Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет Комп'ютерні технології та системи
Кафедра Комп'ютерні інформаційні технології

Пояснювальна записка

до кваліфікаційної роботи
бакалавра

на тему: «Розробка програмного забезпечення агрегації інформації з сайтів новин»
за освітньою програмою **12 Інженерія програмного забезпечення**
зі спеціальності: **121 Інженерія програмного забезпечення**

Виконав: студент групи ПЗ1911: (підпис) /Володимир МАКСИМЧУ

Керівник: (підпис) (посада) /Вадим АНДРЮЩЕНКО/

Нормоконтролер: (підпис) (посада) /Світлана ВОЛКОВА /

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

Ministry of Education and Science of Ukraine

Ukrainian State University of Science and Technologies

Faculty Computer technologies and systems
Department Computer information technology

Explanatory Note

to Bachelor's Thesis
bachelor

on the topic: «Development of software for aggregating information from news sites»

according to educational curriculum **12 software engineering**
in the Speciality: **121 software engineering**

Done by the student of the group PZ1911: ___ _____ /Volodymyr MAKSYMCHUK/
(посада) (підпис)

Scientific Supervisor: ___ _____ /Vadym ANDRIUSHCHENKO/
(посада) (підпис)

Normative controller: ___ _____ /Svitlana VOLKOVA/
(посада) (підпис)

Dnipro – 2023

Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет: Комп'ютерних технологій і систем
Кафедра: Комп’ютерні інформаційні технології
Рівень вищої освіти: магістр
Освітня програма: Інженерія програмного забезпечення
Спеціальність: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Вадим ГОРЯЧКІН
_____ червня 2023 р.

ЗАВДАННЯ

На кваліфікаційну роботу ____Бакалавр_____

Студенту Максимчуку Володимиру Сергійовичу_____

1. Тема дипломної роботи: Розробка програмного забезпечення агрегації інформації з сайтів новин.

   Керівник роботи: Андрющенко Вадим Олександрович, доцент

   затверджені наказом 1196 ст від 05.12. 2022 року

2. Строк подання студентом роботи 19.06. 2023 року

3. Вихідні дані до дипломної роботи: _____.

4. Зміст пояснювальної записки (перелік питань до розробки):
   4.1. Аналіз предметної галузі та її актуальність;
   4.2. Розробка застосунку агрегації новин;
   4.3. Огляд застосунку;
   4.4. Тестування.

5. Перелік демонстраційного матеріалу:
   5.1. доповідь;
   5.2. презентація;
   5.3. демонстраційне відео.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів	Примітка
1	Збір та аналіз вимог	01.03.23 – 10.03.23	10%
2	Зовнішнє проектування	10.03.23 – 20.03.23	20%
3	Внутрішнє проектування	20.03.23 – 05.04.23	30%
4	Розробка серверної частини застосунку	05.04.23 – 25.04.23	40%
5	Розробка клієнтської частини застосунку	25.04.23 – 05.05.23	50%
6	Тестування застосунку	05.05.23 – 15.05.23	60%
7	Рефакторинг програмного коду	15.05.23 – 20.05.23	70%
8	Розробка демонстраційних матеріалів	20.05.23 – 07.06.23	80%
9	Подання кваліфікаційної роботи до кафедри	19.06.23	90%
10	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	26.06.23	100%

Студент _____ /Володимир МАКСИМЧУК/

Керівник роботи _____ /Вадим АНДРЮЩЕНКО/

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра:

143 с., 17 рис., 2 дод., 15 джерела.

Предмет розробки – програмне забезпечення, яке здатне збирати, обробляти та презентувати новинні дані з різних джерел.

Мета роботи – створення програмного забезпечення, яке дозволить зручно та ефективно агрегувати інформацію з різних новинних сайтів. Розроблене програмне забезпечення надає користувачам можливість швидкого та зручного доступу до актуальних новин, збираючи та структуруючи дані з різних джерел.

Сферою застосування є поліпшення доступу до новинних даних для користувачів, які бажають отримувати інформацію з різних джерел в одному зручному місці.

Методи дослідження – аналіз сучасних методів збору та обробки новинних даних, дослідження різних програмних інструментів та технологій, розробка та тестування програмного забезпечення.

Ключові слова: АГРЕГАЦІЯ ІНФОРМАЦІЇ, ДАНІ, МОВИ ПРОГРАМУВАННЯ, НОВИННІ САЙТИ, РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ТЕХНОЛОГІЇ.

.

ЗМІСТ

Перелік умовних позначень, символів, одиниць скорочень і термінів 8
Вступ 9
1 Аналіз предметної галузі та її актуальність 11
1.1 Загальний огляд агрегації новин 11
1.1.1 Визначення агрегації новин 11
1.1.2 Важливість та переваги агрегації новин 12
1.1.3 Виклики та обмеження агрегації новин 13
1.2 Існуючі методи агрегації новин 14
1.2.1 Традиційні методи агрегації новин 14
1.2.2 Автоматизовані підходи до агрегації новин 15
1.2.3 Порівняння різних методів агрегації новин 16
2 Розробка застосунку агрегації новин 17
2.1 Використання трирівневої архітектури 17
2.2 Огляд обраних технологій 25
2.2.1 .NET Core MVC 26
2.2.2. AngleSharp 28
2.2.3 PostgreSQL 29
2.3 База даних 30
2.3.1 Опис моделей 32
2.3.2 Огляд реалізації GenericRepository та UnitOfWork 38
2.4 Головний сервіс ContentAggregator 40
2.4.1 BackgroundNewsAggregator 41
2.5 Огляд головних інтерфейсів та класів 42
2.5.1 INewsSource інтерфейс та його реалізації 42
2.5.2 INewsParser інтерфейс та його реалізації 44

2.6 HomeController 45
3 Огляд застосунку 47
3.1 Головна сторінка 47
3.2 Новини за темами 50
3.3 Сторінка статті 51
4 Тестування 54
4.1 Визначення тестування 54
4.2 Види тестування 55
4.3 Важливість тестування у розробці 57
4.4 Сценарії ручного тестування клієнтської частини 59
4.5 Шляхи розвитку тестування застосунку 65
Висновки 66
Перелік джерел посилання 68
Додатки 70

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ СКОРОЧЕНЬ І ТЕРМІНІВ

Агрегація – процес збору, накопичення та об'єднання даних або інформації з декількох джерел в єдину форму;

БД – база даних;

AngleSharp – бібліотека для парсингу та обробки HTML-документів;

CSS – Cascading Style Sheets – мова стилів, яка використовується для задання зовнішнього вигляду веб-документів;

API – Application Programming Interface – це інтерфейс програмування застосунків, який дозволяє різним додаткам та сервісам обмінюватись інформацією і функціональністю;

ASP .NET Core – Active Server Pages Network Enabled Technologies – це відкрите програмне забезпечення, що дозволяє розробникам створювати веб-додатки за допомогою мов програмування, таких як C# і F# та інші;

C# – C Sharp – це об'єктно-орієнтована мова програмування. Її використовують для розробки різноманітних додатків, від додатків для настільних комп'ютерів до веб-додатків;

HTML – HyperText Markup Language – мова розмітки гіпертексту, яка використовується для створення структури та візуального представлення веб-сторінок;

PostgreSQL – це система управління базами даних з відкритим кодом, що пропонує високу швидкодію та надійність.

ВСТУП

У сучасному світі, що швидко змінюється, важливість новин неможливо переоцінити. Новини слугують життєво важливим джерелом інформації, інформуючи людей про поточні події, розвиток суспільства та глобальні проблеми. Вони відіграють вирішальну роль у формуванні громадської думки, впливають на процеси прийняття рішень і сприяють вихованню поінформованих громадян.

В епоху, позначену технологічним прогресом і поширенням цифрових медіа, споживання новин зазнало значних змін. Традиційні джерела новин, такі як газети і телебачення, були доповнені, а в деяких випадках і перевершені новинними інтернет-сайтами. Ці цифрові платформи пропонують велику кількість інформації, що охоплює широкий спектр тем і точок зору. Однак велика кількість джерел новин і величезний обсяг доступної інформації можуть бути приголомшливими. Споживачі новин часто змушені орієнтуватися в потоці статей, намагаючись відокремити фактичні репортажі від коментарів, а також шукати надійні джерела серед шуму дезінформації та фейкових новин. Саме тут роль програмного забезпечення для агрегації інформації з новинних сайтів стає першорядною.

Консолідуючи новинний контент з різних джерел на централізованій платформі, програмне забезпечення для агрегації новин надає користувачам більш ефективний і впорядкований підхід до отримання інформації. Воно дозволяє людям отримати доступ до широкого спектру новинних статей, аналітичних матеріалів і думок з різних точок зору, і все це в одному місці. Така зручність економить час, зменшує інформаційне перевантаження і дозволяє користувачам отримати комплексне уявлення про поточні події.

Стрімкий розвиток цифрових медіа та інтернету призвів до вибуху інформації, доступної через новинні веб-сайти. З незліченними джерелами і постійним припливом новинних статей залишатися в курсі подій стає дедалі складніше. У відповідь на це інформаційне перевантаження важливим рішенням

стала розробка програмного забезпечення для агрегації інформації з новинних сайтів. Ця кваліфікаційна робота присвячена розробці програмного забезпечення, призначеного для збору, обробки та представлення новинних даних з різних джерел у консолідованому та легкодоступному форматі. Мета полягає в тому, щоб надати користувачам комплексний і спрощений спосіб залишатися в курсі останніх новин з різних джерел, усуваючи необхідність вручну переглядати кілька веб-сайтів.

Щоб осягнути значення цієї розробки, необхідно зрозуміти поточний стан проблеми. Люди, які шукають новини, стикаються з важким завданням відвідувати численні веб-сайти, прокручувати численні статті і стикатися з надлишковою або суперечливою інформацією. Цей процес забирає багато часу і часто призводить до інформаційного перевантаження або пропущених оновлень. Потреба в ефективному та централізованому рішенні для агрегування новинного контенту стає все більш очевидною. Під час аналізу існуючого ландшафту було досліджено різні аналоги та підходи до агрегації новин. Деякі популярні платформи та додатки пропонують базові функції агрегації новин, але їм часто бракує кастомізації, всебічного охоплення та ефективної обробки даних. Це створює прогалину на ринку для складного програмного рішення, яке усуває недоліки існуючих платформ.

Крім того, при спробі розробити ефективне програмне забезпечення для агрегації новин виникають технічні протиріччя. Виклики включають обробку великого обсягу даних з різних джерел, забезпечення точності та релевантності даних, впровадження ефективних механізмів пошуку та фільтрації, а також подання інформації у зручний для користувача спосіб. Ці технічні перешкоди вимагають інноваційних підходів та надійних алгоритмів для їх успішного подолання. З огляду на сучасний стан проблеми, метою даної кваліфікаційної роботи є проектування, розробка та оцінка програмної системи для агрегації інформації з новинних сайтів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ЇЇ АКТУАЛЬНІСТЬ

1.1 Загальний огляд агрегації новин

1.1.1 Визначення агрегації новин

Агрегація новин – це процес систематичного збору, організації та представлення новинного контенту з різних джерел в уніфікованому та доступному форматі. Він передбачає збір інформації з різних новинних веб-сайтів, блогів, соціальних мереж, RSS-каналів та інших джерел і консолідацію їх на одній платформі або в одному додатку. По суті, агрегація новин має на меті спростити користувачам процес споживання новин шляхом надання централізованого центру, де вони можуть отримати доступ до широкого спектру новинних статей, заголовків, резюме та іншого пов'язаного з ними контенту. Замість того, щоб відвідувати кілька веб-сайтів або покладатися на окремі джерела новин, користувачі можуть покладатися на платформи агрегації новин, які курують і надають різноманітний новинний контент, адаптований до їхніх уподобань. Процес агрегації новин зазвичай складається з кількох ключових етапів [6], [11].

По-перше, здійснюється збір даних, де використовуються автоматизовані алгоритми або API (інтерфейси прикладного програмування) для вилучення новинних статей та супутньої інформації з різних джерел. Процес збору даних може ґрунтуватися на заздалегідь визначених критеріях, таких як конкретні ключові слова, категорії або вподобання користувачів. Після того, як дані зібрані, вони проходять етап попередньої обробки, де їх очищають, стандартизують і структурують, щоб забезпечити узгодженість і сумісність у межах платформи агрегації. Ця попередня обробка може включати такі завдання, як нормалізація тексту, видалення дублікатів статей, категоризація контенту за відповідними темами або розділами, а також призначення метаданих, таких як часові мітки, автори та джерела.

Після попередньої обробки агрегований новинний контент організовується і представляється користувачам у зручному інтерфейсі. Цей інтерфейс може включати такі функції, як персоналізовані стрічки новин, функції пошуку, тематичні фільтри, рекомендовані статті, трендові історії та сповіщення про останні новини. Мета полягає в тому, щоб надати користувачам безперебійний та інтуїтивно зрозумілий досвід, який дозволить їм легко орієнтуватися, знаходити та споживати новинний контент, що їх цікавить.

1.1.2 Важливість та переваги агрегації новин

Агрегація новин має величезне значення в сучасну цифрову епоху, надаючи численні переваги, які задовольняють зростаючі потреби споживачів новин. Давайте заглибимося в те, чому агрегація новин є важливою і які переваги вона пропонує.

По-перше, агрегація новин дозволяє людям отримати доступ до різноманітних точок зору та джерел. Замість того, щоб покладатися на один новинний ресурс або бути обмеженими певною редакційною упередженістю, користувачі можуть вивчати контент з різних джерел, включаючи основні ЗМІ, незалежних журналістів і спеціалізовані видання. Таке ознайомлення з різними точками зору сприяє більш повному розумінню складних питань, заохочує критичне мислення та допомагає людям розвивати всебічний погляд на світ.

По-друге, агрегація новин економить дорогоцінний час і підвищує ефективність споживання новин. У сучасному швидкоплинному світі, де перевантаження інформацією є загальною проблемою, платформи агрегації новин слугують зручним універсальним місцем для доступу до новинного контенту. Замість того, щоб відвідувати кілька веб-сайтів чи додатків, користувачі можуть покластися на централізовану платформу, яка курує та організовує новинні статті, заголовки та резюме з різних джерел. Такий спрощений підхід усуває потребу в тривалому пошуку, дозволяючи користувачам швидко сканувати та засвоювати потрібну інформацію [3].

Крім того, агрегація новин пропонує персоналізований перегляд новин, пристосований до індивідуальних уподобань. Користувачі можуть налаштовувати свої стрічки новин на основі конкретних тем, що їх цікавлять, джерел, яким надають перевагу, або навіть впливових журналістів. Така кастомізація гарантує, що користувачі отримують новинний контент, який відповідає їхнім уподобанням, сприяючи більш цікавому та релевантному споживанню новин. Функції персоналізації також дозволяють користувачам відкривати для себе нові джерела і теми, пов'язані з їхніми інтересами, розширюючи свої знання і знайомлячись з різними темами.

Агрегація новин також допомагає вирішити проблему інформаційного перевантаження. З ростом кількості джерел новин і великої кількості доступного контенту, людям може бути важко фільтрувати і визначати пріоритети новин, які вони хочуть споживати. Платформи агрегації новин полегшують цю проблему, куруючи і представляючи контент у стислий і організований спосіб. Пропонуючи різноманітний вибір новинних статей в одному місці, ці платформи дозволяють користувачам залишатися в курсі широкого кола тем без необхідності переходити на різні веб-сайти чи додатки. Крім того, агрегація новин покращує доступність для пошуку, висвітлюючи трендові історії та надаючи рекомендації, засновані на інтересах і звичках користувачів до читання. Ця функція відкриває користувачам доступ до новинних статей і тем, на які вони могли б не натрапити в інший спосіб, сприяючи несподіваному навчанню та розширенню їхніх горизонтів знань.

1.1.3 Виклики та обмеження агрегації новин

Агрегація новин, попри її переваги, також має свої виклики та обмеження. Забезпечення достовірності та якості інформації є значним викликом через велику кількість джерел новин. Фільтрація дезінформації та упередженого контенту вимагає надійних алгоритмів і процесів перевірки фактів. Аспект персоналізації при агрегації новин може призвести до ефекту «бульбашки

фільтрів», коли користувачі бачать лише той контент, який відповідає їхнім уподобанням, що посилює упередженість. Баланс між персоналізацією та різноманітністю контенту має вирішальне значення [13].

Іншою проблемою є потенційний брак контексту та глибини в агрегованому контенті. Зосередженість на стислості може призвести до того, що користувачі втратять важливу довідкову інформацію, всебічний аналіз і нюанси точок зору. Досягнення балансу між наданням стислих оновлень і доступом до більш детальної інформації має важливе значення для всебічного розуміння новинних тем. При агрегації новин також виникають правові та етичні міркування. Необхідно враховувати порушення авторських прав та прав інтелектуальної власності, а також отримувати належні дозволи, якщо це необхідно. Необхідно запровадити етичні практики пошуку джерел та атрибуції, особливо при використанні контенту з невеликих видань.

1.2 Існуючі методи агрегації новин

1.2.1 Традиційні методи агрегації новин

Традиційні методи агрегації новин передбачають ручну курацію редакторами або журналістами, які відбирають і компілюють новинні статті з різних джерел. Ці кураторські колекції часто класифікуються за темами або релевантністю, щоб забезпечити всебічний огляд поточних подій. Традиційні методи агрегації новин покладаються на людські судження та досвід, щоб забезпечити відбір якісного та релевантного контенту. Однак такий підхід може забирати багато часу і бути обмеженим у масштабуванні, особливо з огляду на величезну кількість новин, що з'являються щодня.

1.2.2 Автоматизовані підходи до агрегації новин

Автоматизовані підходи до агрегації новин використовують передові алгоритми та штучний інтелект для оптимізації процесу збору, фільтрації та представлення новинного контенту. Одним із ключових методів автоматизованої агрегації новин є веб-скрепінг, який передбачає вилучення інформації з кількох онлайн-джерел. Це усуває необхідність ручної курації з боку редакторів або журналістів. Алгоритми, що використовуються в автоматизованих новинних агрегаторах, аналізують різні фактори, щоб визначити релевантність, популярність і своєчасність новинних статей. Враховуючи ці фактори, алгоритми можуть ефективно ранжувати та визначати пріоритетність новинного контенту для показу користувачам. Наприклад, новинний агрегатор може надавати більшого значення останнім новинам або трендовим темам, гарантуючи, що користувачі отримають найсвіжішу інформацію [2], [8].

Автоматизація дозволяє агрегаторам новин ефективно обробляти великі обсяги новинного контенту. Зважаючи на постійний потік новинних статей, що публікуються в Інтернеті, вручну курувати та оновлювати платформу агрегації новин у режимі реального часу було б майже неможливо. Автоматизовані підходи вирішують цю проблему шляхом постійного моніторингу джерел новин і своєчасного оновлення контенту. Оновлення в режимі реального часу є значною перевагою автоматизованої агрегації новин. Користувачі можуть отримати доступ до найсвіжіших новинних статей і бути в курсі поточних подій у міру їх розгортання. Така оперативність особливо важлива в сучасному швидкоплинному світі, де новини постійно змінюються. Крім того, автоматизовані методи агрегації новин можуть запропонувати користувачам можливості для кастомізації. Аналізуючи вподобання користувачів, їхню поведінку та попередні взаємодії, алгоритми можуть персоналізувати новинний контент, що надається кожній людині. Такий рівень персоналізації покращує користувацький досвід і гарантує, що користувачі отримують новини, які відповідають їхнім інтересам і потребам.

1.2.3 Порівняння різних методів агрегації новин

Існує широкий спектр методів агрегації новин, кожен з яких має свої переваги та недоліки. Вибір методу залежить від таких факторів, як цільова аудиторія, бажаний рівень кастомізації та технічні можливості платформи. Деякі методи надають пріоритет персоналізації, використовуючи вподобання та поведінку користувача для адаптації рекомендацій новин. Інші зосереджуються на наданні різноманітних точок зору шляхом включення контенту з різних джерел. Крім того, алгоритми, що використовуються для ранжування та фільтрації новинних статей, відрізняються на різних платформах, що впливає на якість та релевантність агрегованого контенту.

Порівняння різних методів агрегації новин передбачає оцінку таких факторів, як точність, своєчасність, різноманітність контенту, зручність для користувачів і здатність адаптуватися до мінливого новинного ландшафту. При виборі найбільш підходящого методу важливо враховувати конкретні вимоги та цілі платформи агрегації новин. Таким чином, існуючі методи агрегації новин охоплюють як традиційні, так і автоматизовані підходи. Традиційні методи покладаються на ручну курацію, тоді як автоматизовані методи використовують алгоритми та штучний інтелект. Порівняння цих методів передбачає аналіз таких факторів, як персоналізація, різноманітність контенту та алгоритмічне ранжування. Вибір відповідного методу залежить від конкретних цілей і вимог платформи агрегації новин. Розуміючи та оцінюючи наявні методи, розробники та зацікавлені сторони можуть приймати обґрунтовані рішення для створення ефективних систем агрегації новин, які задовольняють потреби своїх користувачів.

2 РОЗРОБКА ЗАСТОСУНКУ АГРЕГАЦІЇ НОВИН

2.1 Використання трирівневої архітектури

У цьому розділі ми розглянемо використання трирівневої архітектури при розробці програмного забезпечення для агрегації інформації з новинних сайтів. Трирівнева архітектура, також відома як архітектура Presentation-Logic-Business Logic-Data Access Layer (PL-BLL-DAL), є широко прийнятим шаблоном проектування в розробці програмного забезпечення, який розділяє різні компоненти програми на окремі шари. Цей архітектурний підхід пропонує кілька переваг з точки зору зручності обслуговування, масштабованості та організації коду [1].

По суті, трирівнева архітектура розділяє додаток на три логічні рівні: рівень представлення (PL), рівень бізнес-логіки (BLL) і рівень доступу до даних (DAL). Кожен рівень має свої специфічні обов'язки і взаємодіє з іншими у чітко визначений спосіб, сприяючи модульності та розподілу завдань.

Рівень представлення, також відомий як рівень інтерфейсу користувача, відповідає за обробку взаємодії з користувачем і відображення інформації користувачеві. На цьому рівні користувач взаємодіє з програмним забезпеченням, надаючи вхідні дані та отримуючи вихідні. Він охоплює такі компоненти, як користувацькі інтерфейси, форми та елементи керування. Рівень представлення фокусується на представленні даних і фіксуванні вводу користувача, делегуючи обробку цих даних нижчим рівням [10].

Рівень бізнес-логіки є ядром програми, що містить бізнес-правила, алгоритми та робочі процеси, які керують поведінкою програмного забезпечення. Він інкапсулює логіку та операції, специфічні для області застосування програми. Рівень BLL отримує вхідні дані від рівня представлення, обробляє їх і видає бажані результати. Він виконує такі завдання, як перевірка даних, обчислення та оркестрування операцій. Цей рівень забезпечує узгодженість і цілісність даних, впроваджує бізнес-правила і застосовує заходи безпеки.

Рівень доступу до даних відповідає за управління взаємодією з базовими системами зберігання даних, такими як бази даних або зовнішні API. Він надає методи та функціональні можливості для отримання, зберігання та маніпулювання даними. Рівень DAL абстрагується від конкретних деталей зберігання даних і надає уніфікований інтерфейс для взаємодії рівня BLL з даними. Він виконує такі завдання, як запити до баз даних, виконання операцій з даними та перетворення даних. Інкапсуляція логіки доступу до даних в окремий рівень робить додаток більш гнучким і дозволяє легко переключатися між різними джерелами даних або адаптуватися до змін в базовій інфраструктурі даних.

