МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____Юрій КОНДРАТЕНКО

«____»_____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ВЕБЗАСТОСУНОК ДЛЯ АВТОМАТИЗАЦІЇ OSINT У СОЦІАЛЬНИХ МЕДІАРЕСУРСАХ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____ Тарас ЧЕПУРА

«___»_____ 2025 р.

Керівник ст. викладач _____ Іван БУРЛАЧЕНКО

«___»_____ 2025 р.

Миколаїв – 2025

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

Факультет	Комп’ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп’ютерні науки
Освітня програма	Комп’ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Юрій КОНДРАТЕНКО

«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Чепури Тараса Романовича
(прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Вебзастосунок для автоматизації OSINT у соціальних медіаресурсах».

Керівник роботи: Бурлаченко Іван Сергійович, старший викладач кафедри комп'ютерної інженерії.

Затверджена наказом ЧНУ ім. Петра Могили від «18» листопада 2024 р. № 321.

2. Строк представлення кваліфікаційної роботи «___» _червня_ 2025 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: вебзастосунок, розгорнутий у хмарному середовищі AWS, що демонструє здатність автоматично збирати новини з кількох джерел, групувати їх у події та надавати користувачу узагальнену інформацію про ці події.

4. Перелік питань, що підлягають розробці: аналіз предметної області та існуючих рішень; розробка архітектури системи; реалізація модуля збору з попередньою

підготовкою новин та модуля обробки новин за допомогою ШІ; розробка серверної частини (API); розробка клієнтського інтерфейсу.

5. Перелік графічних матеріалів: презентація.

Керівник роботи _____ Іван БУРЛАЧЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Тарас ЧЕПУРА
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «18» грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Вебзастосунок для автоматизації OSINT у соціальних медіаресурсах

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	18.12.2024	20.12.2024	Виконано
2	Аналіз предметної області та постановка задачі	21.12.2024	30.01.2025	Виконано
3	Огляд літературних джерел за темою кваліфікаційної роботи, зокрема публікацій та аналогічних систем автоматизованого OSINT-моніторингу новинних потоків	31.01.2025	01.03.2025	Виконано
4	Дослідження й порівняння сучасних архітектур штучних нейронних мереж для класифікації, кластеризації та сумаризації новин у контексті OSINT-завдань.	02.03.2025	01.04.2025	Виконано
5	Розроблення й тестування прототипу AI-пайплайну та веб-інтерфейсу, з аналізом точності моделей і швидкодії системи на реальному потоці новин.	02.04.2025	15.05.2025	Виконано
6	Перший попередній захист КР на засіданні комісії кафедри	16.05.2025	16.05.2025	Виконано
7	Корегування роботи за результатами попереднього захисту	29.05.2025	29.05.2025	Виконано
8	Другий попередній захист КР на засіданні комісії кафедри	30.05.2025	30.05.2025	Виконано
9	Доробка та остаточне оформлення КР	31.05.2025	10.06.2025	Виконано
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	11.06.2025	11.06.2025	Виконано

Керівник роботи _____ Іван БУРЛАЧЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Тарас ЧЕПУРА
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«29» січня 2025 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувача групи 401 ЧНУ ім. Петра Могили

Чепури Тараса Романовича

на тему: **«ВЕБЗАСТОСУНОК ДЛЯ АВТОМАТИЗАЦІЇ OSINT У СОЦІАЛЬНИХ МЕДІАРЕСУРСАХ»**

Актуальність даної роботи полягає в забезпеченні швидкого аналізу відкритих новинних потоків шляхом розробки вебзастосунку, що інтегрує трансформерні моделі класифікації та кластеризації.

Об'єктом роботи є процеси збору, обробки і аналізу відкритих новинних даних із використанням технологій веб-скрапінгу, обробки природної мови та хмарних обчислень.

Предметом є комбінований метод кластеризації новин у події, алгоритми аналізу тональності і категоризації подій, моделі генерування описів подій.

Метою роботи є забезпечення швидкого аналізу відкритих новинних потоків шляхом розробки вебзастосунку, що інтегрує трансформерні моделі класифікації та кластеризації.

У першому розділі буде проведений аналіз предметної сфери, що охоплює OSINT і його застосування для аналізу даних із соціальних медіаресурсів. Другий розділ буде присвячений опису моделей і методів, які можна застосувати для вирішення задачі автоматизованого збору OSINT. У третьому розділі буде описано процес моделювання та проєктування інформаційної системи та формат вхідних даних. Четвертий розділ міститиме детальний опис програмної реалізації розробленої системи.

Кваліфікаційна робота складається з 76 сторінок (без додатків), 40 рисунків, 1 додатку та 34 джерел.

Ключові слова: *OSINT, вебзастосунок, кластеризація новин, трансформер, тональний аналіз, резюмування, AWS-EC2, AWS-S3, MongoDB.*

ABSTRACT

to the qualification work by the student of the group 401 of Petro Mohyla Black Sea National University

Taras Chepura

“WEB APPLICATION FOR AUTOMATING OSINT IN SOCIAL-MEDIA RESOURCES”

The relevance of this work lies in providing rapid analysis of open news streams by developing a web application that integrates transformer-based classification and clustering models.

The object is the processes of collecting, processing, and analyzing open news data using web-scraping, natural-language-processing, and cloud-computing technologies.

The subject is a combined method for clustering news into events, algorithms for sentiment analysis and event categorization, and models for generating event descriptions.

The aim of the work is to ensure quick analysis of open news streams by developing a web application that integrates transformer-based classification and clustering models.

Chapter 1 will present an analysis of the subject area, covering OSINT and its application to data analysis from social-media resources. Chapter 2 will be devoted to describing the models and methods that can be applied to the task of automated OSINT collection. Chapter 3 will describe the process of modeling and designing the information system and the input-data format. Chapter 4 will contain a detailed description of the software implementation of the developed system.

The qualification paper consists of 76 pages (excluding appendices), 40 figures, 1 appendix, and 34 sources.

Keywords: OSINT, web application, news clustering, transformer, sentiment analysis, summarization, AWS EC2, AWS S3, MongoDB.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД	– база даних
ЗМІ	– засоби масової інформації
ЦРУ	– Центральне розвідувальне управління
ШІ	– штучний інтелект

AI	– Artificial Intelligence
API	– Application Programming Interface
AWS	– Amazon Web Services
BERT	– Bidirectional Encoder Representations from Transformers
BEGNN	– Bert-Enhanced Graph Neural Network
CAMEO	– Conflict And Mediation Event Observations
CD	– Continuous Deployment
CDN	– Content Delivery Network
CI	– Continuous Integration
CPU	– Central Processing Unit
CSV	– Comma-Separated Values
CSS	– Cascading Style Sheets
CSTS	– Chinese Short Text Summarization
DARPA	– Defense Advanced Research Projects Agency
EC2	– Elastic Compute Cloud
ER	– Entity-Relationship
GCN	– Graph Convolutional Network
GDELT	– Global Database of Events, Language and Tone
HTML	– HyperText Markup Language
HTTP	– HyperText Transfer Protocol
ID	– Identifier
JS	– JavaScript

JSON – JavaScript Object Notation

JSI – Jožef Stefan Institute

LCSTS – Large-Cale Chinese Short Text Summarization

MiniLM – MiniLanguage Model

NLP – Natural Language Processing

NoSQL – Not only SQL

ORM – Object-Relational Mapping

OSINT – Open Source INTelligence

PR – Public Relations

RESTful – Representational State Transfer (architectural style)

RNN – Recurrent Neural Network

RoBERTa – Robustly Optimized BERT Approach

ROUGE – Recall-Oriented Understudy for Gist Evaluation

ROUGE-L – Recall-Oriented Understudy for Gist Evaluation – Longest common subsequence

S3 – Simple Storage Service

SEO – Search Engine Optimisation

SMM – Social Media Monitoring

SPA – Single-Page Application

TDT – Topic Detection and Tracking

TD-NHG – Transformer-based News Headline Generation

UI – User Interface

UML – Unified Modeling Language

URL – Uniform Resource Locator

VAE – Variational AutoEncoder

XML – eXtensible Markup Language

ВСТУП

В умовах глобальної інформаційної взаємодії, коли обсяги відкритих даних зростають експоненційно, оперативне отримання, фільтрація й аналітичне осмислення інформації перетворилося на ключову складову безпеки держав, конкурентоспроможності бізнесу та ефективності журналістських розслідувань. Методологія OSINT протягом останнього десятиліття зазнала значного розвитку: розроблено автоматизовані засоби збору даних з веб-сайтів, соціальних медіа й урядових реєстрів; запроваджено машинне навчання для семантичного аналізу та класифікації текстів; накопичено великі багатомовні корпуси для тренування мовних моделей.

Актуальність цієї роботи зумовлена не тільки зростанням інформаційних потоків, а й необхідністю швидкої верифікації даних у часи гібридних конфліктів та кібервійни. Україна відчуває особливу потребу в ефективних OSINT-інструментах для моніторингу медійного простору, протидії дезінформації та підтримки рішень у реальному часі.

Робота пов’язана з низкою актуальних напрямів: вона спирається на попередні дослідження в галузі семантичного групування новин, використовує сучасні досягнення обробки природної мови (NLP), зокрема трансформерні моделі BERT, RoBERTa та MiniLM, і розвиває концепцію хмарної мікросервісної архітектури.

Підсумовуючи, сучасний стан галузі демонструє значні теоретичні й технологічні здобутки, втім залишає відкритими питання локалізації, доступності та комплексності інструментів OSINT. Запропонована робота актуальна тим, що вирішує цю проблему, створюючи модульний вебсервіс, здатний у напівавтоматичному режимі збирати новинні статті, класифікувати їх за змістом і емоційним тоном, групувати в події та подавати результат користувачу у зручній, візуально зрозумілій формі.

1 АНАЛІЗ ОБЛАСТІ OSINT-МОНІТОРИНГУ НОВИН ТА ФОРМУЛЮВАННЯ ЗАДАЧІ

1.1 Опис предметної сфери

Інформаційний простір соціальних медіа перетворився на ключове джерело відомостей про події у світі. Щодня тут виникають мільйони повідомлень, з-поміж яких оперативно треба виокремлювати достовірні факти, розуміти емоційне тло публіки та фіксувати спроби дезінформації. Саме тому в останнє десятиліття OSINT-підхід на базі штучного інтелекту став стратегічно важливим як для держав, так і для бізнесу.

OSINT – це концепція, методологія та технологія добування і використання інформації з відкритих джерел на законних підставах [1]. Така інформація може бути військовою, політичною, економічною чи іншою, і OSINT широко застосовується для прийняття рішень у сфері національної безпеки, розслідувань тощо. Важливість OSINT постійно зростає в сучасних умовах інформаційного суспільства. За оцінками аналітиків ЦРУ, до 90% розвідувальної інформації може надходити саме з відкритих публічних джерел. Відкриті дані часто не поступаються секретним за цінністю при моніторингу актуальних подій. Це підкреслює значення ефективних методів збору й аналізу відкритої інформації, зокрема новин.

До джерел OSINT належать різноманітні публічні ресурси. Їх умовно поділяють на кілька категорій. Перша – традиційні медіа: друковані видання (газети, журнали), радіо та телебачення різних країн. Друга, найбільш актуальна сьогодні – інтернет-ресурси: онлайн-публікації новинних сайтів, блоги, дискусійні форуми, ресурси громадянської журналістики, соціальні мережі (Facebook, Twitter, Instagram тощо). Саме інтернет-джерела випереджають інші за своєчасністю та легкістю доступу до інформації. Окрім того, використовуються відкриті державні дані (урядові звіти, бюджети, виступи), наукові й професійні публікації, комерційні дані, сіра література (технічні звіти, препринти) тощо [1]. У контексті цієї роботи

основна увага приділяється саме відкритим інтернет-джерелам новинної інформації як об’єкту аналізу.

Проблематика автоматичного виявлення та групування подій. Однією з ключових цілей OSINT у сфері новин є виявлення подій – визначення факту появи нової події або інциденту, що висвітлюється у ЗМІ, та групування новин за цими подіями. Подія в цьому контексті розуміється як реальна подія або тематично єдине явище (наприклад, прес-конференція, бій, економічний звіт), про яке повідомляють кілька незалежних новинних повідомлень. Автоматичне об’єднання таких повідомлень у єдину подію дозволяє отримати цілісне уявлення з різних джерел, виявити ключові факти і тенденції.

Задача групування новинних статей у події є нетривіальною через ряд причин. По-перше, один і той самий факт може описуватися різними словами у різних виданнях; по-друге, новини різними мовами потребують зіставлення (наприклад, події в Україні висвітлюються і українською, і англійською мовами – система повинна розпізнати, що йдеться про ту саму подію). По-третє, події еволюціонують з часом: первинна новина може доповнюватися деталями, з’являються нові пов’язані повідомлення, тож алгоритму потрібно вирішувати, чи віднести нову інформацію до вже існуючої події, чи створити нову. По-четверте, присутній значний потік даних – великі інформаційні агентства публікують сотні новин щодня, і загальний обсяг світових новин вимірюється десятками тисяч статей на добу. Людині неможливо вручну відстежити й згрупувати такий масив інформації оперативно, тому актуальним є використання методів штучного інтелекту для автоматизації цього процесу.

Проблема автоматичного групування новин в події досліджується з кінця 1990-х років у межах програм Topic Detection and Tracking (TDT) – зокрема, DARPA TDT визначила підхід до кластеризації новин за подіями в реальному часі. Класичні методи пропонували представляти тексти у вигляді “мішку слів” та застосовувати статистичні алгоритми кластеризації або первинного виявлення подій (First Story Detection) для визначення нової теми серед потоку документів [2].

Сучасні рішення значно просунулися вперед: використовуються семантичні моделі та embedding-представлення текстів (на основі нейронних мереж), що дозволяє більш точно вимірювати схожість новин навіть різними мовами. Тим не менш, задача залишається складною: необхідно забезпечити як високу точність об’єднання статей однієї події (щоб різні події не зливалися помилково), так і своєчасне виявлення нових подій, які раніше не фігурували в стрічці.

Отже, предметна область даної роботи охоплює автоматизований збір новин з відкритих джерел та їх інтелектуальну обробку з метою виявлення подій. Це поєднує напрямки веб-розробки (створення вебзастосунку, інтеграція з хмарними сервісами), обробки природної мови (розуміння тексту новин), машинного навчання (класифікація, кластеризація, аналіз тональності) та візуалізації даних (аналітичний інтерфейс для представлення результатів).

1.2 Огляд існуючих систем і наукових досліджень у галузі автоматизації OSINT

Для успішної реалізації проєкту необхідно проаналізувати існуючі платформи та системи, що виконують схожі функції, а також сучасні наукові публікації у цій сфері. До відомих прикладів платформ для моніторингу новин та подій належать GDELT, EventRegistry, NewsWhip та ін. Кожна з них реалізує концепцію агрегування відкритих новинних даних і застосовує інтелектуальні алгоритми, проте має свої особливості та фокус.

GDELT – один з наймасштабніших відкритих проєктів з моніторингу новин у світі, логотип якого наведено на рис. 1.1. [3]. Система GDELT в режимі реального часу відстежує новини з усього світу більш ніж на 100 мовах та автоматично виокремлює з них згадки про осіб, організації, локації, теми, а також визначає тональність повідомлень. Ідея проєкту полягає в побудові глобальної бази подій, що відображає життя суспільства через призму медіа: кожна подія, кожен факт у новинах фіксується із зазначенням контексту (хто учасники, де і коли сталося) і

того, «як світ це сприймає» (емоційне забарвлення). GDELT накопичує дані з 1979 року по сьогодення і надає їх у вигляді відкритих наборів (CSV, BigQuery тощо) для дослідників. Для кодування типів подій використовується стандартна схема CAMEO (Conflict and Mediation Event Observations) – кожній події присвоюється категорія (наприклад, «протест», «міжнародна зустріч», «акт насильства» тощо) за спеціальним словником [4].

Рисунок 1.1 – Логотип проєкту GDELT [4]

Архітектура GDELT побудована з акцентом на великих об’ємах даних та машинному навчанні. Як зазначається у блозі проєкту, GDELT застосовує «найбільш потужні алгоритми обробки текстів і глибинного навчання» для екстракції понад 300 категорій подій, тисяч тематичних ознак і відстеження тональності новин у масштабах планети. Фактично GDELT автоматизує повний цикл OSINT для новин: від збору (парсинг світових стрічок новин), через аналіз змісту (класифікація, виділення фактів, визначення тональності/емоцій), до збереження структурованих даних, які можуть бути використані для аналітики і навіть прогнозування. Сильна сторона GDELT – глобальне охоплення і відкритість даних, проте для конкретних прикладних застосувань (наприклад, аналітики подій в Україні) ці дані потребують подальшої фільтрації і обробки. Майбутній проєкт відрізняється від GDELT тим, що спрямований на створення публічного вебзастосунку з конкретним інтерфейсом і функціоналом для кінцевого користувача, тоді як GDELT більше орієнтований на науковців і надання сирих

даних та API.

EventRegistry – це платформа (логотип наведено на рис. 1.2), розроблена дослідниками з JSI (Словенія), що фокусується на агрегації новин за подіями [5]. Вона збирає новини з багатьох джерел по всьому світу і автоматично групує пов'язані новинні статті у так звані «події» або теми. На відміну від GDELT, яка зберігає кожну новину як окрему подію з кодом, EventRegistry формує динамічні кластери статей: якщо різні сайти пишуть про той самий випадок, в системі це представлено як один запис події з набором всіх релевантних статей. Платформа EventRegistry підтримує багатомовність, що дозволяє об'єднувати новини різними мовами. Також вона надає відкритий API для доступу до своїх даних, що дозволяє інтегрувати функціонал пошуку подій у сторонні застосунки. У контексті майбутнього проєкту ідеї EventRegistry є надзвичайно корисними – зокрема, механізми кластеризації новин та багатомовного зіставлення. Планована система буде виконувати схожу задачу групування повідомлень у події, але орієнтуватиметься на потреби українського інформаційного простору і матиме власний інтерфейс представлення подій.

Рисунок 1.2 – Логотип платформи EventRegistry [5]

NewsWhip – комерційна платформа, що підходить до проблеми з іншого боку: вона відстежує поширення новин у соціальних мережах та допомагає виявити, які теми набирають популярність [6]. NewsWhip збирає дані про те, як новинними статтями діляться, як коментують та реагують на них у Facebook, Twitter та інших мережах. Логотип цієї платформи наведено на рис. 1.3.

Рисунок 1.3 – Логотип платформи NewsWhip [6]

Основний продукт – система Spike – дозволяє в режимі реального часу бачити трендові або такі, що набирають популярності, новини з новинних сайтів і соціальних акаунтів [7]. Користувачі (редактори, аналітики SMM, PR-фахівці) можуть фільтрувати ці історії за тематикою, мовою, локацією, конкретною соцмережею, метриками залучення аудиторії тощо. На відміну від GDELT та EventRegistry, платформа NewsWhip менше зосереджена на аналізі змісту новини, а більше – на аналізі реакції читачів. Вона відповідає на питання: “Які новини зараз найбільш резонансні у мережі?”. Наприклад, за допомогою Spike можна дізнатися, які статті про певну подію отримали найбільше поширень у Facebook за останню годину. Для цього NewsWhip використовує власні запатентовані технології відстеження та ранжування контенту за показниками популярності і навіть застосовує алгоритми прогнозування вірусності новин (зокрема, із залученням сучасних моделей AI) [7]. Хоча NewsWhip не агрегує новини у події в явному вигляді, вона надає цінний інструментарій для фільтрації інформаційного потоку за ознаками трендовості. У майбутньому проєкті подібні ідеї можуть використовуватися в модулі фільтрації – наприклад, щоб відсіювати малозначущі або нецікаві для аудиторії новини та виокремлювати найважливіші. Таким чином, функціонал NewsWhip доповнює традиційний OSINT: якщо GDELT та EventRegistry концентруються на контенті повідомлень, то NewsWhip – на реакції соцмереж.

Нижче подано огляд релевантних наукових публікацій, які стосуються застосування технологій штучного інтелекту для аналізу новинних даних в межах OSINT-платформи. Для кожної публікації наведено повну назву, авторів, рік,

джерело, короткий зміст методів/моделей і отриманих результатів, а також релевантність до розробки OSINT-застосунку.

У роботі [8] автором запропоновано модель BEGNN, що об’єднує трансформер BERT із графовою нейронною мережею для задачі класифікації текстів. Модель будує окремий граф слів для кожного документа та витягує ознаки з тексту за допомогою BERT, поєднуючи семантичну інформацію зі структурною (граф взаємозв’язків слів). Додатково введено модуль спільної уваги (co-attention) для інтеграції цих ознак. Експерименти на чотирьох датасетах (в т.ч. новинний Reuters-21578) показали підвищення точності класифікації у порівнянні з базовими методами. Релевантність: метод демонструє ефективність BERT для класифікації новин за категоріями, особливо коли враховано структурні зв’язки в тексті, що корисно для розробки модуля рубрикації новин в OSINT-застосунку.

У праці [9] запропоновано методологію виявлення ключових подій у новинному потоці та автоматичного формування контексту подій. Метод поєднує часово-чутливе подання даних, графові згорткові нейромережі (GCN) та кластеризацію: кожну новину представляють як вузол графа з ознаками, що складаються з семантичних векторів (трансформерні ембедінги) і часових позиційних векторів. Потім застосовують GCN для збагачення вузлів глибокими ознаками і виконують кластеризацію графу для виділення груп новин, що описують одну подію. Для кожного кластера (події) генерується автоматичне резюме, що описує контекст події. Результати: На датасеті новин про спалах Еболи 2014 р. метод показав високу точність і повноту виявлення ключових подій та зміг автоматично згенерувати опис (контекст) події, перевершивши існуючі підходи EvMine тощо. Релевантність: підхід безпосередньо відповідає задачі групування новин за подіями та отримання стислого огляду події в OSINT-застосунку, демонструючи ефективність графових моделей і врахування часу для агрегування новин.

