Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

CMS для управління контентом вебсайтів

(тема)

Виконав:
здобувач 3 року навчання,
групи КІУКІу-22-2
Даніл РИБКІН
(власне ім’я, прізвище)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: доц. Тетяна ФІЛІМОНЧУК
(посада, власне ім’я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Рибкіну Данілу Васильовичу
(прізвище, ім'я, по батькові)

1. Тема роботи CMS для управління контеном вебсайтів

затверджена наказом по університету від " 26 " травня 2025 р. № 425 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 14 липня 2025 р.

3. Вхідні дані до роботи 1) PHP як серверна мова програмування;
2) React для створення клієнтської частини;
3) REST API для взаємодії клієнта і сервера;
4) Flat-file підхід для зберігання даних;

4. Перелік питань, що потрібно опрацювати у роботі
1) аналіз предметної області;
2) аналіз та вибір технологій для розробки вебзастосунку;
3) опис програмної реалізації вебзастосунку;
4) інструкція користувача;
5) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд-презентація – 10 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз існуючих методів вирішення задачі	10.06.2025-13.06.25	
2	Вибір програмного забезпечення та	14.06.2025-17.06.25	
	інструментів розробки		
3	Проєктування архітектури застосунку	18.06.25-21.06.25	
4	Розробка логіки застосунку	22.06.25-28.06.25	
5	Розробка графічного інтерфейсу користувача	29.06.25-02.07.25	
6	Тестування застосунку	03.07.25-05.07.25	
7	Подання кваліфікаційної роботи керівникам	06.07.25-09.07.25	
	для попереднього захисту		
8	Подання кваліфікаційної роботи на	10.07.25-11.07.25	
	рецензування		

Дата видачі завдання " 09 " червня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Тетяна ФІЛІМОНЧУК
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 67 с., 19 рис., 1 дод., 13 джерел.

PHP, REACT, JSON, REST API, HTML, CSS, ANT DESIGN, FLAT-FILE, GIT, GULP, WEBPACK, CMS, UI, SEO.

Метою кваліфікаційної роботи є розробка легкої та інтуїтивної CMS на основі flat-file підходу для управління контентом невеликих вебсайтів. Основне завдання полягає у створенні системи, яка забезпечує редагування тексту, обробку зображень, налаштування мета-інформації та резервне копіювання з інтуїтивним інтерфейсом для користувачів із мінімальним технічним досвідом.

У ході виконання кваліфікаційної роботи було розроблено вебзастосунок, який використовує PHP для серверної логіки та React із бібліотекою Ant Design для клієнтської частини. Реалізовано REST API для асинхронної взаємодії між клієнтом та сервером, а також функції авторизації, редагування контенту, обробки зображень, налаштування SEO-параметрів і резервного копіювання. Інтерфейс включає навігаційну панель, візуальний редактор із підтримкою iframe, модальні вікна для підтверджень та зручне управління сторінками. CMS підходить для блогів та сайтів малого бізнесу, забезпечуючи легке розгортання, мінімальні вимоги до ресурсів і зручність для новачків.

ABSTRACT

Bachelor’s thesis: 67 pages, 19 figures, 1 appendix, 13 sources.

PHP, REACT, JSON, REST API, HTML, CSS, ANT DESIGN, FLAT-FILE, GIT, GULP, WEBPACK, CMS, UI, SEO.

The purpose of the qualification work is to develop a lightweight and intuitive CMS based on a flat-file approach for managing the content of small websites. The main objective is to create a system that enables text editing, image processing, meta-information configuration, and backup functionality with an intuitive interface for users with minimal technical experience.

During the development of the qualification work, a web application was created using PHP for server-side logic and React with the Ant Design library for the client-side interface. A REST API was implemented for asynchronous interaction between the client and server, along with features for authorization, content editing, image processing, SEO parameter configuration, and backup functionality. The interface includes a navigation panel, a visual editor with iframe support, modal windows for confirmations, and convenient page management. The CMS is suitable for blogs and small business websites, ensuring easy deployment, minimal resource requirements, and user-friendliness for beginners.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

AJAX – асинхронний JavaScript і XML (англ., Asynchronous JavaScript and XML)

API – інтерфейс прикладного програмування (англ., Application Programming Interface)

CMS – система керування контентом (англ., Content Management System)

CSS – каскадні таблиці стилів (англ., Cascading Style Sheets)

DOM – об’єктна модель документа (англ., Document Object Model)

HTML – мова розмітки гіпертексту (англ., HyperText Markup Language)

HTTP – протокол передачі гіпертексту (англ., HyperText Transfer Protocol)

JS – мова програмування JavaScript

JSON – формат обміну даними (англ., JavaScript Object Notation)

JSX – синтаксичне розширення для JavaScript (англ., JavaScript XML)

MVC – модель-вид-контролер (англ., Model-View-Controller)

PHP – мова програмування (англ. Hypertext Preprocessor)

REST – архітектурний стиль для вебсервісів (англ., Representational State Transfer)

SEO – оптимізація для пошукових систем (англ., Search Engine Optimization)

UI – інтерфейс користувача (англ., User Interface)

XML – розширювальна мова розмітки (англ., eXtensible Markup Language)

ВСТУП

Інтернет-ресурси в сучасному світі відіграють провідну роль у представленні інформації, забезпеченні комунікації та розвитку бізнес-процесів, стаючи незамінним інструментом для компаній, організацій та приватних осіб. Зростання їхньої популярності та функціонального значення зумовлене стрімким розвитком цифрових технологій, які дозволяють створювати вебсайти різного рівня складності. У центрі цього процесу перебувають системи керування контентом (CMS), які надають користувачам можливість зручно управляти вмістом сайтів: від текстів та зображень до складних структур даних, без необхідності володіти глибокими знаннями у сфері програмування чи веброзробки. Завдяки своїй універсальності та доступності CMS стали невід'ємною частиною сучасної інфраструктури веброзробки, що пояснює їхню високу популярність як серед професіоналів, так і серед початківців.

На даний час на ринку існує значна кількість готових CMS, серед яких найбільш відомими є WordPress [1], Joomla [2], Drupal [3], OpenCart [4] та інші. Ці системи пропонують широкий набір інструментів для створення вебсайтів різного призначення: від блогів та корпоративних порталів до інтернет-магазинів. Вони дозволяють редагувати текстовий вміст, додавати зображення, вставляти посилання, налаштовувати мета-інформацію для пошукової оптимізації та виконувати інші базові функції через інтуїтивно зрозумілий інтерфейс. Проте, попри їхню популярність та розвинений функціонал, такі рішення часто мають суттєві недоліки. Наприклад, надмірна складність для новачків, низька продуктивність при високих навантаженнях, обмежена гнучкість у реалізації нестандартних завдань або вразливість до кібератак через широке поширення. Ці проблеми стають особливо відчутними в проєктах, де потрібен індивідуальний підхід або оптимальна швидкодія. Саме тому розробка власних CMS, адаптованих до конкретних

потреб, набуває дедалі більшого значення в сучасній практиці веброзробки.

Актуальність даної роботи полягає в необхідності створення індивідуальних систем керування контентом, які усувають обмеження готових рішень. Такі системи дозволяють не лише підвищити продуктивність та безпеку, а й забезпечити максимальну простоту використання, що є критично важливим для користувачів із мінімальним технічним досвідом. Розробка подібного продукту відповідає сучасним тенденціям у веброзробці, де акцент дедалі частіше робиться на персоналізацію та оптимізацію робочих процесів. Крім того, індивідуальна CMS може бути більш економічно вигідною для невеликих проєктів, адже не потребує використання надлишкового функціоналу, який часто присутній у популярних системах.

Метою кваліфікаційної роботи є створення максимально простої системи керування контентом, яка забезпечить зручний та зрозумілий інтерфейс для управління вмістом вебсайтів [5]. Система передбачає базовий набір функцій, зокрема редагування текстової інформації, додавання та обробку зображень та налаштування мета-інформації для пошукових систем. Водночас вона буде гнучкою, адаптованою до потреб конкретних користувачів та легкою в освоєнні, що дозволить застосовувати її в проєктах різного масштабу – від особистих блогів до сайтів малого бізнесу. У процесі розробки особлива увага приділятиметься оптимізації коду та простоті інтеграції, щоб система могла слугувати ефективним інструментом навіть для тих, хто робить перші кроки у створенні вебконтенту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Система керування вмістом CMS

Система керування вмістом (Content Management System, CMS) – це програмне забезпечення, яке дозволяє користувачам створювати, редагувати, публікувати вебсайти та керувати ними без глибоких знань про програмне забезпечення чи веброзробку. CMS є важливим інструментом у сучасному цифровому світі, де вебсайти відіграють ключову роль у передачі інформації, комунікації, бізнесі та самовираженні. CMS широко використовується для керування вебсайтами різної складності – від блогів та корпоративних сторінок до інтернет-магазинів та інформаційних порталів.

Основна мета CMS — спростити процес роботи з вебконтентом, посилити його створення та редагування в технічній реалізації сайту. Користувач може додавати тексти, зображення, повідомлення та метадані через зручний інтерфейс без необхідності писати код.

Сучасні CMS поділяються на дві основні категорії: готові рішення (proprietary або open-source CMS) та самостійно розроблені системи. Готові рішення, такі як WordPress, Joomla, Drupal або OpenCart, пропонують широкий функціонал "із коробки" та мають розвинені спільноти користувачів та розробників, що спрощує пошук технічної підтримки.

Переваги готових CMS включають швидке розгортання, наявність документації та підтримку спільноти. Однак їхні недоліки стають очевидними при спробі реалізувати нестандартні задачі. Наприклад, додавання специфічного функціоналу часто потребує встановлення додаткових плагінів, що може призводити до зниження продуктивності сайту. Крім того, велика кількість коду, закладеного в таких системах, ускладнює їх оптимізацію для невеликих проєктів, де важлива швидкодія та простота. Безпека також є проблемою: популярність цих CMS робить їх

привабливими цілями для кібератак, а оновлення не завжди встигають за новими загрозами.

Готові системи часто пропонують більше, ніж потрібно для базового управління контентом, що суперечить принципам мінімалізму та зручності для користувачів із мінімальним досвідом. Основні функції CMS включають можливість додавання, редагування та видалення текстового контенту, завантаження зображень, налаштування мета-тегів для SEO-оптимізації, створення резервних копій.

Самостійно розроблені CMS створюються з урахуванням специфічних вимог сайту та забезпечують гнучкість, кращу продуктивність та вищий рівень безпеки. Таким чином, аналіз існуючих рішень показує необхідність створення легкої альтернативи, яка б поєднувала простоту використання з базовим набором функцій, достатнім для більшості невеликих вебпроєктів.

1.2 Аналіз існуючих CMS

1.2.1 Система керування контентом WordPress

WordPress (рисунок 1.1) є найпопулярнішою системою керування контентом (CMS) у світі, розроблена на PHP та використовує БД MySQL. Вона починалася як платформа для блогів, але з часом перетворилася на універсальний інструмент для створення сайтів різного типу: від особистих сторінок і портфоліо до корпоративних порталів та інтернет-магазинів. Система популярна через величезну екосистему плагінів, тем та активною спільнотою, яка налічує мільйони користувачів і розробників [1]. WordPress пропонує зручний та інтуїтивно зрозумілий інтерфейс адмін-панелі, який дозволяє навіть користувачам без знань програмування легко додавати контент, змінювати дизайн чи налаштовувати базові функції сайту. Також система забезпечує глибокі можливості кастомізації для тих, хто володіє технічними навичками: від редагування коду тем до створення власних плагінів чи інтеграції зі сторонніми сервісами.

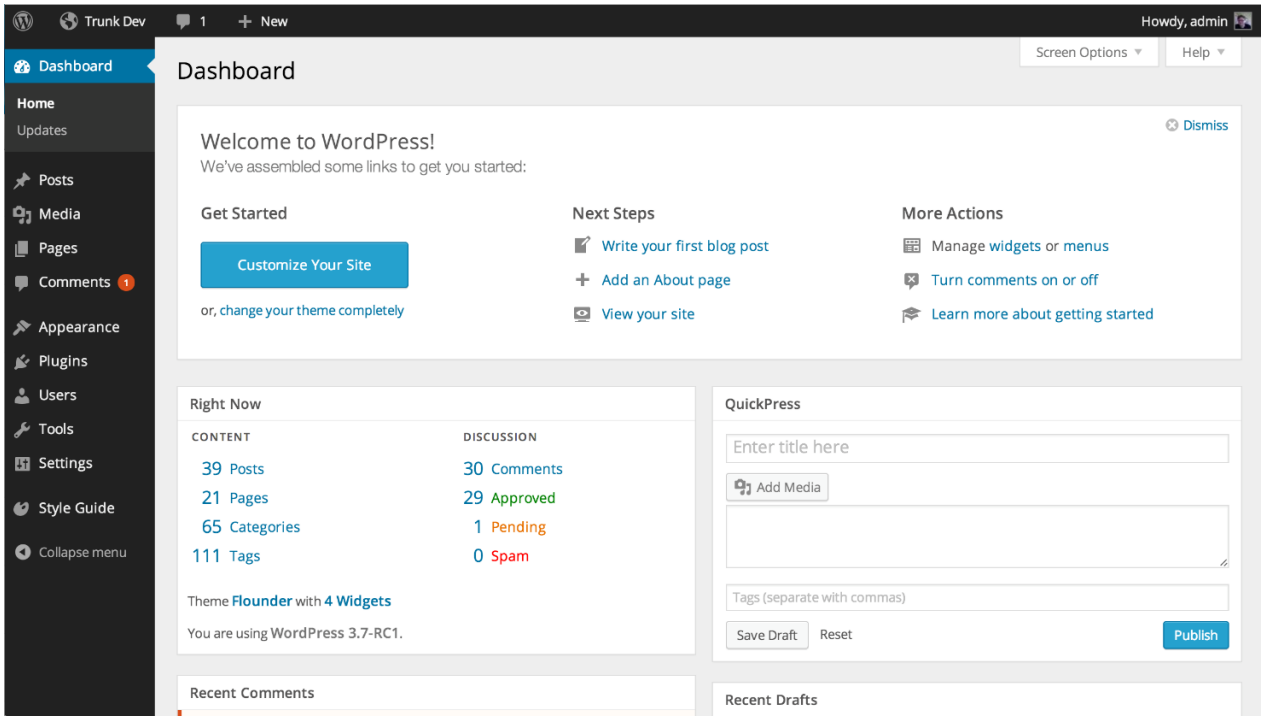

Рисунок 1.1 – Система керування контентом WordPress

Одна з головних сильних сторін WordPress – це її функціональність. Базова версія системи дозволяє створювати сторінки й записи, додавати текст, зображення, відео, налаштовувати меню та віджети. Завдяки плагінам, таким як WooCommerce, Yoast SEO чи Elementor, користувачі можуть розширити можливості до повноцінного інтернет-магазину, оптимізації для пошукових систем чи візуального редагування через drag-and-drop. Наприклад, WooCommerce додає кошик, оплату та управління товарами, що робить WordPress популярним серед малого бізнесу. Проте ця модульність має зворотний бік: встановлення великої кількості плагінів може ускладнити структуру сайту та потребує додаткового часу на налаштування. WordPress пропонує значно більше, але часто це надлишково для простих проєктів.

