має достатній візуальний акцент. Такий вибір кольору також асоціюється зі стабільністю, надійністю та технологічністю, що позитивно впливає на загальне сприйняття застосунку.

Меню навігації (верхня панель)

Верхнє горизонтальне меню надає користувачу доступ до основних функцій вебдодатку (рисунок 3.1). Меню містить пункти:

– «Головна» – повернення до списку новин;

– «Фільтрація» – перехід до фільтрації новин;

– «Додати джерело» – форма для введення RSS-посилання.

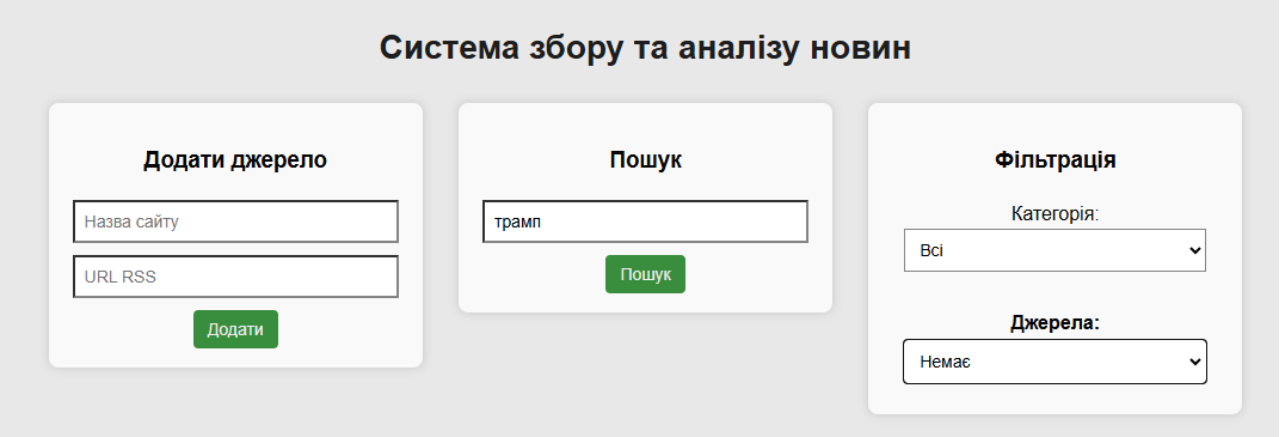

Рисунок 3.1 – Головне меню сайту

Фільтрація новин за категоріями

У правій сторінки реалізовано список доступних категорій, наприклад:

– політика;

– технології;

– економіка;

– спорт.

Користувач може вибрати одну або кілька категорій, після чого вміст сторінки динамічно оновиться й покаже лише релевантні новини (рисунок 3.2).

Крім того, передбачено можливість скасування вибору категорій для повернення до перегляду всіх новин або зміни параметрів пошуку без необхідності оновлення сторінки, що сприяє покращенню зручності використання системи. Завдяки цьому користувач може швидко адаптувати результати пошуку відповідно до своїх поточних потреб — наприклад, миттєво змінити тему новин або джерело інформації без перезавантаження застосунку. Така функціональність забезпечує динамічність інтерфейсу та зменшує час на виконання додаткових дій, що особливо важливо для користувачів.

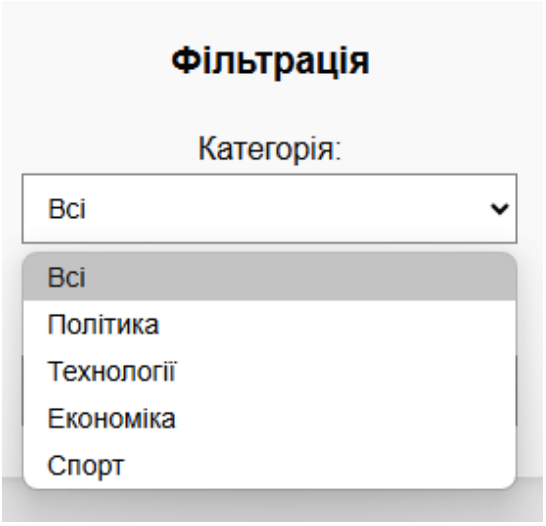

Рисунок 3.2 – Фільтрація по категоріям новин

Фільтрація за джерелами

Окрім категоріальної фільтрації, в системі реалізовано фільтрацію новин за конкретними RSS-джерелами, які користувач самостійно додав. У відповідному бічному меню (sidebar) або випадаючому списку відображається перелік джерел з назвами сайтів або описами RSS-каналів (рисунок 3.3).

Користувач може:

– обрати джерело для перегляду лише їхніх новин;

– швидко скинути фільтр за допомогою кнопки «Немає».

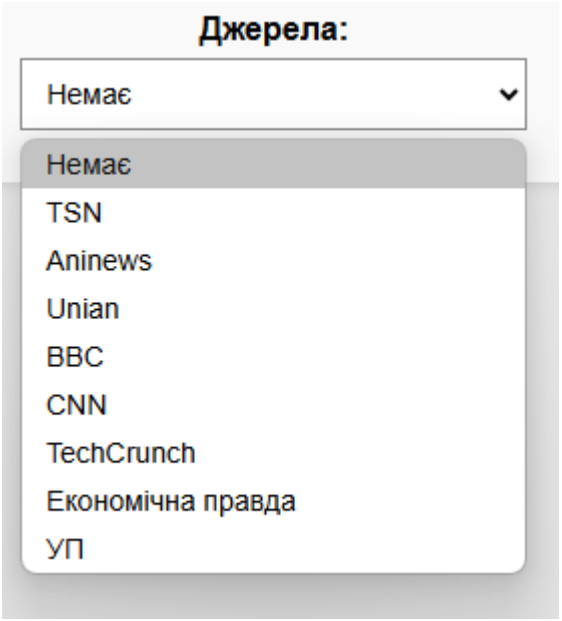

Рисунок 3.3 – Фільтрація по категоріям новин

Список новин

Центральний елемент системи – список новин, зібраних із різних RSS-джерел. Кожна новина відображається у вигляді окремої картки, яка містить:

– заголовок новини;

– короткий опис або перші речення з тексту;

– назву джерела;

– активне посилання на повний текст на оригінальному сайті.

Всі новини автоматично завантажуються у фоновому режимі.

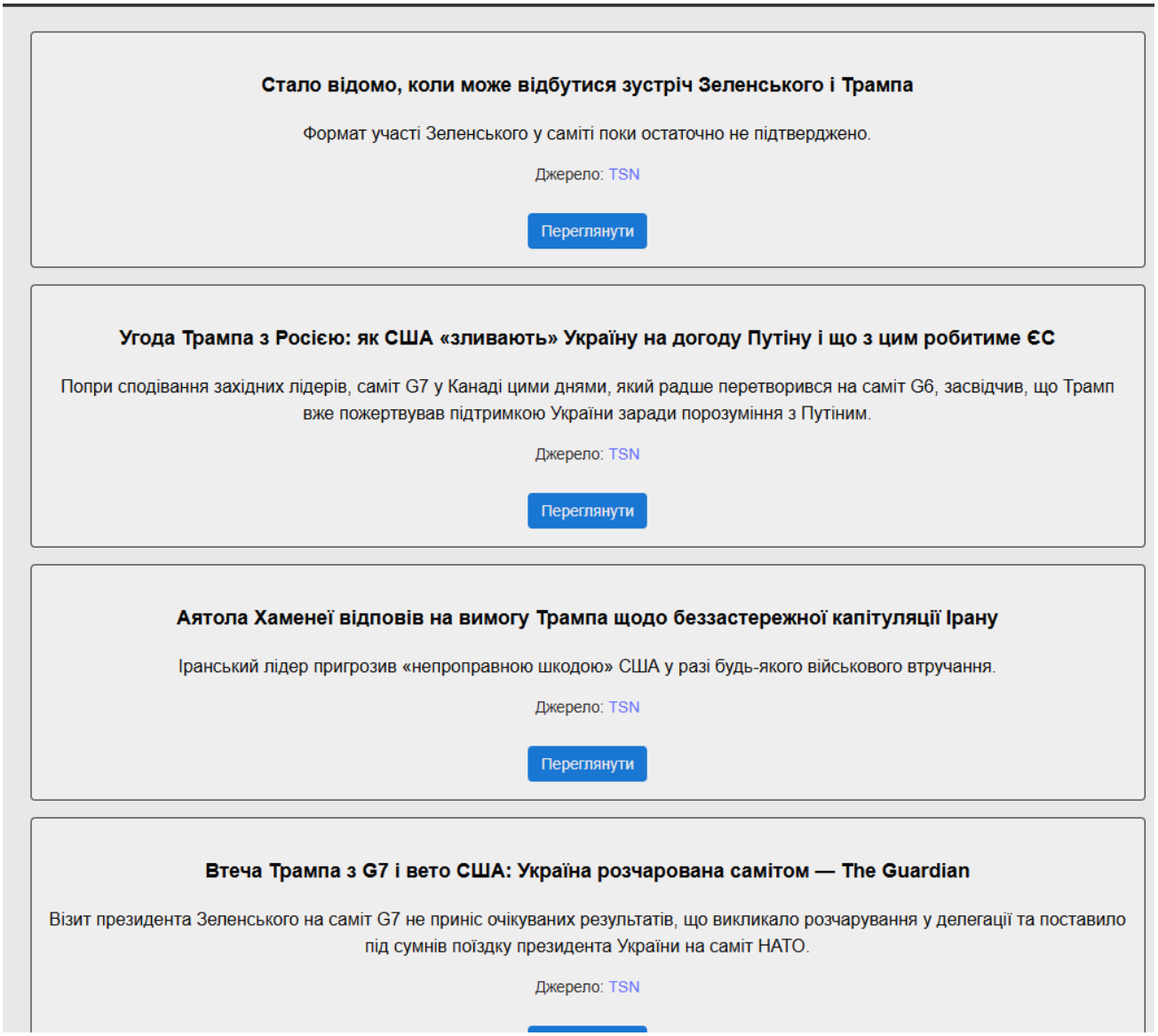

Рисунок 3.4 – Блок списку новин

Сторінка повної інформації новини

Для забезпечення зручного перегляду новин детально реалізовано окрему сторінку перегляду повної інформації про новину. Вона відкривається після натискання на кнопку «Переглянути» на картці новини.

На цій сторінці користувач бачить:

– повний заголовок новини (великий шрифт, з акцентом);

– джерело новини (із посиланням на оригінал);

– повний текст або розширений опис, якщо такий доступний із RSS;

– зображення або прев’ю, якщо надані у стрічці.

Інтерфейс сторінки адаптований для мобільних пристроїв. Для зворотного переходу передбачено кнопку «Назад до результатів» (рисунок 3.5).

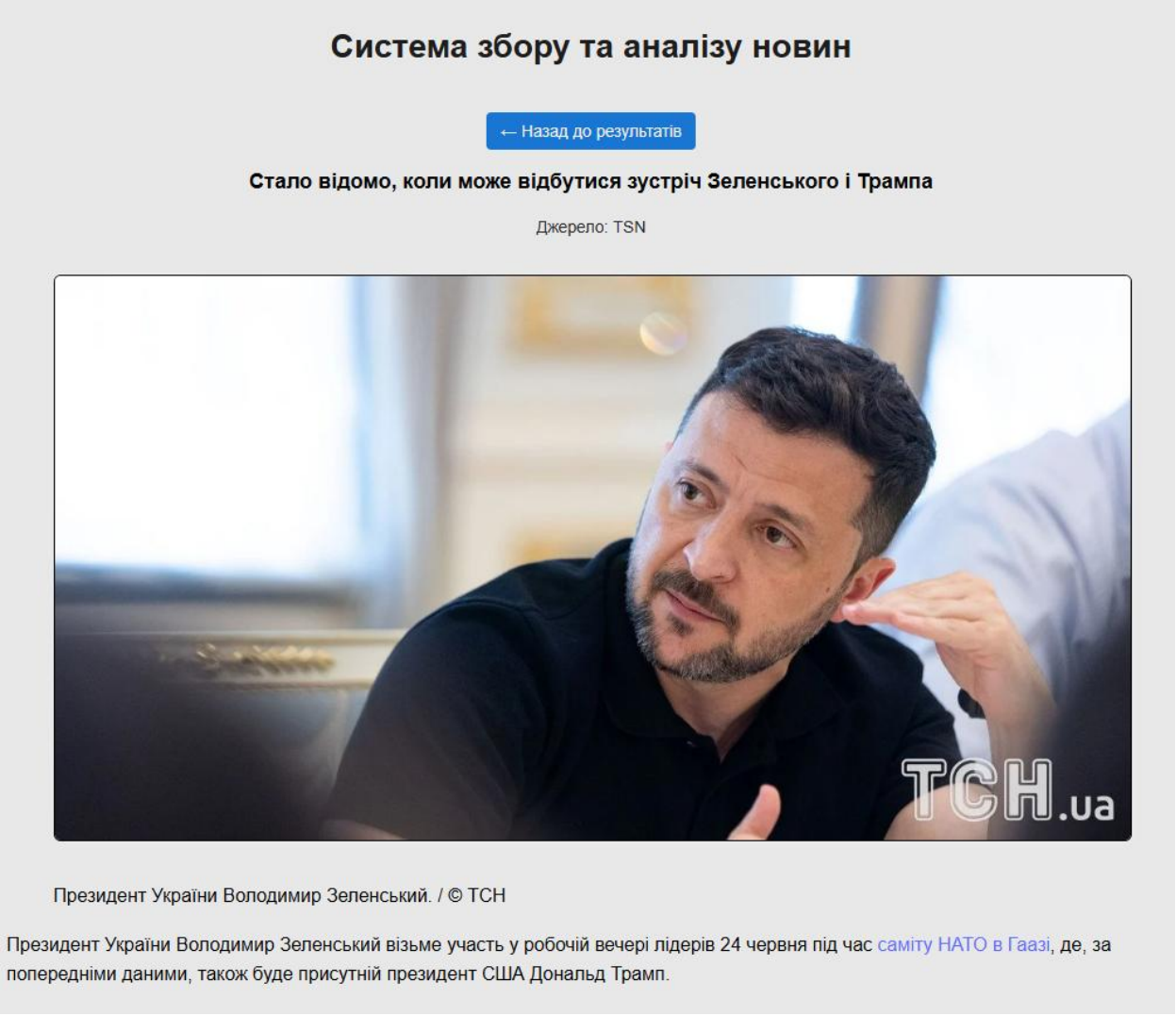

Рисунок 3.5 – Деталі вибраної новини

Ці функції суттєво розширюють можливості користувача щодо навігації та персоналізації відображення новин, роблячи інтерфейс не лише зручним, а й функціонально потужним.

3.3 Розробка моделі системи

Для кращого розуміння роботи модулів програмного засобу збору та аналізу новин було вирішено розробити модель роботи системи у вигляді UML-діаграми діяльності (рис. 3.6). Згідно з вказаною діаграмою, користувач починає взаємодію із програмним продуктом, імпортуючи новинні джерела за допомогою інтерфейсу додавання RSS-URL. Якщо формат джерела некоректний або недоступний, система повідомляє користувача про помилку та пропонує перевірити правильність введеного посилання.