Робота [10] присвячена кластеризації новин, пов’язаних із певними кейсами/справами (наприклад, судовими), із використанням гібридного підходу.

Запропоновано глибоку модель кластеризації, що інтегрує тематичну мережу (topic network) з механізмом багатоголової уваги. Локальні текстові ознаки новини доповнюються глобальними ознаками – структурними зв'язками між новинами через спільні теми та атрибути справи. Використовується варіаційний автоенкодер (VAE) для поєднання локальних і глобальних ознак та їх подання у спільному просторі. Результати: Модель успішно групує новини за окремими випадками; комбінування контекстної інформації, тематики і деталей справи покращило якість кластеризації порівняно з використанням лише традиційних тематичних моделей. Релевантність: підтверджується, що врахування контексту та тематичних зв'язків між документами (через увагу і додаткові знання) суттєво допомагає об'єднувати новини в події. Для OSINT-систем, де новини надходять постійно, такий підхід сприяє точному виявленню і відокремленню інформаційних подій.

У дослідженні [11] аналіз тональності новин використовується для покращення прогнозування економічних показників. Автори сформували великий датасет новин та визначили їхній сентимент (тональність) – позитивний, нейтральний або негативний – за допомогою ансамблю трансформерних моделей. Зокрема, використано чотири попередньо навчені мовні моделі з HuggingFace: DistilBERT-base, два варіанти FinBERT (фінансовий BERT) та Twitter-RoBERTa, які спеціалізуються на визначенні тональності текстів. Комбінування кількох моделей дозволило згладити похибки окремих моделей і підвищити точність визначення настрою. Отримані часові ряди "індексу новинного настрою" були використані в економетричній моделі для прогнозу макроекономічних індикаторів. Результати: Врахування індексу тональності новин покращило точність прогнозування економічної активності порівняно з моделями без новинного індексу. Релевантність: ця робота демонструє можливість автоматичного аналізу тональності новин за допомогою трансформерів та важливість цього аналізу для оцінки ситуації – в OSINT-застосунку відстеження емоційного забарвлення новин може допомогти оцінити громадські настрої щодо подій.

Стаття [12] пропонує підхід до пояснюваного аналізу тональності, який поєднує ієрархічну трансформерну модель для класифікації та механізм екстрактивного реферування тексту. Модель має багаторівневу архітектуру на основі BERT (так званий Hierarchical Transformer): спочатку вона визначає тональність документа, а паралельно генерує короткий екстрактивний виклад (набір ключових речень), що слугує поясненням класифікації. Таким чином, увага трансформера використовується для виокремлення тих частин тексту, які найбільше вплинули на рішення моделі щодо сентименту. Результати: Запропонований підхід дозволяє інтерпретувати, чому модель оцінила тональність певним чином, зберігаючи при цьому високу якість класифікації. Релевантність: для OSINT-моніторингу важливо не лише автоматично визначити тон новини, але й мати обґрунтування (які фрази вказують на негативний чи позитивний тон). Ця робота показує, як можна застосувати ієрархічні трансформери для одержання таких пояснень.

У дослідженні [13] розглядається автоматичне реферування коротких текстів (зокрема новинних повідомлень) із використанням гібридної архітектури: модель об’єднує потужність BERT для генерації контекстуальних ембедінгів та трансформерний енкодер-декодер для генерації стислого тексту. Підхід включає стадію екстракції ключових фрагментів із допомогою BERT та подальше абстрактивне генерування резюме за допомогою декодера на основі трансформера. Така багатоетапна схема дозволяє досягти балансу між точністю (відбір важливих речень) та плавністю викладу (генерація зв’язного тексту). Результати: Запропонований підхід продемонстрував найкращі на поточний момент показники (state-of-the-art) на тестових наборах новин і юридичних документів, перевершивши попередні методи сумаризації. Зокрема, об’єднання контекстних представлень BERT з трансформерним декодуванням значно підвищило якість згенерованих резюме. Релевантність: ця робота підтверджує ефективність трансформерів для автоматичного стислого викладу новин. Для нашого веб-

застосунку це означає можливість швидко отримувати коротке резюме статті, що важливо при моніторингу великих потоків інформації.

Стаття [14] фокусується на задачі генерації заголовків новин, яка розглядається як окремий випадок абстрактивного сумаризації. Автори відзначають типові проблеми секвенційних моделей (RNN) при побудові заголовків: відсутність паралелізму та повторюваність слів у виході. Запропоновано модель TD-NHG – вдосконалений трансформерний декодер спеціально для генерації заголовків. Модель використовує масковану багатоголову самоувагу для кращого вилучення різних аспектів тексту та застосовує комбінацію стратегій декодування (top-k та top-p семплювання) з штрафуванням повторів. Це дозволяє уникнути зайвого повторення слів і захопити важливі фрази оригінальної новини. Результати: На китайських датасетах LCSTS та CSTS модель перевищила базові методи за метриками ROUGE (наприклад, ROUGE-L ~37.5 на CSTS) і генерує заголовки, які точніше відображають зміст новини та є більш різноманітними. Релевантність: якісна автоматична генерація заголовків новин корисна для OSINT-платформи, оскільки дозволяє створити стислий зрозумілий заголовок або короткий підсумок для швидкого ознайомлення з суттю статті.

У роботі [15] автором запропоновано підхід до автоматичного представлення змісту новин у вигляді графу знань, аби полегшити користувачам ознайомлення з суттю великої кількості статей. З корпусу електронних новин автоматично витягуються тріплети «суб’єкт – предикат – об’єкт», на основі яких конструюється знаннява база (knowledge base) у вигляді графу. Граф знань поєднує факти про сутності та події, згадані в новинах, тим самим конденсуючи розрізнені тексти в структуровану форму. Результати: Запропонований підхід дає можливість формувати масштабовані графи знань з новин; як зазначають автори, побудований граф забезпечує комплексне і точне представлення знань для заданого корпусу новин. Релевантність: для OSINT-системи графовий підхід дозволяє здійснювати аналіз зв’язків між особами, організаціями та подіями. Побудова графу на основі

новинного потоку допоможе візуалізувати зв’язки та швидко знаходити приховані взаємозв’язки в даних розвідки з відкритих джерел.

Проаналізувавши аналоги, можна зробити висновок, що на сьогодні немає публічно доступної системи, яка б була повністю сфокусована на українському та міжнародному новинному просторі і водночас об’єднувала такі можливості, як кластеризація подій, тональний аналіз, генерування резюме та зручна візуальна аналітика.

1.3 Постановка науково-практичної задачі та визначення мети роботи

Актуальність теми. В умовах стрімкого зростання обсягів інформації виникає потреба в автоматизованих засобах моніторингу відкритих джерел. Оперативне виявлення важливих подій і отримання узагальненої картини з множинних джерел є критичним для прийняття рішень у державному управлінні, діяльності ЗМІ, аналітичних центрів, бізнес-розвідці. Традиційні новинні агрегатори не надають глибокого аналізу (тональності, впливу), а ручний моніторинг соцмереж забирає надто багато часу. Такий застосунок дозволить об’єднати інформацію з різних відкритих джерел, структуризувати її за подіями, відфільтрувати інформаційний шум і представити користувачу лише суттєві зведення з аналітичними показниками.

Мета роботи – забезпечення швидкого аналізу відкритих новинних потоків шляхом розробки вебзастосунку, що інтегрує трансформерні моделі класифікації та кластеризації.

Об’єкт роботи – процеси збору, обробки і аналізу відкритих новинних даних із використанням технологій веб-скрапінгу, обробки природної мови та хмарних обчислень.

Предмет роботи – комбінований метод кластеризації новин у події, алгоритми аналізу тональності і категоризації подій, моделі генерування описів подій. Планується розробка архітектури і компонентів вебзастосунку для

автоматизації OSINT у соціальних медіаресурсах, а також засоби реалізації серверної та клієнтської частини застосунку в хмарному середовищі.

Відповідно до поставленої мети в роботі необхідно вирішити такі завдання:

– аналіз предметної області та існуючих рішень;
– розробка архітектури системи;
– реалізація модуля збору з попередньою підготовкою новин та модуля обробки новин за допомогою ШІ;
– розробка серверної частини (API);
– розробка клієнтського інтерфейсу.

Очікуваним результатом виконання зазначених завдань має стати прототип вебзастосунку, розгорнутий у хмарному середовищі AWS, що демонструє здатність автоматично агрегувати новини з кількох джерел, групувати їх у події та надавати користувачу узагальнену інформацію про ці події.

Висновки до розділу 1

Проведено аналіз предметної області OSINT щодо відкритих новинних даних, розглянуто особливості роботи з українськими та міжнародними джерелами інформації, обґрунтовано потребу в автоматизації процесу. Вивчено сучасні рішення та наукові підходи: світові платформи (GDELT, EventRegistry, NewsWhip) та наукові публікації на пов’язані теми. На цій основі сформульовано мету і завдання, визначено об’єкт і предмет роботи. Розділ закладає теоретичне підґрунтя для подальшої реалізації технічного рішення, опис якого буде наведено у наступних розділах. Завдання, перелічені у цьому розділі, будуть вирішуватися поетапно: від проєктування архітектури до розробки окремих модулів та інтеграції їх у єдину систему.

2 МОДЕЛІ ШТУЧНОГО ІНТЕЛЕКТУ Й ТЕХНОЛОГІЇ РОЗРОБКИ СИСТЕМИ OSINT-АНАЛІЗУ НОВИН

2.1 Методи класифікації, кластеризації, тонального аналізу та реферування новин

Для розв’язання задачі моніторингу й аналізу новин застосовуються різні методи штучного інтелекту та аналізу даних (рис. 2.1). В цьому проєкті будуть застосовані наступні методи ШІ: класифікація, кластеризація, аналіз тональності та автоматичне реферування. Інші методи наведені для подальшої розробки проєкту.

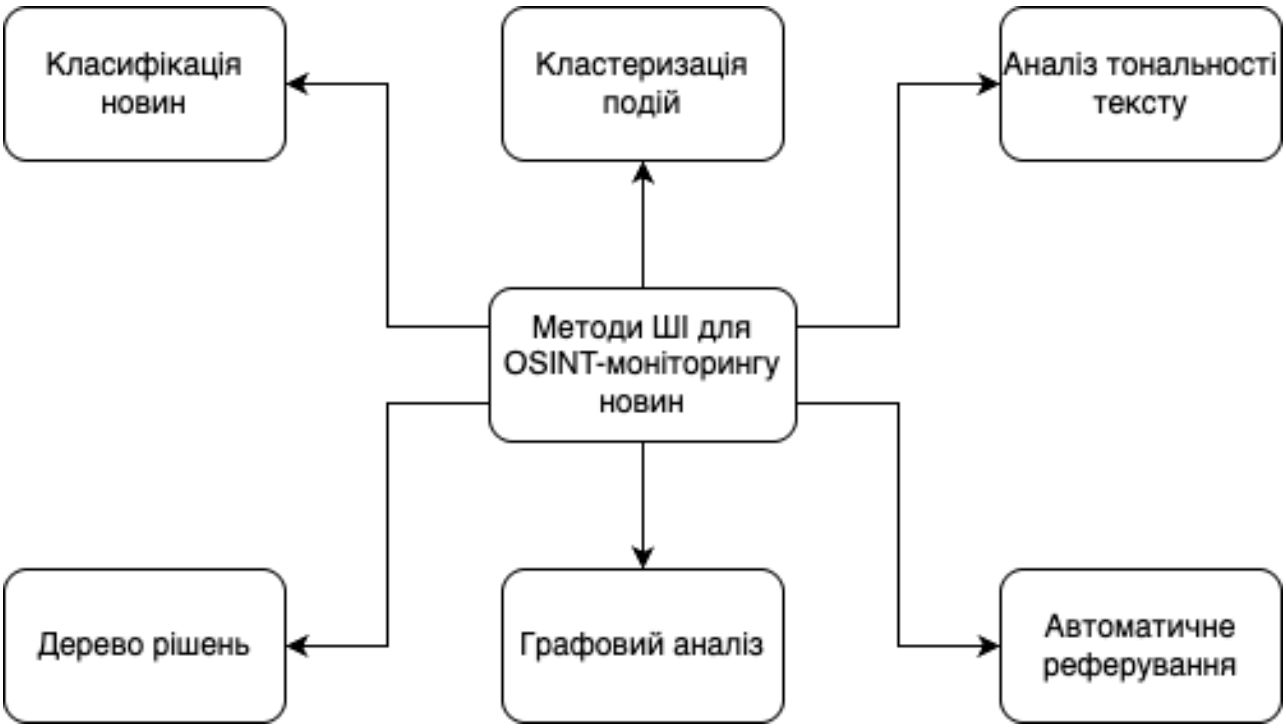

Рисунок 2.1 – Методи ШІ для OSINT-моніторингу новин

Окрім методів ШІ в цьому підрозділі будуть також описані моделі, які застосовувались для вирішення задач класифікації, кластеризації та аналізу тональності. Всі ці моделі були взяті з вебсайту HuggingFace [16].

2.1.1 Класифікація

Класифікація – це задача машинного навчання, що полягає у розподілі об’єктів на заздалегідь визначені категорії або класи на основі їхніх ознак. У контексті новин класифікація буде використовуватися для автоматичного розподілу статей за тематичними категоріями (політика, спорт, технології тощо). Перевагою класифікаційних моделей є їхня здатність автоматично вивчати правила розподілу на класи зі множини розмічених даних. Класифікація застосовується тоді, коли заздалегідь відомо набір цільових категорій і є достатній обсяг прикладів для навчання моделі. Результатом роботи класифікатора є присвоєння кожному новому об’єкту (в нашому випадку – новині) певного класу.

Для вирішення задачі класифікації в ШІ-пайплайні цього вебсайту буде застосована модель MoritzLaurer/mDeBERTa-v3-base-mnli-xnli [17]. Вона базується на архітектурі DeBERTa-v3 із розділеним обчисленням контент і позиційних векторів у механізмі attention, що суттєво покращує подання складних залежностей між словами (рис. 2.2). Вона була попередньо натренована Microsoft на величезному мультилінгвальному корпусі CC100, а потім донавчена на наборах XNLI (15 мов) і MNLI (англійська), завдяки чому може виконувати zero-shot класифікацію природньої мовної інференції на понад 100 мовах без додаткового донавчання.

Архітектура DeBERTa-v3 запроваджує концепцію розділеного attention, у якому контентні й позиційні вектори обробляються окремо та поєднуються на виході, що покращує процес навчання та дозволяє більш точно моделювати синтаксичні й семантичні зв’язки слів.

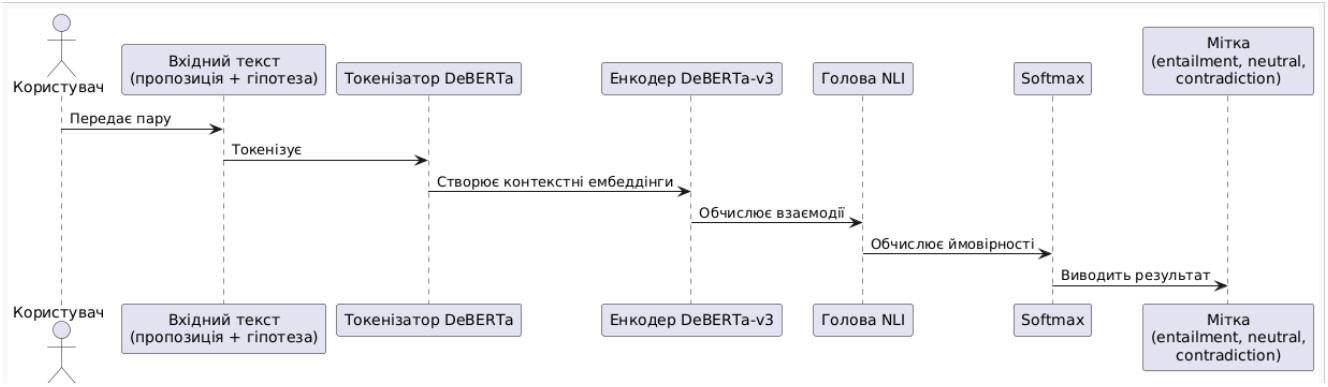

Рисунок 2.2 – Принцип роботи моделі MoritzLaurer/mDeBERTa-v3-base-mnli-xnli

DeBERTa-v3-base-mnli демонструє високу точність на наборі XNLI (15 мов) та MNLI (англійська), випереджаючи більшість інших базових моделей за рахунок оптимізованого pre-training із розширеними задачами Masked Language Modeling та Span Boundary Objective.

2.1.2 Кластеризація

Кластеризація (кластерний аналіз) – це метод аналізу даних, що призначений для об’єднання об’єктів у відносно однорідні групи (кластери) на основі схожості між ними [18]. На відміну від класифікації, кластеризація належить до навчання без вчителя – наперед не задано етикетки класів, і алгоритм самостійно виявляє структуру даних. У задачі моніторингу новин кластеризація буде використана для групування статей, що стосуються однієї події: алгоритми знаходять новини, схожі за змістом та об’єднують їх у кластер, який відповідає певній події. Таким чином вирішується підзадача виявлення подій у потоці розрізнених новин (а саме задача Topic Detection and Tracking, TDT). Подібні системи здатні автоматично знаходити межі новинних історій, визначати, які повідомлення «належать» до однієї події, і виявляти появу нових подій [19]. Перевагою кластеризації є те, що вона дозволяє виявити приховані структури в даних без потреби в ручному маркуванні; її застосовують, коли потрібно згрупувати великі масиви документів за змістом.

Модель paraphrase-multilingual-MiniLM-L12-v2 від команди Sentence-Transformers забезпечує отримання семантичних ембеддінгів речень і абзаців у 384-вимірному просторі, що дозволяє групувати схожі за змістом тексти та виконувати семантичний пошук на понад 50 мовах (рис. 2.3) [20]. Архітектурно ця модель є студентською версією великого трансформера, дистильованою за допомогою технік глибокого дистиляційного навчання із 12 шарами self-attention, що забезпечує високу швидкодію з мінімальною жертвою точності. Ця модель створює ембедінги які потім застосовуються для кластеризації за допомогою ієрархічної агломеративної кластеризації.

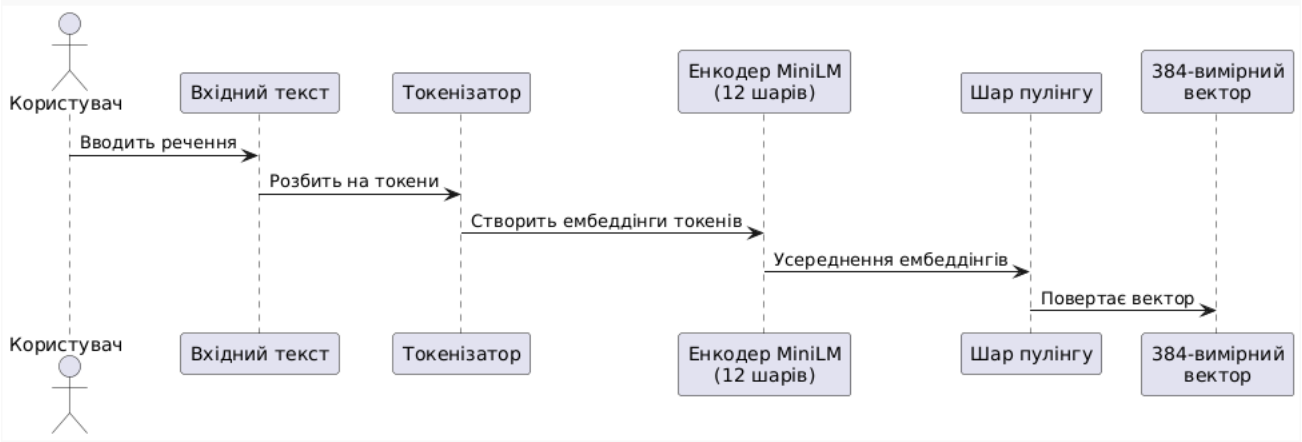

Рисунок 2.3 – Принцип роботи моделі paraphrase-multilingual-MiniLM-L12-v2

Ієрархічна агломеративна кластеризація є методом кластеризації без нагляду, що рекурсивно об’єднує пари найближчих кластерів, мінімізуючи задану відстань зчеплення між ними. Алгоритм формує дерево кластерів (дендрограму), починаючи з окремих спостережень як власних кластерів і послідовно об’єднуючи їх у більші групи (рис. 2.4).

Рисунок 2.4 – Принцип роботи ієрархічної агломеративної кластеризації

На початковому кроці кожен зразок розглядається як окремий кластер, а згодом злиття відбувається відповідно до обраного цільового числа кластерів або порогу відстані. Ієрархічна агломеративна кластеризація часто використовується для групування текстових документів за змістом, сегментації зображень, кластеризації клітин у біології та побудови дендрограм для дослідження відгалужень у даних.

2.1.3 Аналіз тональності тексту

Аналіз тональності тексту – це метод обробки природної мови, спрямований на визначення емоційного забарвлення чи полярності висловлювань у тексті. Інакше кажучи, такий аналіз автоматично класифікує думки чи повідомлення за тональністю – позитивною, негативною або нейтральною [21]. У випадку новинних повідомлень аналіз тональності дозволяє зрозуміти, як саме подається певна інформація в ЗМІ (наприклад, чи висвітлюється подія в позитивному ключі, чи з критикою). Перевагами сучасних підходів до сентимент-аналізу є висока точність визначення тональності та можливість врахування контексту.

Для аналізу тональності новин буде застосовано модель tabularisai/multilingual-sentiment-analysis, яка побудована на базі distilbert/distilbert-base-multilingual-cased, що є дистильованою версією BERT з 6 шарами [22]. Після токенізації та проходження через енкодер DistilBERT вектор CLS-токена подається

в класифікаційну голівку, яка через Softmax визначає найбільш вірогідний клас тональності в п’яти категоріях від «Very Negative» до «Very Positive» (рис. 2.5).