Простота використання WordPress залежить від рівня підготовки користувача. Для новачків адмін-панель із редактором Gutenberg (введеним у 2018 році) є досить інтуїтивною: блочна структура дозволяє легко додавати контент, змінювати його порядок чи стилі. Наявність тисяч безкоштовних тем спрощує старт: достатньо обрати дизайн і почати наповнення. Однак для тих, хто не знайомий із веброзробкою, освоєння плагінів чи вирішення

конфліктів між ними може бути складним. Наприклад, встановлення плагіна кешування WP Super Cache потребує базового розуміння оптимізації, а помилки в налаштуваннях можуть призвести до збоїв.

Продуктивність WordPress часто ставлять під сумнів через її залежність від плагінів та БД. У чистій базовій версії платформа працює досить швидко, але використання додаткових модулів (Elementor або Jetpack), призводить до збільшення часу завантаження сторінок. Згідно з тестами, середній сайт на WordPress із 5-7 плагінами завантажується за 2-3 секунди, що може бути повільно для сучасних стандартів. Крім того, MySQL потребує серверних ресурсів, і на дешевих хостингах продуктивність падає під навантаженням.

Безпека WordPress має як переваги, так і недоліки. З одного боку, регулярні оновлення ядра та захисні плагіни на кшталт Wordfence допомагають убезпечити сайти від атак. З іншого висока популярність платформи робить її привабливою ціллю для хакерів, особливо через уразливості в застарілих плагінах.

Гнучкість WordPress є її сильною стороною. Завдяки відкритому коду й підтримці PHP розробники можуть створювати власні теми, плагіни чи навіть змінювати ядро системи. Наприклад, редагування файлу functions.php дозволяє додати кастомний функціонал (власні віджети). Проте ця гнучкість вимагає знань програмування, і для новачків вона залишається недоступною без сторонньої допомоги.

Вимоги до інфраструктури WordPress включають хостинг із підтримкою PHP версії 7.4 або вище і MySQL, тобто для роботи потрібен сервер, а локальне тестування вимагає інструментів таких як XAMPP або інші аналоги.

WordPress – потужна та універсальна CMS, ідеальна для складних проєктів із великою аудиторією, але її складність, ресурсомісткість та залежність від інфраструктури роблять її надмірною для простих сайтів. Тому кастомна CMS орієнтована на мінімалізм та локальне зберігання, пропонує альтернативу, усуваючи ці недоліки для новачків і малих проєктів.

1.2.2 Система керування контентом Joomla

Однією з провідних CMS з відкритим кодом є Joomla (рисунок 1.2), створена у 2005 році на основі Mambo. Розроблена з використанням PHP та MySQL, вона поєднує в собі простоту WordPress та функціональність Drupal. Joomla орієнтована на створення корпоративних сайтів, порталів, спільнот та вебдодатків, пропонуючи модульну архітектуру та розширені можливості управління контентом. Її популярність пояснюється балансом між гнучкістю для розробників і доступністю для користувачів із базовими технічними навичками, а також активною спільнотою, яка підтримує тисячі розширень і шаблонів [2].

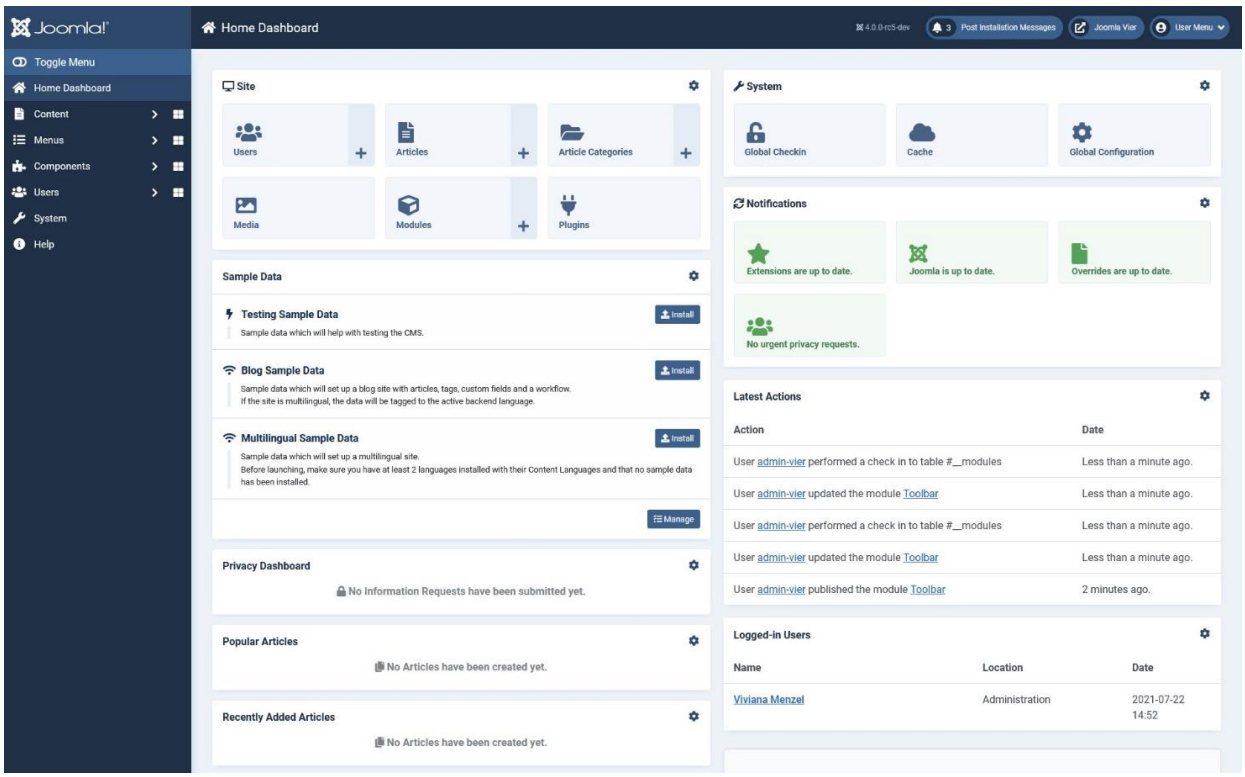

Рисунок 1.2 – Система керування контентом Joomla

Функціональність Joomla охоплює широкий спектр задач. Базова версія підтримує створення сторінок, статей, меню, категорій, а також управління користувачами з різними рівнями доступу. Через розширення, такі як K2 для контенту чи VirtueMart для e-commerce, система може бути адаптована до складних проєктів, таких як інтернет-магазини чи форуми. Наприклад,

VirtueMart додає каталог товарів та систему оплати, що робить Joomla конкурентом WordPress із WooCommerce. Проте Joomla пропонує широкий функціонал, що може бути надлишковим для простих сайтів типу блогів чи лендінгів. Хоч адмін-панель Joomla і має модульний дизайн для швидкого управління контентом та структурою, проте навчитися користуватися нею складніше, ніж у випадку з WordPress. Наприклад, створення меню чи категорій потребує розуміння ієрархії, а встановлення розширень може заплутати новачків через різноманітність опцій. Joomla вимагає більше часу на навчання, особливо для тих, хто не має досвіду в вебадмініструванні, тому мінімалістичний та зрозумілий дизайн має значну перевагу.

Продуктивність Joomla залежить від конфігурації та хостингу. У базовому вигляді система швидша за WordPress із великою кількістю плагінів, але додавання модулів, наприклад, VirtueMart збільшує навантаження на сервер. Такі великі системи значно поступаються легким flat-file CMS.

Безпека Joomla підтримується регулярними оновленнями та вбудованими функціями, такі як двофакторна автентифікація. Проте, як і WordPress, вона вразлива через популярність, Joomla також часто атакується через застарілі розширення. Особиста CMS із її простою архітектурою та відсутністю зовнішніх залежностей на початковому етапі, може бути менш схильною до таких ризиків, хоча й не підтримує складні функції безпеки, як Joomla.

Гнучкість Joomla забезпечується її модульною структурою та підтримкою PHP. Розробники можуть створювати власні компоненти, модулі чи шаблони, редагуючи код через вбудований редактор або файли на сервері. Наприклад, додавання кастомного модуля для відображення новин потребує базового знання PHP та Joomla API.

Joomla – потужна CMS для середніх та великих сайтів із розвиненим функціоналом, але її складність та залежність від інфраструктури роблять її менш оптимальною для простих задач.

1.2.3 Система керування контентом Drupal

Як CMS із відкритим кодом, Drupal (рисунок 1.3) позиціонується як платформа для масштабних та складних вебпроєктів з великим навантаженням. Запущена в 2001 році, вона базується на PHP і використовує MySQL або інші БД, як PostgreSQL для зберігання даних. Її аудиторія – це професійні розробники та організації, які потребують гнучкості та безпеки. Drupal підходить для створення корпоративних порталів, урядових сайтів, освітніх платформ та складних вебдодатків, завдяки модульній архітектурі та акценту на кастомізацію [3].

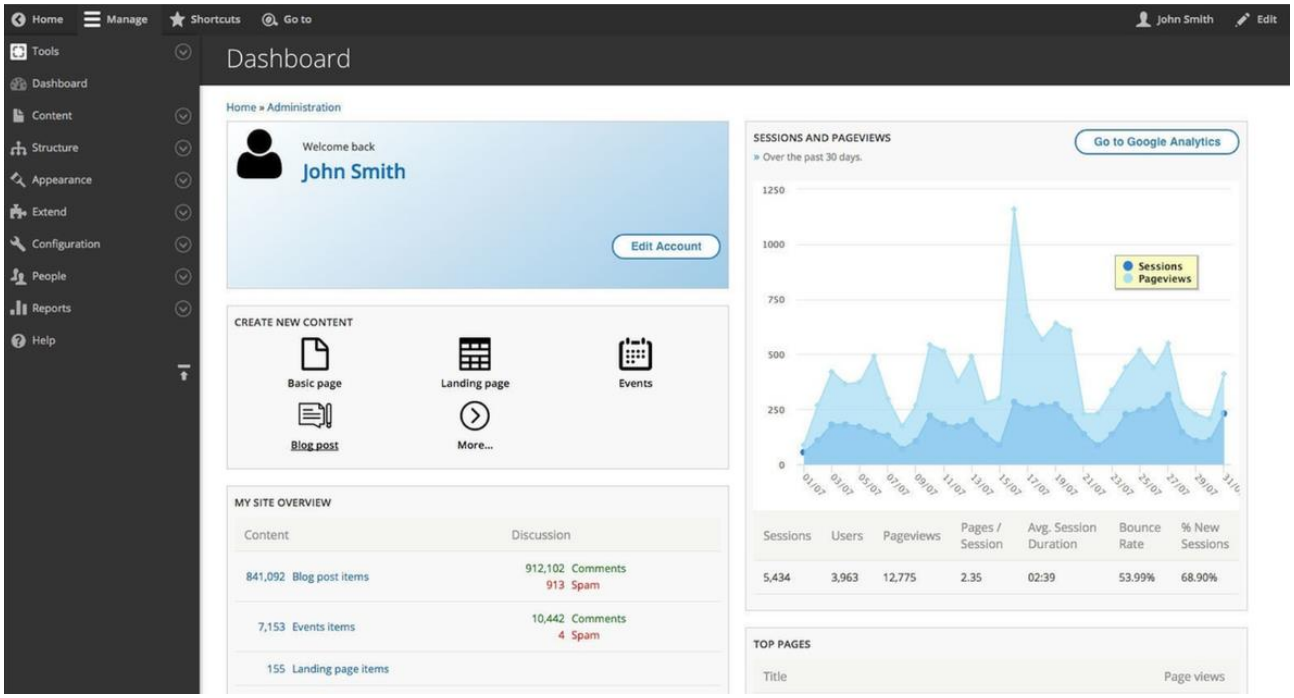

Рисунок 1.3 – Система керування контентом Drupal

Функціональність Drupal є однією з найширших серед CMS. Базова версія підтримує створення контенту, управління користувачами, категоризацію та багатомовність. Модулі, такі як Views для створення кастомних сторінок чи Commerce для e-commerce, дозволяють адаптувати систему до специфічних задач, наприклад, платіжних систем чи інтерактивних дашбордів. Drupal не проста в використанні: адмін-панель нових версій стала інтуїтивнішою завдяки редактору Layout Builder, але все

одно вимагає технічних знань. Наприклад, створення сторінки через Views потребує розуміння фільтрів та зв’язків, що складно для новачків.

Продуктивність Drupal вища за WordPress та Joomla у базовій конфігурації завдяки оптимізованому ядру. Але додавання модулів таких як Commerce збільшує вимоги до сервера. Drupal ефективна для високонавантажених проєктів, але потребує потужного хостингу.

Безпека – одна з найсильніших сторін Drupal. Її ядро вважається найзахищенішим серед популярних CMS: команда безпеки випускає оновлення щомісяця, а вразливості модулів рідші, ніж у WordPress. Проте безпека залежить від правильного налаштування, що складно для нетехнічних користувачів.

Гнучкість Drupal неперевершена для професійних розробників. Вона дозволяє створювати кастомні типи контенту, інтеграції з API чи навіть власні модулі через PHP та Drupal API. Наприклад, модуль Views дає змогу будувати складні запити без коду, але вимагає базових знань.

Вимоги до інфраструктури Drupal найвищі серед масштабних CMS. Потрібен хостинг із PHP 8.0 або вище, MySQL або PostgreSQL і бажано SSD для швидкості. Це робить її дорожчою в розгортанні в порівнянні з особистою CMS, яка буде налаштована для потреб власного сайту, що спрощує старт для користувачів із мінімальними ресурсами. Drupal ідеальна для складних проєктів із високими вимогами до безпеки та гнучкості, але її складність та ресурсомісткість роблять її непрактичною для простих сайтів.

1.2.4 Система керування контентом OpenCart

OpenCart (рисунок 1.4) – CMS на PHP та MySQL, створена спеціально для розробки та підтримки онлайн-магазинів. OpenCart орієнтована на малий та середній бізнес, пропонуючи інструменти для управління каталогами товарів, замовленнями, оплатами та доставкою [4]. Її популярність зумовлена простотою розгортання та спеціалізацією на комерційних проєктах, хоча вона менш універсальна порівняно з WordPress чи Joomla.

