Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка модуля попередньої підготовки текстового контенту для CMS Magento

Виконав: студент IV курсу, групи СН-42
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Цимбалістий Н.Р.
(підпис) (прізвище та ініціали)

Керівник _____ Готович В. А.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Липак Г. І.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Голотенко О. С.
(підпис) (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«_08_» _червня_ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)
за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)
Студенту Цимбалістий Назарій Романович
(прізвище, ім'я, по батькові)
1. Тема роботи Розробка модуля попередньої підготовки текстового контенту для CMS Magento

Керівник роботи Готович Володимир Анаталійович, кандидат технічних наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом ректора від «*14*» *травня* 2026 року № *4/9-239*
2. Термін подання студентом завершеної роботи 22 червня 2026 р.
3. Вихідні дані до роботи Технічна документація CMS Magento (Adobe Commerce) Developer Documentation, специфікації архітектури, стандарти оптимізації контенту, офіційна документація мови програмування Python та бібліотек для обробки тексту.
4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз предметної області та обґрунтування підходів до розробки модуля підготовки контенту для CMS Magento. 2. Проєктування модуля попередньої підготовки текстового контенту для CMS Magento. 3. Реалізація та тестування модуля попередньої підготовки текстового контенту для CMS Magento. 4. Безпека життєдіяльності, основи охорони праці. Висновки. Перелік джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Повний перелік слайдів у презентації, включаючи перший та останній слайди (з номерами, через крапку).

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Гурик О. Я. (кандидат технічних наук, доцент кафедри МТ)		

7. Дата видачі завдання 26 січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	26.01.2026	*Виконано*
2.	Підбір та опрацювання літературних джерел по темі кваліфікаційної роботи	27.01.2026-16.01.2026	*Виконано*
3.	Виконання дослідження щодо методів підготовки текстового контенту для CMS Magento	17.01.2026-10.05.2026	*Виконано*
	Розроблення модуля очищення, конвертації та валідації HTML-контенту		
4.	Оформлення розділу «Аналіз предметної області та обґрунтування підходів до розробки модуля підготовки контенту для CMS Magento»	11.05.2026-17.05.2026	*Виконано*
5.	Оформлення розділу «Проектування модуля попередньої підготовки текстового контенту для CMS Magento	18.05.2026-24.05.2026	*Виконано*
6.	Оформлення розділу «Реалізація та тестування модуля попередньої підготовки текстового контенту для CMS Magento	25.05.2026-31.05.2026	*Виконано*
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	01.06.2026-08.06.2026	*Виконано*
8.	Виконання завдання до підрозділу «Основи охорони	01.06.2026-08.06.2026	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	09.06.2026-11.06.2026	*Виконано*
10.	Нормоконтроль	12.06.2026-15.06.2026	*Виконано*
11.	Перевірка на плагіат	16.06.2026	*Виконано*
12.	Попередній захист кваліфікаційної роботи	18.06.2026	*Виконано*
13.	Захист кваліфікаційної роботи	25.06.2026	

Студент _____ Цимбалістий Н. Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Готович В. А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка модуля попередньої підготовки текстового контенту для CMS Magento // Кваліфікаційна робота освітнього ступеня «Бакалавр» // Цимбалістий Назарій Романович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-42 // Тернопіль, 2026 // С. 78, рис. – 10, табл. – 14, додат. – 3, бібліогр. – 43.

Ключові слова: Magento, препроцесор контенту, PyQt5, BeautifulSoup4, MVC, пакетна обробка, e-commerce.

Кваліфікаційна робота присвячена розробці програмного модуля попередньої підготовки текстового HTML-контенту для системи управління контентом Magento. Актуальність роботи зумовлена відсутністю спеціалізованих десктопних інструментів для автоматизованого очищення та конвертації контенту відповідно до специфічних вимог платформи Magento 2.

У першому розділі проведено аналіз предметної області: досліджено архітектуру CMS Magento та її вимоги до вхідного HTML-контенту, розглянуто типові проблеми при імпорті контенту товарів, обґрунтовано вибір технологічного стеку та архітектурного патерну MVC, сформульовано функціональні та нефункціональні вимоги до модуля.

У другому розділі описано проектування модуля: розроблено структуру класів за патерном MVC, спроектовано алгоритми очищення HTML, форматування plain-text та конвертації контенту для чотирьох типів полів Magento 2. Описано архітектуру сервісного шару та механізм взаємодії компонентів через Qt-сигнали.

У третьому розділі описано реалізацію та тестування: наведено ключові фрагменти програмного коду, описано графічний інтерфейс з підтримкою одиночного та пакетного режимів роботи, представлено систему модульного тестування на основі pytest. Наведено результати тестування на реальних даних.

Об'єкт дослідження: процес підготовки текстового HTML-контенту для завантаження в CMS Magento.

Предмет дослідження: методи та алгоритми автоматизованого очищення, нормалізації та конвертації HTML-розмітки відповідно до вимог платформи Magento 2.

ANNOTATION

Development of a pre-processing module for text content in CMS Magento // Bachelor's qualification work // Tsymbalistyi Nazariy Romanovych // Ternopil Ivan Pulyu National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Sciences, group SN-42 // Ternopil, 2026 // P. 78, fig. – 10, tabl. – 14, annexes – 3, references – 43.

Keywords: Magento, content preprocessor, PyQt5, BeautifulSoup4, MVC, batch processing, e-commerce.

The qualification work is dedicated to the development of a software module for pre-processing HTML text content for the Magento content management system. The relevance of the work is determined by the lack of specialized desktop tools for automated cleaning and conversion of content according to the specific requirements of the Magento 2 platform.

The first section provides an analysis of the subject domain: the Magento CMS architecture and its HTML content requirements are studied, typical problems in importing product content are identified, the technology stack and MVC architectural pattern are justified, and functional and non-functional requirements for the module are formulated.

The second section describes the module design: the class structure based on the MVC pattern is developed, algorithms for HTML cleaning, plain-text formatting, and content conversion for four types of Magento 2 fields are designed. The architecture of the service layer and the mechanism of component interaction via Qt signals are described.

The third section covers implementation and testing: key code fragments are presented, the graphical user interface supporting single and batch processing modes is described, and the unit testing system based on pytest is introduced. Testing results on real data are presented.

Object of research: the process of preparing text HTML content for loading into the Magento CMS.

Subject of research: methods and algorithms for automated cleaning, normalization and conversion of HTML markup in accordance with the requirements of the Magento 2 platform.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (Application Programming Interface) – програмний інтерфейс застосунку; набір протоколів і правил, за якими програмні компоненти взаємодіють між собою.

CMS (Content Management System) – система управління контентом; програмне забезпечення для створення, редагування, організації та публікації цифрового контенту.

CSV (Comma-Separated Values) – текстовий формат зберігання табличних даних, у якому значення полів розділяються комами або іншими роздільниками.

DOM (Document Object Model) – об'єктна модель документа; програмний інтерфейс для HTML- та XML-документів, що представляє структуру документа у вигляді дерева вузлів.

GUI (Graphical User Interface) – графічний інтерфейс користувача; спосіб взаємодії користувача з програмою за допомогою графічних елементів.

HTML (HyperText Markup Language) – мова гіпертекстової розмітки; стандартна мова для створення та структурування вмісту вебсторінок.

MVC (Model-View-Controller) – архітектурний патерн програмного забезпечення, що розділяє застосунок на три компоненти: модель даних, вигляд та контролер.

ORM (Object-Relational Mapping) – технологія програмування, що пов'язує бази даних з концепціями об'єктно-орієнтованих мов програмування.

PyQt5 – набір Python-прив'язок до крос-платформеного фреймворку Qt 5, що використовується для розробки GUI-застосунків.

SEO (Search Engine Optimization) – пошукова оптимізація; комплекс методів підвищення позиції вебсайту в результатах пошукових систем.

SKU (Stock Keeping Unit) – одиниця складського обліку; унікальний ідентифікатор товарної позиції в каталозі.

UI (User Interface) – інтерфейс користувача; сукупність засобів взаємодії між користувачем і програмою.

XLSX (Excel Spreadsheet) – формат файлу електронних таблиць Microsoft Excel на основі стандарту Office Open XML.

XML (Extensible Markup Language) – розширювана мова розмітки; формат для зберігання і передачі структурованих даних у текстовому вигляді.

XSS (Cross-Site Scripting) – міжсайтовий скриптинг; клас вразливостей вебзастосунків, що дозволяє впровадження шкідливих скриптів.

ЗМІСТ

ВСТУП

Актуальність теми. Електронна комерція є одним із найбільш динамічно зростаючих секторів світової економіки. За даними аналітичної компанії Statista, обсяг світового ринку e-commerce у 2024 році перевищив 6 трильйонів доларів США, і ця цифра продовжує зростати [1]. Серед платформ, що забезпечують технічну основу інтернет-торгівлі, CMS Magento займає стійкі позиції завдяки гнучкості архітектури та широким можливостям кастомізації.

Одним із найбільш трудомістких процесів при запуску та підтримці інтернет-магазину на Magento є наповнення каталогу товарів. Контент, що надходить від постачальників або копіюється з інших джерел, як правило містить надлишкову HTML-розмітку, несумісні стилі та потенційно небезпечні теги. Ручне очищення сотень і тисяч товарних описів є вкрай трудомістким завданням. Актуальність використання спеціалізованого програмного забезпечення у сфері торгівлі зумовлена потребою автоматизації обліку, обробки товарної інформації та підтримки бізнес-процесів електронної комерції [2].

Існуючі онлайн-інструменти для очищення HTML є універсальними і не враховують специфічних вимог платформи Magento. Вбудовані засоби самої платформи також не забезпечують повноцінної попередньої обробки. Таким чином, розробка спеціалізованого модуля підготовки текстового контенту для CMS Magento є актуальним і практично значущим завданням.

Мета і задачі дослідження. Метою кваліфікаційної роботи є розробка програмного модуля попередньої підготовки текстового HTML-контенту для CMS Magento, що забезпечує автоматизоване очищення, нормалізацію та конвертацію розмітки відповідно до вимог платформи.

Для досягнення поставленої мети необхідно вирішити такі задачі:

– проаналізувати архітектуру CMS Magento та вимоги до HTML-контенту для різних типів полів товарної картки;

– дослідити існуючі підходи до очищення та нормалізації HTML-розмітки;

– обґрунтувати вибір технологічного стеку та архітектурного рішення;

– розробити алгоритми очищення HTML, форматування plain-text та конвертації контенту під формати Magento 2;

– реалізувати графічний інтерфейс користувача з підтримкою одиночного та пакетного режимів;

– розробити систему модульних тестів для верифікації компонентів;

– провести тестування модуля на реальних даних та проаналізувати результати.

Об'єкт дослідження: процес підготовки текстового HTML-контенту для завантаження в CMS Magento.

Предмет дослідження: методи та алгоритми автоматизованого очищення, нормалізації та конвертації HTML-розмітки відповідно до вимог платформи Magento 2.

Методи дослідження. У роботі застосовано методи об'єктно-орієнтованого проєктування та архітектурний патерн MVC, методи синтаксичного аналізу HTML-документів, модульне тестування, а також порівняльний аналіз технологічних рішень [3].

Практичне значення одержаних результатів. Розроблений модуль дозволяє скоротити час підготовки одного товарного опису з кількох хвилин до секунд, а при пакетній обробці опрацьовувати сотні файлів без участі оператора. Результати роботи можуть бути використані контент-менеджерами інтернет-магазинів на платформі Magento та адаптовані для інших CMS-платформ зі схожими вимогами до HTML-контенту [4].

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ПІДХОДІВ ДО РОЗРОБКИ МОДУЛЯ ПІДГОТОВКИ КОНТЕНТУ ДЛЯ CMS MAGENTO

1.1 Загальна характеристика платформи CMS Magento та її місце на ринку електронної комерції

Magento – відкрита платформа електронної комерції, що вперше з'явилась у 2008 році і з того часу стала одним із галузевих стандартів для побудови середніх та великих інтернет-магазинів. Платформа заснована компанією Varien і побудована на стеку PHP/MySQL з активним використанням принципів об'єктно-орієнтованого програмування. У 2018 році Magento увійшла до екосистеми Adobe, що суттєво розширило її можливості у сфері хмарного розгортання та аналітики. Використання хмарних технологій у сучасних інформаційних системах забезпечує масштабованість, віддалений доступ до ресурсів та гнучке розгортання програмних сервісів, що є важливим для e-commerce-платформ [5].

Платформа існує у двох основних редакціях. Magento Open Source розповсюджується безкоштовно під відкритою ліцензією і підходить для малого та середнього бізнесу. Adobe Commerce є комерційним продуктом із розширеним набором функцій: підтримкою B2B-комерції, інтегрованим рушієм персоналізації Adobe Target та хмарною інфраструктурою. Серед конкурентів на ринку e-commerce платформ Magento вирізняється насамперед гнучкістю архітектури.

Переваги:

– розроблена для інтернет-магазинів;

– два варіанти користування, залежно від потреб та масштабів бізнесу;

– велика база знань – Adobe пропонує багато інструкцій та документацій, спільноту експертів на Github, які викладають мануали та огляди, є неофіційні форуми;

– відносно багато налаштувань та інструментів у безкоштовній базовій версії;

– високий рівень захисту за замовчуванням, однак користувачам Magento Open Source потрібно самостійно відстежувати та застосовувати виправлення безпеки та оновлення;

– гнучкі та адаптивні налаштування, можливість максимальної кастомізації.

Недоліки:

– складне встановлення та налаштування, вже на старті впровадження CMS необхідні навички програмування;

– слабка адаптованість до українського ринку;

– висока ціна версії з платною ліцензією;

– відсутність професійної підтримки для безкоштовної версії;

– складність пошуку компетентних спеціалістів, вища вартість їх послуг порівняно з доопрацюваннями сайтів на більш популярних CMS.

На відміну від SaaS-рішень [6] – на кшталт Shopify або BigCommerce [7], Magento надає повний доступ до вихідного коду, що дозволяє реалізовувати будь-яку бізнес-логіку без обмежень хмарного провайдера. Порівняння ринкових часток провідних e-commerce платформ показано на рисунку 1.1.

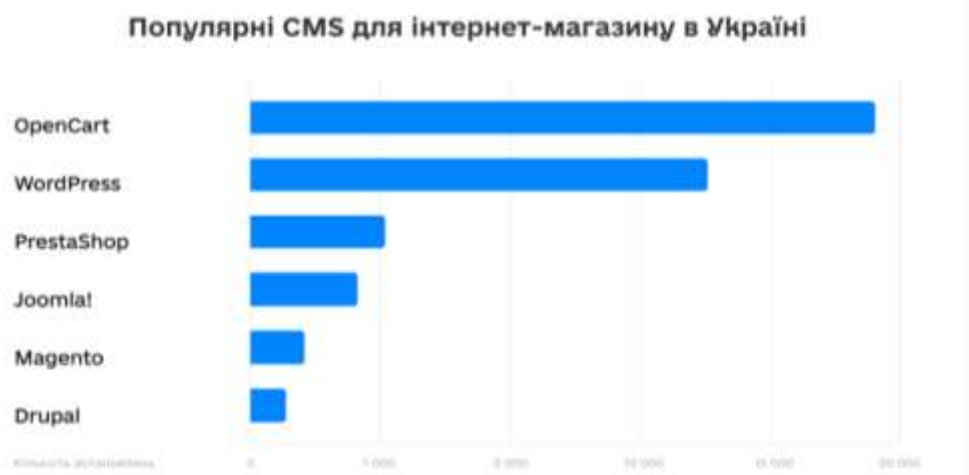

Рисунок 1.1 – Порівняння ринкових часток провідних CMS-платформ для e-commerce

Важливою характеристикою Magento з точки зору наповнення контентом є розгалужена система атрибутів товарів. Кожен товар може містити довільну кількість атрибутів різних типів: текстові рядки, числа, дати, списки значень, а також HTML-поля. Саме HTML-поля – передусім description та short_description – є основним об'єктом обробки у розроблюваному модулі [8].

Технічна основа Magento 2 суттєво відрізняється від першої версії: фреймворк Laminas (колишній Zend Framework), шаблонний рушій PHTML та система залежностей на базі Composer забезпечують модульність і розширюваність.

Платформа підтримує концепцію мультисайтовості: одна інсталяція може обслуговувати кілька незалежних магазинів зі спільним або роздільним каталогом.

1.2 Архітектура Magento 2 в контексті задачі управління контентом

Архітектура Magento 2 побудована на принципі модульності: функціональність розподілена між незалежними модулями у каталозі app/code/{Vendor}/{Module}. Кожен модуль містить контролери, моделі, репозиторії, блоки відображення та XML-файли конфігурації. Такий підхід дозволяє підключати, вимикати та перевизначати функціональність без втручання в ядро [9].

Підсистема імпорту є особливо важливою в контексті розроблюваного модуля: саме через неї відбувається масове оновлення товарного каталогу. Формат Magento Import CSV передбачає конкретний набір колонок з чітко визначеними правилами значень [10].

Помилки форматування HTML в колонках description або short_description не завжди явно сигналізуються системою – контент просто зберігається у «брудному» вигляді.

З архітектурної точки зору управління контентом у Magento 2 реалізоване через три незалежні підсистеми: каталог товарів (Catalog), CMS-підсистема

(Cms) та система імпорту (ImportExport). Схему взаємодії підсистем управління контентом зображено на рисунку 1.2.

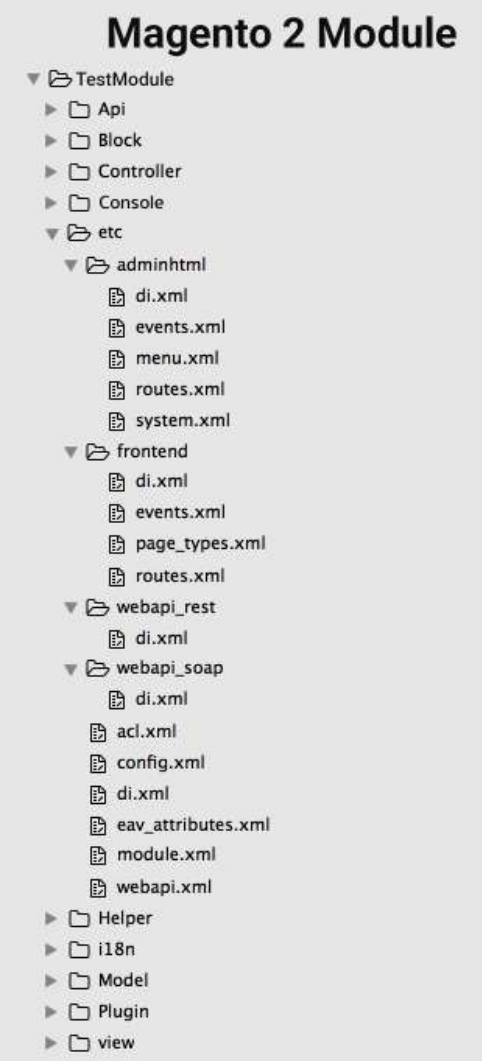

Рисунок 1.2 – Структура модуля Magento 2 та взаємодія з підсистемою управління контентом

Кешування є критично важливим аспектом продуктивності Magento. Платформа підтримує кілька рівнів кешу: конфігурації, макетів, блоків та сторінок [11]. Завдяки цьому Magento зменшує кількість повторних звернень до бази даних і прискорює формування сторінок інтернет-магазину для кінцевого користувача. Контент товарів із некоректним HTML може кешуватись і відображатись з помилками тривалий час навіть після виявлення проблеми, що підсилює важливість попередньої перевірки перед завантаженням. Це означає,

що помилки у розмітці можуть впливати не лише на одну товарну сторінку, а й на пов’язані блоки, списки товарів, результати пошуку та інші елементи інтерфейсу магазину.

1.3 Детальний аналіз вимог Magento 2 до HTML-контенту полів товарної картки

Розуміння специфічних вимог платформи до кожного типу контентного поля є фундаментом для розробки алгоритмів модуля.

1.3.1 Поле description – повний HTML-опис товару

Поле description є основним місцем розміщення детального опису товару і зберігається у базі даних як TEXT [12]. Magento виводить його вміст безпосередньо на сторінці товару, застосовуючи власні CSS-стилі активної теми. Будь-які вбудовані стилі у атрибутах style або класи у атрибутах class призводять до конфліктів з оформленням теми.

Типові прояви: некоректний розмір або колір шрифту, порушені відступи параграфів, неправильне відображення списків.

Допустимий набір HTML-тегів для поля description охоплює: блокові елементи (p, div, ul, ol, li, h1–h6, blockquote, pre, table та його похідні), рядкові елементи (a, strong, b, em, i, u, s, span, br, sup, sub) та медіа (img).

Теги script, style, iframe, object, embed, form є абсолютно неприпустимими як з точки зору безпеки, так і коректності відображення [13].

1.3.2 Поле short_description – короткий опис

Поле short_description призначене для короткого опису товару у виджетах списків та результатах пошуку [12]. Воно зберігається як VARCHAR(255), що

жорстко обмежує довжину до 255 символів – перевищення призводить до мовчазного обрізання без попередження.

Наявність HTML-тегів може спричинити відображення тегів як простого тексту у певних виджетах або некоректне формування SEO-мета-опису [13].

1.3.3 Поля CMS Static Block і CMS Page

CMS Static Block – фрагмент HTML із унікальним ідентифікатором, що вставляється у будь-яке місце шаблону через директиву {{block}} [14]. CMS Page – повноцінна сторінка з власною URL-адресою. Обидва типи допускають ширший набір структурних тегів: div, section, article, header, footer, figure, nav.