У разі успішного додавання джерела, система проводить первинний аналіз його вмісту, перевіряє структуру новин та зберігає отримані дані у базі даних. Далі користувач може перейти до перегляду зібраних новин. Інтерфейс дозволяє переглядати всі новини в загальному списку або використовувати систему фільтрації – наприклад, за джерелами, категоріями чи датами публікації. При виборі певного джерела відображаються лише ті новини, що були отримані саме з нього.

Після вибору конкретної новини, користувач може перейти на сторінку з повною інформацією новини. На цій сторінці відображаються заголовок, повний текст публікації, джерело, дата публікації, а також ключові метадані – наприклад, категорія новини або тегування, якщо воно реалізоване. Інтерфейс цієї сторінки забезпечує зручне читання та можливість повернутися до загального списку або перейти до інших пов’язаних новин.

Окрім перегляду, користувач може ініціювати повторне оновлення джерел для отримання нових новин, після чого система запускає повторне сканування наявних RSS-каналів і оновлює базу даних. Після цього нові записи автоматично додаються до загального списку, з урахуванням обраних фільтрів.

Таким чином, UML-діаграма діяльності відображає ключові етапи роботи користувача з системою: від додавання новинного джерела до перегляду, фільтрації та детального аналізу новин. Вона наочно демонструє логіку послідовної взаємодії користувача з інтерфейсом застосунку, а також показує, як саме система реагує на дії користувача, переходячи між різними станами. Крім того, діаграма може бути використана як основа для подальшого розвитку функціоналу, зокрема – автоматизації певних процесів або додавання нових сценаріїв використання.

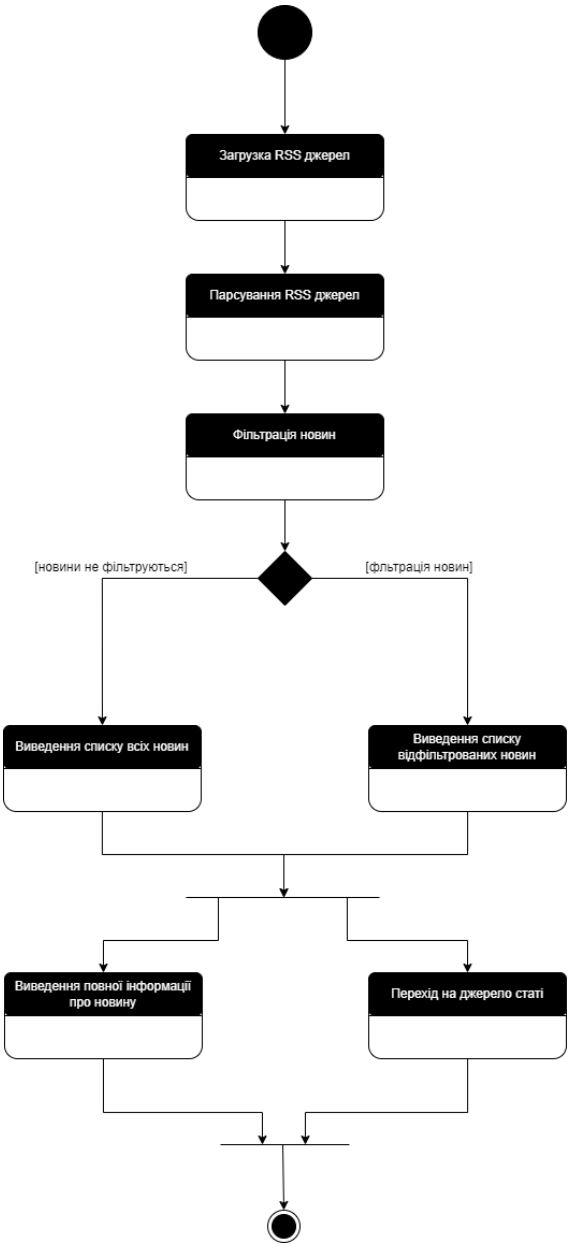

Рисунок 3.6 – Модель роботи системи у вигляді діаграми діяльності

Розробка діаграми діяльності була виконана у програмному забезпеченні draw.io.

3.4 Розробка алгоритмів роботи системи

Для кращого розуміння логіки функціонування програмного застосунку збору та аналізу новин розроблено блок-схему (рис. 3.7), яка ілюструє основні етапи алгоритму роботи системи. Дана діаграма дозволяє наочно уявити послідовність дій користувача, внутрішню обробку команд системою, а також забезпечує краще розуміння структури взаємодії між компонентами застосунку. Вона є зручною як для кінцевого користувача, так і для розробника, оскільки дає змогу швидко оцінити логіку переходів між станами системи.

Згідно з діаграмою, процес починається з запуску вебзастосунку (блок 1–2), під час якого відбувається ініціалізація всіх необхідних модулів та компонентів. Після завантаження головного інтерфейсу користувач отримує доступ до головного меню і переходить до вибору однієї з доступних функцій (блок 3): додавання нового RSS-джерела, фільтрації новин за категоріями або джерелами, або ініціювання автоматичного процесу збору новин (блоки 4–6 відповідно).

У разі вибору функції збору новин (блок 6), система активує механізм парсингу, який виконує обхід раніше доданих джерел, завантажує нові записи та обробляє їх відповідно до заданих параметрів. Результатом цього процесу є виведення списку знайдених новин на екран (блок 7), у якому кожна новина містить короткий опис, джерело, дату публікації та посилання.

Після цього користувач має змогу обрати подальшу дію (блок 8). Зокрема, він може переглянути детальну інформацію новини (блок 10), яка відкривається у форматі розширеного опису з повним текстом або коротким зведенням. Також доступна функція перегляду повної інформації про джерело новини (блок 9), що дозволяє оцінити надійність та контекст джерела.

Після завершення виконання вибраної дії робота користувача із застосунком або завершується (блок 11), або ж переходить до повторного циклу, що включає новий вибір функціоналу – наприклад, додавання ще одного джерела чи новий запит на фільтрацію новин. Такий підхід забезпечує безперервну роботу користувача у зручному циклі, з можливістю миттєвого реагування на зміну інформаційних потреб.

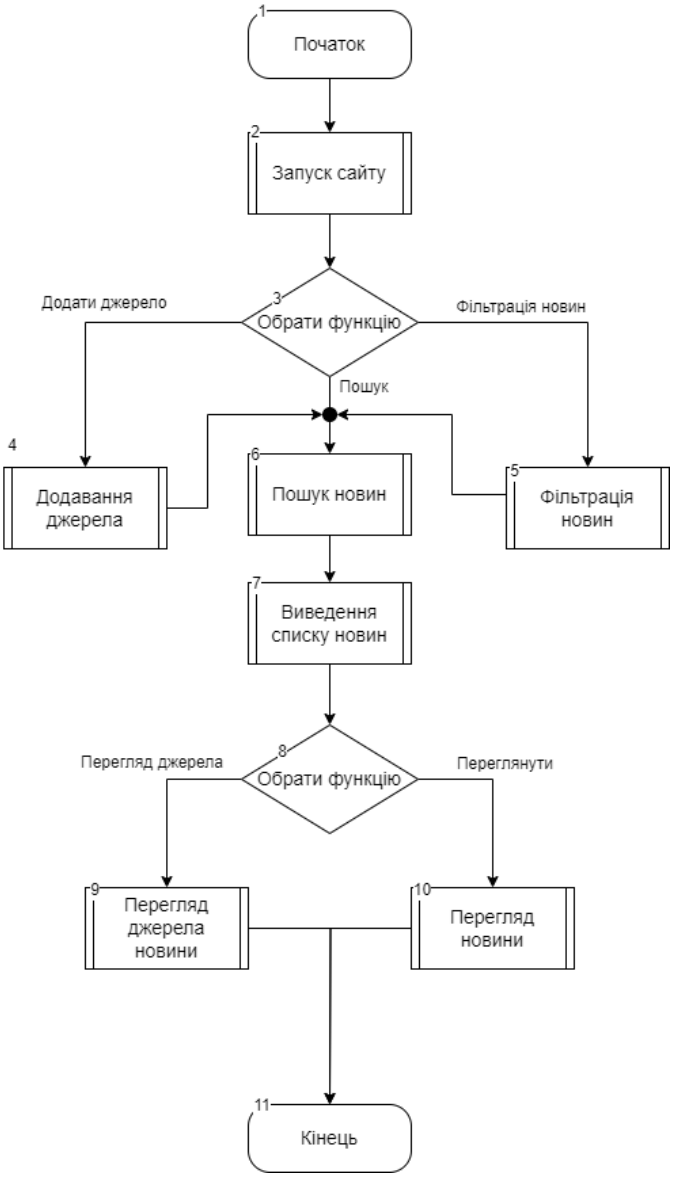

Рисунок 3.7 – Алгоритм роботи програмного застосунку

Ця блок-схема демонструє гнучкість побудови інтерфейсу та логічну послідовність реалізації функціоналу, що, у свою чергу, робить програмний

продукт інтуїтивно зрозумілим, гнучким у використанні та ефективним для аналізу великого обсягу інформаційного контенту з різних джерел. Така структурована побудова логіки також полегшує масштабування та подальшу модернізацію системи.

3.5 Висновки

У третьому розділі було проведено детальний аналіз вхідних та вихідних даних програмного забезпечення для збору та аналізу новин. Визначено основні типи інформації, які обробляє система, зокрема параметри пошукового запиту (ключові слова), вибрані джерела новин (одне або декілька RSS-джерел), категорії новин (наприклад, політика, економіка, спорт, технології), а також структуру новинних статей, що включає заголовок, короткий опис, посилання на оригінальне джерело та повний текст публікації, який завантажується за потреби.

Особливу увагу було приділено формуванню та аналізу структури запитів, що надходять від користувача на сервер, а також відповіді, яку отримує клієнт у вигляді масиву об’єктів новин із необхідними полями. Це дозволяє забезпечити коректну взаємодію між клієнтською та серверною частинами застосунку.

Окрім цього, було розглянуто процес створення графічного інтерфейсу вебзастосунку, орієнтованого на зручну та інтуїтивно зрозумілу взаємодію з користувачем. Визначено основні елементи інтерфейсу: панель пошуку для введення запиту, список доступних джерел новин, модуль фільтрації за категоріями для уточнення результатів, область відображення результатів пошуку, а також блок перегляду повної інформації про вибрану новину. Було докладно описано логіку взаємодії між цими елементами з метою забезпечення максимальної ефективності та простоти роботи користувача із системою. Окремо розглянуто сценарії обробки ситуацій, коли результати пошуку відсутні або введені параметри є некоректними — у таких випадках передбачено відповідні інформаційні повідомлення.

Також розроблено основний алгоритм функціонування системи, який включає послідовні процеси: запуск вебсайту, завантаження списку доступних RSS-джерел з локального файлу, можливість додавання нових джерел через відповідний інтерфейс, виконання пошуку новин за вказаними ключовими словами, застосування фільтрів за джерелами та категоріями новин, а також перегляд повної інформації про вибрану новину. У процесі розробки алгоритму особливу увагу приділено обробці можливих помилкових сценаріїв, таких як відсутність новин за запитом, введення неприпустимих параметрів або спроба пошуку без вибраних джерел, із відповідним сповіщенням користувача для запобігання непередбачуваній поведінці системи.

Для наочного представлення загальної логіки роботи програмного продукту було створено модель функціонування системи у вигляді UML-діаграми діяльності. Діаграма ілюструє основні етапи взаємодії користувача із застосунком: від початкового завантаження сторінки до виконання пошуку, застосування фільтрів, перегляду повної новини або додавання нового джерела. Це моделювання дозволило виявити ключові точки прийняття рішень у системі, перевірити коректність та логічну завершеність усіх сценаріїв використання, а також забезпечити правильну послідовність виконання операцій як на клієнтській, так і на серверній стороні вебзастосунку.

Результати проведеного аналізу стали основою для реалізації функціональних модулів системи та їх подальшого тестування, забезпечуючи відповідність програмного продукту вимогам користувача та поставленим задачам.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування системи

Тестування програмного забезпечення є невід'ємною частиною процесу розробки вебзастосунку, оскільки дозволяє перевірити функціональність системи, стабільність її роботи та відповідність заданим вимогам. Основною метою тестування розробленої системи для збору та аналізу новин є виявлення помилок, перевірка коректності обробки RSS-даних, забезпечення стабільної роботи модулів пошуку, додавання джерел, фільтрації та перегляду повного тексту новин.

Тестування виконувалося у середовищі, де встановлений Node.js-сервер та клієнтська частина, розроблена з використанням Vue.js. Для перевірки роботи використовувалися сучасні браузери (Google Chrome, Mozilla Firefox) та тестова база RSS-джерел.

Першим етапом було функціональне тестування, під час якого перевірялась основна логіка роботи:

– пошук новин за ключовими словами;

– коректне додавання нових джерел через відповідну форму;

– фільтрація новин за обраними джерелами;

– завантаження повного тексту новини з вебсторінки.

Було протестовано, як система реагує на запити з порожнім або некоректним параметром q, та чи повертає вона релевантні результати для запитів, що точно відповідають заголовкам або описам новин у RSS-стрічках.

У всіх цих випадках програма повинна не припиняти свою роботу, а обробити виняток та повідомити користувача про проблему без впливу на решту функціоналу. Було підтверджено, що механізми обробки помилок у пошуку та при завантаженні повного тексту працюють коректно.

На другому етапі було проведено тестування продуктивності, яке включало:

– аналіз швидкодії пошуку новин при великій кількості джерел;

– час завантаження повного тексту новини;

– відгук інтерфейсу при фільтрації джерел.

Функціональне тестування проводиться з метою перевірки коректності роботи основного функціоналу вебзастосунку відповідно до поставлених вимог. Основні цілі на цьому етапі:

Перевірка модуля пошуку. Система повинна знаходити новини, які містять ключові слова в заголовку або описі (рис. 4.1). Тестування охоплювало: запити з різними регістрами (врахування регістру). Часткові та повні збіги з ключовими словами. Порожній запит – у цьому випадку система має повертати всі новини без фільтрації.