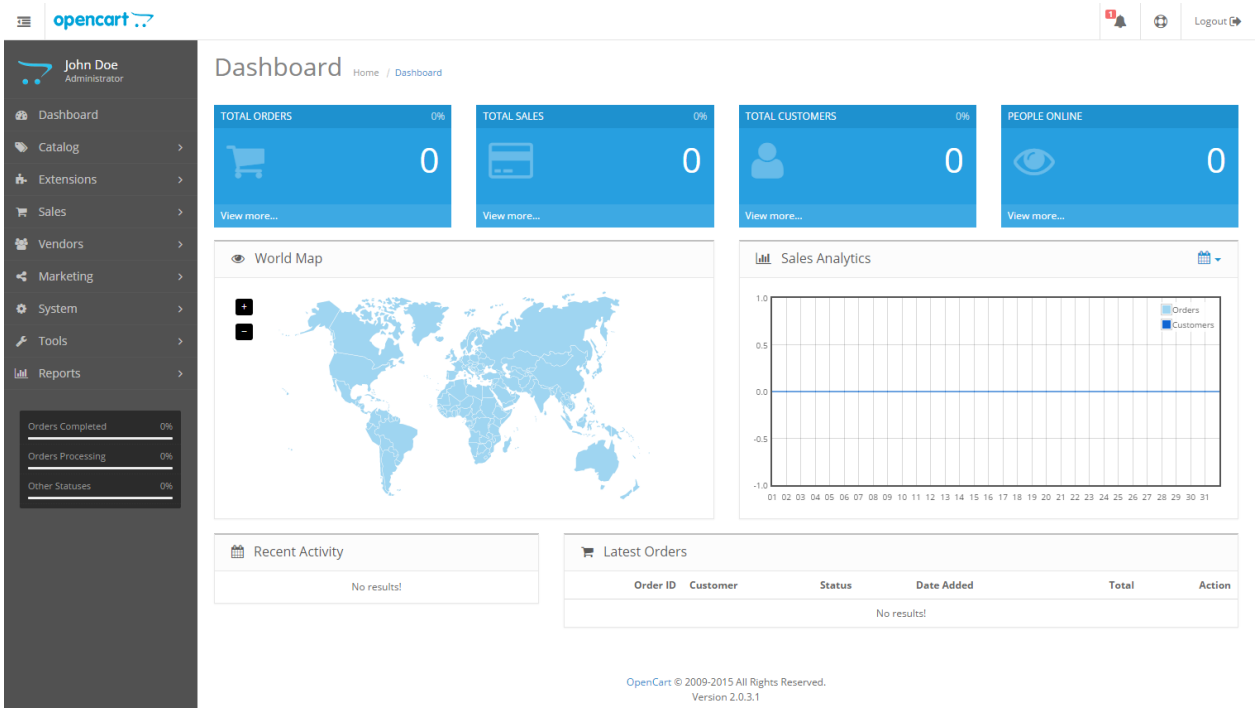

Рисунок 1.4 – Система керування контентом OpenCart

Функціональність OpenCart зосереджена на e-commerce. Базова версія підтримує додавання товарів, категорій, знижок, а також інтеграцію платіжних систем та методів доставки. Розширення, такі як SEO Module, додають можливості оптимізації. Простота використання вища, ніж у Drupal: адмін-панель інтуїтивна для створення магазину, але складна для новачків без досвіду в e-commerce. Продуктивність залежить від хостингу, але великі каталоги можуть сповільнювати роботу.

OpenCart відносна безпечна хоча оновлення рідші, ніж у Drupal, і вразливості в розширеннях трапляються. Гнучкість можлива через кастомізацію за допомогою PHP та модулів, але менш розвинена, ніж у Joomla. Інфраструктура подібна до інших масштабних CMS.

Масштабні CMS, як от WordPress, Joomla, Drupal, OpenCart пропонують широкий функціонал, але мають спільні недоліки: складність для новачків, залежність від бази даних та хостингу, а також потенційні проблеми з продуктивністю та безпекою через численні модулі. WordPress універсальна, але надмірна для простих сайтів; Joomla гнучка, але складна в освоєнні; Drupal потужна, але орієнтований на професіоналів; OpenCart

ефективна для e-commerce, але обмежена для інших типів проєктів. CMS, яка розробляється на PHP та React усуне ці недоліки, пропонуючи мінімалістичний інтерфейс та базові функції без потреби в сервері чи БД, що робить її ідеальною для новачків та малих проєктів із потенціалом розширення.

1.2.5 Система керування контентом Grav

На відміну від класичних CMS Grav (рисунок 1.5) зберігає дані не в БД, а у звичайних файлах таких як Markdown або YAML, тому що використовує підхід flat-file. Вона займає нішу легких та швидких систем, популярних серед розробників і дизайнерів для створення блогів, портфоліо та лендінгів. Її популярність зростає завдяки простоті й продуктивності. Завдяки відсутності необхідності у складних базах даних Grav ідеально підходить для проєктів із невеликою кількістю контенту або для тих, хто цінує мінімалістичний підхід. Ця система підтримує модульну архітектуру через плагіни та теми, а також пропонує адмін-панель для зручного управління контентом, що робить її конкурентом традиційних CMS у сегменті малих проєктів [6].

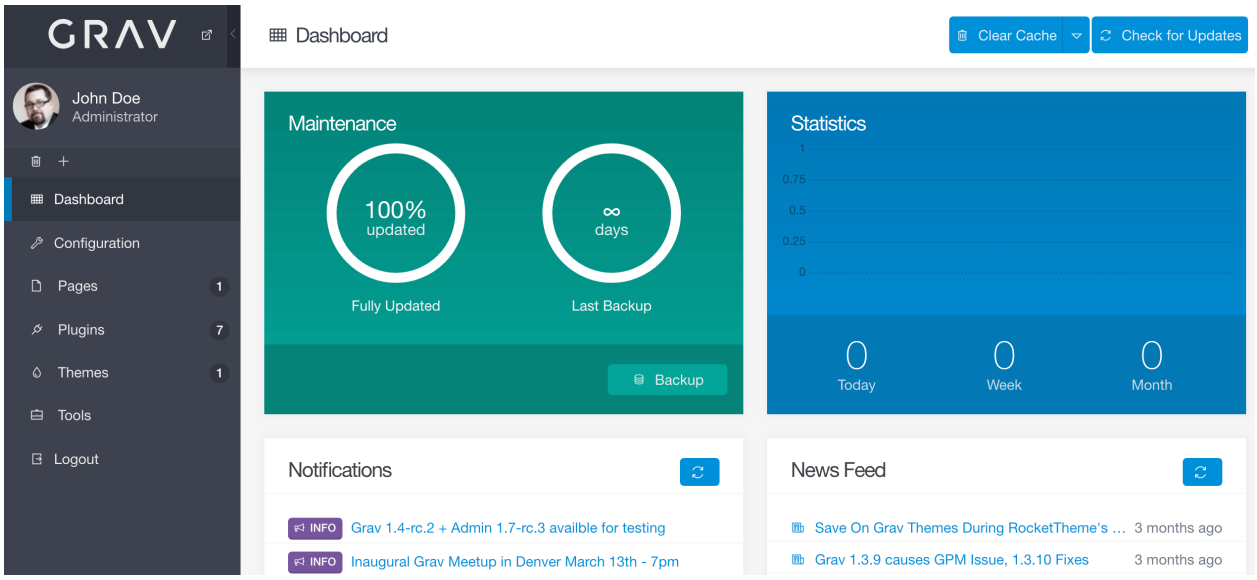

Рисунок 1.5 – Система керування контентом Grav

Функціональність Grav покриває основні потреби вебсайтів. Вона дозволяє створювати сторінки, додавати текст, зображення, метатеги, а також підтримує структуру блогу та багатомовність. Плагіни, такі як Admin для графічного інтерфейсу або Form, додають редагування та створення форм на основі браузера. На відміну від великомасштабних CMS Grav не підтримує складні функції, такі як e-commerce, але вона ідеально підходить для статичних сайтів. Зручність використання залежить від підходу: без панелі адміністратора потрібно знати Markdown та структуру файлів, але з плагіном Admin інтерфейс стає інтуїтивно зрозумілим, хоча й менш зручним для користувача, ніж у WordPress.

Продуктивність Grav її сильна сторона. Відсутність бази даних та використання файлів забезпечують швидке завантаження, що робить Grav ефективною для малих проєктів із низьким навантаженням. Безпека Grav вища, ніж у масштабних CMS, через мінімальну кількість залежностей, але вразливості в плагінах трапляються. Гнучкість забезпечується через PHP та плагіни: розробники можуть створювати кастомні теми чи скрипти, але це вимагає знань.

1.2.6 Система керування контентом Kirby

Запущена в 2009 році Kirby (рисунок 1.6) є легкою CMS з закритим кодом, яка використовує PHP та зберігає контент у текстових файлах формату Markdown або TXT. Вона популярна серед дизайнерів та розробників для створення портфоліо, блогів та невеликих корпоративних сайтів завдяки гнучкості та мінімалістичному підходу [7]. Її цінують за простий та інтуїтивно зрозумілий інтерфейс Kirby Panel, який забезпечує зручне редагування контенту. Відсутність бази даних та фокус на кастомізацію роблять Kirby сильним конкурентом у сегменті flat-file CMS, хоча вона менш відома, ніж Grav.

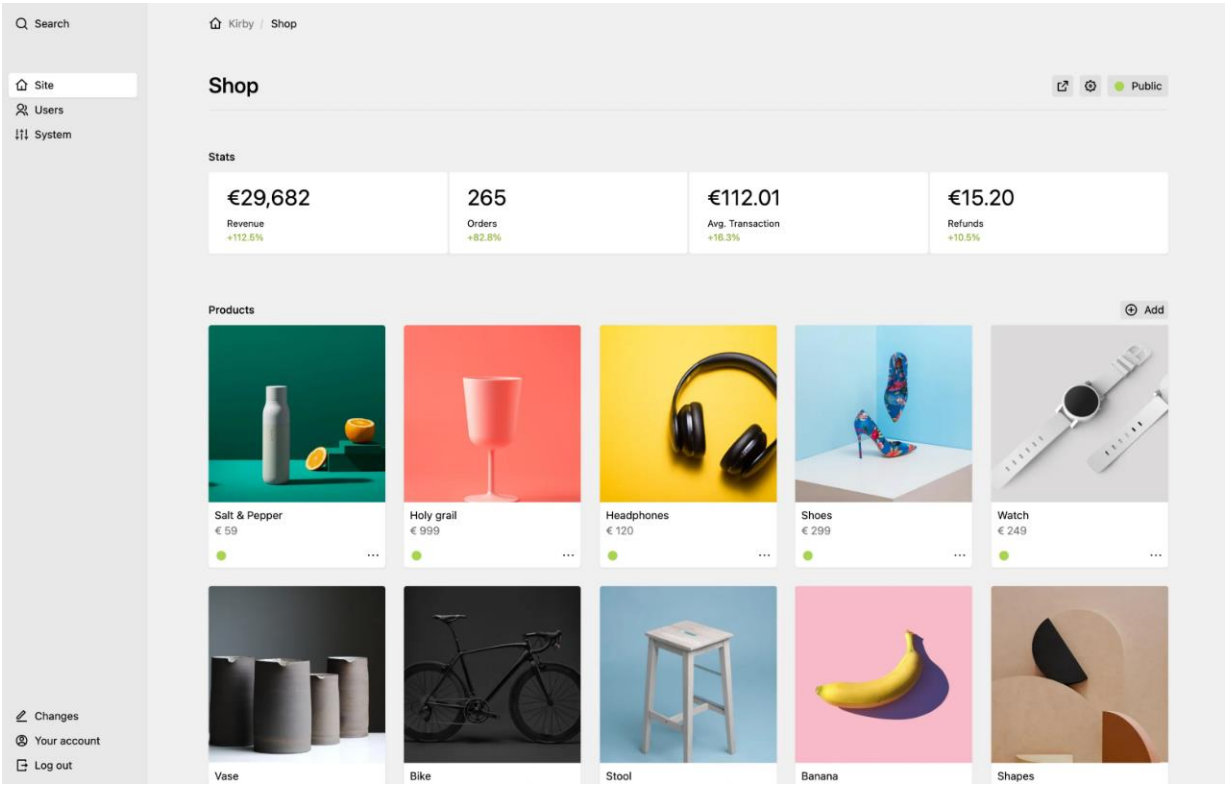

Рисунок 1.6 – Система керування контентом Kirby

Функціональність Kirby охоплює базові потреби вебсайтів: створення сторінок, редагування тексту, зображень, а також підтримка блогових структур та галерей. Плагіни, як наприклад, Kirby Editor додають розширені можливості редагування, а модульна система дозволяє створювати кастомні типи контенту. Простота використання висока завдяки Kirby Panel, яка пропонує інтуїтивний інтерфейс для новачків, хоча базове налаштування наприклад, структура файлів потребує мінімальних знань та розуміння.

Через її легкість Kirby є дуже продуктивною в роботі. Безпека Kirby висока через мінімальну кількість залежностей та закритий код; вразливості рідкісні, але підтримка залежить від платної ліцензії. Гнучкість забезпечується через PHP та текстові файли: розробники можуть створювати кастомні шаблони чи плагіни, але це складніше для новачків. Інфраструктура базова, потрібен лише PHP, тому її легко запустити локально.

Kirby пропонує швидкість та простоту, усуваючи потребу в БД. Її інтуїтивна Kirby Panel робить редагування доступним, але платна ліцензія та необхідність базового розуміння файлової структури відлякують новачків.

1.2.7 Система керування контентом Pico

Pico (рисунок 1.7) є дуже легкою flat-file CMS із відкритим кодом, запущена в 2012 році, працює на PHP та використовує Markdown-файли для зберігання контенту. Вона призначена для мінімалістичних статичних сайтів, таких як блоги чи портфоліо. Pico не пропонує графічної адмін-панелі, а редагування здійснюється через текстовий редактор, що робить її популярною серед розробників, які цінують простоту й автономність, хоча це може бути її недоліком. Крім того, Pico має мінімалістичний дизайн і підтримує розширення через плагіни, що дозволяє додавати необхідні функції без ускладнення системи. Завдяки своїй простоті й гнучкості, Pico часто обирають для проєктів, де важлива легкість та швидкість роботи, а також повний контроль над контентом без залежності від складних інтерфейсів [8].

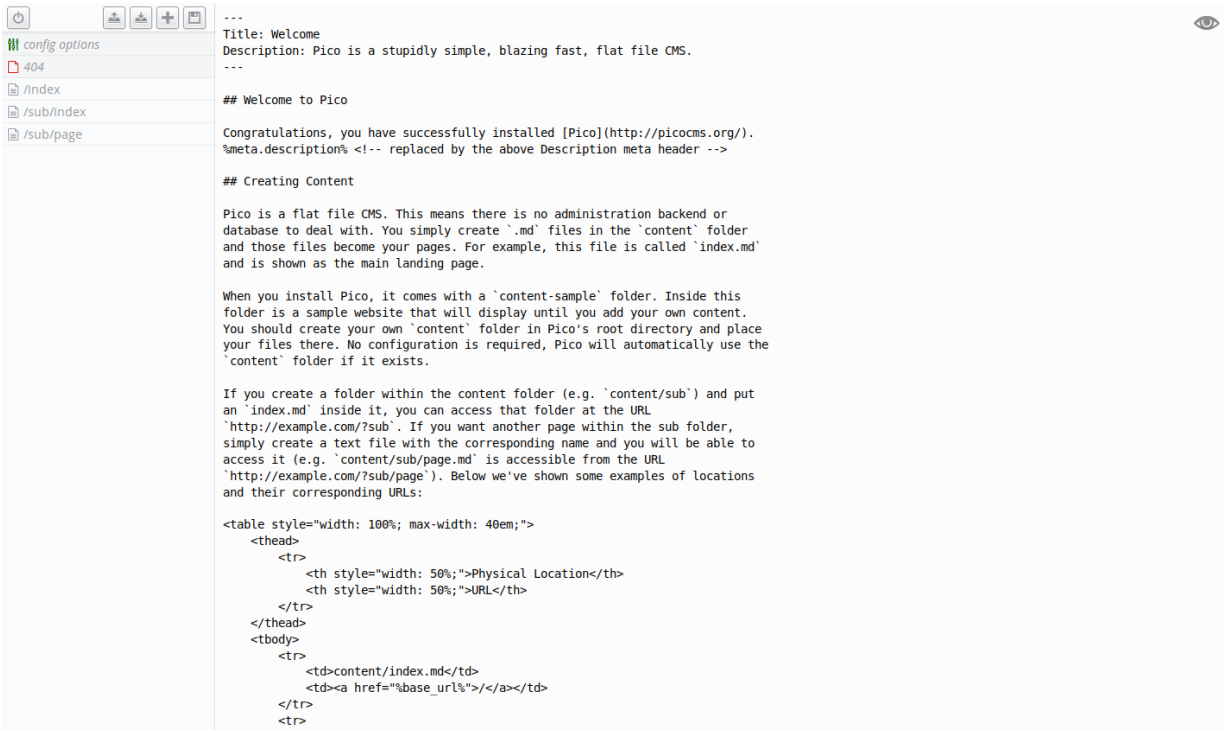

Рисунок 1.7 – Система керування контентом Pico

Функціональність Pico обмежена: створення сторінок, текст, зображення через Markdown, плагіни додають базові функції. Але її простота

низька для новачків через відсутність інтерфейсу, проте через це в цій системі висока продуктивність. Pico виграє в швидкості та автономності, але її текстовий підхід ускладнює використання новачками.