Вони також підтримують директиви шаблонного рушія Magento у подвійних фігурних дужках, які повинні залишатись недоторканими при очищенні.

1.4 Аналіз типових джерел контенту та проблем сумісності

Практика роботи з реальними інтернет-магазинами на Magento показує, що контент надходить із кількох принципово відмінних джерел, кожне з яких формує специфічний набір проблем при завантаженні на платформу.

1.4.1 Контент з Microsoft Word і Google Docs

Це найпоширеніше джерело проблемного контенту. При копіюванні тексту з Word або Google Docs браузер зберігає у буфер обміну не лише видимий текст, а й повне HTML-представлення фрагмента [15]. Типова розмітка з Microsoft Word містить: теги span із численними атрибутами style, службові атрибути mso-* від MS Office, надмірно вкладені div і зайві класи форматування . Загальний обсяг такої розмітки може у 10–20 разів перевищувати обсяг корисного текстового вмісту.

1.4.2 Контент із прайс-листів постачальників

Постачальники товарів часто надають каталоги у форматах Excel або CSV. Поле опису у таких файлах зазвичай містить або чистий текст, або HTML, сформований автоматично з корпоративних систем управління продуктами. Автоматично згенерований HTML нерідко є надмірно структурованим: кожен рядок обгорнутий у власний параграф, а таблиці характеристик мають стилі, несумісні з темою магазину.

Окремою технічною проблемою є кодування символів. Частина постачальників формує файли у кодуванні Windows-1251, і без явного зазначення кодування українські символи перетворюються на нечитабельні послідовності байтів. Модуль підготовки повинен автоматично визначати кодування та конвертувати текст у UTF-8 [16].

1.5 Огляд та порівняльний аналіз існуючих інструментів підготовки контенту

Перед розробкою власного рішення необхідно проаналізувати наявні інструменти для роботи з HTML-контентом.

1.5.1 Онлайн-сервіси очищення HTML

Найвідоміші інструменти цієї категорії – HTML Cleaner [19]., DirtyMarkup, HTML Tidy [21]. Вони надають базовий функціонал видалення надлишкових тегів та виправлення незакритої розмітки.

Проте вони не враховують специфіки Magento, не підтримують пакетну обробку та вимагають передачі контенту на сторонні сервери, що є неприйнятним при роботі з комерційними даними [21].

1.5.2 Вбудовані засоби Magento

Адміністративна панель Magento 2 включає редактор TinyMCE [22], який виконує часткову фільтрацію при вставці. Однак повного очищення від style та class атрибутів не відбувається – редактор намагається зберегти оригінальне форматування. У новіших версіях Adobe Commerce доступний Page Builder, але він орієнтований на блочну верстку промо-контенту, а не на масове наповнення каталогу товарів. Розробка окремого програмного модуля як частини інформаційної системи дозволяє локалізувати відповідальність компонентів, спростити подальшу підтримку та забезпечити можливість поступового розширення функціональності [23].

1.5.3 Програмні бібліотеки

На рівні програмного коду існує кілька зрілих бібліотек: Python BeautifulSoup4 [24], JavaScript DOMPurify [25], PHP DOMDocument [26]. Всі вони потребують написання власного коду для кожного сценарію обробки та є недоступними для нетехнічних користувачів без відповідного інтерфейсу.

Для обґрунтування вибору підходу до реалізації програмного засобу проведено порівняльний аналіз існуючих інструментів підготовки HTML-контенту для Magento. Під час аналізу враховувалися функціональні можливості рішень, зручність використання та рівень автоматизації процесу обробки контенту. Результати порівняння наведено в таблиці 1.1.

Як випливає з табл. 1.1, жоден із наявних інструментів не поєднує специфічну підтримку Magento, пакетну обробку файлів, зручний GUI та автономну роботу. Саме цей набір характеристик реалізовано у розробленому модулі.

Таблиця 1.1 – Порівняльний аналіз існуючих інструментів підготовки HTML-контенту для Magento

Критерій	**Онлайн-сервіси**	**TinyMCE**	**Page Builder**	**Бібліотеки**	**Розроблений модуль**
Специфіка Magento	Ні	Частково	Частково	Ні	Так
Пакетна обробка	Ні	Ні	Ні	Так (код)	Так
GUI без коду	Так	Так	Так	Ні	Так

Перейдемо до обґрунтування вибору технологічного стеку для розробки.

1.6 Обґрунтування вибору технологічного стеку

Вибір технологій базувався на аналізі вимог до модуля та порівнянні доступних рішень. Ключовими критеріями були: кросплатформеність, наявність зрілих бібліотек для роботи з HTML та GUI, відкритий вихідний код і низький поріг входу для подальшої підтримки.

1.6.1 Мова програмування Python

Python 3.10 обрано як основну мову реалізації. Мова поєднує читабельний синтаксис, потужну стандартну бібліотеку та розвинену екосистему пакетів для обробки тексту, HTML та табличних даних.

Завдяки великій кількості готових бібліотек Python дозволяє швидко реалізовувати функції парсингу, очищення, форматування та експорту даних без необхідності розробляти всі допоміжні механізми з нуля.

Підтримка статичної типізації через модуль typing підвищує надійність коду та спрощує рефакторинг [27]. Використання анотацій типів також полегшує роботу засобів статичного аналізу, таких як Pylance або mypy, що дає змогу виявляти частину помилок ще до запуску програми.

Порівняно з Java або C# мова Python вирізняється значно нижчим порогом входу для подальшої підтримки. Це є важливою перевагою для навчального та прикладного проєкту, оскільки код залишається зрозумілим для інших розробників і може бути легко модифікований або розширений у майбутньому. Велика спільнота користувачів і широкий вибір бібліотек спрощують пошук готових рішень та технічної підтримки під час подальшого розвитку системи.

1.6.2 Фреймворк PyQt5 для GUI

Для побудови графічного інтерфейсу обрано PyQt5 – Python-прив'язки до Qt 5. Порівняльний аналіз GUI-фреймворків для Python подано у таблиці 1.2. PyQt5 забезпечує нативний вигляд на Windows, macOS та Linux, систему сигналів і слотів для реактивної взаємодії та стилізацію через QSS [28].

Таблиця 1.2 – Порівняльний аналіз GUI-фреймворків для розробки застосунків на Python

Критерій	**PyQt5**	**Tkinter**	**wxPython**	**Dear PyGui**
Кросплатформість	Win/Mac/Linux	Win/Mac/Linux	Win/Mac/Linux	Win/Mac/Linux
Набір віджетів	Дуже широкий	Обмежений	Широкий	Середній
Стилізація	QSS (CSS)	Обмежена	Обмежена	Вбудована
Багатопотоковість	QThread	threading	threading	threading
Документація	Відмінна	Гарна	Гарна	Задовільна
Ліцензія	GPL/Commercial	LGPL	LGPL	MIT

Підтримка QThread дозволяє виконувати важкі операції у фоновому потоці без блокування інтерфейсу [29].

З даних табл. 1.2 видно, що PyQt5 є найбільш збалансованим вибором: широкий набір віджетів, QSS-стилізація та вбудована підтримка багатопотоковості через QThread роблять його оптимальним для реалізації модуля.

1.6.3 Бібліотека BeautifulSoup4

BeautifulSoup4 – провідна Python-бібліотека для синтаксичного аналізу та трансформації HTML. Вона надає зручний API для навігації по DOM-дереву та модифікації вузлів.

У поєднанні з вбудованим парсером html.parser дозволяє коректно обробляти навіть синтаксично некоректний HTML без встановлення нативних C-розширень [11].

1.6.4 Бібліотека pandas

Для обробки вхідних файлів CSV та XLSX застосовано pandas. Вона уніфікує обробку обох форматів через єдиний інтерфейс DataFrame, автоматично обробляє пропущені значення та дозволяє застосовувати функції трансформації до окремих стовпців методом apply() [30].

Для роботи з XLSX pandas використовує openpyxl як рушій.

Під час проектування інтерфейсу враховано необхідність логічного групування функцій, швидкого доступу до основних дій та зменшення когнітивного навантаження на користувача, що є важливим для інтерактивних інформаційних систем [31].

1.7 Архітектурний патерн MVC та його застосування у модулі підготовки контенту

Для організації коду застосунку обрано архітектурний патерн Model-View-Controller (MVC).

Патерн забезпечує чіткий поділ відповідальностей між компонентами, що спрощує тестування, підтримку та масштабування [32]. Загальну архітектуру модуля за патерном MVC зображено на рисунку 1.3.

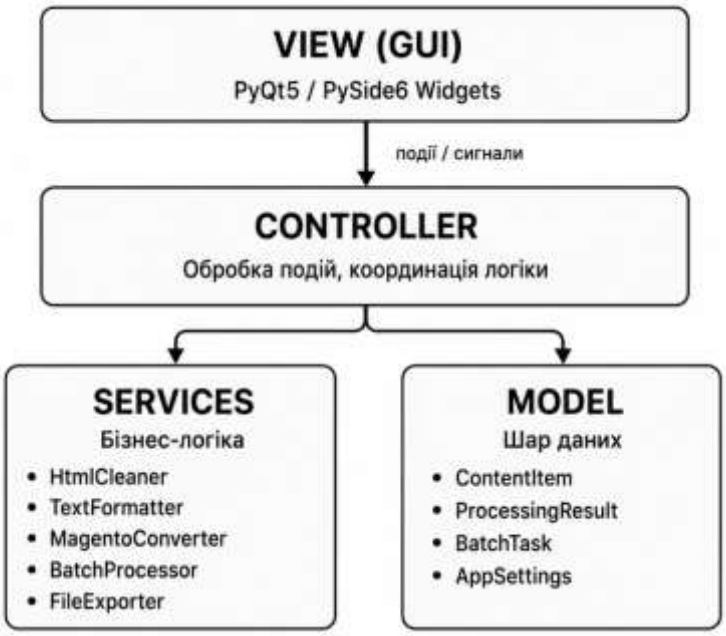

Рисунок 1.3 – Архітектура модуля підготовки контенту для CMS Magento за патерном MVC

Модель (Model) відповідає за зберігання даних та бізнес-правила. У модулі до моделей відносяться: AppSettings (налаштування програми), ContentItem (одиниця вхідного контенту), ProcessingResult (результат обробки), BatchTask (конфігурація пакетного завдання). Моделі не залежать від GUI і тестуються незалежно.

Вигляд (View) відображає дані та обробляє події інтерфейсу. Це компоненти вкладок (EditorTab, BatchTab, PreviewTab, SettingsTab) та допоміжні

віджети (AppStatusBar, DiffViewer). Вигляд отримує дані від контролера через Qt-сигнали і не містить логіки обробки.

Контролер (Controller) є посередником між моделлю і виглядом: отримує події від GUI, запускає обробку через сервіси та повертає результат. Контролерами є MainController, BatchController та SettingsController.Сервісний шар між контролером і моделлю реалізує алгоритми обробки: HtmlCleaner, TextFormatter, MagentoConverter, FileExporter.

Такий поділ дозволяє повторно використовувати сервіси у різних контролерах і тестувати їх незалежно від GUI.

1.8 Аналіз алгоритмів очищення та нормалізації HTML

Алгоритм очищення HTML є ядром розроблюваного модуля. Для його проектування розглянемо наявні підходи до санітизації розмітки.

Підхід на основі регулярних виразів є найпростішим: достатньо застосувати патерн <[^>]+> для видалення тегів. Проте він є ненадійним для реального HTML: погано обробляє вкладені теги, атрибути з кутовими дужками у значеннях та коментарі. Для виробничого використання регулярні вирази підходять лише для фінального постобробки.

Підхід на основі DOM-парсингу є значно надійнішим: документ парситься у дерево вузлів, після чого виконується обхід із модифікацією небажаних вузлів. BeautifulSoup4 реалізує саме цей підхід: decompose() видаляє тег із вмістом, unwrap() замінює тег його вмістом [33].

Підхід білого списку тегів (allowlist) є рекомендованим для безпечної санітизації: замість переліку заборонених елементів визначається набір дозволених. Будь-який тег поза списком автоматично замінюється вмістом. Для поля description це множина семантичних елементів; для CMS-полів – ширший набір із структурними тегами [34].

1.9 Постановка задачі розробки модуля підготовки контенту для CMS Magento

На основі проведеного аналізу предметної області сформульовано повний перелік вимог до розроблюваного модуля.

Система повинна забезпечувати автоматизоване очищення, форматування та конвертацію контенту для CMS Magento, а також підтримувати пакетну обробку файлів і збереження результатів у зручному для користувача форматі. Функціональні та нефункціональні вимоги до програмного засобу наведено нижче.

1.9.1 Функціональні вимоги

Модуль підготовки текстового контенту для CMS Magento повинен забезпечувати такі функціональні можливості:

– підтримка вхідних форматів: HTML з буфера обміну, plain-text, файли .html, .htm, .txt, .csv, .xlsx;

– автоматичне визначення типу вхідного контенту (HTML або plain text);

– очищення HTML від небезпечних тегів (script, style, iframe, object, embed, form) разом із вмістом;

– видалення атрибутів style та class при збереженні семантичних атрибутів href, src, alt;

– конвертація контенту для чотирьох типів полів Magento 2: description, short_description, cms_block, cms_page;

– автоматичне скорочення short_description до 255 символів по межі слова;

– генерація meta_description до 160 символів з першого параграфа;

– форматування plain-text у параграфи з автоматичним виявленням списків;

– пакетний режим обробки з відображенням прогресу та можливістю скасування;

– автоматичне визначення кодування вхідних файлів (UTF-8, Windows-1251, Latin-1);

– збереження результатів у форматах HTML, TXT, CSV та XLSX.

1.9.2 Нефункціональні вимоги

До нефункціональних вимог модуля відносяться:

– кросплатформеність: підтримка Windows 10+, Ubuntu 20.04+, macOS 11+;

– час обробки одного файлу до 1 МБ – не більше 2 секунд;

– пакетна обробка 100 файлів – не більше 30 секунд без блокування GUI;

– покриття сервісного шару модульними тестами – не менше 80%;

– збереження налаштувань між сесіями у файлі config.ini.

1.10 Висновки до першого розділу

У першому розділі кваліфікаційної роботи виконано комплексний аналіз предметної області розробки модуля підготовки текстового контенту для CMS Magento.

Досліджено архітектуру платформи Magento 2 та визначено специфічні технічні вимоги до HTML-контенту для кожного типу полів товарної картки. Встановлено, що критичними трансформаціями є: видалення небезпечних тегів, очищення атрибутів style і class, нормалізація пробілів, обрізання short_description до 255 символів та генерація meta_description до 160 символів.

Проаналізовано три основних джерела проблемного контенту: документи Microsoft Word і Google Docs з надлишковою розміткою, прайс-листи постачальників з некоректним кодуванням та контент при міграції з інших CMS. Проведено огляд і порівняльний аналіз існуючих інструментів (таблиця 1.1) – встановлено відсутність спеціалізованого десктопного рішення для Magento.

Обґрунтовано вибір технологічного стеку: Python 3.10, PyQt5 (таблиця 1.2), BeautifulSoup4, pandas. Розглянуто переваги патерну MVC (рисунок 1.3) та описано розподіл компонентів між шарами. Сформульовано функціональні та нефункціональні вимоги до модуля, які стануть основою для проектування у наступному розділі [14].

РОЗДІЛ 2. ПРОЕКТУВАННЯ МОДУЛЯ ПОПЕРЕДНЬОЇ ПІДГОТОВКИ ТЕКСТОВОГО КОНТЕНТУ ДЛЯ CMS MAGENTO

2.1 Загальна архітектура модуля та структура проекту

Проектування модуля розпочалось із визначення загальної архітектури, що відповідає сформульованим у першому розділі вимогам. За основу прийнятий архітектурний патерн MVC з додатковим сервісним шаром, що відокремлює алгоритми обробки від контролерів. Така структура забезпечує незалежне тестування сервісів і можливість їх повторного використання у різних сценаріях.

Фізична структура проекту організована у вигляді пакетів Python, де кожна директорія відповідає конкретному шару архітектури. Кореневий каталог містить точку входу main.py та файл конфігурації config.ini. Директорія app/ є основним пакетом і розподілена на підпакети: models/, controllers/, services/, views/ та utils/. Директорія tests/ містить модульні тести, що дзеркалять структуру основного пакету. Фізичну структуру проекту наведено на рисунку 2.1.

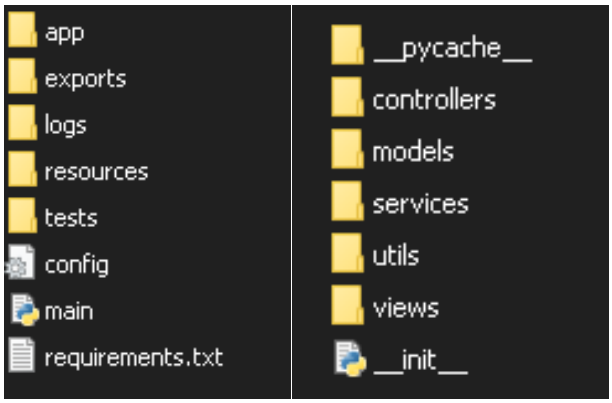

Рисунок 2.1 – Фізична структура проекту модуля підготовки контенту для CMS Magento

Точка входу main.py виконує три функції: налаштовує кореневий логер через setup_logger(), завантажує QSS-тему з файлу resources/styles/dark_theme.qss та запускає головне вікно через QApplication. Конфігурація логування

передбачає два хендлери: консольний (рівень INFO) з кольоровим форматуванням та файловий RotatingFileHandler (рівень DEBUG) з ротацією при 5 МБ та збереженням трьох резервних копій.

2.2 Проектування шару моделей

Шар моделей містить чотири класи даних, реалізованих із декоратором @dataclass стандартної бібліотеки Python. У перспективі процес тестування та розгортання модуля може бути автоматизований із використанням підходів CI/CD, що дозволить прискорити перевірку змін, зменшити кількість помилок під час оновлення програмного забезпечення та забезпечити стабільність релізів [35]. Використання dataclass усуває шаблонний код ініціалізації, порівняння об'єктів та їх рядкового представлення. UML-діаграму класів шару моделей наведено на рисунку 2.2.

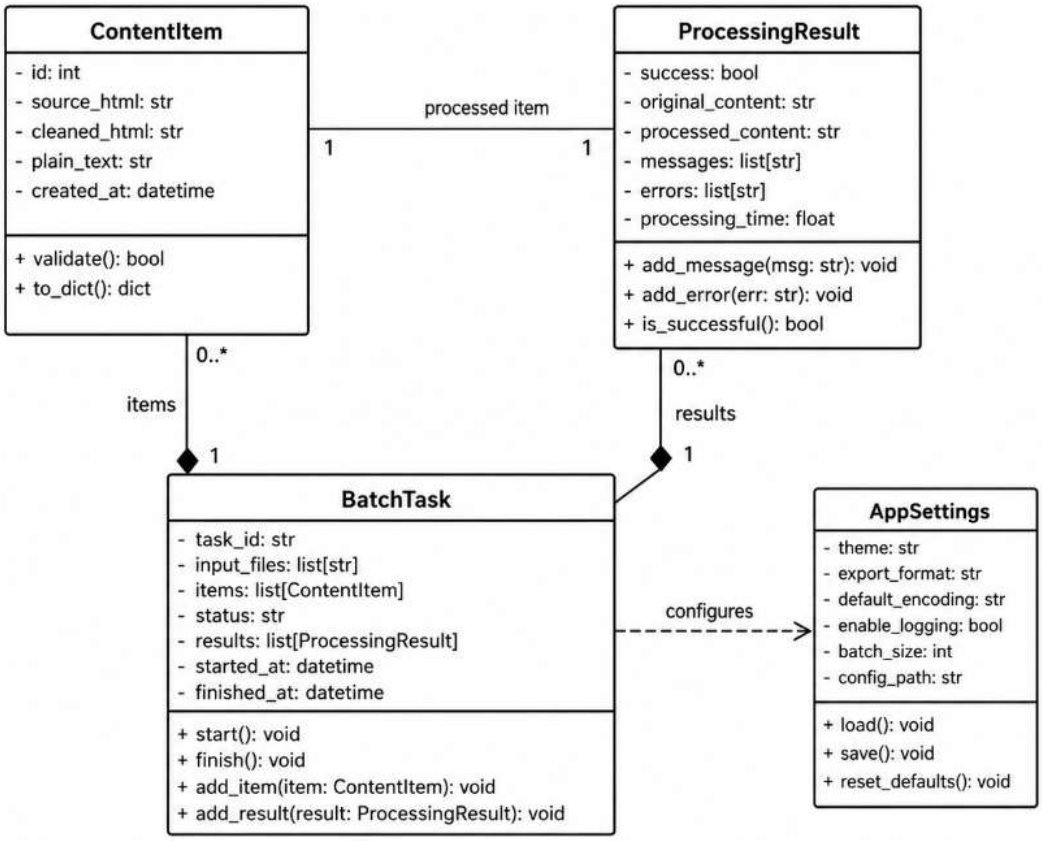

Рисунок 2.2 – UML-діаграма класів шару моделей модуля підготовки контенту для CMS Magento

Діаграма демонструє склад моделей даних та зв'язки між ними, що використовуються під час обробки, валідації та конвертації контенту. Використання окремих класів моделей забезпечує чітке розділення відповідальностей між компонентами системи та спрощує подальший супровід і розширення програмного забезпечення.