Трирівнева архітектура сприяє модульності, повторному використанню та підтримці програмного забезпечення. Кожен рівень можна розробляти, тестувати та підтримувати незалежно, що полегшує управління кодом та майбутні вдосконалення. Розділення завдань гарантує, що зміни, внесені в одному шарі, не впливають на інші, зменшуючи ризик непередбачуваних наслідків. Крім того, ця архітектура уможливлює паралельну розробку різними командами, які одночасно працюють над різними рівнями. Крім того, трирівнева архітектура підтримує масштабованість та оптимізацію продуктивності. Шари можуть бути розгорнуті на різних серверах або розподілені між декількома машинами, що дозволяє балансувати навантаження і підвищити продуктивність. Відокремлення рівня презентації від рівнів бізнес-логіки та доступу до даних дозволяє додатку працювати з великою кількістю одночасних користувачів без шкоди для продуктивності [5].

У контексті розробки програмного забезпечення для агрегації інформації з новинних сайтів трирівнева архітектура має кілька переваг. Рівень представлення може зосередитися на створенні зручного інтерфейсу для взаємодії користувачів і налаштування їхніх уподобань щодо новин. Рівень бізнес-логіки може обробляти складні алгоритми та робочі процеси, необхідні для агрегування та аналізу новинних даних, забезпечуючи точні та релевантні

результати. Рівень доступу до даних може керувати пошуком новинних статей з різних джерел, здійснювати перетворення даних і кешування для ефективного пошуку.

2.1.1 Огляд структури проекту

В процесі розробки застосунку були створенні три проекти, а саме NewsAggregator.DAL, NewsAggregator.BLL, NewsAggregator.PL. Почнемо огляд з первого проекту, а саме NewsAggregator.DAL (рисунок 2.1).

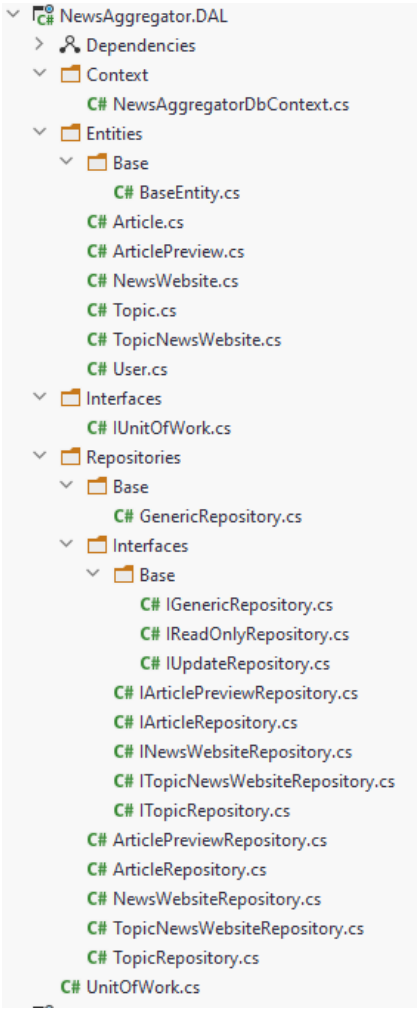

Рисунок 2.1 – Структура проекту NewsAggregator.DAL

У цьому проекті є папка під назвою «DAL», що розшифровується як «рівень доступу до даних». Ця папка містить компоненти, які відповідають за доступ до даних у додатку. У папці DAL є декілька підпапок та файлів. Однією з підпапок є «Context». Ця папка містить файли, пов'язані з контекстом бази даних. Зокрема, у нас є файл під назвою «NewsAggregatorDbContext.cs», який визначає контекст бази даних для програми Агрегатор новин. Цей клас контексту відповідає за взаємодію з базовою базою даних та керування сутностями.

Далі є підпапка «Entities». Ця папка містить класи сутностей, які представляють таблиці в базі даних. У папці Entities ми маємо підпапку під назвою «Base», яка містить файл з назвою «BaseEntity.cs». Цей файл визначає базовий клас сутності, від якого успадковуються інші класи сутностей. Він забезпечує спільні властивості та функціональність, які використовуються всіма сутностями. Крім того, у папці Entities ми маємо окремі файли сутностей, такі як «Article.cs», «ArticlePreviews.cs», «NewsWebsite.cs», «Topic.cs», «TopicNews Website.cs» і «User.cs». Кожен з цих файлів представляє певну сутність в моделі даних додатку.

Далі ми переходимо до підпапки «Interfaces». Тут ми знаходимо файл «IUnitOfWork.cs», який визначає інтерфейс для патерну Unit of Work. Цей патерн допомагає керувати транзакціями та координувати операції з даними у різних сховищах.

Також є підпапка «Repositories». Ця папка містить класи, що відповідають за доступ до даних та реалізують патерн сховища. У папці Repositories є підпапка «Base», яка містить базові класи та інтерфейси, пов'язані зі сховищем даних. До них відносяться такі файли, як «IGenericRepository.cs», «IReadOnlyRepository.cs» та «IUpdateRepository.cs», які визначають інтерфейси для загальних операцій зі сховищем.

Крім того, у нас є окремі файли сховищ, такі як «IArticlePreviewRepository.cs», «ITopicsNewsWebsiteRepository.cs» «IArticleRepository.cs», «INewsWebsiteRepository.cs», і «ITopicRepository.cs», які

представляють конкретні сховища сутностей. Ці класи сховищ надають методи для взаємодії з відповідними сутностями та виконання операцій з даними.

В кінці ми маємо файл «UnitOfWork», який знаходиться поза підпапкою. Цей файл представляє реалізацію патерну Unit of Work і координує операції з даними у різних сховищах.

Далі розглянемо проект NewsAggregator.BLL (рисунок 2.2).

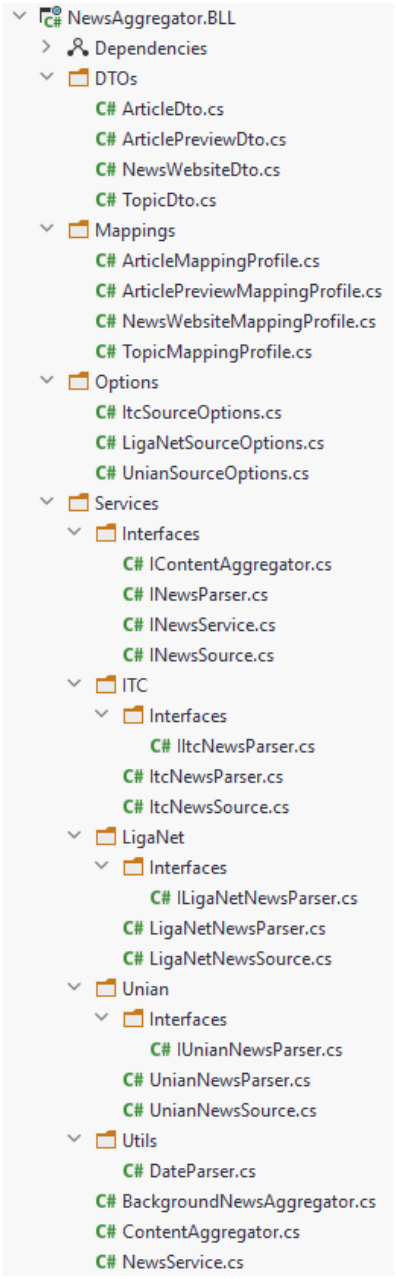

Рисунок 2.2 – Структура проекту NewsAggregator.BLL

Починаючи з підпапки «DTOs», він містить класи об'єктів передачі даних (DTO). Ці класи слугують контейнерами для передачі даних між різними рівнями програми. У цьому випадку ми маємо такі файли, як «ArticleDto.cs», «ArticlePreviewDto.cs», «NewsWebsiteDto.cs» і «TopicDto.cs», які визначають DTO для конкретних об'єктів у додатку.

Перейдемо до підпапки «Mappings», яка містить класи профілів відображення. Ці класи визначають конфігурацію відображення між класами сутностей і відповідними їм класами DTO. Наприклад, у нас є файли «ArticleMappingProfile.cs», «ArticlePreviewMappingProfile.cs», «NewsWebsiteMappingProfile.cs» і «TopicMappingProfile.cs», які керують відображенням між сутностями і DTO.

Далі у нас є підпапка «Options». Ця папка містить класи, які визначають опції або налаштування для певних функцій або сервісів у додатку. Наприклад, ми бачимо файли «ItcSourceOptions.cs», «LigaNetSourceOptions.cs» і «UnianSourceOptions.cs», які зберігають параметри конфігурації для різних джерел новин. Заглибившись у підпапку «Services», ми знайдемо підпапку «Interfaces». Тут ми маємо інтерфейси, які визначають контракти для різних сервісів у додатку. Ці інтерфейси включають «IContentAggregator.cs», «INewsParser.cs», «INewsService.cs» і «INewsSource.cs», які забезпечують абстракцію і визначають операції або функціональні можливості, очікувані від відповідних реалізацій сервісів.

У підпапці «Services» ми маємо підпапки для різних джерел новин. Наприклад, у підтеці «ITC» ми знаходимо ще одну підтеку «Interfaces» з інтерфейсом «IItcNewsParser.cs». Крім того, ми маємо файли «ItcNewsParser.cs» та «ItcNewsSource.cs», які представляють реалізацію парсеру новин та джерела новин спеціально для ITC.

Аналогічно, ми маємо підпапки та файли для джерел новин «ЛігаНет» та «Уніан». Кожна підпапка містить інтерфейси для парсингу новин та відповідні реалізації. Наприклад, «ILigaNetNewsParser.cs», «LigaNetNewsParser.cs», «LigaNetNewsSource.cs» для джерела новин LigaNet. Крім того, у нас є підпапка

«Utils», яка містить утиліти або допоміжні класи. У цій папці знаходиться файл «DateParser.cs», який забезпечує функціональність для розбору дат.

Нарешті, безпосередньо у папці «Services» знаходяться файли «BackgroundNewsAggregator.cs» та «NewsService.cs». Ці файли представляють собою реалізацію агрегатора фонових новин та основного сервісу новин відповідно.

Тепер розглянемо останній проект NewsAggregator.PL (рисунок 2.3).

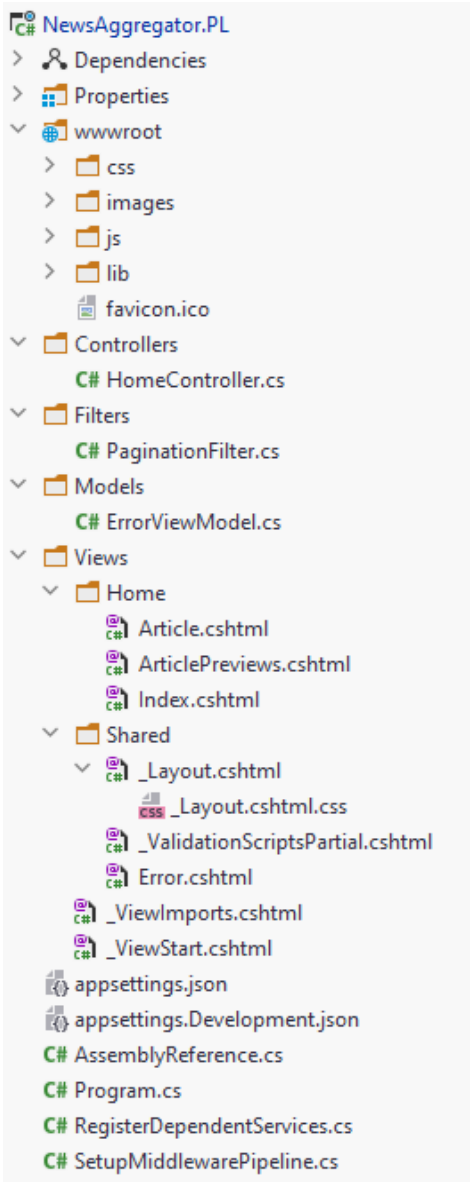

Рисунок 2.3 – Структура проекту NewsAggregator.PL

Цей проект розшифровується як Presentation Layer. Ця папка містить компоненти, пов'язані з користувацьким інтерфейсом та презентацією додатку. У проекті NewsAggregator.PL ми маємо декілька підпапок та файлів. Починаючи з папки «wwwroot», ця папка зазвичай містить статичні файли, такі як таблиці стилів CSS, файли JavaScript та зображення, які обслуговують клієнтську частину програми. Перейдемо до папки «Controllers», яка містить класи контролерів, що відповідають за обробку запитів користувача та координацію потоку даних між користувацьким інтерфейсом та іншими рівнями додатку. У цьому випадку ми маємо файл під назвою «HomeController.cs», який представляє контролер для головних сторінок додатку.

Далі у нас є папка «Filters». Ця папка містить класи фільтрів, які можуть бути застосовані до контролерів або дій для виконання пре- або пост-обробки запитів або відповідей. Наприклад, ми бачимо файл з назвою «PaginationFilter.cs», який може обробляє функціональність, пов'язану з пагінацією.

Продовжуючи з папки «Models», ця папка зазвичай містить класи моделей, які представляють структури даних, що використовуються в шарі представлення додатку. У цьому випадку ми маємо файл під назвою «ErrorViewModel.cs», який може визначати клас моделі для збору та представлення інформації, пов'язаної з помилками.

Перейдемо до папки «Views», в якій зберігаються подання, що визначають шаблони інтерфейсу користувача або макети, які відображаються додатком. У папці «Views» ми маємо підпапки, що відповідають різним контролерам. Наприклад, у підпапці «Home» є такі файли, як «Article.cshtml», «ArticlePreviews.cshtml» та «Index.cshtml», які представляють подання, пов'язані з конкретними діями HomeController. Крім того, у папці «Views» є підпапка «Shared». Ця підпапка містить подання або часткові подання, які є спільними для декількох контролерів або дій. Наприклад, ми можемо знайти файл з назвою «_Layout.cshtml», який представляє шаблон макета, спільний для різних подань.

Перейшовши до кореня проекту, можна побачити такі файли, як «Program.cs», який зазвичай містить точку входу для програми, і «RegisterDependentServices.cs», який обробляє реєстрацію залежних сервісів або залежностей у шарі представлення та інших проектах.

Нарешті, ми маємо файл «SetupMiddlewarePipeline.cs», який відповідає за конфігурацію та налаштування конвеєра проміжного програмного забезпечення додатку. Цей файл визначає послідовність компонентів проміжного програмного забезпечення, які обробляють запити та відповіді, забезпечуючи різні функціональні можливості та поведінку.

2.2 Огляд обраних технологій

Вибір правильних технологій для проекту агрегатора новин має вирішальне значення для забезпечення його успіху та ефективності. Правильний вибір технологій відіграє значну роль у визначенні функціональності, продуктивності та масштабованості проекту. У цифровому середовищі, що швидко розвивається, де попит на новини та інформацію в режимі реального часу досягає свого піку, правильний вибір технології може мати вирішальне значення.

Однією з головних причин ретельного вибору технологій є забезпечення відповідності агрегатора новин функціональним вимогам проекту. Кожна технологія має свій власний набір функцій і можливостей, які можуть значно розширити функціональність агрегатора. Наприклад, використання бібліотек веб-скрепінгу або API може сприяти безперешкодному вилученню даних з різних новинних сайтів, гарантуючи, що найсвіжіші статті та оновлення будуть отримуватись в режимі реального часу. Для аналізу змісту новинних статей можна використовувати інструменти обробки природної мови, що уможливлює такі функції, як аналіз настроїв, класифікацію тем і вилучення ключових слів. Ці функції не лише покращують користувацький досвід, але й надають цінну інформацію для користувачів новинного агрегатора.

Іншим важливим аспектом є продуктивність новинного агрегатора. Оскільки щодня генерується і споживається величезна кількість новинного контенту, стек технологій повинен бути здатним ефективно обробляти великі обсяги даних. Вибір технологій, які відомі своєю оптимізацією продуктивності, може гарантувати, що новинний агрегатор зможе обробляти і відображати новини швидко і безперебійно. Використання механізмів кешування, ефективних систем баз даних та оптимізації методів пошуку даних може значно підвищити швидкість та оперативність роботи програми. Масштабованість – ще один важливий аспект. Зі зростанням користувацької бази агрегатора новин стек технологій повинен мати можливість масштабуватися відповідно до зростаючого трафіку та обробки даних. Вибір технологій, відомих своєю масштабованістю та надійністю, таких як хмарні обчислювальні платформи, розподілені системи або архітектура мікросервісів, може гарантувати, що новинний агрегатор зможе впоратися з великою кількістю одночасних користувачів без шкоди для продуктивності.

Для того, щоб розробити надійний та ефективний агрегатор новин, ретельний розгляд був приділений вибору відповідних технологій. Після ретельної оцінки було визначено, що .NET Core MVC, AngleSharp та PostgreSQL будуть основними технологіями, що використовуються в цьому проекті.

2.2.1 .NET Core MVC

.NET Core MVC – це потужний фреймворк веб-додатків, який є частиною платформи .NET Core. Він надає розробникам гнучкий та ефективний спосіб створення динамічних і багатофункціональних веб-додатків. MVC розшифровується як Model-View-Controller (Модель-Вигляд-Контролер) – це шаблон проектування, який розділяє логіку додатку на три окремі компоненти [4].

Модель представляє дані та бізнес-логіку додатку. Вона інкапсулює стан додатку і надає методи для маніпулювання даними та доступу до них. У контексті новинного агрегатора модель виконує такі завдання, як отримання новинних статей з різних джерел, організація їх у відповідні структури даних і виконання будь-яких необхідних перетворень або перевірок даних [12].

Представлення (View) відповідає за рівень представлення додатку. Він визначає, як дані відображаються і показуються користувачеві. У випадку агрегатора новин, представлення визначатиме макет, форматування та візуальне представлення агрегованого новинного контенту. Це дозволяє створювати зручні та візуально привабливі інтерфейси, забезпечуючи цікавий та інтуїтивно зрозумілий користувачеві досвід.

Контролер виступає посередником між моделлю та поданням. Він отримує запити користувача, обробляє їх і взаємодіє з моделлю для отримання або оновлення даних. Потім контролер передає дані відповідному представленню для рендерингу. У контексті агрегатора новин контролер виконує такі завдання, як отримання запитів користувачів, координація пошуку відповідних новинних статей, застосування необхідних фільтрів або сортування, а також передача відфільтрованих даних до представлення для відображення.

.NET Core MVC пропонує ряд функцій та переваг, які сприяють його популярності серед розробників. Він має модульну та розширювану архітектуру, що дозволяє розробникам легко підключати додаткові компоненти або налаштовувати існуючі відповідно до конкретних вимог проекту. Така гнучкість дозволяє створювати масштабовані та підтримувані веб-додатки. Крім того, .NET Core MVC включає вбудовану підтримку основних функцій веб-розробки, таких як маршрутизація, перевірка даних та обробка форм. Це спрощує реалізацію поширених шаблонів веб-додатків і зменшує кількість шаблонного коду, який потрібно писати розробникам. Це прискорює процес розробки та сприяє повторному використанню коду.

.NET Core MVC легко інтегрується з іншими компонентами екосистеми .NET. Він використовує можливості мови програмування C# та надає доступ до

широкого спектру бібліотек і фреймворків, що сприяє швидкій розробці додатків.

Рішення використовувати .NET Core MVC як фреймворк для веб-додатків зумовлене його універсальністю, продуктивністю та широкою підтримкою спільноти розробників. .NET Core MVC забезпечує міцну основу для створення масштабованих і модульних веб-додатків, дозволяючи безперешкодно інтегрувати різні компоненти. Гнучка архітектура та багатий набір функцій роблять його ідеальним вибором для реалізації користувацького інтерфейсу, обробки запитів та управління потоком даних у новинному агрегаторі.

2.2.2. AngleSharp

AngleSharp – це потужна та універсальна бібліотека для синтаксичного аналізу та маніпулювання HTML-документами для .NET-розробників. Вона надає повний набір функцій та API, які спрощують процес парсингу та маніпулювання HTML-документами, що робить її чудовим вибором для завдань веб-скрепінгу, вилучення даних та контент-аналізу. Однією з ключових переваг AngleSharp є його здатність обробляти складні структури HTML і неправильно сформовані документи. У ньому реалізовано надійний парсер HTML5, який може обробляти різні типи вхідних даних, включаючи невірний або погано відформатований HTML. Це робить його дуже пристосованим до реальних сценаріїв, коли веб-сторінки можуть не відповідати стандартам HTML.

За допомогою AngleSharp розробники можуть легко переміщатися і запитувати HTML-документи, використовуючи знайомий синтаксис CSS-селекторів. Це дозволяє точно і гнучко вибирати елементи HTML, що полегшує вилучення певних даних з веб-сторінок. Ця можливість є особливо цінною в контексті агрегації новин, оскільки дозволяє цілеспрямовано знаходити статті, заголовки, резюме, дати публікацій та іншу необхідну інформацію з різних джерел. AngleSharp також надає потужні можливості маніпулювання, дозволяючи розробникам змінювати, додавати або видаляти елементи, атрибути

та вміст HTML-документів. Така гнучкість дає змогу налаштовувати і трансформувати вилучені дані відповідно до конкретних вимог програми. Наприклад, розробники можуть нормалізувати текст, форматувати дані або перекладати витягнутий новинний контент [7].

AngleSharp буде використовуватися як бібліотека для парсингу та скрапінгу HTML. Завдяки своєму комплексному парсеру HTML5 і селектору CSS AngleSharp спрощує вилучення релевантної інформації з новинних сайтів. Він дозволяє систематично знаходити статті, метадані та інші релевантні деталі, забезпечуючи точне агрегування новинного контенту. Простота використання AngleSharp та обширна документація роблять його ефективним інструментом для аналізу та маніпулювання HTML, що в кінцевому підсумку підвищує загальну функціональність новинного агрегатора [15].

2.2.3 PostgreSQL

PostgreSQL – це популярна реляційна система управління базами даних з відкритим вихідним кодом, відома своєю стійкістю, надійністю та розширеними можливостями. Вона забезпечує масштабоване та ефективне рішення для зберігання, управління та запитів до структурованих даних, що робить її чудовим вибором для широкого спектру додатків, включаючи новинні агрегатори. Однією з ключових переваг PostgreSQL є її відповідність стандартам SQL. Вона забезпечує всебічну підтримку SQL-запитів, дозволяючи розробникам використовувати свої знання та навички SQL при роботі з базою даних. Це полегшує написання складних запитів для отримання та маніпулювання даними, включаючи агрегування новинних статей, фільтрацію контенту на основі певних критеріїв та виконання аналітичних операцій. PostgreSQL пропонує багатий набір вбудованих типів даних, включаючи числові, рядкові, дати/часу та JSON. Ці типи даних дозволяють зберігати та маніпулювати різними типами даних, забезпечуючи гнучкість і точність при роботі з інформацією, пов'язаною з новинами. Наприклад, тип даних JSON в PostgreSQL особливо корисний для

обробки структурованих даних, таких як статті новин, дозволяючи ефективно зберігати і знаходити вміст, закодований у форматі JSON [9].

PostgreSQL також забезпечує підтримку транзакцій і контроль паралелізму, гарантуючи цілісність даних і дозволяючи декільком користувачам одночасно отримувати доступ до бази даних і змінювати її. Ця можливість особливо важлива для новинних агрегаторів, які обробляють великі обсяги даних і потребують одночасного доступу від декількох користувачів. Забезпечуючи цілісність транзакцій і керуючи одночасним доступом, PostgreSQL гарантує узгодженість даних і усуває конфлікти [14].