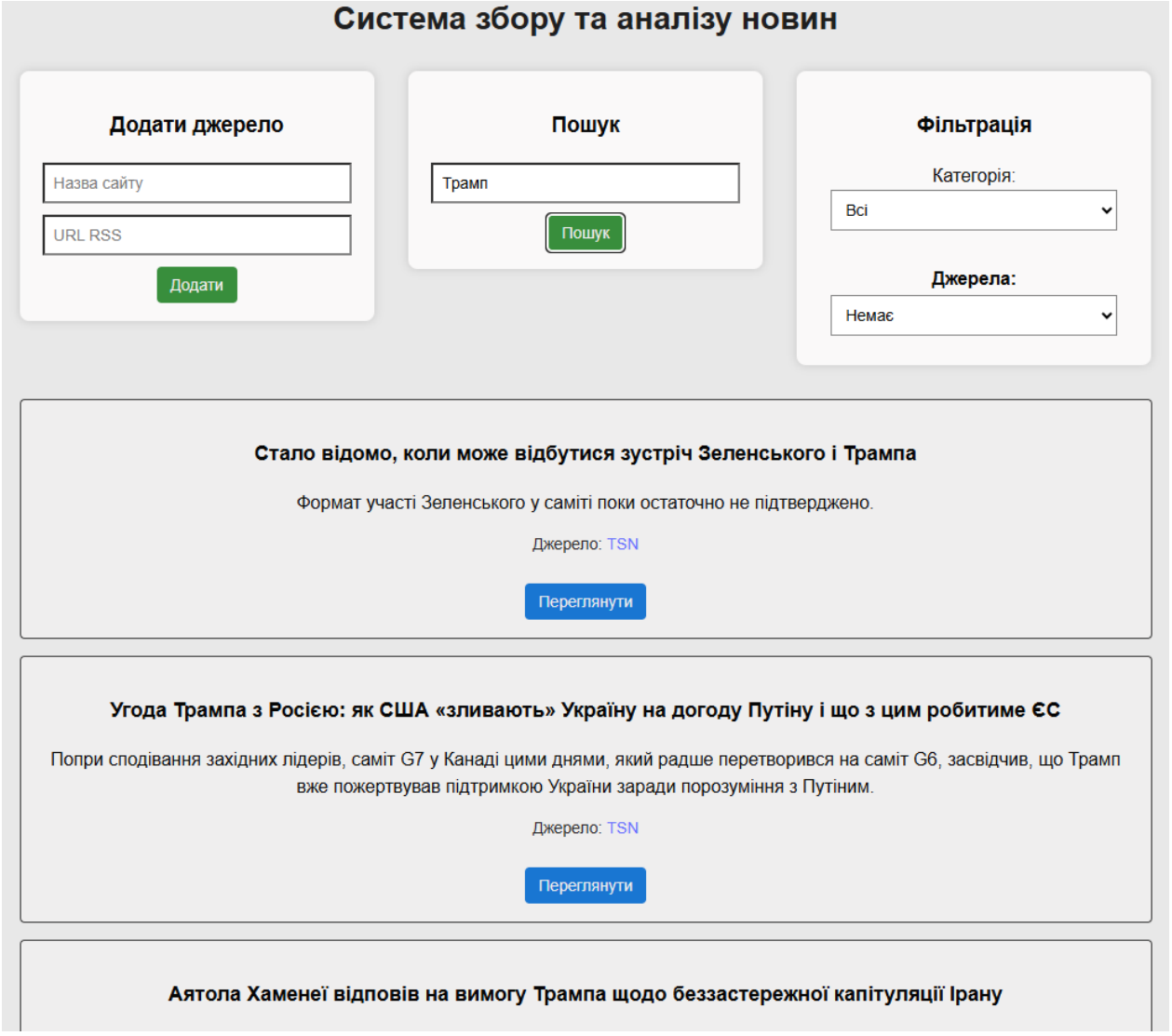

Рисунок 4.1 – Тестування модуля пошуку

Перевірка модуля додавання джерел. Система не повинна приймати порожні поля name або url (рис 4.2).

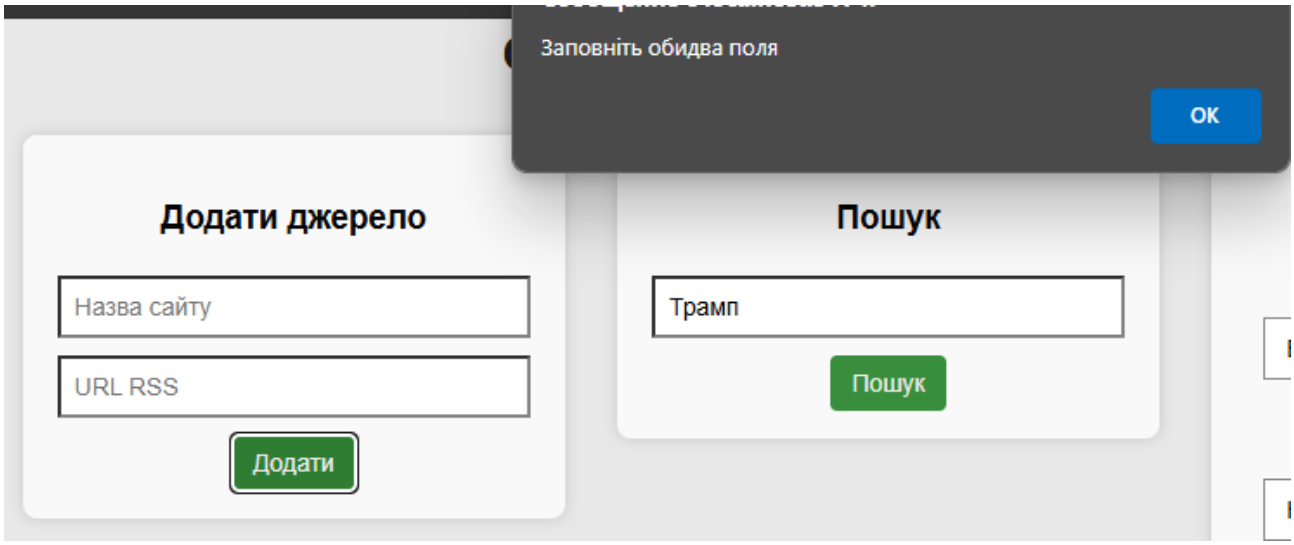

Рисунок 4.2 – Перевірка модуля на пусті поля

При успішному додаванні нового джерела воно має зберігатися до спільного списку (рис. 4.3 – 4.4).

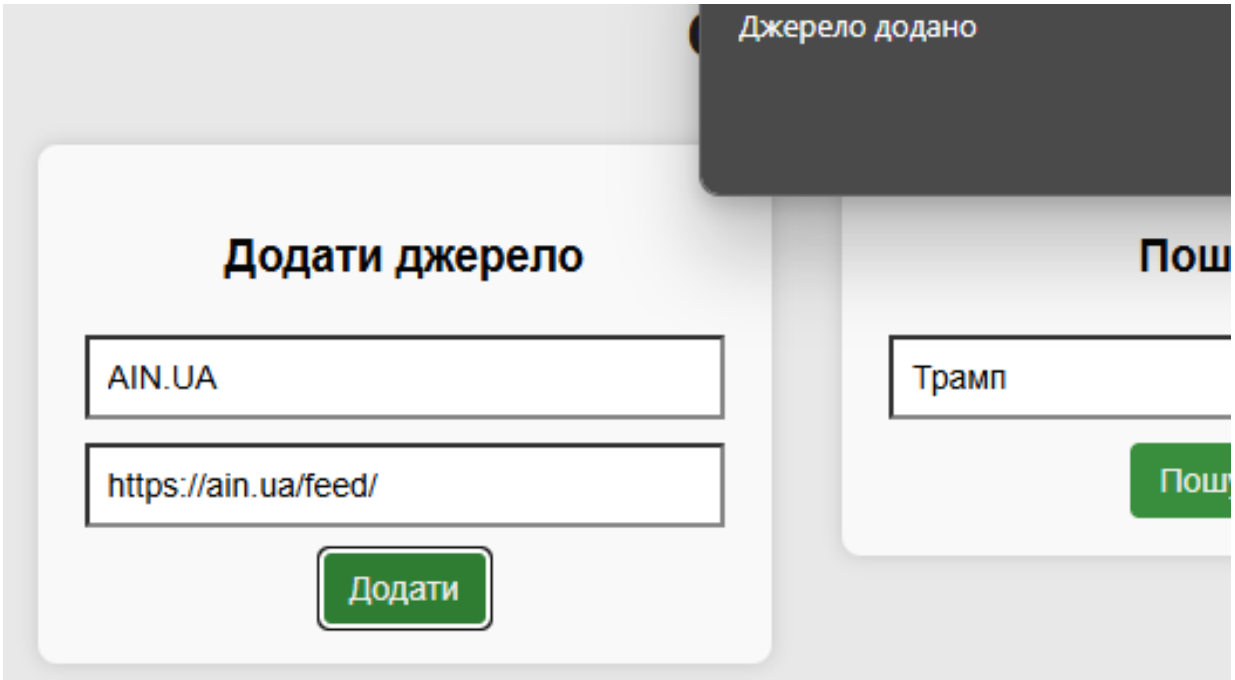

Рисунок 4.3 – Перевірка модуля додавання джерела

```
{
  "name": "CNN",
  "url": "http://rss.cnn.com/rss/edition.rss"
},
{
  "name": "TechCrunch",
  "url": "https://techcrunch.com/feed/"
},
{
  "name": "Економічна правда",
  "url": "https://www.epravda.com.ua/rss/"
},
{
  "name": "УП",
  "url": "https://www.pravda.com.ua/rss/"
},
{
  "name": "AIN.UA",
  "url": "https://ain.ua/feed/"
}
]
```


Рисунок 4.4 – Успішне додавання джерела у список джерел

Перевірка фільтрації новин за джерелами.

Вибір джерела повинен повертати лише новини, які до них належать (рис. 4.5). Якщо не вибрано жодного джерела – виводиться список з усіх.

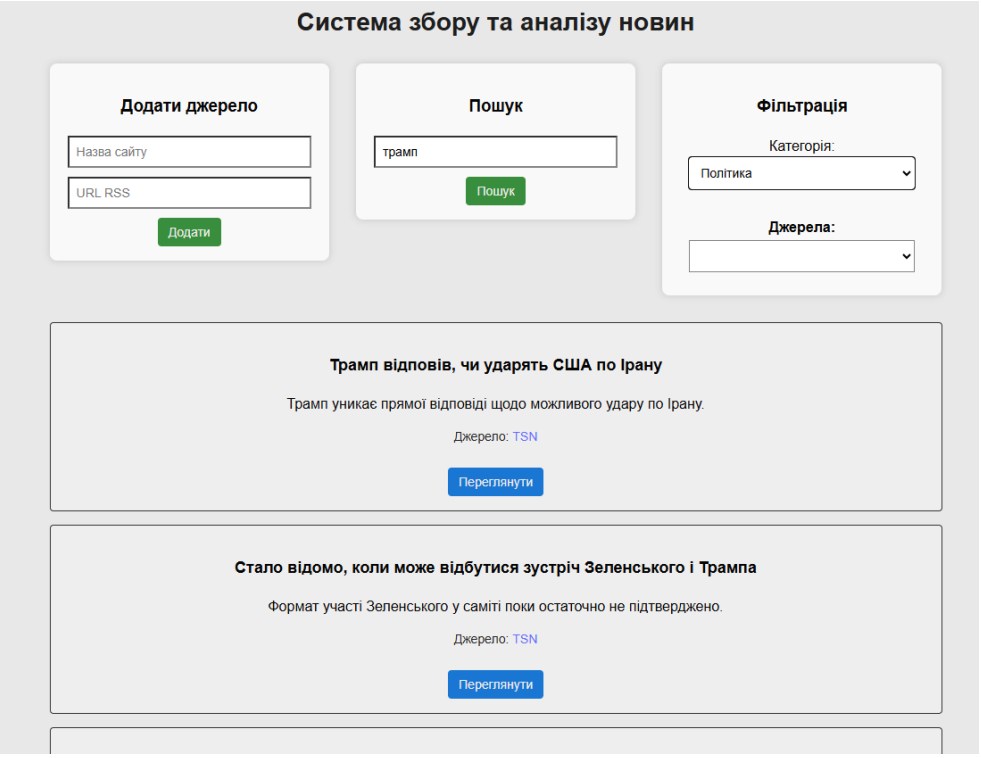

Рисунок 4.5 – Тестування модуля фільтрації

Перевірка модуля отримання повного тексту статті.

При вказаному коректному URL новини повинна повертатись очищена HTML-версія повного тексту (рис 4.6). У разі некоректного або недоступного URL система має відобразити повідомлення про помилку.

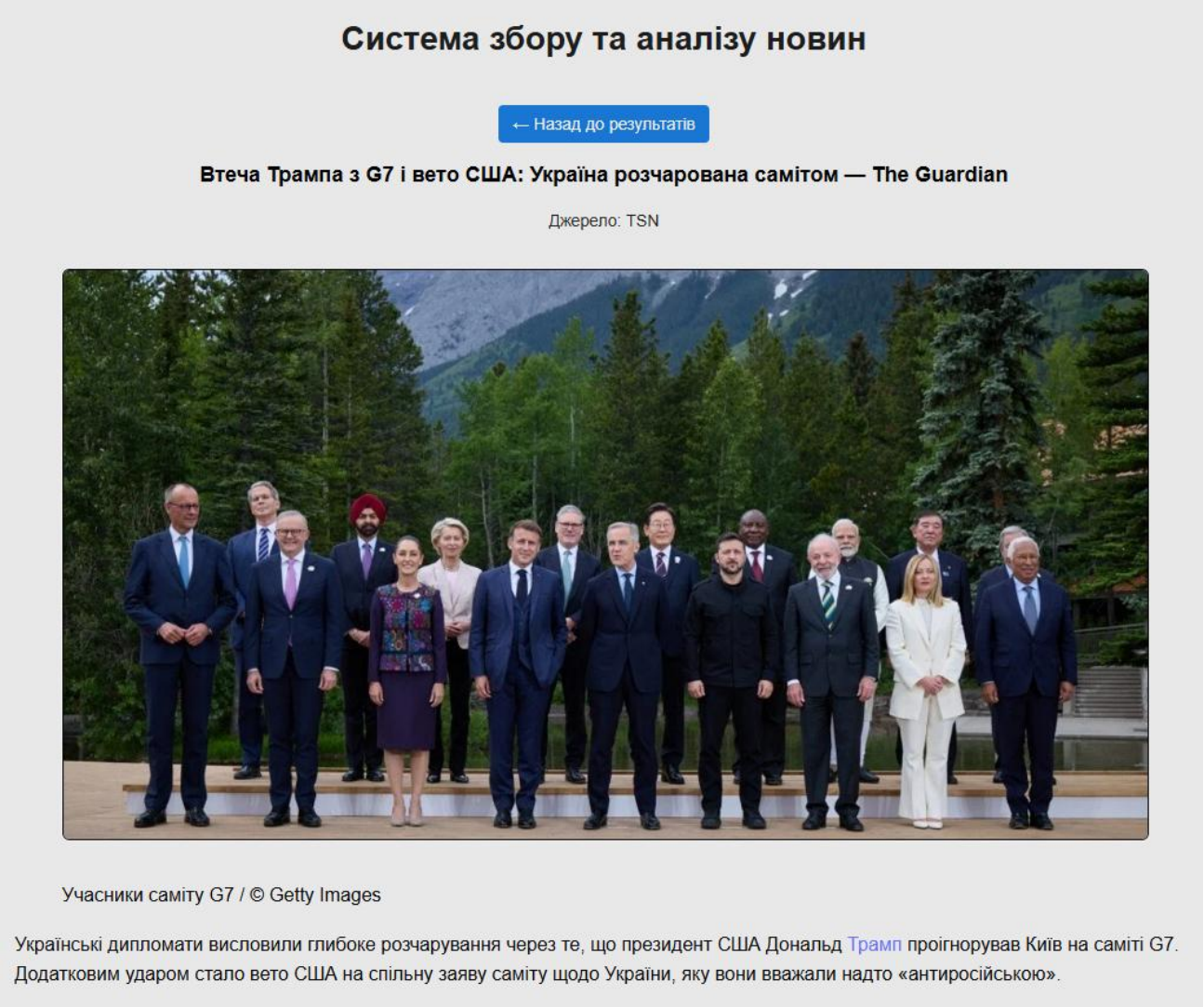

Рисунок 4.6 – Повна інформація про новину

Тестування продуктивності.