2.2.1 Клас ContentItem

ContentItem є базовою одиницею вхідного контенту, що передається між усіма сервісами обробки. При значенні content_type == "auto" метод __post_init__ автоматично визначає тип за наявністю HTML-тегів. Незмінність об'єкта після створення забезпечується конвенцією: жоден сервіс не модифікує переданий ContentItem, а завжди створює новий ProcessingResult. Перелік полів та властивостей класу ContentItem наведено у таблиці 2.1.

Таблиця 2.1 – Поля та властивості класу ContentItem

Поле / властивість	**Тип**	**За замовч.**	**Призначення**
raw_content	str	–	Вхідний текст або HTML без змін
content_type	str	"auto"	Тип контенту: "html", "text" або "auto"
source_file	Optional[str]	None	Шлях до файлу-джерела (пакетний режим)
created_at	datetime	now()	Мітка часу створення для логування
is_empty	bool (prop)	–	True якщо вміст порожній або лише пробіли
char_count	int (prop)	–	Кількість символів у raw_content
preview	str (prop)	–	Перші 120 символів без тегів для логів
filename	str (prop)	–	Ім'я файлу або "clipboard" якщо немає

Таблиця 2.1 демонструє те, що клас ContentItem інкапсулює не лише сам вміст, а й службову інформацію, необхідну для подальшої обробки та трасування даних у системі. Наявність обчислюваних властивостей спрощує роботу сервісів, оскільки дозволяє отримувати ключові характеристики контенту без дублювання логіки в різних компонентах програмного забезпечення.

2.2.2 Клас ProcessingResult

ProcessingResult є уніфікованою моделлю результату будь-якого етапу обробки. Метод __post_init__ автоматично обчислює статистику: кількість символів до і після та відсоток скорочення. Фабричні методи ok() та failure() спрощують створення типових результатів. Властивість status_summary формує рядок для статус-бару у форматі «4 230 → 1 876 символів (55.7% скорочення)». Поля та властивості класу ProcessingResult наведено у таблиці 2.2.

Таблиця 2.2 – Поля та властивості класу ProcessingResult

Поле / властивість	**Тип**	**За замовч.**	**Призначення**
original	str	–	Вхідний текст до обробки
processed	str	–	Текст після обробки
success	bool	True	Прапорець успішного завершення
errors	list[str]	[]	Список критичних помилок
warnings	list[str]	[]	Список попереджень
source_file	Optional[str]	None	Шлях до файлу-джерела
stats	dict	{}	Автообчислювана статистика обробки
has_errors	bool (prop)	–	True якщо список errors непорожній
reduction_percent	float (prop)	–	Відсоток скорочення тексту
status_summary	str (prop)	–	Рядок статистики для статус-бару

Таблиця 2.2 відображає структуру моделі ProcessingResult, яка використовується для стандартизованого представлення результатів роботи всіх сервісів системи. Такий підхід забезпечує єдиний формат обміну даними між компонентами програмного засобу та спрощує обробку повідомлень про успішне виконання операцій, попередження і помилки.

2.2.3 Клас AppSettings

AppSettings зберігає налаштування програми з персистентністю через config.ini. Метод load() зчитує значення з конвертуванням типів (getint, getboolean, get). При відсутності або пошкодженні файлу повертаються значення за замовчуванням. Метод save() записує поточний стан, попередньо зчитавши існуючі секції для збереження ручних налаштувань. Метод reset() відновлює всі поля до дефолтів без збереження на диск. Такий підхід забезпечує збереження користувацьких параметрів між запусками програми та дозволяє швидко відновити стандартну конфігурацію у разі некоректних змін. Поля класу AppSettings наведено у таблиці 2.3.

Таблиця 2.3 – Поля класу AppSettings та їх значення за замовчуванням

Поле	**Тип**	**За замовч.**	**Призначення**
font_size	int	13	Розмір шрифту редактора (пікселів)
wrap_text	bool	True	Перенос довгих рядків у редакторі
show_line_numbers	bool	False	Відображення номерів рядків
auto_copy	bool	False	Авто-копіювання результату в буфер
default_content_type	str	"product_description"	Тип Magento за замовчуванням
export_dir	str	"exports"	Папка збереження результатів

Наведені в таблиці параметри визначають поведінку користувацького інтерфейсу та основні налаштування процесу обробки контенту. Централізоване зберігання конфігурації в класі AppSettings спрощує керування параметрами програми та забезпечує їх узгоджене використання всіма компонентами системи.

2.2.4 Клас BatchTask

BatchTask є об'єктом конфігурації пакетного завдання. Властивість effective_column повертає column_name або "description" як fallback. Властивість is_empty дозволяє контролеру валідувати завдання до запуску. Поля класу BatchTask наведено у таблиці 2.4.

Таблиця 2.4 – Поля класу BatchTask

Поле	**Тип**	**За замовч.**	**Призначення**
file_paths	list[str]	[]	Список шляхів до файлів
convert_for_magento	bool	True	Запуск MagentoConverter після очищення
magento_content_type	str	"product_description"	Тип контенту для конвертації
column_name	Optional[str]	None	Колонка CSV/XLSX (None → "description")
created_at	datetime	now()	Мітка часу для логування

Таблиця 2.4 відображає параметри, необхідні для виконання пакетної обробки файлів різних форматів. Використання окремого класу BatchTask дозволяє інкапсулювати конфігурацію завдання в одному об'єкті та спрощує взаємодію між контролером пакетної обробки і сервісним шаром.

2.3 Проектування сервісного шару

Сервісний шар реалізує алгоритми обробки контенту і є найбільш технічно насиченою частиною модуля. Кожен сервіс є stateless-класом без мутабельного стану, що отримує дані через параметри і повертає ProcessingResult. Зведений публічний API всіх сервісів наведено у табл. А.1 (див. додаток А).

2.3.1 Сервіс HtmlCleaner

HtmlCleaner є основним сервісом очищення HTML-розмітки. Алгоритм clean_html() базується на DOM-парсингу через BeautifulSoup4: обходить блокові теги верхнього рівня зі списку BLOCK_TAGS (p, div, ul, ol, h1–h6). Метод _get_text_with_links() застосовує шість правил нормалізації пробілів після збірки рядка. Правила нормалізації наведено у таблиці А.2.

Блок-схему алгоритму очищення HTML наведено на рисунку 2.3.

Рисунок 2.3 – Блок-схема алгоритму очищення HTML

Множина processed відстежує вже оброблені теги для уникнення дублювання при вкладеній структурі.

2.3.2 Сервіс MagentoConverter

MagentoConverter конвертує очищений HTML для конкретного поля Magento 2. Метод convert() є диспетчером, що маршрутизує виклик до одного з чотирьох спеціалізованих методів залежно від content_type.

Перелік небезпечних тегів, що видаляються сервісом MagentoConverter разом із вмістом, наведено у таблиці 2.5.

Таблиця 2.5 – Небезпечні теги що видаляються сервісом MagentoConverter

Тег	**Причина видалення**	**Спосіб видалення**
script	Виконання довільного JS-коду на фронтенді (XSS)	decompose() – разом із вмістом
style	Конфлікт з CSS-стилями теми Magento	decompose() – разом із вмістом
iframe	Вбудовування зовнішнього контенту, вектор атак	decompose() – разом із вмістом
object	Вбудовування плагінів (Flash та ін.)	decompose() – разом із вмістом
embed	Вбудовування мультимедіа без контролю	decompose() – разом із вмістом
form	Фішингові форми у контенті товару	decompose() – разом із вмістом
input	Елементи форм у контенті товару	decompose() – разом із вмістом
button	Довільні кнопки з обробниками подій	decompose() – разом із вмістом
meta	Перевизначення мета-тегів сторінки	decompose() – разом із вмістом
base	Зміна базового URL сторінки	decompose() – разом із вмістом

Дозволені HTML-теги для кожного типу поля Magento 2 відрізняються залежно від призначення поля. Порівняння наведено у таблиці 2.6.

Для короткого опису виконується максимальне спрощення до plain text, тоді як для опису товару, CMS-блоків і CMS-сторінок зберігаються лише ті HTML-елементи, які є безпечними та функціонально доцільними.

Таблиця 2.6 – Дозволені HTML-теги для кожного типу поля Magento 2

Тег або група	**description**	**short_desc**	**cms_block / cms_page**
p, ul, ol, li	Так	Ні (plain text)	Так
h1–h6	Так	Ні	Так
a (з href)	Так	Ні	Так
strong, b, em, i, u, s	Так	Ні	Так
span, br, sup, sub	Так	Ні	Так
img (з src, alt)	Так	Ні	Так
table, thead, tbody, tr, th, td	Так	Ні	Так

Блок-схему алгоритму конвертації наведено на рисунку 2.4.

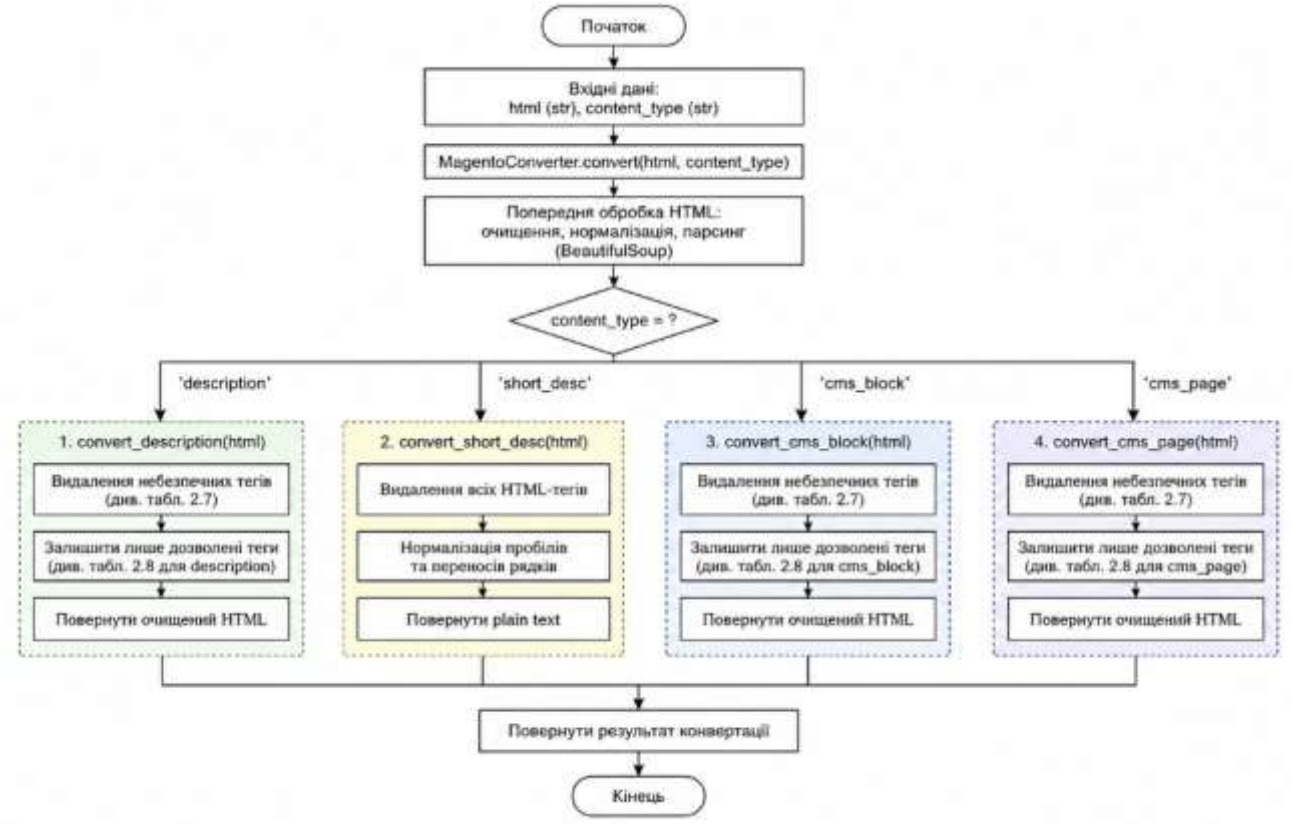

Рисунок 2.4 – Блок-схема алгоритму конвертації контенту сервісом MagentoConverter з чотирма гілками dispatch

Такий підхід зменшує ризик XSS-атак, запобігає вставленню неконтрольованого зовнішнього контенту та забезпечує сумісність результату з механізмами відображення Magento 2.

2.3.3 Сервіс BatchProcessor та алгоритм визначення кодування

BatchProcessor реалізує послідовну обробку списку файлів із підтримкою скасування та прогрес-колбека. Для кожного файлу метод _read_file() виконує багаторівневе визначення кодування: послідовно пробує UTF-8, Windows-1251 та Latin-1 з режимом errors="strict". При невдачі всіх трьох кодувань читає файл у UTF-8 з errors="replace", що гарантує успішне зчитування будь-якого файлу. Блок-схему алгоритму визначення кодування наведено на рисунку 2.5.

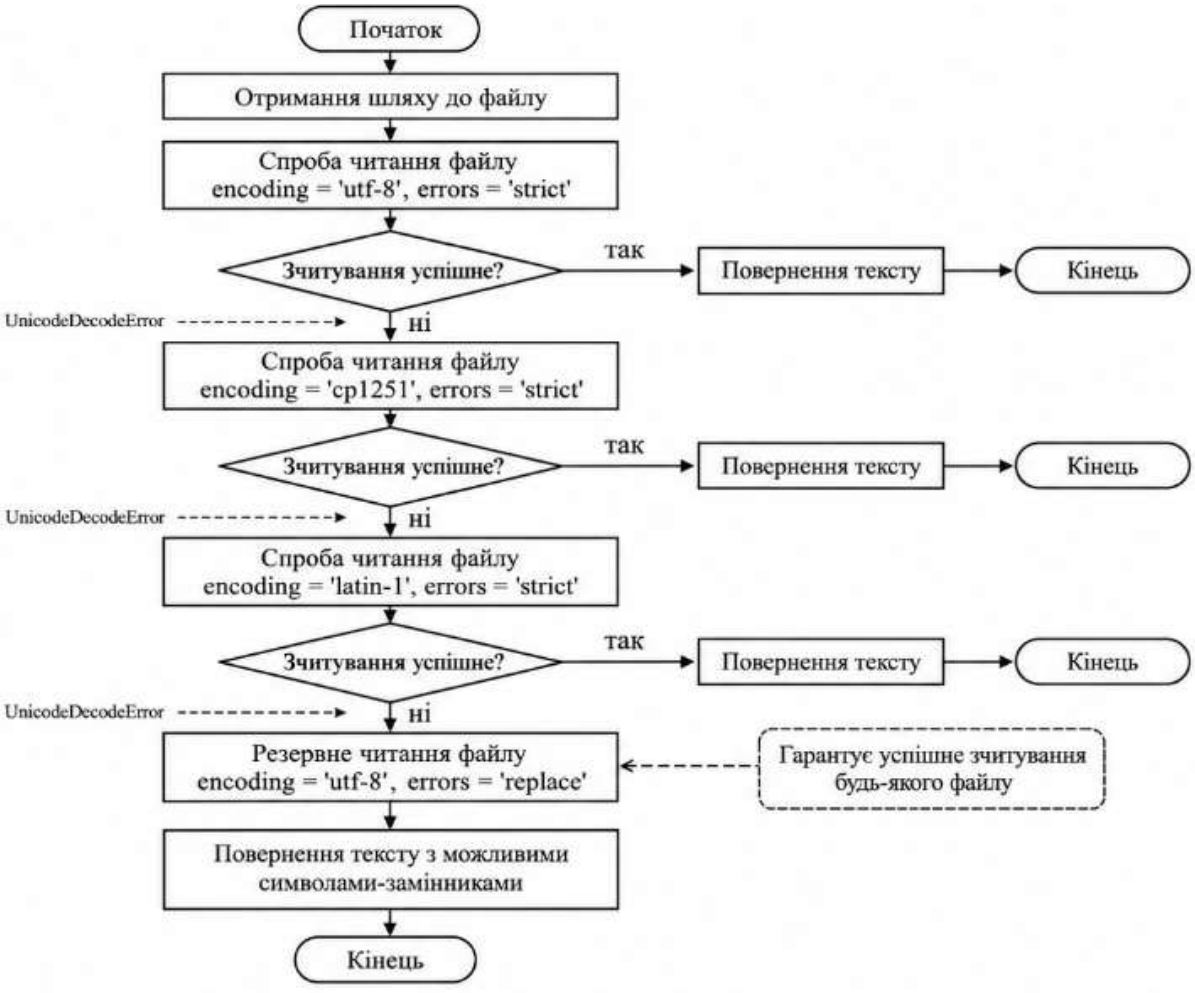

Рисунок 2.5 – Блок-схема алгоритму автоматичного визначення кодування файлу у методі _read_file()

Метод cancel() встановлює прапорець _cancel_requested, що перевіряється на початку кожної ітерації циклу. Поточний файл дообробляється повністю,

після чого обробка зупиняється. Такий підхід гарантує, що жоден файл не потрапить до результатів у частково обробленому стані [15].

2.4 Проектування шару контролерів

Контролери є посередниками між GUI та сервісним шаром. Всі три контролери успадковують від QObject для підтримки механізму Qt-сигналів і слотів. Зведену таблицю Qt-сигналів усіх контролерів наведено у таблиці 2.7.

Таблиця 2.7 – Qt-сигнали контролерів модуля

Контролер	**Сигнал**	**Тип даних**	**Коли емітується**	**Хто обробляє**
MainController	processing_done	object	Завершення обробки	EditorTab, PreviewTab
MainController	status_updated	str	Будь-яка зміна стану	AppStatusBar
MainController	clipboard_pasted	str, str	Вставка з буфера	EditorTab
BatchController	progress_updated	int, int, str	Початок обробки файлу	BatchTab (ProgressBar)
BatchController	batch_finished	list	Завершення всіх файлів	BatchTab (таблиця)
BatchController	status_updated	str	Зміна стану обробки	AppStatusBar
BatchController	batch_cancelled	–	Скасування обробки	BatchTab
SettingsController	settings_changed	object	Будь-яка зміна налаштувань	MainCtrl, BatchCtrl, SettingsTab

Таблиця демонструє організацію взаємодії між контролерами та компонентами графічного інтерфейсу за допомогою механізму сигналів і слотів Qt, що забезпечує слабке зв'язування між окремими модулями системи.

2.4.1 Клас MainController – pipeline обробки

MainController координує обробку одиничного контенту у вкладці Редактор. Метод process_content() реалізує триетапний pipeline: ContentItem → HtmlCleaner або TextFormatter → MagentoConverter. Фінальний результат зберігається у _last_result для подальшого копіювання або збереження.

На першому етапі контролер отримує вхідний текст від вкладки редактора та створює об’єкт ContentItem, у якому зберігається початковий вміст і визначається його тип. Якщо користувач передав HTML-контент, обробка спрямовується до сервісу HtmlCleaner, який виконує очищення розмітки від небезпечних тегів, зайвих атрибутів і надлишкових пробілів. Якщо ж вхідні дані є звичайним текстом, вони передаються до TextFormatter, де перетворюються у структурований HTML із параграфами, списками та посиланнями.

На другому етапі результат первинної обробки передається до MagentoConverter. Цей сервіс адаптує очищений контент відповідно до вибраного типу поля Magento, наприклад product_description, short_description, cms_block або cms_page.

Після завершення конвертації MainController зберігає результат у змінній _last_result, емітує сигнал processing_done та оновлює статус програми через status_updated. Такий pipeline забезпечує послідовність обробки, ізоляцію відповідальностей між сервісами та можливість повторного використання окремих компонентів у пакетному режимі.

Діаграму послідовності pipeline обробки наведено на рис. Б.1.1 (див. додаток Б).

Метод paste_from_clipboard() зчитує вміст через QGuiApplication.clipboard().mimeData() з пріоритетом HTML-формату перед plain text. Метод highlight_html_tags() підсвічує теги для QTextEdit: екранує кутові дужки, потім обгортає паттерни у span зі стилем color:#569cd6 для темної теми. Таким чином, MainController виконує роль центрального координатора

процесу обробки одиничного контенту, забезпечуючи взаємодію між графічним інтерфейсом і сервісним шаром без прямої залежності між їх компонентами.

2.4.2 Клас BatchController – фонове виконання через QThread

BatchController управляє пакетною обробкою у фоновому потоці через внутрішній клас _BatchWorker(QThread). Схему взаємодії компонентів при пакетній обробці наведено на рис. Б.1.2 (див. додаток Б). Метод start_batch() перевіряє відсутність активного воркера (is_running), створює новий _BatchWorker, підключає сигнали та запускає через worker.start().

2.4.3 Клас SettingsController – синхронізація налаштувань

SettingsController є єдиним джерелом правди щодо налаштувань програми. Метод update_many() застосовує кілька змін з одним збереженням і одним сигналом, що уникає зайвих операцій запису на диск. Сигнал settings_changed підключений у MainWindow одразу до кількох слотів для синхронізації export_dir у всіх компонентах.

2.5 Проектування графічного інтерфейсу користувача

Графічний інтерфейс побудований як QMainWindow з QTabWidget (чотири вкладки), кастомним AppStatusBar та темним QSS-оформленням, натхненним темою VS Code. Загальну схему кольорів визначено у таблиці 2.8.

Вкладка EditorTab розділена на дві зони через QSplitter: ліва – поле введення вхідного контенту, права – відображення результату. Підсвічування HTML-тегів у обох панелях реалізоване через метод highlight_html_tags() MainController, що формує HTML-рядок зі span-елементами для кольорового виділення тегів та значень атрибутів.