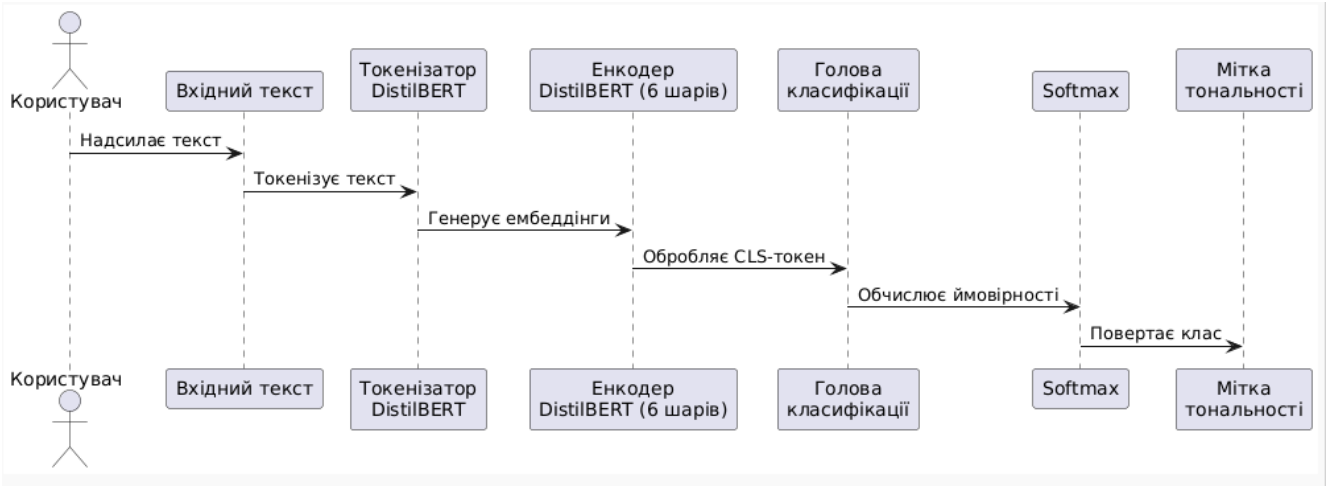

Рисунок 2.5 – Принцип роботи моделі tabularisai/multilingual-sentiment-analysis

Під час тонкого налаштування використовувалися синтетичні дані, згенеровані за допомогою потужних LLM, що надало змогу вбудувати культурні та мовні нюанси кількох десятків мов. У результаті модель досягає високої точності та узгодженості на валідаційних наборах із легких і середніх текстів, включно з відгуками клієнтів, постами в соціальних мережах та оглядами продуктів

2.1.4 Дерево рішень

Дерево рішень – це модель машинного навчання у вигляді дерева, де внутрішні вузли відповідають перевіркам певної ознаки, гілки – результатам перевірки, а листові вузли – кінцевим рішенням або класам. Дерева рішень можуть застосовуватися для задач класифікації або регресії. У контексті цієї системи дерева рішень доцільно розглядати як один із можливих алгоритмів класифікації (наприклад, для тематичної класифікації новин або визначення тональності). Їхня перевага – це інтерпретованість: побудоване дерево фактично являє собою набір правил «якщо – тоді», зрозумілий для людини. Крім того, деревоподібні моделі достатньо швидко навчаються і здатні працювати з різнотипними вхідними

ознаками (числовими, категоріальними). Застосовувати дерева рішень варто тоді, коли важлива прозорість ухвалення рішень моделлю або для швидкого прототипування простого класифікатора.

Ієрархічна структура дерева рішень починається з кореневого вузла, в якому перевіряється певна ознака, і далі для кожного можливого результату цих перевірок формуються відповідні гілки й дочірні вузли. У вузлах враховуються порогові значення або категорії ознак, а у сухостійних листях – остаточне рішення або клас (рис. 2.6).

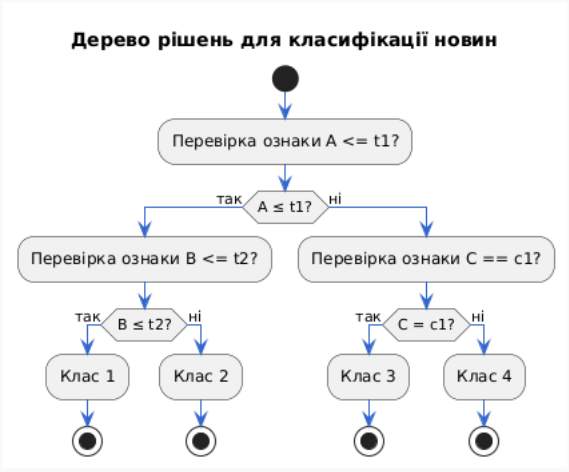

Рисунок 2.6 – Дерево рішень для класифікації новин

Така схема ілюструє послідовність «якщо–тоді», забезпечуючи високу інтерпретованість моделі, де кожне правило може бути безпосередньо прочитане та проаналізоване людиною.

2.1.5 Графовий аналіз

Графовий аналіз – це підхід до структурування та дослідження даних, в якому об’єкти представляються у вигляді вершин графа, а зв’язки між ними – у вигляді ребер. Метою графового аналізу є розуміння того, як об’єкти пов’язані між собою, і виявлення прихованих закономірностей у мережевій структурі даних [23]. На відміну від традиційного аналізу, що оперує лише властивостями окремих об’єктів,

графовий підхід фокусується на відносинах між об’єктами (парах об’єктів) та загальних властивостях мережі. У задачі моніторингу новин графовий аналіз може бути застосований для відстеження зв’язків між подіями, темами та учасниками. Наприклад, якщо розглядати новинні події як вершини графа, можна з’єднувати ті події, що мають спільні сутності (ті самі особи, організації або локації) – це утворює мережу взаємопов’язаних подій. Аналізуючи таку мережу, можна виявляти спільноти подій (кластери тісно пов’язаних новинних історій) або визначати найбільш впливові новини (в термінах графа – вузли з високою центральністю). Загалом, графові алгоритми дозволяють визначити, наскільки важливим є певний вузол у мережі, знайти групи тісно пов’язаних вузлів (спільноти) та виявити найкоротші шляхи зв’язку між об’єктами [24]. Переваги графового аналізу проявляються у випадках, коли досліджуються соціальні зв’язки, наукові цитування, зв’язки між подіями та іншими сутностями тощо.

У графовому представленні кожна новинна подія чи сутність відображається як вершина, а відношення між ними – як ребро. Архітектура графового аналізу наведено на рис. 2.7.

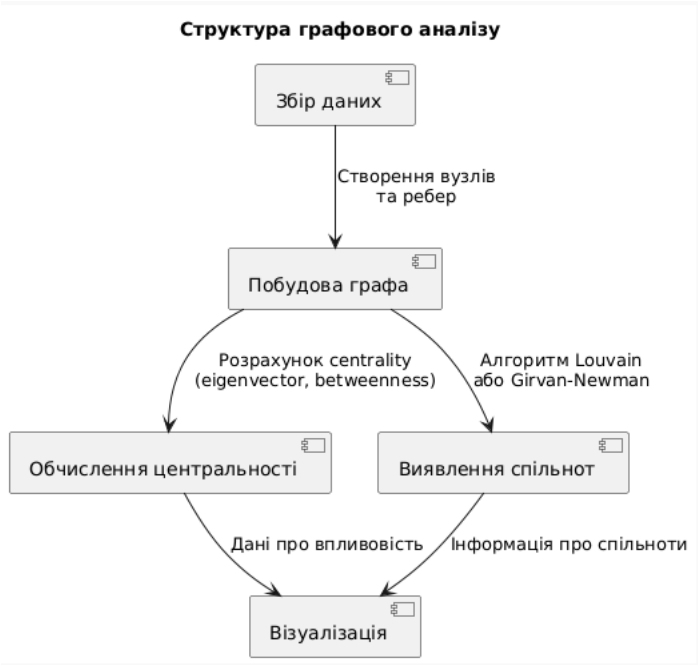

Рисунок 2.7 – Структура графового аналізу

За допомогою аналізу центральності можна виявляти найбільш впливові вузли, а алгоритми знаходження спільнот – групи тісно пов’язаних подій. Ця діаграма демонструє базову структуру графа з декількома вузлами й ребрами, а також показує підсистему обчислення центральності та виявлення спільнот.

2.1.6 Автоматичне реферування

Автоматичне реферування (резюмування) текстів. Реферування полягає у автоматичному стислому викладенні змісту документа – тобто, генерації короткого резюме тексту, що містить його найважливіші положення. В цьому випадку йдеться про можливість автоматично створювати короткий огляд новинної події на основі сукупності окремих статей, що входять до цього кластеру. Існують дві основні парадигми автоматичного реферування: екстрактивне та абстрактивне. При екстрактивному підході резюме формується шляхом вибору найбільш важливих речень із оригінального тексту; при абстрактивному – генерується новий текст, який передає основний зміст, можливо, іншими словами. Кожен з підходів має свої переваги: екстрактивне реферування простіше реалізувати і воно гарантує, що у резюме використовуються достовірні фрази з оригіналу, тоді як абстрактивне дозволяє отримати більш зв’язний і «людяний» виклад, хоча потребує складніших моделей. В контексті OSINT-моніторингу новин автоматичне резюмування корисне тим, що дає змогу швидко отримати короткий звіт по кожній події, зекономивши час аналітика при ознайомленні з великою кількістю схожих повідомлень.

На діаграмі, що зображена на рис. 2.8, показано послідовність етапів автоматичного реферування: попередню обробку, екстрактивний блок із ранжуванням речень та абстрактивний блок із трансформером, а також об’єднання результатів у фінальне резюме.

Рисунок 2.8 – Пайплайн автоматичного реферування

Таким чином, поєднання розглянутих методів – класифікації, кластеризації, аналізу тональності та реферування – забезпечує інтелектуальні можливості для автоматизованого опрацювання великого масиву новин.

2.2 Обґрунтування вибору фреймворків та хмарної інфраструктури для вебзастосунку

Для побудови програмної системи автоматизованого OSINT-моніторингу новин було обрано хмарно-орієнтовану багаторівневу архітектуру. Основними компонентами цієї архітектури є веб-інтерфейс (frontend), сервер застосунку з API (backend), підсистема збору та обробки даних, база даних для збереження результатів, а також зовнішні сервіси штучного інтелекту для аналізу тексту. На рисунку 2.9 показано узагальнену структурну схему системи.

В якості платформи для розгортання системи обрано AWS – одну з найбільших хмарних екосистем, що надає широкий спектр сервісів для зберігання, обробки даних та машинного навчання. Хмарні ресурси дозволяють обробляти великі обсяги даних і підтримувати значну кількість користувачів, динамічно адаптуючись до навантаження.

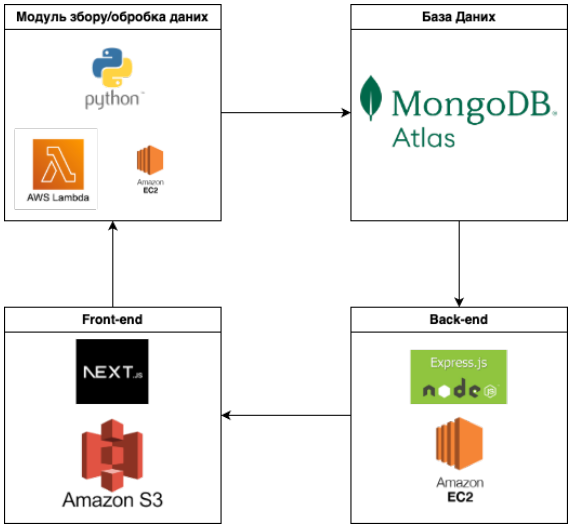

Рисунок 2.9 – Загальна структурна схема системи

Зокрема, в даному проєкті пропонується задіяти такі сервіси AWS:

– amazon EC2 (Elastic Compute Cloud) – для розміщення серверної частини. Планується використання двох типів EC2-екземплярів: постійного серверу API (для забезпечення роботи API в режимі 24/7) та Spot-екземплярів для фонового збору і попередньої обробки даних, який запускається через AWS Lambda[26]. EC2 Spot Instances – це спеціальний тип віртуальних машин AWS, які надаються за залишковим принципом зі знижкою до 90% від звичайної ціни [25]. Використання Spot-екземплярів є доцільним для виконання періодичних або нестандартних завдань (таких як парсинг стрічок новин, попередній аналіз текстів), які можуть витримати короткочасні перерви в разі відкликання ресурсу AWS. Таким чином, поєднання стандартних EC2 для критичних компонент (API) та Spot-EC2 для масового збору даних забезпечить оптимальний баланс;

– amazon S3 (Simple Storage Service) – для розміщення статичних файлів веб-інтерфейсу [27]. Веб-застосунок (інтерфейс користувача), розроблений як односторінковий застосунок на Next.js, буде збиратись у статичну сторінку (HTML, CSS, JS) і розгорнутий в AWS S3 з подальшою роздачою через мережу доставки контенту (CDN) Amazon CloudFront [28]. Такий підхід забезпечує високу продуктивність, масштабованість (S3 автоматично витримує практично будь-яке

навантаження запитів) та знижує витрати на підтримку окремого веб-сервера для інтерфейсу;

– інші сервіси AWS. У разі потреби глибшої аналітики текстів можуть бути інтегровані сервіси штучного інтелекту AWS, такі як Amazon Comprehend (для аналізу тональності та виявлення сутностей у тексті) або Amazon SageMaker (для розгортання власних моделей машинного навчання). Хоча їх використання не є обов'язковим – необхідні модулі аналізу можуть бути реалізовані і самостійно в коді застосунку або за допомогою сторонніх бібліотек Python – наявність таких сервісів в екосистемі AWS підвищує гнучкість архітектури (за потреби, можна швидко масштабуватися, передавши частину задач на керовані сервіси).

Back-end (Node.js API). Серверна частина застосунку буде реалізована з використанням платформи Node.js [29]. Node.js обрано через велику екосистему модулів та її продуктивність на задачах великого потоку запису та зчитування інформації. Застосунок на Node.js надає RESTful API для клієнтської частини, реалізуючи бізнес-логіку: обробку запитів користувачів (запити на отримання списку останніх подій, деталей події, фільтрацію подій тощо) та агрегацію даних з бази. Як фреймворк для API буде використаний Express – для структурованої побудови API [30]. Node.js добре підходить для швидкої серіалізації/десеріалізації JSON-даних, що є перевагою з огляду на вибір бази даних (MongoDB) та формат даних від зовнішніх джерел (новинні API чи веб-сторінки часто представлені у JSON/XML). Сервер API буде запущено на постійному EC2-сервері під керуванням. Важливо, що архітектура передбачає розділення повноважень: Node.js-сервер виконує тільки легкі операції (прийом запитів, вибірки з БД, пересилання даних), а ресурсомісткі завдання обробки (парсинг, NLP-аналіз) винесені на окрему віртуальну машину (EC2 Spot).

База даних (MongoDB). Для збереження зібраних даних та результатів їх обробки обрано нереляційну документну СУБД MongoDB [31]. Вибір MongoDB обґрунтований гнучкістю схеми даних – новинні повідомлення з різних джерел можуть мати різну структуру і набір полів, що зручно зберігати у форматі JSON-

документів. Крім того, MongoDB добре масштабується горизонтально (через шардинг) і має високу продуктивність на операціях вставки та вибірки JSON-подібних даних, що важливо при індексації великої кількості новин. У системі передбачається одна основна колекція для сирих новинних статей (або посилань на них) та пов’язаних з ними метаданих (дата, джерело, тощо), а також колекція для подій, де зберігатимуться згруповані кластери новин з аналітичними полями (тональність, резюме, перелік залучених сутностей). MongoDB планується використовувати як окремий керований сервіс (MongoDB Atlas).

Клієнтська частина системи – веб-інтерфейс для кінцевого користувача – реалізована як односторінковий веб-застосунок (SPA) на основі фреймворку Next.js (React) [32]. Next.js надає можливості серверного рендерингу (для SEO та швидкого першого завантаження) та зручний інструментарій для побудови інтерфейсу. Клієнтська частина відповідає за відображення списку новинних подій, графіків, статистики тональності, та інтерактивний пошук/фільтрацію. Планується реалізувати візуалізацію даних (наприклад, часові діаграми появи нових повідомлень, граф зв’язків подій) за допомогою JavaScript-бібліотек (D3.js, Chart.js тощо). Взаємодія frontend з сервером відбувається через API (HTTP-запити до Node.js сервера). Архітектурне рішення із винесенням клієнтської частини на окремий статичний хостинг та сервер на API-сервер дозволяє масштабувати кожен з цих компонент незалежно: при збільшенні кількості користувачів можна запустити декілька екземплярів Node.js API за балансувальником навантаження (AWS LoadBalancer), в той час як клієнтська частина на S3/CloudFront автоматично масштабується під навантаженням.

Модульність та масштабованість. Система спроектована модульно, що означає чітке розділення на компоненти зі зрозумілими інтерфейсами. Можна виділити такі основні модулі:

– модуль збору даних – відповідає за агрегування новин (скрипти парсингу RSS-стрічок, API джерел або веб-скрапінгу сторінок. Виконується на EC2 Spot за розкладом, результати пишуться в БД);

– модуль обробки та аналізу – здійснює класифікацію, кластеризацію новин в події, аналіз тональності та автореферування;

– модуль зберігання даних – база даних MongoDB з налаштованими індексами та реплікацією;

– API-модуль – Node.js сервер, що виступає посередником між клієнтською частиною та базою/логікою;

– модуль інтерфейсу – клієнтська частина на Next.js.

Така розподілена структура полегшує підтримку та розвиток системи – кожен модуль можна оновлювати або масштабувати окремо. Для забезпечення масштабованості передбачається можливість горизонтального розширення: кількість процесів збору даних можна збільшити (запустивши декілька віртуальних серверів паралельно), базу даних – масштабувати кластером, API – масштабувати додаванням EC2 і Elastic Load Balancer, а клієнтська частина і так розповсюджується через CDN глобально.

Висновки до розділу 2

У другому розділі звіту розглянуто методи, моделі та технології, необхідні для створення системи моніторингу новин на основі OSINT.

По-перше, було проаналізовано низку сучасних методів інтелектуальної обробки текстових даних (класифікація, кластеризація, аналіз тональності, дерева рішень, графовий аналіз, автоматичне реферування) – визначено їхню сутність, переваги та доцільність використання у контексті поставлених завдань.

По-друге, було обрано комбінацію методів ШІ для вирішення поставлених задач, а саме методи кластеризації, класифікації, аналізу тональності та автоматичного реферування.

По-третє, обґрунтовано вибір архітектури програмної системи та її компонентів: обрано хмарне рішення на AWS, що включає серверну частину на Node.js, базу даних MongoDB, клієнтську частину на Next.js, а також допоміжні

інструменти для збору та аналізу даних. Показано, що така архітектура є масштабованою та гнучкою, а використання хмарних сервісів та модульного підходу дозволить ефективно реалізувати функціональність системи.

3 ПРОЄКТУВАННЯ ТА ОЦІНКА ВЕБЗАСТОСУНКУ ДЛЯ АВТОМАТИЗОВАНОГО OSINT-МОНІТОРИНГУ НОВИН

3.1 Інфраструктура й джерела даних системи OSINT-моніторингу

Розроблений вебзастосунок реалізовано в хмарному середовищі AWS та складається з кількох взаємопов’язаних компонентів. Архітектура системи передбачає розподіл завдань між окремими сервісами: збором даних займається окремий модуль на мові програмування Python, який запускається на тимчасовому віртуальному сервері EC2 Spot, API працює постійно на віртуальному сервері EC2 з використанням Node.js. Статичний frontend, створений на Next.js, розгорнутий в AWS S3. Взаємодія між клієнтською частиною і backend здійснюється через API-запити HTTP до серверу Node.js. Центральним сховищем даних у системі є база даних MongoDB, в якій зберігаються зібрані новини та результати їх обробки. Базу даних розгорнуто на окремому сервісі MongoDB Atlas та підключено до API, скрипти Python також мають доступ до неї для запису результатів.

Джерелами даних є перелік відкритих новинних ресурсів, з яких здійснюється збір даних. Кожне джерело описується у колекції sources в базі даних із вказанням параметрів для парсингу. А саме вказується посилання та CSS-селектори для визначення потрібних елементів на сторінці. Для збору даних використовується Playwright, який імітує дії користувача та орієнтується по CSS-селекторам [33]. Для кожного джерела задаються:

– url – базова URL-адреса стрічки новин або розділу сайту, який слід сканувати;

– linkSelector – селектор для вибірки посилань на окремі новини зі сторінки переліку новин;

– linkTitleSelector – селектор для вилучення тексту заголовка зі структури посилання;

– pageIndicatorSelector – селектор елемента, присутнього на сторінці новини, який слугує індикатором успішного завантаження контенту. Він використовується сервісом Playwright для очікування завантаження повної сторінки;

– titleSelector – селектор для основного заголовку новини на сторінці статті;

– contentSelector – селектор для основного блоку контенту статті (тобто області сторінки, що містить текст новини). Скрипт спочатку дістане внутрішній текст, а потім перетворить його в один рядок;

– imageSelector – селектор для основного зображення новини.

Таким чином, щоб додати нове джерело, треба тільки внести відповідний документ у базу даних з потрібними селекторами. У контексті реалізованого прототипу основний акцент зроблено на новинних веб-сайтах як джерелах OSINT-даних, оскільки саме вони забезпечують відкритий доступ до стрічок новин. В подальшому розвитку проєкту можна додати можливість використання різних типів джерел (наприклад, канали Telegram чи публічні стрічки Facebook, але для цього треба реалізувати окремі парсери/API для них).

