Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інтелектуальних інформаційних технологій та автоматики
(повне найменування інституту, назва факультету (відділення))
Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Бакалаврська кваліфікаційна робота на тему:

РОЗРОБКА WEB-ДОДАТКУ ДЛЯ CMS З ВИКОРИСТАННЯМ HEADLESS ПІДХОДУ

Виконав: студент 4 курсу, групи 2КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Назарій КОНОНЕНКО
(прізвище та ініціали)
Керівник: к.т.н., старший викладач каф. КН
Сергій ПЕТРИШИН
(прізвище та ініціали)
«04» червня 2025 р.
Рецензент: к.т.н., доц. каф. АІІТ
Марія БАРАБАН
(прізвище та ініціали)
«05» червня 2025 р.

Допущено до захисту
Зав. кафедри _____
«06» червня 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 "Інформаційні технології"
Спеціальність – 122 "Комп'ютерні науки"
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Андрій ЯРОВИЙ
"05" травня 2025 р

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Кононенку Назарію Валерійовичу

1. Тема роботи: Розробка WEB-додатку для CMS з використанням Headless підходу.
Керівник роботи: к.т.н., старший викладач каф. КН Сергій ПЕТРИШИН.
Затверджені наказом вищого навчального закладу "20" 03 2025 року № 97

2. Термін подання студентом роботи 30 травня 2025 р.

3. Вихідні дані до роботи :
Мова програмування об'єктно-орієнтована;
Кількість користувачів - не менше 100 чол;
Очікуване середнє навантаження на систему: не менше 200 сеансів на день;

4. Зміст текстової частини (перелік питань, які потрібно розробити):
вступ; обґрунтування доцільності розробки Розробка WEB-додатку для CMS з використанням Headless підходу; проектування Розробка WEB-додатку для CMS з використанням Headless підходу; розробка Розробка WEB-додатку для CMS з використанням Headless підходу; тестування Розробка WEB-додатку для CMS з використанням Headless підходу; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): UML-діаграма класів, зображення розробленого додатку, зображення Fullstack архітектури, зображення розгортання backend-частини додатку, зображення схеми роботи класичної CMS.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	к.т.н., старший викладач каф. КН Сергій ПЕТРИШИН		

7. Дата видачі завдання 05 травня 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Приміт
		початок	закінчення	
1	Обгрунтування доцільності розробки WEB-додатку для CMS з використанням Headless підходу	05.05.25	07.05.25	
2	Проектування WEB-додатку для CMS з використанням Headless підходу	08.05.25	11.05.25	
3	Розробка WEB-додатку для CMS з використанням Headless підходу	12.05.25	15.05.25	
4	Тестування WEB-додатку для CMS з використанням Headless підходу	16.05.25	26.05.25	
5	Розробка інструкції користувача	27.05.25	29.05.25	
6	Аналіз отриманих результатів та формування висновків	30.05.25	01.06.25	
7	Оформлення бакалаврської кваліфікаційної роботи	02.06.25	04.06.25	

Студент _____ **Назарій КОНОНЕНКО**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Сергій ПЕТРИШИН**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кононенко Н.В. Розробка WEB-додатку для CMS з використанням Headless підходу. Бакалаврська кваліфікаційна робота складається з 80 сторінок формату А4, на яких є 33 рисунків, список використаних джерел містить 22 найменувань.

Метою бакалаврської роботи є підвищення гнучкості архітектури та оптимізації процесу багатоканальної доставки контенту за рахунок можливостей застосування Headless підходу у системах управління контентом (CMS).

Результатом кваліфікаційного дослідження є WEB-додаток, що реалізований на мові програмування PHP та JavaScript, з використанням CMS WordPress у програмному середовищі PhpStorm.

Ключові слова: Headless, CMS, Headless CMS, WordPress, PHP, React, монолітна архітектура, блог, доставка контенту.

ABSTRACT

Kononenko N.V. Development of a WEB Application for a CMS Using the Headless Approach. The Bachelor's Qualification Work consists of 80 A4 pages, including 33 figures; the list of references contains 22 sources.

The goal of this Bachelor's Qualification Work is to enhance the flexibility of architecture and to optimize the process of multichannel content delivery by utilizing the capabilities of the Headless approach in content management systems (CMS).

The result of the qualification research is a WEB application developed using PHP and JavaScript, based on the WordPress CMS within the PhpStorm development environment.

Keywords: Headless, CMS, Headless CMS, WordPress, PHP, React, monolithic architecture, blog, content delivery.

ЗМІСТ

ВСТУП

Актуальність дослідження. У наш час спостерігається швидке невпинне зростання попиту на використання WEB-застосунків, що побудовані з урахуванням сучасної архітектури. Кількість компаній та бізнесів, що перейшли на нову архітектуру постійно збільшується. Традиційні монолітні підходи у використання CMS все частіше поступаються місцем новим підходам, що значно полегшують процес розробки.

Метою бакалаврської роботи є підвищення гнучкості архітектури та оптимізації процесу багатоканальної доставки контенту за рахунок можливостей застосування Headless підходу у системах управління контентом (CMS).

Об’єкт дослідження – процес створення WEB-додатку використовуючи Headless підхід.

Предметом дослідження є програмні засоби створення WEB-додатку використовуючи Headless підхід.

Задачі дослідження:

– обгрунтувати доцільність WEB-додатку для CMS з використанням Headless підходу;

– провести аналіз сучасних технологій для реалізації WEB-додатку для CMS з використанням Headless підходу;

– спроектувати та створити функціонал платформи;

– провести тестування додатку та оцінити зручність та стабільність його роботи.

