6. Nx Documentation. Monorepo for modern development. Available at: https://nx.dev/
7. Zod Documentation. TypeScript-first schema declaration and validation. Available at: https://zod.dev/
8. Nrwl. Nx: Smart Monorepos, Fast CI. Available at: https://nx.dev/concepts/mental-model
9. Rspack Team. Rspack: The Rust-based web bundler. Available at: https://www.rspack.dev/

TKACHENKO Oleksii,
Student, Department of Applied Mathematics and Informatics, The Bohdan Khmelnytsky National University of Cherkasy, Ukraine
DIDKOWSKY Ruslan,
Doctor of Technical Sciences, Associate Professor, Department of Informatics and Applied Mathematics, The Bohdan Khmelnytsky National University of Cherkasy, Ukraine

RESEARCH INTO MICROFRONTEND ARCHITECTURE FOR BUILDING SCALABLE WEB SYSTEMS

Summary. Introduction. *Modern web applications have evolved from simple informational pages into complex enterprise systems. With the growth in codebase and the number of development teams, traditional monolithic frontend architectures exhaust their potential, creating barriers to further product development. The concept of Micro Frontends emerged as an adaptation of microservice architecture principles to the client side of web applications.*

Purpose. *The aim of this article is to substantiate a methodology for designing and implementing scalable web systems based on microfrontend architecture using Module Federation technology, and to compare its effectiveness with traditional approaches.*

Results. *A comparative analysis of microfrontend integration methods was conducted: Build-time Integration, iframe-based isolation, and Run-time Integration. The Shell-Remote architectural pattern was selected and implemented using Module Federation technology within the Nx monorepo environment with the Rspack bundler. A monorepo structure was developed with a clear separation into responsibility layers (Applications, Feature Libraries, Shared UI, Data Access). A solution to the shared state problem was implemented using native React Context API combined with the Singleton configuration for the React framework instance. An experimental study was conducted comparing build performance and loading metrics between the proposed architecture and traditional Webpack-based monolith.*

Conclusion. *The experimental study confirmed the high effectiveness of the proposed architecture: the transition to Rspack provided a 12-fold build speed increase compared to Webpack; applying the Lazy Loading strategy reduced the initial page bundle size by 81%; Core Web Vitals metrics were achieved (LCP < 1 second); the use of Nx tooling reduced CI/CD testing time by 5–6 times for local changes. The obtained results confirm the practical value of the investigated approach for building enterprise-level web systems.*

Keywords: *microfrontend architecture, Module Federation, monorepo, Nx, Rspack, Shell-Remote, scalability, build performance.*

Одержано редакцією 10.11.2025 р.
Прийнято до публікації 17.12.2025 р.

УДК 004.415.2:004.738.5

DOI 10.31651/2076-5886-2025-1-95-106

PACS 07.05.Tp, 89.20.Ff

ПЕДЧЕНКО Максим Анатолійович
студент спеціальності «Прикладна математика» Черкаського національного університету імені Богдана Хмельницького
e-mail: pedchenko.maksym@vu.cdu.edu.ua

СЕРДЮК Олександр Анатолійович
кандидат економічних наук, доцент, доцент кафедри прикладної математики та інформатики Черкаського національного

університету імені БогданаХмельницького
e-mail: serdyuk@ukr.net
ORCID 0000-0002-3919-4661

ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ІНТЕГРОВАНОЇ СИСТЕМИ УПРАВЛІННЯ ТА АГРЕГАЦІЇ ВЕБ-КОНТЕНТУ

У роботі розглянуто проблему структурованого збереження веб-контенту та запропоновано архітектурне рішення у вигляді браузерного розширення Digital Hub. Описано багатоетапний конвеєр вилучення контенту, що виконує семантичний аналіз веб-сторінок, видалення незначущих елементів та конвертацію у формат Markdown. Запропоновано систему шаблонів із типізованими змінними, ланцюжками фільтрів та тригерами автоматичного вибору шаблону. Охарактеризовано архітектуру браузерного розширення з ізольованими контекстами виконання та типізованим обміном повідомленнями між ними. Наведено порівняльний аналіз обраних технологій та відомих рішень у сфері вилучення контенту. Практичне значення роботи полягає у створенні функціонального інструменту для персонального управління знаннями, що забезпечує структуроване збереження, пошук та організацію веб-контенту.

Ключові слова: *браузерне розширення, вилучення веб-контенту, управління знаннями, система шаблонів, Markdown, семантичний аналіз.*

Вступ

Сучасний веб є основним джерелом інформації для дослідницької, професійної та навчальної діяльності. Користувачі щодня стикаються зі статтями, документацією, навчальними матеріалами та довідковими ресурсами, обсяг яких перевищує можливості їх миттєвого опрацювання. Це породжує проблему збереження контенту: як зафіксувати знайдений матеріал, не втративши тих якостей, що роблять його цінним.

Традиційні підходи до вирішення цієї проблеми мають суттєві обмеження. Закладки зберігають лише адресу URL, яка стає непридатною після переміщення або видалення сторінки. Знімки екрана зберігають візуальний вигляд, але виключають можливість пошуку за текстом. Ручне копіювання контенту є трудомістким процесом, у якому нерідко втрачається структура і метадані. Наслідком є фрагментація інформації: дані виявляються розкиданими по різних сховищах без єдиного механізму пошуку [6].