Обмін даними між компонентами здійснюється через базу даних та HTTP API. Python-скрипт парсингу напряму записує нові статті до колекції news в MongoDB; після кожного циклу збору він також може ініціювати оновлення колекції подій events. Віртуальний сервер з Node.js надає REST API для клієнтської частини, а саме, ендпойнт /api/events повертає список агрегованих подій, зібраних системою. API використовує драйвер mongoose для взаємодії з MongoDB, тобто кожна модель (джерело, новина, подія) має відповідну схему і представлена окремим mongoose-модулем. Клієнтська частина отримує дані через ці API і відображає їх у зручному для користувачів вигляді.

3.2 Архітектура інформаційної системи

Для кращого розуміння архітектури та внутрішніх процесів розробленого застосунку було побудовано UML-діаграми. Вони відображають, як компоненти розгорнуті в середовищі, як взаємодіють між собою, та як відбувається обробка даних. Нижче представлено ключові діаграми: діаграма розгортання, діаграма компонентного складу, діаграма послідовності основних процесів, та діаграма діяльності алгоритмів збору та аналізу даних. Також наведено спроектовану схему MongoDB бази даних (ER-моделі) із основними колекціями та взаємозв'язками. Після кожної діаграми подано опис, що пояснює її елементи.

Діаграма розгортання відображає інфраструктурні елементи системи та з'єднання між ними. На ній показано, як саме компоненти застосунку розподілені по сервісах AWS та як користувач взаємодіє із системою (рис. 3.1).

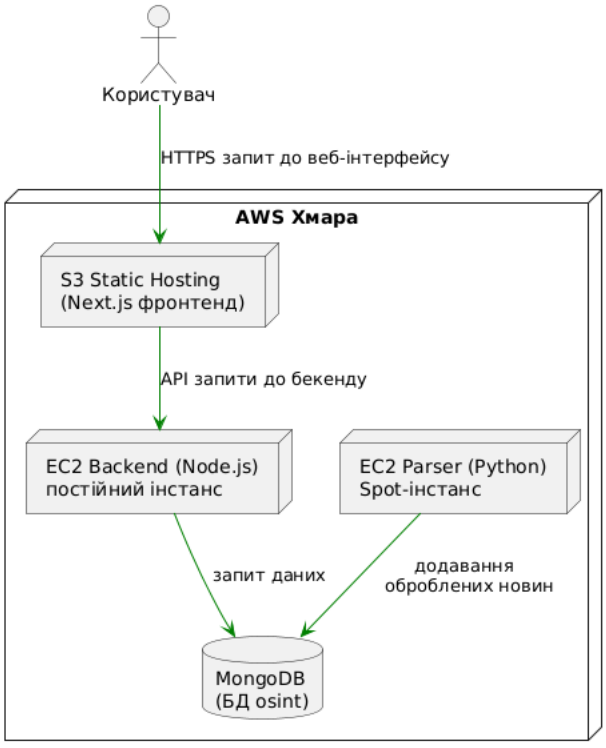

Рисунок 3.1 – Діаграма розгортання системи в AWS

На цій діаграмі показано, що користувач через браузер звертається до вебзастосунку. Клієнтська частина – це зібраний Next.js проєкт, що завантажується з S3 через CloudFront як статичний веб-сайт і потім виконується на боці клієнта. Для отримання динамічних даних вебзастосунок робить запити до API-серверу, запущеного на EC2. Сервер звертається до бази даних MongoDB для читання інформації. Python-парсер працює на іншому EC2-інстансі і також має доступ до тієї ж бази даних – він записує нові документи (новини) та оновлює події. Парсер працює автономно за розкладом. Взаємодії між API та парсером немає, вони пов’язані тільки через базу даних. Діаграма демонструє, що компоненти розподілені, але поєднані спільною БД та мережевими запитами.

Діаграма компонентів (рис. 3.2) відображає програмні компоненти системи та їхні взаємозв’язки на рівні програмного забезпечення. Можна умовно виокремити компоненти для клієнтської частини, серверної частини та модуля аналізу даних. На діаграмі компонентів показано, з яких модулів складається API і Python-скрипт, а також як клієнтська частина підключається до API:

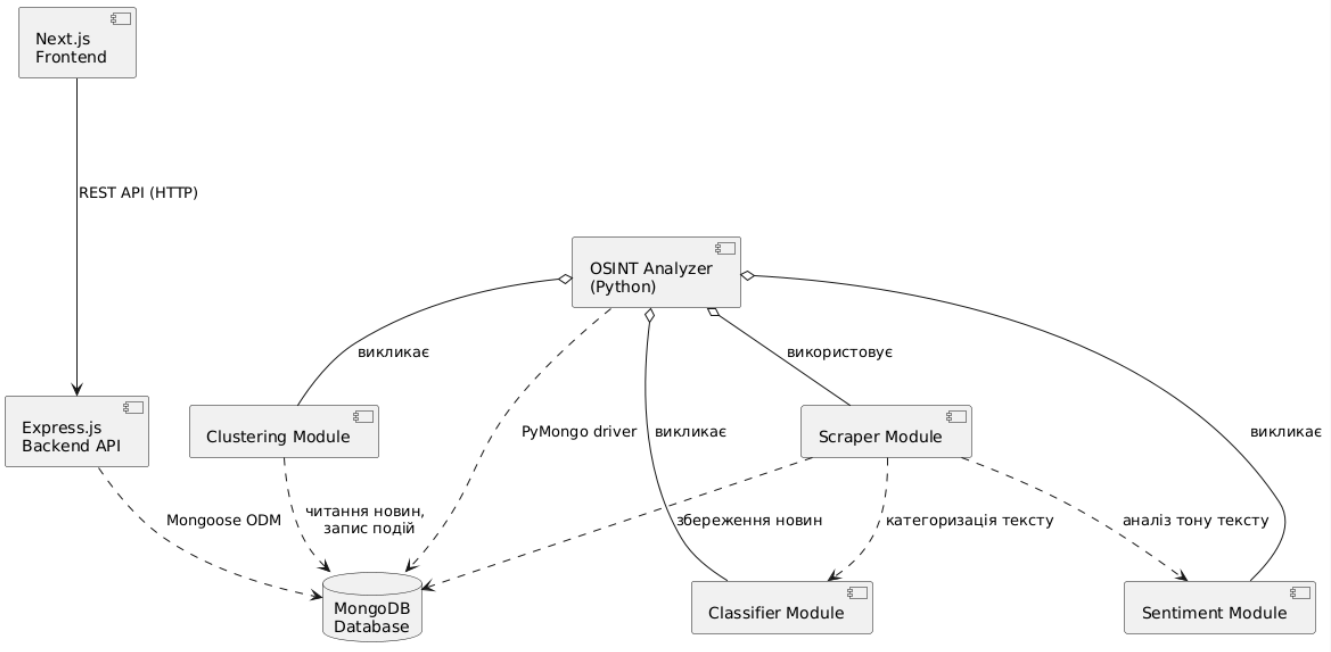

Рисунок 3.2 – Діаграма компонентів системи

На ній видно, що вебзастосунок виступає як окремий компонент, який взаємодіє з API через HTTP-запити (методи GET для отримання даних про новини, події тощо). Сервер безпосередньо спілкується з базою даних через mongoose, реалізуючи доступ до колекцій sources, news, events. Окремим компонентом виступає «OSINT Analyzer (Python)» – так позначено весь python-код для збору й обробки даних. Усередині нього є підкомпоненти (модулі):

– scraper – відповідає за веб-скрапінг (отримання веб-сторінок джерел, витягнення новин). Цей модуль реалізовано з використанням Playwright (headless chromium браузер) для надійного завантаження сторінок. Scraper завантажує список посилань на новини, переходить за кожним посиланням, видобуває текст і далі передає його на обробку;

– classifier – компонент класифікації новин за категоріями. Планується використати модель трансформера XLM-RoBERTa-large-XNLI у режимі zero-shot класифікації. Цей модуль отримує на вхід заголовок та текст новини і повертає мітку категорії (наприклад, «політика», «спорт» тощо);

– sentiment – компонент аналізу тональності (сентимент-аналізу). Він визначає емоційне забарвлення новини: дуже позитивне, позитивне, дуже негативне, негативне чи нейтральне. Для реалізації буде використано готову модель multilingual sentiment analysis. Цей модуль на вхід також бере текст новини і повертає ймовірності для кожної тональності, з яких обирається найвища;

– clustering – компонент кластеризації новин у події. Його завдання – групувати схожі за змістом новини, щоб визначити, які з них стосуються однієї й тієї ж події. Модуль буде реалізовано з використанням методу DBSCAN – алгоритму кластеризації на основі щільності. В якості ознак для кластеризації використано семантичні ембеддінги речень: кожна новина перетворюється на вектор за допомогою моделі SentenceTransformer (конкретно – багатомовна модель paraphrase-multilingual-MiniLM-L12-v2). Ця модель генерує 384-вимірний вектор, який відображає зміст речення/тексту; вона спеціально призначена для задач групування схожих текстів та семантичного пошуку.

На компонентній діаграмі стрілками показано виклики між модулями. Scraper модуль викликає classifier і sentiment для кожної новини (ці модулі працюють як функції, що повертають результат аналізу тексту). Після отримання категорії і тону scraper зберігає новину в базу. Clustering модуль працює незалежно після завершення збору: він зчитує потрібні дані з БД, тому на діаграмі показано його зв’язок з базою даних. API-компонент і клієнтська частина на діаграмі під’єднані до відповідних частин, але не розкриті детально. Для стислості ці деталі опущено, зосереджено на компонентах, що забезпечують головну функціональність збору та обробки інформації.

Далі, щоб показати покрокову взаємодію компонентів у процесі роботи системи, наведено діаграму послідовностей для основного сценарію – додавання нової новини і оновлення подій (рис. 3.3). Ця діаграма ілюструє послідовність дій від моменту запуску парсера до відображення результатів користувачу:

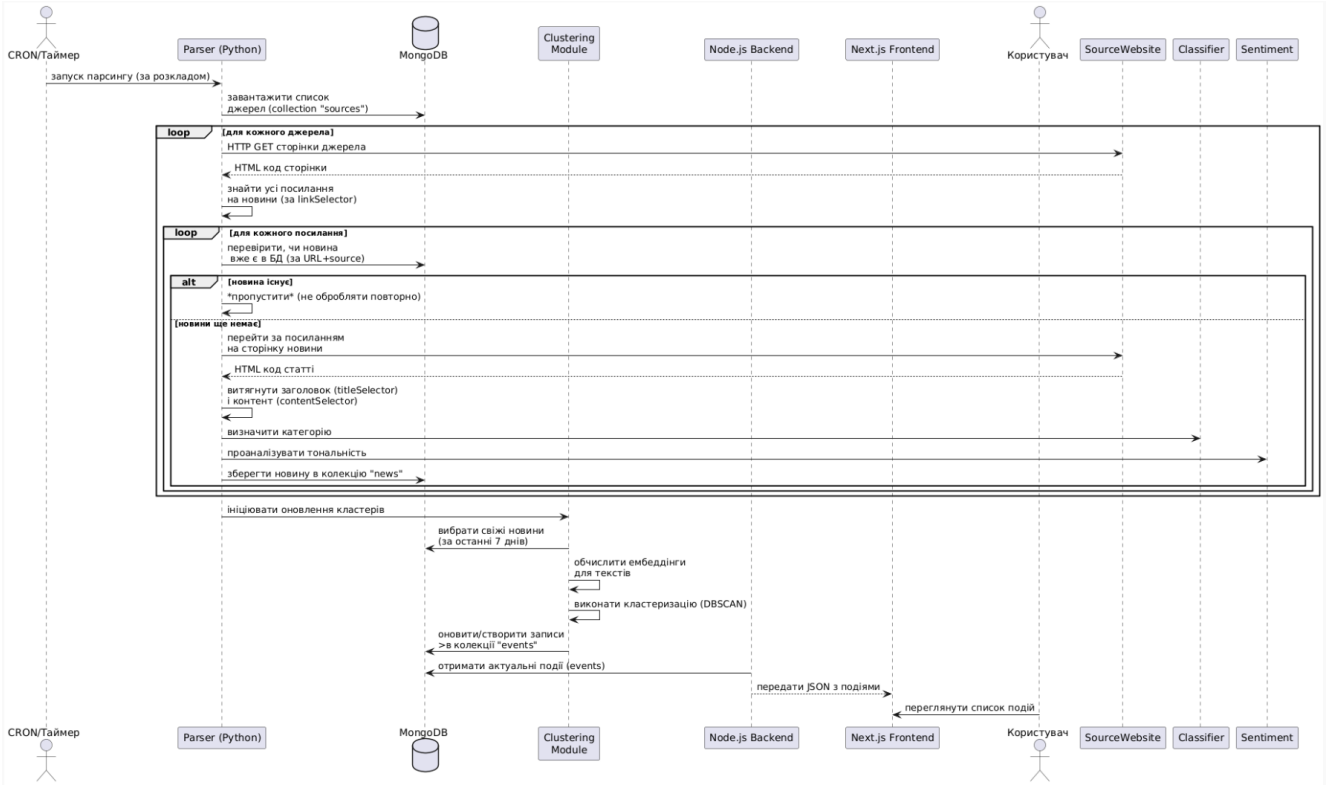

Рисунок 3.3 – Діаграма послідовностей процесу збору новин та оновлення подій

Учасники на цій діаграмі включають і автоматичні процеси, і користувача. Сценарій починається з того, що крон (AWS Lambda) запускає скрипт парсеру. Парсер звертається до бази даних за списком джерел (колекція sources), потім для кожного джерела завантажує в браузері веб-сторінку з переліком новин за допомогою Playwright. Далі відбувається внутрішній цикл: зі сторінки за допомогою заданого селектора джерела вилучаються всі посилання на статті. Для кожного знайденого посилання парсер спершу перевіряє у базі, чи новина з таким URL вже присутня (щоб уникнути дублювання). Якщо запис із таким URL і тим самим джерелом знайдено, значить новина вже була додана раніше – парсер її пропускає. Якщо ж це нова новина, парсер переходить за посиланням (завантажує сторінку статті) і витягує з неї необхідні дані: заголовок, картинку та текст. Після цього парсер викликає модуль classifier для визначення категорії новини та модуль sentiment для визначення тональності. Отримавши ці результати, він формує документ новини та вставляє його в базу (колекція news). Цей цикл повторюється для всіх нових посилань на всіх джерелах.

Після завершення обходу джерел ініціюється крок кластеризації: парсер (або окремий планувальник) викликає clustering модуль. Той звертається до бази даних, вибираючи новини за останній визначений період. Для вибраних новин обчислюються семантичні ембеддінги; потім виконується алгоритм DBSCAN, який групує новини по кластерах. На основі результатів модуль оновлює колекцію events: для кожного знайденого кластера або оновлює існуючий документ події, або створює новий. В документ події заноситься список ідентифікаторів новин, заголовок та згенерований опис. Опис події формується автоматично простим способом: конкатенацією уривків текстів новин кластеру. Хоча такий підхід не дає повноцінного зв’язного резюме, він дозволяє користувачу швидко переглянути ключові деталі з кожної новини. Після оновлення подій основна серверна частина має актуальні дані, готові до відображення.

В заключній частині діаграми показано, як користувач отримує результати. Коли користувач відкриває веб-сторінку, API виконує запит до бази даних для

отримання списку подій. Потім сервер відправляє JSON з подіями клієнту, і користувач бачить оновлений список подій.

Для відображення логіки роботи парсера та кластеризації з точки зору бізнес-процесу, корисно також розглянути діаграму діяльності, що представляє цей процес у вигляді потокової схеми з розгалуженнями (рис. 3.4).

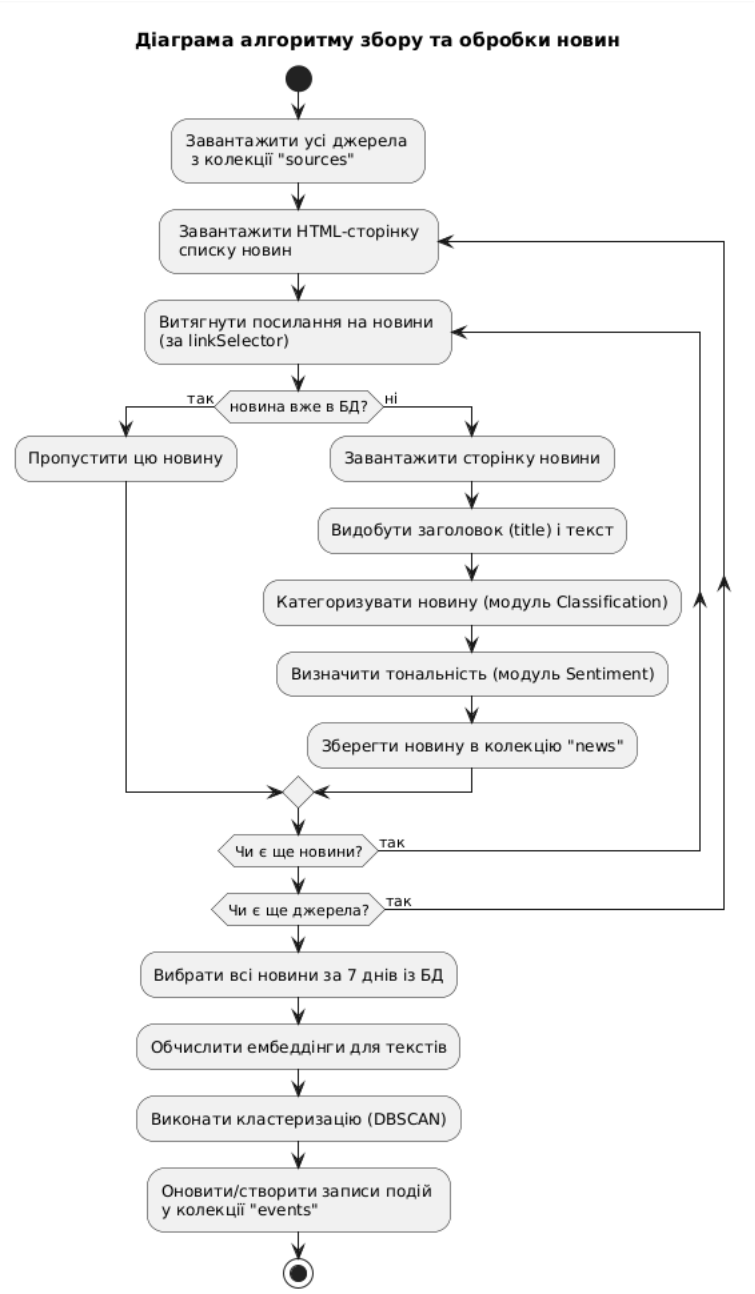

Рисунок 3.4 – Діаграма діяльності алгоритму збору та обробки новин

В цій схемі показано цикли проходження по джерелах і новинах. На початку відбувається отримання списку джерел із БД. Далі для кожного джерела завантаження його стрічки новин і парсинг посилань. Внутрішній цикл перевіряє, чи новина існує: якщо так, то вона пропускається, якщо ні – здійснюється повний процес отримання та аналізу. Дії «Категоризувати новину» та «Визначити тональність» здійснюються AI-модулями. Після виходу з циклів починається стадія кластеризації: вибірка останніх новин, обчислення ембеддінгів (модель MiniLM), виконання DBSCAN. В кінці оновлюються події у БД. Це приводить процес до завершення.

З точки зору структури даних, розроблена система використовує базу даних MongoDB з декількома колекціями. Схема бази даних (ER-модель) включає основні сутності: джерело (Source), новина (News), подія (Event) і пов’язує їх між собою. На рис. 3.5 зображено спрощену ER-діаграму, що показує поля цих колекцій та зв’язки між ними:

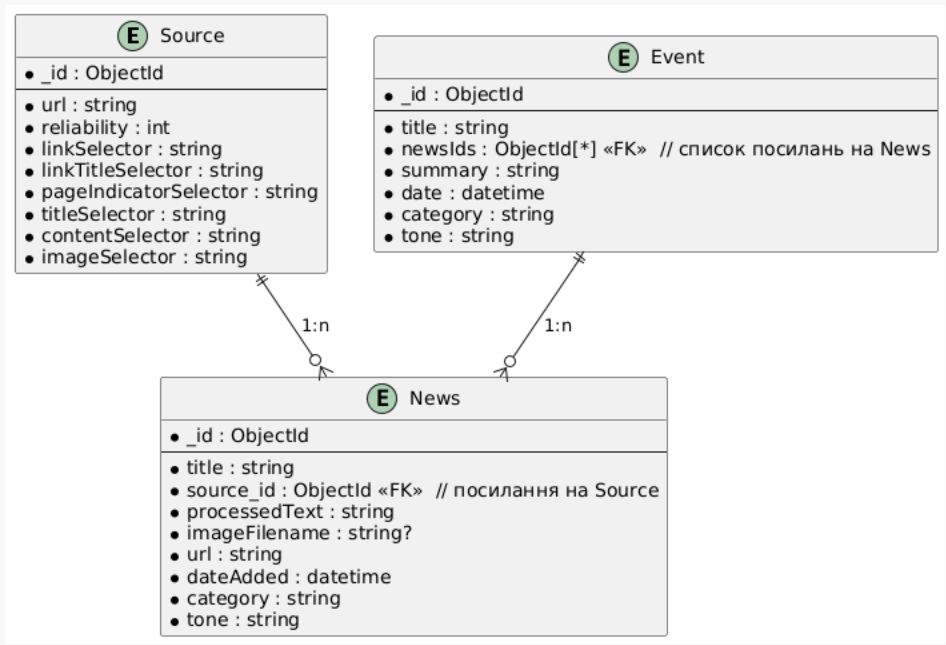

Рисунок 3.5 – ER-модель бази даних MongoDB для основних сутностей.