Для зберігання та управління зібраними даними було обрано систему управління базами даних PostgreSQL. PostgreSQL пропонує низку розширених функцій, включаючи відповідність стандарту ACID, підтримку складних типів даних і потужний механізм оптимізації запитів. Вона забезпечує надійність, масштабованість і продуктивність, що робить її чудовим вибором для зберігання величезної кількості новинних даних, зібраних з різних джерел. Крім того, сумісність PostgreSQL з .NET Core та потужна підтримка спільноти забезпечують безперешкодну інтеграцію та ефективне управління даними в новинному агрегаторі.

2.3 База даних

Правильна розробка зв'язків і таблиць бази даних має вирішальне значення для успіху будь-якого проекту, в тому числі й новинного агрегатора. Це забезпечує ефективне зберігання, пошук та управління даними. Створюючи чітко визначені відносини між таблицями, ми можемо встановити необхідні зв'язки та залежності, що дозволяє безперешкодно інтегрувати дані та маніпулювати ними. Крім того, правильне структурування таблиць допомагає оптимізувати продуктивність запитів, забезпечує цілісність даних і масштабованість. Добре розроблена схема бази даних закладає основу для надійної та гнучкої системи, полегшуючи майбутні вдосконалення та

модифікації. Зрештою, правильна розробка зв'язків і таблиць бази даних забезпечує надійність, продуктивність і ремонтопридатність програми-агрегатора новин.

База даних для проекту агрегатора новин складається з декількох таблиць, які слугують певним цілям в організації та зберіганні інформації. До цих таблиць відносяться:

– NewsWebsites: ця таблиця зберігає інформацію про різні новинні веб-сайти, наприклад, їхні назви та основні URL-адреси. Це допомагає керувати та ідентифікувати різні джерела новин;

– ArticlePreviews: ця таблиця містить узагальнену інформацію про статті новин, включно з їхніми назвами, текстами попереднього перегляду, датами і пов'язаними з ними URL-адресами. Вона надає користувачам швидкий огляд статей;

– Articles: ця таблиця містить повну інформацію про статті новин, включно з їхніми назвами, HTML-текстами, авторами, датами публікації та пов'язаними з ними URL-адресами. Вона зберігає повний вміст для доступу і читання користувачами;

– Topics: ця таблиця зберігає інформацію про різні теми або категорії новинних статей. Вона допомагає класифікувати і впорядковувати статті на основі їхньої тематики;

– TopicsNewsWebsites: ця таблиця встановлює зв'язки між темами і новинними веб-сайтами. Вона допомагає зіставити новинні веб-сайти з певними темами, дозволяючи користувачам переглядати новинні статті на основі їхніх інтересів.

Ці таблиці є важливими для проекту агрегатора новин, оскільки вони забезпечують структурований спосіб зберігання, управління та пошуку інформації, пов'язаної з новинами. Вони забезпечують ефективний пошук, категоризацію та представлення новинних статей користувачам, покращуючи їхній досвід і надаючи їм релевантну та актуальну інформацію з різних джерел. ER-діаграма бази даних зображена нижче (рисунок 2.4).

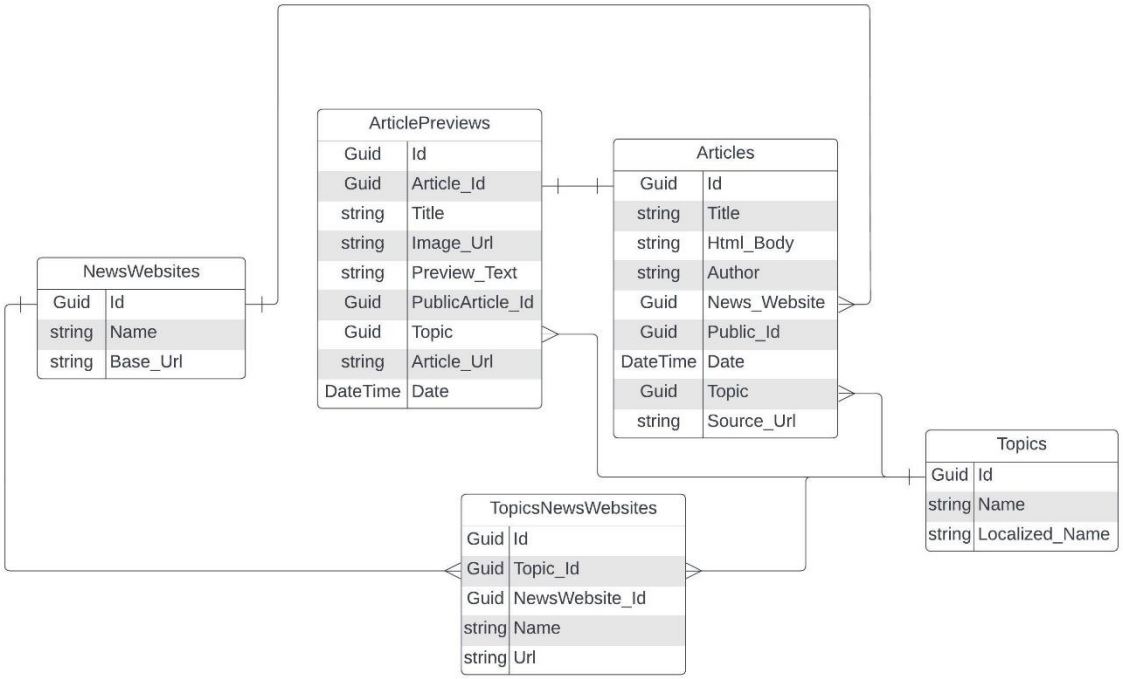

Рисунок 2.4 – ER-діаграма бази даних

2.3.1 Опис моделей

У додатку агрегатору новин були розробдені кілька сутностей для представлення різних компонентів системи. Ці сутності включають в себе:

– BaseEntity.cs: це базовий клас сутності, який надає спільні властивості та функціональність для всіх інших сутностей у додатку;

– Article.cs: ця сутність представляє статтю новин і включає такі властивості, як назва, зміст, автор, дата публікації та URL-адреса джерела;

– ArticlePreviews.cs: ця сутність містить узагальнену інформацію про новинні статті, таку як заголовок, текст попереднього перегляду, дату публікації та посилання на повну версію статті;

– NewsWebsite.cs: ця сутність представляє новинний веб-сайт і містить такі властивості, як назва та основна URL-адреса. Вона допомагає керувати та ідентифікувати різні джерела новин;

– Topic.cs: ця сутність представляє тему або категорію новинних статей. Допомагає класифікувати і впорядковувати статті на основі їхньої тематики;

– TopicNewsWebsite.cs: ця сутність встановлює зв'язки між темами і новинними веб-сайтами, дозволяючи користувачам переглядати новини на основі їхніх інтересів.

Ці сутності формують основну структуру нашого додатку, забезпечуючи ефективне зберігання, пошук і керування даними, пов'язаними з новинами.

Розглянемо кожну модель більш детальніше. Є базова абстрактна модель BaseEntity (рисунок 2.5).

```
public abstract class BaseEntity
{
    38 usages
    public Guid Id { get; set; }
}
```

Рисунок 2.5 – Програмний код моделі BaseEntity

Наведений код представляє абстрактний клас з назвою «BaseEntity», який слугує основою для інших сутностей у додатку. Нижче наведено опис моделі:

– клас має єдину властивість з іменем «Id» типу «Guid». Тип даних «Guid» представляє глобально унікальний ідентифікатор;

– властивість «Id» має як геттер, так і сеттер, що дозволяє іншим класам, похідним від BaseEntity, отримувати доступ до цієї властивості та змінювати її;

– призначенням властивості «Id» є надання унікального ідентифікатора для кожної сутності, похідної від BaseEntity. Цей ідентифікатор можна використовувати для різних цілей, таких як операції з базами даних, ідентифікація об'єктів або посилання на пов'язані сутності.

Визначаючи BaseEntity як абстрактний клас, він забезпечує загальну структуру і функціональність, яку можна використовувати для різних сутностей у додатку. Успадкування від BaseEntity дозволяє похідним класам мати доступ

до властивості «Id» та будь-якої додаткової функціональності або властивостей, визначених у базовому класі.

Розглянемо модель Article (рисунок 2.6).

```
public class Article : BaseEntity
{
    2 usages
    public string Title { get; set; } = null!;
    2 usages
    public string HtmlBody { get; set; } = null!;
    2 usages
    public string? Author { get; set; } = null!;
    3 usages
    public DateTime Date { get; set; }
    2 usages
    public Guid PublicId { get; set; }
    3 usages
    public Guid NewsWebsite { get; set; }
    3 usages
    public Guid Topic { get; set; }
    2 usages
    public string SourceUrl { get; set; } = null!;
}
```

Рисунок 2.6 – Програмний код моделі Article

Клас «Article» – це модель, яка представляє новинну статтю. Він успадковується від класу «BaseEntity», який надає унікальний ідентифікатор для кожної статті. Нижче наведено огляд властивостей класу «Article»:

– «Title»: відображає заголовок статті;

– «HtmlBody»: містить HTML-вміст статті;

– «Author»: відображає автора статті. Ця властивість може бути нульовою, що вказує на те, що вона може мати або не мати значення;

– «Date»: відображає дату створення статті;

– «PublicId»: забезпечує унікальний ідентифікатор для статті в публічному контексті;

– «NewsWebsite»: ідентифікує новинний веб-сайт, з якого походить стаття;

– «Topic»: представляє тему або категорію, до якої належить стаття;

– «SourceUrl»: містить URL-адресу або посилання на джерело статті.

Ці властивості визначають основні атрибути статті і дозволяють програмі зберігати і отримувати інформацію про окремі статті новин.

Далі розглянемо модель ArticlePreview (рисунок 2.7).

```
public class ArticlePreview : BaseEntity
{
    3 usages
    public Guid ArticleId { get; set; }
    3 usages
    public string Title { get; set; } = null!;
    3 usages
    public string? ImageUrl { get; set; }
    3 usages
    public string? PreviewText { get; set; }
    3 usages
    public Guid PublicArticleId { get; set; }
    5 usages
    public Guid Topic { get; set; }
    7 usages
    public string ArticleUrl { get; set; } = null!;
    5 usages
    public DateTime Date { get; set; }
}
```

Рисунок 2.7 – Програмний код моделі ArticlePreview

Клас «ArticlePreview» є ще однією моделлю в додатку. Нижче наведено огляд його властивостей:

– «ArticleId»: представляє ідентифікатор пов'язаної статті;

– «Title»: містить заголовок попереднього перегляду статті;

– «ImageUrl»: відображає URL-адресу зображення, пов'язаного з прев'ю статті. Ця властивість може бути нульовою, що означає, що вона може мати або не мати значення;

– «PreviewText»: містить короткий попередній перегляд або анотацію статті. Ця властивість може бути нульовою;

– «PublicArticleId»: відображає публічний ідентифікатор статті в певному контексті;

– «Topic»: відображає тему або категорію, до якої належить прев'ю статті;

– «ArticleUrl»: містить URL-адресу або посилання на джерело повного тексту статті;

– «Data»: відображає дату попереднього перегляду статті.

Клас «ArticlePreview» дозволяє програмі зберігати і отримувати узагальнену інформацію про статті, таку як їхні назви, тексти попереднього перегляду, пов'язані з ними зображення і деталі публікації. Він надає зручний спосіб представити користувачам скорочену версію статті, дозволяючи їм вирішити, чи хочуть вони отримати доступ до повного змісту, перейшовши за наданою URL-адресою статті.

Тепер розглянемо модель NewsWebsite (рисунок 2.8).

```
public class NewsWebsite : BaseEntity
{
    11 usages
    public string Name { get; set; } = null!;
    3 usages
    public string BaseUrl { get; set; } = null!;
}
```

Рисунок 2.8 – Програмний код моделі NewsWebsite

Клас «NewsWebsite» представляє у додатку сутність веб-сайту новин. Ось короткий опис його властивостей:

– «Name»: представляє ім'я або назву веб-сайту новин;

– «BaseUrl»: містить базову URL-адресу веб-сайту новин.

Клас «NewsWebsite» дозволяє програмі зберігати інформацію про різні новинні веб-сайти. Він надає властивості для зберігання назви та базової URL-адреси кожного веб-сайту. Ця інформація може бути використана для ідентифікації та посилання на конкретні новинні веб-сайти під час агрегування та пошуку новинних статей.

Розглянемо модель Topic (рисунок 2.9).

```
public class Topic : BaseEntity
{
    11 usages
    public string Name { get; set; } = null!;
    9 usages
    public string LocalizedName { get; set; } = null!;
}
```

Рисунок 2.9 – Програмний код моделі Topic

Клас «Topic» – це модель, що представляє сутність теми у додатку. Нижче наведено огляд його властивостей:

– «Name»: відображає назву теми;

– «LocalizedName»: містить локалізовану назву теми.

Клас «Topic» дозволяє програмі визначати та керувати різними темами або категоріями для новинних статей. Це використано для фільтрації та відображення статей на основі певних тем, покращуючи користувацький досвід і надаючи цільовий контент користувачам, які цікавляться певними темами.

Розглянемо модель TopicNewsWebsite (рисунок 2.10).

```
public class TopicNewsWebsite : BaseEntity
{
    18 usages
    public Guid TopicId { get; set; }
    18 usages
    public Guid NewsWebsiteId { get; set; }
    14 usages
    public string Name { get; set; } = null!;
    17 usages
    public string Url { get; set; } = null!;
}
```

Рисунок 2.10 – Програмний код моделі TopicNewsWebsite

Клас «TopicNewsWebsite» представляє зв'язок між темами та новинними сайтами у додатку. Ось короткий огляд його властивостей:

– «TopicId»: представляє унікальний ідентифікатор теми, пов'язаної з новинним веб-сайтом;

– «NewsWebsiteId»: унікальний ідентифікатор веб-сайту новин, пов'язаного з темою;

– «Title»: містить ім'я або назву зв'язку між темою та новинним веб-сайтом;

– «Url»: містить URL-адресу, пов'язану зі зв'язком між темою і новинним веб-сайтом.

Клас «TopicNewsWebsite» дозволяє програмі встановлювати зв'язки між певними темами і новинними веб-сайтами. Він надає властивості для зберігання ідентифікаторів теми і новинного веб-сайту, що беруть участь у зв'язку, а також додаткову інформацію, таку як назва і URL-адреса. Ця інформація може бути використана для зв'язування тем з відповідними новинними веб-сайтами, уможливлюючи агрегування та представлення новинних статей, специфічних для певних тем, з визначених джерел.

2.3.2 Огляд реалізації GenericRepository та UnitOfWork

Використання універсального репозиторію дає кілька переваг при розробці програмного забезпечення. Він сприяє повторному використанню коду і спрощує доступ до даних, надаючи загальний інтерфейс для взаємодії з різними об'єктами в додатку. За допомогою універсального репозиторію розробники можуть писати загальні методи, які можна використовувати для різних об'єктів без необхідності повторювати код. Такий підхід також покращує зручність супроводу, оскільки зміни до базового рівня доступу до даних можна вносити централізовано. Крім того, загальний репозиторій сприяє створенню більш модульної та масштабованої архітектури, що полегшує тестування та майбутнє розширення програми.

У застосунку був реалізований універсальний репозиторій. Повний код репозиторію поданий додатку А.

Клас «GenericRepository<TEntity, TContext>» – це гнучка і багаторазова реалізація загального сховища у програмному додатку. Він призначений для забезпечення спільного набору операцій для доступу до даних та маніпулювання ними для різних типів сутностей. Клас приймає два загальні параметри, «TEntity» та «TContext», які представляють тип сутності та тип контексту бази даних відповідно. Він реалізує інтерфейс «IGenericRepository<TEntity, TContext>», що забезпечує дотримання попередньо визначеного контракту.

При створенні клас отримує екземпляр контексту бази даних через свій конструктор, який використовується для доступу до базового сховища даних. Поле «DbSet<TEntity>» «_entities» ініціалізується набором сутностей типу «TEntity» у наданому контексті. Клас пропонує різноманітні методи для виконання типових операцій з даними. Він включає методи для додавання окремих сутностей, додавання декількох сутностей, видалення сутностей та оновлення існуючих сутностей. Крім того, він надає методи для пошуку сутностей за їхнім унікальним ідентифікатором, запиту сутностей на основі певних критеріїв і пошуку всіх сутностей заданого типу.

Клас використовує можливості асинхронного програмування, пропонуючи відповідні асинхронні версії своїх методів для підтримки неблокуючих операцій. Ці асинхронні методи приймають параметр «CancellationToken», що дозволяє краще контролювати довготривалі або скасовані операції.

Універсальний репозиторій підтримує як швидке, так і ліниве завантаження. Метод «GetAllAsNoTracking()» отримує всі сутності без відстеження змін, що може бути корисним у сценаріях, де достатньо доступу лише для читання або для оптимізації продуктивності.

2.4 Головний сервіс ContentAggregator

Головним сервісом у застосунку є ContentAggregator. Код сервісу поданий у додатку Б.

Клас «ContentAggregator» відповідає за агрегацію та збір новинного контенту з різних джерел до бази даних додатку. Він реалізує інтерфейс «IContentAggregator», який визначає функціонал агрегації. Під час реалізації клас отримує декілька залежностей через свій конструктор. Ці залежності включають сховища для статей, прев'ю статей, тем, новинних сайтів, зв'язки між темами та новинними сайтами, а також блок роботи для управління транзакціями в базі даних. Крім того, він отримує колекцію джерел новин, опції для конкретних джерел новин, маппер для відображення об'єктів і логгер для ведення журналу.

Основним методом у цьому класі є метод «AggregateAsync», який виконує власне процес агрегації. Він приймає параметр «CancellationToken» для підтримки скасування операції у разі необхідності. У методі «AggregateAsync» клас ітераційно переглядає кожне джерело новин і знаходить відповідний новинний веб-сайт зі сховища. Потім він продовжує ітерацію по кожній темі, перевіряючи, чи тема пов'язана з новинним веб-сайтом через репозиторій «тема-новинний веб-сайт». Якщо ні, програма переходить до наступної теми.

Для кожної теми, пов'язаної з новинним веб-сайтом, клас отримує статті, пов'язані з цією темою, зі сховища статей на основі певних критеріїв, таких як ідентифікатор новинного веб-сайту та ідентифікатор теми. Він визначає початкову і кінцеву дати для отримання статей на основі назви веб-сайту новин і попередньо визначених параметрів. Якщо для теми знайдено статті, клас отримує як статті, так і їхні прев'ю з джерела новин за допомогою методу «FetchArticlesAsync». Він застосовує певну логіку фільтрації та обробки, щоб забезпечити отримання унікальних статей і прев'ю статей.

Нарешті, клас додає окремі статті та прев'ю до відповідних сховищ за допомогою методу «AddRangeAsync». Після того, як всі статті та прев'ю статей додано, зміни фіксуються в базі даних за допомогою блоку роботи. У разі

виникнення будь-яких помилок під час процесу агрегації, клас реєструє повідомлення про критичну помилку за допомогою наданого логгера. Загалом, клас «ContentAggregator» відіграє важливу роль у отриманні новинного контенту з різних джерел, забезпеченні узгодженості даних та збереженні агрегованого контенту в базі даних додатку для подальшої обробки та споживання.

2.4.1 BackgroundNewsAggregator

Для автоматизації процесу агрегації був розроблений сервіс BackgroundNewsAggregator (додаток Б). Клас «BackgroundNewsAggregator» відповідає за виконання агрегації фонових новин через певні проміжки часу. Він реалізує інтерфейси «IHostedService» та «IDisposable» для підтримки хостингу та утилізації ресурсів. При створенні екземпляру клас отримує залежності «IConfiguration» та «IServiceScopeFactory» через свій конструктор. «IConfiguration» надає доступ до налаштувань конфігурації, тоді як «IServiceScopeFactory» використовується для створення нової області обслуговування для вирішення залежностей.

Клас визначає декілька методів, необхідних інтерфейсу «IHostedService». Метод «StartAsync» викликається при запуску програми. Він ініціює процес агрегації шляхом одноразового виклику методу «DoWorkAsync», а потім встановлює таймер для запуску агрегації через певний інтервал часу. Період агрегації можна отримати з налаштувань конфігурації.

Метод «DoWorkAsync» є фактичною робочою частиною класу. Він створює нову область обслуговування за допомогою «IServiceScopeFactory» та отримує екземпляр «IContentAggregator» від постачальника служб. Потім він викликає метод «AggregateAsync» в «IContentAggregator» для виконання агрегації новин.

Метод «StopAsync» викликається при завершенні роботи програми. Він зупиняє та утилізує таймер, щоб запобігти подальшій агрегації. У класі також реалізовано метод «Dispose» для належної утилізації таймера, коли він більше не потрібний.

Загалом, клас «BackgroundNewsAggregator» забезпечує автоматичну агрегацію фонових новин за допомогою таймера та сервісу «IContentAggregator». Він запускає процес агрегації під час запуску програми та повторює його через певні проміжки часу, гарантуючи, що свіжий новинний контент безперервно збирається та зберігається в базі даних програми.

2.5 Огляд головних інтерфейсів та класів

2.5.1 INewsSource інтерфейс та його реалізації

Для кожного з новинних ресурсів були розроблені відповідні сервіси які реалізують інтерфейс INewsSource (рисунок 2.11).

```
public interface INewsSource
{
    1 usage  3 implementations  Vova Maksimchuk
    public string Name { get; }
    3 implementations  Vova Maksimchuk
    public string Url { get; }
    1 usage  3 implementations  new *
    public Task<List<(ArticlePreviewDto, ArticleDto)>?> FetchArticlesAsync(
        TopicDto topic, DateTime dateFrom, DateTime dateTo, CancellationToken cancellationToken);
}
```

Рисунок 2.11 – Програмний код інтерфейсу INewsSource

Один із прикладів реалізації цього інтерфейсу є сервіс UnianNewsService (додаток Б). Клас «UnianNewsSource<TNewsParser>» представляє джерело новин, спеціально розроблене для сайту новин УНІАН. Він реалізує інтерфейс «INewsSource» і параметризується аргументом типу «TNewsParser», який повинен реалізувати інтерфейс «IUnianNewsParser». Клас має декілька залежностей, що вводяться через його конструктор. Ці залежності включають опції для налаштування джерела новин УНІАН, реалізацію парсеру новин, репозиторії для керування даними про сайт новин, теми та статті, маппер

для відображення об'єктів, репозиторій для попереднього перегляду статей та логгер.

Клас має властивості для назви та URL-адреси джерела новин, які є похідними від параметрів джерела УНІАН. Він також має внутрішнє посилання на реалізацію парсеру новин. Метод «FetchArticlesAsync» відповідає за вибірку статей з сайту новин УНІАН на основі заданої теми, діапазону дат і токену скасування. Спочатку метод перевіряє, чи існує зв'язок між темою та новинним сайтом для вказаної теми та джерелом новин УНІАН. Якщо такого зв'язку не знайдено, він повертає порожній список. Далі метод викликає парсер новин, щоб отримати прев'ю статей з сайту новин УНІАН для вказаної теми та діапазону дат. Якщо прев'ю не знайдено, повертається нуль. Потім метод перевіряє, чи прев'ю статті вже існують у базі даних, порівнюючи їхні URL-адреси. Якщо знайдено дублікати, вони виключаються з подальшої обробки.

Для кожного прев'ю статті, що залишилося, метод викликає парсер новин, щоб отримати повну інформацію про статтю. Він зіставляє прев'ю статті, саму статтю і пов'язані з нею сутності (наприклад, тему і джерело новин) з відповідними DTO. Він призначає публічний ідентифікатор статті, генерує унікальний публічний ідентифікатор і відповідно встановлює публічний ідентифікатор статті у прев'ю статті. Якщо прев'ю статті не містить URL-адреси зображення, метод намагається витягти його з тіла HTML статті. Якщо зображення не знайдено, призначається URL-адреса зображення за замовчуванням. Отриманий кортеж DTO попереднього перегляду статті та DTO статті додається до списку результатів. Нарешті, метод повертає список кортежів, що містять прев'ю та статті. У разі виникнення винятків у процесі вибірки статей генерується лог помилок, а також повертається null.