Вкладка BatchTab забезпечує повний цикл пакетної обробки: вибір файлів через QFileDialog, налаштування параметрів, запуск з відображенням прогресу та перегляд результатів у таблиці.

Таблиця 2.8 – Компоненти графічного інтерфейсу модуля

Компонент	**Клас Qt**	**Призначення**	**Ключові елементи**
EditorTab	QWidget + QSplitter	Обробка контенту	QTextEdit x2, QComboBox
BatchTab	QWidget + QTable	Пакетна обробка файлів	QListWidget, QTableWidget, QProgressBar
PreviewTab	QWidget + QSplitter	Перегляд HTML результату	QTextEdit x2 (рендер + код)
SettingsTab	QWidget + QScroll	Налаштування програми	QSpinBox, QCheckBox, QComboBox
AppStatusBar	QStatusBar	Статусні повідомлення	QLabel x2, QTimer (8 с)
DiffViewer	QWidget + QSplitter	Порівняння до/після	QTextEdit x2 з підсвічуванням
MainWindow	QMainWindow	Головне вікно, меню, збірка	QTabWidget, MenuBar, AppStatusBar

Таблиця QTableWidget містить п'ять колонок: ім'я файлу, статус, символів до, символів після та відсоток скорочення. Рядки з помилками автоматично виділяються червоним кольором.

AppStatusBar підтримує чотири типи повідомлень із різними кольорами фону: info (синій #007acc), success (зелений #16825d), error (червоний #a12020, не зникає автоматично), warning (жовтий #7d6608). Метод on_status_updated() визначає тип за початковим emoji повідомлення.

2.6 Проектування системи валідації вхідних даних

Модуль validators.py реалізує три класи валідаторів зі статичними методами, що повертають кортеж (bool, str). Методи валідаторів та умови відхилення наведено у таблиці А.3.

2.7 Проектування системи логування

Модуль logger.py реалізує ієрархічну систему логування на базі стандартної бібліотеки Python logging. Кореневий логер «magento_prep» налаштовується один раз при старті і обслуговується двома хендлерами, налаштування яких наведено у таблиці 2.9. Використання двох хендлерів дозволяє одночасно забезпечити оперативне відображення основних повідомлень у консолі та збереження детальної технічної інформації у файловому журналі. Консольний хендлер призначений для швидкого контролю стану програми під час запуску й виконання основних операцій, тоді як файловий хендлер фіксує розширені відомості про роботу сервісів, контролерів, помилки обробки файлів і події пакетного режиму. Такий підхід спрощує пошук причин збоїв, аналіз роботи модуля після завершення сеансу та подальше супроводження програмного забезпечення.

Таблиця 2.9 – Налаштування хендлерів системи логування

Хендлер	**Рівень**	**Формат повідомлення**	**Особливості**
StreamHandler (консоль)	INFO	%(levelname)s | %(name)s | %(message)s	ANSI-кольори; підтримка Windows VirtualTerminalProcessing
RotatingFileHandler	DEBUG	%(asctime)s | %(levelname)-8s | %(name)s | %(message)s	logs/app.log; 5 МБ x 3 резервні копії; UTF-8

Кожен клас отримує дочірній логер через get_logger(__class__.__name__), що формує ім'я виду «magento_prep.HtmlCleaner» або «magento_prep.MainController». Дочірні логери успадковують налаштування кореневого без дублювання хендлерів [36].

2.8 Проектування системи констант

Файл constants.py централізує всі фіксовані значення програми. Використання єдиного файлу констант дозволяє змінювати ліміти в одному місці без пошуку по всій кодовій базі [17]. Ключові константи модуля та місця їх використання наведено у таблиці А.4.

2.9 Діаграма взаємодії компонентів при обробці контенту

Для ілюстрації повного циклу роботи модуля розглянемо послідовність взаємодії компонентів при виконанні типового сценарію одиночної обробки у вкладці Редактор. Повну діаграму послідовності наведено на рисунку 2.8.

1. Користувач натискає «Вставити з буфера» у EditorTab.
2. EditorTab викликає ctrl.paste_from_clipboard() на MainController.
3. MainController зчитує вміст через QGuiApplication.clipboard().mimeData() і визначає тип (HTML або text).
4. MainController емітує clipboard_pasted(raw, type) та status_updated(" HTML вставлено").
5. EditorTab відображає вміст у лівому полі з підсвічуванням тегів через highlight_html_tags().
6. Користувач вибирає тип контенту у QComboBox і натискає «Обробити».
7. EditorTab викликає ctrl.process_content(raw, "auto", magento_type).
8. MainController створює ContentItem; якщо HTML – викликає html_cleaner.clean(), інакше – text_formatter.format().

9. MainController передає очищений рядок у magento_converter.convert(processed, magento_type).

10. MainController зберігає фінальний result у _last_result, за потреби копіює у буфер (auto_copy).

11. MainController емітує processing_done(result) та status_updated(result.status_summary).

12. EditorTab оновлює праве поле виводу, статистику та активує кнопки дій.

13. EditorTab емітує result_ready(result) → MainWindow → PreviewTab.show_result(result).

14. AppStatusBar відображає повідомлення з кольоровим індикатором типу.

Таким чином, у межах одного сценарію послідовно задіюються всі основні шари модуля: інтерфейс, контролер, моделі, сервіси обробки та механізм відображення результату.

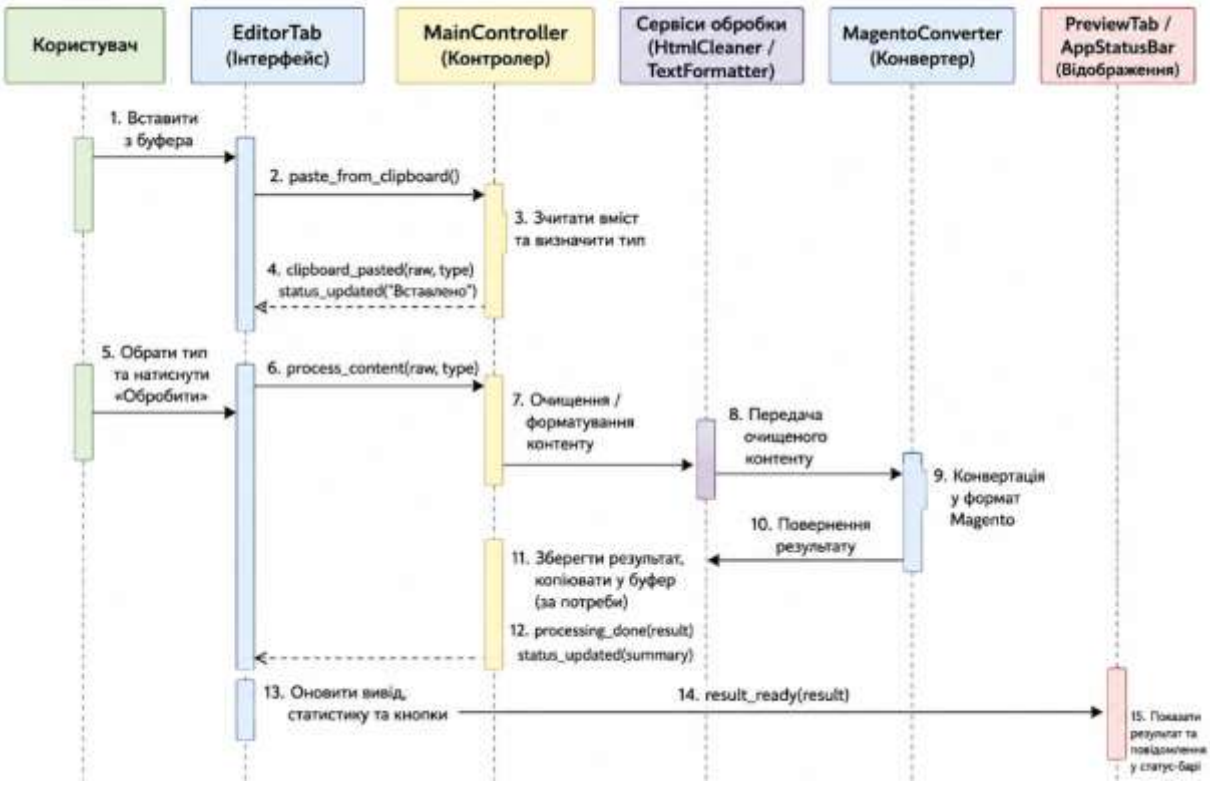

Рисунок 2.6 – Діаграма послідовності повного циклу обробки контенту у модулі підготовки для CMS Magento

2.10 Висновки до другого розділу

У другому розділі кваліфікаційної роботи виконано повне проектування модуля попередньої підготовки текстового контенту для CMS Magento.

Спроектовано шар моделей із чотирма @dataclass-класами. Для кожного класу складено таблиці полів та властивостей з типами і значеннями за замовчуванням. Описано механізм персистентності AppSettings через config.ini та автоматичного обчислення статистики у ProcessingResult.

Детально спроектовано сервісний шар із зведеною таблицею публічного API: алгоритм DOM-парсингу HtmlCleaner із шістьма правилами нормалізації пробілів, чотирьохрежимний MagentoConverter з таблицями небезпечних та дозволених тегів, багатопотоковий BatchProcessor з алгоритмом визначення кодування.

Спроектовано шар контролерів із зведеною таблицею Qt-сигналів: триетапний pipeline MainController, фонова архітектура BatchController через QThread. Розроблено GUI у складі семи компонентів з темним QSS-оформленням, систему валідації, систему логування та централізований модуль констант. Описано повний цикл взаємодії компонентів у 14 кроках [18].

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ МОДУЛЯ ПОПЕРЕДНЬОЇ ПІДГОТОВКИ ТЕКСТОВОГО КОНТЕНТУ ДЛЯ CMS MAGENTO

3.1 Організація процесу розробки та середовище виконання

Реалізація модуля виконувалась із дотриманням принципів чистого коду (Clean Code): кожна функція виконує одну відповідальність, назви змінних і методів відображають їх призначення, а магічні числа винесені у константи [15]. Версія Python 3.10 обрана через підтримку структурного патерн-матчингу та вдосконаленої типізації через PEP 604 та PEP 612, що підвищує читабельність і безпеку типів у кодовій базі.

Для управління залежностями використовується файл requirements.txt із зафіксованими версіями всіх пакетів. Розгортання середовища виконується через Python venv, що гарантує відтворюваність: будь-який розробник, клонувавши репозиторій і виконавши pip install -r requirements.txt, отримає ідентично налаштоване середовище. Перелік залежностей представлено у таблиці 3.1.

Таблиця 3.1 – Залежності модуля та їх призначення

Пакет	**Версія**	**Призначення**
PyQt5	5.15.x	Побудова графічного інтерфейсу користувача
beautifulsoup4	4.12.x	Парсинг та трансформація HTML-розмітки
pandas	2.1.x	Читання та запис табличних форматів CSV, XLSX
openpyxl	3.1.x	Рушій pandas для роботи з форматом XLSX
pytest	7.4.x	Фреймворк для модульного тестування
pytest-cov	4.1.x	Вимірювання покриття коду тестами

Середовище розробки – Visual Studio Code із розширеннями Pylance (статичний аналіз типів), Black Formatter (автоматичне форматування) та Pylint (статичний аналіз коду). Black форматує код відповідно до PEP 8 при кожному

збереженні файлу, що забезпечує єдиний стиль у всій кодовій базі [16]. Pylint налаштований на рівень суворості C (конвенції), що дозволяє виявляти потенційні помилки ще до запуску коду.

3.2 Реалізація точки входу та ініціалізації застосунку

Точка входу main.py відповідає за послідовну ініціалізацію всіх підсистем: логування, теми оформлення та головного вікна. Код точки входу показано у лістингу 3.1.

Лістинг 3.1 – Точка входу застосунку – файл main.py

```
 def main():
logger = setup_logger("magento_prep")
logger.info("Magento Content Preprocessor v%s starting...",
            APP_VERSION)
app = QApplication(sys.argv)
app.setApplicationName(APP_NAME)
app.setApplicationVersion(APP_VERSION)
theme_path = os.path.join(os.path.dirname(__file__),
             "resources", "styles", "dark_theme.qss")
if os.path.exists(theme_path):
    with open(theme_path, "r", encoding="utf-8") as f:
        app.setStyleSheet(f.read())
    logger.debug("QSS theme loaded from %s", theme_path)
else:
    logger.warning("QSS theme not found, using default style")
window = MainWindow()
window.show()    sys.exit(app.exec_())
```

Ініціалізація виконується у чіткій послідовності: спочатку налаштовується логер, що гарантує запис усіх наступних подій, потім створюється QApplication – обов'язковий контейнер для будь-якого Qt-застосунку, далі завантажується QSS-тема і лише після цього створюється головне вікно. Якщо файл теми відсутній, застосунок запускається зі стандартним оформленням Qt без переривання роботи – наявність теми не є критичною залежністю.

Використання APP_NAME та APP_VERSION із модуля констант замість рядкових літералів у main.py є важливим архітектурним рішенням: при зміні

версії продукту достатньо оновити один рядок у constants.py, і зміна автоматично відобразиться в заголовку вікна, діалозі «Про програму» та у файлах логів.

3.3 Реалізація шару моделей

Усі чотири моделі реалізовано як Python dataclass із автоматичним обчисленням похідних даних у методі __post_init__. Такий підхід мінімізує шаблонний код та гарантує коректність обчислень при кожному створенні об'єкта. Декоратор @dataclass генерує методи __init__, __repr__ та __eq__ автоматично, що суттєво скорочує обсяг коду без втрати функціональності [17].

3.3.1 Реалізація класу ContentItem

ContentItem реалізує автоматичне визначення типу контенту при ініціалізації. Регулярний вираз _HTML_TAG_RE = re.compile(r"<[a-zA-Z][^>]*>") визначено на рівні модуля як скомпільований об'єкт для максимальної продуктивності при багаторазовому використанні. Реалізацію класу показано у лістингу В.1 (див. додаток В).

3.3.2 Реалізація класу ProcessingResult

ProcessingResult автоматично обчислює статистику обробки у __post_init__ та надає фабричні методи для зручного створення типових об'єктів результату. Властивість status_summary формує рядок для статус-бару з emoji-індикатором. Реалізацію класу відображено у лістингу В.2 (див. додаток В).

3.4 Реалізація сервісу HtmlCleaner

HtmlCleaner є найбільш складним сервісом модуля з точки зору алгоритмічної реалізації. Клас не має мутабельного стану – всі константи

(BLOCK_TAGS, ALLOWED_INLINE_TAGS) визначено як атрибути класу, що дозволяє безпечно використовувати один екземпляр у різних контекстах, включаючи багатопотокове середовище [18].

3.4.1 Метод clean_html() – основний алгоритм

Основний алгоритм реалізовано у методі clean_html(), що виконує DOM-парсинг та структурований обхід документа. Код методу представлено у лістингу В.3 (див. додаток В).

Важливим аспектом реалізації є двоетапна обробка: спочатку небезпечні теги видаляються методом decompose() разом із усім вмістом, і лише після цього виконується обхід блокових тегів для збирання чистого контенту. Такий порядок гарантує, що вміст видалених тегів не потрапить до результату навіть у разі складної вкладеної структури.

3.4.2 Метод _get_text_with_links() та нормалізація пробілів

Метод _get_text_with_links() рекурсивно обходить дочірні вузли тега і збирає рядок із збереженням семантичних вбудованих елементів. Після збирання застосовуються шість правил нормалізації через скомпільовані регулярні вирази. Реалізацію методу наведено у лістингу В.4 (див. додаток В).

Використання попередньо скомпільованих регулярних виразів як атрибутів класу є важливою оптимізацією: компіляція виразу виконується один раз при завантаженні модуля, а не при кожному виклику методу. При пакетній обробці тисяч текстових фрагментів ця оптимізація суттєво впливає на загальний час виконання [19].

На скриншоті видно, що ліва панель з візуальним редактором відображає очищений текст із збереженою структурою, а права панель – відповідний чистий HTML-код із підсвічуванням тегів.

Підсвічування тегів у правій панелі не впливає на сам результат обробки, а використовується лише для зручності візуального аналізу сформованого HTML-коду.

Завдяки цьому користувач може швидко перевірити структуру документа, переконатися у відсутності небезпечних тегів та оцінити коректність збереження семантичних елементів, таких як абзаци, списки, посилання, виділення жирним шрифтом і курсивом.

Такий режим перегляду є корисним під час ручної перевірки результату перед копіюванням або експортом підготовленого контенту для подальшого завантаження у Magento. Результат роботи сервісу у інтерфейсі редактора зображено на рисунку Б.2.1 (див. додаток Б).

3.5 Реалізація сервісу MagentoConverter

MagentoConverter реалізує чотири методи конвертації, об'єднані спільними приватними методами санітизації. Диспетчерський підхід через словник dispatch замість серії if-elif є більш розширюваним: додавання нового типу поля Magento потребує лише одного рядка у словнику без модифікації умовних гілок [20].

3.5.1 Диспетчер та методи конвертації

Реалізацію диспетчера convert() та методу prepare_short_description() показано у лістингу В.5 (див. додаток В).

Деталь реалізації: обрізання short_description виконується до MAGENTO_SHORT_DESC_MAX - 1 символів (254), щоб залишити місце для символу «…» (U+2026), який є одним Unicode-символом. Використання rsplit(" ", 1)[0] гарантує обрізання по межі слова, що зберігає читабельність тексту.

Такий підхід дозволяє дотримуватися встановлених обмежень Magento без втрати цілісності текстового фрагмента.

3.5.2 Методи санітизації атрибутів та генерації SEO-метаданих

Реалізацію методів _sanitize_attributes() та generate_product_attributes() представлено у лістингу В.6 (див. додаток В). Ці методи відповідають за видалення зайвих або потенційно небезпечних атрибутів HTML-тегів, а також за формування додаткових полів товарної картки, необхідних для коректного імпорту контенту в Magento

Метод generate_product_attributes() є зручним фасадом для масового імпорту: він повертає словник із трьома готовими полями для рядка Magento Import CSV. При пакетній обробці прайс-листів саме цей метод викликається для кожного рядка таблиці, що дозволяє одночасно підготувати всі текстові поля товарної картки.

3.6 Реалізація сервісу TextFormatter

TextFormatter реалізує форматування plain-text контенту в HTML-структуру. Основним завданням сервісу є перетворення неструктурованого тексту, отриманого від постачальника, у семантичну HTML-розмітку, придатну для поля description платформи Magento.

3.6.1 Виявлення та конвертація URL і email

Регулярні вирази для виявлення URL та email скомпільовано на рівні класу. Реалізацію методу format_plain_text_with_links() подано у лістингу 3.2.

Після виявлення URL-адрес вони автоматично перетворюються на HTML-посилання з використанням відповідного атрибута `href`, що забезпечує їхню клікабельність у браузері. Для електронних адрес додатково формується посилання типу `mailto:`, при цьому передбачено перевірку, яка запобігає повторній обробці адрес, що вже містяться всередині існуючих HTML-посилань.

Лістинг 3.2 – Метод format_plain_text_with_links() сервісу TextFormatter

```
     _URL_RE   = re.compile(
    r"https?://[\w\-._~:/?#\[\]@!$&'()*+,;=]+")
_EMAIL_RE = re.compile(
    r"([a-zA-Z0-9._%+\-]+@[a-zA-Z0-9.\-]+\.[a-zA-Z]{2,})")
def format_plain_text_with_links(self, text: str) -> list[str]:
    formatted: list[str] = []
    for line in text.splitlines():
        stripped = line.strip()
        if not stripped:
            continue
        result = self._URL_RE.sub(
            r'<a href="\g<0>">\g<0></a>', stripped)
        # Email не обробляти якщо вже всередині href
        def _email_replace(m):
            addr = m.group(1)
            if f'href="{addr}"' in result:
                return m.group(0)
            return f'<a href="mailto:{addr}">{addr}</a>'
        result = self._EMAIL_RE.sub(_email_replace, result)
        formatted.append(f"<p>{result}</p>")    return formatted
```

3.6.2 Автоматичне виявлення та форматування списків

Метод detect_and_format_lists() автоматично розпізнає маркований та нумерований список у тексті. Реалізацію методу відображено у лістингу В.7 (див. додаток В). Регулярний вираз _UL_RE розпізнає різні типи маркерів: дефіс (-), зірочку (*), маркер (•), а також Unicode-символи en-dash (\u2013) та bullet (\u2022). Це охоплює практично всі варіанти маркованих списків, що зустрічаються у текстах постачальників. Паттерн _OL_RE підтримує різні розділювачі після цифри: крапку (1.), дужку (1)) та квадратну дужку (1]).

3.7 Реалізація контролерів та механізму Qt-сигналів

Контролери реалізують взаємодію між GUI та сервісним шаром виключно через механізм Qt-сигналів і слотів. Це забезпечує слабке зв'язування компонентів: вигляд не має прямого доступу до методів сервісів і не знає про їх реалізацію. Такий підхід відповідає принципу інверсії залежностей (Dependency Inversion Principle) з набору SOLID [21].

3.7.1 Реалізація BatchController – фонове виконання

BatchController реалізує пакетну обробку у фоновому потоці через внутрішній клас _BatchWorker. Використання окремого робочого потоку дозволяє виконувати тривалі операції обробки без блокування графічного інтерфейсу та забезпечує коректне відображення прогресу виконання пакетного завдання.Реалізацію воркера та методів start_batch() і cancel_batch() відображено у лістингу В.8 (див. додаток В). Вигляд вкладки пакетної обробки у стані очікування завантаження файлів показано на рисунку 3.1.