Мета цього етапу – визначити швидкість реакції системи та її поведінку при високих навантаженнях.

Швидкість пошуку.

Перевірка часу виконання запиту при різній кількості підключених RSS-джерел (від 5 до 50). У середньому при 15 джерелах пошук. виконується за <1 с, але при більшій кількості статей потрібні більше часу на обробку (рис. 4.7).

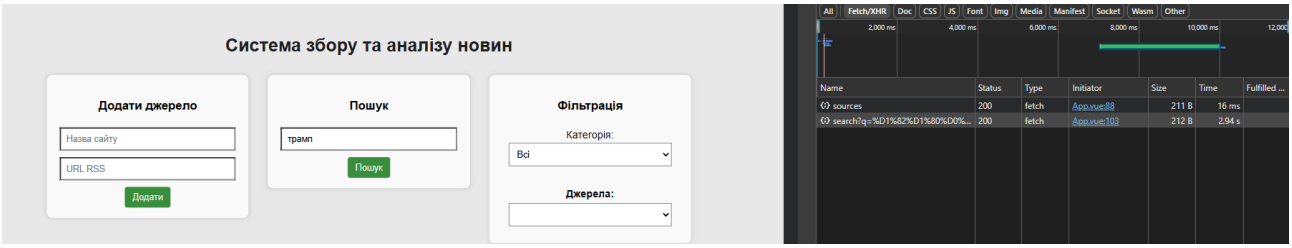

Рисунок 4.7 – Тестування швидкості пошуку

Час отримання повного тексту статті.

Залежно від складності HTML сторінки та швидкості мережі – 1–3 секунди. Обробка HTML через Readability виконується ефективно.

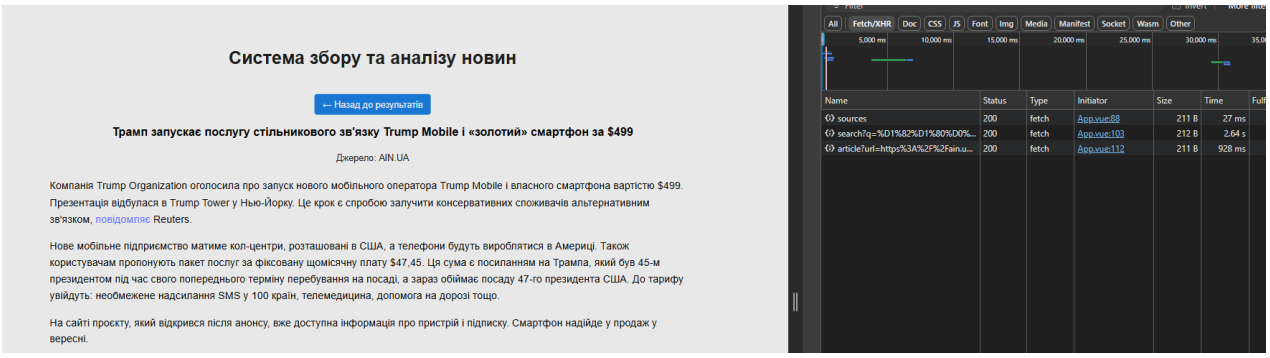

Рисунок 4.8 – Тестування швидкості отримання повного тексту статті

Під час тестування відмовостійкості система показала стабільну роботу навіть за умов відсутності доступу до деяких джерел або помилок у вхідних даних. Тестування продуктивності підтвердило, що вебзастосунок демонструє прийнятну швидкість обробки запитів. А тестування сумісності показало, що система є кросплатформенною і коректно працює в популярних браузерах та на різних пристроях.

Загалом, результати тестування свідчать про надійність, стабільність та зручність розробленого програмного забезпечення.

4.2 Розробка інструкції користувача

Інструкція користувача передбачає визначення технічних вимог для запуску програмного продукту. Деталі щодо мінімальної та рекомендованої конфігурації розміщено в таблиці 4.1.

Таблиця 4.1 – Вимоги до апаратного забезпечення

Вимоги до конфігурування	Рекомендовано	Мінімально
Процесор	4-ядерний, 2.5 ГГц або більше	2-ядерний, 1.8 ГГц
Оперативна пам'ять	8 ГБ або більше	4 ГБ RAM
Браузер	Остання версія Google Chrome або Mozilla Firefox	Сучасний браузер з підтримкою JavaScript (Chrome, Firefox)
Операційна система	Windows 11 / Ubuntu 22.04 / macOS 12+	Windows 10 / Linux Ubuntu 20.04 / macOS 10.14
Інтернет-з’єднання	20 Мбіт/с або вище для стабільної роботи з джерелами	5 Мбіт/с

Після натискання на гіперпосиланя користувач переходить на головну сторінку вебінтерфейсу, де відображається список останніх новин із попередньо доданих RSS-джерел.

Першим етапом взаємодії користувача передбачається можливість додавання власних джерел новин. Для цього в інтерфейсі передбачено форму, де необхідно ввести назву джерела та посилання на RSS-стрічку. Після підтвердження ці дані зберігаються на сервері, і з нового джерела одразу починає надходити інформація. Якщо додане джерело недоступне або має некоректний формат, система повідомить про помилку.

Наступним кроком є використання функції пошуку новин. Користувач може ввести ключові слова, за якими програма здійснить фільтрацію новин у всіх

доданих або обраних джерелах. Результати пошуку відображаються у вигляді списку новин, що містять заголовки, короткий опис, джерело та посилання на повну статтю.

Для детальнішого перегляду новини користувач натискає на відповідне посилання, після чого активується модуль повної інформації про новину. Цей модуль автоматично завантажує HTML-код сторінки за вказаним URL і, за допомогою алгоритму Readability, виділяє основний текст статті, очищаючи його від реклами та непотрібних елементів.

Крім того, у програмі реалізовано фільтрацію за джерелами: користувач може вибрати одне або кілька RSS-джерел зі списку для перегляду новин лише з них. Це дозволяє гнучко керувати потоком інформації.

Після завершення роботи користувач може оновити список новин, переглянути повну статтю, або додати нове джерело для подальшої роботи. Завдяки інтуїтивному інтерфейсу та оптимізованому бекенду, всі дії виконуються швидко, із відповідними повідомленнями про статус виконання запитів.

4.3 Висновки

У четвертому розділі було проведено повне та всебічне тестування розробленої вебсистеми збору та аналізу новин. Перевірка охопила всі основні функціональні модулі застосунку, включаючи пошук новин за ключовими словами, додавання нових RSS-джерел через відповідний інтерфейс, фільтрацію новин за вибраними джерелами та категоріями, а також перегляд повної інформації про новину з можливістю завантаження повного тексту статті.

Особлива увага приділялася тестуванню поведінки системи у випадках введення некоректних або відсутніх даних. Це дало змогу перевірити стійкість застосунку до типових помилок користувача, некоректних запитів та інших виняткових ситуацій. Система коректно обробляє такі випадки: повідомляє про відсутність результатів пошуку, попереджає про обов'язкові параметри при

додаванні нових джерел, а також запобігає збереженню дублюючих або неприпустимих записів.

Було здійснено як функціональне, так і навантажувальне тестування, що дозволило оцінити ефективність та продуктивність роботи системи при обробці запитів до декількох RSS-стрічок одночасно. Результати показали стабільну роботу застосунку навіть за умови одночасного звернення до великої кількості новинних джерел.

Крім технічних перевірок, проведено оцінку зручності початкової роботи з програмою для нових користувачів. У рамках цього було розроблено інструкцію користувача, яка детально описує типову послідовність дій під час роботи із системою: від запуску програми до отримання результатів пошуку та перегляду новин. Це забезпечує легкість освоєння функціоналу навіть для осіб без спеціальної технічної підготовки.

Усі отримані результати підтверджують працездатність, стабільність та готовність вебзастосунку до реального використання в умовах реального інформаційного середовища. Застосунок виконує всі поставлені задачі відповідно до вимог технічного завдання, демонструючи високу функціональність та адаптивність до потреб кінцевих користувачів.

ВИСНОВКИ

У рамках бакалаврській кваліфікаційній роботі було розроблено програмні модулі вебзастосунку для збору, фільтрації, пошуку та перегляду новин із використанням RSS-джерел. В основу розробки покладено сучасні вебтехнології, а саме JavaScript (Node.js) для серверної частини та Vue.js для інтерфейсу користувача. Результати роботи повністю відповідають поставленій темі та завданням роботи. Реалізація проєкту відповідає методичним вказівкам [22].

Під час виконання проєкту було проведено аналіз предметної області, зокрема досліджено існуючі рішення для агрегації новин, їхні переваги та недоліки. На основі цього аналізу сформульовано функціональні вимоги до системи, а також обґрунтовано вибір інструментів розробки.

У ході роботи над бакалаврської кваліфікаційної роботи було:

– сформульовано та реалізовано всі основні функціональні вимоги до системи: підтримка роботи з RSS-джерелами, можливість пошуку новин за ключовими словами, фільтрація результатів за категоріями та джерелами, а також перегляд повної інформації про новину. Під час тестування підтверджено відповідність функціоналу сформульованим вимогам;

– реалізовано та успішно протестовано алгоритми завантаження RSS-стрічок, їх обробки, парсингу контенту, пошуку за ключовими словами та формування списку новин. Перевірка показала коректну роботу алгоритмів як при наявності валідних, так і некоректних або відсутніх даних;

– у системі працює модуль додавання RSS-джерел, який дозволяє користувачу самостійно розширювати перелік джерел без змін коду. В ході тестування перевірено обробку коректних та некоректних даних, валідацію введених параметрів та збереження змін у локальному файлі. Функціонал показав стабільну та передбачувану роботу;

– у системі реалізовано модуль фільтрації новин за вибраними категоріями та джерелами. Під час тестування підтверджено правильну роботу фільтра,

динамічне оновлення відображуваних новин та можливість комбінувати параметри для більш гнучкого пошуку інформації;

– проведено функціональне, навантажувальне та стрес-тестування системи. Перевірка показала, що програма коректно реагує на помилкові введення, відсутність даних та інші виняткові ситуації, забезпечуючи стабільність роботи. Окрім цього, створено інструкцію користувача, яка дозволяє швидко розібратися у використанні системи навіть новачкові, що свідчить про достатній рівень зручності застосунку.

У процесі розробки були побудовані UML-діаграми системи, розроблено структуру модуля джерел новин, а також забезпечено збереження та обробку даних у форматі JSON. Особлива увага приділялась перевірці системи на стійкість до помилок, зокрема при відсутності підключення до джерела або введенні некоректних параметрів пошуку.

У завершальному етапі було проведено тестування програмного продукту, яке підтвердило його працездатність та відповідність технічному завданню. Також розроблено інструкцію для користувача, що описує типову взаємодію із системою.

Таким чином, поставлені цілі та завдання бакалаврської кваліфікаційної роботи досягнуті, а розроблений застосунок може бути використаний як основа для подальшого розвитку систем моніторингу та аналізу інформаційних потоків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке RSS і як його використовувати? [Електронний ресурс] – Режим доступу до ресурсу: https://www.ukraine.com.ua/blog/marketing/chto-takoe-rss-i-kak-ego-ispolzovat.html (дата звернення 14.06.2025).

2. Ліщинська Л. Б., Бондар В.О. Методи збору та аналізу інформації з веб-джерел / Молодь в науці: дослідження, проблеми, перспективи (МН-2025) / Уклад. Л. Б. Ліщинська, В.О. Бондар [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25673 (дата звернення: 14.06.2025).

3. Vue.js – навіщо він нам? [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@jstify.community/vue-js-навіщо-він-нам-981b5e17af39 (дата звернення 14.06.2025).

4. Що таке Node JS простими словами. [Електронний ресурс] – Режим доступу до ресурсу: https://dan-it.com.ua/uk/blog/chto-jeto-takoe-node-js-prostymi-slovami/ (дата звернення 14.06.2025).

5. Створення веб-додатків на EXPRESS.JS. [Електронний ресурс] – Режим доступу до ресурсу: https://brander.ua/technologies/expressjs (дата звернення 25.04.2025).

6. Axios. [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Аксіос (дата звернення 14.06.2025).

7. Google News. [Електронний ресурс] – Режим доступу до ресурсу: https://news.google.com/home?hl=ua&gl=UA&ceid=UA (дата звернення 25.04.2025).

8. Feedly. [Електронний ресурс] – Режим доступу до ресурсу:: https://feedly.com (дата звернення 14.06.2025).

9. Flipboard. [Електронний ресурс] – Режим доступу до ресурсу:: https://about.flipboard.com (дата звернення 14.06.2025).

10. Particle. [Електронний ресурс] – Режим доступу до ресурсу: https://www.particle.io (дата звернення 14.06.2025).

11. SilkData News Analyzer. [Електронний ресурс] – Режим доступу до ресурсу: https://silkdata.tech (дата звернення 15.06.2025).

12. Що таке файл XML. [Електронний ресурс] – Режим доступу до ресурсу: https://horoshop.ua/ua/blog/xml-file/ (дата звернення 15.06.2025).

13. Що таке html?. [Електронний ресурс] – Режим доступу до ресурсу: https://css.in.ua/article/shcho-take-css_3 (дата звернення 15.06.2025).

14. Що таке обробка природної мови (NLP) та як вона може використовуватися у бізнесі. [Електронний ресурс] – Режим доступу до ресурсу: https://metinvest.digital/ua/page/1052 (дата звернення 15.06.2025).

15. Алгоритм TF-IDF і визначення релевантності тексту. [Електронний ресурс] – Режим доступу до ресурсу: https://expans.ua/blog/algorytm-tf-idf-i-vyznachennya-relevantnosti-tekstu/ (дата звернення 15.06.2025).

16. KMeans. [Електронний ресурс] – Режим доступу до ресурсу: https://www.mathworks.com/help/stats/kmeans.html (дата звернення 15.06.2025).

17. Naive Bayes. [Електронний ресурс] – Режим доступу до ресурсу: https://itwiki.dev/data-science/ml-reference/ml-glossary/naive-bayes-models (дата звернення 15.06.2025).

18. SVM. [Електронний ресурс] – Режим доступу до ресурсу: https://itwiki.dev/data-science/ml-reference/ml-glossary/support-vector-machines (дата звернення 15.06.2025).

19. REST API як спосіб спілкування компонент веб-додатків. [Електронний ресурс] – Режим доступу до ресурсу: https://foxminded.ua/shcho-take-rest-api/ (дата звернення 15.06.2025).

20. Що таке JSON. Усе про цей формат передачі даних в інтернеті. [Електронний ресурс] – Режим доступу до ресурсу: https://apix-drive.com/ua/blog/useful/scho-take-json (дата звернення 15.06.2025).