Колекція Source зберігає конфігурацію джерел; кожен документ містить поля, які були описані. Первинним ключем є _id типу ObjectId, який автоматично

генерується. Колекція News містить зібрані новини. Ключові поля: title (заголовок), processedText (текст новини після очищення, без зайвих пробілів, переносів), url (ідентифікатор новини – у нашій реалізації зберігається відносний URL сторінки на сайті-джерелі), dateAdded (дата і час додання до БД), category (категорія, визначена класифікатором), tone (тональність, визначена модулем аналізу тональності). Поле source (або source_id) є зовнішнім ключем, що посилається на _id відповідного джерела в колекції Source – таким чином кожна новина знає, з якого вона джерела. Також для зручності зберігається назва файлу зображення в полі imageFilename. Колекція Event представляє агреговані події. Її поля: title (назва події), summary (стислий опис події), date (дата формування події), category (категорія події), tone (тональність події). Ключовим елементом є масив newsIds – список ObjectId новин, що належать до даної події. У NoSQL базі даних такий зв’язок представлено через збереження масиву ключів замість окремої колекції зв’язків. На ER-діаграмі зв’язок Event – News показано як 1:many, хоча слід зауважити, що одна новина може бути потенційно включена до однієї події. Зв’язок Source – News – класичний 1:many (кожне джерело може мати багато новин, кожна новина прив’язана до одного джерела).

Наостанок, варто описати модульну структуру застосунку і залежності між модулями у текстовому вигляді (граф залежностей). Node.js API побудовано з використанням Express та Mongoose. Він структурований на кілька модулів:

– models/ – містить визначення схем Mongoose для кожної колекції: Source.js, News.js, Event.js. Кожен з цих файлів експортує Mongoose-модель, яку потім використовують маршрути;

– routes/ – містить маршрути API: sources.ts, news.ts, events.ts. Кожен маршрут імпортує відповідну модель та визначає ендпойнти. Наприклад, events.ts при GET /api/events виконує Event.find().populate('newsIds') щоб отримати всі події з приєднаними документами новин та повертає їх клієнту. Маршрути організовані за ресурсами і підключені в головному файлі index.ts під префіксами (app.use('/api/events', eventsRouter) тощо). Таким чином, backend реалізує типове

REST API. Взаємозалежності: маршрути залежать від моделей (імпортують їх), а моделі залежать від бібліотеки Mongoose. Крім того, backend використовує middleware cors для дозволу запитів з інших доменів та express.json() для парсингу JSON запитів.

Python-аналітичний модуль теж структурований по файлах:

– modules/scraping.py – основний сценарій збору даних. Він залежить від playwright для браузерної автоматизації, pymongo для доступу до MongoDB, а також імпортує модулі classification, sentiment_analysis та text_processing. Тобто Scraper має залежності на всі інші внутрішні модулі, оскільки викликає їхні функції. Також Scraper імпортує Pydantic-моделі News і Source (визначені в models/), що використовуються для валідації і структурування даних перед записом;

– modules/classification.py – містить завантаження моделі HuggingFace для zero-shot класифікації (pipeline з joeddav/xlm-roberta-large-xnli) та функцію predict_category(title, text) для визначення категорії. Цей модуль залежить від бібліотеки transformers і, по суті, автономний – його може викликати будь-який інший компонент, надаючи текст і отримуючи рядок категорії;

– modules/sentiment_analysis.py – аналогічно завантажує pipeline для аналізу тональності (tabularisai/multilingual-sentiment-analysis) і надає функцію analyze_sentiment(title, text). Він теж залежить тільки від transformers. Classifier і Sentiment модулі незалежні один від одного і можуть використовуватися паралельно;

– modules/text_processing.py – містить допоміжні функції опрацювання тексту, а саме функцію clean_text() для видалення зайвих пробілів та переносів. Цей модуль дуже простий, але концептуально його винесено окремо, щоб можна було розширити (наприклад, лематизація тексту, видалення стоп-слів тощо). Інші модулі (Scraper, Clustering) залежать від text_processing для попередньої обробки текстових даних;

– modules/clustering.py – реалізує функцію cluster_events(), яка виконує кластеризацію. Залежить від pymongo (для вибірки/запису даних у БД), від sentence_transformers (для завантаження моделі MiniLM), та sklearn (для DBSCAN). Він також імпортує text_processing (щоб очистити тексти перед побудовою ембеддінгів). Clustering може працювати автономно від scraping.py – тобто його можна запускати за розкладом окремо. У нашій реалізації Scraper після збору новин викликає cluster_events() безпосередньо, тому можна вважати, що Scraper модуль залежить від Clustering (в сенсі логіки виклику).

3.3 Оцінка результатів функціонування й досягнення цілей проєкту

Розроблений вебзастосунок успішно виконує поставлені завдання з автоматизованого збору та аналізу OSINT-даних. Після розгортання системи були проведені тестові запуски збору новин з декількох інтернет-джерел та перевірено коректність роботи модулів класифікації, кластеризації, резюмування та візуалізації результатів. Демонстраційні результати підтвердили, що кожен компонент системи функціонує як очікувалось, а їх взаємодія забезпечує цілісний аналітичний процес.

Зокрема, було налаштовано декілька джерел новин (в якості прикладу – стрічки новин загальнонаціональних і регіональних видань). Система в автоматичному режимі (за розкладом) збирала нові публікації з цих сайтів. В ході тестування було зібрано десятки новинних статей. Кожна зі статей одразу проходила через модуль класифікації, який призначав їй одну з наперед визначених категорій. Використання zero-shot класифікатора на базі XLM-RoBERTa виявилось ефективним для українських текстів: результати класифікації здебільшого співпадали з інтуїтивно очікуваними. Таким чином, завдання класифікації новин за темами було вирішено успішно: кожна новина в базі має поле category, яке надалі використовується для фільтрації або для узагальнення подій.

Паралельно, модуль аналізу тональності визначав емоційний тон повідомлень. У тестових новинах, що були зібрані, переважали нейтральні або негативні тони (що очікувано, адже багато новин стосувались воєнних дій, надзвичайних ситуацій тощо). Система присвоювала мітки тональності від дуже негативних до дуже позитивних.

Наступним кроком оцінено роботу модуля кластеризації, що групує новини у події. В результаті автоматичної кластеризації, система змогла об’єднати новини з різних джерел, які стосувалися однієї події. А саме було зібрано декілька статей про певну надзвичайну ситуацію з різних сайтів – система визначила, що всі вони схожі за змістом (векторні представлення текстів були близькі), і згрупувала їх в один кластер. У колекції events з’явився документ події з заголовком, співзвучним темі, та зі списком ID новин з різних джерел, що описували цю подію. У якості summary події автоматично було сформовано текст, що поєднав ключові фрази з кожної статті – він містив основні деталі. В цілому, завдання кластеризації було досягнуто – система здатна автоматично виявляти, що розрізнені повідомлення стосуються одного інформаційного приводу (події), і представляти їх користувачу згруповано.

Варто зазначити, що вибраний метод (DBSCAN по семантичним ембеддінгам) хоч і є відносно простим, показав себе дієвим на практиці. Були випадки, коли дуже короткі новини або майже ідентичні повідомлення могли створити окремі кластери з однієї новини, але це вирішувалось підбором параметра щільності eps. Більш складні підходи з графовими нейромережами чи тематичними моделями могли б підвищити точність, але і без них отримано задовільний результат для прототипу.

Резюмування (стислий опис подій) у нашій реалізації має обмежений характер: система бере перші ~100 символів з тексту кожної новини у кластері та об’єднує їх. Це дає користувачу приблизне уявлення про контекст кожної новини в межах події, але може виглядати як набір фрагментів. Попри це, навіть такий підхід

продемонстрував користь – на інтерфейсі користувач може швидко пробігти очима зведення і вирішити, чи потрібно читати повні тексти.

Отримані дані відображаються у веб-інтерфейсі, який було спеціально розроблено для зручності OSINT-аналітика. Інтерфейс реалізований у стилі інформаційної панелі (dashboard): він містить кілька розділів або секцій:

– список виявлених подій (рис. 3.6). Головна секція на стартовій сторінці – перелік карток подій. Кожна картка відображає назву події, дату, визначену категорію та тональність;

– аналітичні графіки (рис. 3.6). Для демонстрації аналітичних можливостей, на панелі присутній простий графік – діаграма розподілу подій за тональністю. В тестовому інтерфейсі реалізовано кругову діаграму (Pie Chart), що показує частку “добрих”, “нейтральних” і “поганих” подій серед усіх виявлених;

– детальний перегляд події (рис. 3.7). Користувач має змогу натиснути на конкретну подію в списку, після чого відкривається сторінка деталей події. На цій сторінці відображаються: повна назва події, дата формування, категорія та тональність, стислий опис, а також наводиться список посилань на оригінальні новини.

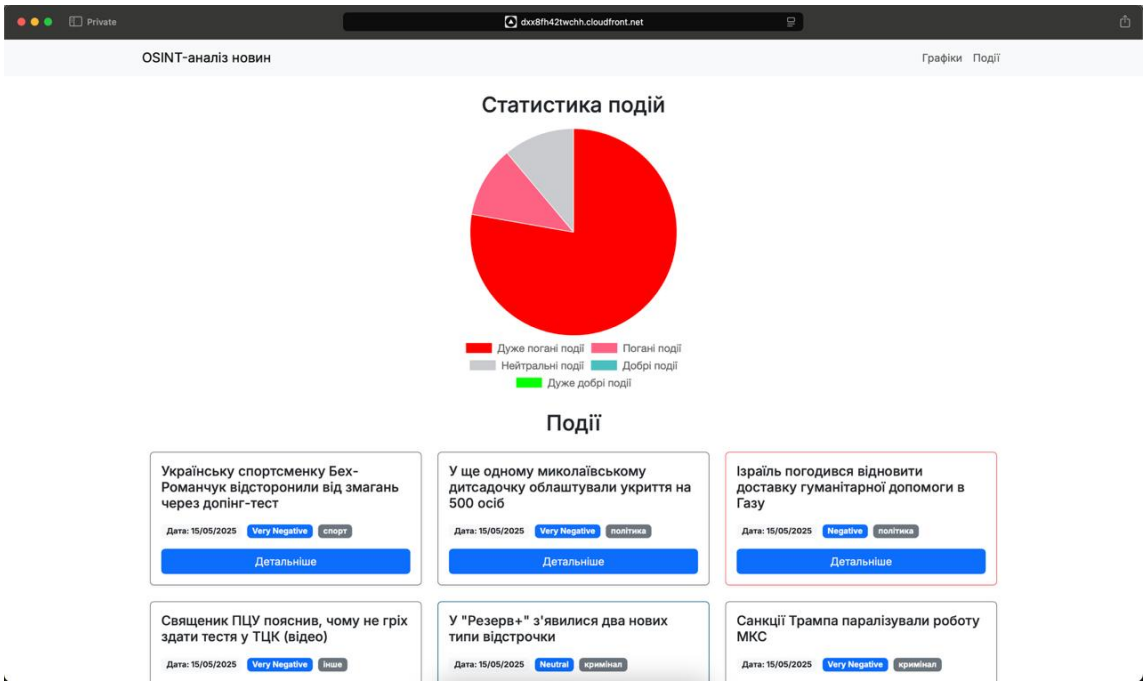

Рисунок 3.6 – Головна сторінка вебзастосунку з усіма подіями.

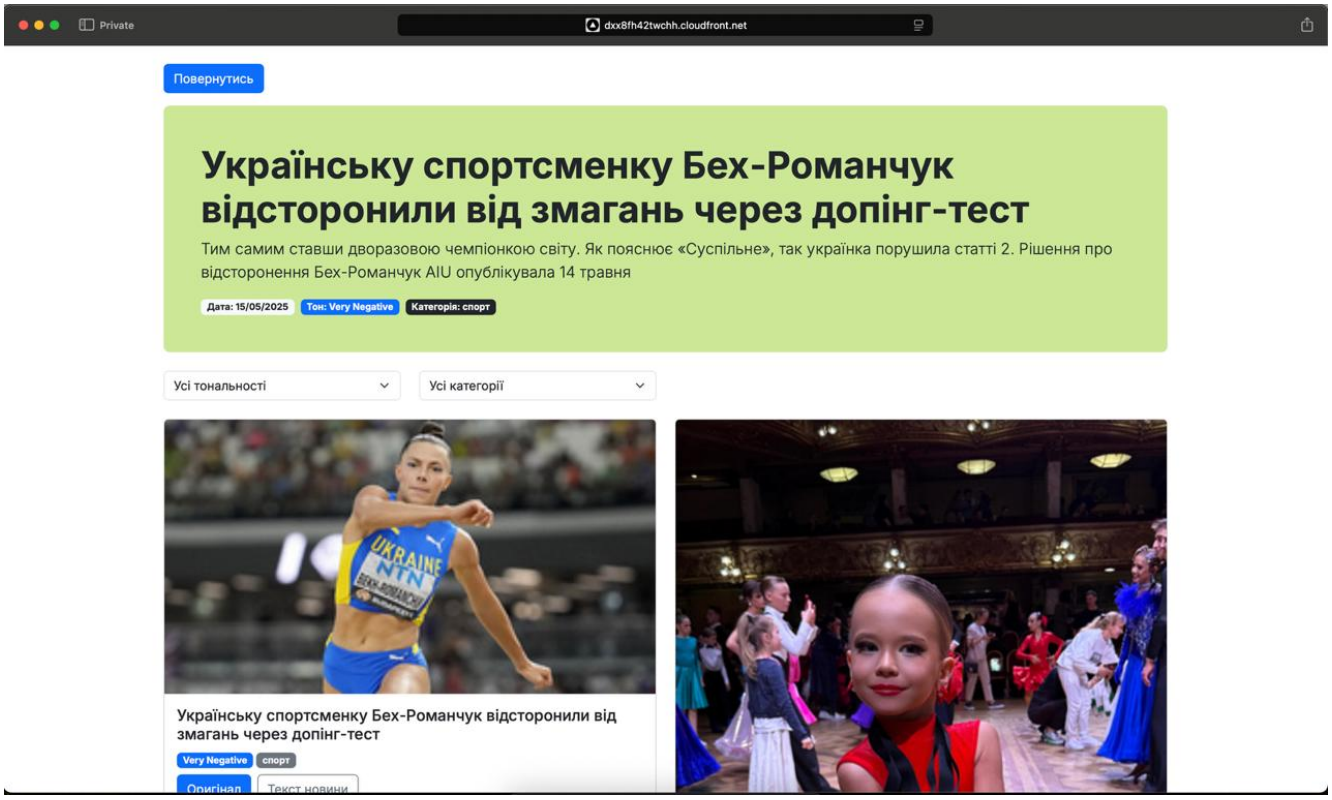

Рисунок 3.7 – Сторінка з конкретною подією.

Отримані результати демонструють, що усі основні функціональні можливості, визначені в вимогах проєкту, були реалізовані:

– збір даних. Система автономно здійснює збір відкритих даних із заданих веб-джерел. Процес збору стабільний: у тестових запусках парсер коректно обробив HTML-структуру цільових сайтів і зібрав необхідну інформацію;

– фільтрація та первинна обробка. Зібрані дані одразу проходять через модулі AI-обробки;

– агрегація (кластеризація). Система успішно агрегує потік новин в узагальнені події;

– аналітика та візуалізація. Проєкт включає елементи аналітичного інтерфейсу – це підтверджує графік розподілу тональності подій, а також відображення категорій і джерел.

Загалом, цілі проєкту досягнуті: створено працюючий прототип OSINT-системи, який інтегрує методи штучного інтелекту для автоматизації збору і

обробки даних із відкритих джерел. Система демонструє здатність ефективно збирати великі обсяги неструктурованої інформації, структуризувати їх, виявляти приховані зв’язки та надавати користувачу стислі зведення з можливістю поглибленого аналізу.

Висновки до розділу 3

У третьому розділі було розглянуто процес розробки та результати впровадження вебзастосунку для автоматизованого OSINT-моніторингу новин. Описано інфраструктуру системи в AWS, зокрема архітектуру із розподілом на клієнтську частину, API, AI-пайплайн та базу даних. Детально показано модель даних – схему бази даних MongoDB з основними сутностями (джерела, новини, події). Розроблено діаграми, що відображають компоненти застосунку та інформаційні потоки між ними, а також алгоритми обробки (збору, кластеризації). Проведена оцінка функціонування системи підтвердила успішність реалізації: система здатна автоматично збирати новини з різних відкритих джерел, застосовувати AI-модулі для їх аналізу, об’єднувати схожі повідомлення у події та презентувати результати у зручному веб-інтерфейсі. Отримані результати свідчать, що поставлених цілей досягнуто – створено прототип, який демонструє можливості штучного інтелекту в задачах OSINT і закладає основу для подальшого розвитку повноцінної системи розвідки з відкритих джерел.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ ДЛЯ АВТОМАТИЗОВАНОГО OSINT-МОНІТОРИНГУ НОВИН

4.1 Реалізація API, клієнтського SPA та AI-пайплайну

У цьому розділі описано реалізацію програмних компонентів вебзастосунку для автоматизованого OSINT-моніторингу новин. Система складається з трьох основних частин: backend (серверна частина застосунку), frontend (клієнтський односторінковий веб-застосунок, SPA) та AI-пайплайну для опрацювання новин (модулі штучного інтелекту для класифікації, аналізу тональності, кластеризації новин у події та автоматичного реферування).

4.1.1 Серверна частина застосунку (backend)

Репозиторій backend містить код серверної частини вебзастосунку, яка реалізує REST API для клієнтського інтерфейсу та взаємодіє з базою даних. API забезпечує основну бізнес-логіку: отримання та збереження даних новин, формування подій, отримання списку джерел. Структура коду організована по модулях відповідно до сфер відповідальності. На сервері-API встановлено Typescript, який додає типізацію.

Основний файл запуску (рис. 4.1) демонструє наступні частини: точка входу, яка ініціалізує веб-сервер (10 рядок), налаштовує підключення до бази даних (17 рядок) та реєструє маршрути (рядки 21-23) для обробки HTTP-запитів. Також в цьому файлі підключаються middleware для ввімкнення CORS та для увімкнення підтримки JSON (рядки 13-14).

```
10  const app = express();
11  const port = process.env.PORT || 3000;
12
13  app.use(cors());
14  app.use(express.json());
15
16  const mongoUri = process.env.MONGO_URI || 'mongodb://localhost:27017/osint';
17  mongoose.connect(mongoUri)
18    .then(() => console.log('MongoDB connected'))
19    .catch(err => console.error('MongoDB connection error:', err));
20
21  app.use('/api/sources', sourcesRouter);
22  app.use('/api/news',    newsRouter);
23  app.use('/api/events', eventsRouter);
```

Рисунок 4.1 – Фрагмент коду точки входу API-серверу index.ts

Роутери (директорія routes/): містить обробники API запитів. Для кожного типу ресурсу передбачено окремий маршрут: новини, події, джерела тощо. А саме, маршрут /api/events повертає список подій, /api/news – список новин, /api/sources – керування джерелами. Кожен маршрут викликає відповідні функції контролерів, що реалізують логіку обробки запиту. Фрагмент коду роутера наведено на рис. 4.2., на якому можна побачити обробку двох роутів – отримання всіх подій та події за id.

```
6   router.get('/', async (_req, res): Promise<void> => {
7     const items = await Event.find().populate('newsIds');
8     res.json(items);
9   });
10
11  router.get('/:id', async (req, res): Promise<void> => {
12    const item = await Event.findById(req.params.id).populate('newsIds');
13    if (!item) {
14      res.status(404).json({ error: 'Not Found' });
15      return;
16    }
17    res.json(item);
18  });
```

Рисунок 4.2 – Фрагмент коду роутери events.ts

Моделі даних знаходяться в директорії models та містять опис структури даних та схем для збереження у базі даних. Виділено моделі для новин, подій, та джерел даних. В моделях визначено поля об’єктів, типи даних та зв’язки між ними. Для прикладу наведено фрагмент коду з моделі подій, на якій можна побачити поля та їх типи, які відповідають полям в колекції в БД.

```
3   export interface IEvent extends Document {
4     newsIds: Types.ObjectId[];
5     title: string;
6     summary: string;
7     date: Date;
8     tone: string;
9     category: string;
10  }
11
12  const EventSchema = new Schema<IEvent>({
13    newsIds: [{ type: Schema.Types.ObjectId, ref: 'News', required: true }],
14    title: { type: String, required: true },
15    summary: { type: String, required: true },
16    date: { type: Date, required: true },
17    tone: { type: String, required: true },
18    category: { type: String, required: true },
19  }, { timestamps: true });
20
21  export default model<IEvent>('Event', EventSchema);
```

Рисунок 4.3 – Фрагмент коду моделі Events

Таким чином, сервер виконує роль посередника між даними та користувацьким інтерфейсом. Він отримує всі зібрані та оброблені новини й події з бази даних. Коли AI-пайплайн генерує нові дані (класифікує та групує новини), результати записуються до БД у вигляді нових об’єктів подій і новин. API-методи читають ці дані з бази та віддають клієнту у форматі JSON.

4.1.2 Клієнтський односторінковий застосунок (frontend)

Репозиторій frontend містить код клієнтської частини – SPA веб-застосунку, що надає інтерфейс користувача для моніторингу подій та новин. Клієнтський веб-

інтерфейс розроблено з використанням TypeScript та фреймворку NextJS. Структура каталогу наведена на рис. 4.4. та включає вихідний код застосунку в директорії src, конфігураційні файли збірки та залежностей (package.json, next.config.ts та інші), а також статику для розгортання.