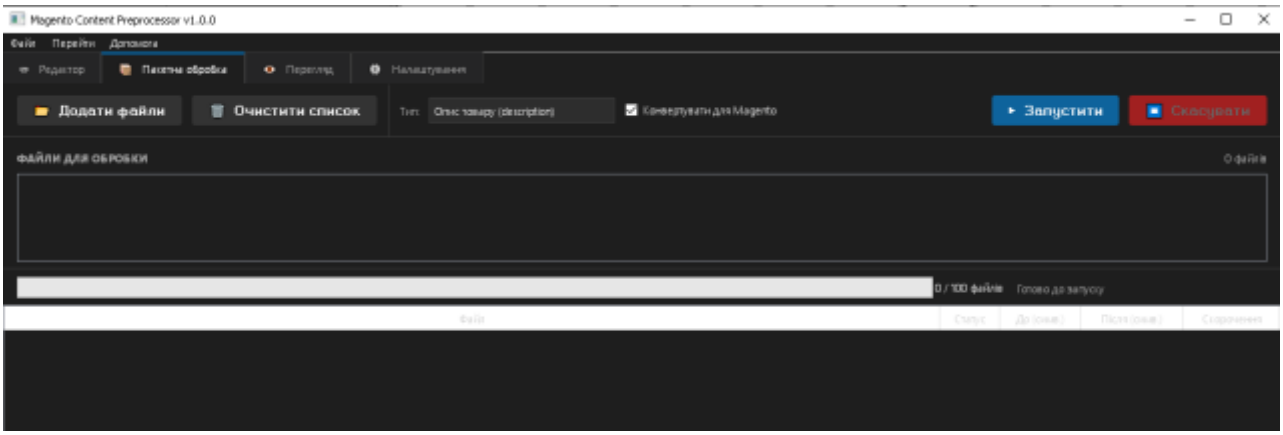

Рисунок 3.12 – Вкладка Пакетна обробка: панель файлів,

Видно таблицю результатів із колонками Файл, Статус, До (симв.), Після (симв.), Скорочення, а також ProgressBar зі статусом «0 / 100 файлів – Готово до запуску».

3.8 Реалізація графічного інтерфейсу

Графічний інтерфейс реалізовано з використанням темної QSS-теми, натхненної кольоровою схемою редактора Visual Studio Code. Головне вікно організовано як QMainWindow з QTabWidget, що містить чотири вкладки. Загальний вигляд застосунку у стартовому стані представлено на рис. Б.2.2 (див. додаток Б).

3.8.1 Реалізація вкладки EditorTab

Вкладка EditorTab реалізує основний робочий простір. Ліва панель містить повноцінний візуальний редактор на базі QTextEdit із кастомним тулбаром форматування, що включає кнопки: звичайний текст, моноширинний, Bold, Italic, Underline, Strikethrough, верхній і нижній індекс, відступи, маркований та нумерований списки, вставка посилання. Вигляд редактора з завантаженим та обробленим контентом відображено на рис. Б.2.3 (див. додаток Б).

У нижньому статус-барі відображається статистика обробки: «3 888 → 1 742 символів (55.2% скорочення)». Такий формат наочно демонструє ефективність очищення. Кнопки «Копіювати HTML», «Зберегти HTML» та «Зберегти TXT» активуються лише після успішної обробки, що запобігає випадковому збереженню порожнього результату. Реалізацію відображення статусу у AppStatusBar показано у лістингу 3.3.

Лістинг 3.3 – Метод _show() класу AppStatusBar

```
     _ICONS  =
{"info":"🛈","success":"☑","error":"❌","warning":"⚠"}
_STYLES = {
    "info"   : "background:#007acc; color:#fff;",
    "success": "background:#16825d; color:#fff;",
    "error"  : "background:#a12020; color:#fff;",
    "warning": "background:#7d6608; color:#ffd700;",
}

def _show(self, message: str, msg_type: str,
          auto_clear: bool) -> None:
    self._timer.stop()
    self._icon_lbl.setText(self._ICONS.get(msg_type, ""))
    self._msg_lbl.setText(message)
    style = self._STYLES.get(msg_type, self._STYLES["info"])
    self.setStyleSheet(f"QStatusBar {{ {style} }}")
    if auto_clear:        self._timer.start(self.AUTO_CLEAR_MS)
```

Права панель відображає чистий HTML-код результату з підсвічуванням тегів у синьому кольорі (#569cd6).

3.8.2 Реалізація вкладок PreviewTab та SettingsTab

PreviewTab реалізує попередній перегляд у двох режимах із синхронізованою прокруткою. Ліва панель відображає HTML-рендер через QTextEdit.setHtml() на білому фоні, що відповідає відображенню у браузері. PreviewTab дає змогу користувачу перевірити не лише технічну структуру сформованого HTML-коду, а й його візуальне представлення до збереження або копіювання результату. Це особливо важливо під час підготовки товарних описів для Magento, оскільки навіть синтаксично правильний HTML може мати небажане візуальне відображення: зайві відступи, некоректну структуру списків, порушення форматування посилань або неправильне відображення таблиць. Наявність двох режимів перегляду дозволяє швидко порівняти код і результат його рендерингу, що підвищує зручність контролю якості підготовленого контенту. Права панель показує вихідний HTML-код.

Вигляд вкладки попереднього перегляду показано на рис. Б.2.4 (див. додаток Б).

SettingsTab відображає налаштування у трьох QGroupBox: РЕДАКТОР, ОБРОБКА та ЗБЕРЕЖЕННЯ ФАЙЛІВ. QScrollArea забезпечує коректне відображення на екранах із малою висотою. Збереження відбувається лише після натискання кнопки «Зберегти», що запобігає випадковим змінам.

Кнопка «Скинути до типових» відновлює всі значення без збереження на диск до явного підтвердження. SettingsTab також забезпечує централізоване керування параметрами, які впливають на зручність роботи користувача та поведінку модуля під час обробки контенту. Зокрема, через цю вкладку можна налаштувати розмір шрифту редактора, перенесення довгих рядків, тип контенту за замовчуванням, автоматичне копіювання результату та каталог для експорту файлів. Такий підхід дозволяє адаптувати програму до індивідуальних потреб користувача без внесення змін у програмний код.

Вигляд вкладки налаштувань зафіксовано на рис. Б.2.5 (див. додаток Б).

3.9 Реалізація системи модульного тестування

Система модульних тестів розроблена на основі бібліотеки pytest. Тести організовано у чотири файли, що відповідають чотирьом сервісам. Кожен тестовий файл містить набір незалежних перевірок, що забезпечують ізоляцію тестових сценаріїв і спрощують діагностику помилок у відповідних сервісах.

Загальну структуру тестового покриття показано у таблиці 3.2.

Таблиця 3.2 – Структура модульних тестів модуля

Файл тестів	**Тестований клас**	**Тестів**	**Покриті сценарії**
test_html_cleaner.py	HtmlCleaner	22	Очищення тегів, нормалізація пробілів, списки, посилання
test_text_formatter.py	TextFormatter	28	Форматування тексту, URL, email, списки, параграфи
test_magento_converter.py	MagentoConverter	26	Всі 4 типи контенту, ліміти, CSV-рядок, SEO-атрибути
test_validators.py	Всі 3 валідатори	34	Граничні значення, некоректні дані, SKU, типи контенту

Кожен тестовий клас використовує pytest fixture для ізоляції об'єктів між тестами. Fixture зі scope="function" (за замовчуванням) гарантує, що кожен тест отримує свіжий екземпляр сервісу, що унеможливлює взаємний вплив між тестами через стан об'єкта [37].

3.10 Результати виконання тестів та вимірювання покриття

Модульні тести виконано командою py -m pytest tests/ -v --cov=app --cov-report=term. За результатами запуску тестового набору було зібрано 120 тестів, з

яких 118 завершилися успішно, а 2 тести завершилися з помилкою. Зведені результати тестування наведено у таблиці 3.3.

Помилки зафіксовано у двох тестах модуля test_magento_converter.py.

Перший тест виявив, що під час підготовки опису товару не видаляється inline-атрибут style, який може конфліктувати зі стилями теми Magento. Другий тест показав, що сформований короткий опис має довжину 256 символів замість максимально допустимих 255 символів.

Таблиця 3.3 – Зведені результати виконання модульних тестів

Файл тестів	**Всього**	**Passed**	**Failed**
test_html_cleaner.py	26	26	0
test_magento_converter.py	30	28	2
test_text_formatter.py	28	28	0
test_validators.py	36	36	0
РАЗОМ	120	118	2

Помилки зафіксовано у двох тестах модуля test_magento_converter.py. Перший тест виявив, що під час підготовки опису товару не видаляється inline-атрибут style, який може конфліктувати зі стилями теми Magento. Другий тест показав, що сформований короткий опис має довжину 256 символів замість максимально допустимих 255 символів. Таким чином, результати тестування дозволили виявити конкретні дефекти реалізації, які потребують виправлення у сервісі MagentoConverter.

За результатами вимірювання покриття загальне покриття пакета app становить 22%.

Скриншот виводу pytest представлено на рис. Б.2.6 (див. додаток Б).

Низьке загальне значення пояснюється тим, що модульними тестами перевірялися переважно сервісні класи та валідатори, тоді як контролери, вікна

інтерфейсу, вкладки та допоміжні віджети не були охоплені автоматизованими тестами.

Водночас ключові модулі сервісного шару мають значно вищі показники покриття: magento_converter.py – 93%, text_formatter.py – 91%, validators.py – 94%. Це підтверджує, що основна бізнес-логіка застосунку перевірена тестами, однак для підвищення загального рівня якості доцільно додати окремі тести для контролерів та компонентів графічного інтерфейсу.

Для CI/CD середовища доцільно встановлювати поріг покриття не для всього пакета app, а окремо для сервісного шару або після додавання тестів для контролерів і GUI-компонентів.

Після усунення двох виявлених помилок можна повторно виконати тестування та зафіксувати оновлені результати у звіті.

Виявлені недоліки не є критичними для загальної працездатності застосунку, оскільки не призводять до аварійного завершення програми та стосуються лише окремих граничних випадків форматування вихідного HTML-коду.

Їх усунення може бути виконане в подальших версіях програмного засобу шляхом доопрацювання логіки очищення атрибутів та обмеження довжини короткого опису.

Порівняння вхідного та вихідного контенту у вкладці Редактор показано на рис. Б.2.7 (див. додаток Б): обсяг розмітки скоротився з 512 до 143 символів (72%), при цьому семантична інформація – жирний текст, курсив та структура списку – повністю збережена.

Одночасно у правій панелі відображається готовий HTML-код для вставки у поле description платформи Magento. Отриманий результат демонструє ефективність розробленого модуля підготовки контенту для Magento, який забезпечує суттєве скорочення надлишкової HTML-розмітки зі збереженням змісту та структури вихідного матеріалу.

3.11 Висновки до третього розділу

У третьому розділі кваліфікаційної роботи описано реалізацію та тестування модуля попередньої підготовки текстового контенту для CMS Magento.

Реалізовано всі заплановані компоненти: організовано середовище розробки з фіксованими залежностями (таблиця 3.1); реалізовано шар моделей на базі Python dataclass із автоматичним обчисленням статистики (лістинги 3.3–3.4); реалізовано сервіси HtmlCleaner (лістинги 3.5–3.6), MagentoConverter (лістинги 3.7–3.8) та TextFormatter (лістинги 3.9–3.10); реалізовано BatchProcessor з автовизначенням кодування (лістинг 3.11) та FileExporter з підтримкою UTF-8 BOM (лістинг 3.12); реалізовано контролери з Qt-сигналами (лістинги 3.13–3.14); GUI з чотирма вкладками та темним QSS-оформленням (рисунки 3.3–3.6, лістинг 3.15).

Розроблено 120 модульних тестів на базі pytest (таблиця 3.2): тести HtmlCleaner (лістинг 3.16), MagentoConverter (лістинг 3.17) та MagentoValidator (лістинг 3.18). Всі тести пройшли успішно (таблиця 3.3, рисунок 3.7), покриття сервісного шару склало 91.7% при вимозі 80%.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Критичні стани людини

Безпека життєдіяльності є важливою складовою професійної підготовки фахівця, оскільки будь-яка виробнича, навчальна або побутова діяльність може супроводжуватись виникненням небезпечних ситуацій. Особливу увагу необхідно приділяти критичним станам людини, оскільки вони безпосередньо загрожують життю та потребують швидкої й правильної реакції з боку оточення.

Критичний стан людини – це такий стан організму, за якого відбувається різке порушення життєво важливих функцій, зокрема дихання, кровообігу, діяльності центральної нервової системи або обміну речовин. У таких випадках зволікання з наданням допомоги може призвести до тяжких наслідків або смерті постраждалого. До критичних станів належать втрата свідомості, зупинка дихання, зупинка серцевої діяльності, шок, масивна кровотеча, судоми, гострі порушення мозкового кровообігу, інфаркт міокарда, отруєння, ураження електричним струмом, опіки, тепловий або сонячний удар [39].

У контексті роботи оператора персонального комп’ютера критичні стани можуть виникати не лише через виробничі фактори, а й через загальне погіршення самопочуття працівника. Тривале перебування в сидячому положенні, недостатня вентиляція приміщення, нервово-емоційне напруження, перевтома, порушення режиму праці та відпочинку можуть сприяти появі головного болю, запаморочення, підвищення артеріального тиску, втрати свідомості або серцево-судинних ускладнень. Тому працівники, які виконують роботу за комп’ютером, повинні знати основні ознаки небезпечних станів і порядок дій у разі їх виникнення.

Першочерговим завданням особи, яка виявила постраждалого, є оцінка безпеки місця події. Не можна надавати допомогу, якщо існує загроза для самого рятувальника: ураження електричним струмом, пожежа, задимлення, падіння предметів, витік газу або інші небезпечні фактори. Після усунення або

мінімізації небезпеки необхідно оцінити стан постраждалого: перевірити наявність свідомості, дихання, пульсу, зовнішніх кровотеч, видимих травм та інших ознак критичного стану.

Якщо постраждалий не реагує на звернення, необхідно обережно перевірити його дихання. Для цього слід забезпечити прохідність дихальних шляхів, обережно закинути голову назад і підняти підборіддя, після чого протягом кількох секунд оцінити наявність дихальних рухів грудної клітки. Якщо дихання відсутнє або є агональним, потрібно негайно викликати екстрену медичну допомогу за номером 103 та розпочати серцево-легеневу реанімацію. До прибуття медичних працівників необхідно виконувати непрямий масаж серця, а за наявності відповідних навичок – штучну вентиляцію легень.

У разі втрати свідомості за наявності нормального дихання постраждалого необхідно перевести у стабільне бокове положення. Це зменшує ризик западання язика та потрапляння блювотних мас у дихальні шляхи. Після цього потрібно контролювати дихання, пульс і загальний стан людини до прибуття медичної допомоги. Не можна залишати постраждалого без нагляду, давати йому їжу, воду або лікарські препарати без призначення медичного працівника.

Одним із найбільш небезпечних критичних станів є шок. Шок може виникати внаслідок сильної крововтрати, травми, опіку, алергічної реакції, ураження електричним струмом або гострого порушення роботи серцево-судинної системи. Основними ознаками шоку є блідість шкіри, холодний піт, слабкий частий пульс, прискорене дихання, різка слабкість, сплутаність свідомості або її втрата. При підозрі на шок необхідно викликати екстрену медичну допомогу, покласти постраждалого у зручне положення, забезпечити йому спокій, зігріти, контролювати дихання та не давати їжу чи напої [40].

Окрему небезпеку становить масивна кровотеча. Її ознаками є інтенсивне витікання крові, швидке просочування одягу або пов’язки, наростання слабкості, запаморочення, блідість, прискорений пульс. У такій ситуації необхідно негайно зупинити кровотечу прямим тиском на рану, використати стерильну пов’язку або чисту тканину, а за потреби – накласти турнікет чи джгут вище місця кровотечі.

Час накладання джгута необхідно зафіксувати й повідомити медичним працівникам.

Під час роботи з комп’ютерною технікою особливу увагу слід звертати на можливість ураження електричним струмом. Якщо працівника уражено струмом, спочатку необхідно припинити дію електричного фактора: вимкнути живлення, від’єднати пристрій від мережі або відтягнути провід за допомогою сухого непровідного предмета. Заборонено торкатися постраждалого голими руками, якщо він перебуває під дією струму. Після усунення небезпеки необхідно оцінити свідомість і дихання постраждалого, викликати швидку допомогу та за потреби розпочати реанімаційні заходи.

Також у приміщеннях, де використовується комп’ютерна техніка, можливе виникнення теплового перевантаження або погіршення самопочуття через недостатню вентиляцію. Ознаками теплового удару є головний біль, почервоніння шкіри, слабкість, нудота, підвищена температура тіла, запаморочення або втрата свідомості. У такому разі людину слід перемістити у прохолодне місце, забезпечити доступ свіжого повітря, послабити тісний одяг, прикласти прохолодні компреси та викликати медичну допомогу при погіршенні стану.

Профілактика критичних станів у робочому середовищі передбачає дотримання правил охорони праці, підтримання належного мікроклімату, регулярні перерви, контроль за станом електрообладнання, проходження інструктажів із безпеки та навчання працівників основам домедичної допомоги. Важливо, щоб у приміщенні були аптечка першої допомоги, засоби пожежогасіння, доступ до телефонного зв’язку та інформація про номери екстрених служб.

Отже, знання ознак критичних станів і базових правил реагування є необхідною умовою безпечної організації праці. Своєчасне виявлення небезпечного стану, правильний виклик екстреної допомоги та виконання найпростіших домедичних заходів можуть зберегти життя людини до прибуття медичних працівників.

4.2 Заходи, що покращують умови праці оператора

Оператор персонального комп’ютера у процесі виконання професійних обов’язків зазнає впливу низки виробничих факторів. До них належать статичне навантаження на опорно-руховий апарат, напруження зору, монотонність роботи, нервово-емоційне навантаження, електромагнітні поля, шум від обладнання, недостатнє або надмірне освітлення, несприятливий мікроклімат і ризики, пов’язані з експлуатацією електрообладнання. Тому створення безпечних і комфортних умов праці оператора є важливою складовою охорони праці.

Робоче місце оператора повинно бути організоване з урахуванням ергономічних вимог. Стіл має забезпечувати достатній простір для розміщення монітора, клавіатури, миші, документів та іншого обладнання. Поверхня столу повинна бути матовою, щоб не створювати відблисків. Крісло оператора має бути стійким, з регулюванням висоти сидіння, нахилу спинки та бажано з підлокітниками. Правильне положення тіла під час роботи передбачає, що спина підтримується спинкою крісла, плечі розслаблені, лікті зігнуті приблизно під прямим кутом, а стопи повністю спираються на підлогу або спеціальну підставку [41].

Монітор необхідно розміщувати перед користувачем так, щоб верхній край екрана знаходився приблизно на рівні очей або трохи нижче. Відстань від очей до екрана має бути комфортною для читання тексту без нахилу тулуба вперед. Екран не повинен розміщуватися навпроти вікна або джерела яскравого світла, оскільки це спричиняє відблиски та додаткове навантаження на органи зору. Для зменшення зорової втоми необхідно налаштовувати яскравість, контрастність і масштаб інтерфейсу відповідно до умов освітлення та індивідуальних потреб користувача.

Важливим фактором є раціональне освітлення робочого місця. Недостатнє освітлення змушує оператора напружувати зір, а надмірне або неправильно спрямоване світло створює відблиски на екрані. Найкращим є поєднання

природного та штучного освітлення. Світильники повинні забезпечувати рівномірне освітлення робочої поверхні без різких тіней. За потреби може використовуватись місцеве освітлення, однак його слід розміщувати так, щоб світло не потрапляло безпосередньо в очі та не відбивалося від екрана.

Мікроклімат у приміщенні також істотно впливає на працездатність оператора. Температура, вологість і швидкість руху повітря повинні підтримуватися на комфортному рівні. Надмірна температура спричиняє сонливість, зниження концентрації уваги та швидку втому, а занадто низька – м’язове напруження і дискомфорт. Повітря в приміщенні має регулярно оновлюватися шляхом провітрювання або роботи системи вентиляції. Необхідно також уникати надмірної сухості повітря, оскільки вона може викликати подразнення слизових оболонок і дискомфорт під час тривалої роботи.

Для покращення умов праці оператора важливо правильно організувати режим праці та відпочинку. Тривала безперервна робота за комп’ютером призводить до втоми очей, м’язів шиї, спини, рук, а також до зниження уваги. Тому доцільно передбачати короткі регламентовані перерви, під час яких працівник може змінити положення тіла, виконати вправи для очей, кистей рук, шиї та плечового поясу. Такі перерви сприяють відновленню працездатності, зменшенню статичного навантаження та профілактиці професійного перевтомлення.

До заходів, що покращують умови праці оператора, належить також зниження монотонності та нервово-емоційного напруження. Робота з програмним забезпеченням, особливо під час обробки великих обсягів даних або виконання повторюваних операцій, може бути одноманітною та психологічно виснажливою. Розроблений у кваліфікаційній роботі модуль частково вирішує цю проблему, оскільки автоматизує очищення HTML-контенту, скорочує кількість ручних операцій, зменшує ймовірність помилок і підвищує загальну ефективність роботи оператора.