21. JSDOM. [Електронний ресурс] – Режим доступу до ресурсу: https://javascript.org.ua/shho-take-jsdom/ (дата звернення 15.06.2025).

22. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення» / уклад. : О. Н. Романюк, Г. О. Черноволик. – Вінниця : ВНТУ, 2022. – 51 с.

ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
"28" 03 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка системи збору та аналізу новин із вебджерел» за спеціальністю
121 Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
д.т.н., проф. каф. ПЗ Л.Б. Ліщинська
"24" 03 2025 р.
Виконав:
студент гр.2ПІ-21Б В.О. Бондар
"24" 03 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка системи збору та аналізу новин із вебджерел».

Галузь застосування - інформаційні системи, автоматизація збору даних, веб-технології, журналістика даних.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 ректора по ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки.

Метою бакалаврської кваліфікаційної роботи є покращення ефективності отримання актуальної інформації шляхом розробки вебзастосунку для автоматичного збору, фільтрації та візуалізації новин із різних джерел.

Призначення роботи – забезпечення користувача інструментом для автоматизованого моніторингу інформаційного простору та аналізу новинного контенту.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР.

1. Jurafsky, D., & Martin, J. H. (2020). Speech and Language Processing. — Pearson.

2. Russell, M. A. (2013). Mining the Social Web. — O’Reilly Media.

3. Mitchell, R. (2018). Web Scraping with Python: Collecting Data from the Modern Web. — O’Reilly Media.

5. Технічні вимоги

Платформа: вебзастосунок; Джерела: підтримка RSS та HTML-сторінок;

База даних: зберігання новин, джерел, категорій; Основні функції: додавання та керування джерелами новин; фільтрація за категоріями, датами, ключовими словами; інтерфейс для перегляду й сортування новин;

6. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до БКР;
– технічне завдання;
– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз предметної області та формування вимог до системи	25.03.25 – 05.04.25
2	Розробка архітектури та вибір інструментів реалізації	06.04.25 – 15.04.25
3	Реалізація модуля збору новин з веб-джерел	16.04.25 – 23.04.25
4	Розробка модуля фільтрації, категоризації та інтерфейсу перегляду	24.04.25 – 10.05.25
5	Тестування програми	10.05.25 – 20.05.25
6	Оформлення матеріалів до захисту БКР	20.05.25 - 30.05.25

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Захист бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

65

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка системи збору та аналізу новин із вебджерел

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ: кафедра програмного забезпечення, ФІТКІ, 2ПІ-21Б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 7.4 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Ліщинська Л. Б., д.т.н., проф. каф. ПЗ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Бондар В.О.
(підпис) (прізвище, ініціали)

Додаток В – Лістинг програми

```
<template>
  <div class="container">
    <h1 class="main-title">Система збору та аналізу новин</h1>

    <!-- Верхня панель: фільтр, пошук, додавання джерел -->
    <div v-if="!selectedArticle" class="top-panel">
      <!-- Ліва частина: додавання джерела -->
      <div class="left-panel">
        <h3>Додати джерело</h3>
        <input v-model="newSource.name" placeholder="Назва сайту" />
        <input v-model="newSource.url" placeholder="URL RSS" />
        <button @click="addSource">Додати</button>
      </div>

      <!-- Центральна частина: пошук -->
      <div class="center-panel">
        <h3>Пошук</h3>
        <input
            v-model="query"
            @keyup.enter="search"
            placeholder="Введіть запит..."
        />
        <button @click="search">Пошук</button>
      </div>

      <!-- Права частина: категорії -->
      <div class="right-panel">
        <h3>Фільтрація</h3>
        <label>Категорія:
          <select v-model="selectedCategory">
            <option value="">Всі</option>
            <option value="політика">Політика</option>
            <option value="технології">Технології</option>
            <option value="економіка">Економіка</option>
            <option value="спорт">Спорт</option>
          </select>
        </label>
        <div class="source-dropdown" v-if="!selectedArticle">
          <h4>Джерела:</h4>
          <select id="source-select" v-model="selectedSources" class="select-multiple">
            <option value="null">Немає</option>
            <option v-for="source in sources" :key="source.name" :value="source.name">
              {{ source.name }}
            </option>
          </select>
        </div>
      </div>
    </div>
```

```
    <!-- Список новин -->
    <div v-if="!selectedArticle">
      <div v-for="article in results" :key="article.url" class="news-card">
        <h3 class="news-title">{{ article.title }}</h3>
        <p class="news-description">{{ article.description }}</p>
        <p class="news-source">
          Джерело:
          <a :href="article.url" target="_blank">{{ article.source }}</a>
        </p>
        <button @click="openArticle(article)" class="btn-view">Переглянути</button>
      </div>
    </div>

    <!-- Повна новина -->
    <div v-else>
      <button @click="goBack" class="btn-back">← Назад до результатів</button>
      <h2 class="news-title">{{ selectedArticle.title }}</h2>
      <p class="news-source">Джерело: {{ selectedArticle.source }}</p>
      <div v-if="loading" class="loading">Завантаження...</div>
      <div v-else v-html="fullHtml" class="article-content"></div>
    </div>
  </div>
</template>

<script setup>
import { ref, onMounted } from 'vue'

const query = ref('')
const results = ref([])
const selectedArticle = ref(null)
const fullHtml = ref('')
const loading = ref(false)
const sources = ref([])
const selectedSources = ref([])
const selectedCategory = ref('')
const newSource = ref({ name: '', url: '' })

const loadSources = async () => {
  const res = await fetch('http://localhost:3000/sources')
  sources.value = await res.json()
}

onMounted(() => {
  loadSources()
})

const search = async () => {
  selectedArticle.value = null

  const params = new URLSearchParams()
  params.append('q', query.value)
```

```
  selectedSources.value.forEach(s => params.append('sources', s))

  const res = await fetch(`http://localhost:3000/search?${params.toString()}`)
  results.value = await res.json()
}

const openArticle = async (article) => {
  selectedArticle.value = article
  loading.value = true
  fullHtml.value = ''
  try {
    const res = await fetch(`http://localhost:3000/article?url=${encodeURIComponent(article.url)}`)
    const data = await res.json()
    fullHtml.value = data.html
  } catch (err) {
    fullHtml.value = '<p>Не вдалося завантажити текст статті.</p>'
  } finally {
    loading.value = false
  }
}

const goBack = () => {
  selectedArticle.value = null
}

const addSource = async () => {
  if (!newSource.value.name || !newSource.value.url) return alert('Заповніть обидва поля')
  await fetch('http://localhost:3000/sources', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(newSource.value)
  })
  newSource.value = { name: '', url: '' }
  alert('Джерело додано')
}
</script>

<style>
.container {
  max-width: 1000px;
  margin: auto;
  padding: 20px;
  font-family: sans-serif;
  color: white;
  min-height: 100vh;
}

.main-title {
  font-size: 28px;
  font-weight: bold;
  margin-bottom: 30px;
  text-align: center;
```

```
}

.top-panel {
  display: flex;
  align-items: flex-start;
  justify-content: space-between;
  gap: 30px;
  margin-bottom: 30px;
  flex-wrap: wrap;
}

.left-panel, .center-panel, .right-panel {
  flex: 1;
  min-width: 220px;
  background: #1e1e1e;
  padding: 16px;
  border-radius: 8px;
  box-shadow: 0 0 8px rgba(0, 0, 0, 0.2);
}

input, select, select-multiple {
  width: 90%;
  padding: 8px;
  margin-bottom: 10px;
  font-size: 14px;
}

button {
  padding: 8px 12px;
  font-size: 14px;
  background-color: #388e3c;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

button:hover {
  background-color: #2e7d32;
}

.news-card {
  border: 1px solid #333;
  padding: 16px;
  margin-bottom: 15px;
  border-radius: 4px;
  background-color: #242424;
}

.news-title {
  font-size: 18px;
  font-weight: bold;
```

```
}

.news-description {
  margin: 10px 0;
}

.news-source {
  font-size: 14px;
  color: #b0bec5;
}

.btn-view, .btn-back {
  margin-top: 10px;
  background-color: #1976d2;
}

.btn-view:hover, .btn-back:hover {
  background-color: #1565c0;
}

.loading {
  font-style: italic;
  color: gray;
  margin-top: 20px;
}

.article-content {
  margin-top: 20px;
  line-height: 1.6;
  text-align: left;
}

.article-content img {
  max-width: 100%;
  height: auto;
  border: 1px solid #ccc;
  border-radius: 6px;
  margin: 10px 0;
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);
}

.article-content p {
  margin-bottom: 1em;
}

.article-content h1,
.article-content h2,
.article-content h3 {
  margin-top: 1.2em;
  margin-bottom: 0.5em;
}
```

```
.article-content ul,
.article-content ol {
  padding-left: 20px;
  margin-bottom: 1em;
}
.sources-block {
  border: 1px solid #ccc;
  padding: 12px;
  margin-bottom: 16px;
  background-color: #1e1e1e;
  border-radius: 6px;
}
.sources-block h3 {
  margin-bottom: 8px;
}
.sources-block label {
  display: block;
  margin-bottom: 6px;
}
h4{
  margin-block-end: 2px;
}
</style>

// index.js
const express = require('express')
const cors = require('cors')
const Parser = require('rss-parser')
const { JSDOM } = require('jsdom')
const { Readability } = require('@mozilla/readability')

const app = express()
const port = 3000
const parser = new Parser()
const fs = require('fs')
const path = require('path')

const axios = require('axios')
const cheerio = require('cheerio')

app.use(cors())

const sourcesPath = path.join(__dirname, 'sources.json')
function loadSources() {
   return JSON.parse(fs.readFileSync(sourcesPath, 'utf-8'))
}

function saveSources(sources) {
   fs.writeFileSync(sourcesPath, JSON.stringify(sources, null, 2))
}

app.get('/article', async (req, res) => {
```

```
    const url = req.query.url
    try {
        const response = await fetch(url)
        const html = await response.text()
        const dom = new JSDOM(html, { url })
        const reader = new Readability(dom.window.document)
        const article = reader.parse()
        res.json({ html: article?.content || '<p>Не вдалося витягти текст статті.</p>' })
    } catch (e) {
        console.error(e)
        res.status(500).json({ html: '<p>Помилка при завантаженні статті</p>' })
    }
})

app.use(express.json()) // для body в POST-запитах

const sourcesFile = path.join(__dirname, 'sources.json')

// Якщо немає файлу – створити
if (!fs.existsSync(sourcesFile)) {
    fs.writeFileSync(sourcesFile, JSON.stringify([
        { name: 'Ukr.net', url: 'https://www.ukr.net/rss/news-ukrnet.rss' },
        { name: 'TSN', url: 'https://tsn.ua/rss' }
    ], null, 2))
}

app.get('/sources', (req, res) => {
    res.json(loadSources())
})

app.post('/sources', (req, res) => {
    const { name, url } = req.body
    if (!name || !url) return res.status(400).json({ error: 'name і url обов’язкові' })

    const sources = loadSources()
    sources.push({ name, url })
    saveSources(sources)

    res.json({ message: 'Джерело додано' })
})

// Список RSS-джерел (можна згодом зробити динамічним)
const sources = [
    { name: 'Ukr.net', url: 'https://www.ukr.net/rss/news-ukrnet.rss' },
    { name: 'TSN', url: 'https://tsn.ua/rss' },
    { name: 'BBC', url: 'https://feeds.bbci.co.uk/news/rss.xml' },
    { name: 'CNN', url: 'http://rss.cnn.com/rss/edition.rss' },
    { name: 'TechCrunch', url: 'https://techcrunch.com/feed/' },
    { name: 'Економічна правда', url: 'https://www.epravda.com.ua/rss/' },
    { name: 'УП', url: 'https://www.pravda.com.ua/rss/' },
]
```

```
app.get('/search', async (req, res) => {
  const query = req.query.q?.toLowerCase() || ''
  const selectedSources = req.query.sources // може бути масивом або рядком

  const allSources = JSON.parse(fs.readFileSync(sourcesPath))
  let sourcesToSearch = allSources

  // Якщо передані джерела – фільтруємо
  if (selectedSources) {
    const selectedArray = Array.isArray(selectedSources)
      ? selectedSources
      : [selectedSources]
    sourcesToSearch = allSources.filter(src => selectedArray.includes(src.name))
  }

  const results = []

  for (const source of sourcesToSearch) {
    try {
      const feed = await parser.parseURL(source.url)
      feed.items.forEach(item => {
        const title = item.title || ''
        const description = item.contentSnippet || ''
        if (title.toLowerCase().includes(query) || description.toLowerCase().includes(query)) {
          results.push({
            title,
            description,
            url: item.link,
            source: source.name
          })
        }
      })
    } catch (err) {
      console.error(`Помилка читання ${source.name}:`, err.message)
    }
  }
  res.json(results)
})

app.get('/analytics', async (req, res) => {
  const query = req.query.q?.toLowerCase() || ''
  const sources = JSON.parse(fs.readFileSync(sourcesPath))
  const stats = {}

  for (const source of sources) {
    try {
      const feed = await parser.parseURL(source.url)
      const matched = feed.items.filter(item =>
        (item.title || '').toLowerCase().includes(query) ||
        (item.contentSnippet || '').toLowerCase().includes(query)
      )
      stats[source.name] = matched.length
```

```
        } catch (err) {
            console.error(`Помилка читання ${source.name}:`, err.message)
            stats[source.name] = 'error'
        }
    }

    res.json(stats)
})


app.get('/history', (req, res) => {
    if (!fs.existsSync(historyFile)) return res.json([])
    const history = JSON.parse(fs.readFileSync(historyFile))
    res.json(history)
})

app.get('/latest', async (req, res) => {
    const sources = loadSources()
    const results = []
    const seen = new Set()

    for (const source of sources) {
        try {
            const feed = await parser.parseURL(source.url)
            feed.items.forEach(item => {
                const title = item.title || ''
                const description = item.contentSnippet || ''
                const key = hashText(title + description)
                if (seen.has(key)) return
                seen.add(key)

                results.push({
                    title,
                    description,
                    url: item.link,
                    source: source.name,
                    date: item.pubDate || null
                })
            })
        } catch (err) {
            console.error(`Помилка джерела ${source.name}:`, err.message)
        }
    }

    res.json(results.slice(0, 50))
});


app.get('/favorites', (req, res) => {
    if (!fs.existsSync(favoritesFile)) return res.json([])
    const data = JSON.parse(fs.readFileSync(favoritesFile))
    res.json(data)
```

```
})

app.post('/favorites', (req, res) => {
  const { title, url, source, description } = req.body
  if (!url) return res.status(400).json({ error: 'url обов’язковий' })
  const list = fs.existsSync(favoritesFile) ? JSON.parse(fs.readFileSync(favoritesFile)) : []
  if (!list.find(f => f.url === url)) list.push({ title, url, source, description })
  fs.writeFileSync(favoritesFile, JSON.stringify(list, null, 2))
  res.json({ success: true })
})

app.delete('/favorites', (req, res) => {
  const { url } = req.query
  if (!url) return res.status(400).json({ error: 'url обов’язковий' })
  const list = JSON.parse(fs.readFileSync(favoritesFile))
  const updated = list.filter(f => f.url !== url)
  fs.writeFileSync(favoritesFile, JSON.stringify(updated, null, 2))
  res.json({ success: true })
})

app.get('/export/csv', async (req, res) => {
  const query = req.query.q?.toLowerCase() || ''
  const sources = loadSources()
  let results = []

  for (const source of sources) {
    try {
      const feed = await parser.parseURL(source.url)
      feed.items.forEach(item => {
        const title = item.title || ''
        const description = item.contentSnippet || ''
        if (title.toLowerCase().includes(query) || description.toLowerCase().includes(query)) {
          results.push({ title, description, url: item.link, source: source.name })
        }
      })
    } catch {}
  }

  const csv = ['"Title","Description","URL","Source"']
  results.forEach(r => {
    csv.push(`"${r.title.replace(/"/g, '""')}","${r.description.replace(/"/g,
'""')}","${r.url}","${r.source}"`)
  })

  res.setHeader('Content-Type', 'text/csv')
  res.setHeader('Content-Disposition', 'attachment; filename="news_export.csv"')
  res.send(csv.join('\n'))
})

app.get('/recommendations', async (req, res) => {
  if (!fs.existsSync(historyFile)) return res.json([])
```

```
    const history = JSON.parse(fs.readFileSync(historyFile))
    const recentQueries = [...new Set(history.map(h => h.query))].slice(-5)
    const allSources = loadSources()
    const results = []
    const seenLinks = new Set()

    for (const query of recentQueries) {
      for (const source of allSources) {
        try {
          const feed = await parser.parseURL(source.url)
          feed.items.forEach(item => {
            const text = `${item.title} ${item.contentSnippet}`.toLowerCase()
            if (
              text.includes(query.toLowerCase()) &&
              !seenLinks.has(item.link)
            ) {
              results.push({
                title: item.title,
                description: item.contentSnippet,
                url: item.link,
                source: source.name
              })
              seenLinks.add(item.link)
            }
          })
        } catch {}
      }
    }

    res.json(results)
})

app.get('/word-frequency-by-source', async (req, res) => {
    const sources = loadSources()
    const result = {}

    for (const source of sources) {
      const freq = {}
      try {
        const feed = await parser.parseURL(source.url)
        feed.items.forEach(item => {
          const text = `${item.title} ${item.contentSnippet}`.toLowerCase()
          text.split(/\W+/).forEach(word => {
            if (word.length > 3) {
              freq[word] = (freq[word] || 0) + 1
            }
          })
        })
        result[source.name] = Object.entries(freq)
          .sort((a, b) => b[1] - a[1])
          .slice(0, 15)
          .reduce((acc, [w, c]) => ({ ...acc, [w]: c }), {})
```

```
      } catch (err) {
        result[source.name] = { error: true }
      }
    }

    res.json(result)
})




app.listen(port, () => {
    console.log(`Сервер працює на http://localhost:${port}`)
})

{
  "name": "news-backend",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "axios": "^1.9.0",
    "cheerio": "^1.0.0",
    "cors": "^2.8.5",
    "express": "^5.1.0",
    "rss-parser": "^3.13.0"
  }
}



<template>
  <div>
    <h2 class="text-xl font-bold mb-4">Результати пошуку</h2>
    <div v-if="articles.length === 0">Немає результатів</div>
    <div v-for="article in articles" :key="article.url" class="mb-4 border p-4 rounded shadow">
      <h3 class="text-lg font-semibold">{{ article.title }}</h3>
      <p class="text-gray-700 my-2">{{ article.description }}</p>
      <p class="text-sm text-gray-500">
        Джерело:
        <a :href="article.url" target="_blank" class="text-blue-500 hover:underline">
          {{ article.source }}
        </a>
        <button @click="$emit('open', article)" class="mt-2 px-3 py-1 bg-blue-600 text-white rounded
```

```
hover:bg-blue-700">
        Переглянути
       </button>
     </p>
   </div>
  </div>
</template>

<script setup>
definePropsdefineEmits(['open'])
</script>



<template>
  <div v-if="article" class="fixed inset-0 bg-black bg-opacity-50 flex justify-center items-center z-
50">
   <div class="bg-white p-6 rounded w-11/12 md:w-3/4 lg:w-1/2 max-h-[80vh] overflow-y-auto">
    <h2 class="text-xl font-bold mb-2">{{ article.title }}</h2>
    <p class="text-gray-500 text-sm mb-4">Джерело: {{ article.source }}</p>
    <div v-if="loading" class="text-center">Завантаження...</div>
    <div v-else class="whitespace-pre-wrap text-gray-800">{{ fullText }}</div>
    <button @click="$emit('close')" class="mt-4 px-4 py-2 bg-red-500 text-white rounded
hover:bg-red-600">
      Закрити
    </button>
   </div>
  </div>
</template>

<script setup>
import { onMounted, ref } from 'vue'
const props = definePropsdefineEmits(['close'])

const fullText = ref('')
const loading = ref(true)

onMounted(async () => {
  if (!props.article) return
  try {
   const res = await
fetch(`http://localhost:3000/article?url=${encodeURIComponent(props.article.url)}`)
   const data = await res.json()
   fullText.value = data.text
  } catch (err) {
   fullText.value = 'Не вдалося завантажити текст статті.'
  } finally {
   loading.value = false
  }
```

```
})
</script>

[
  {
    "name": "Ukr.net",
    "url": "https://www.ukr.net/rss/news-ukrnet.rss"
  },
  {
    "name": "TSN",
    "url": "https://tsn.ua/rss"
  },
  {
    "name": "Aninews",
    "url": "https://aninews.in.ua/rss"
  },
  {
    "name": "Unian",
    "url": "https://unian.ua/rss"
  },
  {
    "name": "BBC",
    "url": "https://feeds.bbci.co.uk/news/rss.xml"
  },
  {
    "name": "CNN",
    "url": "http://rss.cnn.com/rss/edition.rss"
  },
  {
    "name": "TechCrunch",
    "url": "https://techcrunch.com/feed/"
  },
  {
    "name": "Економічна правда",
    "url": "https://www.epravda.com.ua/rss/"
  },
  {
    "name": "УП",
    "url": "https://www.pravda.com.ua/rss/"
  },
  {
    "name": "ESPN",
    "url": "https://www.espn.com/espn/rss/news"
  },
  {
    "name": "Цензор.НЕТ",
    "url": "https://censor.net/rss/news.politics"
  },

{
 "name": "Sport.ua",
 "url": "https://www.sport.ua/rss.xml"
```

```
},
{
 "name": "ITCenter (IT-новини)",
 "url": "https://itc.ua/feed/"
},
{
 "name": "TechCrunch",
 "url": "https://techcrunch.com/feed/"
},
{
 "name": "Deutsche Welle Україна",
 "url": "https://rss.dw.com/rdf/rss-uk-all"
},
{
 "name": "The Guardian (World)",
 "url": "https://www.theguardian.com/world/rss"
}
```

```
import { createApp } from 'vue'
import './style.css'
import App from './App.vue'

createApp(App).mount('#app')
```