Актуальність теми визначається зростаючою залежністю сучасних робочих процесів від веб-ресурсів. Браузерні розширення займають унікальну позицію у цьому контексті, оскільки функціонують саме в той момент, коли користувач натрапляє на цінний матеріал. Систематизований підхід до захоплення контенту – такий, що зберігає текст, підтримує структуру документа та автоматично вилучає доступні метадані – забезпечує якісно ліпшу організацію знань порівняно зі звичайним збереженням посилань.

Метою дослідження є проектування та реалізація браузерного розширення для структурованого захоплення та управління веб-контентом, а також аналіз архітектурних рішень, що забезпечують ефективне вилучення, обробку та зберігання інформації з веб-сторінок.

Для досягнення мети поставлено такі завдання: розробити конвеєр вилучення основного контенту веб-сторінок; спроектувати гнучку систему шаблонів для визначення правил вилучення; реалізувати інтерфейси захоплення та управління нотатками; дослідити архітектурні особливості браузерних розширень у контексті ізоляції контекстів виконання.

Виклад основного матеріалу

1. Огляд існуючих рішень у сфері вилучення веб-контенту

Проблема вилучення основного контенту з веб-сторінок вивчається в контексті веб-скрейпінгу та обробки природньої мови вже декілька десятиліть [6]. Нині існує ряд практичних інструментів, кожен з яких має власні переваги та обмеження.

Mozilla Readability – найширше розповсюджена бібліотека вилучення контенту, яка живить режим читача Firefox і слугує основою для численних інших інструментів [7]. Вона перетворює HTML-сторінки на спрощений формат, придатний для читання без відволікань. Бібліотека використовує переважно підхід на основі оцінювання: елементи оцінюються за щільністю тексту, наявністю параграфів та евристиками, відшліфованими протягом років. Обмеженням є те, що Readability не виконує розгорнутої нормалізації структур, а її підхід орієнтований на відображення, а не на подальшу обробку даних.

Системи управління нотатками (Obsidian, Notion, Roam Research) надають інструменти для організації знань, але не вирішують проблему структурованого вилучення контенту з веб. Зазвичай вони покладаються на плагіни або зовнішні сервіси для захоплення інформації.

Порівняльний аналіз підходів до вилучення контенту наведено у таблиці 1.

Аналіз таблиці свідчить, що жоден із існуючих підходів не поєднує автоматичне структуроване вилучення з гнучкою системою шаблонів та вбудованим пошуком.

Таблиця 1

Порівняльний аналіз підходів до вилучення веб-контенту

Критерій	Mozilla Readability	Ручне копіювання	Браузерні закладки	Digital Hub
Збереження структури	Часткове	Залежить від інструменту	Ні	Так (Markdown)
Вилучення метаданих	Часткове	Ручне	Ні	Автоматичне
Підтримка шаблонів	Ні	Ні	Ні	Так
Пошук за контентом	Ні	Залежить від інструменту	Ні	Так
Налаштовуваність	Мінімальна	Повна	Ні	Висока

2. Архітектура системи Digital Hub

Система Digital Hub реалізована у вигляді браузерного розширення з чотирма основними компонентами: конвеєром вилучення контенту, системою шаблонів, спливаючим інтерфейсом захоплення та хаб-застосунком для управління нотатками.

2.1. Архітектура браузерного розширення

Браузерні розширення функціонують принципово інакше, ніж звичайні веб-застосунки: їхня функціональність розподілена між кількома ізольованими контекстами виконання, кожен з яких має власні можливості та обмеження [13].

У системі Digital Hub використовується п'ять основних контекстів (рис. 1):

– фоновий service worker – обробляє події життєвого циклу розширення, маршрутизує повідомлення між контекстами та реагує на комбінації клавіш;

– контентні скрипти – виконуються в контексті веб-сторінок, забезпечуючи запуск конвеєра вилучення та моніторинг виділення тексту;
– спливаюче вікно (popup) – компактний інтерфейс для захоплення контенту;
– бічна панель (sidepanel) – аналогічний інтерфейс у форматі постійної бічної панелі;
– хаб-застосунок – повноекранний інтерфейс для управління нотатками.