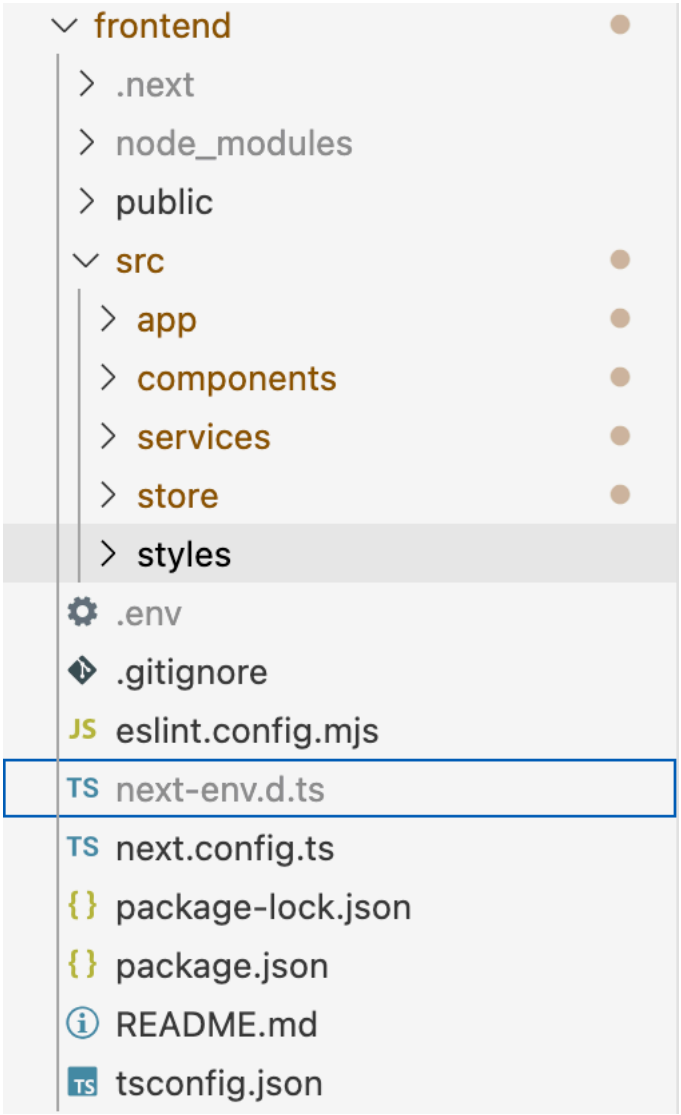

Рисунок 4.4 – Структура клієнтського веб-інтерфейсу

Основу клієнтської частини складає головний модуль застосунку та набір компонентів інтерфейсу. Головна HTML-сторінка (index.html) завантажує SPA-застосунок і містить контейнер для динамічного контенту. Далі логіка роботи побудована на компонентах та маршрутах додатку.

Сторінка списку подій – основний екран, який відображає агрегований перелік подій. Реалізована окремим компонентом EventsSection. Цей компонент завантажує список актуальних подій через REST API (HTTP GET запит до /api/events) та відображає їх у вигляді карточок. Фрагмент коду із використанням redux для отримання подій зі store наведено на рис. 4.5. Для кожної події показуються основні атрибути: назва події, час, категорія та тональність. Це виконується в компоненті EventCard, який показаний на рис. 4.6.

```
const dispatch = useAppDispatch();
const { items: events, status } = useAppSelector(state => state.events);

useEffect(() => {
  if (status === 'idle') dispatch(fetchEvents());
}, [status, dispatch]);

if (status !== 'succeeded') {
  return <div id="events" className="my-4">Завантаження подій...</div>;
}
```

Рисунок 4.5 – Виклик slice з головного компонента подій

```
export default function EventCard({ event }: EventCardProps) {
  return (
    <div className="card mb-3">
      <div className="card-body">
        <h5 className="card-title">{event.title}</h5>
        <p className="card-text"><small className="text-muted">Дата: {new
        <p className="card-text">Тональність: {event.tone}</p>
        <p className="card-text">Категорія: {event.category}</p>
        <Link href={`/events/${event._id}`} className="btn btn-primary">
          Детальніше
        </Link>
      </div>
    </div>
  );
}
```

Рисунок 4.6 – Виклик slice з головного компонента подій

Сторінка деталей події – екран, що відкривається при виборі конкретної події зі списку. Реалізовано окремим компонентом EventDetailClient. Цей компонент отримує по API повну інформацію про подію, включно зі списком новин, що відносяться до події. На екрані деталізації користувач бачить інформацію про

подію, стислий опис та перелік новин у вигляді посилання на оригінал. Фрагмент коду компонента наведена на рис. 4.7.

```
return (
  <div className="container mt-4">
    <Link href="/" className="btn btn-secondary mb-3">← Назад</Link>
    <h1>{event.title}</h1>
    <p><strong>Дата:</strong> {new Date(event.date).toLocaleString()}</p>
    <p><strong>Тональність:</strong> {event.tone}</p>
    <p><strong>Категорія:</strong> {event.category}</p>
    <p><strong>Стислий опис:</strong> {event.summary}</p>

    <h2 className="mt-4">Новини</h2>
    <ul>
      {event.newsIds.map(news => (
        <li key={news._id}>
          <a href={news.url} target="_blank" rel="noopener noreferrer">
            {news.title}
          </a> – {news.dateAdded.slice(0, 10)}
        </li>
      ))}
    </ul>
  </div>
);
```

Рисунок 4.7 – Компонент деталей про подію

У структурі SPA виділені сервіси для централізованого виконання HTTP-запитів до API-серверу. Вони інкапсулюють виклики fetch для звернення до REST-ендпоїнтів. Фрагмент основного коду наведено на рис. 4.8., на якому можна побачити загальну функцію handle та використання її для отримання джерел, подій та новин.

```
async function handle<ResponseType>(path: string): Promise<ResponseType> {
  const res = await fetch(`${BASE}${path}`);
  if (!res.ok) throw new Error(`API error ${res.status} on ${path}`);
  return res.json();
}

export function fetchSources(): Promise<Source[]> {
  return handle<Source[]>('/api/sources');
}

export function fetchNews(): Promise<News[]> {
  return handle<News[]>('/api/news');
}

export function fetchEvents(): Promise<Event[]> {      You, yesterday • Unc
  return handle<Event[]>('/api/events');
}
```

Рисунок 4.8 – Реалізація виконання API-запитів

Після розробки, клієнтська частина вебзастосунку збирається у оптимізований статичний бандл. У результаті формуються статичні файли (HTML, CSS, JS), готові для розгортання на S3. Клієнтський застосунок повністю працює в браузері користувача, звертаючись до API лише для отримання даних. Інтерфейс забезпечує інтерактивність, а саме є можливість переглядати різні події без перезавантаження сторінки.

4.1.3 AI-пайплайн опрацювання новин (директорія newsProcessing/)

Репозиторій newsProcessing містить модулі та скрипти, відповідальні за автоматизований збір та аналіз OSINT-даних з новинних джерел. Цей компонент реалізує AI-пайплайн – послідовність кроків обробки даних, що включає збирання новин, їх попередню обробку, класифікацію за категоріями, аналіз тональності та кластеризацію новин у події. Пайплайн написаний мовою Python із використанням відповідних бібліотек для обробки тексту та ШІ-методів.

Модуль збору новин: відповідає за отримання нових даних із визначених джерел. Модуль періодично проходить по списку джерел з бази даних і збирає нові

публікації. Зібрані новини нормалізуються – виділяються потрібні поля (заголовок, текст, дата, джерело). Після цього нові записи передаються на подальшу обробку. На рис. 4.9. наведено фрагмент коду, на якому видно застосування playwright для проходу по джерелам та підключення до бази даних для отримання джерел.

```
async def scrape_news() -> None:
    client: MongoClient = MongoClient("mongodb://localhost:27017")
    db = client["osint"]

    source_coll: Collection = db["sources"]
    sources = list(source_coll.find())

    news_coll: Collection = db["news"]

    async with async_playwright() as pw:
        browser = await pw.chromium.launch(headless=True)
        page = await browser.new_page()
        for source in sources:
            source_id = source["_id"]
```

Рисунок 4.9 – Фрагмент модуля скрапінгу

Модуль класифікації: реалізує автоматичну тематичну класифікацію новин. Тут використовується модель машинного навчання, навчена попередньо на корпусі новин для розподілу за категоріями. Фрагмент цього модуля наведено на рис. 4.10, де видно категорії, модель «joeddav/xlm-roberta-large-xnli» та застосування моделі. Новини, що надходять на вхід класифікатора, аналізуються текстово: спочатку в попередньому модулі виконується попередня обробка, потім підготовлений текст новини поєднуються разом із заголовком і подається в модель, яка видає передбачену категорію. Результат – для кожної новини визначено одну. Ця категорія додається як атрибут новини (поле “category”).

```python
classifier = pipeline(
    "zero-shot-classification",
    model="joeddav/xlm-roberta-large-xnli"
)

CANDIDATE_LABELS = [
    "політика", "війна", "економіка", "суспільство", "технології",
    "спорт", "кримінал", "культура", "інше"
]

def predict_category(title: str, text: str) -> str:
    """
    Повертає категорію з найвищою вірогідністю
    """
    sequence = f"{title}. {text}"
    result = classifier(sequence, candidate_labels=CANDIDATE_LABELS)
    return result["labels"][0]
```

Рисунок 4.10 – Фрагмент модуля класифікатора

Модуль аналізу тональності: визначає емоційний тон тексту новини – позитивний, негативний чи нейтральний. Кожна новина проходить через цей модуль, який повертає оцінку тональності. Оцінка повертається у вигляді категорії 'Very Negative', 'Negative', 'Neutral', 'Positive', 'Very Positive'. У системі зберігається тональність кожної новини як окреме поле (tone). Фрагмент цього модуля наведений на рис. 4.11.

```python
sentiment_analyzer = pipeline(
    "text-classification",
    model="tabularisai/multilingual-sentiment-analysis",
    return_all_scores=True
)

def analyze_sentiment(title: str, text: str) -> str:
    """
    Аналізує тональність комбінації заголовку та тексту новини.
    Повертає одну з 5 міток:
    'Very Negative', 'Negative', 'Neutral', 'Positive', 'Very Positive'.
    """
    seq = f"{title}. {text}"
    all_scores = sentiment_analyzer(seq, truncation=True, max_length=512)
    scores = all_scores[0]              # список dict з п'ятьма мітками
    best = max(scores, key=lambda x: x["score"])
    return best["label"]
```

Рисунок 4.11 – Фрагмент модуля аналізу тональності

Модуль кластеризації подій: один з ключових AI-модулів, що групує окремі новини у події. Кластеризація базується на схожості змісту новин: припускається, що новини про одну й ту ж саму подію (наприклад, про конкретний інцидент або тему) матимуть схожий зміст, ключові слова, імена осіб чи місць. Фрагмент коду цього модуля наведений на рис. 4.12. На початку модуль завантажує попередньо натреновану багатомовну модель paraphrase-multilingual-MiniLM-L12-v2 з пакета Sentence-Transformers. У процесі кластеризації враховуються лише документи, додані за останні сім діб – це обмеження не допускає накопичення застарілих новин у пам'яті й пришвидшує DBSCAN. Комбінований заголовок із очищеним текстом кожної новини перетворюється на щільний вектор ембеддінгу. Після перетворення векторів запускається алгоритм DBSCAN з косинусною метрикою. Отримані мітки кластерів групуються у словник clusters. Для кожного кластера перевіряється, чи містить він принаймні одну новину зі списку «додані за годину» – саме такий критерій гарантує, що система оновлює лише актуальні події.

```
embedder = SentenceTransformer('paraphrase-multilingual-MiniLM-L12-v2')

def cluster_events():
    """
    Кластеризує новини за останній тиждень у події.
    Створює або оновлює документи в колекції events лише у тих кластерах,
    де є хоча б одна новина, додана за останню годину.
    """
    client = MongoClient("mongodb://localhost:27017")
    db = client["osint"]
    news_coll = db["news"]
    events_coll = db["events"]
```

Рисунок 4.12 – Фрагмент модуля кластеризації

AI-пайплайн працює у тісній зв'язці з базою даних. Він фактично є "двигуном" системи моніторингу, який у фоновому режимі збирає та аналізує інформацію, тоді як API та клієнтська частина займаються відображенням цієї інформації користувачу.

4.1.4 Розгортання на AWS: сервери та розміщення сервісів

Інфраструктура для хостингу рішення реалізована на базі хмарної платформи Amazon Web Services (AWS). У межах проєкту використано інстанси AWS EC2, Elastic Beanstalk, Lambda-функцію та S3 – сховище для файлів, на якому зберігається статична сторінка.

У проєкті використано два окремі екземпляри Amazon EC2, кожен із чітко визначеним набором обов'язків. Перший інстанс відповідає за збір та обробку новин. Це Spot-екземпляр, запущений у режимі флоту Spot, щоб скористатися значною знижкою на простої обчислювальні ресурси. На ньому розгорнуто контейнерний пайплайн на Python: спочатку скрипт-парсер за розкладом піднімає сесію, обходить список джерел, зберігає сирі тексти, а далі послідовно запускаються модулі обробки природної мови–класифікація, часова кластеризація, аналіз тональності, генерування стислого опису події. Результати кожного етапу одразу фіксуються у віддаленій базі MongoDB; при достроковому відключенні Spot EC2 втрачається лише останній неповний пакет, що не критично для потокової аналітики. AWS Lambda автоматично піднімає новий екземпляр кожні декілька годин.

Другий інстанс виконує роль постійного API-шлюзу. Це on-demand EC2 типу, розміщений у тій самій VPC, що й база даних, аби мінімізувати затримку. На ньому працює Node.js-сервер із фреймворком ExpressJS, який надає публічні REST-ендпойнти для клієнтської частини Next.js і для сторонніх сервісів. Інстанс утримує в пам'яті кеш найактуальніших подій, обробляє запити користувачів, а також в майбутньому буде ініціювати email-розсилку, коли в базі з'являються нові або оновлені кластери. Оскільки API-навантаження відрізняється від пакетної обробки, сервер тримається на окремому, гарантовано доступному екземплярі з можливістю вертикально масштабуватись до більшої конфігурації, якщо кількість одночасних користувачів зросте.

Використання двох окремих EC2 інстансів підвищує надійність і масштабованість: веб-застосунок може обслуговувати запити користувачів незалежно від навантаження на AI-процеси. У разі зростання числа джерел або обсягу даних, можна вертикально масштабувати ще один сервер (додавши ресурсів) або навіть розподілити обробку на декілька контейнерів/інстансів. AWS надає гнучкість для такого розвитку: за потреби, архітектуру можна перенести на ECS (Elastic Container Service) або Kubernetes (EKS), що дозволить автоматично керувати кількома контейнерами і авто-масштабуванням.

4.1.5 CI/CD за допомогою AWS CDK

CI/CD-процес побудований навколо AWS CDK та простих скриптів оболонки, які автоматизують збірку, упаковку і розгортання окремих компонентів [34]. Загальна структура проєкту розгортання CDK наведена на рис. 4.13.

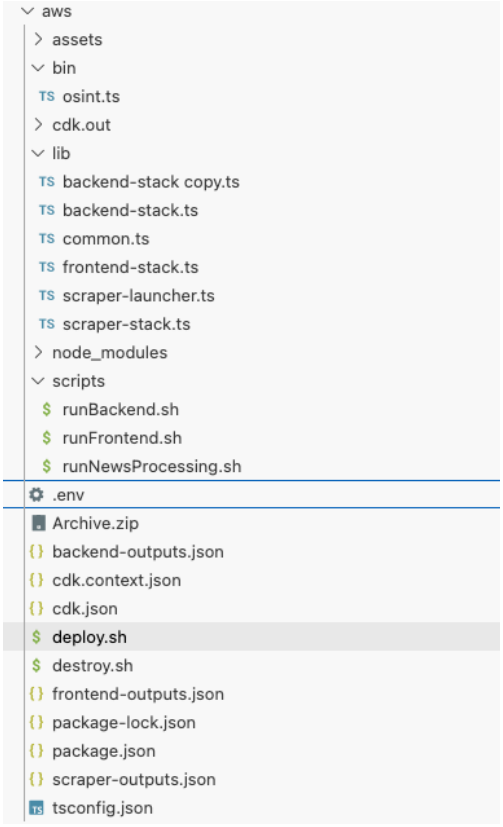

Рисунок 4.13 – Структура проєкту CDK

В цій структурі є 2 головних файли, а саме deploy.sh для запуску стеків, та destroy.sh для їх видалення. В директорії lib розміщені самі стеки, які відповідають за окремі частини проєкту. Директорія scripts містить окремі скрипти для запуску стеків. Інші файли потрібні для запуску автоматичного створення проєкту або згенеровані. В ітозі проєкт запускається за допомогою однієї команди та зупиняється за допомогою іншої.

Першим кроком локально запускається скрипт, що архівує зкомпільований backend у ZIP-файл і завантажує його у спеціальний S3-bucket, створений CDK під час bootstrap'у. Після успішного завантаження без ручного втручання виконується команда cdk deploy для стеку BackendStack із передачею всіх необхідних параметрів (наприклад, URI до MongoDB Atlas). Стек створює «application» в сервісі AWS Elastic Beanstalk, який керує інстансом EC2. Результатом деплою стає JSON-файл із виводами стека, з якого автоматично зчитується адресний вихід Beanstalk-домену та передається далі. Фрагмент коду стеку, який створює всі необхідні компоненти, наведений на рис. 4.14.

aws > lib > TS backend-stack.ts > BackendStack > constructor > vpc

```
1  import * as cdk from "aws-cdk-lib";
2  import { Aws } from "aws-cdk-lib";
3  import { Construct } from "constructs";   6.6k (gzipped: 2.5k)
4  import * as elasticbeanstalk from "aws-cdk-lib/aws-elasticbeanstalk";
5  import * as iam from "aws-cdk-lib/aws-iam";
6  import * as s3assets from "aws-cdk-lib/aws-s3-assets";
7  import * as ec2 from "aws-cdk-lib/aws-ec2";
8
9  interface BackendStackProps extends cdk.StackProps {
10   mongoAtlasUri: string;
11 }
12
13 export class BackendStack extends cdk.Stack {
14   constructor(scope: Construct, id: string, props: BackendStackProps) {
15     super(scope, id, props);
16
17     const vpc = new ec2.Vpc(this, "EBVPC", {
18       maxAzs: 1,
19       natGateways: 0,
20       subnetConfiguration: [
21         { name: "Public", subnetType: ec2.SubnetType.PUBLIC },
22       ],
23     });
24     vpc.applyRemovalPolicy(cdk.RemovalPolicy.DESTROY);
25
26     const appName = "osint-backend-app";
27     const application = new elasticbeanstalk.CfnApplication(this, "EBApp", {
28       applicationName: appName,
29     });
```

Рисунок 4.14 – Фрагмент коду backend-стеку

Другий скрипт у цій самій консолі підхоплює отримане API URL з JSON-виходу, формує середовище для frontend-стеку й викликає cdk deploy для FrontendStack (рис. 4.15). У ньому створюється статичний сайт на S3 з дистрибуцією через CloudFront, підставлений до отриманої API-адреси як Origin. Після успішного створення дистрибуції скрипт синхронізує згенеровану папку out/ із S3-bucket командою aws s3 sync. Таким чином frontend негайно оновлюється без необхідності запускати вручну команди в консолі браузера.

aws > lib > TS frontend-stack.ts > FrontendStack > constructor

```
1   import * as cdk from "aws-cdk-lib";
2   import { Construct } from "constructs";   6.6k (gzipped: 2.5k)
3   import { Bucket, BlockPublicAccess } from "aws-cdk-lib/aws-s3";
4   import { Distribution, ViewerProtocolPolicy, OriginProtocolPolicy } from "aws-cdk-lib/aws-cloudfront";
5   import { S3StaticWebsiteOrigin, HttpOrigin } from "aws-cdk-lib/aws-cloudfront-origins";
6   import { AllowedMethods, CachePolicy } from "aws-cdk-lib/aws-cloudfront";
7   import * as fs from 'fs';
8   import * as path from 'path';
9
10  export interface FrontendStackProps extends cdk.StackProps {}
11
12  export class FrontendStack extends cdk.Stack {
13    constructor(scope: Construct, id: string, props: FrontendStackProps) {
14      super(scope, id, props);
15
16      const siteBucket = new Bucket(this, "SiteBucket", {
17        blockPublicAccess: new BlockPublicAccess({
18          blockPublicAcls: true,
19          blockPublicPolicy: false,
20        }),
21        publicReadAccess: true,
22        autoDeleteObjects: true,
23        removalPolicy: cdk.RemovalPolicy.DESTROY,
24        websiteIndexDocument: "index.html",
25      });
```

Рисунок 4.15 – Фрагмент коду frontend-стеку

Третій етап CI/CD відповідає за скрапер: ZIP-файл із Python-скриптом і залежностями завантажується в S3, а потім cdk deploy для ScraperStack розгортає Lambda-функцію (фрагмент коду на рис. 4.16), яка налаштована для періодичного створення EC2 Spot-інстанс із налаштованим User Data (рис. 4.17). Інстанс автоматично завантажує код зі S3, розпаковує архів та встановлює потрібні пакети.

```
constructor() {
  this.ec2 = new EC2Client({});
  this.mongoUri = process.env.MONGO_URI!;
  this.codeBucket = process.env.CODE_BUCKET!;
  this.codeKey = process.env.CODE_KEY!;
  this.instanceProfile = process.env.INSTANCE_PROFILE_NAME!;
  this.subnetId = process.env.SUBNET_ID!;
  this.securityGroupId = process.env.SECURITY_GROUP_ID!;
}


public async handler(): Promise<{ launched: string[] }> {
  const commands = [
    '#!/bin/bash',
    'yum update -y',
    'yum install -y unzip python3 python3-pip aws-cli',
    `aws s3 cp s3://${this.codeBucket}/${this.codeKey} /home/ec2-user/scraper.zip`,
    'unzip /home/ec2-user/scraper.zip -d /home/ec2-user/osint',
    'pip3 install -r /home/ec2-user/osint/requirements.txt',
    `export MONGO_URI=${this.mongoUri}`,
    'python3 /home/ec2-user/osint/index.py',
    'shutdown -h now'
  ];
  const userData = Buffer.from(commands.join('\n')).toString('base64');
```

Рисунок 4.16 – Фрагмент коду для Lambda-функції

aws > lib > TS scraper-stack.ts > ...

```
1   import * as cdk from 'aws-cdk-lib';
2   import { Construct } from 'constructs';   6.6k (gzipped: 2.5k)
3   import * as s3assets from 'aws-cdk-lib/aws-s3-assets';
4   import * as lambdaNodejs from 'aws-cdk-lib/aws-lambda-nodejs';
5   import * as lambda from 'aws-cdk-lib/aws-lambda';
6   import * as ec2 from 'aws-cdk-lib/aws-ec2';
7   import * as iam from 'aws-cdk-lib/aws-iam';
8   import * as events from 'aws-cdk-lib/aws-events';
9   import * as targets from 'aws-cdk-lib/aws-events-targets';
10  import * as path from 'path';
11
12  interface ScraperStackProps extends cdk.StackProps {
13    mongoAtlasUri: string;
14    mongoDbName: string;
15  }
16
17  export class ScraperStack extends cdk.Stack {
18    constructor(scope: Construct, id: string, props: ScraperStackProps) {
19      super(scope, id, props);
20
21      const vpc = new ec2.Vpc(this, 'ScraperVPC', { maxAzs: 2 });
22
23      const sg = new ec2.SecurityGroup(this, 'ScraperSG', {
24        vpc,
25        description: 'Allow outbound to MongoDB Atlas and S3',
26        allowAllOutbound: true,
27      });
28      sg.addEgressRule(ec2.Peer.anyIpv4(), ec2.Port.tcp(27017));
```

Рисунок 4.17 – Фрагмент коду scraper-стеку

У результаті, не виходячи з термінала, ми за одну команду виконуємо побудову, пакування, інфраструктурний деплой та конфігурацію всіх трьох компонентів системи в AWS. Такий підхід гарантує відтворюваність, прозорість і швидкість оновлень у продакшені.