Додаток Г – Графічна частина

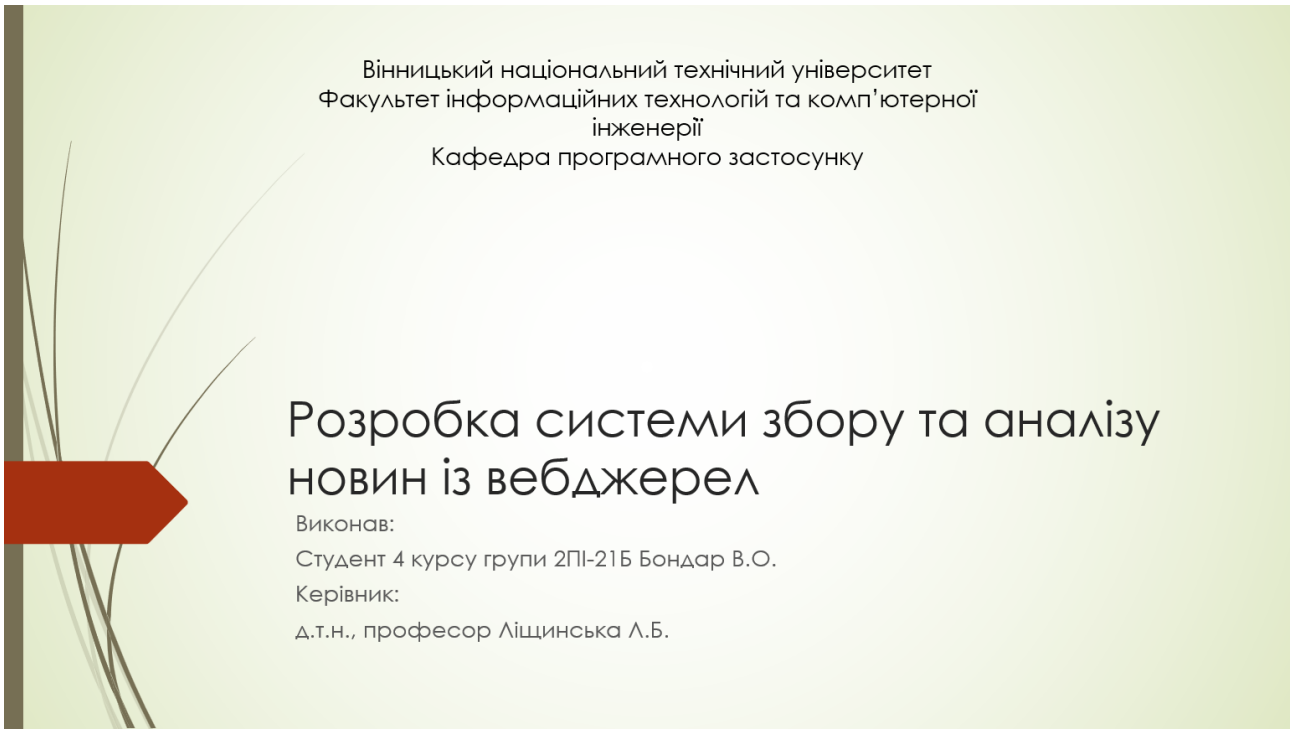

Рисунок Г.1 – Титульний слайд

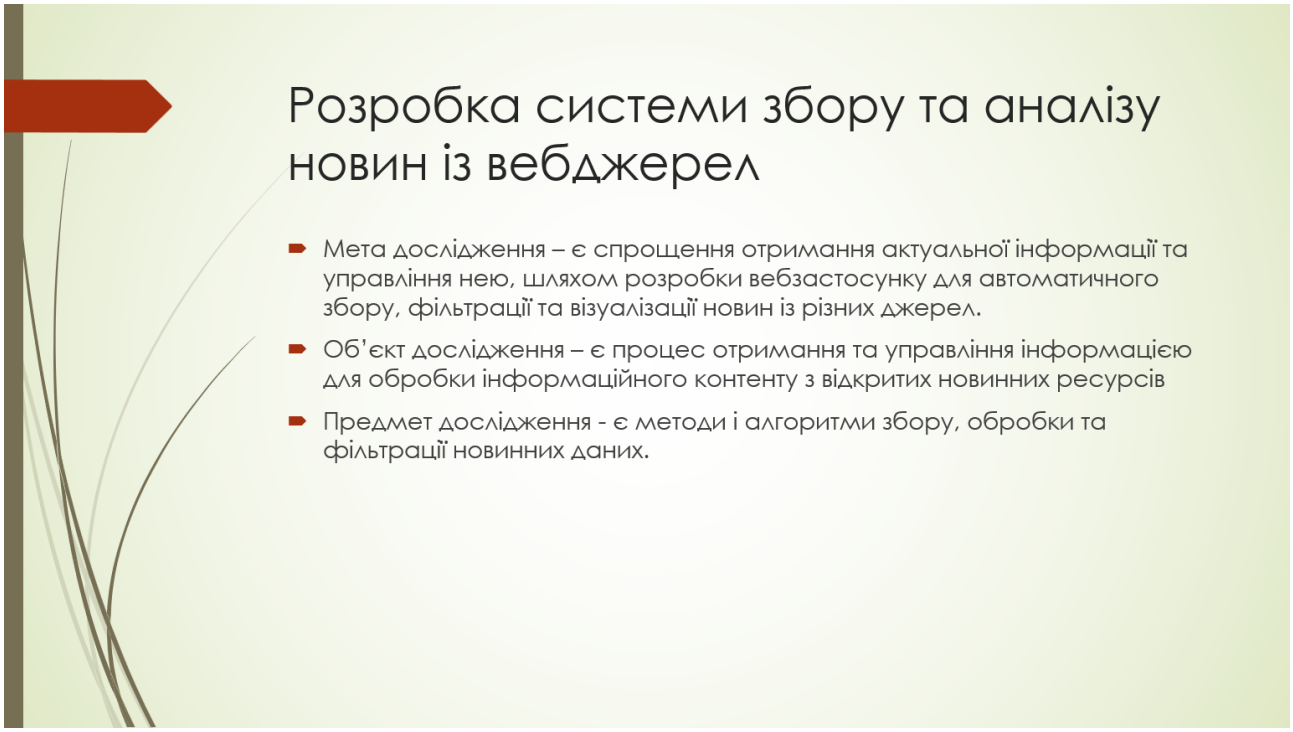

Рисунок Г.2 – Мета, об'єкт і предмет дослідження

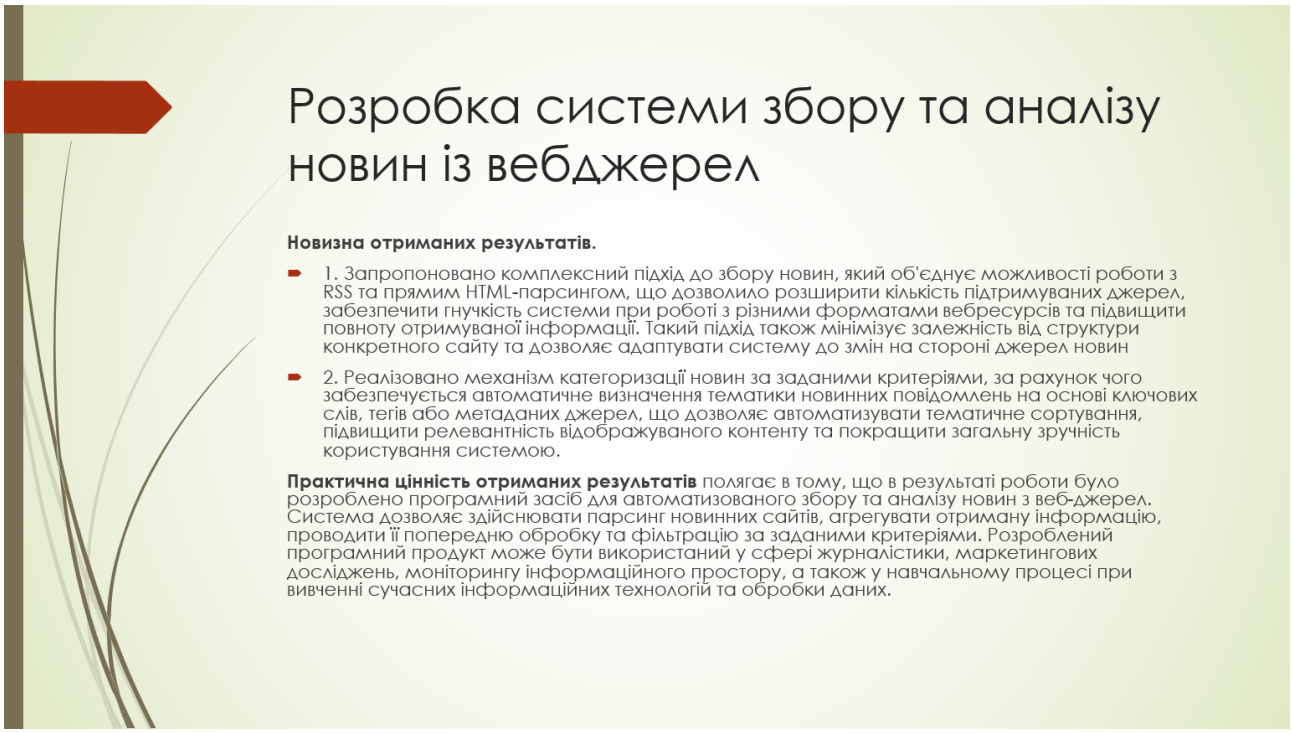

Рисунок Г.3 – Наукова новизна та практична цінність

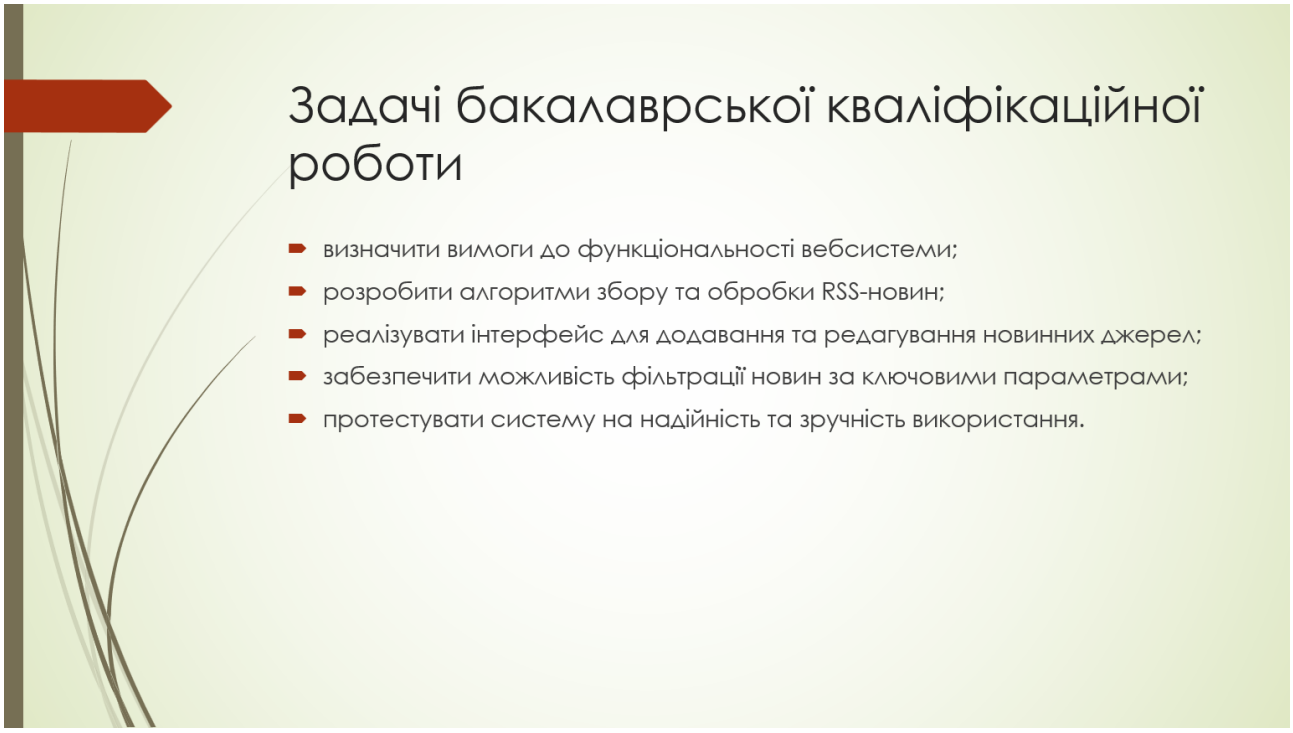

Рисунок Г.4 – Задачі бакалаврської кваліфікаційної роботи

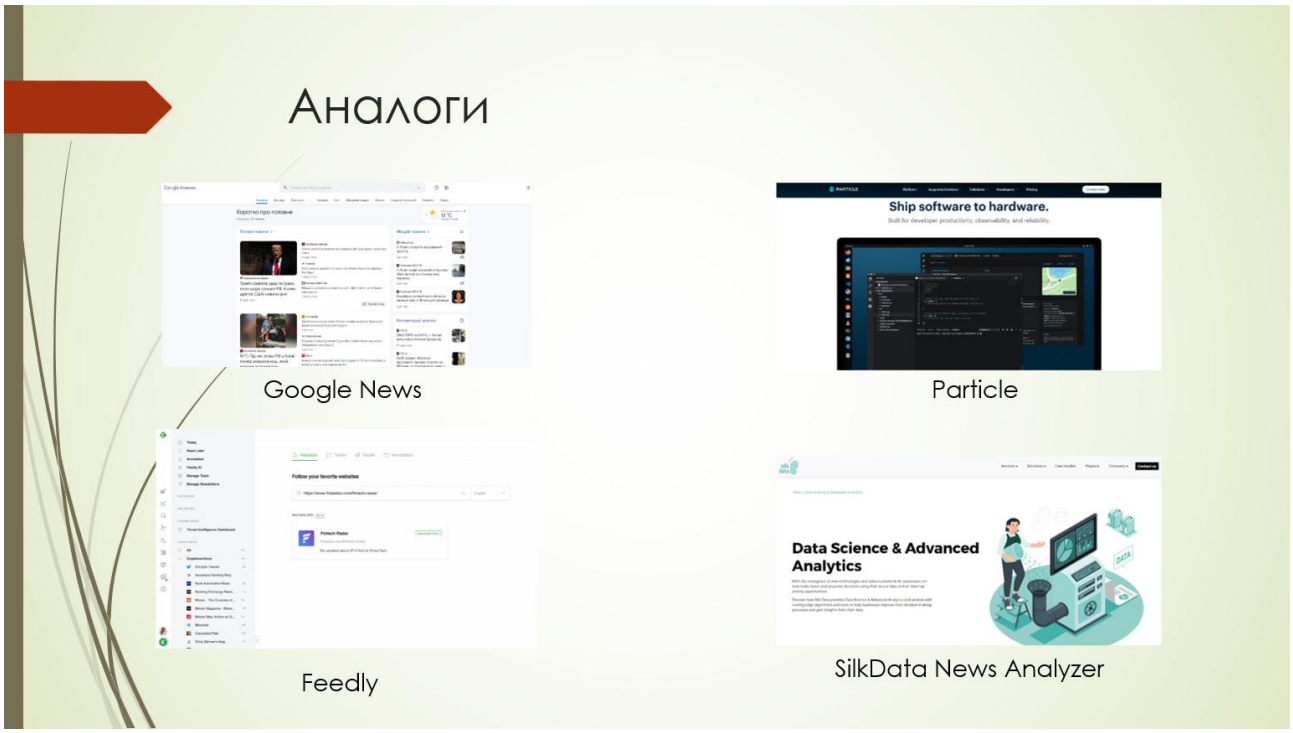

Рисунок Г.5 – Аналоги

Критерій	Google News	Feedly	Flipboard	Particle	SilkData News Analyzer	Власний застосунок
Візуалізація даних	-	-	+	+	+	+
Порівняння новин з різних джерел	-	-	-	+	+	+
Підтримка української мови	+	+	+	-	-	+
Швидкість роботи	+	-	+	+	-	+
Підтримка RSS-джерел	-	+	-	-	-	+