Загалом, клас «UnianNewsSource<TNewsParser>» інкапсулює логіку вибірки статей з сайту новин УНІАН на певну тему. Він покладається на реалізацію парсеру новин для вилучення даних статей і надає зручний інтерфейс для інтеграції джерела новин УНІАН в систему агрегатора контенту.

2.5.2 INewsParser інтерфейс та його реалізації

Для кожного з парсерів новин були розроблені відповідні сервіси які реалізують інтерфейс INewsParser (рисунок 2.12). Один із прикладів реалізації парсеру є сервіс парсингу UnianNewsParser (додаток Б).

```
public interface INewsParser
{
    3 usages  3 implementations  new *
    public Task<List<ArticlePreviewDto>?> ParseArticlePreviewsAsync(string url,
        DateTime dateFrom, DateTime dateTo, CancellationToken cancellationToken);
    3 usages  3 implementations  new *
    public Task<ArticleDto?> ParseArticleAsync(string url,
        CancellationToken cancellationToken);
}
```

Рисунок 2.12 – Програмний код інтерфейсу INewsParser

Клас UnianNewsParser відповідає за синтаксичний аналіз прев'ю та повних текстів статей з новинного сайту УНІАН. Конструктор класу UnianNewsParser отримує декілька залежностей, зокрема IBrowsingContext для перегляду веб-сторінок, WebClient для здійснення веб-запитів, IOptions<UnianSourceOptions> для отримання опцій парсеру та ILogger<UnianNewsParser> для ведення журналу.

Метод ParseArticlePreviewsAsync використовується для парсингу прев'ю статей за вказаною URL-адресою в межах заданого діапазону дат. Він використовує IBrowsingContext для відкриття сторінки, а потім запитує сторінку на наявність елементів попереднього перегляду статті за допомогою селекторів CSS. Він витягує таку інформацію, як назва статті, URL-адреса, дата і URL-адреса мініатюрного зображення з кожного елемента попереднього перегляду і створює об'єкти ArticlePreviewDto. Метод продовжує синтаксичний аналіз прев'ю, поки не досягне кінця доступних прев'ю або вказаного діапазону дат. Проаналізовані попередні перегляди статей повертаються у вигляді списку List<ArticlePreviewDto>.

Метод ParseArticleAsync використовується для розбору повної статті за вказаною URL-адресою. Спочатку перевіряється, чи вказує URL на статтю або публікацію. Якщо це стаття, він витягує назву статті, автора, дату і вміст зі сторінки за допомогою селекторів CSS. Він повторює цикл над елементами вмісту, такими як абзаци, заголовки і зображення, і створює тіло статті у форматі HTML. Якщо це публікація, він витягує назву публікації, автора, дату і вміст аналогічним чином. Проаналізована стаття або публікація повертається як об'єкт ArticleDto.

Метод ParsePublicationAsync є допоміжним методом, який викликається ParseArticleAsync, коли URL вказує на публікацію, а не на статтю. Він витягує назву публікації, автора, дату і головне зображення зі сторінки. Потім він ітераційно переглядає елементи вмісту публікації і створює тіло HTML для публікації.

Протягом усього процесу синтаксичного аналізу для побудови HTML-тексту статей або публікацій використовується конструктор рядків _stringBuilder. Логгер _logger використовується для реєстрації інформації, пов'язаної з синтаксичним аналізом, та помилок. Загалом, клас UnianNewsParser надає методи для синтаксичного аналізу прев'ю та повних текстів статей/публікацій з новинного сайту УНІАН, використовуючи надані залежності та методи синтаксичного аналізу HTML.

2.6 HomeController

Для того, щоб користувач мав можливість отримувати інформацію з сервісів бізнес логіки та читати новини, був створений HomeController. Код реалізації цього контролеру поданий у додатку Б. У HomeController все починається з конструктору. Конструктор приймає такі залежності, як IContentAggregator, ILogger<HomeController> та INewsService. Ці залежності підключаються за допомогою ін'єкції залежностей.

Метод дії Index є дією за замовчуванням для головної сторінки ("/"). Він викликає метод GetLastArticlePreviewAsync з _newsService, щоб отримати попередній перегляд останньої статті. Він також викликає метод GetArticlePreviewsAsync з _newsService, щоб отримати прев'ю статей для певних тем (наприклад, «ukraine», «war», «technology»). Отримані прев'ю статей зберігаються у властивостях ViewBag і передаються у подання.

Метод дії ArticlePreviews використовується для обробки запитів на прев'ю статей певної теми ("/{topic}"). Він отримує прев'ю статті для вказаної теми за допомогою методу GetArticlePreviewsAsync сервісу _newsService. Він також отримує додаткову інформацію, таку як деталі теми, загальна кількість записів і кількість сторінок. Отримані дані зберігаються у властивостях ViewBag і передаються до подання.

Метод дії GetArticle використовується для обробки запитів на окремі статті ("/{topic}/{publicArticleId}"). Він отримує дані про тему і дані про статтю за допомогою методів GetTopicAsync і GetArticleAsync методу _newsService. Отримані дані зберігаються у властивостях ViewBag і передаються у подання «Article».

Загалом, HomeController виступає мостом між запитами користувача, INewsService та представленнями, отримуючи необхідні дані від сервісу та передаючи їх представленням для рендерингу.

3 ОГЛЯД ЗАСТОСУНКУ

3.1 Головна сторінка

Інтерфейс сайту агрегації новин (рисунок 3.1) розроблений таким чином, щоб забезпечити користувачам легкий доступ до різних новинних тем та останніх оновлень. Нижче наведено опис головної сторінки на основі наданого скріншоту.

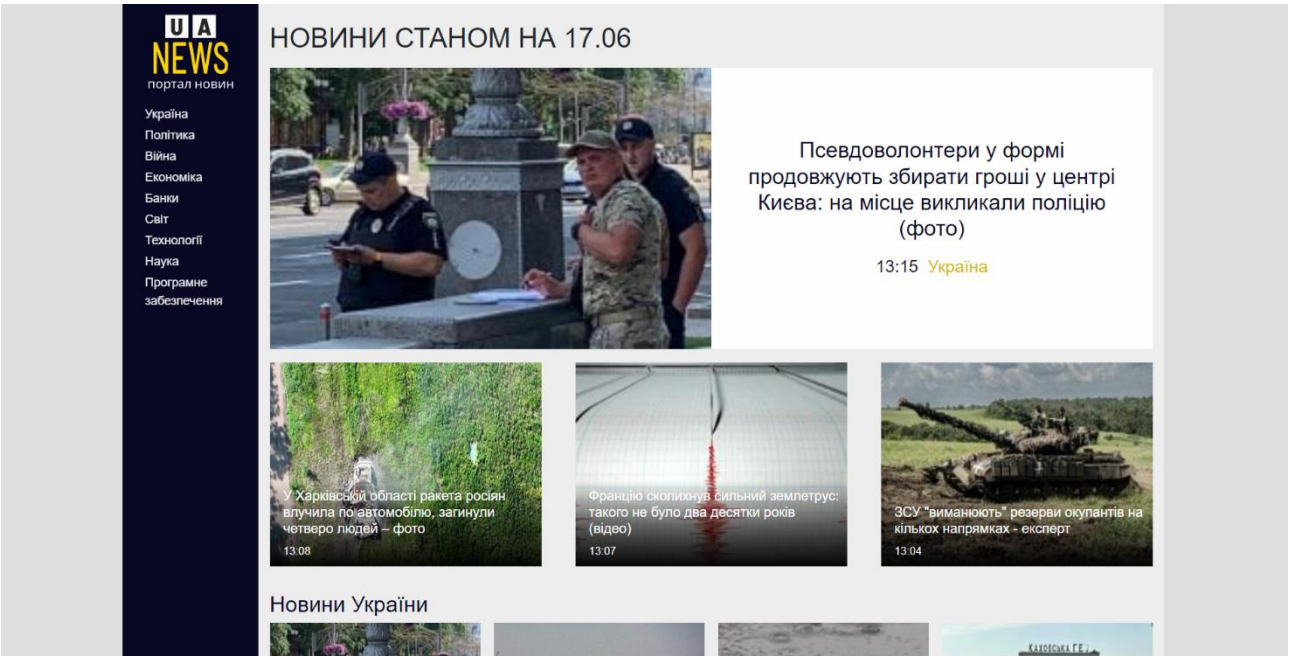

Рисунок 3.1 – Головна сторінка сайту агрегації новин

У верхньому лівому куті інтерфейсу розміщено логотип сайту, який слугує посиланням на головну сторінку. Це дозволяє користувачам швидко повернутися на головну сторінку з будь-якого іншого розділу сайту. Поруч з логотипом розташоване меню, яке надає вибір тем новин. Це меню дозволяє користувачам вибрати бажану категорію новин. Доступні теми включають Україну, політику, війну, економіку, банки, світ, технології, науку та програмне забезпечення. Вибравши певну тему, користувачі можуть отримати доступ до новин, що стосуються саме цієї категорії.

Основна частина сторінки присвячена відображенню новинного контенту. Інтерфейс вказує на те, що новини, які відображаються, є актуальними на сьогоднішній день. Ця інформація допомагає користувачам залишатися в курсі останніх подій і гарантує, що новини, які відображаються, є актуальними.

В основній області контенту користувачі можуть знайти попередній перегляд останніх новин. Цей попередній перегляд, найімовірніше, включає заголовок або назву новинної статті, що супроводжується коротким резюме або уривком. Це дозволяє користувачам отримати уявлення про найсвіжіші новини з першого погляду. Крім того, на головній сторінці також відображаються інші нещодавні новини. Ці статті розташовані в хронологічному порядку, причому найсвіжіші з них відображаються зліва.

Головна сторінка також розділена на три розділи (рисунок 3.2), в кожному з яких представлені останні новини за певними темами: Україна, війна та технології.

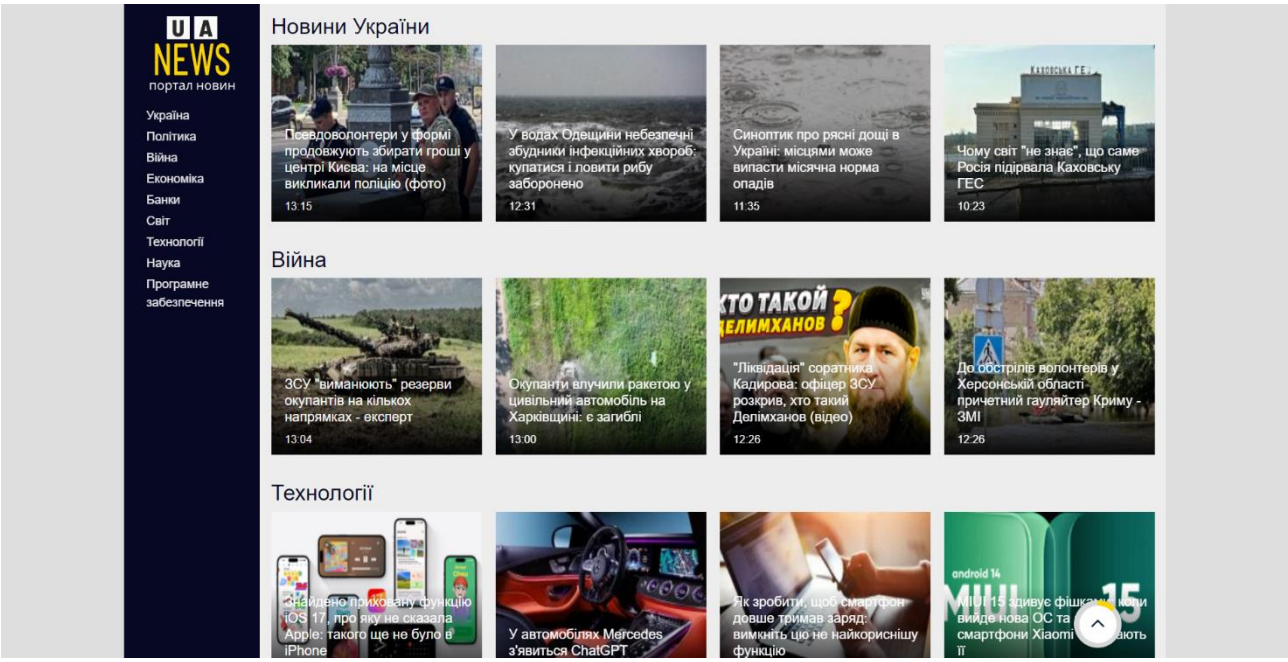

Рисунок 3.2 – Категорії останніх новин

У розділі «Україна» відображаються чотири останні новини. Ці новини, містять заголовки та фотографії. Користувачі можуть швидко переглянути ці новини, щоб отримати огляд останніх подій в Україні.

Аналогічно, розділ «Війна» також містить чотири останні новини, пов'язані з конфліктами або військовими питаннями. Новини в цьому розділі висвітлюють ключові події, розробки або аналіз, пов'язані з війною або збройними конфліктами.

Розділ «Технології» присвячений останнім новинам у сфері технологій. Він також складається з чотирьох новин, які висвітлюють такі теми, як інновації, досягнення, гаджети або тенденції в технологічній індустрії. Користувачі, зацікавлені в тому, щоб бути в курсі останніх технологічних розробок, можуть знайти відповідні новини в цьому розділі.

Розташування новин у кожному розділі, базується на їхній свіжості, причому найсвіжіші новини з'являються зліва. Це дозволяє користувачам швидко отримувати доступ до останніх оновлень і залишатися в курсі бажаних тем.

Інтерфейс забезпечує чітке візуальне розділення між трьома розділами, що полегшує користувачам ідентифікацію та навігацію між різними тематичними областями. Він пропонує моментальний знімок найсвіжіших новин у кожній категорії, дозволяючи користувачам натискати на окремі новини, щоб прочитати повну версію статті або отримати доступ до більш детальної інформації.

Загалом, головна сторінка сайту агрегації новин має зручний інтерфейс з меню для вибору тем новин, попереднім переглядом останніх новин і добіркою інших нещодавніх новинних статей. Такий макет має на меті полегшити навігацію та забезпечити користувачам доступ до найактуальнішого та найсвіжішого новинного контенту.

3.2 Новини за темами

На скріншоті (рисунок 3.3) представлено сторінку сайту агрегації новин, яка фокусується на новинах, пов'язаних з обраною темою – Україна. Нижче наведено опис елементів інтерфейсу, які видно на скріншоті.

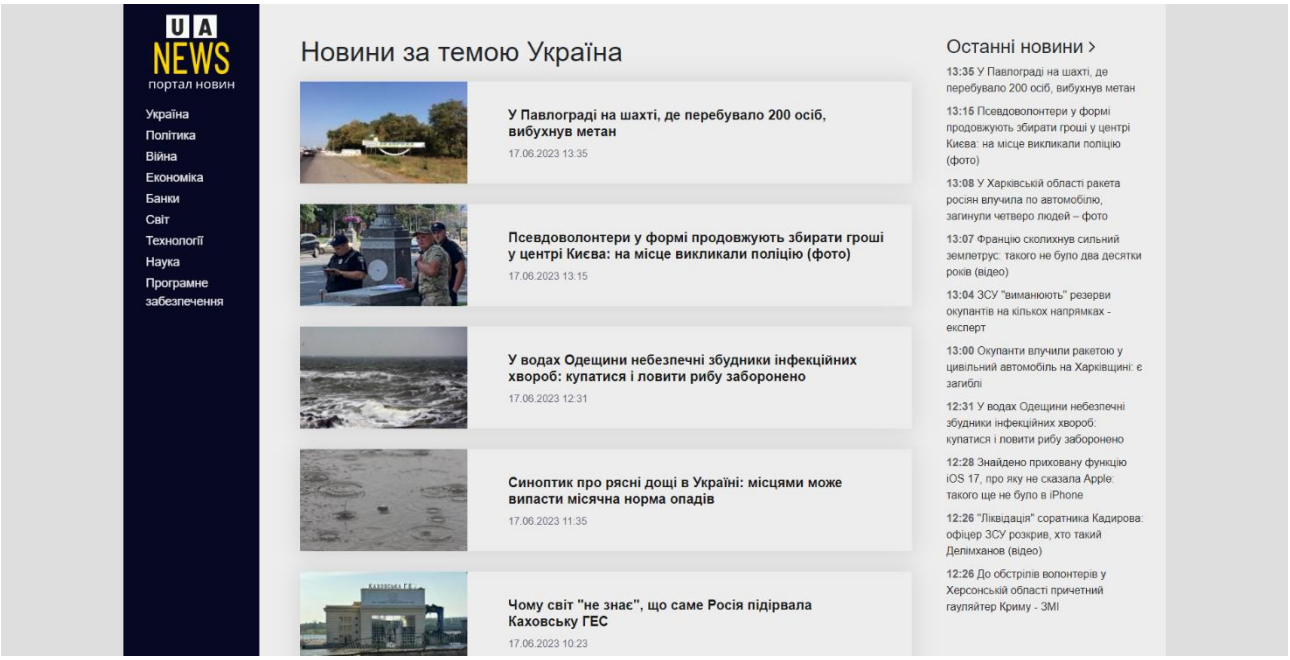

Рисунок 3.3 – Новини за темою Україна

Основний зміст сторінки складається з прев'ю останніх новин, безпосередньо пов'язаних з Україною. Ці прев'ю новин відображаються у структурованому вигляді, що дозволяє користувачам швидко переглядати заголовки або назви та отримувати уявлення про зміст новинних статей. Реалізовано функцію посторінкової навігації, що дозволяє користувачам переходити між кількома сторінками новин. На кожній сторінці відображається по 20 новин, що забезпечує користувачам доступ до значної кількості останніх статей.

Користувачі можуть натиснути на будь-яку з прев'ю новин, щоб отримати доступ до повної версії статті. Ця функція дозволяє читачам глибше зануритися в конкретну новину і отримати вичерпну інформацію.

У правій частині сторінки розташоване меню «Останні новини». Це меню слугує зручним інструментом для користувачів, щоб бути в курсі останніх новин з усіх тем, а не лише з України. Меню містить заголовки або назви останніх новинних статей з різних категорій. Записи в меню «Останні новини» є клікабельними, що надає користувачам можливість отримати доступ до повних версій новинних статей. Натиснувши на будь-який із заголовків, користувачі будуть перенаправлені на повну версію статті, незалежно від того, до якої теми вона належить.

Загалом, інтерфейс цієї сторінки на сайті агрегатора новин ефективно фокусується на новинах, пов'язаних з обраною темою. Він пропонує користувачам можливість ознайомитися з останніми подіями в обраній темі за допомогою попереднього перегляду новин, реалізує пагінацію для зручної навігації, а також надає окреме меню під назвою «Останні новини», щоб тримати користувачів в курсі останніх статей за всіма темами.

3.3 Сторінка статті

Скріншот (рисунок 3.4) відображає сторінку на сайті агрегації новин, яка представляє повну версію новинної статті.

Рисунок 3.4 – Повна версія новинної статті

У верхній частині сторінки, на видному місці, відображається заголовок новини. Це дозволяє користувачам швидко визначити тему або предмет новини, яку вони збираються прочитати. Поруч із заголовком згадується автор новини. Поруч з іменем автора відображається дата публікації новини. Ця інформація вказує, коли стаття була опублікована, що дозволяє користувачам оцінити свіжість і актуальність новини.

Якщо новина містить зображення для попереднього перегляду, воно відображається в інтерфейсі. Цей візуальний елемент дає користувачам змогу зазирнути у зміст новини та покращує загальний досвід читання.

Крім того, інтерфейс, містить основний текст новини (рисунок 3.5), надаючи читачам повний текст і додаткові подробиці на цю тему.

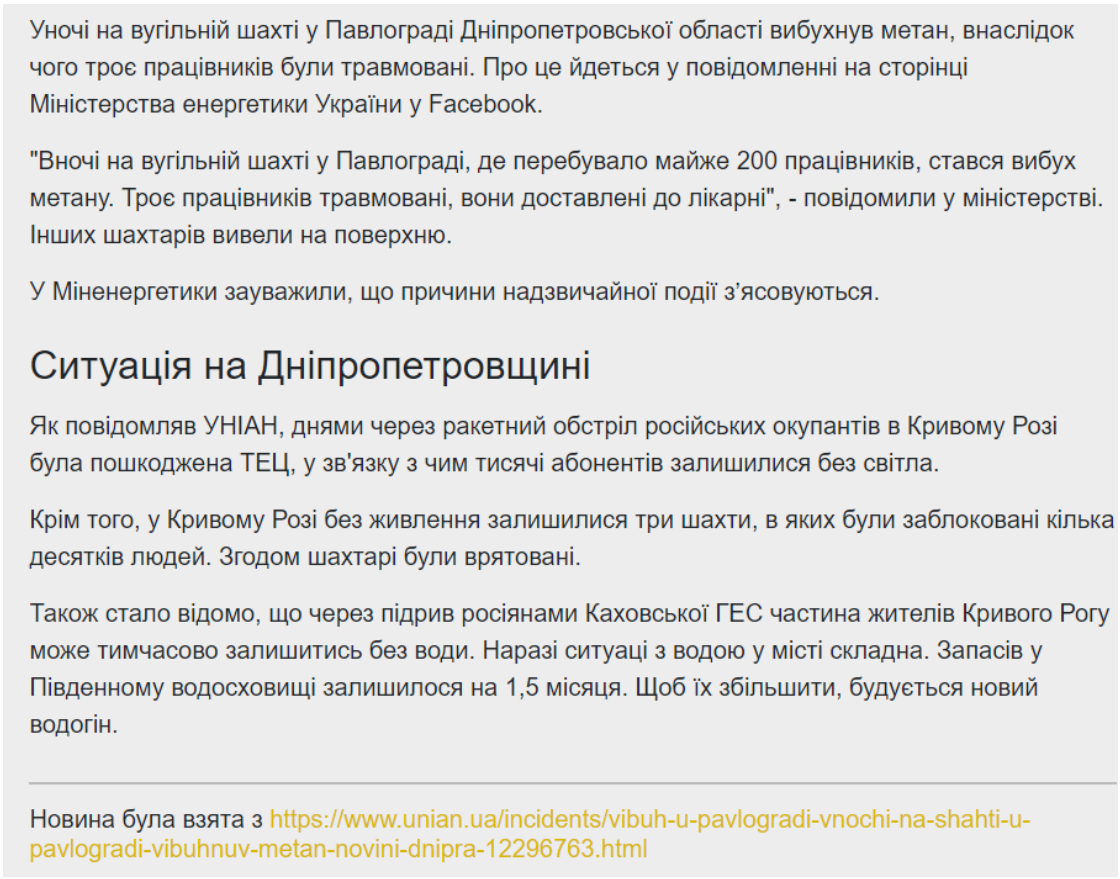

Рисунок 3.5 – Основний текст новини

Центральним елементом сторінки є основний текст новини. Цей розділ містить вичерпні деталі, аналіз і контекст, що оточує тему новини. Це дозволяє користувачам зануритися в зміст і отримати повне розуміння новини.

Наприкінці статті міститься посилання на першоджерело новини. Це посилання спрямовує користувачів на зовнішній веб-сайт або публікацію, де новина була спочатку опублікована. Це дозволяє читачам отримати доступ до повного тексту статті на платформі-джерелі, якщо вони хочуть дізнатися більше або перевірити інформацію. Включення посилання на джерело новин посилює прозорість і довіру до них, оскільки вказує на першоджерело і дозволяє користувачам отримати доступ до повного тексту статті в її оригінальному контексті.