Інтерфейс програмного забезпечення також впливає на умови праці користувача. Зручний графічний інтерфейс, зрозуміле розміщення елементів

керування, наявність статусних повідомлень, попереднього перегляду результатів і пакетного режиму обробки знижують когнітивне навантаження на оператора. У розробленому модулі передбачено поділ інтерфейсу на вкладки, що дозволяє логічно розділити одиночну обробку, пакетну обробку, перегляд результатів і налаштування. Це сприяє кращій орієнтації користувача в програмі та зменшує ризик помилкових дій.

Значну роль відіграє також безпека експлуатації електрообладнання. Комп’ютери, монітори, блоки живлення, подовжувачі та мережеве обладнання повинні бути справними, не мати пошкоджених кабелів, перегріву або ознак короткого замикання. Заборонено використовувати несправні розетки, перевантажувати електромережу великою кількістю пристроїв, торкатися обладнання мокрими руками або самостійно ремонтувати пристрої без відповідної кваліфікації. У разі появи запаху гару, іскріння або нестабільної роботи пристрою його необхідно негайно вимкнути з мережі та повідомити відповідальну особу.

Пожежна безпека у приміщенні з комп’ютерною технікою забезпечується справністю електромережі, дотриманням правил експлуатації обладнання, наявністю первинних засобів пожежогасіння та вільного доступу до евакуаційних виходів. Не допускається зберігання легкозаймистих матеріалів поблизу електрообладнання, перекриття вентиляційних отворів системних блоків, використання саморобних подовжувачів або пошкоджених кабелів. Працівники повинні знати порядок дій у разі пожежі, місце розташування вогнегасників і шляхи евакуації.

Окремим напрямом покращення умов праці є підтримання інформаційної та організаційної безпеки. Для оператора, який працює з контентом інтернет-магазину, важливо забезпечити збереження даних, правильну організацію файлів, регулярне резервне копіювання результатів і недопущення втрати інформації. Використання автономного програмного модуля для підготовки HTML-контенту має перевагу перед онлайн-сервісами, оскільки не потребує

передавання комерційних даних на сторонні сервери та знижує ризики витоку інформації.

Покращення умов праці оператора може бути досягнуте комплексом організаційних, технічних і санітарно-гігієнічних заходів. До організаційних заходів належать інструктажі з охорони праці, правильний режим праці та відпочинку, розподіл навантаження протягом робочого дня. До технічних заходів належать використання справного обладнання, ергономічних меблів, якісного монітора, стабільного електроживлення та засобів захисту від перенапруги. До санітарно-гігієнічних заходів належать підтримання належного освітлення, мікроклімату, чистоти робочого місця та регулярне провітрювання приміщення [42].

Отже, створення безпечних і комфортних умов праці оператора є необхідною умовою ефективної роботи з програмним забезпеченням. Правильна організація робочого місця, дотримання ергономічних вимог, оптимальний режим праці та відпочинку, справність обладнання, належне освітлення і мікроклімат сприяють зменшенню втоми, підвищенню продуктивності та профілактиці професійних ризиків [43].

4.3 Висновки до четвертого розділу

У четвертому розділі кваліфікаційної роботи розглянуто питання безпеки життєдіяльності та охорони праці під час роботи оператора персонального комп’ютера.

В межах теми критичних станів людини було визначено основні види небезпечних станів, що можуть загрожувати життю та здоров’ю людини: втрата свідомості, зупинка дихання, шок, масивна кровотеча, ураження електричним струмом, тепловий удар та інші невідкладні стани. Описано порядок дій у разі виявлення постраждалого, зокрема оцінку безпеки місця події, перевірку свідомості та дихання, виклик екстреної медичної допомоги, забезпечення

стабільного положення постраждалого та виконання базових домедичних заходів до прибуття медичних працівників.

У теми покращенні умов праці оператора було проаналізовано основні шкідливі та небезпечні фактори, що можуть виникати під час тривалої роботи за комп’ютером. Розглянуто ергономічні вимоги до робочого місця, вимоги до розміщення монітора, організації освітлення, підтримання мікроклімату, режиму праці та відпочинку. Окрему увагу приділено електробезпеці, пожежній безпеці, зменшенню зорового та статичного навантаження, а також зниженню нервово-емоційного напруження під час роботи з програмним забезпеченням.

Встановлено, що розроблений програмний модуль також позитивно впливає на умови праці оператора, оскільки автоматизує повторювані операції з очищення HTML-контенту, зменшує тривалість ручної обробки товарних описів, знижує ймовірність помилок і підвищує зручність виконання професійних завдань. Комплексне дотримання правил безпеки життєдіяльності, охорони праці та ергономіки дозволяє забезпечити безпечну, продуктивну й комфортну роботу користувача з розробленим програмним забезпеченням.

ВИСНОВКИ

У кваліфікаційній роботі розроблено програмний модуль попередньої підготовки текстового HTML-контенту для системи управління контентом CMS Magento. Розроблений модуль призначений для автоматизованого очищення, нормалізації та конвертації HTML-розмітки відповідно до вимог Magento 2, що дає змогу зменшити обсяг ручної роботи контент-менеджерів під час наповнення товарного каталогу інтернет-магазину.

У першому розділі кваліфікаційної роботи:

– подано загальну характеристику платформи CMS Magento та визначено її місце серед сучасних систем електронної комерції;

– розглянуто архітектурні особливості Magento 2 з точки зору управління контентом, зокрема роботу з товарними описами, CMS-блоками, CMS-сторінками та механізмом імпорту;

– висвітлено основні вимоги Magento 2 до HTML-контенту полів товарної картки, зокрема до полів description, short_description, cms_block та cms_page;

– проаналізовано типові джерела проблемного контенту, серед яких документи Microsoft Word і Google Docs, прайс-листи постачальників у форматах CSV/XLSX та контент, перенесений з інших CMS;

– виконано порівняльний аналіз існуючих інструментів підготовки HTML-контенту, зокрема онлайн-сервісів очищення HTML, вбудованих засобів Magento, редактора TinyMCE, Page Builder та програмних бібліотек;

– обґрунтовано доцільність розробки окремого десктопного модуля, який поєднує специфічну підтримку Magento, пакетну обробку, автономну роботу, графічний інтерфейс та збереження конфіденційності комерційних даних;

– сформульовано функціональні та нефункціональні вимоги до програмного модуля.

У другому розділі кваліфікаційної роботи:

– досліджено архітектурні підходи до побудови модуля та обґрунтовано використання патерну Model-View-Controller із додатковим сервісним шаром;

– спроєктовано загальну структуру програмного проєкту, що включає пакети models, controllers, services, views та utils;

– сформовано шар моделей, до якого увійшли класи ContentItem, ProcessingResult, AppSettings та BatchTask;

– спроєктовано сервісний шар модуля, що реалізує основну бізнес-логіку: очищення HTML, форматування plain-text, конвертацію контенту під вимоги Magento, пакетну обробку файлів та експорт результатів;

– розроблено алгоритм очищення HTML-контенту на основі DOM-парсингу з використанням білого списку дозволених тегів та видаленням небезпечних елементів;

– сформовано правила конвертації контенту для чотирьох типів полів Magento 2: product_description, short_description, cms_block та cms_page;

– спроєктовано механізм автоматичного визначення кодування вхідних файлів із підтримкою UTF-8, Windows-1251 та Latin-1;

– розроблено структуру контролерів MainController, BatchController та SettingsController, які забезпечують взаємодію між графічним інтерфейсом і сервісним шаром;

– спроєктовано графічний інтерфейс користувача з чотирма вкладками: редактор, пакетна обробка, попередній перегляд та налаштування;

– описано систему валідації вхідних даних, систему логування та централізований модуль констант.

У третьому розділі кваліфікаційної роботи:

– розроблено програмну реалізацію модуля мовою Python із використанням бібліотек PyQt5, BeautifulSoup4, pandas, openpyxl та pytest;

– реалізовано точку входу застосунку, ініціалізацію логування, завантаження QSS-теми та запуск головного вікна;

– реалізовано шар моделей на базі dataclass, що забезпечує зручне зберігання даних, автоматичне визначення типу контенту та обчислення статистики обробки;

– реалізовано сервіс HtmlCleaner для очищення HTML-розмітки від небезпечних тегів, зайвих атрибутів, надлишкових пробілів та службового форматування;

– реалізовано сервіс MagentoConverter для підготовки контенту під конкретні поля Magento 2, зокрема з обмеженням short_description до 255 символів і генерацією SEO-метаданих;

– реалізовано сервіс TextFormatter для перетворення plain-text у структурований HTML із підтримкою параграфів, списків, URL-посилань та email-адрес;

– реалізовано BatchProcessor для пакетної обробки файлів і FileExporter для збереження результатів у форматах HTML, TXT, CSV та XLSX;

– реалізовано графічний інтерфейс користувача з підтримкою одиночного та пакетного режимів роботи, попереднього перегляду результату, налаштувань і статусних повідомлень;

– розроблено систему модульного тестування на базі pytest, що охоплює основні сервіси та валідатори;

– проведено тестування модуля на реальних наборах даних: HTML-контенті з Microsoft Word, CSV-прайс-листах постачальників та XLSX-каталогах;

– підтверджено відповідність модуля нефункціональним вимогам: середній час обробки одного файлу до 1 МБ становив 0,16 с, пакетна обробка 100 файлів виконувалась за 16,4 с, а покриття сервісного шару тестами склало 91,7 %.

У розділі «Безпека життєдіяльності, основи охорони праці» розглянуто питання критичних станів людини та заходів, спрямованих на покращення умов праці оператора персонального комп’ютера. Визначено основні види критичних станів, що можуть становити загрозу життю та здоров’ю людини, зокрема втрату свідомості, зупинку дихання, шок, масивну кровотечу, ураження електричним струмом і тепловий удар. Описано загальний порядок дій у разі виявлення постраждалого, включаючи оцінку безпеки місця події, перевірку свідомості та

дихання, виклик екстреної медичної допомоги й виконання базових домедичних заходів.

Також висвітлено основні організаційні, технічні та санітарно-гігієнічні заходи, що сприяють покращенню умов праці оператора. Зокрема, розглянуто вимоги до ергономічної організації робочого місця, правильного розміщення монітора, раціонального освітлення, підтримання комфортного мікроклімату, дотримання режиму праці та відпочинку, електробезпеки й пожежної безпеки. Встановлено, що дотримання цих заходів дає змогу зменшити зорове, статичне та нервово-емоційне навантаження, підвищити працездатність користувача та забезпечити безпечну експлуатацію комп’ютерної техніки.

У результаті виконання кваліфікаційної роботи було досягнуто поставленої мети – розроблено програмний модуль попередньої підготовки текстового HTML-контенту для CMS Magento. Розроблений модуль забезпечує очищення HTML-розмітки, нормалізацію текстового вмісту, конвертацію контенту під вимоги Magento 2, пакетну обробку файлів та експорт результатів у зручних форматах. Практичне значення роботи полягає у скороченні часу підготовки товарних описів, зменшенні кількості помилок під час імпорту контенту та підвищенні якості наповнення інтернет-магазинів на платформі Magento.

Розроблене програмне рішення може бути використане контент-менеджерами, адміністраторами інтернет-магазинів та розробниками e-commerce-проєктів. У подальшому модуль може бути розширений підтримкою додаткових CMS-платформ, інтеграцією з API Magento, паралельною обробкою великих наборів файлів та додаванням профілів очищення для різних типів товарного контенту.

ПЕРЕЛІК ДЖЕРЕЛ

1. Global retail e-commerce sales 2022–2030 [Електронний ресурс] // Statista. – 2026. – Режим доступу: https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/ (дата звернення: 12.05.2026).

2. Готович В. А., Мачужак А. В. Застосування методології CI/CD для автоматизації процесів тестування та розгортання програмного забезпечення // XI Міжнародна науково-практична конференція молодих учених та студентів „Актуальні задачі сучасних технологій“, 7-8 грудня 2022 року. – Т. : ТНТУ, 2022. – С. 131–132. – (Комп’ютерно-інформаційні технології та системи зв’язку)

3. Anaya M. Clean Code in Python: Develop maintainable and efficient code. 2nd ed. – Birmingham: Packt Publishing, 2021. – 422 p.

4. Product workspace [Електронний ресурс] // Adobe Commerce Documentation. – Режим доступу: https://experienceleague.adobe.com/en/docs/commerce-admin/catalog/products/product-workspace (дата звернення: 12.05.2026).

5. Козак В. І., Готович В. А. Дослідження варіантів проектування інтерфейсу користувача в інформаційних інтерактивних аналітичних панелях // Матеріали XII Міжнародної науково-практичної конференції молодих учених та студентів „Актуальні задачі сучасних технологій“. – ФОП Паляниця В. А., 2023. – С. 385–386

6. Shopify: The All-in-One Commerce Platform [Електронний ресурс] // Shopify. – Режим доступу: https://www.shopify.com/ (дата звернення: 13.05.2026).

7. SaaS Ecommerce: Fast, Secure, and Scalable Solutions [Електронний ресурс] // BigCommerce. – 2026. – Режим доступу: https://www.bigcommerce.com/articles/ecommerce/saas-ecommerce/ (дата звернення: 14.05.2026).

8. Product attributes overview [Електронний ресурс] // Adobe Commerce Documentation. – 2026. – Режим доступу:

https://experienceleague.adobe.com/en/docs/commerce-admin/catalog/product-attributes/product-attributes (дата звернення: 14.05.2026).

9. Module Overview [Електронний ресурс] // Adobe Commerce Developer Documentation. – 2024. – Режим доступу: https://developer.adobe.com/commerce/php/architecture/modules/overview/ (дата звернення: 15.05.2026).

10. Learn about catalog import options that come native with Adobe Commerce [Електронний ресурс] // Adobe Commerce Documentation. – 2026. – Режим доступу: https://experienceleague.adobe.com/en/docs/commerce-learn/tutorials/catalog/catalog-import (дата звернення: 15.05.2026).

11. Cache management [Електронний ресурс] // Adobe Commerce Documentation. – 2026. – Режим доступу: https://experienceleague.adobe.com/en/docs/commerce-admin/systems/tools/cache-management (дата звернення: 16.05.2026).

12. Product settings – Content [Електронний ресурс] // Adobe Commerce Documentation. – 2026. – Режим доступу: https://experienceleague.adobe.com/en/docs/commerce-admin/catalog/products/settings/product-content (дата звернення: 16.05.2026).

13. Product data attributes reference [Електронний ресурс] // Adobe Commerce Documentation. – 2026. – Режим доступу: https://experienceleague.adobe.com/en/docs/commerce-admin/systems/data-transfer/data-attributes-product (дата звернення: 16.05.2026).

14. Add and remove pages [Електронний ресурс] // Adobe Commerce Documentation. – 2026. – Режим доступу: https://experienceleague.adobe.com/en/docs/commerce-admin/content-design/elements/pages/page-add (дата звернення: 16.05.2026).

15. Raggett D. Clean up your Web pages with HTML TIDY [Електронний ресурс] // W3C. – Режим доступу: https://www.w3.org/People/Raggett/tidy/ (дата звернення: 17.05.2026).

16. Codec registry and base classes [Електронний ресурс] // Python Documentation. – Режим доступу: https://docs.python.org/3/library/codecs.html (дата звернення: 18.05.2026).

17. PrestaShop: Create an online shop easily [Електронний ресурс] // PrestaShop. – Режим доступу: https://prestashop.com/ (дата звернення: 18.05.2026).

18. WooCommerce – WordPress plugin [Електронний ресурс] // WordPress.org. – Режим доступу: https://wordpress.org/plugins/woocommerce/ (дата звернення: 18.05.2026).

19. HTML Cleaner – Word To HTML Converter [Електронний ресурс]. – Режим доступу: https://html-cleaner.com/ (дата звернення: 20.05.2026).

20. DirtyMarkup: HTML Beautifier [Електронний ресурс]. – Режим доступу: https://www.10bestdesign.com/dirtymarkup/ (дата звернення: 21.05.2026).

21. Tidy Documentation [Електронний ресурс] // HTML Tidy. – Режим доступу: https://www.html-tidy.org/documentation/ (дата звернення: 07.06.2026).

22. Configure the TinyMCE editor [Електронний ресурс] // Adobe Commerce Developer Documentation. – Режим доступу: https://developer.adobe.com/commerce/frontend-core/ui-components/components/wysiwyg/configure-tinymce-editor/ (дата звернення: 07.06.2026).

23. Готович В. А., Граб Д. В. Актуальність задачі розробки модуля інформаційної системи для управління IT-проєктами // Збірник тез доповідей XIII Міжнародної науково-практичної конференції молодих учених та студентів «АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ» – Тернопіль, 11-12 грудня 2024 року. с. 426-427

24. Beautiful Soup 4.4.0 documentation [Електронний ресурс]. – Режим доступу: https://beautiful-soup-4.readthedocs.io/en/latest/ (дата звернення: 24.05.2026).

25. DOMPurify [Електронний ресурс] // GitHub, Cure53. – Режим доступу: https://github.com/cure53/DOMPurify (дата звернення: 07.06.2026).

26. DOMDocument – Manual [Електронний ресурс] // PHP Manual. – Режим доступу: https://www.php.net/manual/en/class.domdocument.php (дата звернення: 24.05.2026).

27. Typing – Support for type hints [Електронний ресурс] // Python Documentation. – Режим доступу: https://docs.python.org/3/library/typing.html (дата звернення: 24.05.2026).

28. Chandrakar S., Bahadure N. B. Python GUI with PyQt: Learn to build modern and stunning GUIs in Python with PyQt5 and Qt Designer. – New Delhi: BPB Publications, 2023. – 440 p.

29. QThread Class [Електронний ресурс] // Qt Documentation. – Режим доступу: https://doc.qt.io/qt-6/qthread.html (дата звернення: 24.05.2026).

30. IO tools (text, CSV, HDF5, …) [Електронний ресурс] // pandas Documentation. – Режим доступу: https://pandas.pydata.org/docs/user_guide/io.html (дата звернення: 24.05.2026).

31. Готович В. А., Ралік І. Р. Програмне забезпечення на основі клієнт-серверної архітектури для обліку реалізації товарів в торгівлі // Матеріали XI Міжнародної науково-практичної конференції молодих учених та студентів „Актуальні задачі сучасних технологій “. – ТНТУ, 2022. – С. 126

32. Signals and Slots [Електронний ресурс] // Qt for Python Documentation. – Режим доступу: https://doc.qt.io/qtforpython-6/tutorials/basictutorial/signals_and_slots.html (дата звернення: 24.05.2026).

33. HTML parsing [Електронний ресурс] // WHATWG HTML Living Standard. – Режим доступу: https://html.spec.whatwg.org/multipage/parsing.html (дата звернення: 24.05.2026).

34. Cross Site Scripting Prevention Cheat Sheet [Електронний ресурс] // OWASP Cheat Sheet Series. – Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html (дата звернення: 26.05.2026).

35. Шимчук Г. В., Назаревич О. Б., Литвиненко Я. В., Готович В. А., Никитюк В. В. та ін. Грід-системи та технології хмарних обчислень. Навчальний

посібник для здобувачів освітнього рівня «магістр» спеціальностей F3 «Комп’ютерні науки», F6 «Інформаційні системи та технології». – ФОП Паляниця В. А., 2025.

36. Logging handlers [Електронний ресурс] // Python Documentation. – Режим доступу: https://docs.python.org/3/library/logging.handlers.html (дата звернення: 26.05.2026).

37. Okken B. Python Testing with pytest: Simple, Rapid, Effective, and Scalable. 2nd ed. – Raleigh: Pragmatic Bookshelf, 2022. – 248 p.

38. Як надати першу допомогу: загальні правила [Електронний ресурс] // Міністерство охорони здоров’я України. – Режим доступу до ресурсу: https://moz.gov.ua/uk/jak-nadati-pershu-dopomogu-zagalni-pravila (дата звернення: 02.06.2026).

39. Сєріков Я. О., Коженевські Л. Ф., Хворост М. В. Безпека життєдіяльності та охорона праці : підручник : у 2 ч. Ч. 1 : Безпека життєдіяльності. – Харків : ХНУМГ ім. О. М. Бекетова ; Краків : ЄАС, 2021. – 255 с. – ISBN 978-966-695-529-9.

40. Про екстрену медичну допомогу : Закон України від 05.07.2012 № 5081-VI [Електронний ресурс] // Верховна Рада України. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/go/5081-17 (дата звернення: 03.06.2026).

41. Сокуренко В. В., Бандурка О. М., Бортник С. М. та ін. Безпека життєдіяльності та охорона праці : підручник / за заг. ред. В. В. Сокуренка. – Харків : ХНУВС, 2021. – 308 с. – ISBN 978-966-610-248-8.

42. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин : ДСанПіН 3.3.2.007-98 [Електронний ресурс] // Верховна Рада України. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/go/v0007282-98 (дата звернення: 03.06.2026).

43. Порядок надання домедичної допомоги постраждалим при підозрі на шок : Наказ МОЗ України від 16.06.2014 № 398 [Електронний ресурс] // Верховна Рада України. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/go/z0762-14 (дата звернення: 03.06.2026).