Рисунок Г.6 – Порівняльний аналіз аналогів

Рисунок Г.7 – Діаграма діяльності програмного застосунку

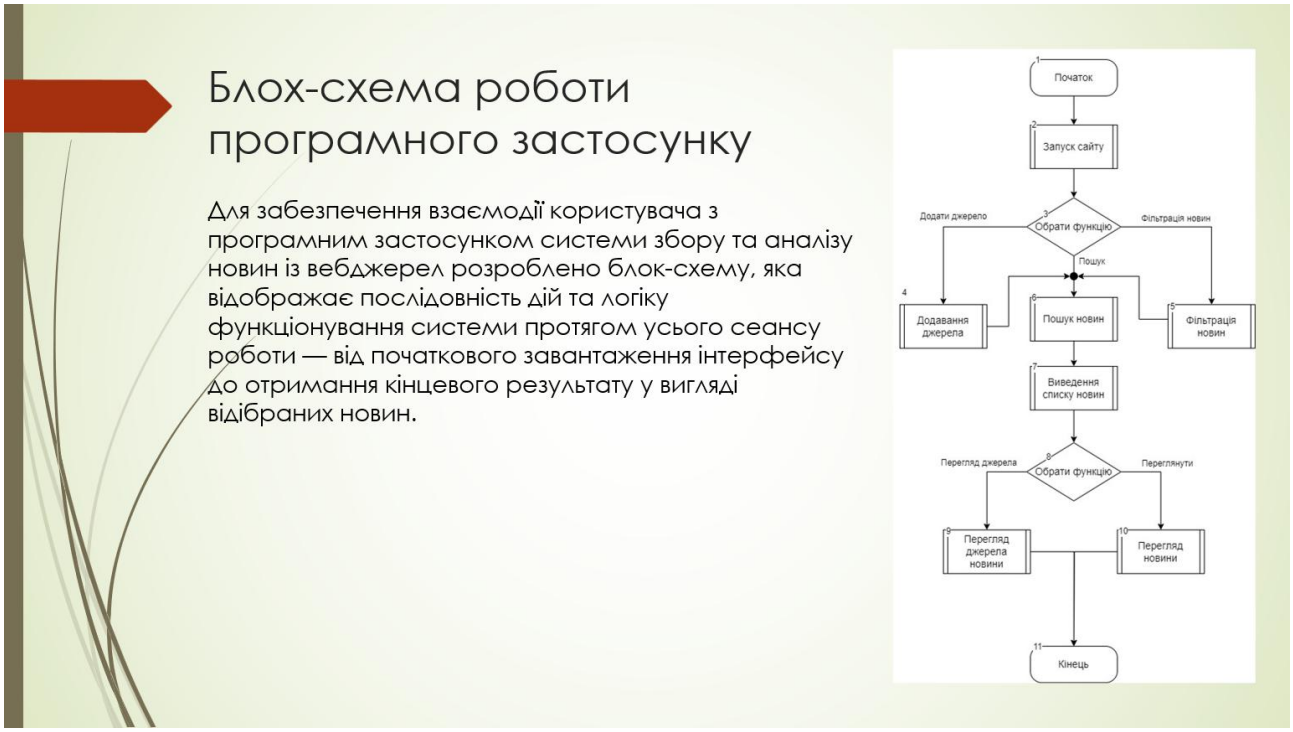

Рисунок Г.8 – Блок-схема роботи програмного застосунку

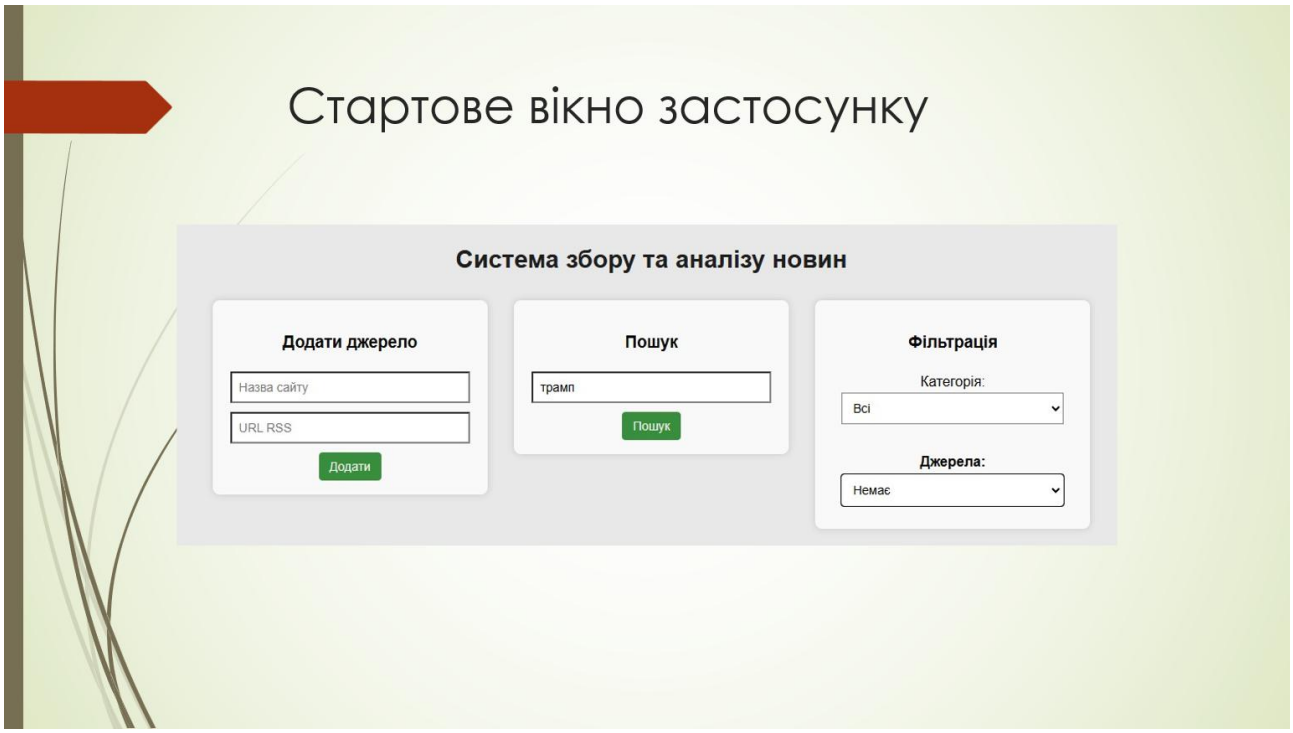

Рисунок Г.9 – Стартове вікно застосунку

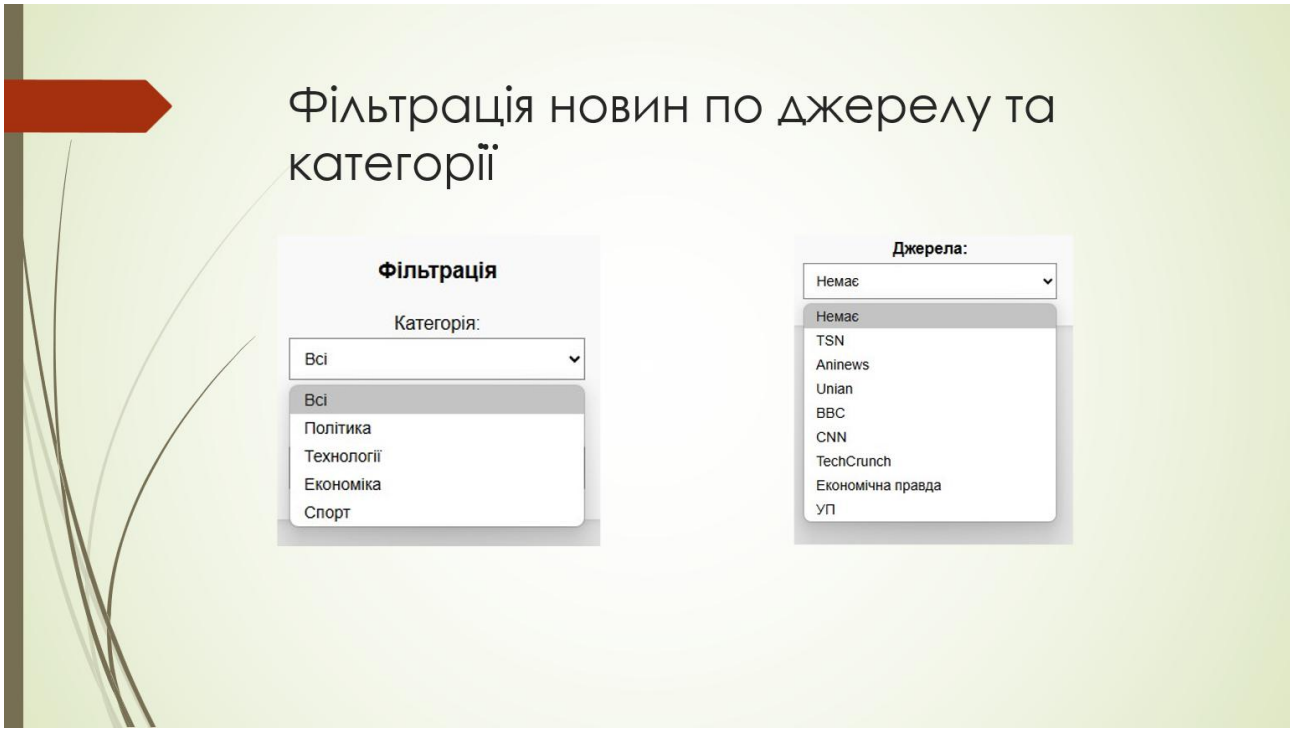