Загалом, дизайн інтерфейсу на цій сторінці сайту агрегації новин гарантує, що користувачі можуть легко отримати доступ до повного тексту новини. Представляючи основний текст новини та надаючи посилання на джерело новини, сайт пропонує комплексний досвід читання, а також вказує на першоджерело і сприяє подальшому вивченню теми, якщо це необхідно.

4 ТЕСТУВАННЯ

4.1 Визначення тестування

Тестування – це систематичний і структурований процес оцінки програмної системи для визначення її якості, функціональності та продуктивності. Він передбачає виконання програмних компонентів або всієї системи для виявлення дефектів, помилок або відхилень від бажаної поведінки. Мета тестування полягає в тому, щоб переконатися, що програмне забезпечення відповідає заданим вимогам, працює за призначенням і забезпечує надійні та задовільні результати для кінцевих користувачів. Основна мета тестування полягає в тому щоб виявити будь-які розбіжності між фактичною поведінкою програмного забезпечення та очікуваною поведінкою, тим самим підвищуючи його надійність і зручність використання. Тестування допомагає виявити та виправити дефекти, помилки та проблеми, які можуть вплинути на продуктивність, функціональність, безпеку та зручність роботи з програмним забезпеченням.

Тестування здійснюється за допомогою тестових кейсів, які призначені для перевірки різних аспектів програмного забезпечення, включаючи його входи, виходи, функції та взаємодію. Ці тестові кейси виконуються систематично, щоб перевірити поведінку програмного забезпечення та виявити будь-які відхилення або збої. Процес тестування включає в себе різні етапи, починаючи з планування та розробки тестів, за якими слідує виконання тестів, аналіз результатів тестування та звітування про дефекти. Він може включати ручне тестування, коли тестові кейси виконуються тестувальниками, а також автоматизоване тестування, яке передбачає використання спеціалізованих інструментів і скриптів для автоматизації процесу тестування.

У процесі тестування використовуються різні типи методів тестування, спрямовані на конкретні аспекти програмного забезпечення. До них належать функціональне тестування, яке перевіряє правильність функціонування окремих

компонентів або функцій; інтеграційне тестування, яке перевіряє взаємодію та сумісність між різними модулями або підсистемами; тестування продуктивності, яке оцінює поведінку програмного забезпечення за різних робочих навантажень та умов; тестування безпеки, яке фокусується на виявленні вразливостей та забезпеченні захисту даних; юзабіліті-тестування, яке оцінює зручність та простоту використання програмного забезпечення; та багато інших.

Тестування також передбачає порівняння фактичних результатів з очікуваними, визначеними вимогами та специфікаціями до програмного забезпечення. Відхилення або розбіжності повідомляються як дефекти або помилки, які потім визначаються пріоритетами, відстежуються і вирішуються командою розробників. Процес триває ітеративно, поки програмне забезпечення не буде відповідати бажаним стандартам якості і не буде готове до розгортання.

Крім того, тестування – це не одноразова діяльність, а безперервний процес протягом усього життєвого циклу розробки програмного забезпечення. Він починається на ранніх стадіях розробки і продовжується під час обслуговування та оновлень, щоб забезпечити надійність, стабільність і продуктивність програмного забезпечення.

4.2 Види тестування

У процесі розробки застосовуються різні види тестування для перевірки різних аспектів програмного забезпечення. Кожен тип тестування має певну мету і фокусується на конкретних аспектах функціональності, продуктивності, безпеки або зручності використання програмного забезпечення.

Модульне тестування фокусується на перевірці окремих компонентів або блоків програмного забезпечення, таких як функції, методи або класи. Воно гарантує, що кожен модуль поводиться так, як очікується, і відповідає його специфічним вимогам. Зазвичай модульні тести пишуться розробниками і виконуються ізольовано за допомогою фреймворків, таких як JUnit або NUnit.

Інтеграційне тестування перевіряє взаємодію та інтеграцію між різними модулями, підсистемами або зовнішніми системами. Воно гарантує, що ці компоненти працюють разом правильно, точно обмінюються даними і підтримують сумісність. Інтеграційне тестування може бути виконане з використанням різних підходів, таких як тестування зверху-вниз, знизу-вгору або сендвіч-тестування.

Функціональне тестування перевіряє функціональну поведінку програмного забезпечення шляхом тестування його можливостей, функцій та взаємодії з користувачем. Воно гарантує, що програмне забезпечення працює відповідно до заданих вимог і дає очікувані результати. Функціональне тестування може включати як ручне тестування, так і автоматизовані методи тестування.

Тестування продуктивності оцінює чуйність, масштабованість і стабільність програмного забезпечення при різних робочих навантаженнях і умовах. Воно вимірює такі фактори, як час відгуку, пропускну здатність, використання ресурсів та вузькі місця системи. Методи тестування продуктивності включають навантажувальне тестування, стрес-тестування та тестування на витривалість.

Тестування безпеки фокусується на виявленні вразливостей, слабких місць і потенційних ризиків безпеки в програмному забезпеченні. Воно забезпечує конфіденційність, цілісність і доступність даних шляхом тестування на наявність потенційних загроз, несанкціонованого доступу, витоку даних та інших недоліків безпеки. Тестування безпеки може включати тестування на проникнення, сканування вразливостей і перевірку коду безпеки.

Юзабіліті-тестування оцінює зручність програмного забезпечення, простоту використання та загальний користувацький досвід. Воно передбачає тестування програмного забезпечення реальними користувачами для збору відгуків про його інтуїтивність, навігацію та дизайн інтерфейсу. Юзабіліті-тестування допомагає виявити проблеми юзабіліті та надає ідеї для покращення зручності використання програмного забезпечення.

Регресійне тестування виконується для того, щоб переконатися, що зміни або оновлення програмного забезпечення не призведуть до появи нових дефектів або порушення існуючої функціональності. Воно включає в себе повторне тестування раніше протестованих функцій для перевірки їхньої постійної коректності. Регресійне тестування допомагає підтримувати цілісність програмного забезпечення після модифікацій.

Приймальне тестування проводиться для того, щоб визначити, чи відповідає програмне забезпечення критеріям прийнятності та задовольняє вимоги зацікавлених сторін. Воно перевіряє програмне забезпечення на відповідність реальним сценаріям і очікуванням користувачів. Приймальне тестування можуть проводити кінцеві користувачі або спеціальна команда тестувальників, щоб отримати впевненість у готовності програмного забезпечення до розгортання.

4.3 Важливість тестування у розробці

Тестування відіграє життєво важливу роль у процесі розробки програмного забезпечення і має вирішальне значення для забезпечення якості, надійності та успіху програмного додатку. Це інвестиція, яка приносить значні вигоди протягом усього життєвого циклу розробки і навіть після нього. Давайте розглянемо важливість тестування більш детально:

– виявлення помилок і запобігання проблемам: тестування допомагає виявити помилки, дефекти та проблеми на ранній стадії циклу розробки. Завдяки виявленню та вирішенню цих проблем на ранній стадії, вартість та зусилля, необхідні для їх виправлення, значно зменшуються. Ефективне тестування мінімізує ймовірність того, що критичні проблеми дійдуть до кінцевих користувачів і негативно вплинуть на їхній досвід;

– покращення якості програмного забезпечення: тестування є засобом забезпечення того, що програмне забезпечення відповідає визначеним вимогам і

функціонує як очікується. Воно допомагає виявити розбіжності між бажаною та фактичною поведінкою системи, тим самим дозволяючи розробникам вдосконалювати та покращувати програмне забезпечення. Завдяки ретельному тестуванню можна покращити якість програмного забезпечення, що призведе до створення надійного та стійкого додатку;

– покращення користувацького досвіду: тестування має важливе значення для забезпечення позитивного користувацького досвіду. Перевіряючи функціональність, зручність використання та продуктивність програмного забезпечення, тестування гарантує, що користувачі можуть взаємодіяти з додатком безперешкодно та інтуїтивно. Тестування також допомагає виявити і вирішити проблеми, пов'язані з дизайном інтерфейсу користувача, навігацією і швидкістю реагування, що в кінцевому підсумку призводить до створення зручного для користувача додатку;

– підвищення надійності та стабільності: тестування допомагає встановити надійність і стабільність програмного забезпечення. Воно гарантує, що додаток може обробляти різні сценарії, вводити дані та взаємодіяти з користувачем без збоїв і неочікуваних помилок. Надійні методи тестування вселяють впевненість у стабільності програмного забезпечення, що дозволяє користувачам покладатися на нього у своїх потребах;

– відповідність та безпека: тестування відіграє важливу роль у забезпеченні відповідності галузевим стандартам, нормам і вимогам безпеки. Воно допомагає виявити вразливості, слабкі місця та потенційні ризики для безпеки, дозволяючи розробникам впроваджувати відповідні заходи безпеки та захищати конфіденційні дані. Тестування також допомагає перевірити, чи відповідає програмне забезпечення певним нормам або найкращим галузевим практикам;

– економія коштів та часу: хоча тестування вимагає вкладення часу і ресурсів, воно в кінцевому підсумку призводить до економії коштів і часу. Раннє виявлення та вирішення проблем за допомогою тестування допомагає уникнути дорогих доопрацювань, скарг клієнтів та потенційних юридичних наслідків.

Крім того, ефективне тестування скорочує час, необхідний для виправлення помилок, і забезпечує більш плавний процес розгортання і випуску продукту;

– задоволеність клієнтів та бізнес-успіх: тестування безпосередньо впливає на задоволеність клієнтів, оскільки воно гарантує, що програмне забезпечення відповідає їхнім очікуванням і приносить користь. Надаючи високоякісний і надійний продукт, компанії можуть покращити свою репутацію, завоювати лояльність клієнтів і досягти довгострокового успіху. Тестування також допомагає визначити сфери для вдосконалення та інновацій, що дозволяє компаніям залишатися попереду на конкурентному ринку.

4.4 Сценарії ручного тестування клієнтської частини

Далі будуть розглянутий набір сценаріїв ручного тестування сторінок та меню сайту-агрегатора новин. Ручне тестування – це важлива частина процесу тестування програмного забезпечення, яка передбачає виконання тестових кейсів вручну, імітацію взаємодії з користувачем і перевірку очікуваної поведінки системи. Виконуючи ці сценарії тестування, ми прагнемо переконатися, що клієнтська частина застосунку працює правильно.

Кожен сценарій тестування описує конкретні кроки, які необхідно виконати, і очікувану поведінку для кожної кнопки на різних сторінках. Ці сценарії допоможуть перевірити функціональність і швидкість реагування кнопок, забезпечуючи безперебійну та інтуїтивно зрозумілу роботу користувачів. Завдяки ретельному тестуванню ми можемо виявити і вирішити будь-які проблеми або невідповідності, які можуть виникнути, забезпечуючи надійність і зручність використання сайту-агрегатора новин.

Сценарій: перенаправлення на головну сторінку при натисканні на логотип.

Етапи тестування:

– відкрийте веб-сайт і переконайтеся, що відображається початковий екран;

– натисніть на логотип.

Очікувана поведінка: веб-сайт повинен перенаправляти на головну сторінку.

Сценарій: перенаправлення на новини про Україну при натисканні на пункт меню «Україна».

Кроки тестування:

– відкрийте веб-сайт і переконайтеся, що відображається початковий екран;

– натисніть на пункт меню «Україна».

Очікувана поведінка: веб-сайт повинен перенаправляти на новини про Україну.

Сценарій: перенаправлення на новини про політику при натисканні на пункт меню «Політика».

Етапи тестування:

– відкрийте веб-сайт і переконайтеся, що відображається початковий екран;

– натисніть на пункт меню «Політика».

Очікувана поведінка: веб-сайт повинен перенаправляти на новини про політику.

Сценарій: перенаправлення на новини на тему війни при натисканні на пункт меню «Війна».

Етапи тестування:

– відкрийте веб-сайт і переконайтеся, що відображається початковий екран;

– натисніть на пункт меню «Війна».

Очікувана поведінка: веб-сайт повинен перенаправляти на новини на тему війни.

Сценарій: перенаправлення на новини на тему економіки при натисканні на пункт меню «Економіка».

Етапи тестування:

– відкрийте веб-сайт і переконайтеся, що відображається початковий екран;

– натисніть на пункт меню «Економіка»;

Очікувана поведінка: веб-сайт повинен перенаправляти на новини на економічну тематику.

Сценарій: перенаправлення на новини про банки при натисканні на пункт меню «Банки».

Етапи тестування:

– відкрийте веб-сайт і переконайтеся, що відображається початковий екран;

– натисніть на пункт меню «Банки».

Очікувана поведінка: сайт повинен перенаправляти на новини про банки.

Сценарій: перенаправлення на новини про світ при натисканні на пункт меню «Світ».

Етапи тестування:

– відкрийте веб-сайт і переконайтеся, що відображається початковий екран;

– натисніть на пункт меню «Світ».

Очікувана поведінка: веб-сайт повинен перенаправляти на новини про мир.

Сценарій: перенаправлення на новини на тему технологій при натисканні на пункт меню «Технології».

Етапи тестування:

– відкрийте веб-сайт і переконайтеся, що відображається початковий екран;

– натисніть на пункт меню «Технології».

Очікувана поведінка: веб-сайт повинен перенаправляти на новини на тему технологій.

Сценарій: перенаправлення на новини на тему науки при натисканні на пункт меню «Наука».

Етапи тестування:

– відкрийте веб-сайт і переконайтеся, що відображається початковий екран;

– натисніть на пункт меню «Наука».

Очікувана поведінка: веб-сайт повинен перенаправляти на новини на тему науки.

Сценарій: перенаправлення на новини на тему програмного забезпечення при натисканні на пункт меню «Програмне забезпечення».

Етапи тестування:

– відкрийте веб-сайт і переконайтеся, що відображається початковий екран.

– натисніть на пункт меню «Програмне забезпечення».

Очікувана поведінка: веб-сайт повинен перенаправляти на новини на тему програмного забезпечення.

Сценарій: перегляд прев'ю останніх новин на головній сторінці.

Етапи тестування:

– відкрийте веб-сайт і переконайтеся, що головна сторінка відображається;

– переконайтеся, що прев'ю останніх новин відображаються.

Очікувана поведінка: на головній сторінці повинні відображатися прев'ю останніх новин.

Сценарій: Перегляд прев'ю новин на тему «Україна» на головній сторінці.

Етапи тестування:

– відкрийте веб-сайт і переконайтеся, що відображається головна сторінка;

– натисніть на новину з розділу «Україна».

Очікувана поведінка: веб-сайт повинен перейти до повної версії новини, пов'язаної з Україною.

Сценарій: перегляд прев'ю новин на тему «Війна» на головній сторінці.

Етапи тестування:

– відкрийте веб-сайт і переконайтеся, що відображається головна сторінка;

– натисніть на новини з розділу «Війна».

Очікувана поведінка: веб-сайт повинен перейти до повної версії новини, пов'язаної з війною.

Сценарій: перегляд прев'ю новин на тему «Технології» на головній сторінці.

Етапи тестування:

– відкрийте веб-сайт і переконайтеся, що відображається головна сторінка;

– Натисніть на розділу «Технології».

Очікувана поведінка: веб-сайт повинен переходити до повної версії новини, пов'язаної з технологіями.

Сценарій: перегляд прев'ю новин на певну тему.

Тестові кроки:

– відкрийте сторінку новин на певну тему;

– переконайтеся, що відображається 20 прев'ю новин.

Очікувана поведінка: на сторінці новин повинно бути показано 20 прев'ю новинних статей.

Сценарій: переглянути повну версію новини, натиснувши на прев'ю.

Етапи тестування:

– відкрийте сторінку новин на певну тему;

– натисніть на прев'ю новини.

Очікувана поведінка: веб-сайт повинен перейти на повну версію новини, на яку було натиснуто.

Сценарій: перехід на попередню сторінку за допомогою кнопки «Назад»

Етапи тестування:

– відкрийте сторінку новин на певну тему;

– натисніть на кнопку «Назад» в меню пагінації.

Очікувана поведінка: веб-сайт повинен переходити на попередню сторінку новин.

Сценарій: перехід на наступну сторінку за допомогою кнопки «Вперед»

Етапи тестування:

– відкрийте сторінку новин на певну тему;

– натисніть на кнопку «Вперед» в меню пагінації.

Очікувана поведінка: веб-сайт повинен переходити на наступну сторінку з новинами.

Сценарій: перехід на певну сторінку при натисканні на номер сторінки.

Етапи тестування:

– відкрийте сторінку новин на певну тему;

– натисніть на певний номер сторінки в меню нумерації сторінок.

Очікувана поведінка: веб-сайт повинен перейти на сторінку, що відповідає натиснутому номеру сторінки.

Сценарій: перегляд прев'ю останніх новин в меню «Останні новини»

Етапи тестування:

– відкрийте сторінку новин на певну тему;

– переконайтеся, що меню «Останні новини» відображається праворуч.

– натисніть на прев'ю новини в меню «Останні новини».

Очікувана поведінка: веб-сайт повинен перейти на повну версію натиснутої новини.

4.5 Шляхи розвитку тестування застосунку

По-перше, важливо розглянути поточне покриття тестів і виявити будь-які прогалини або області, які потребують поліпшення. Аналіз існуючих тестів може допомогти визначити, які частини програми добре протестовані, а які потребують додаткової уваги.

Один із способів покращити тестування – збільшити кількість модульних тестів. Юніт-тести фокусуються на окремих компонентах або блоках коду, таких як методи або функції, і перевіряють їх поведінку ізольовано. Написавши більше модульних тестів, буде впевненість, що кожен компонент програми функціонує правильно і видає очікувані результати для різних вхідних даних і сценаріїв.

Ще один аспект, який варто розглянути – це інтеграційне тестування. Інтеграційні тести зосереджені на перевірці взаємодії між різними компонентами або модулями всередині програми. У випадку програми-агрегатора новин інтеграційні тести можна написати для перевірки інтеграції між рівнем бізнес-логіки (BLL) і рівнем доступу до даних (DAL). Ці тести можуть допомогти виявити будь-які проблеми або невідповідності в отриманні даних, маніпуляціях з ними або їх збереженні.

На завершення, щоб покращити тестування додатку агрегатора новин, важливо зосередитися на збільшенні покриття модульних тестів, покращенні тестування інтеграції між різними рівнями, впровадженні автоматизованих методів тестування та впровадженні тест-орієнтованої розробки.

ВИСНОВКИ

У цій кваліфікаційній роботі було успішно розроблене програмне забезпечення для агрегації інформації з новинних сайтів. Програмне забезпечення призначене для зручного та ефективного збору, обробки та представлення новинних даних з різних джерел. Воно надає користувачам швидкий та зручний доступ до актуальних новин, збираючи та структуруючи дані з різних новинних сайтів.

Актуальність цього проекту полягає у зростаючому попиті на централізований доступ до новинного контенту. Зі стрімким розвитком цифрової епохи користувачі шукають ефективні способи бути в курсі останніх новин без необхідності відвідувати безліч сайтів. Наше програмне забезпечення задовольняє цю потребу, пропонуючи єдину платформу, де користувачі можуть отримати доступ до новинних статей з різних джерел в одному зручному місці.

У початковому аналізі предметної галузі ми надали загальний огляд агрегації новин. Ми дали визначення агрегації новин і підкреслили її важливість та переваги, такі як економія часу, персоналізований контент і різноманітні перспективи. Ми також обговорили виклики та обмеження, з якими стикаються в цій галузі, такі як достовірність даних та алгоритмічні упередження. Крім того, були досліджені існуючі методи агрегації новин, включно з традиційними підходами та автоматизованими методами. Порівнюючи різні методи, ми отримали уявлення про сильні та слабкі сторони кожного підходу, що допомогло сформувати наш процес розробки.

Розробка додатку для агрегації новин відбувалася за трирівневою архітектурою, що забезпечує модульність і масштабованість системи. Були обрані та використані різні технології, включаючи .NET Core MVC для фреймворку веб-додатків, AngleSharp для парсингу HTML та PostgreSQL як систему управління базами даних. Ці технології забезпечили міцну основу для ефективного збору та обробки новинних даних.

База даних відігравала вирішальну роль у додатку, були описані моделі, що використовуються для зберігання новинних даних. Крім того, був наданий огляд реалізації патернів GenericRepository та UnitOfWork, які полегшили взаємодію між додатком та базою даних. Основний сервіс додатку, ContentAggregator, продемонстрував основний функціонал збору новинних даних з різних джерел. За допомогою BackgroundNewsAggregator ми досягли автоматизованого та регулярного оновлення новинного контенту, забезпечуючи доступність найсвіжішої інформації.

Також був представлений огляд основних інтерфейсів та класів нашого додатку, таких як інтерфейс INewsSource та його реалізацій, що дозволило легко інтегрувати нові джерела новин. Інтерфейс INewsParser та його реалізації дозволили аналізувати та витягувати релевантні дані з джерел новин.

З боку користувацького інтерфейсу ми продемонстрували головні сторінки нашого додатку. Головна сторінка надавала вичерпний огляд останніх новин, тоді як розділ новин за темами дозволяв користувачам вивчати статті, згруповані за певними темами, що їх цікавлять. Нарешті, сторінка статті надавала детальний огляд обраної новини, включаючи назву, автора, дату і додаткове зображення для попереднього перегляду.

Таким чином, було успішно розроблене програмне забезпечення, яке виконує своє призначення - агрегування інформації з новинних сайтів. Додаток пропонує користувачам зручний та ефективний спосіб доступу до актуальних новин з різних джерел на одній централізованій платформі. Використовуючи сучасні технології та слідуючи найкращим практикам, було створене надійне та масштабоване рішення для агрегації новин.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Annieahujaweb2020. Three-Tier client server architecture in distributed system. GeeksForGeeks. URL: https://www.geeksforgeeks.org/three-tier-client-server-architecture-in-distributed-system/ (дата звернення: 10.06.2023).

2. Catone J. 5 news aggregation methods compared. Readwrite. URL: https://readwrite.com/news_aggregation_methods/ (дата звернення: 10.06.2023).

3. Donovan M. News aggregators- problems and advantages to readers. Medium. URL: https://medium.com/@donovanmarco/news-aggregators-problems-and-advantages-to-readers-bb49add8dc2 (дата звернення: 10.06.2023).

4. Dot Net Tutorials. Introduction to ASP.NET core MVC Framework. DotNetTutorials. URL: https://dotnettutorials.net/lesson/introduction-asp-net-core-mvc/ (дата звернення: 10.06.2023).

5. InsightSoftware. 5 benefits of a 3-tier architecture. InsightSoftware. URL: https://insightsoftware.com/blog/5-benefits-of-a-3-tier-architecture/ (дата звернення: 10.06.2023).

6. Lutkevich B. Content aggregator definition. TechTarget. URL: https://www.techtarget.com/searchcontentmanagement/definition/content-aggregator (дата звернення: 10.06.2023).

7. Martijin. How to write a Web Scraper in C# .NET with AngleSharp. ScrapeShark. URL: https://scrapeshark.com/blog/web-scraper-csharp-dotnet-anglesharp (дата звернення: 10.06.2023).

8. Octorparse. How to build a news aggregator with text classification. Medium. URL: https://medium.com/technology-hits/how-to-build-a-news-aggregator-with-text-classification-cd84dab608aa (дата звернення: 10.06.2023).

9. PosgteSQLTutorial. What is PostgreSQL?. PosgteSQLTutorial. URL: https://www.postgresqltutorial.com/postgresql-getting-started/what-is-postgresql/ (дата звернення: 10.06.2023).

10. Rouse M. Three-Tier architecture. Technopedia. URL: https://www.techopedia.com/definition/24649/three-tier-architecture (дата звернення: 10.06.2023).

11. Skaggs K. What's new in news aggregation?. The Guargian. URL: https://www.theguardian.com/media-network/media-network-blog/2012/jul/17/what-is-new-news-aggregation (дата звернення: 10.06.2023).