1.3 Роль CMS у сучасній веброзробці

У світі, де цифрові технології розвиваються стрімкими темпами, а присутність в Інтернеті є обов'язковою для бізнесу, організацій та приватних осіб, CMS відіграють ключову роль у веброзробці. Вони дозволяють користувачам без глибоких знань програмування створювати функціональні сайти, що відповідають сучасним стандартам, і водночас надають розробникам інструменти для реалізації складних проєктів. Розуміння ролі CMS у цьому процесі є важливим для оцінки їхнього впливу та визначення напрямків удосконалення, зокрема створення простих систем.

Однією з основних функцій CMS у сучасній веброзробці є спрощення роботи з контентом. Завдяки інтуїтивним інтерфейсам користувачі можуть додавати тексти, зображення, відео чи посилання, не звертаючись до коду. Це особливо важливо в умовах зростання попиту на швидке оновлення інформації: бізнеси потребують регулярно публікувати новини, блогери – ділитися новими постами, а магазини – оновлювати каталоги товарів. Наприклад, такі системи, як WordPress, дозволяють за лічені хвилини створити сторінку чи пост, що робить їх незамінними для динамічних вебсайтів.

CMS також відіграють важливу роль у підвищенні ефективності розробки. Завдяки модульній структурі та готовим шаблонам вони скорочують час, необхідний для запуску сайту, порівняно з розробкою з нуля. Наприклад, OpenCart дозволяє швидко розгорнути інтернет-магазин із базовими функціями кошика та оплати, тоді як Grav чи Kirby пропонують легкі рішення для статичних сайтів. Однак ця ефективність має свою ціну: популярні CMS часто перевантажені функціями, які не завжди потрібні конкретному користувачу, що може призводити до зниження швидкодії чи

ускладнення освоєння системи. Саме тут з’являється тренд на простоту, який набуває популярності в сучасній веброзробці: дедалі більше користувачів та розробників шукають легкі альтернативи, які поєднують базовий функціонал із мінімальними вимогами до ресурсів.

Безпека є ще одним аспектом, де роль CMS у веброзробці має двоїстий характер. З одного боку, розвинені системи, такі як WordPress, пропонують регулярні оновлення та плагіни для захисту від атак. З іншого боку, їхня популярність робить їх привабливими цілями для хакерів, а залежність від сторонніх модулів може створювати вразливості. Легші CMS, такі як Pico чи Kirby, завдяки своїй простоті та відсутності складних залежностей, часто виявляються безпечнішими, але менш універсальними. Цей баланс між функціональністю та безпекою підкреслює потребу в адаптованих рішеннях, які б відповідали конкретним вимогам проєктів.

Роль CMS у сучасній веброзробці можна узагальнити як забезпечення доступності, ефективності та гнучкості у створенні вебсайтів. Вони дозволяють як новачкам, так і професіоналам швидко реалізовувати ідеї, адаптуватися до змін та відповідати вимогам аудиторії. Проте зростання попиту на персоналізацію та оптимізацію призводить до того, що універсальні CMS не завжди відповідають потребам усіх користувачів. Це відкриває простір для розробки простих систем, які фокусуються на ключових функціях і легкості використання, що є основою даної роботи.

1.4 Постановка задачі кваліфікаційної роботи

Аналіз предметної області показав, що системи керування контентом (CMS) є ключовим інструментом у сучасній веброзробці, однак існуючі рішення не завжди відповідають потребам користувачів, які шукають простоту, швидкість та мінімалізм. Популярні CMS, такі як WordPress чи Joomla, пропонують широкий функціонал, але їхня складність та ресурсомісткість ускладнюють використання для новачків або невеликих проєктів. Легші альтернативи, як от Grav чи Pico, хоч і наближаються до ідеї

простоти, все ще потребують певних технічних знань або мають обмеження в адаптивності. У цьому контексті виникає потреба в розробці власної CMS, базовим набором функцій та легкістю розгортання, відповідаючи запитам цільової аудиторії – користувачів із мінімальним досвідом у веброзробці.

Метою даної роботи є створення простої системи керування контентом, яка дозволить користувачам ефективно управляти вмістом вебсайтів без надлишкової складності. Основна задача полягає в розробці системи, що забезпечить базовий функціонал для створення та редагування вебконтенту, зберігаючи при цьому простоту використання й мінімальні вимоги до ресурсів [5]. Така CMS має бути доступною для новачків, власників малого бізнесу чи блогерів, які прагнуть швидко запустити сайт та підтримувати його без залучення спеціалістів.

2 АНАЛІЗ ТЕХНОЛОГІЙ, ЩО ВИКОРИСТОВУЮТЬСЯ ПРИ РОЗРОБЦІ ВЕБЗАСТОСУНКУ

2.1 Серверна мова програмування PHP

PHP (Hypertext Preprocessor) – це серверна мова програмування з відкритим кодом, яка залишається однією з найпопулярніших для веброзробки. Мова використовується для створення динамічних вебдодатків, обробки запитів, роботи з даними та інтеграції з різними форматами такими як JSON. PHP є простою та зручною мовою, з широкою підтримкою [9].

В PHP є широкий набір можливостей для швидкої серверної розробки. Вона забезпечує динамічну генерацію HTML, обробку HTTP-запитів маніпуляції з файловою системою та парсинг даних у форматах JSON та XML. Також є підтримка об’єктно-орієнтованого програмування, модульної структури та сучасних функцій таких як строга типізація, стрілкові функції, які додали в останніх версіях, що значно підвищує читабельність та чистоту коду. Простота синтаксису, подібного до C, робить PHP доступним для розробників із базовим досвідом, що важливо для швидкої реалізації вебзастосунків.

Однією з ключових переваг PHP є її універсальність [10]. Практично всі вебсервери, як наприклад, Apache або Nginx та хостинг-провайдери сумісні з PHP, що спрощує розгортання застосунків. Локальне тестування можливе через вбудований сервер або інструменти типу XAMPP або MAMP, вони не потребують складних налаштувань. Актуальна версія PHP 8 включає JIT-компіляцію, що покращує продуктивність для обчислювальних задач, хоча для типових вебзадач, як наприклад, обробка форм, збереження файлів приріст менш помітний. Завдяки низьким вимогам до ресурсів, мова PHP ефективна для проєктів із обмеженим бюджетом, де не потрібні складні серверні середовища, як для Node.js чи Python.

Стандартний PHP можна порівняти з альтернативними технологіями,

такими як Node.js, Django та побудований на мові PHP фреймворк Laravel. Laravel спрощує розробку великих застосунків завдяки зручним інструментам, такі як маршрутизація, ORM Eloquent, та MVC-архітектура, але його залежності та складність налаштування роблять його надмірним для простих проєктів, тому в таких випадках доречніше використати стандартний PHP. Наприклад, збереження даних у JSON за допомогою нативного PHP потребує кількох рядків коду, тоді як у Laravel це вимагало б конфігурації моделі, контролера та роутингу. Node.js, побудований на JavaScript та асинхронній моделі, ефективний для застосунків реального часу наприклад, чати чи потокові сервіси, але його екосистема складніша для початківців. Крім того, Node.js менш поширений на бюджетних хостингах, що обмежує доступність порівняно з PHP. Python із фреймворком Django пропонує чистий синтаксис та потужні інструменти, але потребує додаткових серверних налаштувань, що ускладнює розгортання для невеликих проєктів.

Недоліки PHP включають історичні проблеми, такі як відсутність єдиної структури синтаксису та слабку типізацію в старих версіях, що може ускладнювати підтримку коду в масштабних проєктах. Проте сучасні версії усувають ці недоліки завдяки строгій типізації та покращеній продуктивності. Ще один мінус – нижча ефективність у високонавантажених проєктах, порівняно з Node.js, але для невеликих вебзастосунків із базовими операціями це не критично. У порівнянні з Python, PHP виграє в простоті розгортання, оскільки не потребує додаткових серверних компонентів.

2.2 Бібліотека React

React є бібліотекою JavaScript із відкритим кодом, була розроблена компанією Facebook у 2013 році, призначена для створення інтерактивних інтерфейсів користувача вебзастосунків. React залишається однією з найпопулярніших фронтенд-технологій [11], вона застосовується для побудови динамічних та компонентно-орієнтованих інтерфейсів, що робить її

ефективною для створення вебзастосунків.

Основною особливістю React є компонентний підхід (рисунок 2.1), який дозволяє розбивати інтерфейс на незалежні компоненти, які можливо повторно використовувати. Кожен компонент інкапсулює логіку, стилі та розмітку, що спрощує розробку та підтримку коду. React використовує JSX – це синтаксичне розширення JavaScript [12], яке поєднує HTML та програмну логіку, цей підхід значно полегшує створення динамічних елементів, таких як форми чи списки.

Наприклад, компонент для редагування тексту може поєднувати поле введення та логіку відправлення даних на сервер, що робить код модульним та зрозумілим. Ще одна ключова особливість – це віртуальний DOM, який оптимізує оновлення інтерфейсу, порівнюючи зміни в пам’яті перед їх застосуванням до реального DOM, що підвищує продуктивність вебзастосунків.

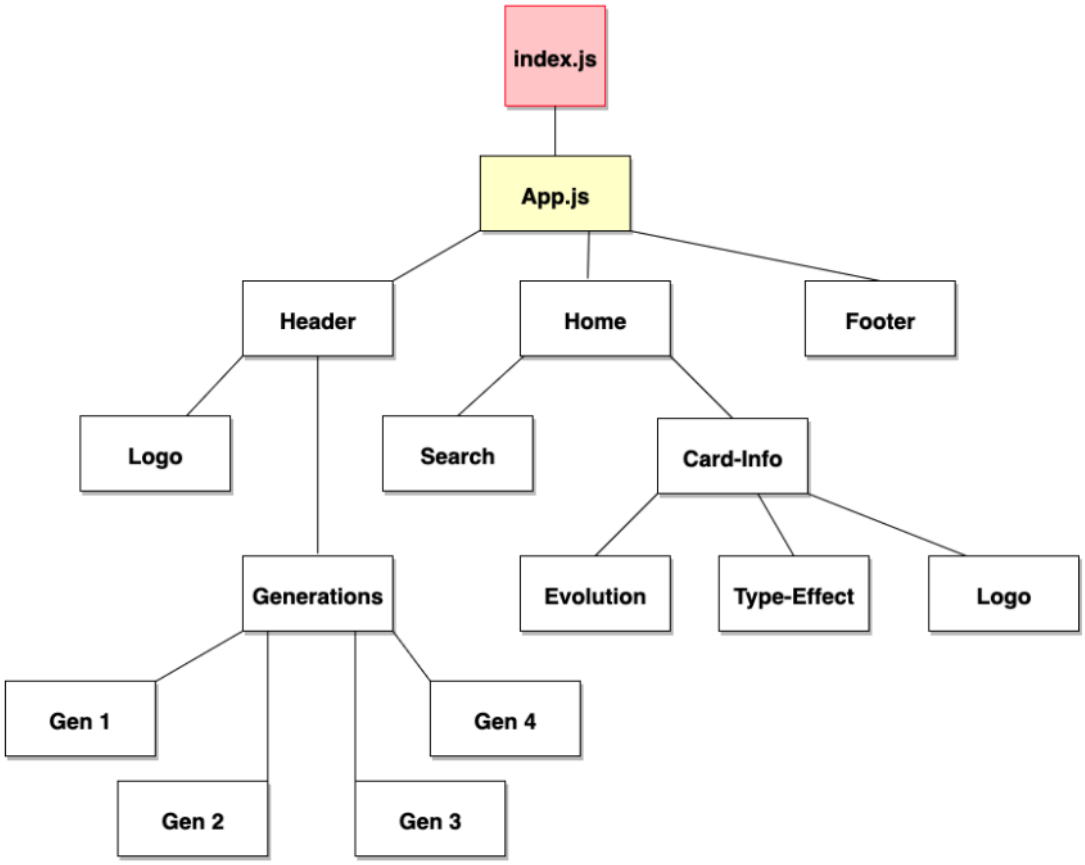

Рисунок 2.1 – Схема компонентів React

React пропонує низку переваг для фронтенд-розробки. По-перше, бібліотека має низький поріг входу порівняно з іншими фреймворками: розробники з базовими знаннями JavaScript можуть швидко розпочати роботу, використовуючи інструменти типу Create React App або Vite для ініціалізації проєкту. По-друге, екосистема React включає численні бібліотеки, як-от React Router для навігації чи Redux для управління станом, що дозволяє розширювати функціонал без надмірної складності. По-третє, широка спільнота та документація забезпечують доступ до навчальних ресурсів і готових рішень, що значно полегшує розробку.

Альтернативами React є такі інструменти, як Vue.js та Angular. Vue.js, створений у 2014 році, подібний до React за компонентним підходом та простотою, але має меншу екосистему та спільноту. Vue використовує шаблони замість JSX, що може бути зручнішим для новачків, але менш гнучким для складних інтерфейсів. Наприклад, створення динамічної форми в React із JSX дозволяє легко інтегрувати JavaScript-логіку, тоді як у Vue це потребує додаткових директив. Angular – фреймворк від Google, пропонує повноцінну MVC-архітектуру та вбудовані інструменти але складність та строга типізація роблять його надмірним для невеликих проєктів. Angular вимагає глибшого розуміння концепцій модулів та сервісів, що підвищує поріг входу.

Недоліки React також варто враховувати. Бібліотека не є повноцінним фреймворком, тому для таких функцій, як маршрутизація чи управління станом, потрібні додаткові бібліотеки. Наприклад, порівняно з Angular, який має вбудовані інструменти, React вимагає ручного підбору бібліотек, що може сповільнити розробку для новачків. Крім того, JSX може бути незвичним для розробників, які звикли до чистих HTML-шаблонів, хоча з часом він стає інтуїтивно зрозумілим. Порівняно з Vue.js, React є дещо складнішим через необхідність розуміння базових концепцій, таких як стан та хуки, але це компенсується його гнучкістю.

2.3 Бібліотека Axios

Axios є популярною JavaScript бібліотекою із відкритим кодом, яка призначена для виконання HTTP-запитів із клієнтської або серверної сторони вебзастосунків. Її популярність зумовлена завдяки простоті використання, підтримці асинхронних операцій і сумісності з браузерами та Node.js.

Основна функціональність Axios полягає в спрощенні виконання HTTP-запитів, таких як GET, POST, PUT та DELETE, із підтримкою асинхронного програмування через Promise та async/await. Бібліотека автоматично обробляє JSON-дані, що дозволяє легко надсилати та отримувати структуровані об’єкти без додаткового парсингу. Наприклад, відправлення даних форми на сервер або отримання списку ресурсів із API виконується кількома рядками коду з чітким синтаксисом.

Axios також пропонує вбудовані можливості, такі як перехоплення запитів та відповідей (interceptors), обробка помилок, підтримка заголовків. Ці функції роблять бібліотеку зручною для створення надійних та гнучких вебзастосунків.