Рисунок Г.10 – Фільтрація новин по джерелу та категорії

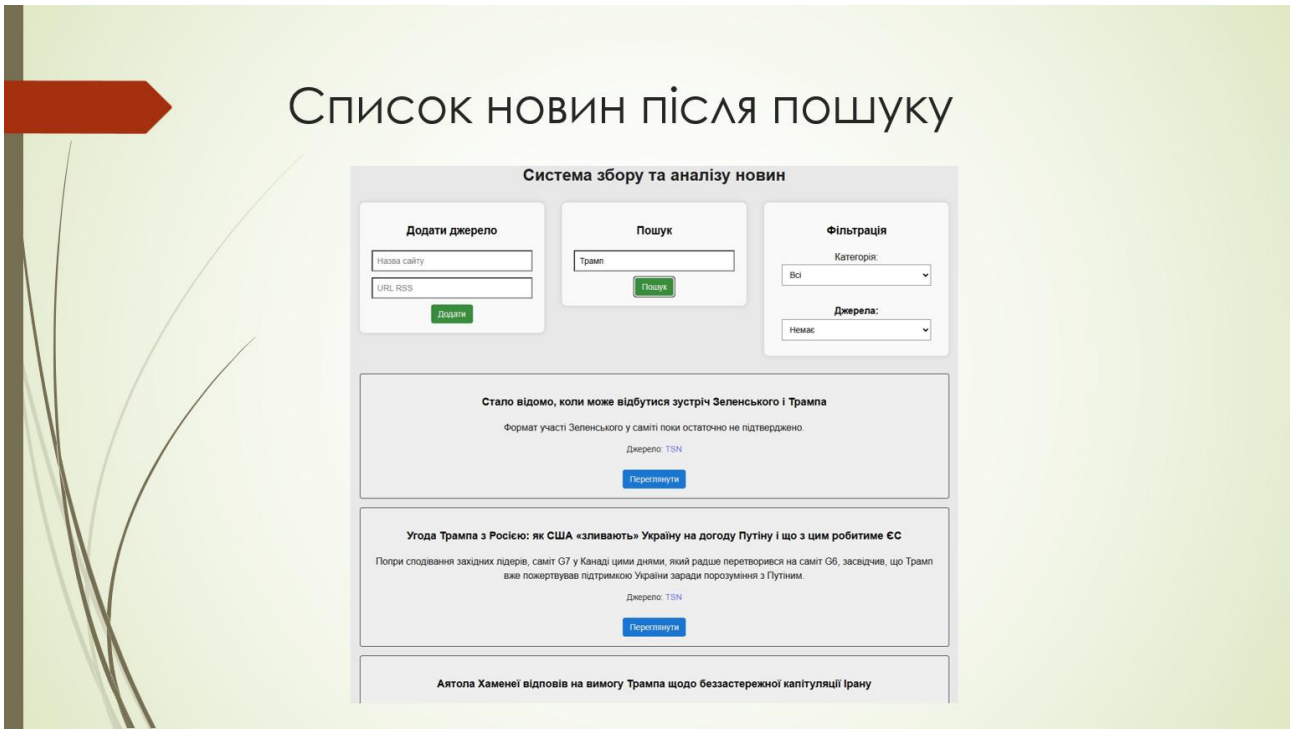

Рисунок Г.11 – Список новин після пошуку

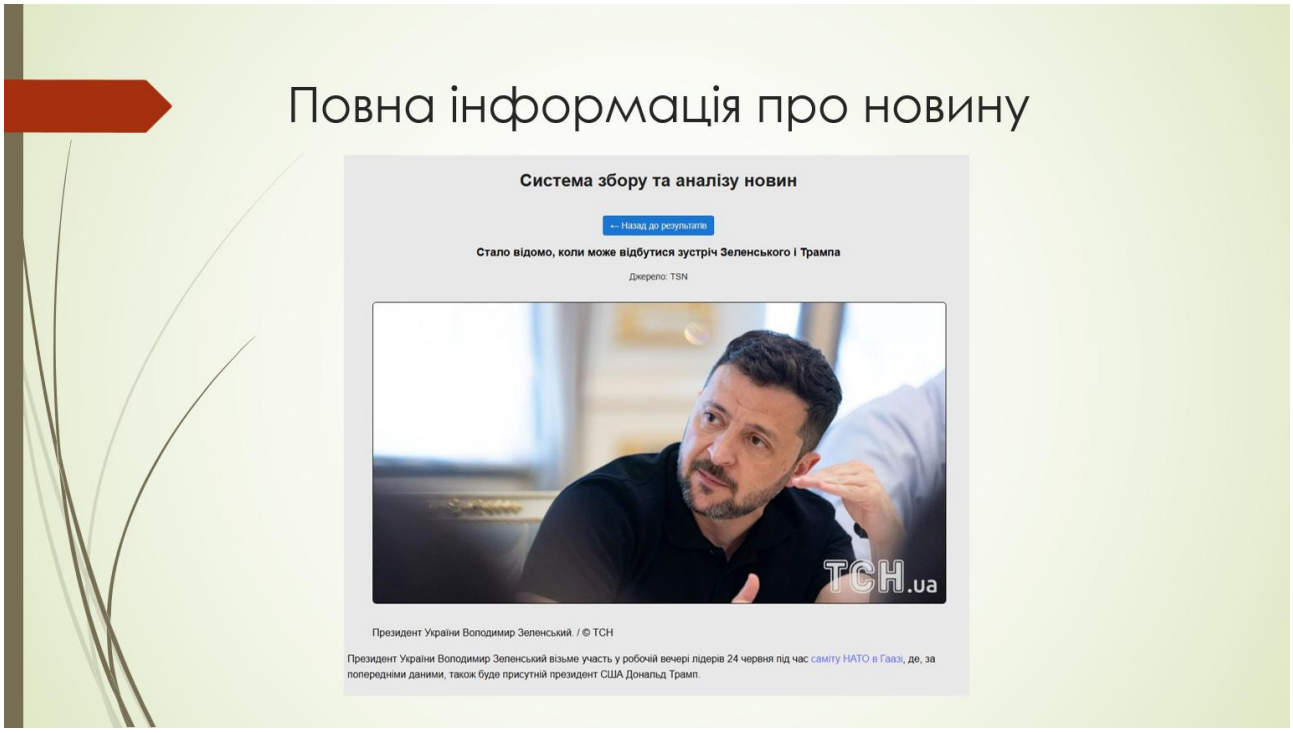

Рисунок Г.12 – Повна інформація про новину

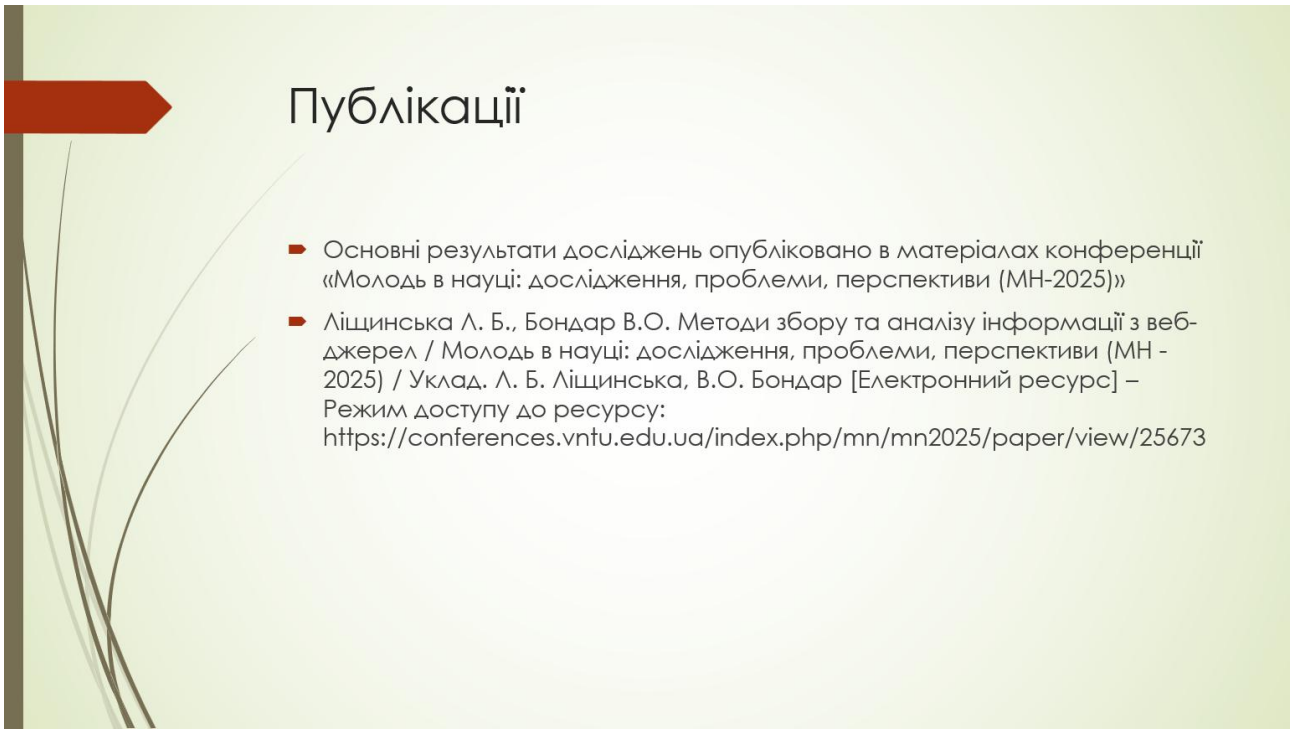

Рисунок Г.13 – Публікації

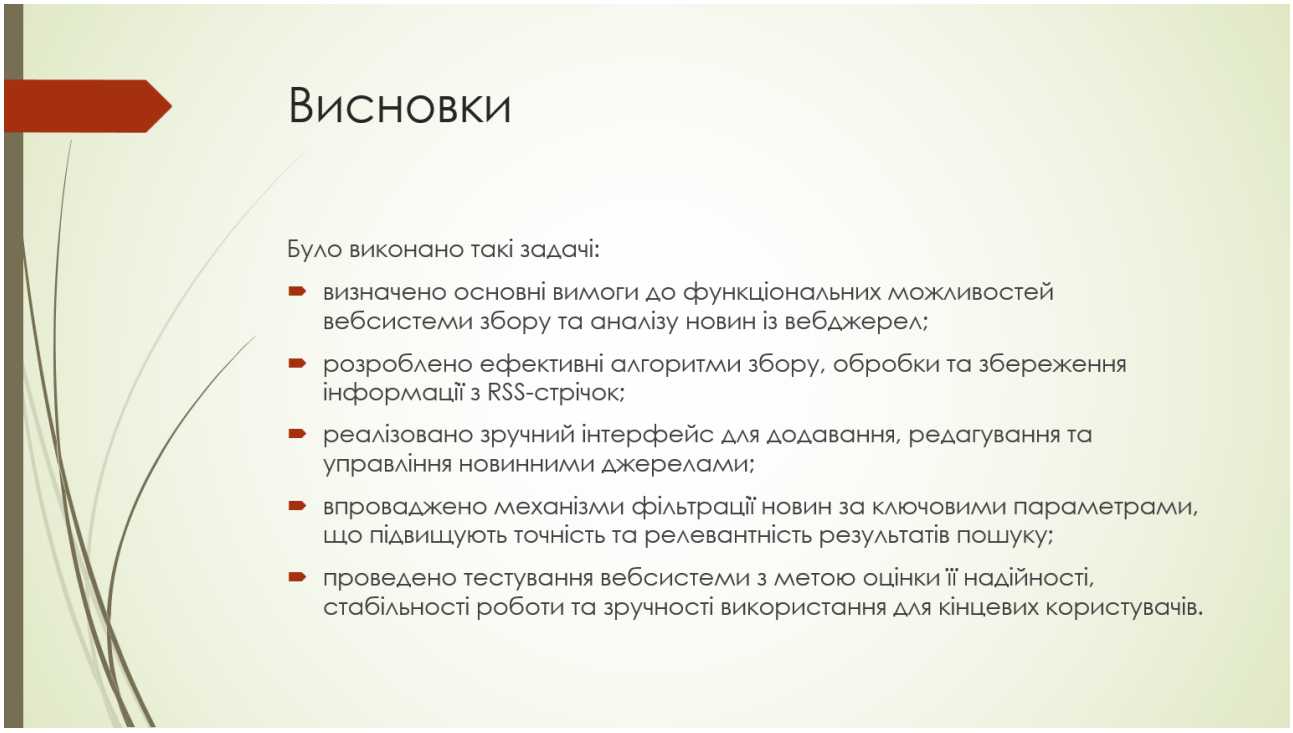

Рисунок Г.14 – Висновки