12. Smith S. Overview of ASP.NET core MVC. Microsoft. URL: https://learn.microsoft.com/ru-ru/aspnet/core/mvc/overview?view=aspnetcore-7.0 (дата звернення: 10.06.2023).

13. Titenok Y. How to create a news aggregator website?. Sloboda-Studio. URL: https://sloboda-studio.com/blog/how-to-build-a-news-aggregator-website/ (дата звернення: 10.06.2023).

14. TutorialsPoint. PosgtreSQL - overview. TutorialsPoint. URL: https://www.tutorialspoint.com/postgresql/postgresql_overview.html (дата звернення: 10.06.2023).

15. ZetCode. C# parse HTML with AngleSharp. ZetCode. URL: https://zetcode.com/csharp/anglesharp-parse-html/ (дата звернення: 10.06.2023).

ДОДАТКИ

ДОДАТОК А

ЗАТВЕРДЖЕНО

44165850.94106 -01-ЛЗ

Програмне забезпечення агрегації інформації з сайтів новин

Технічне завдання

44165850.94106-01

Листів 13

2023

ЗМІСТ

1. Введення……..3
2. Підстава для розробки……4
3. Призначення розробки……...5
4. Вимоги до програми або програмного продукту……6
4.1 вимоги до функціональних характеристик……..6
4.2 вимоги до надійності……..6
4.3 умови експлуатації……6
4.4 вимоги до складу і параметрів технічних засобів……...7
4.5 вимоги до інформаційної і програмної сумісності……7
4.6 вимоги до маркування і упаковки……7
4.7 вимоги до транспортування і зберігання……8
5. Вимоги до програмної документації……9
6. Стадії та етапи розробки……...10
7. Порядок контролю і приймання……...11
8. Бібліографічний список……12

1. ВВЕДЕННЯ

Програмне забезпечення агрегації інформації з сайтів новин

Основна термінологія:

Агрегація даних – це процес збору та об'єднання інформації з різних джерел або джерел даних в одну єдину структуру або формат. Цей процес включає збір, обробку, фільтрацію та узагальнення даних з різних джерел з метою створення цілісної та повної картини або представлення інформації..

Причина виникнення продукту ─ присутність аналогів, які все ж таки не спрямовані на задовільнення функціональних потреб замовника.

Область застосування – агрегатор новин може бути використаний в компаніях, організаціях та установах, де важлива актуальна інформація з різних джерел. Основна перевага нашого агрегатора полягає в тому, що він збирає, фільтрує та структурує новини з різних джерел, забезпечуючи користувачам зручний доступ до необхідної інформації.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є навчальний план для спеціальності 121 Інженерія програмного забезпечення, затверджений ректором Українського державного університету науки і технологій 30.06.19 р.

Тема дипломної роботи – «Розробка програмного забезпечення агрегації інформації з сайтів новин».

Керівник – доцент Андрющенко В. О.

3.ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення: створення ефективного та зручного інструменту, що дозволяє користувачам отримувати цільовану та актуальну інформацію з різних джерел в одному місці. Основні функції програмного забезпечення включають:

– агрегація даних: програмне забезпечення здатне збирати інформацію з різних новинних сайтів і об'єднувати її в одну централізовану базу даних. Це дозволяє користувачам зручно переглядати новини з різних джерел без необхідності відвідувати кожний сайт окремо;

– фільтрація та сортування: програмне забезпечення надає можливість користувачам налаштовувати фільтри та критерії сортування для відбору самої важливої та цікавої інформації. Користувачі можуть обирати тематики, щоб отримувати лише ту інформацію, яка їх цікавить.

Експлуатаційне призначення: створення зручного та ефективного інструменту для користувачів, що дозволяє їм з легкістю отримувати актуальну та релевантну інформацію з різних джерел новин.

4. ВИМОГИ ДО ПРОГРАМИ АБО ПРОГРАМНОГО ПРОДУКТУ

4.1 Вимоги до функціональних характеристик

Вимоги до характеристик наступні:

– можливість агрегувати новини з різних сайтів в одному місці;

– забезпечення можливості вибору джерел новин для агрегації;

– фільтрація новин за категоріям;

–забезпечення сортування новин за різними критеріями.

4.2 Вимоги до надійності

Вимоги до надійності наступні:

– забезпечення регулярного резервного копіювання бази даних новин;

– гарантування стійкого функціонування програмного забезпечення при обробці великого обсягу даних та одночасного доступу користувачів;

– захист вхідної та вихідної інформації шляхом застосування механізмів шифрування та контролю доступу;

– забезпечення захисту від копіювання програмного забезпечення шляхом використання відповідних заходів захисту (наприклад, цифрові підписи або ліцензування).

4.3 Умови експлуатації

Умови експлуатації наступні:

– підтримка роботи програмного забезпечення на операційних системах Windows, Linux та macOS;

– забезпечення роботи програми при температурних умовах від 5 до 35 °C та відносній вологості в межах 30-80%;

– наявність інструкції користувача, яка пояснює основні функції та використання програмного забезпечення;

– забезпечення можливості одночасної роботи з програмним забезпеченням для кількох користувачів;

Для роботи із ПЗ достатньо однієї людини, що має досвід роботи із ПК та ознайомлена із керівництвом користувача.

4.4 Вимоги до складу і параметрів технічних засобів

Мінімальні технічні вимоги до серверної частини:

– операційна система: Windows Server або Linux з підтримкою ASP.NET Core;

– веб-сервер: IIS або Nginx;

– база даних: PostgreSQL або інша сумісна з ORM-фреймворком;

– процесор: 2 ядра з тактовою частотою 2 ГГц або вище;

– оперативна пам'ять: 4 ГБ або більше;

– доступ до Інтернету.

Мінімальні вимоги до клієнтської частини:

– операційна система: Windows 7/8/10 або Linux або macOS;

– процесор: 1 ГГц або вище;

– оперативна пам'ять: 2 ГБ або більше;

– мінімальний розмір екрану: 1280x1024 пікселів;

–браузер: Остання версія Google Chrome, Mozilla Firefox або Microsoft Edge.

4.5 Вимоги до інформаційної і програмної сумісності

Програма має функціонувати під управлінням ОС Windows Xp\7\8\10\11.

На системі має бути встановлений ASP .NET Core, PostgreSQL.

4.6 Вимоги до маркування і упаковки

Упаковка програмного продукту, включаючи документацію повинна бути захищена від пошкоджень різного роду (механічних, кліматичних). На упаковці повинно бути вказана назва продукту, номер версії (якщо вона

змінювалась), мінімальні системні вимоги. На зворотній стороні упаковки вказується розробник та його юридична адреса.

4.7 Вимоги до транспортування і зберігання

Транспортування повинне забезпечувати збереження програмного продукту, його цілісність і запобігання несанкціонованого доступу до нього. Програмний виріб міститься на хмарному носії, переданий на флешці.

5. ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу програмної документації мають входити:

- специфікація;
- текст програми;
- опис програми;
- керівництво користувача.

Програмна документація повинна відповідати вимогам ДСТУ [1].

6. СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

В табл. 1 приведені стадії та етапи розробки.

Таблиця 1. Стадії та етапи розробки

Етапи розробки	Строки виконання
Обґрунтування критеріїв розробки. Попередній вибір методів рішення задач. Визначення вимог до технічних засобів. Узгодження і затвердження технічного завдання.	01.03.23 – 10.03.23
Розробка і узгодження ТЗ	
Зовнішнє проектування	10.03.23-20.05.23
Внутрішнє проектування	
Розробка серверної частини застосунку	
Розробка клієнтської частини застосунку	
Тестування застосунку	
Рефакторинг програмного коду	

7. ПОРЯДОК КОНТРОЛЮ І ПРИЙМАННЯ

Проводиться перевірка коректного виконання програмою закладених у ній функцій, тобто здійснюється функціональне тестування програми. Також здійснюється візуальна перевірка інтерфейсу програми. Строки проведення випробувань обговорюються додатково.

Контроль виконання здійснює керівник розробки доц. Андрющенко В. О.

8. БІБЛІОГРАФІЧНИЙ СПИСОК

1. C# documentation. Microsoft. URL: https://learn.microsoft.com/en-us/dotnet/csharp/.

2. Getting started with ASP.NET MVC 5. Microsoft. URL: https://learn.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/getting-started.

3/ PostgreSQL documentation. PostgreSQL. URL: https://www.postgresql.org/docs/.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського державного університету науки і технологій
Анатолій РАДКЕВИЧ
30.06.23

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ АГРЕГАЦІЇ ІНФОРМАЦІЇ З САЙТІВ НОВИН

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
4416580.94106-01-ЛЗ

Представники підприємства-розробника
Завідувач кафедри
Вадим ГОРЯЧКІН
30.06.23

Керівник розробки
Вадим АНДРЮЩЕНКО
30.06.23

Виконавець
Володимир МАКСИМЧУК
30.06.23

Нормоконтролер
Світлана ВОЛКОВА
30.06.23

2023

ДОДАТОК Б

ЗАТВЕРДЖЕНО

44165850.94106 -01 12 01-ЛЗ

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ АГРЕГАЦІЇ ІНФОРМАЦІЇ З САЙТІВ НОВИН

Текст програми

44165850.94106-01 12 01

Листів 59

2023

ЗМІСТ

1. АНОТАЦІЯ……..3
1. Код програми ……..4

АНОТАЦІЯ

Документ 44165850.94103-01 12 01 “ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ АГРЕГАЦІЇ ІНФОРМАЦІЇ З САЙТІВ НОВИН Текст програми” входить до складу програмної документації на додаток, що реалізують програмне забезпечення агрегації інформації з сайтів новин.

У даному документі представлений текст програми. Програма написана на мові C# за об’єктно-орієнтованою методологією. Об'єм пам'яті, що займають програми комплексу, складає 12 Мб. Конфігурація комп'ютера стандартна. Комплекс функціонує в середовищі MS WINDOWS XP/7/8/8.1/10/11.

КОД ПРОГРАМИ

GenericRepository.cs:

```
public class GenericRepository<TEntity, TContext> : IGenericRepository<TEntity, TContext>
    where TEntity : BaseEntity
    where TContext : DbContext
{
    protected TContext Context;
    private readonly DbSet<TEntity> _entities;

    public GenericRepository(TContext context)
    {
        Context = context;
        _entities = Context.Set<TEntity>();
    }

    public void Add(TEntity entity)
    {
        _entities.Add(entity);
    }

    public async Task AddAsync(TEntity entity, CancellationToken cancellationToken)
    {
        await _entities.AddAsync(entity, cancellationToken);
    }

    public void AddRange(IEnumerable<TEntity> entities)
    {
        _entities.AddRange(entities);
    }
```

```
    public async Task AddRangeAsync(IEnumerable<TEntity> entities,
CancellationToken cancellationToken)
    {
        await _entities.AddRangeAsync(entities, cancellationToken);
    }

    public void Remove(TEntity entity)
    {
        _entities.Remove(entity);
    }

    public void RemoveRange(IEnumerable<TEntity> entities)
    {
        _entities.RemoveRange(entities);
    }

    public void Update(TEntity entity)
    {
        if (entity is null) throw new
ArgumentException(nameof(entity));

        Context.Entry(entity).State = EntityState.Modified;
    }

    public TEntity? Get(Guid id)
    {
        return _entities.Find(id);
    }

    public async Task<TEntity?> GetAsync(Guid id, CancellationToken
cancellationToken)
    {
        return await _entities.FindAsync(id, cancellationToken);
    }

    public IEnumerable<TEntity>? GetAll()
```

```
    {
        return _entities.ToList();
    }

    public async Task<IEnumerable<TEntity>?>
GetAllAsync(CancellationToken cancellationToken)
    {
        return await _entities.ToListAsync(cancellationToken);
    }

    public IEnumerable<TEntity>? Find(Expression<Func<TEntity, bool>>
predicate)
    {
        return _entities.Where(predicate);
    }

    public async Task<IEnumerable<TEntity>?>
FindAsync(Expression<Func<TEntity, bool>> predicate, CancellationToken
cancellationToken)
    {
        return await
_entities.Where(predicate).ToListAsync(cancellationToken);
    }

    public TEntity? SingleOrDefault(Expression<Func<TEntity, bool>>
predicate)
    {
        return _entities.SingleOrDefault(predicate);
    }

    public async Task<TEntity?>
SingleOrDefaultAsync(Expression<Func<TEntity, bool>> predicate,
CancellationToken cancellationToken)
    {
        return await _entities.SingleOrDefaultAsync(predicate,
cancellationToken);
```

```
    }

    public IEnumerable<TEntity> GetAllAsNoTracking()
    {
        return _entities.AsNoTracking();
    }
}
```

UnitOfWork.cs:

```
public class UnitOfWork<TContext> : IUnitOfWork<TContext>
    where TContext : DbContext
{
    private readonly TContext _context;

    public UnitOfWork(TContext context)
    {
        _context = context;
    }

    public void Commit()
    {
        _context.SaveChanges();
    }

    public async Task CommitAsync(CancellationToken cancellationToken)
    {
        await _context.SaveChangesAsync(cancellationToken);
    }
}
```

ContentAggregator.cs:

```
public class ContentAggregator : IContentAggregator
{
    private readonly IEnumerable<INewsSource> _newsSources;
    private readonly IArticleRepository _articleRepository;
```

```
    private readonly IArticlePreviewRepository
_articlePreviewRepository;
    private readonly ITopicRepository _topicRepository;
    private readonly INewsWebsiteRepository _newsWebsiteRepository;
    private readonly ITopicNewsWebsiteRepository
_topicNewsWebsiteRepository;
    private readonly IUnitOfWork<NewsAggregatorDbContext> _unitOfWork;
    private readonly IMapper _mapper;
    private readonly UnianSourceOptions _unianSourceOptions;
    private readonly LigaNetSourceOptions _ligaNetSourceOptions;
    private readonly ItcSourceOptions _itcSourceOptions;
    private readonly ILogger<ContentAggregator> _logger;

    public ContentAggregator(
        IEnumerable<INewsSource> newsSources,
        IArticleRepository articleRepository,
        IArticlePreviewRepository articlePreviewRepository,
        ITopicRepository topicRepository,
        INewsWebsiteRepository newsWebsiteRepository,
        ITopicNewsWebsiteRepository topicNewsWebsiteRepository,
        IUnitOfWork<NewsAggregatorDbContext> unitOfWork,
        IOptions<UnianSourceOptions> unianSourceOptions,
        IOptions<LigaNetSourceOptions> ligaNetSourceOptions,
        IOptions<ItcSourceOptions> itcSourceOption,
        IMapper mapper, ILogger<ContentAggregator> logger)
    {
        _newsSources = newsSources;
        _articleRepository = articleRepository;
        _articlePreviewRepository = articlePreviewRepository;
        _topicRepository = topicRepository;
        _newsWebsiteRepository = newsWebsiteRepository;
        _topicNewsWebsiteRepository = topicNewsWebsiteRepository;
        _unitOfWork = unitOfWork;
        _unianSourceOptions = unianSourceOptions.Value;
        _ligaNetSourceOptions = ligaNetSourceOptions.Value;
        _itcSourceOptions = itcSourceOption.Value;
```

```
        _mapper = mapper;
        _logger = logger;
    }

    public async Task AggregateAsync(CancellationToken
cancellationToken)
    {
        try
        {
            var topics = (await
_topicRepository.GetAllAsync(cancellationToken)).ToList();
            foreach (var newsSource in _newsSources)
            {
                var newsWebsite =
                    await
_newsWebsiteRepository.SingleOrDefaultAsync(n => n.Name ==
newsSource.Name,
                        cancellationToken);

                if(newsWebsite is null) continue;

                foreach (var topic in topics)
                {
                    if (await
_topicNewsWebsiteRepository.SingleOrDefaultAsync(
                            t => t.TopicId == topic.Id &&
t.NewsWebsiteId == newsWebsite.Id, cancellationToken) is null)
                    {
                        continue;
                    }

                    var articlesByTopic =
                        await _articleRepository.FindAsync(a =>
a.NewsWebsite == newsWebsite.Id && a.Topic == topic.Id,
                            cancellationToken);
```

```
                    var dateFrom = newsWebsite.Name switch
                    {
                        "Unian"                                          =>
_unianSourceOptions.StartParsingDate,
                        "Liga.net"                                       =>
_ligaNetSourceOptions.StartParsingDate,
                        "ITC" => _itcSourceOptions.StartParsingDate,
                        _ => DateTime.Now.AddDays(-2)
                    };

                    var dateTo = DateTime.Now;

                    if    (articlesByTopic    is    not    null    &&
articlesByTopic.Any())
                    {
                        dateFrom    =    articlesByTopic.Max(a    =>
a.Date).ToLocalTime().AddSeconds(1);
                    }

                    var       articlesAndPreviews       =       await
newsSource.FetchArticlesAsync(_mapper.Map<TopicDto>(topic),
                        dateFrom,
                        dateTo, cancellationToken);

                    if     (articlesAndPreviews     is     null     ||
!articlesAndPreviews.Any()) continue;

                    var  articles  =  articlesAndPreviews.Select(a  =>
a.Item2);
                    var articlePreviews = articlesAndPreviews.Select(a
=> a.Item1);

                    var  distinctArticles  =  articles.GroupBy(obj  =>
obj.SourceUrl)
                        .Select(group => group.First())
                        .ToList();
```

```
                    var           distinctArticlePreviews           =
articlePreviews.GroupBy(obj => obj.ArticleUrl)
                        .Select(group => group.First())
                        .ToList();


                    await _articleRepository.AddRangeAsync(

_mapper.Map<IEnumerable<Article>>(distinctArticles),
cancellationToken);
                    await _articlePreviewRepository.AddRangeAsync(

_mapper.Map<IEnumerable<ArticlePreview>>(distinctArticlePreviews),
cancellationToken);
                }
            }

            await _unitOfWork.CommitAsync(cancellationToken);
        }
        catch (Exception e)
        {
            _logger.LogCritical($"Aggregation error: {e.Message}");
        }
    }
}
```

BackgroundNewsAggregator.cs:

```
public class BackgroundNewsAggregator : IHostedService, IDisposable
{
    private readonly IServiceScopeFactory _scopeFactory;
    private Timer _timer;
    private readonly IConfiguration _configuration;
```

```
    public BackgroundNewsAggregator(IConfiguration configuration,
IServiceScopeFactory scopeFactory)
    {
        _configuration = configuration;
        _scopeFactory = scopeFactory;
    }

    public async Task StartAsync(CancellationToken cancellationToken)
    {
        await DoWorkAsync(null, null, cancellationToken);
        _timer = new Timer();
        var aggregationPeriod =
Convert.ToInt32(_configuration["NewsAggregationTimePeriod"]);
        _timer = new
Timer(TimeSpan.FromSeconds(aggregationPeriod).TotalMilliseconds);
        _timer.Elapsed += async (sender, e) => await
DoWorkAsync(sender, e, cancellationToken);
        _timer.Start();

    }

    private async Task DoWorkAsync(object sender, ElapsedEventArgs e,
CancellationToken cancellationToken)
    {
        using (var scope = _scopeFactory.CreateScope())
        {
            var contentAggregator =
scope.ServiceProvider.GetRequiredService<IContentAggregator>();
            await contentAggregator.AggregateAsync(cancellationToken);
        }
    }

    public Task StopAsync(CancellationToken cancellationToken)
    {
        _timer?.Stop();
        _timer?.Dispose();
```

```
        return Task.CompletedTask;
    }

    public void Dispose()
    {
        _timer?.Dispose();
    }
}
```

ItcNewsSource.cs:

```
public class ItcNewsSource<TNewsParser> : INewsSource
    where TNewsParser : IItcNewsParser
{
    public string Name { get; }
    public string Url { get; }
    private TNewsParser _newsParser;
    private readonly ItcSourceOptions _sourceOptions;
    private readonly NewsWebsiteDto _newsWebsiteDto;
    private readonly IArticlePreviewRepository
_articlePreviewRepository;
    private readonly INewsWebsiteRepository _newsWebsiteRepository;
    private readonly ITopicNewsWebsiteRepository
_topicNewsWebsiteRepository;
    private readonly IMapper _mapper;
    private readonly ILogger<ItcNewsParser> _logger;

    public ItcNewsSource(
        IOptions<ItcSourceOptions> sourceOptions,
        TNewsParser newsParser,
        INewsWebsiteRepository newsWebsiteRepository,
        ITopicNewsWebsiteRepository topicNewsWebsiteRepository,
        IMapper mapper,
        IArticlePreviewRepository articlePreviewRepository,
ILogger<ItcNewsParser> logger)
```

```
    {
        _sourceOptions = sourceOptions.Value;
        _newsParser = newsParser;
        Name = _sourceOptions.SourceName;
        Url = _sourceOptions.BaseUrl;
        _newsWebsiteRepository = newsWebsiteRepository;
        _topicNewsWebsiteRepository = topicNewsWebsiteRepository;
        _mapper = mapper;
        _articlePreviewRepository = articlePreviewRepository;
        _logger = logger;
        _newsWebsiteDto                                                    =
_mapper.Map<NewsWebsiteDto>(_newsWebsiteRepository.SingleOrDefault(n
=> n.Name == Name));
    }

    public    async    Task<List<(ArticlePreviewDto,    ArticleDto)>?>
FetchArticlesAsync(TopicDto topic, DateTime dateFrom,
        DateTime dateTo, CancellationToken cancellationToken)
    {
        try
        {
            var result = new List<(ArticlePreviewDto, ArticleDto)>();
            var topicNewsWebsite =
                await
_topicNewsWebsiteRepository.SingleOrDefaultAsync(
                    t => t.TopicId == topic.Id && t.NewsWebsiteId ==
_newsWebsiteDto.Id, cancellationToken);
            if     (topicNewsWebsite     is     null)     return     new
List<(ArticlePreviewDto, ArticleDto)>();

            var articlePreviews =
                await
_newsParser.ParseArticlePreviewsAsync(topicNewsWebsite.Url,  dateFrom,
dateTo, cancellationToken);
            if  (articlePreviews  is  null  ||  !articlePreviews.Any())
return null;
```

```
            var newsSource = await
_newsWebsiteRepository.SingleOrDefaultAsync(n => n.Name == Name,
cancellationToken);
            var newsSourceDto =
_mapper.Map<NewsWebsiteDto>(newsSource);

            var existingArticlePreviews = await
_articlePreviewRepository.GetAllAsync(cancellationToken);
            if (existingArticlePreviews is not null ||
existingArticlePreviews.Any())
            {
                articlePreviews = articlePreviews.Where(article =>
                        !existingArticlePreviews.Any(
                            existingArticle =>
existingArticle.ArticleUrl == article.ArticleUrl))
                    .ToList();
            }

            foreach (var articlePreview in articlePreviews)
            {
                articlePreview.Topic = topic;
                var article = await
_newsParser.ParseArticleAsync(articlePreview.ArticleUrl,
cancellationToken);
                if (article is null) continue;
                article.Id = articlePreview.ArticleId;
                article.Topic = topic;
                article.Source = newsSourceDto;
                article.PublicId = Guid.NewGuid();
                articlePreview.PublicArticleId = article.PublicId;
                if (articlePreview.ImageUrl is null)
                {
                    var doc = new HtmlDocument();
                    doc.LoadHtml(article.HtmlBody);
```

```
                    var                    imgNode                    =
doc.DocumentNode.SelectSingleNode("//img");
                    if (imgNode != null)
                    {
                        var                imageUrl                =
imgNode.GetAttributeValue("src", "");
                        articlePreview.ImageUrl = imageUrl;
                    }
                    else
                    {
                        articlePreview.ImageUrl = "images/itc.jpg";
                    }
                }

                result.Add((articlePreview, article));
            }

            return result;
        }
        catch (Exception e)
        {
            _logger.LogCritical($"Fetching    articles    error    in
ItcNewsSource. Exception: {e.Message}");
            return null;
        }
    }
```