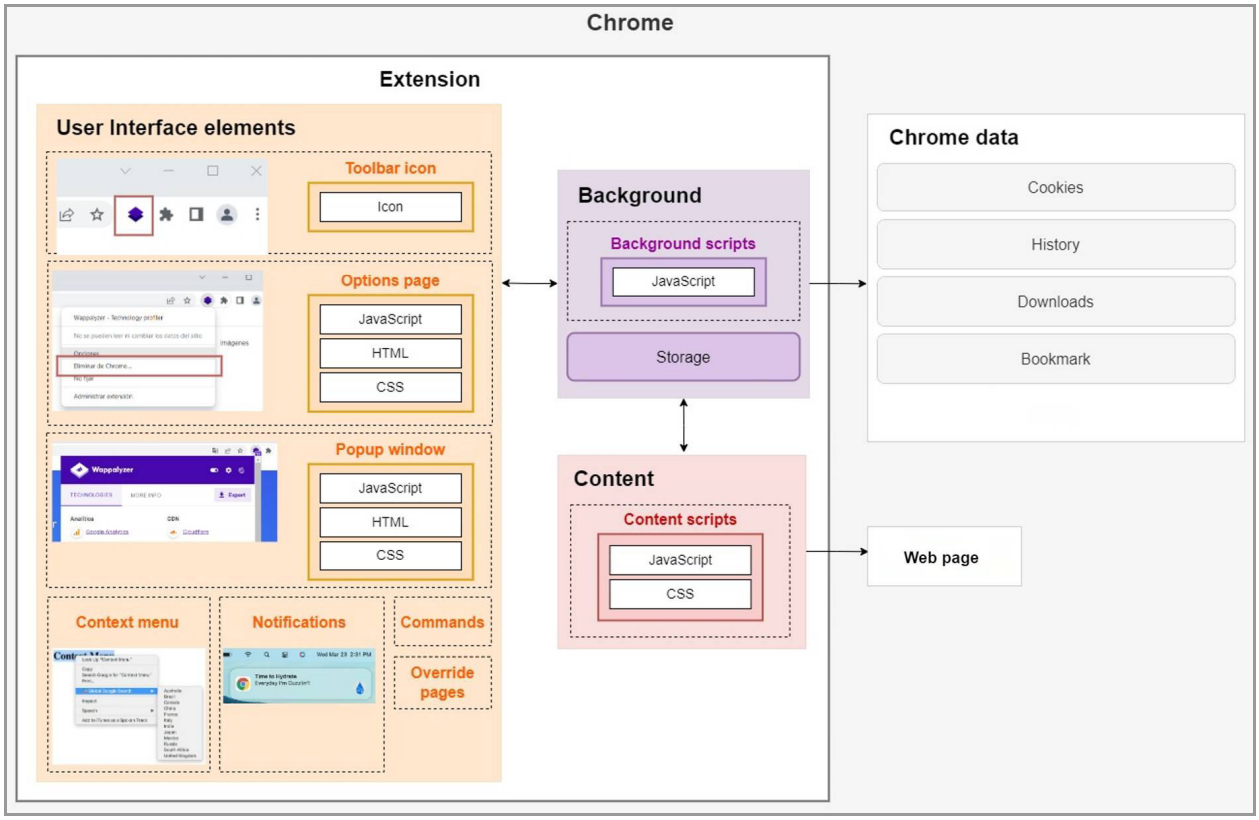

Рис. 1. Архітектура браузерного розширення Chrome із зазначенням контекстів виконання та каналів зв'язку між ними

Принциповою проблемою є комунікація між ізольованими контекстами, яка традиційно здійснюється через нетипізований обмін повідомленнями. У роботі цю проблему вирішено за допомогою типізованої бібліотеки обміну повідомленнями: визначається протокольний інтерфейс, що описує всі допустимі повідомлення разом з очікуваними корисними навантаженнями та типами повернення. Невідповідність сигнатури обробника визначенню протоколу виявляється під час компіляції TypeScript, а не в процесі виконання.

2.2. Конвеєр вилучення контенту

Веб-сторінки містять значну кількість «шуму»: меню навігації, рекламні блоки, бокові панелі, нижні колонтитули та різноманітні елементи інтерфейсу. Конвеєр вилучення перетворює такий хаотичний HTML на чистий структурований контент шляхом послідовного виконання п'яти етапів (рис. 2).

Етап 1: Ініціалізація. Виконується глибоке копіювання документа – усі подальші операції здійснюються над копією, що унеможливлює модифікацію вихідної сторінки. Одночасно обчислюються медіа-запити CSS для захоплення адаптивних стилів, а також ідентифікуються дрібні зображення та іконки, що є елементами інтерфейсу, а не контентом.

Етап 2: Виявлення основного контенту. Конвеєр виконує пошук основного

контейнера контенту за ієрархічним списком семантичних селекторів: від специфічних (#post, .article-content) до загальних (article, main, body). Якщо семантичне виявлення не дає результату, задіюються резервні механізми: виявлення на основі таблиць для застарілих макетів та оцінювання блочних елементів для всього документа.

Етап 3: Видалення зайвого. Приховані елементи видаляються першими. Далі застосовується алгоритм оцінювання блоків: елементи з ознаками не-контенту (висока щільність посилань, навігаційні текстові патерни) вилучаються. Нарешті, точні та часткові CSS-селектори видаляють елементи, що відповідають відомим патернам зайвого контенту (sidebar, newsletter тощо).

Етап 4: Стандартизація та трансформація. Нормалізуються пробіли, реструктуруються заголовки (всі H1 перетворюються на H2, щоб назва документа виступала логічним H1), уніфікуються виноски. Трансформатори конвертують блоки коду різних систем підсвічування синтаксису до стандартного формату <pre><code>, обробляють зображення з відкладеним завантаженням та перетворюють div-структури на семантичні списки.

Етап 5: Фінальне очищення та контроль якості. Видаляються непотрібні атрибути, порожні елементи та «хвостові» заголовки. Якщо обсяг результату становить менше 2000 символів, конвеєр запускається повторно з пом'якшеними налаштуваннями, і повертається той результат, що містить більше контенту. Такий механізм повторної спроби балансує між агресивним очищенням та збереженням контенту.