Для порівняння Fetch API, вбудований у сучасні браузери, є легким та безкоштовним від залежностей рішенням для HTTP-запитів. Однак Fetch менш зручний через відсутність автоматичної обробки JSON, складнішу обробку помилок. jQuery AJAX, популярний у минулому, пропонує простий синтаксис, але залежить від бібліотеки jQuery, яка додає значний обсяг коду і рідко використовується в сучасних проєктах через неактуальність та малу підтримку. Крім того, jQuery AJAX менш адаптований до асинхронного програмування порівняно з Axios.

Недоліки Axios включають додаткову залежність, яка збільшує розмір проєкту порівняно з Fetch, що може бути критичним для дуже легких застосчунків. Проте ці недоліки компенсуються зручністю та розширеними можливостями, які Axios пропонує для складніших сценаріїв, таких як серійні запити чи обробка складних API.

2.4 UI бібліотека Ant Design

Ant Design (рисунок 2.2) – це бібліотека компонентів інтерфейсу користувача (UI) із відкритим кодом, розроблена компанією Ant Financial у 2015 році для створення сучасних та функціональних вебінтерфейсів. Написана на JavaScript із підтримкою React, вона пропонує набір готових компонентів для фронтенд-розробки.

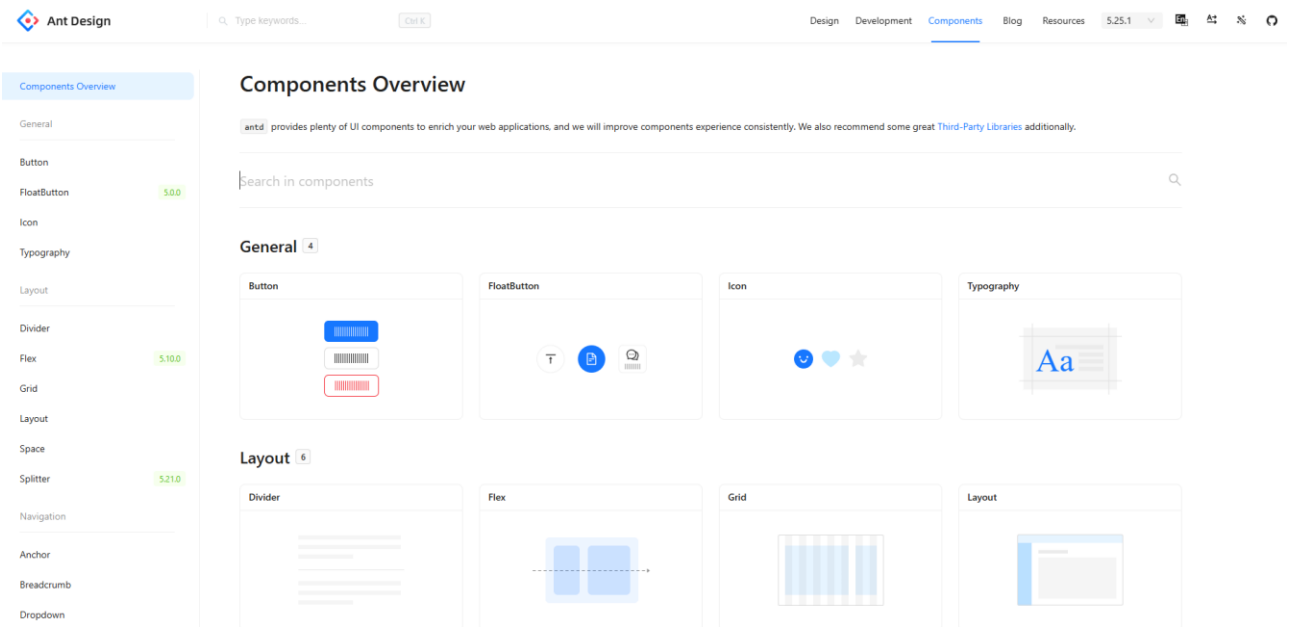

Рисунок 2.2 – Готові компоненти Ant Design

В цій бібліотеці є великий набір React-компонентів, які охоплюють широкий спектр елементів інтерфейсу: від базових (таких як кнопки, поля введення, таблиці) до складних (модальні вікна, меню, компоненти для роботи з даними). Компоненти Ant Design оптимізовані для React, використовуючи JSX для інтеграції з логікою застосунку, це дозволяє створювати модульні інтерфейси та інтерфейси, які повторно використовуються. Наприклад, компонент Form спрощує створення форм із валідацією, а Table підтримує сортування та пагінацію даних. Ant Design також пропонує адаптивний дизайн, підтримку світлої та темної теми та вбудовані інструменти для стилізації через CSS-in-JS або Less, що забезпечує гнучкість у кастомізації.

Переваги Ant Design [13] роблять її привабливим вибором для фронтенд-розробки. По-перше, бібліотека пропонує комплексний набір компонентів, що покриває більшість потреб вебзастосунків, зменшуючи необхідність створення елементів із нуля. По-друге, antd має сучасний та професійний дизайн, який відповідає стандартам UX/UI, що важливо для створення інтерфейсів із високим рівнем досвіду користувача. По-третє, інтеграція з React є безшовною: компоненти antd розроблені спеціально для екосистеми React, що спрощує їх використання з хуками та управлінням стану. Також слід зазначити, що активна спільнота та регулярні оновлення забезпечують актуальність та підтримку.

Ant Design добре вирізняється на тлі альтернатив, зокрема Bootstrap та Material-UI. Bootstrap, одна з найстаріших UI-бібліотек, пропонує простий набір компонентів та адаптивну сітку, але її дизайн вважається застарілим. Наприклад, створення складної форми в Bootstrap вимагає більше стилізації, ніж в Ant Design, де валідація та стилі вбудовані. Material-UI, подібно до antd, розроблена для React і базується на принципах Material Design від Google, пропонуючи сучасний вигляд та гнучкість. Однак Material-UI має строгіший дизайн, що може обмежувати кастомізацію, тоді як antd пропонує більше варіантів тем та стилів. Крім того, Material-UI має дещо складніший API для початківців, наприклад, через використання styled-components. Інші бібліотеки, як-от Chakra UI чи Tailwind UI, менш універсальні або потребують додаткової роботи зі стилізацією, що робить antd кращим вибором.

2.5 Інструменти Gulp та Webpack

У сучасній веброзробці Gulp та Webpack, як інструменти з відкритим кодом, забезпечують автоматизацію та ефективну збірку застосунків. Gulp, створений у 2013 році, є потоковим таск-менеджером, тоді як Webpack, запущений у 2012 році, – це модульний бандлер, орієнтований на збирання JavaScript. Gulp та Webpack чудово підходять для автоматизації задач та

збірки проєкту.

Gulp базується на концепції потоків, що дозволяє обробляти файли через послідовність задач. Він використовує плагіни для таких операцій, як компіляція SCSS у CSS, мініфікація скриптів чи оптимізація зображень. Наприклад, Gulp може автоматично трансформувати SCSS у стиснений CSS та перезавантажувати браузер через BrowserSync, що прискорює розробку. Основна перевага Gulp – простота конфігурації та швидкість виконання завдяки потоковій обробці в пам’яті, що зменшує операції введення/виведення. Gulp є гнучким інструментом, який дозволяє розробникам створювати кастомні задачі через JavaScript, адаптуючи процес збірки до специфічних потреб проєкту.

Webpack, натомість, фокусується на модульній зборці, об’єднуючи JavaScript-модулі, стилі, зображення в єдиний бандл. Він використовує граф залежностей, аналізуючи імпорти, щоб оптимізувати ресурси. Webpack підтримує такі функції, як видалення невикористовуваного коду, розбиття бандла на частини та гаряча заміна модулів, що прискорює розробку. Webpack може зібрати React-компоненти, SCSS та зображення в один стиснений файл, оптимізуючи час завантаження. Завдяки плагінам, як-от Terser для мініфікації чи CSS Minimizer, Webpack забезпечує високу гнучкість та продуктивність, що робить його стандартом для складних фронтенд-проєктів.

Переваги Gulp та Webpack доповнюють одна одну в веброзробці. Gulp вирізняється швидкістю та простотою для задач, які не потребують складної модульної структури, таких як обробка стилів чи копіювання файлів. Його конфігурація через JavaScript є інтуїтивною для розробників, а велика екосистема плагінів окриває більшість потреб. Webpack незамінний для проєктів із великою кількістю залежностей, особливо в React-екосистемі, де потрібно обробляти численні модулі та оптимізувати бандл для продакшену. Webpack також має активну спільноту, регулярні оновлення та детальну документацію, що полегшує інтеграцію з сучасними інструментами.

Поєднання Gulp та Webpack дозволяє розділити задачі: Gulp для швидкої обробки розроблювального проєкту, а Webpack для збірки JavaScript та залежностей.

Ці інструменти також не позбавлені недоліків. Налаштування Gulp вимагає підбору плагінів та написання задач, що може сповільнити початок роботи для тих, хто не знайомий із його екосистемою, також слід перевіряти актуальність плагінів. Webpack, зі своїм складним файлом конфігурації, ще більш вимогливий: неправильне налаштування лоадерів чи плагінів може призвести до помилок зборки. Втім, ці труднощі виправдовуються їхньою гнучкістю та здатністю підлаштовуватись під потреби проєкту. У даному випадку Gulp спрощує роботу зі стилями та файлами, дозволяючи швидко вносити зміни та одразу бачити результат, а Webpack забезпечує оптимізований бандл, який зменшує навантаження на клієнтську сторону. Їхня велика екосистема, із тисячами плагінів та активною підтримкою через npm, додає впевненості в тому, що інструменти залишаться актуальними.

Gulp та Webpack формують ефективний робочий процес, де кожен інструмент виконує свою роль: Gulp прискорить розробку через автоматизацію, Webpack гарантує оптимізацію ресурсів для зібраного проєкту. Їхнє поєднання відображає баланс між швидкістю, контролем та якістю, що робить їх незамінними для створення сучасних вебзастосунків.

2.6 Система контролю версій

У сучасній веброзробці система контролю версій Git є ключовим інструментом. Розроблений у 2005 році Лінусом Торвальдсом, є однією з найпоширеніших систем контролю версій у сучасній розробці програмного забезпечення. Цей інструмент дозволяє ефективно управляти змінами в коді, забезпечуючи відстеження історії проєкту, організацію робочих процесів та можливість співпраці між розробниками. Git вирізняється своєю розподіленою архітектурою, що робить його універсальним для проєктів різного масштабу.

Головна функція Git – це фіксація змін у файлах шляхом створення комітів та внесення змін до проєкту в певний момент часу. Кожен створює опис змін, що дозволяє розробникам документувати етапи роботи, наприклад, додавання нового функціоналу та виправлення помилок. У веброзробці це особливо корисно, оскільки проєкти часто включають різноманітні файли – скрипти, стилі, розмітку, конфігурації. Git зберігає історію всіх змін, дозволяючи переглядати попередні версії, порівнювати їх або відновлювати втрачені фрагменти. Наприклад, якщо оновлення стилів спричинило неочікувані проблеми в інтерфейсі, розробник може повернутися до стабільної версії, використовуючи збережену історію комітів, що мінімізує ризик втрати прогресу.

Важливою особливістю Git є його гілкова модель, яка забезпечує ізольовану розробку паралельних версій проєкту. Гілки дозволяють створювати окремі робочі простори для експериментів, тестування нових функцій чи виправлення помилок, не впливаючи на основний код. У веброзробці гілки часто використовуються для реалізації окремих задач, наприклад, розробки нового компонента інтерфейсу чи оптимізації серверної логіки. Після завершення роботи гілка зливається з основною через процес злиття, що інтегрує зміни. Git автоматично обробляє більшість конфліктів, хоча в складних випадках може знадобитися ручне втручання. Така модель підтримує ітеративний підхід, дозволяючи розробникам працювати над кількома задачами одночасно, що є стандартом у створенні вебзастосунків.

Git інтегрується в робочий процес через набір команд, які дозволяють управляти проєктом із високою точністю. У веброзробці ці команди використовуються для управління різноманітними аспектами проєкту, від фронтенд-компонентів до серверних скриптів. Наприклад, порівняння змін між версіями допомагає ідентифікувати, які модифікації спричинили помилку в поведінці застосунку. Git також підтримує інструменти для аналізу історії, дозволяючи відстежувати, коли та які зміни були внесені, що полегшує діагностику проблем. Сумісність Git із середовищами розробки,

такими як Visual Studio Code, де вбудовані розширення спрощують доступ до команд, робить його зручним для повсякденного використання.

Використання Git не обходиться без труднощів. Наприклад, вирішення конфліктів при злитті гілок може вимагати ретельного аналізу змін, особливо в проєктах із великою кількістю файлів. Крім того, складніші функції, такі як переписування історії чи робота із віддаленими репозиторіями, потребують глибшого розуміння. Проте ці виклики компенсуються здатністю Git забезпечувати порядок та контроль у процесі розробки.

2.7 Середовище розробки Visual Studio Code

Visual Studio Code (VS Code) є одним із провідних середовищ розробки для роботи з різноманітними технологіями, підтримуючи розробників у створенні ефективних та структурованих проєктів.

Основна функціональність VS Code полягає в наданні інтуїтивного інтерфейсу для редагування коду з підтримкою широкого спектру мов програмування, включаючи JavaScript, PHP та CSS. Редактор пропонує вбудовані інструменти для автодоповнення, підсвітки синтаксису та перевірки помилок, що полегшує написання коду та знижує ймовірність синтаксичних неточностей. Наприклад, під час роботи з JavaScript-файлами VS Code автоматично пропонує підказки для методів та змінних, що прискорює розробку. Крім того, вбудований термінал дозволяє виконувати команди, такі як запуск локального сервера чи запити до системи контролю версій, безпосередньо в середовищі редактора, що оптимізує робочий процес.

Гнучкість VS Code забезпечується через розширення, які дозволяють адаптувати редактор до специфічних потреб веброзробки. Розширення для форматування коду, дебагінгу чи інтеграції з інструментами зборки додають функціональності, що відповідають вимогам проєкту. Наприклад, розширення для роботи з системами контролю версій спрощують створення комітів та перегляд змін.

3 ОПИС РЕАЛІЗАЦІЇ ВЕБЗАСТОСУНКУ

3.1 Архітектура вебзастосунку

У структурі створеного вебзастосунку лежить клієнт-серверна архітектура (рисунок 3.1). Такий підхід дозволяє організувати ефективну взаємодію між інтерфейсом користувача та логікою обробки даних, що є основою для реалізації функціональності системи керування контентом, яка орієнтована на простоту використання. Клієнтська частина відповідає за взаємодію з користувачем, а серверна – за обробку запитів та управління даними.