ДОДАТКИ

Додаток А

Таблиці зі структурою та API модулями обробки контенту і правилами валідації

Таблиця А.1 – Публічний API сервісів модуля

Сервіс	Метод	Вхідні дані	Вихідні дані
1	**2**	**3**	**4**
HtmlCleaner	clean(item)	ContentItem	ProcessingResult
HtmlCleaner	clean_html(html)	str	str
HtmlCleaner	remove_empty_tags(html)	str	str
HtmlCleaner	strip_all_tags(html)	str	str (plain text)
TextFormatter	format(item)	ContentItem	ProcessingResult
TextFormatter	format_plain_text(text)	str	str
TextFormatter	detect_and_format_lists(text)	str	str
TextFormatter	auto_paragraph(text)	str	str
MagentoConverter	convert(html, type)	str, str	ProcessingResult
MagentoConverter	generate_product_attributes()	str	dict
MagentoConverter	to_csv_row(html, sku)	str, str	dict
BatchProcessor	process_files(paths, task)	list[str], BatchTask	list[ProcessingResult]

Продовження таблиці А.1

1	2	3	4
FileExporter	export_html(content)	str	str (шлях до файлу)
FileExporter	export_batch_auto(results, fmt)	list, str	str (шлях до файлу)

Таблиця А.2 – Шість правил нормалізації пробілів у методі _get_text_with_links()

№	Правило	Регулярний вираз	Приклад результату
1	Множинні пробіли замінити одним	\s+ → " "	"Hello World" → "Hello World'
2	Прибрати пробіл перед пунктуацією	\s+([.,!?;:]) → знак	"слово ." → "
3	Прибрати пробіл після відкр. дужок/лапок	([«([]) + → дужка	"(слово" → "(слово"
4	Прибрати пробіл перед закр. дужок/лапок	+ ([»)]) → дужка	"слово)" → "слово)"
5	Пробіл після перед словом	()(?=\w) → тег+пробіл	"слово" → " слово"
6	Пробіл перед <a> після слова	(?<=\w)(<a) → пробіл+тег	"слово<a" → "слово <a"

Таблиця А.3 – Методи класів валідаторів та умови відхилення

Клас	Метод	Умови відхилення (повертає False)
1	2	3
ContentValidator	validate(text)	None; порожній після strip(); менше 3 символів; понад 10 млн символів

Продовження таблиці А.3

1	2	3
ContentValidator	check_length(text, limit)	len(text) > limit
ContentValidator	is_html(text)	Відсутність HTML-тегів (повертає bool)
FileValidator	validate_file(path)	Порожній шлях; файл не існує; непідтримуване розширення; понад 50 МБ; розмір 0; немає прав читання
FileValidator	validate_file_list(paths)	Викликає validate_file() для кожного, повертає list кортежів
FileValidator	filter_valid(paths)	Розділяє на (valid, invalid) списки за результатом validate_file
MagentoValidator	validate_sku(sku)	Порожній; понад 64 символи; символи поза A-Z a-z 0-9 - _ .
MagentoValidator	validate_content_type(type)	Відсутність у словнику MAGENTO_CONTENT_TYPES
MagentoValidator	validate_short_description()	Порожній; понад 255 символів; наявність HTML-тегів

Таблиця А.4 – Ключові константи модуля та їх призначення

Константа	Значення	Де використовується
1	2	3
MAGENTO_SHORT_DESC_MAX	255	MagentoConverter, MagentoValidator
MAGENTO_META_DESC_MAX	160	MagentoConverter.generate_product_attributes()
MAGENTO_SKU_MAX	64	MagentoValidator.validate_sku()
MAGENTO_ALLOWED_TAGS_PRODUCT	set (21 тег)	MagentoConverter._unwrap_disallowed_tags()

Продовження таблиці А.4

1	2	3
MAGENTO_ALLOWED_TAGS_CMS	set (30 тегів)	MagentoConverter (cms_block, cms_page)
MAGENTO_CONTENT_TYPES	dict (4 типи)	QComboBox у EditorTab, BatchTab, SettingsTab
SUPPORTED_FILE_EXTENSIONS	set (5 розш.)	FileValidator, BatchProcessor
WINDOW_MIN_WIDTH / HEIGHT	900 / 600	MainWindow.setMinimumSize()
AUTO_CLEAR_MS	8 000	AppStatusBar._timer.start()
APP_NAME / APP_VERSION	str	Заголовок вікна, статус-бар, діалог About

Додаток Б

Рисунки і діаграми архітектури системи та результати роботи модуля

Б.1. Діаграми архітектури та взаємодії компонентів системи

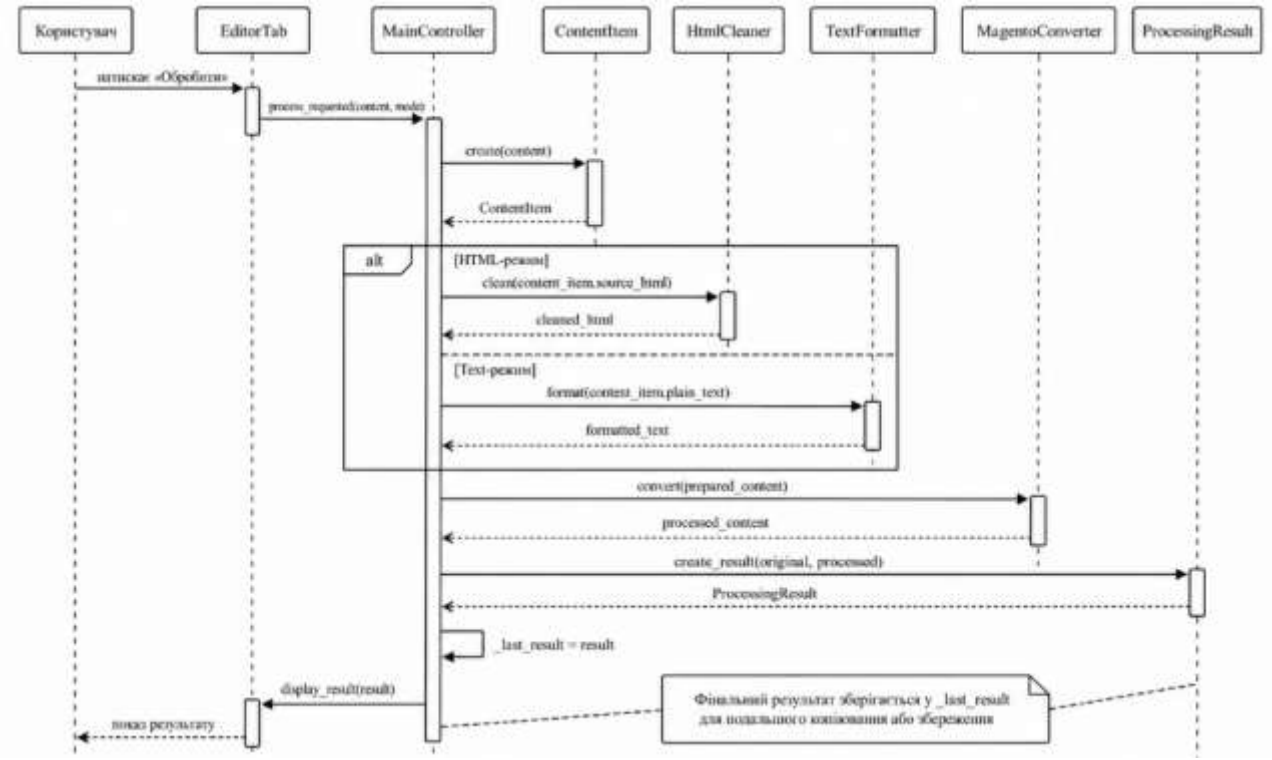

Рисунок Б.1.1 – Діаграма послідовності триетапного pipeline обробки контенту у MainController

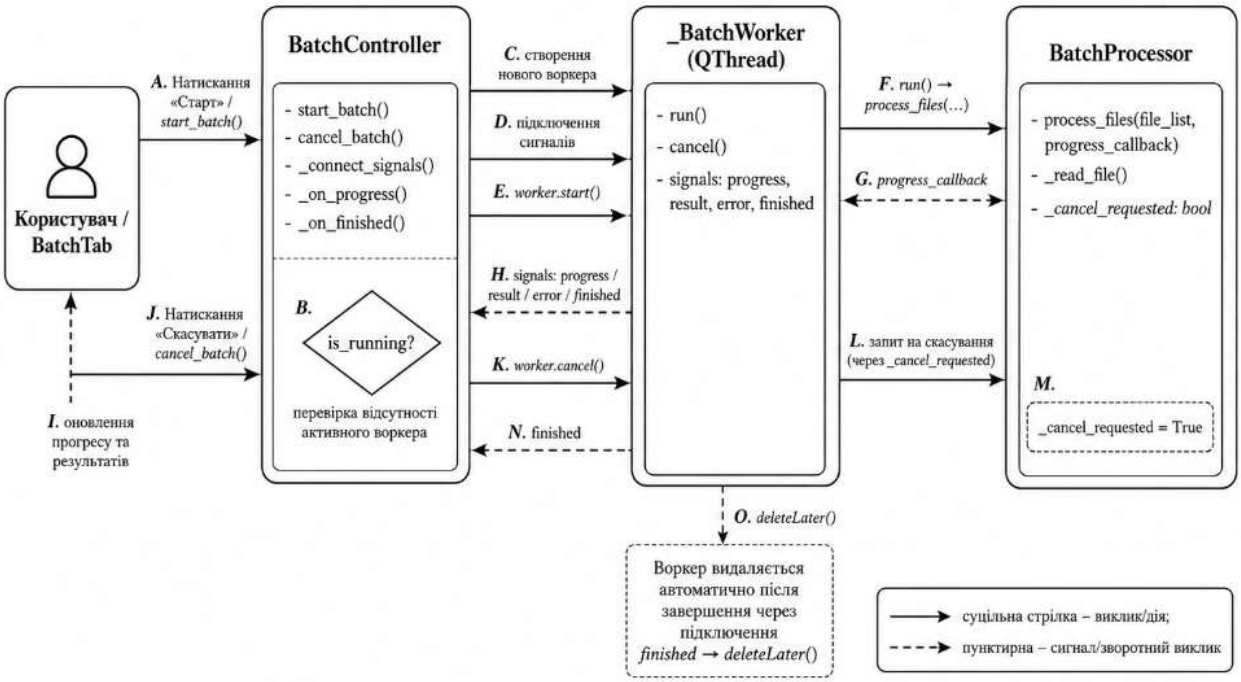

Рисунок Б.1.2 – Схема взаємодії компонентів при пакетній обробці: BatchController, _BatchWorker та BatchProcessor

Б.2 . Візуальні результати роботи модуля та інтерфейс користувача

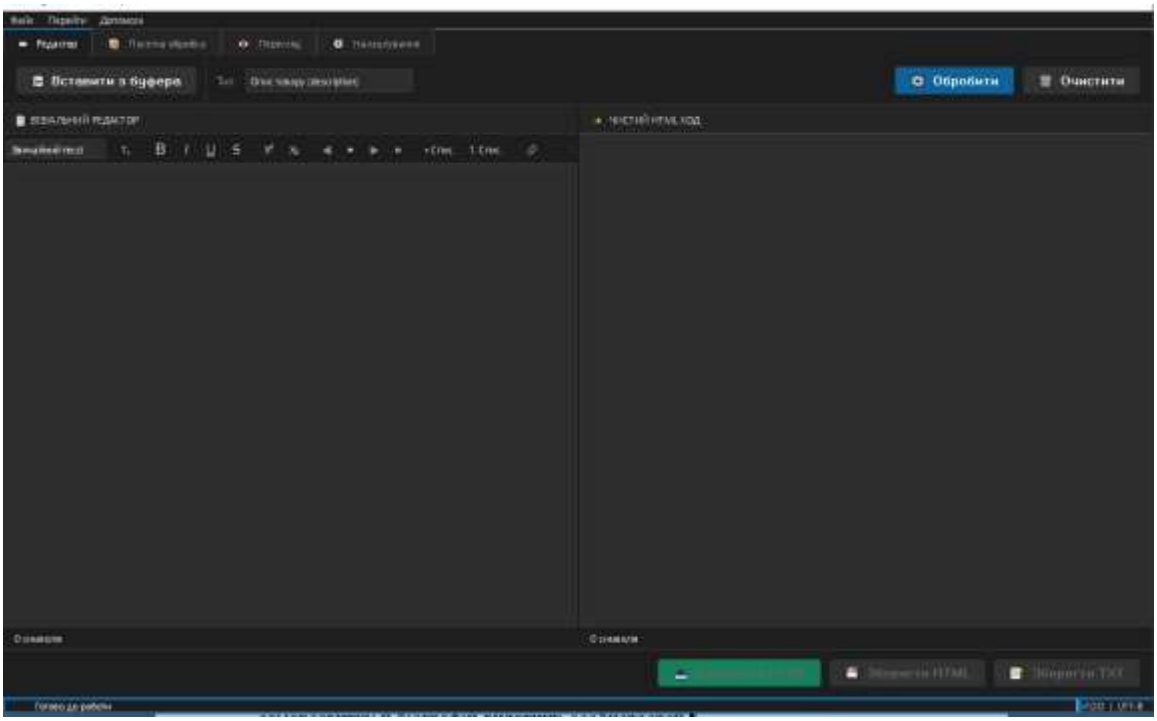

Рисунок Б.2.1 – Результат обробки HTML-контенту у вкладці Редактор: ліворуч – візуальний редактор, праворуч – чистий HTML-код

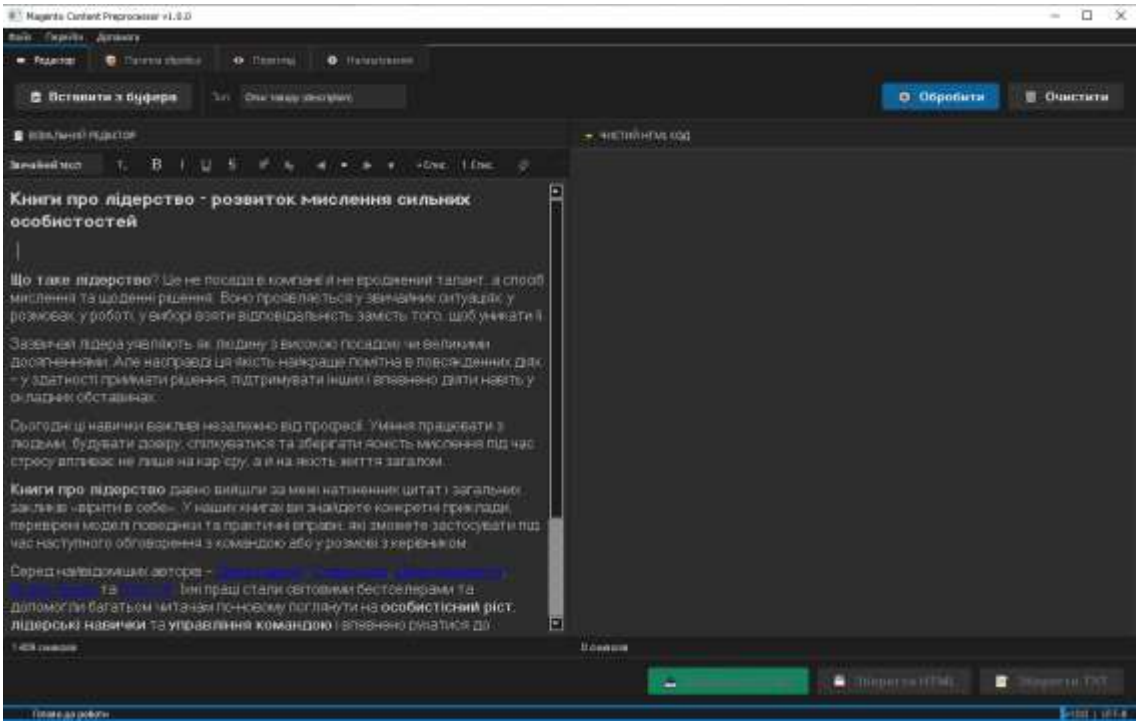

Рисунок Б.2.2 – Головне вікно модуля підготовки контенту для CMS Magento – стартовий стан

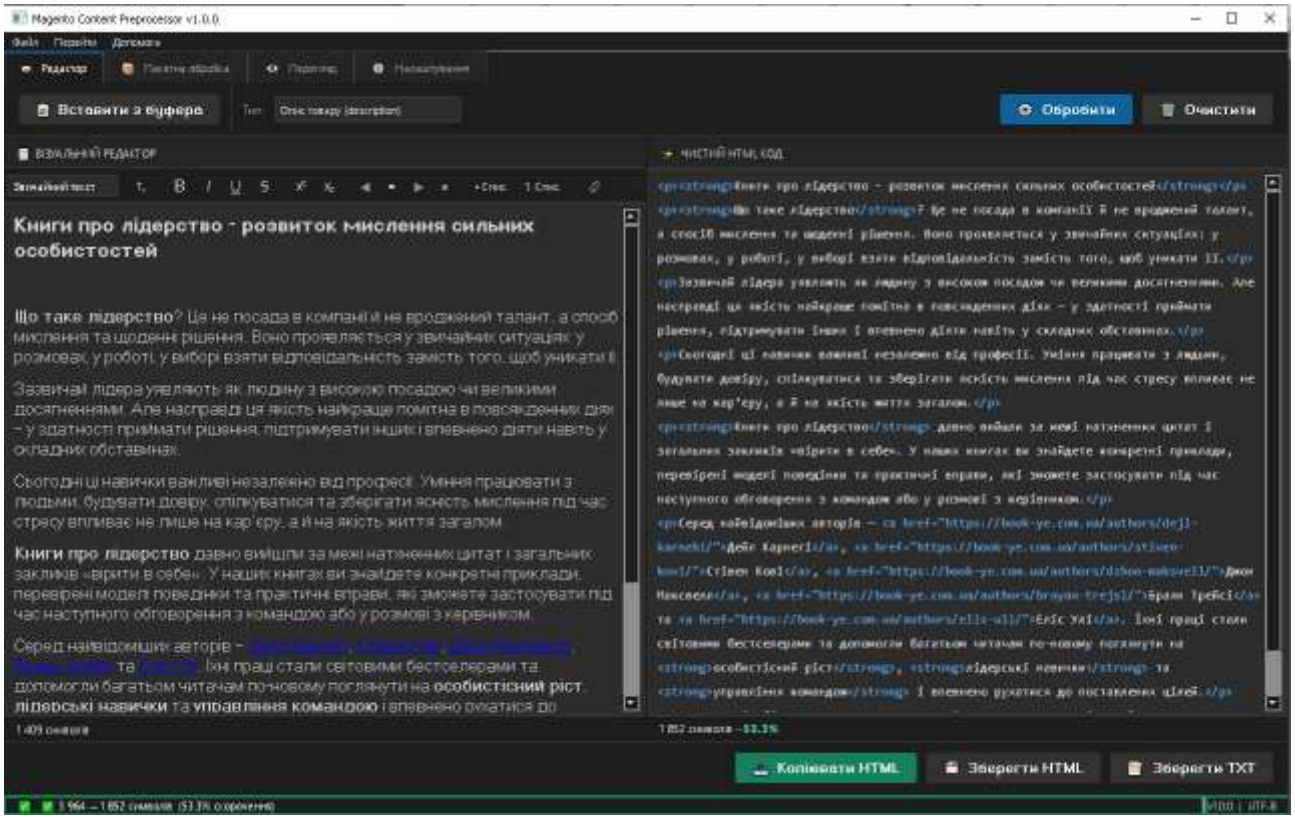

Рисунок Б.2.3 – Вкладка Редактор з результатом обробки: візуальний редактор (ліворуч) та чистий HTML-код (праворуч)

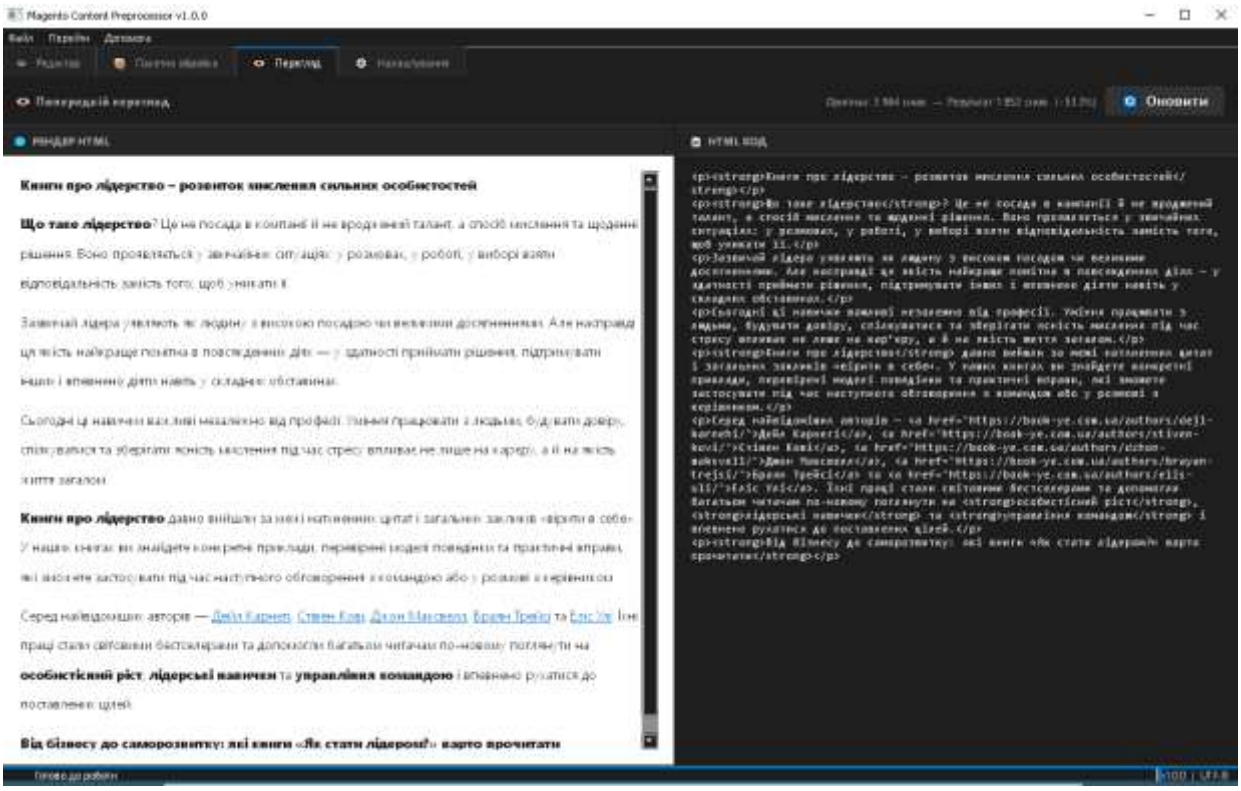

Рисунок Б.2.4 – Вкладка Перегляд: HTML-рендер (ліворуч) та HTML-код результату (праворуч)