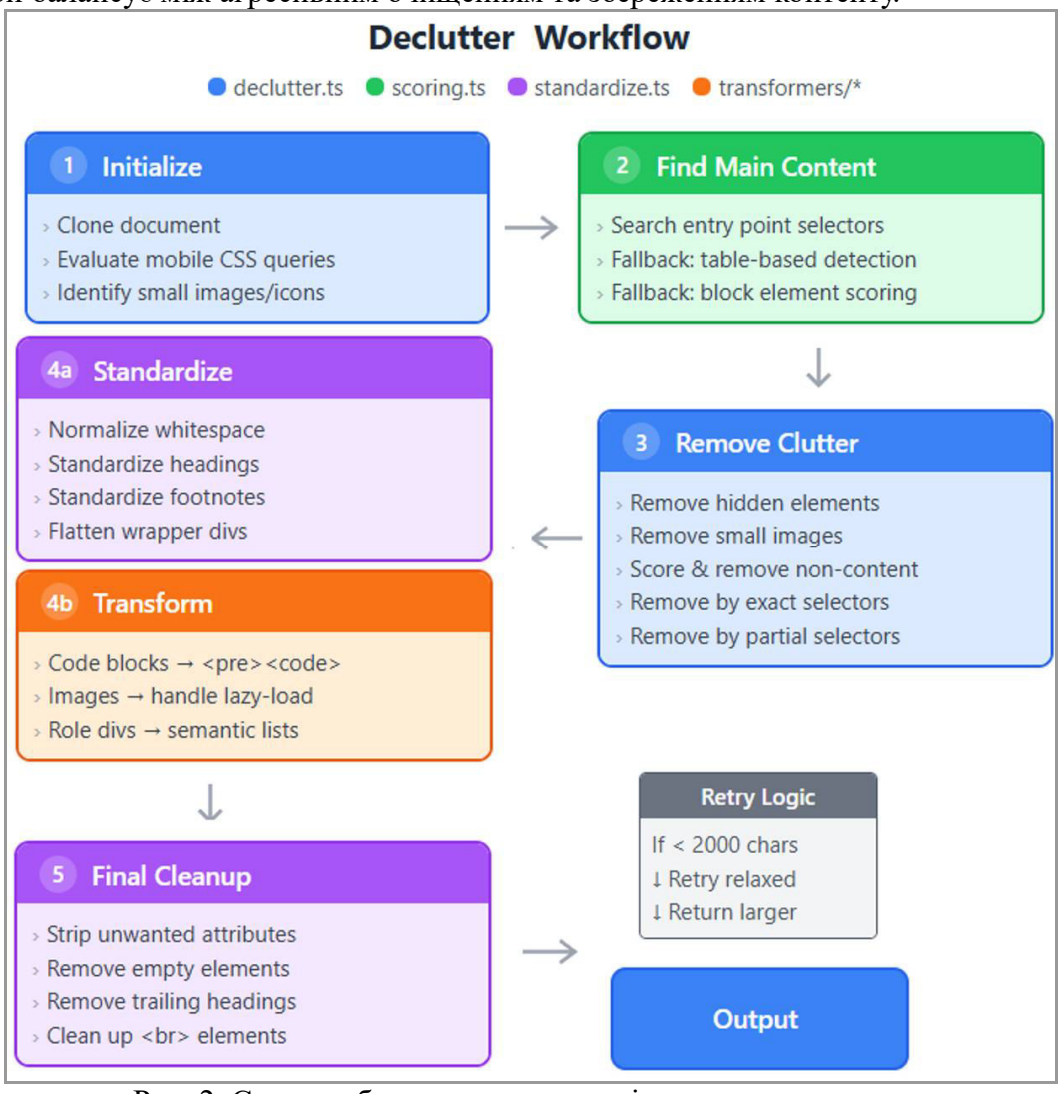

Рис. 2. Схема робочого процесу рушія очищення контенту

Принциповою відмінністю від Mozilla Readability є інвертований пріоритет: тоді як Readability покладається переважно на оцінювання вмісту, конвеєр Digital Hub використовує семантичні селектори як основну стратегію, а оцінювання – як резервну. Це прискорює роботу на сучасних сайтах, збудованих відповідно до стандартів HTML5. Додатково конвеєр враховує мобільні медіа-запити CSS – клас патернів, що не підтримується бібліотекою Readability.

2.3. Вилучення метаданих

Веб-сторінки містять структуровані метадані, що описують їхній зміст. Конвеєр вилучає ці дані з трьох джерел:

HTML мета-теги. Стандартні мета-теги та теги протоколу Open Graph [8] містять заголовок, опис, зображення та тип контенту. Протокол Open Graph, запроваджений для забезпечення якісного попереднього перегляду при поширенні посилань, набув практично загального поширення серед видавців контенту. Його властивості (og:title, og:description, og:image) часто є більш точними, ніж стандартні HTML-елементи.

Schema.org JSON-LD. Словник Schema.org [9] підтримує складні вкладені структури, що описують сутності та їхні зв'язки. Конвеєр обходить вміст JSON-LD рекурсивно, генеруючи шляхи у точковій нотації для кожного знайденого значення. Наприклад, schema:@Article:author[0].name надає доступ до імені першого автора.

Узагальнені змінні. Одна й та сама інформація нерідко зустрічається у кількох місцях. Замість того, щоб вимагати від авторів шаблонів розуміння цих варіацій, узагальнені змінні резолюються через ланцюжки запасних варіантів: для заголовка спочатку перевіряється `og:title`, потім `twitter:title`, далі `Schema.org headline` і, врешті, HTML-елемент `<title>`.

2.4. Система шаблонів

Система шаблонів надає механізм для визначення правил перетворення вилучених даних у структуровані нотатки. Шаблон містить рядки формату для заголовка та змісту, масив визначень властивостей і необов'язкові тригери для автоматичного вибору.

Типи змінних. Базовий синтаксис замінників використовує подвійні фігурні дужки за угодами, встановленими безлогічними системами шаблонів типу Mustache [12]. Підтримуються п'ять типів змінних (таблиця 2).