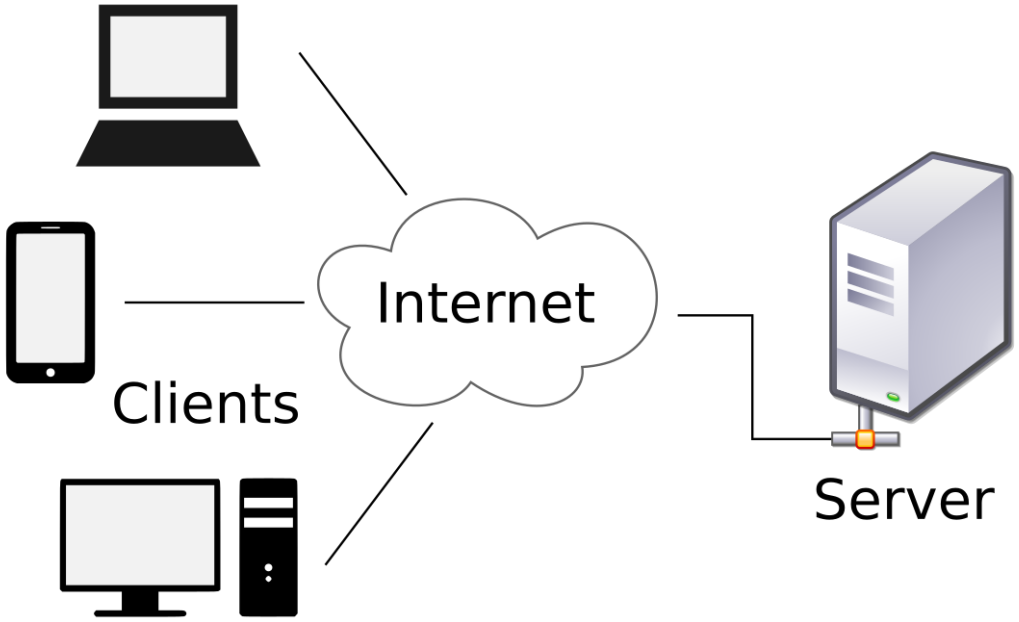

Рисунок 3.1 – Клієнт-серверна архітектура вебзастосунку

Клієнтська частина вебзастосунку забезпечує відображення інтерфейсу, який дозволяє користувачу створювати, редагувати та переглядати вміст сайту. За допомогою неї користувач має можливість вводити текст, змінювати зображення та мета-дані сайту, також реалізовано збереження результату та відновлення резервних копій. Клієнтська частина формує

запити до сервера для отримання або оновлення даних, забезпечуючи динамічне відображення вмісту без необхідності перезавантаження сторінки.

Серверна частина обробляє запити від клієнта, виконуючи операції з даними. Вона відповідає за прийом вхідних команд, їх валідацію та виконання, наприклад, збереження зміненого контенту. Серверна логіка забезпечує читання даних для відображення в інтерфейсі, а також створення резервних копій для захисту інформації.

Клієнт та сервер взаємодіють через асинхронні запити, що дозволяє обробляти дані без необхідності перезавантаження сторінки, що суттєво оптимізує швидкість роботи вебзастосунку, зменшує затримки у взаємодії з користувачем та забезпечує плавну та безперервну роботу інтерфейсу. Завдяки цьому клієнт отримує лише необхідні фрагменти інформації, що знижує навантаження як на мережу, так і на сервер.

3.2 Реалізація серверної логіки

Серверна частина вебзастосунку базується на flat-file підході, що забезпечує простоту розгортання та мінімальні вимоги до інфраструктури, усуваючи потребу в базі даних та дозволяючи працювати на будь-якому хостингу з підтримкою PHP. Вона обробляє запити від клієнтської частини через REST API, яке реалізовано з використанням HTTP-методів GET, POST, PUT, DELETE для чіткої взаємодії між клієнтом та сервером, забезпечуючи авторизацію користувачів, зберігання сторінок, завантаження зображень, а також отримання списків сторінок та резервних копій.

Для авторизації реалізовано механізм на основі сесій: користувач вводить логін та пароль, які передаються через POST-запит у форматі JSON; пароль перевіряється з налаштуваннями, збереженими у файлі settings.json, і при успішному збігу створюється сесійна змінна $_SESSION["auth"]. Вхід виконується через файл login.php, де також повертається статус автентифікації у JSON-форматі (лістинг 3.1).

Лістинг 3.1 – Реалізація авторизації

```
<?php
session_start();
$_POST = json_decode(file_get_contents("php://input"), true);
$password = $_POST["password"];
if ($password) {
  $settings = json_decode(file_get_contents('./settings.json'),
true);
  if ($password == $settings["password"]) {
    $_SESSION["auth"] = true;
    echo json_encode(array("auth" => true));
  } else { echo json_encode(array("auth" => false));}
} else {
  header("HTTP/1.0 400 Bad Request");
}
```

Для забезпечення безпечного завершення сеансу користувача у вебзастосунку реалізовано окремий файл logout.php, який відповідає за вихід із системи. Після виклику цього скрипту сесійна змінна $_SESSION["auth"] скидається, сесія завершується, і користувач втрачає доступ до захищених функцій застосунку (лістинг 3.2).

Лістинг 3.2 – Реалізація виходу користувача

```
<?php
session_start();
if ($_SESSION["auth"] == true) {
    $_SESSION["auth"] = false;
    unset($_SESSION["auth"]);
    session_destroy();
}
```

Для перевірки статусу авторизації клієнтська частина застосунку може періодично звертатися до файлу checkauth.php, що повертає JSON-відповідь, яка вказує, чи є користувач авторизованим у поточній сесії (лістинг 3.3).

Лістинг 3.3 – Перевірка статусу авторизації

```
<?php
session_start();
if ($_SESSION["auth"] == true) {
  echo json_encode(array("auth" => true));
} else {
  echo json_encode(array("auth" => false));
}
```

Щоб надати користувачеві можливість обирати та редагувати існуючі сторінки, реалізовано серверний скрипт pageList.php, який повертає список HTML-файлів, що зберігаються в кореневому каталозі вебсайту. Перед виконанням основної логіки перевіряється авторизованість користувача за допомогою змінної $_SESSION["auth"]. У разі відсутності авторизації сервер повертає статус 403 Forbidden (лістинг 3.4).

Лістинг 3.4 – Отримання списку сторінок

```
<?php
session_start();
if ($_SESSION["auth"] != true) {
    header("HTTP/1.0 403 Forbidden");
    die;
}
$htmlfiles = glob("../../*.html");
$response = [];
foreach ($htmlfiles as $file) {
    array_push($response, basename($file));
}
echo json_encode($response);
```

Для збереження оновленого HTML-коду сторінки використовується файл savePage.php. Перед перезаписом старого вмісту застосунок автоматично створює резервну копію сторінки, копіюючи її до окремого каталогу backups. Ім’я резервного файлу генерується за допомогою uniqid для уникнення конфліктів. До кожної резервної копії додається інформація про час збереження, яка записується у файл backups.json. Після цього новий HTML-код зберігається у відповідний файл (лістинг 3.5).

Лістинг 3.5 – Фрагмент збереження змін і створення резервної копії

```
$_POST = json_decode( file_get_contents("php://input"), true );
$file = $_POST["pageName"];
$newHTML = $_POST["html"];
if (!is_dir("../backups/")) {mkdir("../backups/");}
if ($newHTML && $file) {
    $backupFN = uniqid() . ".html";
    copy("../../" . $file, "../backups/" . $backupFN );
    array_push($backups, ["page" => $file, "file" => $backupFN,
"time" => date("H:i:s d:m:y")]);
    file_put_contents("../backups/backups.json", json_encode(
$backups ));
```

```
    file_put_contents("../../" . $file, $newHTML);
} else {
    header("HTTP/1.0 400 Bad Request");
}
```

У процесі редагування HTML-сторінки з інтерфейсу користувача може використовуватись модифікований формат HTML із кастомними тегами, які спрощують візуальне редагування. Щоб не змінювати оригінальний файл одразу, всі зміни спочатку зберігаються у тимчасовий файл. Для цього реалізовано скрипт saveTempPage.php, який приймає HTML-код у форматі JSON через POST-запит і записує його в окремий файл asdl12wqjk331sd.html з унікальною назвою, щоб не було конфліктів (лістинг 3.6).

Лістинг 3.6 – Попереднє збереження HTML-сторінки

```
<?php
session_start();
if ($_SESSION["auth"] != true) {
    header("HTTP/1.0 403 Forbidden");
    die;
}
$_POST = json_decode( file_get_contents("php://input"), true );
$newFile = "../../asdl12wqjk331sd.html";
if ($_POST["html"]) { file_put_contents($newFile,
$_POST["html"]);
} else {header("HTTP/1.0 400 Bad Request");}
```

Такий підхід дозволяє редагувати сторінку з використанням допоміжних тегів, не змінюючи справжній вміст оригінального HTML-файлу до моменту збереження. Після завершення редагування кастомні теги видаляються та очищений HTML-код записується в оригінальний файл.

Для забезпечення безпеки даних та можливості повернення до попередніх версій сторінок у вебзастосунку реалізовано механізм резервного копіювання. У разі потреби користувач може відновити одну з попередніх версій сторінки. Для цього використовується серверний скрипт restoreBackup.php, який копіює резервний файл з каталогу backups на місце оригінальної сторінки (лістинг 3.7).

Лістинг 3.7 – Відновлення резервної копії сторінки

```
$_POST = json_decode( file_get_contents("php://input"), true );
$file = $_POST["file"];
$page = $_POST["page"];
if ($page && $file) {
    copy("../backups/" . $file, "../../" . $page );
} else {
    header("HTTP/1.0 400 Bad Request");
}
```

Запит на відновлення надсилається через POST-запит у форматі JSON, де вказуються назви резервного файлу та сторінки, що потребує відновлення. У разі успішної перевірки прав доступу та наявності необхідних параметрів відбувається копіювання резервної версії поверх поточної. Це дає змогу легко повертати сторінку до стабільного стану після редагування або помилок.

3.3 Реалізація клієнтської логіки

Клієнтська частина вебзастосунку реалізована з використанням бібліотеки React, що забезпечує побудову зручного, динамічного та масштабованого інтерфейсу. У проєкті також використано додаткові інструменти, такі як Ant Design, Axios, core-js, React та ReactDOM. Всі залежності описано у файлі package.json.

Лістинг 3.8 – Залежності з package.json

```
"dependencies": {
  "antd": "^5.25.1",
  "axios": "^1.7.7",
  "core-js": "^3.38.1",
  "react": "^18.3.1",
  "react-dom": "^18.3.1"
}
```

Для автоматизації збирання, компіляції стилів, копіювання файлів та спостереження за змінами використовується Gulp. Він забезпечує ефективний процес розробки, відслідковування змін та оновлення файлів у реальному часі. Серед основних задач Gulp – компіляція SCSS, збирання

JSX-компонентів, копіювання HTML, API та ресурсів. Спостереження за змінами реалізовано через gulp.watch (лістинг 3.9).

Лістинг 3.9 – Завдання Gulp для стеження за змінами

```
gulp.task("watch", () => {
  gulp.watch("./app/src/index.html", gulp.parallel("copy-
html"));
  gulp.watch("./app/assets/**/*.*", gulp.parallel("copy-
assets"));
  gulp.watch("./app/api/**/*.*", gulp.parallel("copy-api"));
  gulp.watch("./app/scss/**/*.scss", gulp.parallel("build-
sass"));
  gulp.watch("./app/src/**/*.{js,jsx}", gulp.parallel("build-
js"));
});
```

Таким чином, під час розробки всі зміни в коді автоматично обробляються, що значно пришвидшує процес та спрощує контроль за структурою проєкту.

Одним із ключових елементів взаємодії з користувачем є форма авторизації, реалізована у вигляді React-класу Login. Вона відповідає за прийом пароля, перевірку його довжини та ініціацію процесу входу. Інтерфейс побудований за допомогою компонентів з бібліотеки Ant Design, що забезпечує зручний та сучасний вигляд форми (лістинг 3.10).

Лістинг 3.10 – Фрагмент компонента авторизації

```
<Input.Password
  placeholder="Пароль"
  value={pass}
  onChange={this.onPasswordChange}
/>
<Button
  type="primary"
  block
  onClick={() => login(pass)}
> Вхід </Button>
```

Цей компонент є першою точкою входу у систему, що перевіряє доступ до адміністративної панелі. Після введення правильного пароля користувача авторизують через session, а сервер надає дозвіл на подальші дії. Усі помилки передаються в компонент через props, що дає змогу контролювати логіку

валідації на рівні батьківського компонента або модуля авторизації.

Наступним важливим елементом клієнтської логіки є компонент панелі управління Panel, який містить кнопки, що відкривають відповідні модальні вікна для виконання основних дій:

- відкриття HTML-сторінки;
- публікація змін;
- редагування мета-інформації;
- відновлення резервної копії;
- вихід з адмін-панелі.

Лістинг 3.11 – Фрагмент компонента панелі

```
<Button onClick={() => onOpenPageModal()}> Відкрити </Button>
<Button onClick={() => onOpenSaveModal()}> Опублікувати
</Button>
<Button onClick={() => onOpenMetaModal()}> Редагувати META
</Button>
<Button onClick={() => onOpenBackupModal()}> Відновити </Button>
<Button onClick={() => onOpenExitModal()}> Вихід </Button>
```

Компонент використовує стилі Ant Design (Button із параметром shape="round"), а обробники подій передаються через props, що забезпечує гнучкість та можливість повторного використання.

Одним із головних елементів застосунку є клас Editor (лістинг 3.12), який реалізує повноцінний функціонал візуального редактора сторінок. Він відповідає за завантаження HTML-коду сторінки у iframe, обробку DOM, редагування текстів, зображень, мета-даних, а також збереження змін.

Лістинг 3.12 – Фрагмент класу Editor

```
export default class Editor extends Component {
  constructor() {
    super();
    this.currentPage = "index.html";
    this.state = {
      // Стан класу };
    this.isLoading = this.isLoading.bind(this);
    this.isLoaded = this.isLoaded.bind(this);
    this.save = this.save.bind(this);
    this.init = this.init.bind(this);
```

```
    this.login = this.login.bind(this);
    this.logout = this.logout.bind(this);
    this.restoreBackup = this.restoreBackup.bind(this);
  }
```

Клас Editor є класовим React-компонентом та використовує різні сервіси, такі як axios, DOMHelper, а також дочірні компоненти.

Для початку роботи редактора важливо переконатися, що користувач авторизований. Цей крок виконується одразу після монтування головного компонента завдяки методу життєвого циклу componentDidMount (лістинг 3.13). Він викликає функцію checkAuth, яка перевіряє, чи має користувач доступ до редактора. Цей метод є стандартною частиною класових компонентів React та автоматично викликається після того, як компонент вперше відображено на сторінці. Його головна задача – ініціювати процес перевірки прав доступу, щоб захистити функціонал редактора від неавторизованих користувачів.

Лістинг 3.13 – Метод життєвого циклу componentDidMount

```
componentDidMount() {
  this.checkAuth();
}
```

Метод componentDidUpdate (лістинг 3.14) реагує на оновлення стану компонента. У випадку, коли статус авторизації (auth) змінився, наприклад, коли користувач успішно ввійшов у систему, запускається метод init, який ініціалізує редактор для поточної сторінки. Це дозволяє завантажити всі необхідні дані лише після того, як користувач отримав доступ.

Лістинг 3.14 – Метод життєвого циклу componentDidUpdate

```
componentDidUpdate(prevProps, prevState) {
  if (this.state.auth !== prevState.auth) {
    this.init(null, this.currentPage);
  }}
```

Важливою частиною роботи редактора є система авторизації, яка реалізується через методи checkAuth, login та logout. Вони забезпечують

перевірку доступу, вхід за паролем та вихід із системи відповідно. Для взаємодії з сервером використовується бібліотека axios, яка дозволяє виконувати HTTP-запити.

Метод checkAuth (лістинг 3.15) виконується одразу після монтування компонента через componentDidMount та надсилає GET-запит до серверного скрипту checkAuth.php. У відповідь сервер надсилає статус авторизації, який зберігається в стані auth. Цей механізм дозволяє перевірити, чи вже ввійшов користувач раніше, наприклад, через збережену сесію.

Лістинг 3.15 – Метод checkAuth

```
checkAuth() {
  axios.get("./api/checkAuth.php").then((res) => {
    this.setState({
      auth: res.data.auth,
    });
  });}
```

Метод login (лістинг 3.16) відповідає за вхід користувача. Перед відправкою запиту перевіряється, чи пароль містить більше ніж 5 символів. Якщо умова виконана, надсилається POST-запит на login.php із введеним паролем. У відповіді сервер повертає статус авторизації, який зберігається в state. У випадку помилки виводиться відповідне повідомлення: або неправильний пароль loginError, або занадто короткий loginLengthError.