4.2 Інструкція для кінцевого користувача

Розроблений вебзастосунок призначений для кінцевих користувачів, які будуть використовувати інтерфейс для відстеження новинних подій. Інтерфейс спроектовано інтуїтивно: користувач зможе переглядати поточні події та деталізувати їх вміст. Нижче наведено інструкцію щодо використання основних можливостей системи.

Після запуску вебзастосунку користувач потрапляє на головну сторінку «Події», де відображається перелік актуальних подій, виявлених системою. Кожна подія представлена у вигляді окремого блоку, який містить ключову інформацію про подію. Приклад блоку наведено на рис. 4.18. А саме показано заголовок події, дату, до якого належать новини події, категорія та тональність.

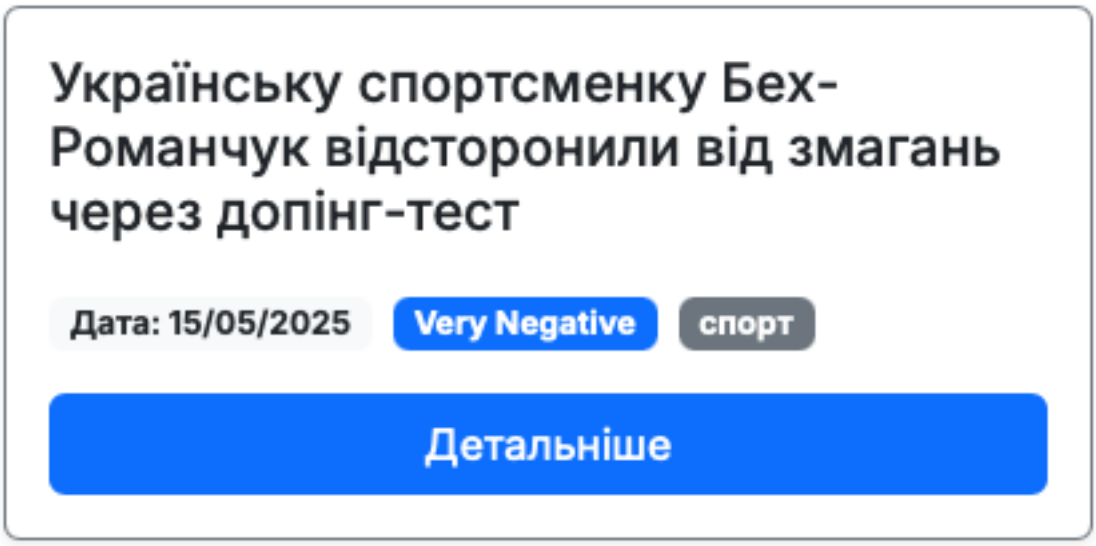

Рисунок 4.18 – Приклад блоку події

Перегляд списку подій. Користувач може прокручувати сторінку, переглядаючи всі виявлені події. Для кожної події відображається короткий опис.

Деталізація події. Коли користувач знаходить у списку подію, яка його зацікавила, він може переглянути деталі цієї події. Для цього потрібно натиснути на кнопку «Детальніше» у відповідного елемента списку подій та інтерфейс завантажить сторінку деталізації. На сторінці деталей події користувач бачить повний перелік новин, що входять до кластеру. Інформація по кожній новині складається із заголовку новини (посилання на оригінал) та точного часу публікації.

Користувач може клікнути на кнопку «Оригінал» в блоці конкретної новини, щоб ознайомитися з повним текстом – при цьому відкриється нова вкладка браузера на оригінальному сайті джерела (рис. 4.19). Також користувач може натиснути на кнопку «Текст новини» й в модальному вікні прочитати текст новини, який був зібраний з джерела.

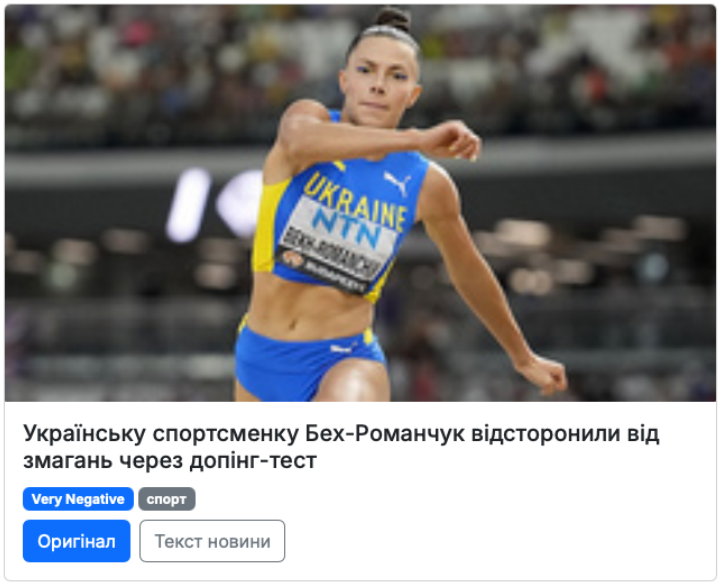

Рисунок 4.19 – Приклад блоку новини

Після перегляду деталей користувач має змогу повернутися до події та загального списку подій.

4.3 Перевірка функцій частин проєкту

Після реалізації всіх компонентів проведено тестування вебзастосунку з метою перевірки відповідності функціональним вимогам та оцінки продуктивності під навантаженням. Тестування включало як перевірку окремих функцій мікросервісів, так і оцінку роботи системи в цілому при обробці певного обсягу даних. У цьому підрозділі представлені результати функціонального тестування, опис перевірки роботи інтерфейсу вручну, а також аналіз навантаження на систему під час виконання завдань AI-пайплайну.

4.3.1 Функціональне тестування REST API

Функціональне тестування серверної частини проведено шляхом виконання серії запитів до REST API та перевірки коректності отриманих відповідей. Для автоматизації та зручності було використано програмний додаток Postman (рис. 4.20 – 4.22).

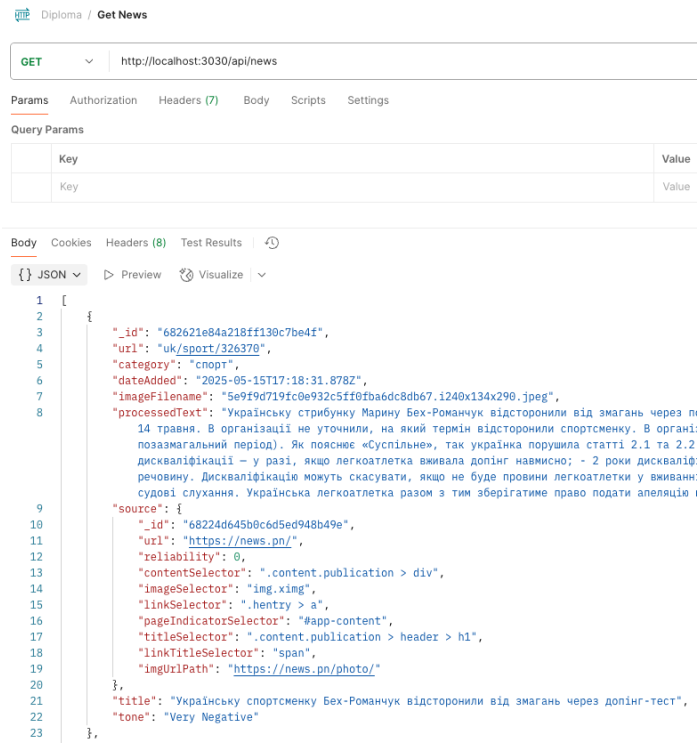

Рисунок 4.20 – Тестування запиту новин за допомогою Postman

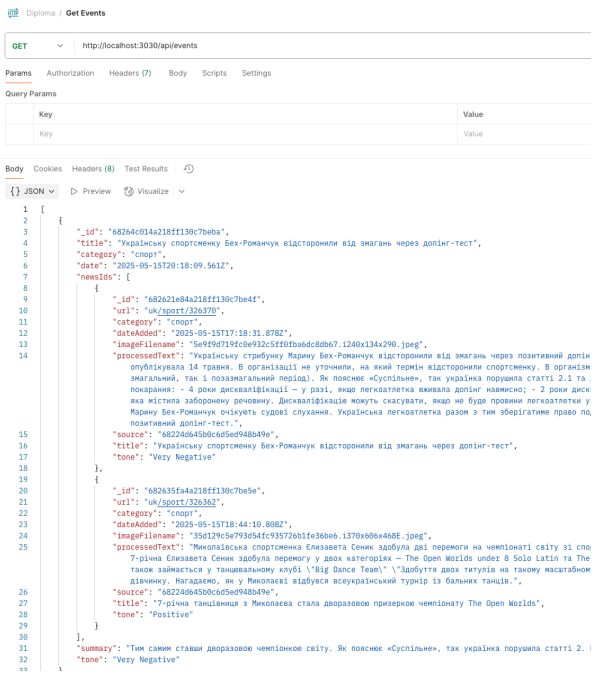

Рисунок 4.21 – Тестування запиту подій за допомогою Postman

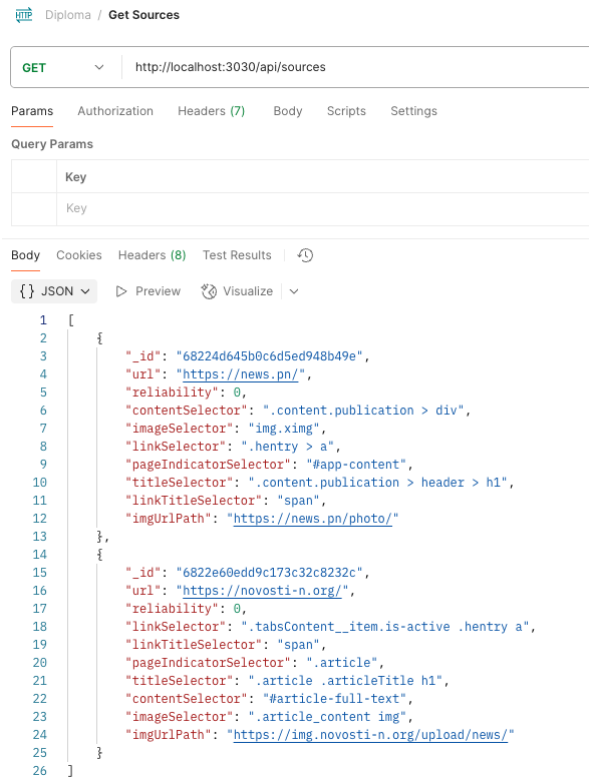

Рисунок 4.22 – Тестування запиту джерел за допомогою Postman

Було створено колекцію запитів, що охоплюють всі основні ендпоїнти API: отримання списку подій, деталей однієї події, списку новин.

Основні протестовані REST-ендпоїнти API включають:

– GET /api/events – отримання переліку всіх подій. Перевірялося, що повертається статус 200 OK і тіло відповіді містить масив об’єктів подій у форматі JSON. Кожен об’єкт події повинен мати поля (id, назва, категорія, тональність, список новин);

– GET /api/events/{eventId} – отримання деталей конкретної події. У відповіді очікувався об’єкт події з розгорнутим списком новин або окремий масив новин цієї події (залежно від реалізації API). Код відповіді 200 OK при валідному eventId. Для неіснуючого або видаленого eventId перевірено, що повертається 404 Not Found з відповідним повідомленням про помилку;

– GET /api/news – отримання списку усіх новин. Цей ендпоїнт використовується рідше, оскільки користувачі переглядають новини через події, але для повноти реалізації він є. Перевірено отримання статусу 200 і наявність масиву новин з полями (id, заголовок, текст, джерело, дата, категорія, тональність);

– GET /api/sources – отримання списку всіх джерел. Очікувано 200 OK і масив об’єктів джерел (id, URL). Якщо джерел немає, повертається порожній масив.

Для кожного тестового запиту зафіксовано фактичний результат і порівняно з очікуваним. Всі ключові функції API спрацювали згідно вимог – сервер повернув правильні статуси та дані.

4.3.2 Перевірка користувацького інтерфейсу (UI-тестування)

Функціональність клієнтської частини перевірялася шляхом ручного тестування основних сценаріїв використання застосунку. Метою було впевнитися, що інтерфейс правильно відображає дані з API, елементи керування працюють, а взаємодія є зручною та без помилок.

Після відкриття головної сторінки додатку перевірено, що завантажується список подій. На тестових даних відображалося 10 подій на першому екрані. Візуально підтверджено, що для кожної події показується назва, дата, категорія та

індикатор тональності. Дані співпадають з наявними у базі (те саме перевірено в API). Сторінка не містить помилок консолі, всі зображення/іконки завантажилися.

Натиснуто на одну з подій у списку. Відкрилася сторінка з детальною інформацією. Перевірено, що заголовок сторінки співпадає з назвою події, присутній підзаголовок та опис, список новин завантажився. Показані всі новини події, кожна з посиланням на оригінал й текстом. На одній з новин натиснуто «Оригінал» – воно коректно відкрило оригінальну статтю в новій вкладці. Повернувшись до застосунку, закрито деталі переходом на головну сторінку. Знову видно список подій. Всі ці переходи відбулися швидко, без перезавантаження сторінки.

На рис. 4.21 продемонстрована статистика завантаження сторінки, піднятої в AWS. Сторінка робить 17 запитів та повністю вантажиться за 4,3 с, демонструючи збалансованість між обсягом даних і часом відображення контенту.

Усі метрики свідчать про стабільний і передбачуваний час завантаження, завдяки розумному розподілу ресурсів. Миттєвий рендер статичного каркасу – HTML і CSS готові за частки секунди. Легка інтерактивність – невеликі JS-бандли дозволяють відразу обробляти події користувача. Актуальні дані – XHR-запити гарантовано доставляють свіже наповнення без додаткових кешованих прошарків. Масштабованість – за потреби можна легко розширити кількість запитів до API, зберігаючи подібний рівень продуктивності.

Цю сторінку ще можна оптимізувати для зменшення обсягу даних та пришвидшення завантаження сорінки. Наприклад, в подальшій розробці проєкту можна винести великі дані з HTML, оптимізувати XHR, кешувати відповіді, додати lazy-loading для відкладеного завантаження окремих компонентів сторінки.

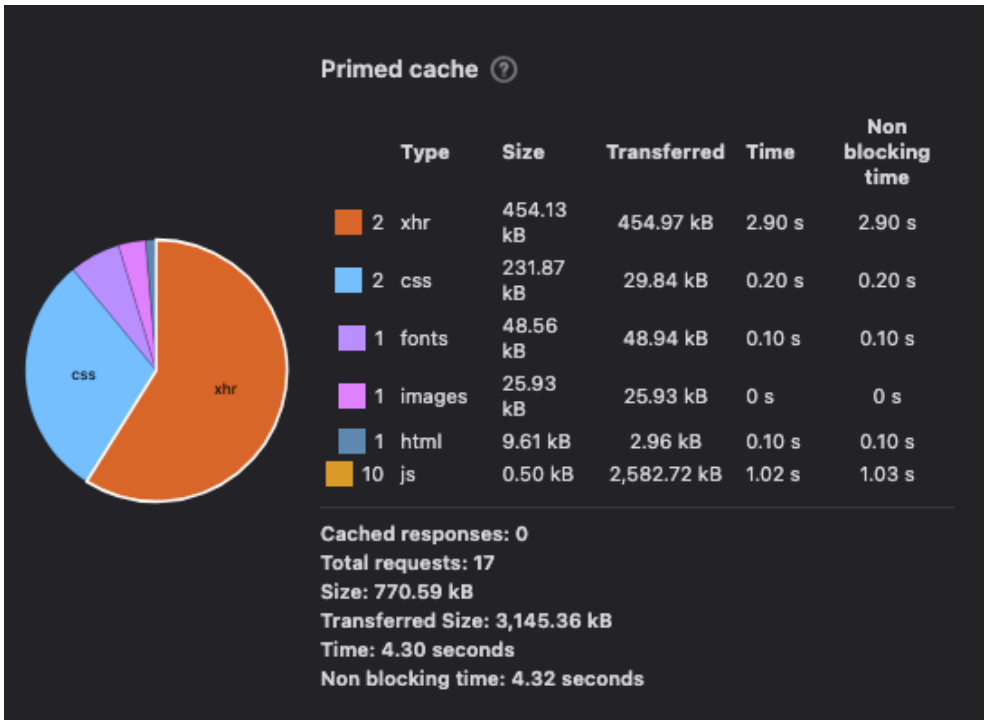

Рисунок 4.21 – Час завантаження та використання пам’яті сторінкою

Протягом тестування інтерфейс залишався стабільним, не виникало критичних помилок або зависань. Перевірено відображення на різних розмірах екрану: адаптивність реалізована базово – на вузькому екрані (мобільному) події теж читабельні, хоча інтерфейс оптимізовано насамперед під десктоп.

В результаті ручного тестування UI встановлено, що всі основні сценарії роботи користувача виконуються правильно. Система дозволяє переглядати всі події та детально ознайомлюватися з конкретною.

Висновки до розділу 4

У четвертому розділі дипломної роботи представлено практичну реалізацію програмного забезпечення для автоматизованого OSINT-моніторингу новин, а також результати його тестування. Було детально розглянуто структуру створеного вебзастосунку: серверна частина забезпечує надання даних через REST API, клієнтський SPA-інтерфейс надає зручні засоби візуалізації і взаємодії, а AI-пайплайн виконує інтелектуальний аналіз контенту (класифікація, аналіз

тональності, кластеризація подій). Продемонстровано, що архітектура, побудована на мікросервісних принципах, успішно розгорнута в хмарному середовищі AWS із застосуванням CI/CD конвеєра.

Результати функціонального тестування підтвердили відповідність реалізованого функціоналу висунутим вимогам: система правильно збирає дані з джерел, групує їх у події, дозволяє фільтрувати та переглядати інформацію користувачу, а також керувати джерелами. Тестування навантаженням показало, що рішення здатне обробляти передбачений обсяг даних із прийнятними витратами ресурсів і має резерв для масштабування. Таким чином, розроблений вебзастосунок відповідає поставленій меті та вимогам: він автоматизує процес OSINT-моніторингу новин, використовуючи методи штучного інтелекту, і надає інструменти для ефективного аналізу зібраної інформації кінцевому користувачу.

ВИСНОВКИ

У кваліфікаційній роботі здійснено повноцінний цикл розробки вебзастосунку для автоматизованого моніторингу відкритих джерел інформації (OSINT), зосередженого на новинному контенті. Метою проєкту стало створення системи, здатної оперативно збирати, класифікувати та узагальнювати великі обсяги новин за допомогою сучасних методів штучного інтелекту, з подальшим представленням результатів у зручному вебінтерфейсі для кінцевого користувача.

На етапі аналізу предметної області було досліджено провідні рішення, такі як GDELT та EventRegistry, що дозволило чітко сформулювати потребу в україноцентричній системі, яка б поєднувала тематичну класифікацію, аналіз тональності та автоматичне згрупування новин у події. Водночас така система мала б залишатися відкритою для модифікації та розширення.

В обґрунтуванні методичного підходу перевагу віддано багатомовним трансформерним моделям. Зокрема, для класифікації використано XLM-RoBERTa в режимі zero-shot, що дозволило обійтися без додаткового навчання на локальному датасеті. Для оцінки тональності застосовано готовий багатомовний pipeline, а для кластеризації – семантичні ембеддинги SentenceTransformer у поєднанні з DBSCAN, що дало змогу об’єднувати новини без задання кількості тем.

Системна архітектура реалізована на базі хмарної інфраструктури AWS. Один EC2-сервер виконує функції API, інший – обробляє ресурсоємні задачі на основі моделей ШІ. Такий розподіл навантаження підвищив стабільність і забезпечив потенціал для масштабування.

Під час проєктування було створено UML-діаграми, які відобразили структуру системи, взаємодію між модулями та логіку обробки даних. Це дозволило чітко окреслити рамки реалізації та уникнути неузгодженостей у коді. У практичному втіленні система складається з трьох основних компонентів: API на Node.js з Express, клієнтська частина на Next.js та окремий Python-модуль, який

реалізує повний цикл ШІ-аналізу – від збору новин до формування коротких висновків.