Фільтри. Синтаксис конвеєра дозволяє підключати фільтри до будь-якого посилання на змінну: {{title|upper}} переводить заголовок у верхній регістр, {{published|date:YYYY-MM-DD}} форматує дату. Кілька фільтрів можна поєднувати в ланцюжок, і кожен отримує вихід попереднього як вхід. Парсер фільтрів відстежує стан при ітерації по рядку фільтра: чи перебуває він всередині лапок, всередині регулярного виразу або всередині вкладених дужок. Лише символи вертикальної риски поза цими захищеними контекстами слугують роздільниками фільтрів.

Тригери. Тригери забезпечують автоматичний вибір шаблону на основі характеристик сторінки. Система підтримує три типи патернів: збіг підрядка URL, регулярні вирази та збіг Schema.org. Наприклад, шаблон для Goodreads може мати два тригери: "https://www.goodreads.com/book/" (підрядок URL) та "schema:@Book" (тип Schema.org). Збіг Schema.org може перевіряти наявність типу, наявність властивості або значення властивості з різним ступенем специфічності.

Механізм кешування покращує продуктивність оцінки тригерів: повторні оцінки для однієї й тієї ж комбінації URL та типу Schema.org повертають кешовані результати.

Таблиця 2

Типи змінних системи шаблонів

Тип	Синтаксис	Приклад	Джерело
Загальні	{{title}}	{{author}}	Абстраговані метадані
Мета	{{meta:property:og:image}}	{{meta:name:description}}	HTML мета-теги
Schema	{{schema:@Book:author[*].name}}	{{schema:headline}}	JSON-LD структуровані дані
Селектор	{{selector:h1.title}}	{{selectorHtml:.content}}	Живий запит до DOM
LLM-підказка	{{"3 теги через кому"}}	{{"короткий опис"}}	Відкладена обробка ШІ

2.5. Конвертація HTML у Markdown

Формат Markdown [10] обрано як формат зберігання контенту завдяки поєднанню людиночитабельності, структурної виразності та портативності. Контент у Markdown залишається придатним для використання в інших застосунках, редагуванням без спеціальних інструментів і версійного контролю.

Бібліотека Turndown [11] виконує конвертацію з HTML у Markdown за допомогою правил-трансформацій. Для ряду типів елементів знадобилась спеціальна обробка:

- Таблиці: Якщо таблиця містить атрибути colspan або rowspan, вона зберігається як HTML, щоб уникнути втрат при конвертації. Прості таблиці конвертуються у синтаксис Markdown.
- Вкладені списки: Для більш надійного вкладення у різних Markdown-рендерах замість пробілів використовуються символи табуляції.
- Блоки коду: Правило вилучає інформацію про мову програмування з різних джерел (класи, атрибути даних) і формує огороджений блок коду з відповідним тегом мови.
- Виноски: Конвертуються у посилання у стилі посилань, підтримуваних розширеним Markdown.

Усі відносні URL резолюються до абсолютної форми до початку конвертації Markdown, щоб посилання залишалися дійсними поза вихідним контекстом сторінки.

2.6. Технологічний стек

Технологічний стек системи формувався з урахуванням вимог до продуктивності розробки та довгострокової підтримуваності. У таблиці 3 наведено основні технологічні рішення та обґрунтування їх вибору.

Таблиця 3

Основні технологічні рішення системи Digital Hub

Компонент	Обрана технологія	Основне обґрунтування
UI-фреймворк	React 19 [1]	Компонентна архітектура, hooks-модель, широка екосистема
Мова програмування	TypeScript [2]	Статична перевірка типів, виявлення помилок під час компіляції
Фреймворк розширення	WXT [4]	Конвенція над конфігурацією, вбудована абстракція сховища
UI-компоненти	Mantine [5]	Відповідна естетика, hooks-бібліотека, підтримка CSS-in-JS
Конвертація HTML→MD	Turndown [11]	Зрілість, активна підтримка, розширювані правила
Маршрутизація	TanStack Router	Підтримка хеш-маршрутизації для chrome-extension протоколу

Вибір TypeScript суттєво вплинув на якість коду: розширення передбачає складні моделі даних (нотатки з властивостями та метаданими, шаблони з типізованими фільтрами, виділення з контекстуальною інформацією), і статична перевірка типів під час компіляції дозволила виявити ряд потенційних помилок ще до тестування.

Браузерні розширення функціонують у кількох ізольованих контекстах, і комунікація між ними традиційно є нетипізованою. Типізований протокол обміну повідомленнями, реалізований за допомогою TypeScript, гарантує, що обробники повідомлень відповідають очікуваним сигнатурам, що виявляє невідповідності під час компіляції.

Хаб-застосунок використовує хеш-маршрутизацію через обмеження протоколу chrome-extension://, який не підтримує маніпуляцію з pushState. Це накладало архітектурні обмеження, які було враховано при виборі маршрутизатора.

2.7. Інтерфейси системи

Спливаюче вікно (Popup). Спливаюче вікно та бічна панель використовують один і той самий компонент, що обробляє обидва режими відображення (рис. 3). Інтерфейс містить: заголовок з вибором шаблону та редагованим полем заголовка; основну область, організовану у вкладки (Огляд, Властивості, Змінні, Контент, Виділення); нижній колонтитул з елементами керування.