Лістинг 3.16 – Метод login

```
login(pass) {
if (pass.length > 5) { axios.post("./api/login.php", {
password: pass }).then((res) => {this.setState({ auth:
res.data.auth, loginError: !res.data.auth, loginLengthError:
false,});});} else {this.setState({loginError:
false,loginLengthError: true,});}}
```

Для виходу користувача з системи реалізовано Метод logout (лістинг 3.17). Він надсилає запит до logout.php, який на сервері завершує сесію. Після цього відбувається перенаправлення на головну сторінку, що фактично виконує вихід із системи.

Лістинг 3.17 – Метод logout

```
logout() {
  axios.get("./api/logout.php").then(() => {
    window.location.replace("/");
});
}
```

Метод init (лістинг 3.18) відповідає за початкову підготовку редактора до роботи. Якщо виклик ініційовано з події (наприклад, кліком на пункт меню), спочатку блокується стандартна поведінка браузера. Далі, якщо користувач авторизований, виконується:

- активація індикатора завантаження isLoading;
- отримання посилання на iframe, у якому відкривається редагована сторінка;
- виклик методу open для завантаження вмісту;
- завантаження списку сторінок loadPageList та резервних копій loadBackupsList.

Лістинг 3.18 – Метод init

```
init(e, page) {
    if (e) {e.preventDefault();}
    if (this.state.auth) {
      this.isLoading();
      this.iframe = document.querySelector("iframe");
      this.open(page, this.isLoaded);
      this.loadPageList();
      this.loadBackupsList();    }}
```

Метод open (лістинг 3.19) запускає повний процес завантаження обраної сторінки: завантаження HTML, його обробка, а саме обгортання текстів та зображень у власний тег, збереження тимчасової версії сторінки та відображення її в iframe. У фіналі викликається enableEditing – метод, який підключає функції редагування.

Лістинг 3.19 – Метод open

```
open(page) {
  this.setLoading(true);
  this.currentPage = page;
  axios
    .get(`../pages/${page}.html`)
    .then(({ data }) => DOMHelper.parse(data))
    .then((doc) => DOMHelper.wrapTextNodes(doc.body))
    .then((doc) => DOMHelper.wrapImages(doc.body))
    .then((doc) => {
      this.virtualDom = doc;
      return DOMHelper.serialize(doc);
    })
    .then((html) => {
      return axios.post('./api/saveTempPage.php', { html });
    })
    .then(() => this.iframe.load('../pages/temp.html'))
    .then(() => this.enableEditing())
    .catch((err) => console.log('open', err))
    .finally(() => this.setLoading(false));}
```

Після успішної авторизації та ініціалізації редактора виконується завантаження важливих даних: доступного списку сторінок для редагування та резервних копій поточної сторінки. За це відповідають методи loadPageList та loadBackupsList.

Метод loadPageList (лістинг 3.20) надсилає GET-запит до серверного скрипту pageList.php, який повертає список сторінок, доступних для редагування. Отримані дані зберігаються у стані pageList, що дозволяє надалі відображати їх у інтерфейсі, наприклад, у списку вибору сторінки. Це дозволяє користувачу швидко перемикатися між різними розділами сайту.

Лістинг 3.20 – Метод loadPageList

```
loadPageList() {
  axios
    .get("./api/pageList.php")
    .then((res) => this.setState({ pageList: res.data }));}
```

Для завантаження резервних копій реалізовано метод loadBackupsList (лістинг 3.21). Цей метод завантажує JSON-файл із локальної директорії backups, який містить інформацію про всі збережені резервні копії. Після отримання даних виконується фільтрація: залишаються лише ті резервні копії, що відповідають поточній сторінці (this.currentPage). Результат

записується у стан backupsList, що дозволяє зручно керувати резервуванням змін для кожної окремої сторінки.

Лістинг 3.21 – Метод loadBackupsList

```
loadBackupsList() {
  axios.get("./backups/backups.json").then((res) =>
    this.setState({
      backupsList: res.data.filter((backup) => {
        return backup.page === this.currentPage;
}),}));}
```

Одним із етапів підготовки сторінки до редагування є виділення всіх текстових вузлів (тексту між HTML-тегами) та обгортання їх спеціальними тегами, які можна редагувати. За це відповідає статичний метод wrapTextNodes (лістинг 3.22) з допоміжного класу DOMHelper, який трансформує DOM-структуру, додаючи до неї спеціальні елементи text-editor.

Лістинг 3.22 – Метод wrapTextNodes

```
static wrapTextNodes(dom) {
  const body = dom.body;
  let textNodes = [];
  function recursy(element) {
    element.childNodes.forEach(node => {
      if (node.nodeName === "#text" &&
node.nodeValue.replace(/\s+/g, "")
.length > 0) { textNodes.push(node);
 } else { recursy(node);}
    });
  }
  recursy(body);
  textNodes.forEach((node, i) => {
    const wrapper = dom.createElement('text-editor');
    node.parentNode.replaceChild(wrapper, node);
    wrapper.appendChild(node);
    wrapper.setAttribute("nodeid", i);
  });
  return dom;
}
```

Метод wrapTextNodes рекурсивно обходить усі елементи в межах тегу body, знаходить текстові вузли (які не є тегами), фільтрує з них порожні або такі, що містять лише пробіли, та обгортає кожен із них у спеціальний

елемент text-editor. Усередині цього елемента зберігається оригінальний текст, а також додається атрибут nodeid з унікальним індексом для подальшої ідентифікації та зв’язку з віртуальним DOM.

Одним із важливих кроків під час роботи редактора є перехід у режим редагування, в якому користувач може змінювати текстові блоки та зображення безпосередньо на сторінці. За це відповідають методи enableEditing та injectStyles, які працюють з елементами у вбудованому фреймі (iframe), де відображається сторінка сайту.

Метод enableEditing (лістинг 3.23) активує редагування тексту та зображень безпосередньо у вмісті фрейму. Спочатку він знаходить усі елементи з тегом text-editor та створює для кожного екземпляр класу EditorText, який відповідає за редагування текстових блоків. Далі аналогічно обробляються всі зображення з атрибутом editableimgid, для яких створюються об’єкти EditorImages. Вони дозволяють редагувати зображення у візуальному інтерфейсі, використовуючи функції завантаження, підказки та індикацію процесу. Важливу роль у цьому відіграє використання віртуального DOM – virtualDom, який забезпечує точне співставлення реальних елементів з їх віртуальними копіями, необхідними для збереження та подальшого оновлення контенту.

Лістинг 3.23 – Метод enableEditing

```
enableEditing() {
    this.iframe.contentDocument.body
      .querySelectorAll("text-editor")
      .forEach((element) => {
        const id = element.getAttribute("nodeid");
        const virtualElement =
this.virtualDom.body.querySelector(
          `[nodeid="${id}"]`
        );
        new EditorText(element, virtualElement);
      });
    this.iframe.contentDocument.body
      .querySelectorAll("[editableimgid]")
      .forEach((element) => {
        const id = element.getAttribute("editableimgid");
        const virtualElement =
```

```
this.virtualDom.body.querySelector(
          `[editableimgid="${id}"]`
        );
        new EditorImages(
          element,
          virtualElement,
          this.isLoading,
          this.isLoaded,
          this.showNotifications
        );
      });
  }
```

Метод injectStyles (лістинг 3.24) додає до вмісту сторінки у фреймі спеціальні CSS-стилі, які візуально підсвічують редаговані елементи. Наприклад, при наведенні курсора на блок з текстом або зображенням, він обводиться зеленою рамкою, а при фокусі – червоною. Це покращує зручність взаємодії, дозволяючи користувачу легко розпізнавати, які елементи можна редагувати.

Лістинг 3.24 – Метод injectStyles

```
injectStyles() {
    const style =
this.iframe.contentDocument.createElement("style");
    style.innerHTML = `
     text-editor:hover { outline: 3px solid green; outline-
offset: 8px;}
     text-editor:focus { outline: 3px solid red; outline-offset:
8px;}
    [editableimgid]:hover { outline: 3px solid green; outline-
offset: 8px;}`;
this.iframe.contentDocument.head.appendChild(style);
}
```

Метод restoreBackup (лістинг 3.25) відповідає за відновлення сторінки з резервної копії у вебзастосунку. Він приймає параметри "e" (об’єкт події) та "backup" (назва файлу резервної копії), блокує стандартну поведінку браузера за допомогою e.preventDefault, якщо подія передана, і викликає модальне вікно підтвердження через Modal.confirm бібліотеки Ant Design. Після підтвердження користувачем метод надсилає асинхронний POST-запит до restoreBackup.php із назвою сторінки this.currentPage та файлу резервної копії,

активуючи індикатор завантаження через this.isLoading.

У разі успішного виконання запиту метод викликає this.open для завантаження відновленої сторінки в редактор. При помилці, наприклад, через відсутність файлу, відображається повідомлення через message.error, а деталі виводяться в консоль.

Лістинг 3.25 – Метод restoreBackup

```
restoreBackup(e, backup) {
    if (e) {e.preventDefault();}
    Modal.confirm({
      title: "Ви дійсно хочете відновити сторінку з цієї
резервної копії?",
      content: "Усі незбережені дані буде втрачено!",
      okText: "Відновити",
      cancelText: "Скасувати",
      onOk: async () => {
        try {
          this.isLoading();
          await axios
            .post("./api/restoreBackup.php", {
              page: this.currentPage,
              file: backup,
            })
            .then(() => {
              this.open(this.currentPage, this.isLoaded);
            });
        } catch (error) {
          message.error("Помилка при відновленні");
          console.error(error);
}},});}
```

Метод save (лістинг 3.26) відповідає за збереження відредагованої сторінки у вебзастосунку: створює копію віртуального DOM (this.virtualDom); видаляє спеціальні теги text-editor та атрибути зображень за допомогою методів DOMHelper.unwrapTextNodes та DOMHelper.unwrapImages; після чого серіалізує DOM у HTML-рядок через DOMHelper.serializeDOMToString. Асинхронний POST-запит до savePage.php передає назву сторінки (this.currentPage) та HTML-код, активуючи індикатор завантаження через this.isLoading().

Після успішного збереження відображається повідомлення про успіх

через this.showNotifications, а при помилці – повідомлення про невдачу. У будь-якому випадку викликається this.isLoaded() для завершення індикації завантаження, а метод this.loadBackupsList() оновлює список резервних копій.

Лістинг 3.26 – Метод save

```
async save() {
    this.isLoading();
    const newDom = this.virtualDom.cloneNode(this.virtualDom);
    DOMHelper.unwrapTextNodes(newDom);
    DOMHelper.unwrapImages(newDom);
    const html = DOMHelper.serializeDOMToString(newDom);
    await axios
      .post("./api/savePage.php", { pageName: this.currentPage,
html })
      .then(() => this.showNotifications("Успішно збережено",
"success"))
      .catch(() => this.showNotifications("Помилка збереження",
"error"))
      .finally(this.isLoaded);
    this.loadBackupsList();
}
```

4 ІНСТРУКЦІЯ КОРИСТУВАЧА

Для початку роботи з вебзастосунком необхідно розмістити систему в каталог, в якому знаходиться сам сайт. Після цього відкрити браузер і ввести адресу, після чого відкриється сторінка з формою авторизації (рисунок 4.1).

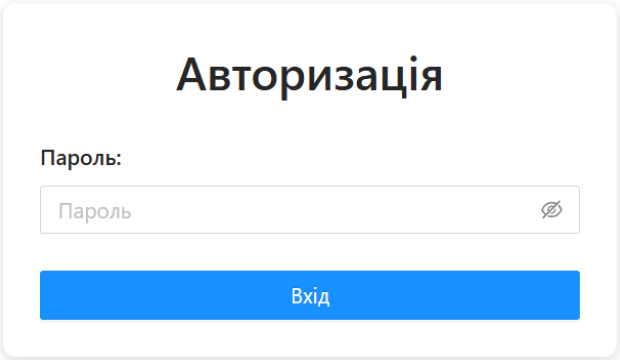

Рисунок 4.1 – Форма авторизації вебзастосунку

На цій сторінці потрібно ввести пароль, який попередньо визначено в налаштуваннях. Після введення паролю система перевіряє його коректність та, за умови успішної валідації, надає доступ до основного інтерфейсу. У разі помилки відображається повідомлення про необхідність повторного правильного введення (рисунок 4.2).

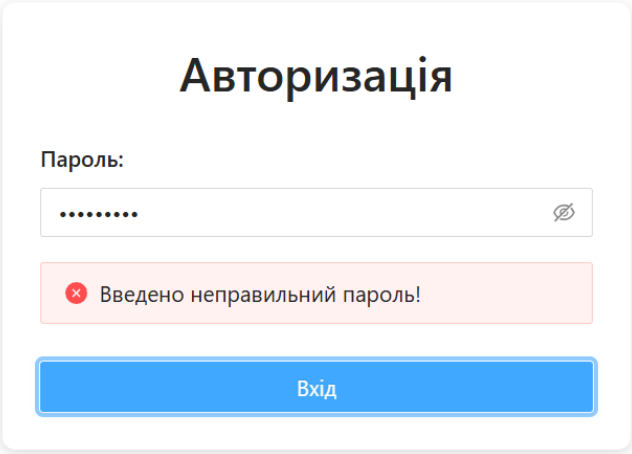

Рисунок 4.2 – Помилка авторизації вебзастосунку

Після авторизації відкриється доступ до інтерфейсу системи, де користувач може редагувати зміст сторінок. Зверху сторінки знаходиться зручна навігаційна панель, на якій розташовані всі необхідні інструменти для роботи (рисунок 4.3).

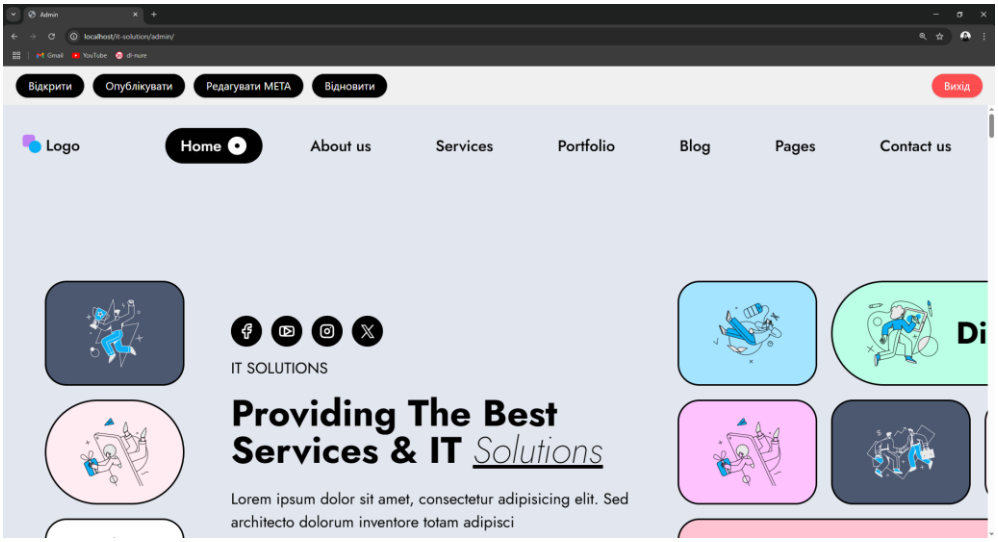

Рисунок 4.3 – Інтерфейс вебзастосунку

При наведенні курсора на текстове поле воно підсвічується зеленою рамкою, що сигналізує про можливість редагування. Якщо натиснути на текст, рамка змінить колір на червоний – це означає, що активовано режим редагування, і текст можна змінювати безпосередньо в полі (рисунок 4.4).