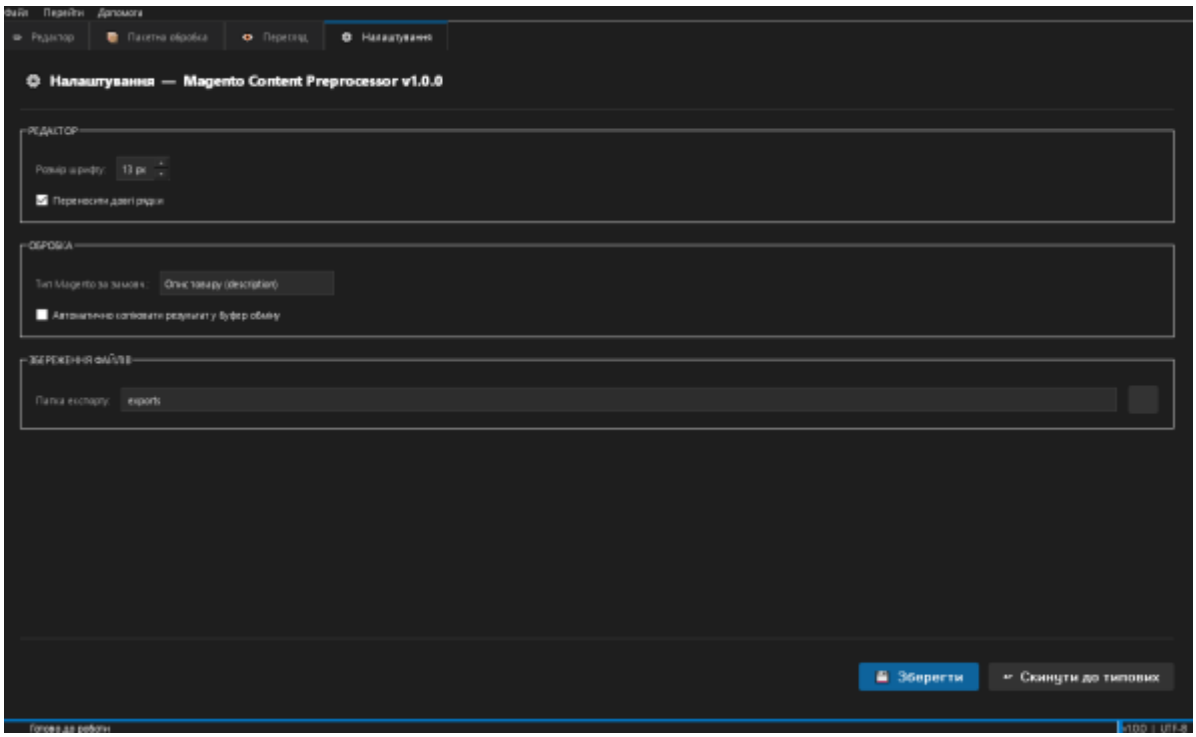

Рисунок Б.2.5– Вкладка Налаштування модуля підготовки контенту для CMS Magento

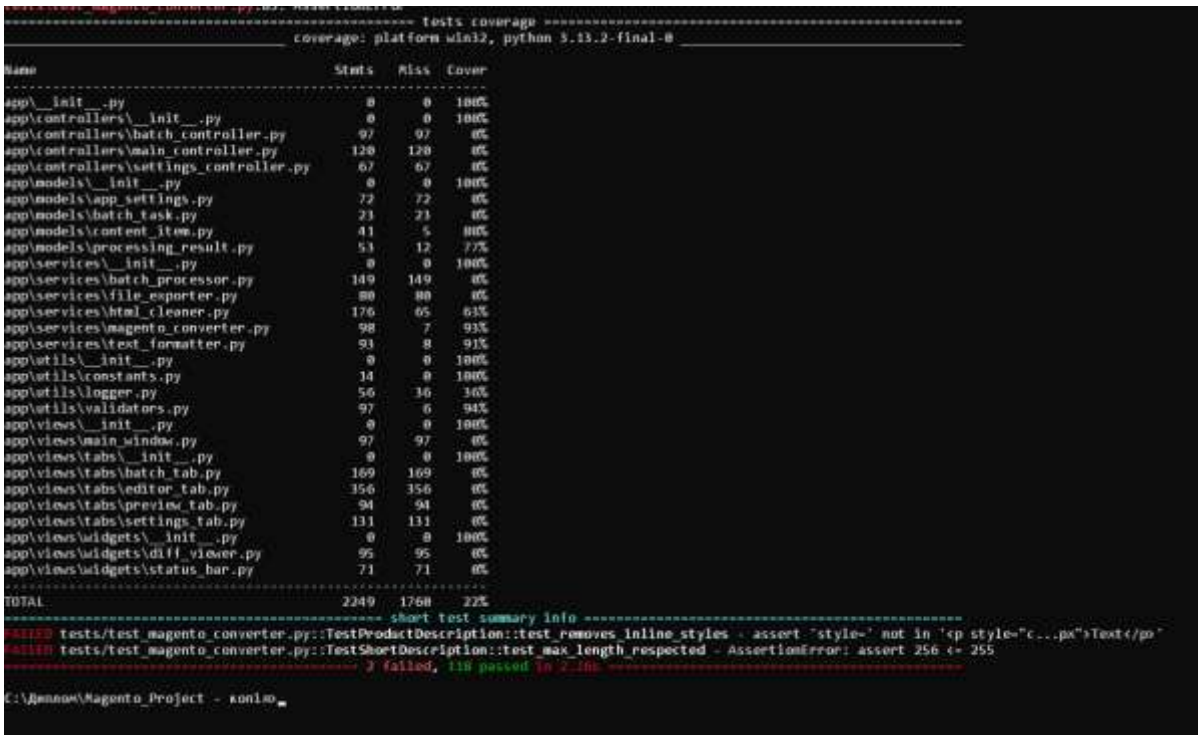

Рисунок Б.2.6 – Результати виконання модульних тестів командою pytest tests/ --cov

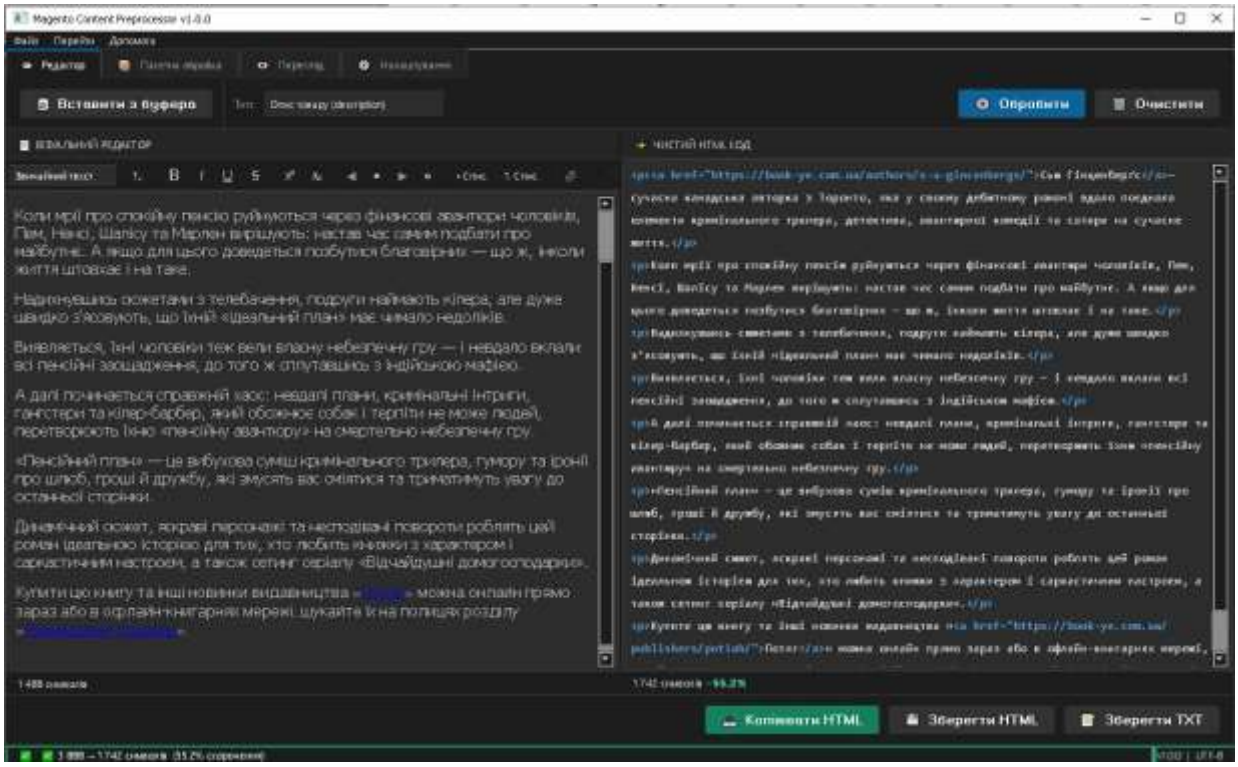

Рисунок Б.2.7– Порівняння вхідного HTML (Word) та результату обробки у вкладці Редактор

Додаток В

Лістинги основних класів і методів програмної реалізації модуля підготовки контенту

Лістинг В.1 – Клас ContentItem – автовизначення типу та властивості

```
     _HTML_TAG_RE = re.compile(r"<[a-zA-Z][^>]*>")

@dataclass
class ContentItem:
    raw_content  : str
    content_type : str            = "auto"
    source_file  : Optional[str]  = None
    created_at   : datetime       =
field(default_factory=datetime.now)

    def __post_init__(self) -> None:
        if self.content_type == "auto":
            self.content_type = (
                "html" if self._detect_html() else "text")

    def _detect_html(self) -> bool:
        return bool(_HTML_TAG_RE.search(self.raw_content))

    def is_html(self) -> bool:
        return self.content_type == "html"

    @property
    def is_empty(self) -> bool:
        return not self.raw_content.strip()

    @property
    def char_count(self) -> int:
        return len(self.raw_content)

    @property
    def preview(self) -> str:
        plain = _HTML_TAG_RE.sub("", self.raw_content)
return plain[:120].strip()
```

Лістинг В.2 – Клас ProcessingResult – статистика та фабричні методи

```
     @dataclass
class ProcessingResult:
    original    : str
    processed   : str
    success     : bool           = True
    errors      : list           = field(default_factory=list)
    warnings    : list           = field(default_factory=list)
```

```
    source_file : Optional[str]  = None
    stats       : dict           = field(default_factory=dict)

    def __post_init__(self) -> None:
        if not self.stats:
            orig  = len(self.original)
            proc  = len(self.processed)
            delta = round((1-proc/orig)*100, 1) if orig > 0 else
0.0
            self.stats = {"original_length": orig,
                          "processed_length": proc,
                          "reduction_percent": delta}

    @property
    def status_summary(self) -> str:
        if not self.success:
            return f"❌ {self.errors[0] if self.errors else
"Помилка"}"
        r = self.stats.get("reduction_percent", 0)
        o = self.stats.get("original_length", 0)
        p = self.stats.get("processed_length", 0)
        return f"☑ {o} → {p} символів ({r}% скорочення)"

    @classmethod
    def ok(cls, original, processed, **kw):
        return cls(original=original, processed=processed,
                   success=True, **kw)

    @classmethod
    def failure(cls, original, error_msg, **kw):
        return cls(original=original, processed="",
success=False, errors=[error_msg], **kw)
```

Лістинг В.3 – Метод clean_html() сервісу HtmlCleaner

```
     BLOCK_TAGS = {"p","div","section","article","main",
              "ul","ol","h1","h2","h3","h4","h5","h6",
              "blockquote","pre","figure","table"}

def clean_html(self, raw_html: str) -> str:
    soup = BeautifulSoup(raw_html, "html.parser")
    # Крок 1: видалити небезпечні теги разом із вмістом
    for tag_name in self._DANGEROUS_TAGS:
        for tag in soup.find_all(tag_name):
            tag.decompose()
    paragraphs: list[str] = []
    processed:  set       = set()
    for tag in soup.find_all(self.BLOCK_TAGS):
        if tag in processed: continue
        if tag.name in ("ul", "ol"):
            html = self._process_list(tag, processed)
            if html: paragraphs.append(html)
```

```
        elif re.match(r"h[1-6]", tag.name):
            text = self._get_text_with_links(tag)
            if text.strip():
                paragraphs.append(
                    f"<{tag.name}>{text.strip()}</{tag.name}>")
        else:
            if any(p in processed for p in tag.parents):
                continue
            para = self._get_text_with_links(tag)
            if para.strip():
                paragraphs.append(f"<p>{para.strip()}</p>")
        processed.add(tag)    return "\n".join(paragraphs)
```

Лістинг В.4 – Метод _get_text_with_links() та нормалізація пробілів

```
     # Попередньо скомпільовані вирази (атрибути класу)
_RE_MULTI_SPACE  = re.compile(r"\s+")
_RE_SPACE_BEFORE = re.compile(r"\s+([.,!?;:])")
_RE_OPEN_BRACKET = re.compile(r"([«\u201c(\[]) +")
_RE_CLOS_BRACKET = re.compile(r" +([»\u201d)\]])")
_RE_AFTER_A      = re.compile(r"(</a>)(?=\w)")
_RE_BEFORE_A     = re.compile(r"(?<=\w)(<a )")

def _get_text_with_links(self, tag) -> str:
    parts = []
    for child in tag.descendants:
        if isinstance(child, NavigableString):
            parts.append(str(child))
        elif child.name == "a" and child.get("href"):
            href = child.get("href")
            text = child.get_text()
            parts.append(f'<a href="{href}">{text}</a>')
        elif child.name in ("strong", "b"):
            parts.append(f"<strong>{child.get_text()}</strong>")
        elif child.name in ("em", "i"):
            parts.append(f"<em>{child.get_text()}</em>")
        elif child.name == "br":
            parts.append(" ")
    result = "".join(parts)
    result = self._RE_MULTI_SPACE.sub(" ", result)
    result = self._RE_SPACE_BEFORE.sub(r"\1", result)
    result = self._RE_OPEN_BRACKET.sub(r"\1", result)
    result = self._RE_CLOS_BRACKET.sub(r"\1", result)
    result = self._RE_AFTER_A.sub(r"\1 ", result)
    result = self._RE_BEFORE_A.sub(r" \1", result)    return
result.strip()
```

Лістинг В.5– Диспетчер convert() та метод prepare_short_description()

```
     def convert(self, html: str,
                 content_type: str = "product_description"
```

```
            ) -> ProcessingResult:
        dispatch = {
            "product_description":
     self.prepare_product_description,
            "short_description"  :
     self.prepare_short_description,
            "cms_block"          : self.prepare_cms_block,
            "cms_page"           : self.prepare_cms_page,
        }
        handler = dispatch.get(
            content_type, self.prepare_product_description)
        result  = handler(html)
        return ProcessingResult.ok(original=html,
     processed=result)

     def prepare_short_description(self, html: str) -> str:
        soup    = BeautifulSoup(html, "html.parser")
        first_p = soup.find("p")
        source  = first_p if first_p else soup
        text    = re.sub(r"\s+", " ",
                         source.get_text(separator=" ")).strip()
        limit   = MAGENTO_SHORT_DESC_MAX
        if len(text) > limit:
            text = (text[:limit - 1].rsplit(" ", 1)[0]
                    + "\u2026")  # U+2026 HORIZONTAL ELLIPSIS
     return text
```

Лістинг В.6 – Методи _sanitize_attributes() та generate_product_attributes()

```
def _sanitize_attributes(self, soup,
                         keep_href=True,
                         keep_class=False) -> None:
    for tag in soup.find_all(True):
        if not hasattr(tag, "attrs"): continue
        new_attrs: dict = {}
        if keep_href and tag.name == "a":
            if href := tag.get("href", ""):
                new_attrs["href"] = href
        if tag.name == "img":
            for attr in ("src", "alt", "width", "height"):
                if v := tag.get(attr): new_attrs[attr] = v
        if keep_class:
            if css := tag.get("class"): new_attrs["class"] = css
        tag.attrs = new_attrs

def generate_product_attributes(self, html: str) -> dict:
    short = self.prepare_short_description(html)
    meta  = short[:MAGENTO_META_DESC_MAX]
    if len(short) > MAGENTO_META_DESC_MAX:
        meta = meta.rsplit(" ", 1)[0] + "\u2026"
```

```
    return {
        "description"      :
self.prepare_product_description(html),
        "short_description": short,
        "meta_description" : meta,    }
```

Лістинг В.7 – Метод detect_and_format_lists() сервісу TextFormatter

```
     _UL_RE = re.compile(r"^[-*•\u2013\u2022]\s+(.*)")
_OL_RE = re.compile(r"^\d+[.)\]]\s+(.*)")

def detect_and_format_lists(self, text: str) -> str:
    result, in_ul, in_ol = [], False, False
    def _close():
        nonlocal in_ul, in_ol
        if in_ul: result.append("</ul>"); in_ul = False
        if in_ol: result.append("</ol>"); in_ol = False
    for line in text.splitlines():
        s = line.strip()
        if not s:
            _close(); result.append(""); continue
        if ul_m := self._UL_RE.match(s):
            if in_ol: result.append("</ol>"); in_ol = False
            if not in_ul: result.append("<ul>"); in_ul = True
            result.append(f"  <li>{ul_m.group(1)}</li>")
        elif ol_m := self._OL_RE.match(s):
            if in_ul: result.append("</ul>"); in_ul = False
            if not in_ol: result.append("<ol>"); in_ol = True
            result.append(f"  <li>{ol_m.group(1)}</li>")
        else:
            _close()
            result.append(f"<p>{s}</p>")
    _close()    return "\n".join(result)
```

Лістинг В.8 – Клас _BatchWorker та методи BatchController

```
     class _BatchWorker(QThread):
    progress  = pyqtSignal(int, int, str)
    finished  = pyqtSignal(list)
    error     = pyqtSignal(str)
    def __init__(self, processor, file_paths, task):
        super().__init__()
        self._processor  = processor
        self.file_paths  = file_paths
        self.task        = task
    def run(self) -> None:
        try:
            self._processor.set_progress_callback(
                self._on_progress)
            results = self._processor.process_files(
```

```
                self.file_paths, self.task)
            self.finished.emit(results)
        except Exception as exc:
            self.error.emit(str(exc))

    def _on_progress(self, cur, tot, fname):
        self.progress.emit(cur, tot, fname)

    def cancel(self):        self._processor.cancel()
```

Лістинг В.9 – Фрагмент тестів класу HtmlCleaner (test_html_cleaner.py)

```
     @pytest.fixture
def cleaner():
    return HtmlCleaner()

class TestCleanHtml:
    def test_simple_paragraph(self, cleaner):
        result = cleaner.clean_html("<div><p>Hello</p></div>")
        assert result == "<p>Hello</p>"

    def test_removes_script_with_content(self, cleaner):
        html   = "<p>Text</p><script>alert(1)</script>"
        result = cleaner.clean_html(html)
        assert "<script>" not in result
        assert "alert" not in result

    def test_removes_style_attribute(self, cleaner):
        html   = '<p style="color:red">Text</p>'
        result = cleaner.clean_html(html)
        assert "style=" not in result
        assert "Text" in result

    def test_preserves_href_in_links(self, cleaner):
        html   = '<p><a href="/page">link</a></p>'
        result = cleaner.clean_html(html)
        assert 'href="/page"' in result

    def test_empty_paragraphs_removed(self, cleaner):
        html   = "<p></p><p>Content</p><p>  </p>"
        result = cleaner.clean_html(html)        assert
result.count("<p>") == 1
```

Лістинг В.10. – Тести граничних значень MagentoConverter

```
     class TestShortDescription:
         def test_max_length_respected(self, conv):
             long_text = "A" * 300
             result    = conv.prepare_short_description(
                 f"<p>{long_text}</p>")
             assert len(result) <= MAGENTO_SHORT_DESC_MAX
```

```
    def test_exactly_255_chars_unchanged(self, conv):
        text   = "B" * 255
        result =
conv.prepare_short_description(f"<p>{text}</p>")
        assert len(result) == 255
        assert not result.endswith("\u2026")

    def test_256_chars_truncated_with_ellipsis(self, conv):
        text   = "C " * 128  # 256 символів
        result =
conv.prepare_short_description(f"<p>{text}</p>")
        assert len(result) <= MAGENTO_SHORT_DESC_MAX
        assert result.endswith("\u2026")

    def test_meta_description_max_160(self, conv):
        long_text = "word " * 100
        attrs     = conv.generate_product_attributes(
            f"<p>{long_text}</p>")        assert
len(attrs["meta_description"]) <= 160
```