Дані передаються через серію спеціалізованих хуків, кожен з яких відповідає за окремий аспект процесу захоплення: отримання налаштувань, координація з контентним скриптом для отримання контенту сторінки, перетворення метаданих у категоризовані записи для системи шаблонів, виявлення шаблону та його обробка.

Хаб-застосунок. Хаб надає повноекранний інтерфейс для управління нотатками з розділеним макетом, що відображає список нотаток поруч з вибраною нотаткою. Навігація організовує нотатки за тегами, зарезервованими категоріями (Вхідні, Вибране) та результатами повнотекстового пошуку (рис. 4). Пошук охоплює заголовки

нотаток, контент, теги, властивості та текст виділень.

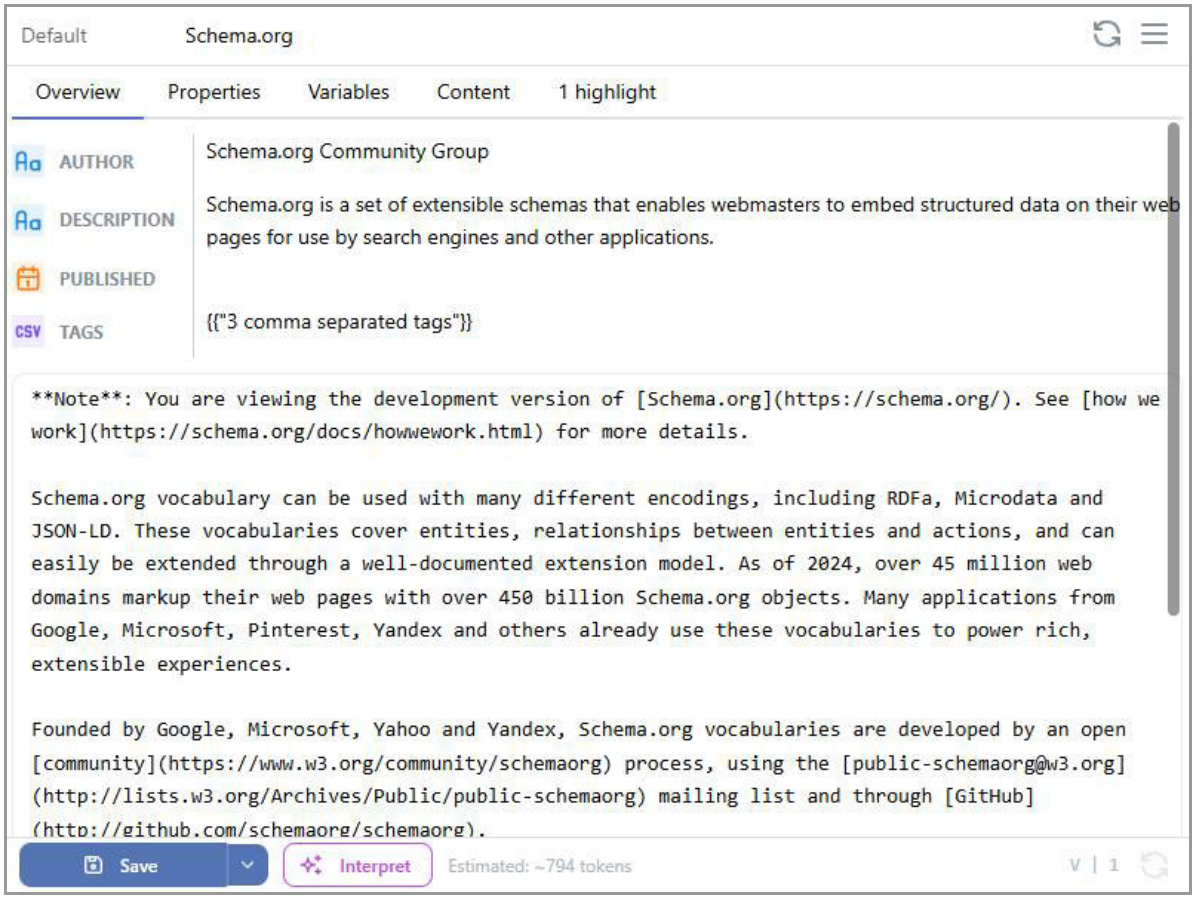

Рис. 3. Інтерфейс спливаючого вікна з відображенням вилученого контенту у вкладці «Огляд»

2.8. Наукова новизна та отримані результати

Науковою новизною роботи є комплексна система шаблонів, що поєднує типізоване вилучення змінних з кількох джерел (мета-теги, Schema.org, живі DOM-запити), ланцюжки фільтрів для перетворення значень та тригерні патерни для автоматичного вибору шаблону на основі характеристик сторінки. Такий підхід відрізняється від існуючих рішень, орієнтованих переважно на відображення контенту без підтримки структурованої екстракції даних.

Порівняно з Mozilla Readability, яка є найширше розповсюдженим аналогом, конвеєр Digital Hub забезпечує: пріоритет семантичних селекторів над оцінюванням для швидшої роботи на сучасних сайтах; підтримку мобільних медіа-запитів CSS для виявлення прихованих елементів; розширену систему трансформаторів для нормалізації нестандартних структур; механізм повторної спроби для балансування між агресивним очищенням і збереженням контенту.

Механізм захоплення виділень (Highlighter) забезпечує легковагий спосіб захоплення тексту незалежно від повного конвеєра вилучення. Кожне виділення зберігається з оточуючим контекстом для полегшення пошуку у майбутньому.