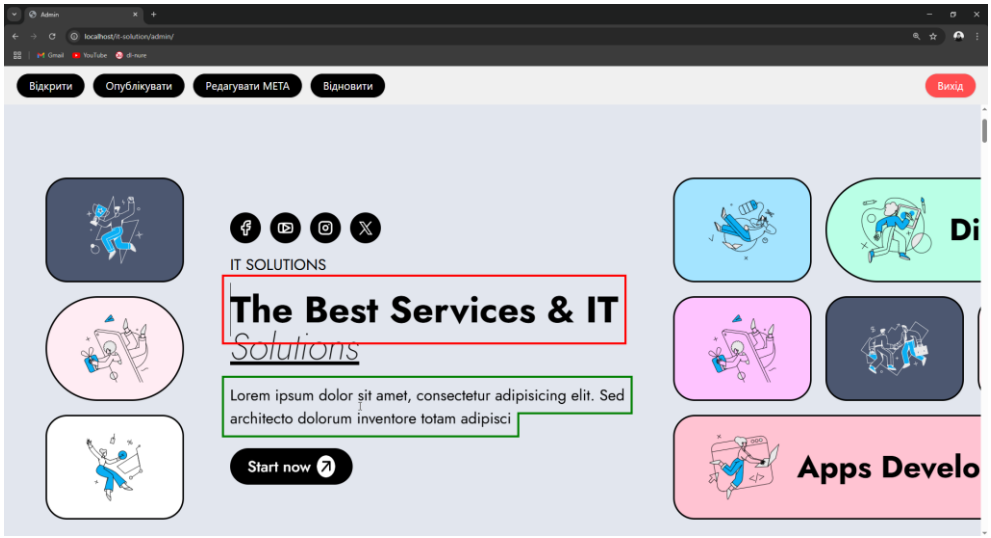

Рисунок 4.4 – Редагування текста

Після завершення редагування, щоб опублікувати оновлену версію сайту, потрібно натиснути кнопку "Опублікувати". Після цього відкриється модальне вікно з підтвердженням дії (рисунок 4.5).

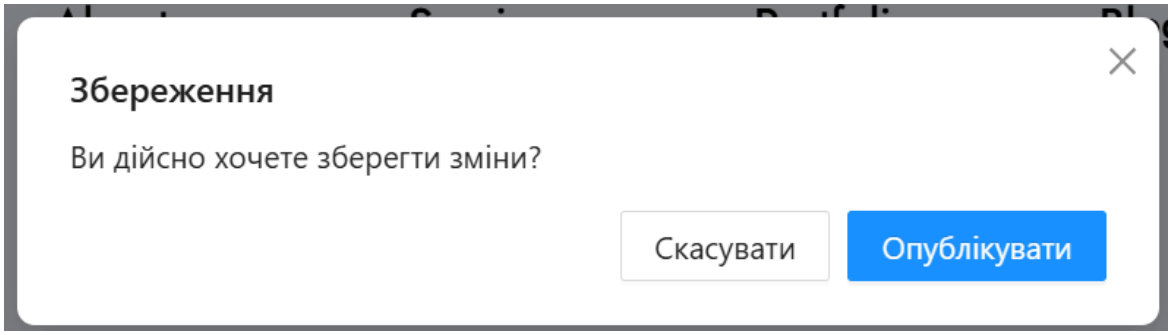

Рисунок 4.5 – Підтвердження збереження

З метою захисту даних система забезпечує можливість відновлення данних з резервної копії. Користувач може відновити дані за допомогою відповідного пункту меню. Для відновлення інформації необхідно натиснути на кнопку "Відновити" після чого відкриється модальне вікно зі списком резервних копій (рисунок 4.6).

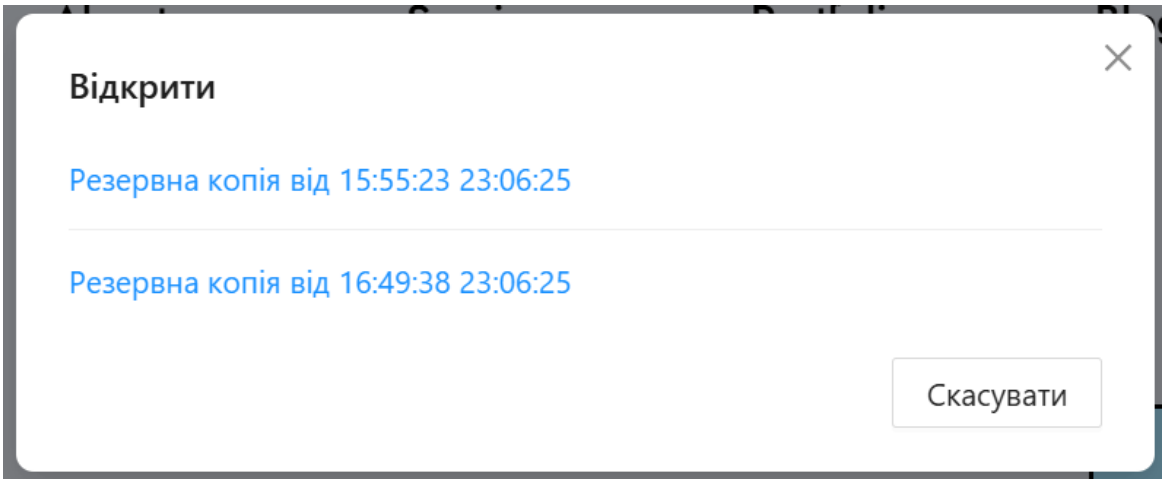

Рисунок 4.5 – Список резервних копій

Система також містить функціонал для вибору іншої сторінки сайту з метою редагування. Користувач може змінювати активну сторінку через спеціальне меню, після чого відкривається відповідний вміст, доступний для

подальшого редагування. Для цього необхідно натиснути на кнопку "Відкрити" та обрати необхідну сторінку зі списку (рисунок 4.6).

Рисунок 4.6 – Список сторінок сайту

У системі передбачено можливість редагування метаданих сайту, зокрема таких як title, description та keywords. Це дозволяє користувачеві налаштовувати SEO-параметри кожної сторінки безпосередньо через інтерфейс, що сприяє покращенню видимості сайту в пошукових системах. Необіхдно натиснути на кнопку "Редагувати META" після чого відкриється відповідна форма (рисунок 4.7).

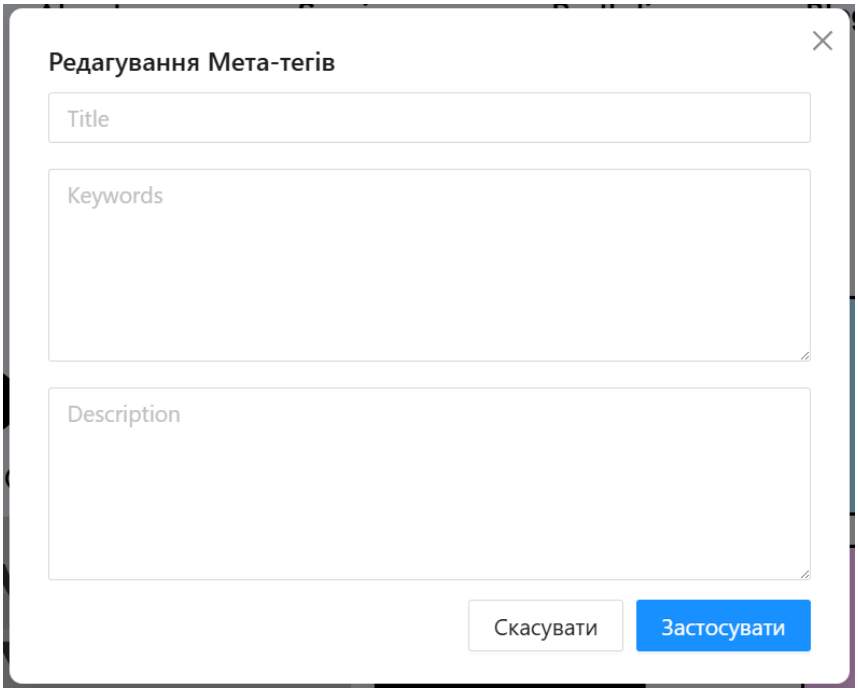

Рисунок 4.7 – Редагування мета-тегів

Для додавання зображень передбачено окрему функцію, де користувач може вибрати файл із локального диска та завантажити його, після чого зображення інтегрується в сторінку (рисунок 4.8).

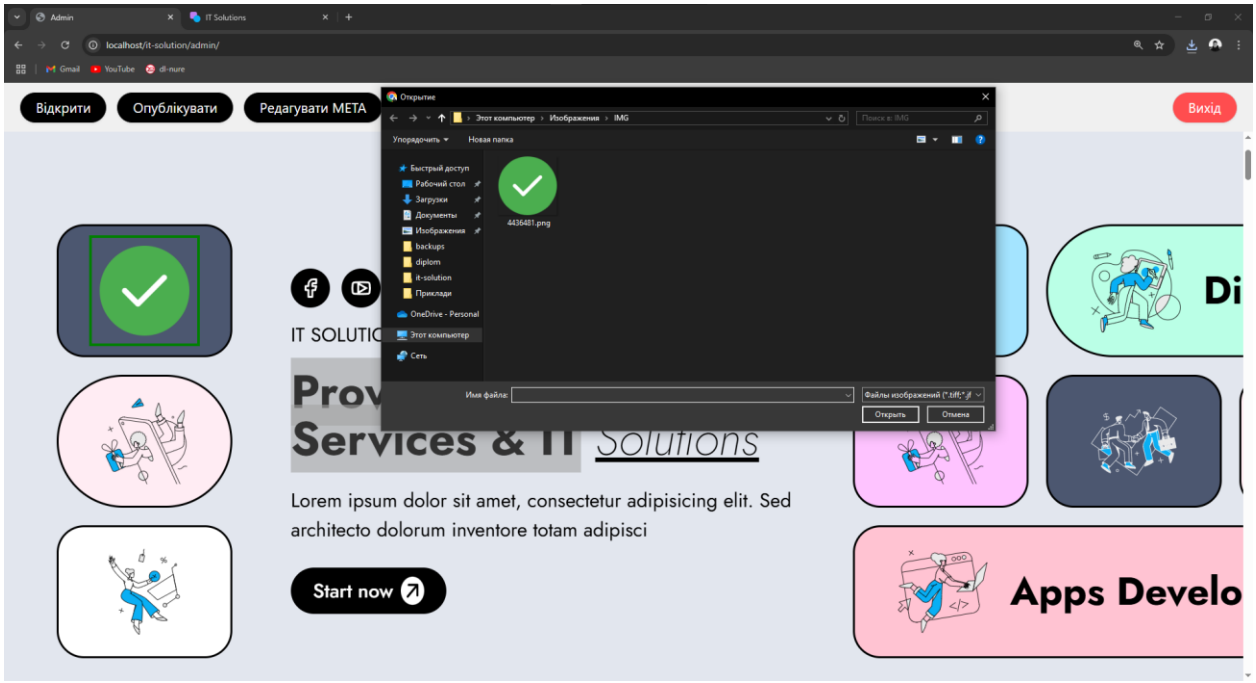

Рисунок 4.7 – Додавання зображення

Для завершення роботи користувач має обрати опцію виходу в інтерфейсі, що призводить до очищення сесії авторизації та організує безпеку даних. Після цього система повертається до початкової сторінки, вимагаючи повторного введення пароля для наступного входу.

У результаті вебзастосунок забезпечує користувачеві простий та зручний інтерфейс для редагування вмісту сайту, додавання зображень, зміни тексту та метаданих, що дозволяє швидко оновлювати інформацію та підтримувати актуальність ресурсу.

ВИСНОВКИ

У рамках кваліфікаційної роботи було проведено аналіз сучасних систем керування контентом (CMS), їхніх переваг, недоліків та ролі в веброзробці, а також розроблено власну просту CMS, яка відповідає потребам користувачів із мінімальним технічним досвідом. Аналіз предметної області показав, що популярні CMS, такі як WordPress, Joomla, Drupal та OpenCart, пропонують широкий функціонал, але мають суттєві недоліки: складність освоєння, ресурсомісткість, залежність від баз даних та вразливість до кібератак. Легші flat-file CMS, такі як Grav, Kirby та Pico, є ефективними для невеликих проєктів, але часто потребують технічних знань або мають обмежену адаптивність.

Розроблена CMS на основі PHP та React із використанням бібліотек Axios, Ant Design та інструментів Gulp та Webpack забезпечує базовий набір функцій: редагування тексту, обробку зображень, налаштування мета-інформації та створення резервних копій. Завдяки flat-file підходу система не потребує бази даних, що спрощує розгортання та знижує вимоги до інфраструктури. Клієнт-серверна архітектура з REST API забезпечує швидку та асинхронну взаємодію, а інтуїтивний інтерфейс, побудований на Ant Design, робить систему доступною для новачків.

Реалізація серверної логіки на PHP включає авторизацію, обробку сторінок та резервне копіювання, тоді як клієнтська частина на React забезпечує динамічне редагування контенту через візуальний редактор. Використання Git та Visual Studio Code оптимізувало процес розробки, а Gulp та Webpack автоматизували збірку проєкту. Розроблена CMS є економічно вигідною та гнучкою альтернативою для невеликих вебпроєктів, таких як блоги чи сайти малого бізнесу, усуваючи надлишковий функціонал та складність традиційних систем. Робота підтвердила актуальність створення простих та адаптованих CMS, які відповідають сучасним тенденціям веброзробки, зокрема персоналізації та оптимізації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. WordPress Codex. WordPress Documentation URL: https://wordpress.org/documentation.

2. Joomla Documentation. Official Guide. URL: https://docs.joomla.org/Portal:Beginners/en

3. Drupal Developer Documentation URL: https://www.drupal.org/docs

4. OpenCart Documentation URL: https://docs.opencart.com/en-gb/introduction.

5. Рибкін Д.В., Філімончук Т.В., Волощук О.Б. CMS для управління контентом вебсайтів. Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління: тези доповідей п’ятнадцятої міжнародної науково-технічної конференції. Т.2: секція 2. Баку: ІСУ АР; Харків: НТУ «ХПІ»;. Харків: ХНУРЕ; Харків: НАУ «ХАІ»; Жиліна: УмЖ. 2025. с. 39.

6. Grav. Grav Documentation. URL: https://learn.getgrav.org/17.

7. Kirby. Kirby Guide. URL: https://getkirby.com/docs/guide.

8. Pico. Pico Documentation. URL: https://picocms.org/docs.

9. Wolf J. HTML and CSS: The Comprehensive Guide to Excelling in HTML5 and CSS3 for Responsive Web Design, Dynamic Content, and Modern Layouts. Bonn: Rheinwerk Computing, 2023. 850 p.

10. Steinmetz, K., Tatroe, P., MacIntyre, P. Programming PHP: Creating Dynamic Web Pages. Sebastopol, CA: O'Reilly Media, 2020. 548 p.

11. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. Sebastopol, CA: O'Reilly Media, 2020. 310 p.

12. Flanagan D. JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language. Sebastopol, CA: O'Reilly Media, 2020. 704 p.

13. Ant Design. React UI framework. URL: https://ant.design/components/overview.