ItcNewsParser.cs:

```
public class ItcNewsParser : IItcNewsParser
{
    private readonly IBrowsingContext _browsingContext;
    private readonly WebClient _webClient;
    private readonly ItcSourceOptions _sourceOptions;
    private readonly StringBuilder _stringBuilder;
    private readonly ILogger<ItcNewsParser> _logger;
```

```
    public ItcNewsParser(
        IBrowsingContext browsingContext,
        WebClient webClient,
        IOptions<ItcSourceOptions>                          sourceOptions,
ILogger<ItcNewsParser> logger)
    {
        _browsingContext = browsingContext;
        _webClient = webClient;
        _logger = logger;
        _sourceOptions = sourceOptions.Value;
        _stringBuilder = new StringBuilder();
    }

    public            async            Task<List<ArticlePreviewDto>?>
ParseArticlePreviewsAsync(
        string url,
        DateTime dateFrom,
        DateTime dateTo,
        CancellationToken cancellationToken)
    {
        try
        {
            var isFinal = false;
            var parsedArticlePreviews = new List<ArticlePreviewDto>();
            var pageCount = 1;

            while (!isFinal)
            {
                var  page  =  await  _browsingContext.OpenAsync(url  +
$"/page/{pageCount}", cancellationToken);
                var             articlePreviewsContainer             =
page.QuerySelector("#content");
                if (articlePreviewsContainer is null) break;
                var                 articlePreviews                  =
articlePreviewsContainer.QuerySelectorAll(".post");
```

```
                if (articlePreviews.Length > 1)
                {
                    for (var i = 0; i < articlePreviews.Length; i++)
                    {
                        var articleInfo = articlePreviews[i].QuerySelector(".row");
                        if (articleInfo is null) continue;
                        var articleTitle = articleInfo?.QuerySelector(".entry-title")?.TextContent.Replace("\n", "");
                        var articleUrl = articleInfo?.QuerySelector("a.thumb-responsive")?.Attributes
                            .GetNamedItem("href")
                            ?.Value;
                        var articleDate = articleInfo?.QuerySelector(".date")?.TextContent.Replace("\n", "");

                        var parsedArticleDate = DateParser.ParseDateTime(articleDate);

                        if (parsedArticleDate > dateTo) continue;

                        if (parsedArticleDate < dateFrom)
                        {
                            isFinal = true;
                            break;
                        }

                        var articleThumbnail = articleInfo.QuerySelector("a.thumb-responsive");

                        var imageUrl = articleThumbnail?.Attributes.GetNamedItem("data-bg")?.Value;
```

```
                        parsedArticlePreviews.Add(new
ArticlePreviewDto
                        {
                            Id = Guid.NewGuid(),
                            ArticleId = Guid.NewGuid(),
                            Title = articleTitle,
                            ImageUrl = imageUrl,
                            ArticleUrl = articleUrl,
                            Date = parsedArticleDate.ToUniversalTime()
                        });

                        _logger.LogInformation(
                            $"Parsed article preview in ItcNewsParser.
Page {pageCount}, preview number {i})");
                    }

                    await   Task.Delay(TimeSpan.FromMilliseconds(500),
cancellationToken);
                    if (parsedArticlePreviews.Count < 10 && pageCount
> 10) isFinal = true;
                        pageCount++;
                }
                else
                {
                    await  Task.Delay(TimeSpan.FromMilliseconds(2000),
cancellationToken);
                }
            }

            return parsedArticlePreviews;
        }
        catch (Exception e)
        {
            _logger.LogCritical($"Parsing  article  previews  error  in
ItcNewsParser: {e.Message}");
            return null;
```

```
        }
    }

    public async Task<ArticleDto?> ParseArticleAsync(string url,
CancellationToken cancellationToken)
    {
        try
        {
            var page = await _browsingContext.OpenAsync(url);

            var articleContainer = page.QuerySelector("#wrapper
.container");
            if (articleContainer is null) return null;

            var articleTitle = articleContainer.QuerySelector(".entry-
title")?.TextContent.Replace("\n", "").Trim();
            var articleDate =
articleContainer.QuerySelector(".date")?.TextContent.Trim();
            var parsedArticleDate =
DateParser.ParseDateTime(articleDate);

            var articleContent =
articleContainer.QuerySelector(".post-txt");
            if (articleContent is null) return null;
            var articleAuthor =
articleContainer.QuerySelector(".author-right a")?.TextContent;
            var isFinal = false;


            foreach (var articleElement in articleContent.Children)
            {
                if (isFinal) break;

                switch (articleElement)
                {
                    case IHtmlParagraphElement paragraph:
```

```
                        if (paragraph.TextContent != string.Empty)
                        {
                            if
(paragraph.TextContent.Contains("Джерело: "))
                            {
                                var            linkElement            =
paragraph.QuerySelector("a");
                                if (linkElement != null)
                                {
                                    var            linkUrl            =
linkElement.Attributes.GetNamedItem("href")?.Value;
                                    _stringBuilder.Append(
                                        $"<p>Джерело:                <a
href='{linkUrl}'>{linkElement.TextContent}</a></p>");
                                }

                                break;
                            }


_stringBuilder.Append($"<p>{paragraph.TextContent.Replace("\n",
"")}</p>");
                        }

                        var  imageElement  =  paragraph.QuerySelector("a
img");
                        if (imageElement != null)
                        {
                            var           paragraphImageUrl           =
imageElement?.Attributes.GetNamedItem("data-src")?.Value ??

imageElement?.Attributes.GetNamedItem("src")?.Value;
                            var           paragraphImageAlt           =
imageElement?.Attributes.GetNamedItem("alt")?.Value;
                            var          paragraphImageTitle          =
imageElement?.Attributes.GetNamedItem("title")?.Value;
```

```
                        _stringBuilder.Append(
                            $"<img       alt='{paragraphImageAlt}'
src='{paragraphImageUrl}' title='{paragraphImageTitle}'>");
                    }

                    break;
                case IHtmlHeadingElement heading:
                    if (heading.TextContent != string.Empty)
                    {

_stringBuilder.Append($"<{heading.LocalName}>{heading.TextContent}</{h
eading.LocalName}>");
                    }

                    break;

                case IHtmlImageElement htmlImageElement:
                    var                 imageUrl                =
htmlImageElement?.Attributes.GetNamedItem("data-src")?.Value ??

htmlImageElement?.Attributes.GetNamedItem("src")?.Value;
                    var                 imageAlt                =
htmlImageElement?.Attributes.GetNamedItem("alt")?.Value;
                    var                imageTitle               =
htmlImageElement?.Attributes.GetNamedItem("title")?.Value;
                    _stringBuilder.Append($"<img  alt='{imageAlt}'
src='{imageUrl}' title='{imageTitle}'>");
                    break;

                case  IHtmlElement  {  ClassName:  "quote_blue"  }
htmlQuoteElement:
                    var             paragraphElement            =
htmlQuoteElement.QuerySelector("p");
                    if (paragraphElement is not null)
                    {
                        var text = paragraphElement.TextContent;
```

```
                            if (text != string.Empty)
                            {

_stringBuilder.Append($"<p>{text}</p>");
                            }
                        }

                        break;
                    case                    IHtmlUnorderedListElement
unorderedListElement:
                        _stringBuilder.Append("<ul>");
                        foreach          (var          li          in
unorderedListElement.Children)
                        {
                            if (li is IHtmlListItemElement)
                            {

_stringBuilder.Append($"<li>{li.TextContent}</li>");
                            }
                        }

                        _stringBuilder.Append("</ul>");
                        break;
                }
            }

            var article = new ArticleDto
            {
                Title = articleTitle,
                HtmlBody = _stringBuilder.ToString(),
                Author = articleAuthor,
                Date = parsedArticleDate.ToUniversalTime(),
                SourceUrl = url
            };
```

```
            _logger.LogInformation($"Parsed article in ItcNewsParser.
Article url: {url}");
            _stringBuilder.Clear();

            return article;
        }
        catch (Exception e)
        {
            _logger.LogCritical($"Parsing    article    error    in
ItcNewsParser (Article url: {url}) {e.Message}");
            return null;
        }
    }
}
```

LigaNetNewsSource.cs:

```
public class LigaNetNewsSource<TNewsParser> : INewsSource
    where TNewsParser : ILigaNetNewsParser
{
    public string Name { get; }
    public string Url { get; }
    private TNewsParser _newsParser;
    private readonly LigaNetSourceOptions _sourceOptions;
    private readonly NewsWebsiteDto _newsWebsiteDto;
    private             readonly             IArticlePreviewRepository
_articlePreviewRepository;
    private readonly INewsWebsiteRepository _newsWebsiteRepository;
    private            readonly            ITopicNewsWebsiteRepository
_topicNewsWebsiteRepository;
    private readonly IMapper _mapper;
    private readonly ILogger<LigaNetNewsParser> _logger;

    public LigaNetNewsSource(
        IOptions<LigaNetSourceOptions> sourceOptions,
        TNewsParser newsParser,
```

```
        INewsWebsiteRepository newsWebsiteRepository,
        ITopicNewsWebsiteRepository topicNewsWebsiteRepository,
        IMapper mapper, IArticlePreviewRepository
articlePreviewRepository, ILogger<LigaNetNewsParser> logger)
    {
        _sourceOptions = sourceOptions.Value;
        _newsParser = newsParser;
        Name = _sourceOptions.SourceName;
        Url = _sourceOptions.BaseUrl;
        _newsWebsiteRepository = newsWebsiteRepository;
        _topicNewsWebsiteRepository = topicNewsWebsiteRepository;
        _mapper = mapper;
        _articlePreviewRepository = articlePreviewRepository;
        _logger = logger;
        _newsWebsiteDto =
_mapper.Map<NewsWebsiteDto>(_newsWebsiteRepository.SingleOrDefault(n
=> n.Name == Name));
    }

    public async Task<List<(ArticlePreviewDto, ArticleDto)>?>
FetchArticlesAsync(TopicDto topic, DateTime dateFrom,
        DateTime dateTo, CancellationToken cancellationToken)
    {
        try
        {
            var result = new List<(ArticlePreviewDto, ArticleDto)>();
            var topicNewsWebsite =
                await
_topicNewsWebsiteRepository.SingleOrDefaultAsync(
                    t => t.TopicId == topic.Id && t.NewsWebsiteId ==
_newsWebsiteDto.Id, cancellationToken);
            if (topicNewsWebsite is null) return new
List<(ArticlePreviewDto, ArticleDto)>();

            var articlePreviews =
```

```
                await
_newsParser.ParseArticlePreviewsAsync(topicNewsWebsite.Url, dateFrom,
dateTo, cancellationToken);
            if (articlePreviews is null || !articlePreviews.Any())
return null;

            var newsSource = await
_newsWebsiteRepository.SingleOrDefaultAsync(n => n.Name == Name,
cancellationToken);
            var newsSourceDto =
_mapper.Map<NewsWebsiteDto>(newsSource);

            var existingArticlePreviews = await
_articlePreviewRepository.GetAllAsync(cancellationToken);
            if (existingArticlePreviews is not null ||
existingArticlePreviews.Any())
            {
                articlePreviews = articlePreviews.Where(article =>
                        !existingArticlePreviews.Any(
                            existingArticle =>
existingArticle.ArticleUrl == article.ArticleUrl))
                    .ToList();
            }

            foreach (var articlePreview in articlePreviews)
            {
                articlePreview.Topic = topic;
                var article = await
_newsParser.ParseArticleAsync(articlePreview.ArticleUrl,
cancellationToken);
                if (article is null) continue;
                article.Id = articlePreview.ArticleId;
                article.Topic = topic;
                article.Source = newsSourceDto;
                article.PublicId = Guid.NewGuid();
                articlePreview.PublicArticleId = article.PublicId;
```

```
                if (articlePreview.ImageUrl is null)
                {
                    var doc = new HtmlDocument();
                    doc.LoadHtml(article.HtmlBody);

                    var imgNode = doc.DocumentNode.SelectSingleNode("//img");
                    if (imgNode != null)
                    {
                        var imageUrl = imgNode.GetAttributeValue("src", "");
                        articlePreview.ImageUrl = imageUrl;
                    }
                    else
                    {
                        articlePreview.ImageUrl = "images/liga_net.png";
                    }
                }

                result.Add((articlePreview, article));
            }

            return result;
        }
        catch (Exception e)
        {
            _logger.LogCritical($"Fetching articles error in LigaNetNewsSource. Exception: {e.Message}");
            return null;
        }
    }
}
```

LigaNetNewsParser.cs:

```
public class LigaNetNewsParser : ILigaNetNewsParser
{
    private readonly IBrowsingContext _browsingContext;
    private readonly WebClient _webClient;
    private readonly StringBuilder _stringBuilder;
    private readonly ILogger<LigaNetNewsParser> _logger;

    public LigaNetNewsParser(IBrowsingContext browsingContext,
WebClient webClient, ILogger<LigaNetNewsParser> logger)
    {
        _browsingContext = browsingContext;
        _webClient = webClient;
        _logger = logger;
        _stringBuilder = new StringBuilder();
    }

    public async Task<List<ArticlePreviewDto>?>
ParseArticlePreviewsAsync(string url, DateTime dateFrom,
        DateTime dateTo,
        CancellationToken cancellationToken)
    {
        try
        {
            var isFinal = false;
            var parsedArticlePreviews = new List<ArticlePreviewDto>();
            var pageCount = 1;

            while (!isFinal)
            {
                var page = await _browsingContext.OpenAsync(url +
$"/page/{pageCount}", cancellationToken);
```

```
                var articlePreviews = page.QuerySelectorAll(".news-
col");

                if (articlePreviews.Any())
                {
                    for (var i = 0; i < articlePreviews.Length; i++)
                    {
                        var articleInfo =
articlePreviews[i].QuerySelector("ul li");
                        var articleTitle =
articleInfo?.QuerySelector(".title")?.TextContent.Replace("\n", "");
                        if (articleTitle.ToLower().Contains("відео"))
continue;

                        var articleUrl =
articleInfo?.QuerySelector(".title")?.Attributes.GetNamedItem("href")?
.Value;
                        if (articleUrl.ToLower().Contains("video"))
continue;
                        var articleDate =
articleInfo?.QuerySelector(".time")?.TextContent.Replace("\n",
"").Trim();
                        if (articleDate is null) continue;

                        var parsedArticleDate =
DateParser.ParseDateTime(articleDate);

                        if (parsedArticleDate > dateTo) continue;

                        if (parsedArticleDate < dateFrom)
                        {
                            isFinal = true;
                            break;
                        }
```

```
                        parsedArticlePreviews.Add(new
ArticlePreviewDto
                        {
                            Id = Guid.NewGuid(),
                            ArticleId = Guid.NewGuid(),
                            Title = articleTitle,
                            Date                                        =
parsedArticleDate.ToUniversalTime(),
                            ArticleUrl = articleUrl
                        });

                        _logger.LogInformation(
                            $"Parsed      article      preview      in
LigaNetNewsParser. Page {pageCount}, preview number {i})");
                    }

                    var           articleWithImagePreviews           =
page.QuerySelectorAll(".with-photo");


                    for      (var      i      =      0;      i      <
articleWithImagePreviews.Length; i++)
                    {
                        var articleInfo = articleWithImagePreviews[i];
                        var               articleTitle               =
articleInfo?.QuerySelector(".title a")?.TextContent.Replace("\n", "");
                        if  (articleTitle.ToLower().Contains("відео"))
continue;

                        var               articleUrl                =
articleInfo?.QuerySelector(".title
a")?.Attributes.GetNamedItem("href")?.Value;
                        if    (articleUrl.ToLower().Contains("video"))
continue;
```

```
                        var                articleDate                =
articleInfo?.QuerySelector(".time")?.TextContent.Replace("\n",
"").Trim();
                        if (articleDate is null) continue;

                        var             parsedArticleDate             =
DateParser.ParseDateTime(articleDate);

                        if (parsedArticleDate > dateTo) continue;

                        var              articleThumbnail              =
articleInfo.QuerySelector("img");
                        var                  imageUrl                  =
articleThumbnail?.Attributes.GetNamedItem("data-src")?.Value;

                        if (parsedArticleDate < dateFrom)
                        {
                            isFinal = true;
                            break;
                        }

                        parsedArticlePreviews.Add(new
ArticlePreviewDto
                        {
                            Id = Guid.NewGuid(),
                            ArticleId = Guid.NewGuid(),
                            Title = articleTitle,
                            Date                                        =
parsedArticleDate.ToUniversalTime(),
                            ArticleUrl = articleUrl,
                            ImageUrl = imageUrl
                        });

                        _logger.LogInformation(
                            $"Parsed      article      preview      in
LigaNetNewsParser. Page {pageCount}, preview number {i})");
```

```
                    }
                }
                else if (page.QuerySelectorAll(".news li:not(.with-photo)").Any())
                {
                    var newsPreviews = page.QuerySelectorAll(".news li:not(.with-photo)");
                    for (var i = 0; i < newsPreviews.Length; i++)
                    {
                        var articleInfo = newsPreviews[i];
                        var articleTitle = articleInfo?.QuerySelector("a")?.TextContent.Replace("\n", "");
                        if (articleTitle.ToLower().Contains("відео")) continue;

                        var articleUrl = articleInfo?.QuerySelector("a")?.Attributes.GetNamedItem("href")?.Value;
                        if (articleUrl.ToLower().Contains("video")) continue;
                        var articleDate = articleInfo?.QuerySelector(".time")?.TextContent.Replace("\n", "").Trim();
                        if (articleDate is null) continue;

                        var parsedArticleDate = DateParser.ParseDateTime(articleDate);

                        if (parsedArticleDate > dateTo) continue;

                        if (parsedArticleDate < dateFrom)
                        {
                            isFinal = true;
                            break;
                        }
```

```
                        parsedArticlePreviews.Add(new
ArticlePreviewDto
                        {
                            Id = Guid.NewGuid(),
                            ArticleId = Guid.NewGuid(),
                            Title = articleTitle,
                            Date                                    =
parsedArticleDate.ToUniversalTime(),
                            ArticleUrl = articleUrl
                        });
                    }

                    var           articleWithImagePreviews          =
page.QuerySelectorAll(".with-photo");


                    for      (var      i      =      0;      i      <
articleWithImagePreviews.Length; i++)
                    {
                        var articleInfo = articleWithImagePreviews[i];
                        var               articleTitle              =
articleInfo?.QuerySelector("a:not(.image
a)")?.TextContent.Replace("\n", "");
                        if  (articleTitle.ToLower().Contains("відео"))
continue;

                        var                articleUrl               =
articleInfo?.QuerySelector("a:not(.image
a)")?.Attributes.GetNamedItem("href")
                            ?.Value;
                        if    (articleUrl.ToLower().Contains("video"))
continue;
                        var               articleDate               =
articleInfo?.QuerySelector(".time")?.TextContent.Replace("\n",
"").Trim();
```

```
                        var             parsedArticleDate             =
DateParser.ParseDateTime(articleDate);

                        if (parsedArticleDate > dateTo) continue;

                        var             articleThumbnail              =
articleInfo.QuerySelector(".image img");
                        var                 imageUrl                  =
articleThumbnail?.Attributes.GetNamedItem("data-src")?.Value;

                        if (parsedArticleDate < dateFrom)
                        {
                            isFinal = true;
                            break;
                        }

                        parsedArticlePreviews.Add(new
ArticlePreviewDto
                        {
                            Id = Guid.NewGuid(),
                            ArticleId = Guid.NewGuid(),
                            Title = articleTitle,
                            Date                                      =
parsedArticleDate.ToUniversalTime(),
                            ArticleUrl = articleUrl,
                            ImageUrl = imageUrl
                        });

                        _logger.LogInformation(
                            $"Parsed      article      preview      in
LigaNetNewsParser. Page {pageCount}, preview number {i})");
                    }
                }
                else
                {
```

```
                    await  Task.Delay(TimeSpan.FromMilliseconds(2000),
cancellationToken);
                    continue;
                }

                await       Task.Delay(TimeSpan.FromMilliseconds(500),
cancellationToken);
                if (parsedArticlePreviews.Count < 10 && pageCount > 10)
isFinal = true;
                pageCount++;
            }

            return parsedArticlePreviews;
        }
        catch (Exception e)
        {
            _logger.LogCritical($"Parsing  article  previews  error  in
LigaNetNewsParser: {e.Message}");
            return null;
        }
    }

    public  async  Task<ArticleDto?>  ParseArticleAsync(string  url,
CancellationToken cancellationToken)
    {
        try
        {
            var page = await _browsingContext.OpenAsync(url);

            var   articleContainer   =   page.QuerySelector(".article-
content");
            if (articleContainer is null) return null;

            var                    articleTitle                    =
articleContainer.QuerySelector("h1").TextContent.Replace("\n", "");
```

```
            var                      articleDate                      =
articleContainer.QuerySelector(".article-time")?.TextContent.Trim();
            var                  parsedArticleDate                    =
DateParser.ParseDateTime(articleDate);

            var previewImage = articleContainer.QuerySelector(".col-12
img");
            if (previewImage is not null)
            {
                var                     imageUrl                     =
previewImage.Attributes.GetNamedItem("src").Value;
                var                     imageAlt                     =
previewImage.Attributes.GetNamedItem("alt").Value;
                var                    imageTitle                    =
previewImage.Attributes.GetNamedItem("title").Value;
                _stringBuilder.Append($"<img          alt='{imageAlt}'
src='{imageUrl}' title='{imageTitle}'>");
            }

            var                   articleContent                     =
articleContainer.QuerySelector("#news-text");
            if (articleContent is null) return null;
            var articleAuthor = articleContent.QuerySelector(".author-
redactor")?.TextContent;
            var isFinal = false;


            foreach (var articleElement in articleContent.Children)
            {
                if (isFinal) break;

                switch (articleElement)
                {
                    case IHtmlParagraphElement paragraph:
                        if (paragraph.TextContent != string.Empty)
                        {
```

```
_stringBuilder.Append($"<p>{paragraph.TextContent}</p>");
                        }

                        break;
                    case IHtmlHeadingElement heading:
                        if (heading.TextContent != string.Empty)
                        {

_stringBuilder.Append($"<{heading.LocalName}>{heading.TextContent}</{h
eading.LocalName}>");
                        }

                        break;

                    case IHtmlElement { ClassName: "content-image" }
htmlElement:
                        var image = htmlElement.QuerySelector("img");
                        var                 imageUrl                =
image?.Attributes.GetNamedItem("data-src")?.Value ??

image?.Attributes.GetNamedItem("src")?.Value;
                        var                 imageAlt                =
image?.Attributes.GetNamedItem("alt")?.Value;
                        var               imageTitle                =
image?.Attributes.GetNamedItem("title")?.Value;
                        _stringBuilder.Append($"<img  alt='{imageAlt}'
src='{imageUrl}' title='{imageTitle}'>");
                        break;
                    case                    IHtmlUnorderedListElement
unorderedListElement:
                        _stringBuilder.Append("<ul>");
                        foreach          (var          li          in
unorderedListElement.Children)
                        {
                            if (li is IHtmlListItemElement)
```

```
                            {
_stringBuilder.Append($"<li>{li.TextContent}</li>");
                            }
                        }

                        _stringBuilder.Append("</ul>");
                        break;
                    }
                }

                var article = new ArticleDto
                {
                    Title = articleTitle,
                    HtmlBody = _stringBuilder.ToString(),
                    Author = articleAuthor,
                    Date = parsedArticleDate.ToUniversalTime(),
                    SourceUrl = url
                };

                _logger.LogInformation($"Parsed article in
LigaNetNewsParser. Article url: {url}");

                _stringBuilder.Clear();

                return article;
            }

            catch (Exception e)
            {
                _logger.LogCritical($"Parsing article error in
LigaNetNewsParser (Article url: {url}) {e.Message}");
                return null;
            }
        }
}
```