Інтеграція великих мовних моделей реалізована через змінні LLM-підказок, що відкладають генерацію значень до окремої фази обробки з передачею контексту сторінки. Це дозволяє автоматично генерувати теги, короткі описи та інші структуровані властивості на основі вмісту сторінки.

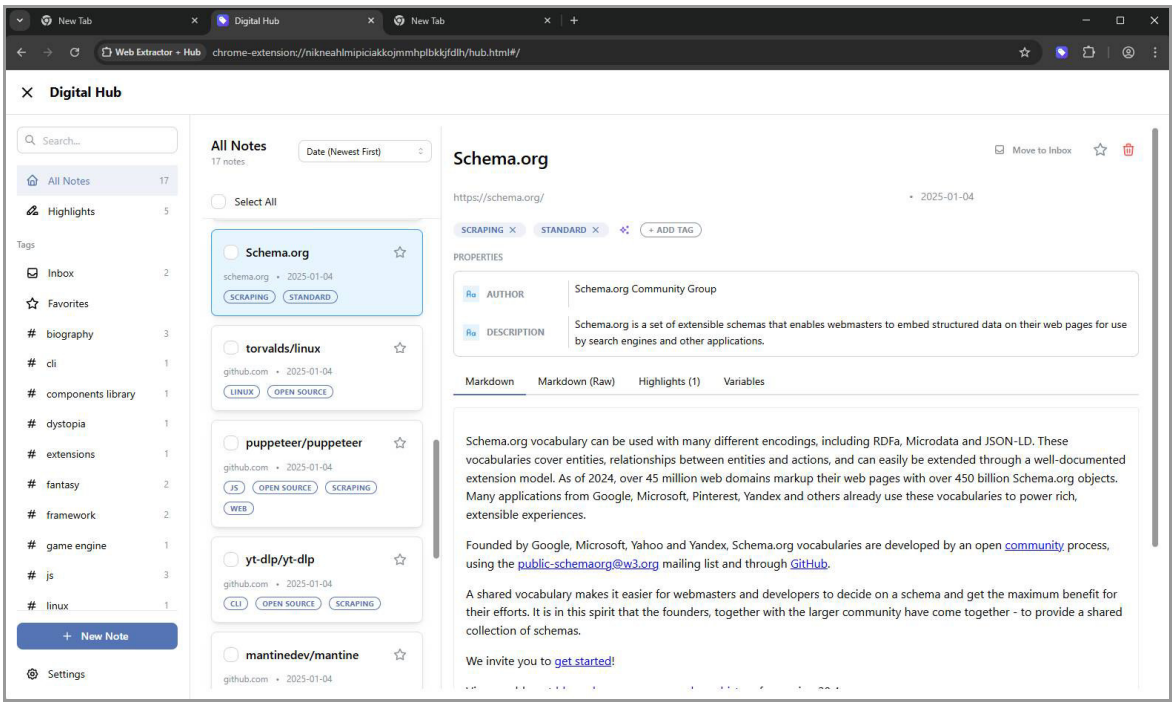

Рис. 4. Головний вигляд хабу з бічною навігаційною панеллю, списком нотаток та панеллю деталей

Висновки

У статті описано архітектуру та реалізацію браузерного розширення Digital Hub – системи для структурованого захоплення та управління веб-контентом. Розроблений конвеєр вилучення контенту реалізує п'ятиетапний процес семантичного аналізу, видалення зайвого та нормалізації з механізмом повторної спроби. Система шаблонів з п'ятьма типами змінних, гнучкими ланцюжками фільтрів та тригерами автоматичного вибору надає користувачам інструмент визначення структурованих правил вилучення для різних типів контенту.

Архітектура браузерного розширення з кількома ізольованими контекстами виконання та типізованим протоколом обміну повідомленнями демонструє підхід до побудови складних клієнтських систем з гарантіями типобезпеки під час компіляції.

Отримані результати показують, що поєднання автоматизованого вилучення з визначуваними користувачем правилами обробки пропонує ефективний компроміс між повністю ручним захопленням (трудомістким, але точним) і повністю автоматичними системами (зручними, але негнучкими). Система забезпечує функціональну основу для робочих процесів персонального управління знаннями, перетворюючи ефемерні веб-сторінки на структуровані, придатні для пошуку нотатки.

Подальші напрямки розвитку системи включають: розширення підтримки форматів виведення; покращення алгоритму оцінювання контенту для нестандартних макетів; інтеграцію з популярними системами управління знаннями.

Список використаної літератури

1. React Documentation [Електронний ресурс] // React. – Режим доступу: https://react.dev. – Назва з екрана.
2. TypeScript Documentation [Електронний ресурс] // TypeScript. – Режим доступу: https://www.typescriptlang.org/docs. – Назва з екрана.
3. Banks A. Learning React / A. Banks, E. Porcello. – 2nd ed. – Sebastopol : O'Reilly Media, 2020. – 310 p.
4. WXT Documentation [Електронний ресурс] // WXT. – Режим доступу: https://wxt.dev. – Назва з екрана.