Функціональність розробки повністю відповідає поставленим у роботі завданням. Система автоматично збирає новини, визначає їхню тематику, тональність, групує їх у події та надає результати у формі інтерактивної аналітики. Завдяки зручному вебінтерфейсу користувач має змогу швидко ознайомитися з ключовими тенденціями та деталями подій, без потреби переглядати великі масиви джерел вручну.

Розробка має практичну цінність і може бути розгорнута в умовах реального середовища. Для повноцінного впровадження достатньо активувати безперервний режим роботи, налаштувати autoscaling для AI-компонентів та розширити список джерел, додавши локальні ЗМІ або Telegram-канали. У подальшому система може бути вдосконалена через використання інкрементальних методів кластеризації та впровадження абстрактивного резюмування для створення узагальнених описів подій.

Робота повністю відповідає сучасним викликам у сфері інформаційної аналітики та безпеки, а також демонструє чіткий вектор розвитку системи з перспективою інтеграції в державні чи комерційні інформаційні сервіси.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. OSINT – розвідка з відкритих джерел та інформаційна безпека. PROMETHEUS. URL: https://prometheus.org.ua/prometheus-free/osint-open-source-intelligence (дата звернення: 01.05.2025).

2. A wikipedia based semantic graph model for topic tracking in blogosphere. Welcome to IJCAI | IJCAI. URL: https://www.ijcai.org/Proceedings/11/Papers/389.pdf (Last accessed: 30.04.2025).

3. The GDELT Project. The GDELT Project. URL: https://www.gdeltproject.org (Last accessed: 30.04.2025).

4. Contributors to Wikimedia projects. GDELT project - wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/GDELT (Last accessed: 30.04.2025).

5. Event Registry - Use the power of AI to turn news content into actionable insights. Event Registry. URL: https://eventregistry.org (Last accessed: 30.04.2025).

6. NewsWhip. Newswhip. URL: https://www.newswhip.com/ (Last accessed: 30.04.2025).

7. Spike. Newswhip. URL: https://www.newswhip.com/spike-real-time-media-monitoring/ (Last accessed: 01.05.2025).

8. Bert-Enhanced text graph neural network for classification. MDPI. URL: https://doi.org/10.3390/e23111536 (Last accessed: 30.04.2025).

9. Key news event detection and event context using graphic convolution, clustering, and summarizing methods. MDPI. URL: https://doi.org/10.3390/app13095510 (Last accessed: 30.04.2025).

10. A clustering method of case-involved news by combining topic network and multi-head attention mechanism. MDPI. URL: https://doi.org/10.3390/s21227501 (Last accessed: 30.04.2025).

11. Evaluation of news sentiment in economic activity forecasting. MDPI. URL: https://doi.org/10.3390/ASEC2022-13790 (Last accessed: 30.04.2025).

12. Explainable sentiment analysis: a hierarchical transformer-based extractive summarization approach. MDPI. URL: https://doi.org/10.3390/electronics10182195 (Last accessed: 30.04.2025).

13. Deep learning-based short text summarization: an integrated BERT and transformer encoder–decoder approach. MDPI. URL: https://doi.org/10.3390/computation13040096 (Last accessed: 30.04.2025).

14. News headline generation based on improved decoder from transformer - Scientific Reports. Nature. URL: https://www.nature.com/articles/s41598-022-15817-z (Last accessed: 30.04.2025).

15. Knowledge graphs representation for event-related e-news articles. MDPI. URL: https://doi.org/10.3390/make3040040 (Last accessed: 30.04.2025).

16. Hugging face – the AI community building the future. Hugging Face – The AI community building the future. URL: https://huggingface.co/ (Last accessed: 16.05.2025).

17. MoritzLaurer/mDeBERTa-v3-base-mnli-xnli. Hugging Face – The AI community building the future. URL: https://huggingface.co/MoritzLaurer/mDeBERTa-v3-base-mnli-xnli (Last accessed: 16.05.2025).

18. Дослідження методів семантичної кластеризації для аналізу новин. Kharkiv National University of Radio Electronics Electronic Archive. URL: https://doi.org/10.30837/IYF.CVSAMM.2024.031 (дата звернення: 01.05.2025).

19. Clustering news articles for topic detection - IRE journals. IRE Journals. URL: https://www.irejournals.com/index.php/paper-details/1700671 (Last accessed: 30.04.2025).

20. Sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2 · hugging face. Hugging Face – The AI community building the future. URL: https://huggingface.co/sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2 (Last accessed: 16.05.2025).

21. IBM. What is NLP (natural language processing)? | IBM. IBM - United States. URL: https://www.ibm.com/think/topics/natural-language-processing (Last accessed: 30.04.2025).

22. Tabularisai/multilingual-sentiment-analysis · hugging face. Hugging Face – The AI community building the future. URL: https://huggingface.co/tabularisai/multilingual-sentiment-analysis (Last accessed: 16.05.2025).

23. Explainer: what is graph analytics? | NVIDIA technical blog. NVIDIA Technical Blog. URL: https://developer.nvidia.com/blog/explainer-what-is-graph-analytics/ (Last accessed: 30.04.2025).

24. Graph analytics. PNNL. URL: https://www.pnnl.gov/explainer-articles/graph-analytics (Last accessed: 30.04.2025).

25. Spot instance - amazon EC2 spot instances - AWS. Amazon Web Services, Inc. URL: https://aws.amazon.com/ec2/spot/ (Last accessed: 30.05.2025).

26. Serverless function, faas serverless - AWS lambda - AWS. Amazon Web Services, Inc. URL: https://aws.amazon.com/lambda/ (Last accessed: 30.05.2025).

27. Amazon S3 - cloud object storage - AWS. Amazon Web Services, Inc. URL: https://aws.amazon.com/s3/ (Last accessed: 30.05.2025).

28. CDN cloud service - amazon cloudfront - AWS. Amazon Web Services, Inc. URL: https://aws.amazon.com/cloudfront/ (Last accessed: 30.05.2025).

29. Node.js – Run JavaScript Everywhere. Node.js – Run JavaScript Everywhere. URL: https://nodejs.org/en (Last accessed: 30.05.2025).

30. Express - Node.js web application framework. Express - Node.js web application framework. URL: https://expressjs.com/ (Last accessed: 30.05.2025).

31. MongoDB: the world’s leading modern database. MongoDB. URL: https://www.mongodb.com/ (Last accessed: 30.05.2025).

32. Next.js by vercel - the react framework. Next.js by Vercel - The React Framework. URL: https://nextjs.org/ (Last accessed: 30.05.2025).

33. Fast and reliable end-to-end testing for modern web apps | Playwright. Fast and reliable end-to-end testing for modern web apps | Playwright. URL: https://playwright.dev/ (Last accessed: 30.05.2025).

34. Cloud development framework - AWS cloud development kit - AWS. Amazon Web Services, Inc. URL: https://aws.amazon.com/cdk/ (Last accessed: 30.05.2025).

ДОДАТОК А

Код вебзастосунку

Лістинг коду scraping.py

```
#!/usr/bin/env python3
from datetime import datetime, timezone

from playwright.async_api import async_playwright
from pymongo import MongoClient
from pymongo.collection import Collection

# Импорт Pydantic-моделей
from models.source import Source
from models.news import News

from modules.text_processing import clean_text
from modules.classification import predict_category
from modules.sentiment_analysis import analyze_sentiment

import os

async def scrape_news() -> None:
    client: MongoClient = MongoClient(os.getenv('MONGO_URI'))
    db = client["osint"]

    source_coll: Collection = db["sources"]
    sources = list(source_coll.find())

    news_coll: Collection = db["news"]

    async with async_playwright() as pw:
        browser = await pw.chromium.launch(headless=True)
        page = await browser.new_page()
        for source in sources:
```

```
            source_id = source["_id"]
            base_url = source["url"]
            link_selector = source.get("linkSelector", ".hentry > a")
            page_indicator_selector  =  source.get("pageIndicatorSelector",
"#app-content")
            title_selector          =          source.get("titleSelector",
".content.publication > header > h1")
            content_selector        =        source.get("contentSelector",
".content.publication > div")
            image_selector = source.get("imageSelector", "img.ximg")
            link_title_sel = source.get("linkTitleSelector", "span")

            print(base_url)
            print('LOG: start of source processing')

            await page.goto(base_url)
            await page.wait_for_selector(link_selector)
            article_elements = await page.query_selector_all(link_selector)
            links = []
            print('LOG: before preparing links')
            for el in article_elements:
                href = await el.get_attribute("href")
                if not href:
                    continue
                if href.startswith("http"):
                    full_url = href
                    relative_url = href.replace(base_url, "", 1).lstrip("/")
                else:
                    relative_url = href.lstrip("/")
                    full_url = base_url.rstrip('/') + '/' + relative_url
                title_list = None
                title_elem = await el.query_selector(link_title_sel)
                if title_elem:
                    title_list = await title_elem.inner_text()
```

```
                # skip if already in DB
                if title_list and news_coll.find_one({"title": title_list,
"source": source_id}):
                    continue
                links.append({"full": full_url, "relative": relative_url,
"title": title_list})

            print('LOG: before processing links')

            for link in links:
                full_url = link['full']
                rel_url = link['relative']
                title_list = link['title']

                print('\r\n\r\n')
                print(title_list)

                await page.goto(full_url)
                await page.wait_for_selector(page_indicator_selector)

                title = title_list
                if not title:
                    title_el = await page.query_selector(title_selector)
                    title = await title_el.inner_text() if title_el else ""

                full_article                    =                    await
page.query_selector(page_indicator_selector)
                text_el                       =                    await
full_article.query_selector(content_selector)

                processed_text = await text_el.inner_text() if text_el else
""
                processed_text = clean_text(processed_text)
```

```
                print('LOG: before classification')
                category = predict_category(title, processed_text)

                print('LOG: before sentiment analyzing')
                tone = analyze_sentiment(title, processed_text)

                img_el = await full_article.query_selector(image_selector)
                img_url = await img_el.get_attribute("src") if img_el else
None
                image_filename = img_url.split("/")[-1] if img_url else None

                print('LOG: before adding to DB')
                item = News(
                    title=title,
                    source=str(source_id),
                    processedText=processed_text,
                    imageFilename=image_filename,
                    url=rel_url,
                    dateAdded=datetime.now(timezone.utc),
                    category=category,
                    tone=tone
                )

                doc = item.model_dump(by_alias=True)
                doc["url"] = rel_url
                doc["source"] = source_id

                news_coll.update_one(
                    {"url": rel_url},
                    {"$set": doc},
                    upsert=True
                )

        await browser.close()
```

Лістинг коду classification.py

```
import torch
from transformers import pipeline, AutoTokenizer,
AutoModelForSequenceClassification

DEVICE = 0 if torch.cuda.is_available() else -1
USE_MPS = torch.backends.mps.is_available()

MODEL_ID = "MoritzLaurer/mDeBERTa-v3-base-mnli-xnli"
_tok = AutoTokenizer.from_pretrained(MODEL_ID, use_fast=True)
_mod = AutoModelForSequenceClassification.from_pretrained(
    MODEL_ID,
    load_in_8bit= not USE_MPS and torch.cuda.is_available(),
    torch_dtype="auto"
)
# set model to eval mode for faster inference
_mod.eval()

classifier = pipeline(
    "zero-shot-classification",
    model=_mod,
    tokenizer=_tok,
    device=DEVICE,
    batch_size=16      # важливо!
)

CANDIDATE_LABELS = [
    "політика", "війна", "економіка", "суспільство", "технології",
    "спорт", "кримінал", "культура", "інше"
]

@torch.inference_mode()
```

```
def predict_category(title: str, text: str) -> str:
    sequence = f"{title}. {text}"
    # run in batch mode for single sequence to reduce overhead
    results = classifier([sequence], candidate_labels=CANDIDATE_LABELS,
top_k=1)
    return results[0]["labels"][0]

@torch.inference_mode()
def predict_category_batch(titles: list[str], texts: list[str]) ->
list[str]:
    sequences = [f"{t}. {x}" for t, x in zip(titles, texts)]
    results = classifier(sequences, candidate_labels=CANDIDATE_LABELS,
top_k=1)
    return [r["labels"][0] for r in results]
```

Лістинг коду clustering.py

```
from pymongo import MongoClient
from datetime import datetime, timedelta
from sklearn.cluster import AgglomerativeClustering
from modules.text_processing import clean_text
from sentence_transformers import SentenceTransformer
import numpy as np
import torch
import os

_embed_model = SentenceTransformer('paraphrase-multilingual-MiniLM-L12-v2')
device = 0 if torch.cuda.is_available() else -1

def cluster_and_summarize(
    mongo_uri: str = os.getenv('MONGO_URI'),
    db_name: str = os.getenv('MONGO_DB_NAME'),
    eps: float = 0.7,
    min_samples: int = 1,
```

```
    lookback_days: int = 7,
    summary_sentences: int = 3
) -> None:
    client = MongoClient(mongo_uri)
    db = client[db_name]
    news_coll = db['news']
    events_coll = db['events']

    cutoff = datetime.utcnow() - timedelta(days=lookback_days)
    recent_news = list(news_coll.find({'dateAdded': {'$gte': cutoff}}))
    if not recent_news:
        return

    texts = [clean_text(f"{n['title']}. {n['processedText']}") for n in
recent_news]
    embeddings = _embed_model.encode(texts, convert_to_numpy=True)

    clustering = AgglomerativeClustering(
        n_clusters=None,
        distance_threshold=eps,
        metric='cosine',
        linkage='average'
    )
    labels = clustering.fit_predict(embeddings)

    clusters = {}
    for item, label in zip(recent_news, labels):
        clusters.setdefault(label, []).append(item)

    for label, items in clusters.items():
        if label == -1:
            continue

        snippet_text = ' '.join(item['processedText'] for item in items[:5])
```

```
        sentences = [s.strip() for s in snippet_text.split('.') if
len(s.split()) > 5]
        sentences = sorted(sentences, key=len)[:summary_sentences]
        extract = '. '.join(sentences)

        summary = extract
        title = items[0]['title']

        tones = [item.get('tone', 'neutral') for item in items]
        categories = [item.get('category', 'general') for item in items]
        modal_tone = max(set(tones), key=tones.count)
        modal_category = max(set(categories), key=categories.count)

        events_coll.update_one(
            {"title": title, 'date': {'$gte': cutoff}},
            {'$set': {
                'newsIds': [item['_id'] for item in items],
                'title': title,
                'summary': summary,
                'date': datetime.utcnow(),
                'tone': modal_tone,
                'category': modal_category
            }},
            upsert=True
        )

if __name__ == '__main__':
    cluster_and_summarize()
```

Лістинг коду sentiment_analysis.py

```
from transformers import pipeline, AutoTokenizer,
AutoModelForSequenceClassification
```

```
sentiment_analyzer = pipeline(
    "text-classification",
    model="tabularisai/multilingual-sentiment-analysis",
    return_all_scores=True
)

def analyze_sentiment(title: str, text: str) -> str:
    """
    'Very Negative', 'Negative', 'Neutral', 'Positive', 'Very Positive'.
    """
    seq = f"{title}. {text}"
    all_scores = sentiment_analyzer(seq, truncation=True, max_length=512)
    scores = all_scores[0]
    best = max(scores, key=lambda x: x["score"])
    return best["label"]
```

Лістинг коду text_processing.py

```
def clean_text(text: str) -> str:
    return ' '.join(text.replace('\n', ' ').split())
```

Лістинг коду event.py

```
from pydantic import BaseModel, ConfigDict
from typing import List
from datetime import datetime

class Event(BaseModel):
    newsIds: List[str]
    title: str
    summary: str
    date: datetime
    tone: str
    category: str
```

```
    reliability: int
```

Лістинг коду news.py

```
from datetime import datetime, timezone
from pydantic import BaseModel, HttpUrl, Field, ConfigDict
from typing import Optional

class News(BaseModel):
    model_config = ConfigDict(populate_by_name=True)

    title: str
    source: str
    processedText: str = Field(..., alias="processedText")
    imageFilename: Optional[str] = Field(None, alias="imageFilename")
    url: str
    dateAdded:        datetime        =        Field(default_factory=lambda:
datetime.now(timezone.utc), alias="dateAdded")
    category: str = Field(..., alias="category")
    tone: str = Field(..., alias="tone")
```

Лістинг коду source.py

```
from pydantic import BaseModel, HttpUrl

class Source(BaseModel):
    url: HttpUrl
    reliability: int
    linkSelector: str
    linkTitleSelector: str
    pageIndicatorSelector: str
    titleSelector: str
    contentSelector: str
    imageSelector: str
    imgUrlPath: str
```

Лістинг коду api.ts

```
import { News, Source, Event } from "./api.interfaces";

// services/api.ts
// const BASE = process.env.NEXT_PUBLIC_API_BASE || 'http://localhost:3000';

async function handle<ResponseType>(path: string): Promise<ResponseType> {
  try {
    const res = await fetch(path);
    if (!res.ok) {
      const text = await res.text().catch(() => res.statusText);
      throw new Error(`API error ${res.status} on ${path}: ${text}`);
    }
    return res.json();
  } catch (err) {
    console.error(`Fetch failed for ${path}:`, err);
    throw new Error(`Network error while fetching ${path}`);
  }
}

export function fetchSources(): Promise<Source[]> {
  return handle<Source[]>('/api/sources');
}

export function fetchNews(): Promise<News[]> {
  return handle<News[]>('/api/news');
}

export function fetchEvents(): Promise<Event[]> {
  return handle<Event[]>('/api/events');
}
```

Лістинг коду EventsSection.tsx

```
'use client';

import { useEffect } from 'react';
import { useAppDispatch, useAppSelector } from '@/store/hooks';
import { fetchEvents } from '@/store/events.slice';
import EventCard from './EventCard';
import { Container, Row, Col, Spinner } from 'react-bootstrap';

export default function EventsSection() {
  const dispatch = useAppDispatch();
  const { items: events, status } = useAppSelector(state => state.events);

  useEffect(() => {
    if (status === 'idle') dispatch(fetchEvents());
  }, [status, dispatch]);

  if (status === 'loading' || status === 'idle') {
    return (
      <Container className="my-5 d-flex justify-content-center">
        <Spinner animation="border" role="status">
          <span className="visually-hidden">Завантаження...</span>
        </Spinner>
      </Container>
    );
  }

  if (status === 'failed') {
    return   <p   className="text-center   my-4">Не   вдалося   завантажити
події.</p>;
  }

  return (
    <Container id="events" className="my-4">
```

```
      <h2 className="mb-4 text-center">Події</h2>
      <Row xs={1} md={2} lg={3} className="g-4">
        {events.map(ev => (
          <Col key={ev._id}>
            <EventCard event={ev} />
          </Col>
        ))}
      </Row>
    </Container>
  );
}
```

Лістинг коду aws/bin/osint.ts

```
// bin/osint.ts
import 'source-map-support/register';
import * as cdk from 'aws-cdk-lib';
import dotenv from 'dotenv';
import { FrontendStack } from '../lib/frontend-stack';
import { BackendStack } from '../lib/backend-stack';
import { ScraperStack } from '../lib/scraper-stack';

// Load .env variables into process.env
dotenv.config();

// Ensure required ENV variables are present
const AWS_ACCOUNT_ID  = process.env.AWS_ACCOUNT_ID!;
const AWS_REGION      = process.env.AWS_REGION!;
const MONGO_URI       = process.env.MONGO_URI!;
const MONGO_DB_NAME       = process.env.MONGO_DB_NAME!;
const DOMAIN_NAME     = process.env.DOMAIN_NAME; // optional

// Initialize CDK app
const app = new cdk.App();
```

```
// Shared AWS env for stacks
const env = {
  account: AWS_ACCOUNT_ID,
  region: AWS_REGION,
};

// Deploy FrontendStack (includes domainName if provided)
new FrontendStack(app, 'FrontendStack', {
  env,
  ...(DOMAIN_NAME ? { domainName: DOMAIN_NAME } : {}),
});

// Deploy BackendStack with Mongo Atlas URI
new BackendStack(app, 'BackendStack', {
  env,
  mongoAtlasUri: MONGO_URI,
});

// Deploy ScraperStack (Lambda + EventBridge) with same Mongo URI
new ScraperStack(app, 'ScraperStack', {
  env,
  mongoAtlasUri: MONGO_URI,
  mongoDbName: MONGO_DB_NAME
});
```

Лістинг коду aws/deploy.sh

```
#!/usr/bin/env bash
set -euo pipefail

# Завантажуємо змінні з .env
if [ -f .env ]; then
  export $(grep -v '^#' .env | xargs)
else
```

```
  echo "❌ File .env not found! Створіть його з необхідними змінними." >&2
  exit 1
fi

. ./scripts/runBackend.sh

. ./scripts/runFrontend.sh

. ./scripts/runNewsProcessing.sh

# 9) Показуємо результати
echo
echo "✅ DEPLOY COMPLETE!"
echo "Backend URL:   $(jq -r .BackendStack.BackendURL backend-outputs.json)"
echo "Frontend CloudFrontDomain: $(jq -r .FrontendStack.CloudFrontDomain frontend-outputs.json)"
echo "Frontend S3WebsiteEndpoint: $(jq -r .FrontendStack.S3WebsiteEndpoint frontend-outputs.json)"
echo "Scraper Lambda ARN: $(jq -r .ScraperStack.ScraperLambdaARN scraper-outputs.json)"
echo "Scraper Code:  s3://$(jq -r .ScraperStack.ScraperCodeBucket scraper-outputs.json)/$(jq -r .ScraperStack.ScraperCodeKey scraper-outputs.json)"
```

Лістинг коду aws/destroy.sh

```
#!/usr/bin/env bash
set -euo pipefail

# 1) Видалити стекі
npx cdk destroy ScraperStack --force
npx cdk destroy FrontendStack --force
npx cdk destroy BackendStack --force
```