UnianNetNewsSource.cs:

```
public class UnianNewsSource<TNewsParser> : INewsSource
    where TNewsParser : IUnianNewsParser
{
    public string Name { get; }
    public string Url { get; }
    private TNewsParser _newsParser;
    private readonly UnianSourceOptions _sourceOptions;
    private readonly IArticlePreviewRepository
_articlePreviewRepository;
    private readonly NewsWebsiteDto _newsWebsiteDto;
    private readonly INewsWebsiteRepository _newsWebsiteRepository;
    private readonly ITopicNewsWebsiteRepository
_topicNewsWebsiteRepository;
    private readonly IMapper _mapper;
    private readonly ILogger<UnianNewsParser> _logger;

    public UnianNewsSource(
        IOptions<UnianSourceOptions> sourceOptions,
        TNewsParser newsParser,
        INewsWebsiteRepository newsWebsiteRepository,
        ITopicNewsWebsiteRepository topicNewsWebsiteRepository,
        IMapper mapper, IArticlePreviewRepository
articlePreviewRepository, ILogger<UnianNewsParser> logger)
    {
        _sourceOptions = sourceOptions.Value;
        _newsParser = newsParser;
        Name = _sourceOptions.SourceName;
        Url = _sourceOptions.BaseUrl;
        _newsWebsiteRepository = newsWebsiteRepository;
        _topicNewsWebsiteRepository = topicNewsWebsiteRepository;
        _mapper = mapper;
        _articlePreviewRepository = articlePreviewRepository;
        _logger = logger;
```

```
        _newsWebsiteDto                                                =
_mapper.Map<NewsWebsiteDto>(_newsWebsiteRepository.SingleOrDefault(n
=> n.Name == Name));
    }

    public    async    Task<List<(ArticlePreviewDto,    ArticleDto)>?>
FetchArticlesAsync(
        TopicDto topic,
        DateTime dateFrom,
        DateTime dateTo,
        CancellationToken cancellationToken)
    {
        try
        {
            var result = new List<(ArticlePreviewDto, ArticleDto)>();
            var topicNewsWebsite =
                await
_topicNewsWebsiteRepository.SingleOrDefaultAsync(
                    t => t.TopicId == topic.Id && t.NewsWebsiteId ==
_newsWebsiteDto.Id, cancellationToken);
            if     (topicNewsWebsite     is     null)     return     new
List<(ArticlePreviewDto, ArticleDto)>();

            var articlePreviews =
                await
_newsParser.ParseArticlePreviewsAsync(topicNewsWebsite.Url,  dateFrom,
dateTo, cancellationToken);
            if  (articlePreviews  is  null  ||  !articlePreviews.Any())
return null;

            var              newsSource              =              await
_newsWebsiteRepository.SingleOrDefaultAsync(n   =>   n.Name   ==   Name,
cancellationToken);
            var                      newsSourceDto                      =
_mapper.Map<NewsWebsiteDto>(newsSource);
```

```
            var existingArticlePreviews = await
_articlePreviewRepository.GetAllAsync(cancellationToken);
            if (existingArticlePreviews is not null ||
existingArticlePreviews.Any())
            {
                articlePreviews = articlePreviews.Where(article =>
                        !existingArticlePreviews.Any(
                            existingArticle =>
existingArticle.ArticleUrl == article.ArticleUrl))
                    .ToList();
            }

            foreach (var articlePreview in articlePreviews)
            {
                articlePreview.Topic = topic;
                var article = await
_newsParser.ParseArticleAsync(articlePreview.ArticleUrl,
cancellationToken);
                article.Id = articlePreview.ArticleId;
                article.Topic = topic;
                article.Source = newsSourceDto;
                article.PublicId = Guid.NewGuid();
                articlePreview.PublicArticleId = article.PublicId;
                if (articlePreview.ImageUrl is null)
                {
                    var doc = new HtmlDocument();
                    doc.LoadHtml(article.HtmlBody);

                    var imgNode =
doc.DocumentNode.SelectSingleNode("//img");
                    if (imgNode != null)
                    {
                        var imageUrl =
imgNode.GetAttributeValue("src", "");
                        articlePreview.ImageUrl = imageUrl;
                    }
```

```
                    else
                    {
                        articlePreview.ImageUrl = "images/unian.png";
                    }
                }

                result.Add((articlePreview, article));
            }

            return result;
        }
        catch (Exception e)
        {
            _logger.LogCritical($"Fetching articles error in
UnianNewsSource. Exception: {e.Message}");
            return null;
        }
    }
}
```

UnianNetNewsParser.cs:

```
public class UnianNewsParser : IUnianNewsParser
{
    private readonly IBrowsingContext _browsingContext;
    private readonly WebClient _webClient;
    private readonly UnianSourceOptions _sourceOptions;
    private readonly StringBuilder _stringBuilder;
    private readonly ILogger<UnianNewsParser> _logger;

    public UnianNewsParser(IBrowsingContext browsingContext, WebClient
webClient,
        IOptions<UnianSourceOptions> parserOptions,
ILogger<UnianNewsParser> logger)
    {
        _browsingContext = browsingContext;
```

```
        _webClient = webClient;
        _logger = logger;
        _sourceOptions = parserOptions.Value;
        _stringBuilder = new StringBuilder();
    }

    public async Task<List<ArticlePreviewDto>?>
ParseArticlePreviewsAsync(
        string url,
        DateTime dateFrom,
        DateTime dateTo,
        CancellationToken cancellationToken)
    {
        try
        {
            var isFinal = false;
            var parsedArticlePreviews = new List<ArticlePreviewDto>();
            var pageCount = 1;

            while (!isFinal)
            {
                var page = await _browsingContext.OpenAsync(url +
$"?page={pageCount}", cancellationToken);


                var articlePreviews = page.QuerySelectorAll(".list-
thumbs__item");
                if (articlePreviews.Length > 1)
                {
                    for (var i = 0; i < articlePreviews.Length; i++)
                    {
                        var articleInfo =
articlePreviews[i].QuerySelector(".list-thumbs__info");
                        var articleTitle =
articleInfo?.QuerySelector("h3 a")?.TextContent.Replace("\n", "");
```

```
                        var articleUrl =
articleInfo?.QuerySelector("h3
a")?.Attributes.GetNamedItem("href")?.Value;
                        var articleDate =
articleInfo?.QuerySelector(".list-thumbs__time")?.TextContent;
                        const string dateFormat = "HH:mm, dd.MM.yyyy";

                        var parsedArticleDate =
                            DateTime.ParseExact(articleDate,
dateFormat, CultureInfo.InvariantCulture);

                        if (parsedArticleDate > dateTo) continue;

                        if (parsedArticleDate < dateFrom)
                        {
                            isFinal = true;
                            break;
                        }

                        var articleThumbnail =
articlePreviews[i].QuerySelector(".list-thumbs__image");
                        var imageUrl =
articleThumbnail?.QuerySelector("img")?.Attributes.GetNamedItem("data-
src")
                            ?.Value;

                        parsedArticlePreviews.Add(new
ArticlePreviewDto
                        {
                            Id = Guid.NewGuid(),
                            ArticleId = Guid.NewGuid(),
                            Title = articleTitle,
                            ImageUrl = imageUrl,
                            ArticleUrl = articleUrl,
                            Date = parsedArticleDate.ToUniversalTime()
                        });
```

```
                    _logger.LogInformation(
                        $"Parsed article preview in
UnianNewsParser. Page {pageCount}, preview number {i})");
                }

                await Task.Delay(TimeSpan.FromMilliseconds(800),
cancellationToken);
                pageCount++;
            }
            else
            {
                await Task.Delay(TimeSpan.FromMilliseconds(2000),
cancellationToken);
            }
        }

        return parsedArticlePreviews;
    }
    catch (Exception e)
    {
        _logger.LogCritical($"Parsing article previews error in
UnianNewsParser: {e.Message}");
        return null;
    }
}

public async Task<ArticleDto?> ParseArticleAsync(string url,
CancellationToken cancellationToken)
{
    try
    {
        var page = await _browsingContext.OpenAsync(url,
cancellationToken);

        var articleContainer = page.QuerySelector(".article");
```

```
            if (articleContainer is null) return await
ParsePublicationAsync(url, cancellationToken);

            var articleTitle =
articleContainer.QuerySelector("h1").TextContent;
            var articleInfo =
articleContainer.QuerySelector(".article__info");
            var articleAuthor = articleInfo.QuerySelector("div
.article__author--bottom .article__author-name")
                ?.TextContent
                .Replace("\n", "");
            var articleDate = articleInfo.QuerySelector("div
.article__info-item")?.TextContent;

            var dateFormat = "HH:mm, dd.MM.yy";
            var parsedArticleDate = DateTime.ParseExact(articleDate,
dateFormat, CultureInfo.InvariantCulture);
            var articleTextPreview =
articleContainer.QuerySelector(".article__like-h2")?.TextContent;

            var articleContent =
articleContainer.QuerySelector(".article-text");

            if (articleTextPreview != string.Empty)
_stringBuilder.Append($"<h2>{articleTextPreview}</h2>");

            var isFinal = false;

            foreach (var articleElement in articleContent.Children)
            {
                if (isFinal) break;

                switch (articleElement)
                {
                    case IHtmlParagraphElement paragraph:
                        if (paragraph.TextContent != string.Empty)
```

```
                        {
_stringBuilder.Append($"<p>{paragraph.TextContent}</p>");
                        }

                        break;

                    case IHtmlHeadingElement heading:
                        if (heading.TextContent == "Вас також можуть зацікавити новини:")
                        {
                            isFinal = true;
                            break;
                        }

                        if (heading.TextContent != string.Empty)
                        {
_stringBuilder.Append($"<{heading.LocalName}>{heading.TextContent}</{heading.LocalName}>");
                        }

                        break;

                    case IHtmlElement { ClassName: "photo_block" } htmlElement:
                        var image = htmlElement.QuerySelector("img");
                        var imageUrl = image.Attributes.GetNamedItem("data-src").Value;
                        var imageAlt = image.Attributes.GetNamedItem("alt").Value;
                        var imageTitle = image.Attributes.GetNamedItem("title").Value;
                        _stringBuilder.Append($"<img alt='{imageAlt}' src='{imageUrl}' title='{imageTitle}'>");
                        break;
```

```
                }
            }

            var article = new ArticleDto
            {
                Title = articleTitle,
                HtmlBody = _stringBuilder.ToString(),
                Author = articleAuthor,
                Date = parsedArticleDate.ToUniversalTime(),
                SourceUrl = url
            };

            _logger.LogInformation($"Parsed article in UnianNewsParser. Article url: {url}");

            _stringBuilder.Clear();

            return article;
        }
        catch (Exception e)
        {
            _logger.LogCritical($"Parsing article error in UnianNewsParser (Article url: {url}) {e.Message}");
            return null;
        }
    }

    private async Task<ArticleDto?> ParsePublicationAsync(string url, CancellationToken cancellationToken)
    {
        try
        {
            var page = await _browsingContext.OpenAsync(url, cancellationToken);
```

```
            var                  publicationContainer                  =
page.QuerySelector(".publication");

            var                    publicationTitle                    =
publicationContainer?.QuerySelector("h1")?.TextContent.Replace("\n",
"");
            var                    publicationInfo                     =
publicationContainer?.QuerySelector(".publication__info");
            var                   publicationAuthor                    =
publicationContainer?.QuerySelector(".publication__left-info")

?.QuerySelector(".publication__author")?.QuerySelector("a")?.TextConte
nt.Replace("\n", "");
            var                    publicationDate                     =
publicationInfo?.QuerySelector(".time")?.TextContent;
            var                    mainImageElement                    =
publicationContainer?.QuerySelector(".publication__main-image");
            var mainImage = mainImageElement?.QuerySelector("img");
            var                     mainImageUrl                       =
mainImage?.Attributes.GetNamedItem("src")?.Value;
            var                     mainImageAlt                       =
mainImage?.Attributes.GetNamedItem("alt")?.Value;
            var                    mainImageTitle                      =
mainImage?.Attributes.GetNamedItem("title")?.Value;

            _stringBuilder.Append($"<img          alt='{mainImageAlt}'
src='{mainImageUrl}' title='{mainImageTitle}'>");


            var dateFormat = "HH:mm, dd.MM.yyyy";
            var                 parsedPublicationDate                  =
DateTime.ParseExact(publicationDate,                        dateFormat,
CultureInfo.InvariantCulture);

            var                    publicationContent                  =
publicationContainer?.QuerySelector(".publication-text");
```

```
            var isFinal = false;

            foreach (var articleElement in
publicationContent?.Children)
            {
                if (isFinal) break;

                switch (articleElement)
                {
                    case IHtmlParagraphElement paragraph:
                        if (paragraph.TextContent != string.Empty)
                        {

_stringBuilder.Append($"<p>{paragraph.TextContent}</p>");
                        }

                        break;
                    case IHtmlHeadingElement heading:
                        if (heading.TextContent == "Вас також можуть
зацікавити новини:")
                        {
                            isFinal = true;
                            break;
                        }

                        if (heading.TextContent != string.Empty)
                        {

_stringBuilder.Append($"<{heading.LocalName}>{heading.TextContent}</{h
eading.LocalName}>");
                        }

                        break;
                    case IHtmlElement { ClassName: "photo_block" }
htmlElement:
```

```
                        var image = htmlElement.QuerySelector("img");
                        var                  imageUrl                  =
image.Attributes.GetNamedItem("src").Value;
                        var                  imageAlt                  =
image.Attributes.GetNamedItem("alt").Value;
                        var                imageTitle                  =
image.Attributes.GetNamedItem("title").Value;
                        _stringBuilder.Append($"<img  alt='{imageAlt}'
src='{imageUrl}' title='{imageTitle}'>");
                        break;
                    case      IHtmlDivElement      {      ClassName:
"publication__tags" } htmlDivElement:
                        isFinal = true;
                        break;
                }
            }

            var article = new ArticleDto
            {
                Title = publicationTitle,
                HtmlBody = _stringBuilder.ToString(),
                Author = publicationAuthor,
                Date = parsedPublicationDate.ToUniversalTime(),
                SourceUrl = url
            };
            _stringBuilder.Append($"<h3>{publicationAuthor}</h3>");

            _logger.LogInformation($"Parsed      publication      in
UnianNewsParser. Publication url: {url}");

            _stringBuilder.Clear();

            return article;
        }
        catch (Exception e)
        {
```

```
            _logger.LogCritical($"Parsing publication error in
UnianNewsParser (Publication url: {url}) {e.Message}");
            return null;
        }
    }
}
```

HomeController.cs:

```
public class HomeController : Controller
{
    private readonly INewsService _newsService;
    private readonly ILogger<HomeController> _logger;
    private readonly IContentAggregator _contentAggregator;

    public HomeController(IContentAggregator contentAggregator,
ILogger<HomeController> logger, INewsService newsService)
    {
        _contentAggregator = contentAggregator;
        _logger = logger;
        _newsService = newsService;
    }

    public async Task<IActionResult> Index(CancellationToken
cancellationToken)
    {
        var lastArticlePreviews = await
_newsService.GetLastArticlePreviewAsync(24, cancellationToken);

        var ukraineArticlePreviews = await
_newsService.GetArticlePreviewsAsync(cancellationToken, "ukraine",
            1, 4);

        var warArticlePreviews = await
_newsService.GetArticlePreviewsAsync(cancellationToken, "war",
            1, 4);
```

```
        var technologyArticlePreviews = await
_newsService.GetArticlePreviewsAsync(cancellationToken, "technology",
            1, 4);

        ViewBag.LastArticlePreviews = lastArticlePreviews;
        ViewBag.UkraineArticlePreviews = ukraineArticlePreviews;
        ViewBag.WarArticlePreviews = warArticlePreviews;
        ViewBag.TechnologyArticlePreviews = technologyArticlePreviews;

        return View();
    }

    [Route("/{topic}")]
    public async Task<IActionResult> ArticlePreviews(string topic,
PaginationFilter paginationFilter, CancellationToken
cancellationToken)
    {
        try
        {
            var articlePreviews = await
_newsService.GetArticlePreviewsAsync(cancellationToken, topic,
                paginationFilter.PageNumber,
paginationFilter.PageSize);

            var topicDto = await _newsService.GetTopicAsync(topic,
cancellationToken);

            if (topicDto is null || !articlePreviews.Any() ||
articlePreviews is null) return NotFound();


            var recordsCount = await
_newsService.GetRecordsCountAsync(cancellationToken, p => p.Topic ==
topicDto.Id);
```

```
            var pageCount = (int)Math.Ceiling((double) recordsCount /
paginationFilter.PageSize);
            var          lastArticlePreviews          =          await
_newsService.GetLastArticlePreviewAsync(10, cancellationToken);

            ViewBag.LastArticlePreviews = lastArticlePreviews;
            ViewBag.PageCount = pageCount;
            ViewBag.PageNumber = paginationFilter.PageNumber;
            ViewBag.PageSize = paginationFilter.PageSize;
            ViewBag.Topic = topicDto;
            return View(articlePreviews);
        }
        catch (Exception e)
        {
            return NotFound();
        }
    }

    [Route("/{topic}/{publicArticleId}")]
    public async Task<IActionResult> GetArticle(string topic, Guid
publicArticleId, CancellationToken cancellationToken)
    {
        try
        {
            var topicDto = await _newsService.GetTopicAsync(topic,
cancellationToken);

            if (topicDto is null) return NotFound();

            var             articleDto             =             await
_newsService.GetArticleAsync(topicDto,               publicArticleId,
cancellationToken);

            if (articleDto is null) return NotFound();
```

```
            var lastArticlePreview = await
_newsService.GetLastArticlePreviewAsync(10, cancellationToken);

            ViewBag.LastArticlePreview = lastArticlePreview;
            return View("Article", articleDto);

        }
        catch (Exception e)
        {
            return NotFound();
        }
    }
}
```

NewsService.cs:

```
public class NewsService : INewsService
{
    private readonly IArticlePreviewRepository
_articlePreviewRepository;
    private readonly IArticleRepository _articleRepository;
    private readonly ITopicRepository _topicRepository;
    private readonly INewsWebsiteRepository _newsWebsiteRepository;
    private readonly IMapper _mapper;

    public NewsService(IArticlePreviewRepository
articlePreviewRepository, IArticleRepository articleRepository,
ITopicRepository topicRepository, IMapper mapper,
INewsWebsiteRepository newsWebsiteRepository)
    {
        _articlePreviewRepository = articlePreviewRepository;
        _articleRepository = articleRepository;
        _topicRepository = topicRepository;
        _mapper = mapper;
        _newsWebsiteRepository = newsWebsiteRepository;
    }
```

```
    public async Task<List<ArticlePreviewDto>> GetArticlePreviewsAsync(CancellationToken cancellationToken, string? topic = null, int pageNumber = 1,
        int pageSize = 20)
    {
        var topicFromDb = await _topicRepository.SingleOrDefaultAsync(t => t.Name.ToLower() == topic.ToLower(), cancellationToken);
        if (topicFromDb is null) throw new ArgumentException($"Topic with name {topic} doesn't exist");

        var articlePreviews =
            await _articlePreviewRepository.GetPaginatedListAsync(pageSize, pageNumber, cancellationToken, topicFromDb);

        var articlePreviewsDtos = new List<ArticlePreviewDto>();
        var topics = (await _topicRepository.GetAllAsync(cancellationToken) ?? Array.Empty<Topic>()).ToList();
        foreach (var articlePreview in articlePreviews)
        {
            var articleTopic = topics.SingleOrDefault(t => t.Id == articlePreview.Topic);
            articlePreviewsDtos.Add(new ArticlePreviewDto
            {
                Id = articlePreview.Id,
                ArticleId = articlePreview.ArticleId,
                Title = articlePreview.Title,
                ImageUrl = articlePreview.ImageUrl,
                PreviewText = articlePreview.PreviewText,
                Date = articlePreview.Date.ToLocalTime(),
                ArticleUrl = articlePreview.ArticleUrl,
                PublicArticleId = articlePreview.PublicArticleId,
                Topic = _mapper.Map<TopicDto>(articleTopic)
            });
```

```
        }

        return articlePreviewsDtos;
    }

    public async Task<List<ArticleDto>> GetArticleAsync(Guid articleId,
CancellationToken cancellationToken)
    {
        throw new NotImplementedException();
    }

    public async Task<int> GetRecordsCountAsync(CancellationToken
cancellationToken, Expression<Func<ArticlePreview, bool>>? predicate =
null)
    {
        return                                                  await
_articlePreviewRepository.GetRecordsCountAsync(cancellationToken,
predicate);
    }

    public async Task<TopicDto> GetTopicAsync(string topicName,
CancellationToken cancellationToken)
    {
        var topic = await _topicRepository.SingleOrDefaultAsync(t =>
t.Name.ToLower() == topicName.ToLower(), cancellationToken);
        return _mapper.Map<TopicDto>(topic);
    }

    public async Task<ArticleDto?> GetArticleAsync(TopicDto topic, Guid
publicArticleId,
        CancellationToken cancellationToken)
    {
        var article =
            await _articleRepository.SingleOrDefaultAsync(a => a.Topic
== topic.Id && a.PublicId == publicArticleId,
                cancellationToken);
```

```
        if (article != null)
        {
            var newsWebsite = await
_newsWebsiteRepository.GetAsync(article.NewsWebsite,
cancellationToken);

            var articleDto = new ArticleDto
            {
                Id = article.Id,
                Date = article.Date,
                Author = article.Author,
                HtmlBody = article.HtmlBody,
                PublicId = publicArticleId,
                SourceUrl = article.SourceUrl,
                Title = article.Title,
                Topic = topic
            };

            return articleDto;
        }

        return null;
    }

    public async Task<List<ArticlePreviewDto>>
GetLastArticlePreviewAsync(int count, CancellationToken
cancellationToken)
    {
        var lastArticlePreview = await
_articlePreviewRepository.GetLastArticlePreviewAsync(count,
cancellationToken);
        var articlePreviewsDtos = new List<ArticlePreviewDto>();
```

```
        var topics = (await
_topicRepository.GetAllAsync(cancellationToken) ??
Array.Empty<Topic>()).ToList();

        foreach (var articlePreview in lastArticlePreview)
        {
            var articleTopic = topics.SingleOrDefault(t => t.Id ==
articlePreview.Topic);
            articlePreviewsDtos.Add(new ArticlePreviewDto
            {
                Id = articlePreview.Id,
                ArticleId = articlePreview.ArticleId,
                Title = articlePreview.Title,
                ImageUrl = articlePreview.ImageUrl,
                PreviewText = articlePreview.PreviewText,
                Date = articlePreview.Date.ToLocalTime(),
                ArticleUrl = articlePreview.ArticleUrl,
                PublicArticleId = articlePreview.PublicArticleId,
                Topic = _mapper.Map<TopicDto>(articleTopic)
            });
        }

        return articlePreviewsDtos.ToList();
    }
}
```

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського державного університету науки і технологій
Анатолій РАДКЕВИЧ
30.06.23

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АГРЕГАЦІЇ ІНФОРМАЦІЇ З САЙТІВ

Текст програми
ЛИСТ ЗАТВЕРДЖЕННЯ
4416580.94106-01 12 01-ЛЗ

Представники підприємства-розробника
Завідувач кафедри
Вадим ГОРЯЧКІН
30.06.23

Керівник розробки
Вадим Андрющенко
30.06.23

Виконавець
Володимир МАКСИМЧУК
30.06.23

Нормоконтролер
Світлана ВОЛКОВА
30.06.23

2023