5. Mantine Documentation [Електронний ресурс] // Mantine. – Режим доступу: https://mantine.dev. – Назва з екрана.
6. Mitchell R. Web Scraping with Python: Collecting More Data from the Modern Web / R. Mitchell. – 2nd ed. – Sebastopol : O'Reilly Media, 2018. – 306 p.
7. Readability [Електронний ресурс] // GitHub. – Режим доступу: https://github.com/mozilla/readability. – Назва з екрана.
8. The Open Graph protocol [Електронний ресурс]. – Режим доступу: https://ogp.me. – Назва з екрана.
9. Schema.org [Електронний ресурс]. – Режим доступу: https://schema.org. – Назва з екрана.
10. Gruber J. Markdown: Syntax [Електронний ресурс] // Markdown Syntax. – Режим доступу: https://daringfireball.net/projects/markdown/syntax. – Назва з екрана.
11. Turndown [Електронний ресурс] // GitHub. – Режим доступу: https://github.com/mixmark-io/turndown. – Назва з екрана.
12. Mustache: Logic-less templates [Електронний ресурс] // Mustache. – Режим доступу: https://mustache.github.io/mustache.5.html. – Назва з екрана.
13. Anatomy of an extension [Електронний ресурс] // MDN Web Docs. – Режим доступу: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Anatomy_of_a_WebExtension. – Назва з екрана.
14. Norman D. A. The Design of Everyday Things / D. A. Norman. – MIT Press Ltd, 2014. – 368 p.

References

1. React Documentation [Electronic resource] // React. – Access mode: https://react.dev. – Title from screen.
2. TypeScript Documentation [Electronic resource] // TypeScript. – Access mode: https://www.typescriptlang.org/docs. – Title from screen.
3. Banks A. Learning React / A. Banks, E. Porcello. – 2nd ed. – Sebastopol : O'Reilly Media, 2020. – 310 p.
4. WXT Documentation [Electronic resource] // WXT. – Access mode: https://wxt.dev. – Title from screen.
5. Mantine Documentation [Electronic resource] // Mantine. – Access mode: https://mantine.dev. – Title from screen.
6. Mitchell R. Web Scraping with Python: Collecting More Data from the Modern Web / R. Mitchell. – 2nd ed. – Sebastopol : O'Reilly Media, 2018. – 306 p.
7. Readability [Electronic resource] // GitHub. – Access mode: https://github.com/mozilla/readability. – Title from screen.
8. The Open Graph protocol [Electronic resource]. – Access mode: https://ogp.me. – Title from screen.
9. Schema.org [Electronic resource]. – Access mode: https://schema.org. – Title from screen.
10. Gruber J. Markdown: Syntax [Electronic resource] // Markdown Syntax. – Access mode: https://daringfireball.net/projects/markdown/syntax. – Title from screen.
11. Turndown [Electronic resource] // GitHub. – Access mode: https://github.com/mixmark-io/turndown. – Title from screen.
12. Mustache: Logic-less templates [Electronic resource] // Mustache. – Access mode: https://mustache.github.io/mustache.5.html. – Title from screen.
13. Anatomy of an extension [Electronic resource] // MDN Web Docs. – Access mode: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Anatomy_of_a_WebExtension. – Title from screen.
14. Norman D. A. The Design of Everyday Things / D. A. Norman. – MIT Press Ltd, 2014. – 368 p.

PEDCHENKO Maksym,
Student, Department of Applied Mathematics and Informatics, The Bohdan Khmelnytsky National University of Cherkasy, Ukraine
SERDIUK Oleksandr,
Candidate of Economic Sciences, Associate Professor, Department of Informatics and Applied Mathematics, The Bohdan Khmelnytsky National University of Cherkasy, Ukraine

DESIGN AND IMPLEMENTATION OF AN INTEGRATED WEB CONTENT MANAGEMENT AND AGGREGATION SYSTEM

Summary. Introduction. *The modern web is an overwhelming source of information for research, professional and learning activities. Traditional approaches to content preservation – bookmarks, screenshots, manual copying – have significant drawbacks. Bookmarks store only URLs that become useless when pages move or disappear. Screenshots sacrifice searchability. Manual copying is tedious and strips away metadata. The result is fragmented information scattered across multiple tools, difficult to search and disconnected from its original context. Browser extensions*

occupy a unique position in this workflow, operating at the moment when users encounter valuable content.

Purpose. *The primary objective is to design and implement a browser extension for structured web content capture and management, and to analyse the architectural solutions that ensure effective extraction, processing and storage of information from web pages.*

Results. *The Digital Hub browser extension implements a multi-stage content extraction pipeline that performs semantic analysis of web pages, removal of irrelevant elements and conversion to Markdown format. The pipeline prioritises semantic CSS selectors for content detection over scoring-based approaches, providing faster operation on modern HTML5-compliant sites compared to solutions like Mozilla Readability. A template system with five variable types (common, meta, schema, selector and LLM prompt), filter chains for value transformation, and trigger patterns for automatic template selection enables user-defined extraction rules for different content types. The architecture of the browser extension with five isolated execution contexts and a typed messaging protocol ensures type safety at compile time. A comparative analysis of existing approaches demonstrates that no current solution combines automatic structured extraction with a flexible template system and built-in search.*

Conclusion. *The combination of automated extraction with user-defined processing rules offers an effective compromise between fully manual capture and fully automatic systems. The system provides a functional foundation for personal knowledge management workflows, transforming ephemeral web pages into structured, searchable notes. Further development directions include expanding output format support and improving the content scoring algorithm for non-standard page layouts.*

Keywords: *browser extension, web content extraction, knowledge management, template system, Markdown, semantic analysis.*

Одержано редакцією 03.11.2025 р.
Прийнято до публікації 17.12.2025 р.