Публікації: за основними результатами досліджень опубліковано тези доповіді на науково-технічній конференції “Молодь в науці: дослідження, проблеми, перспективи (МН-2025)” [1].

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ WEB-ДОДАТКУ ДЛЯ CMS З ВИКОРИСТАННЯМ HEADLESS ПІДХОДУ

1.1 Аналіз предметної області

Сучасні WEB-додатки вже давно вийшли за рамки опрацювання лише рушієм браузеру. Світ WEB-розробки невпинно розвивається та основний акцент зосереджується як на швидкодії WEB-сайту або додатку, так і на “developer experience”, безпосередньо досвіду розробника, що працює із інструментами. Інструменти для розробки повинні бути, у першу чергу зручними для програміста, адже у погоні за швидкодією, можна втратити безпосередньо людей, що готові на цьому інструменті розробляти.

Проблему швидкодії можна назвати історичною і пов’язати з неправильною оцінкою важливості WEB-сайтів у сучасному світі. Розробники не закладали аспекти швидкодії як стрижневі для WEB-застосунків, адже перші гіпертекстові системи інтернет-документів були запроваджені ще наприкінці 90-х. Поступово сайти ставали більше складними, отримуючи нові можливості, для браузерів створювались окремі мови програмування, якими були написані рушії, наприклад JavaScript чи PHP – мова, яка створена для генерації HTML сторінок на стороні сервера. Оскільки WEB-сайти ставали все важчими для обробки, навіть передовому рушію браузеру важко обробити таку кількість інформації і звідси постає головна проблема швидкодії: повільна обробка DOM-дерева.

Щоб глибше проілюструвати проблему швидкодії та пояснити доцільність використання Headless підходу замість традиційних способів розробки, наведу приклад створення JS-бібліотеки React, з використанням якої буде безпосередньо побудовано користувацький інтерфейс у WEB-додатку, що розроблятиметься у рамках виконання кваліфікаційної роботи. Безпосередньо React з’явився з дуже простої проблеми: компанія Facebook усіма можливими способами намагалась оптимізувати роботу свого месенджера безпосередньо зіштовхнулись зі всіма

складнощами світу frontend. Клієнтська частина отримує абстрактні дані, обробляє їх та повинна відмалювати результат, оперуючи вище згаданим DOM-деревом. Але операції над HTML катастрофічно повільні, а якщо викликається процес reflow у рушії браузера, тобто динамічна зміна позиціонування елементів, затримки стають надзвичайно великими та значно знижують швидкодії. Отже, враховуючи вищесказане, щоб вирішити проблему потрібно оновлювати HTML як можна менше, відмалювавши у браузері лише змінені елементи. Також, touch-інтерфейси принесли проблему з чуйністю браузера. Мінімальна кількість кадрів, що підтримує телефон це 60 к/с, а отже у рушія браузера є 1000/60, що приблизно дорівнює 16 мілісекундам, аби відмалювати інтерфейс під час свайпу чи скролу та реагувати на дії користувача.

Проблема швидкодії вирішується шляхом використання механізму клонування DOM-дерева. При цьому здійснюється порівняння поточного стану екрана з попереднім. У разі необхідності оновлення інтерфейсу два дерева, що зберігаються у віртуальній пам’яті, аналізуються на відмінності, після чого виконується оновлення відповідних елементів реального HTML-документа.

Із розвитком підходів до розробки WEB-додатків набув популярності так званий Headless підхід до побудови CMS (Content Management System). Традиційна CMS, така як WordPress або Joomla, одночасно відповідає за зберігання, управління та відображення контенту, тобто об'єднує back-end та front-end у єдину систему. Натомість у Headless CMS ці компоненти розділені: система керує лише контентом і надає його через API (часто REST або GraphQL), а за відображення даних відповідає окремий frontend, який може бути реалізований за допомогою сучасних фреймворків, таких як React, Vue.js або Next.js.

Такий підхід дозволяє досягти більшої гнучкості, масштабованості та покращити продуктивність, особливо у багатоплатформних рішеннях. Наприклад, один і той самий контент, який зберігається в Headless CMS.

У контексті розробки вебдодатків із використанням React, перевага Headless полягає в можливості повного контролю над інтерфейсом користувача,

що особливо важливо для побудови інтерактивних SPA (Single Page Applications), де ефективне управління DOM (через Virtual DOM) відіграє ключову роль. У процесі розробки WEB-додатку було визначено низку ключових вимог, яких необхідно дотримуватися для забезпечення якісного та конкурентоспроможного продукту:

– використання сучасних бібліотек та фреймворків. Для створення інтуїтивно зрозумілого та привабливого інтерфейсу користувача слід застосовувати сучасні JavaScript-бібліотеки (наприклад, React), які забезпечують гнучкість, модульність та високу продуктивність при розробці інтерфейсів;

– швидкодія WEB-додатку. Оптимізація продуктивності є критичним фактором у користувацькому досвіді. Додаток має швидко реагувати на дії користувача, ефективно працювати з API, мінімізувати час завантаження та використовувати механізми віртуального DOM або серверного рендерингу, якщо це доречно;

– адаптивність інтерфейсу. Вебдодаток повинен коректно відображатися на пристроях із різними розмірами екранів (десктопи, планшети, смартфони).

1.2 Огляд аналогів розробки WEB-додатку для CMS

На етапі проєктування WEB-додатку з використанням Headless-архітектури доцільно провести аналіз існуючих рішень та підходів. Це дає змогу не лише оцінити ефективність готових інструментів, їх функціонал та обмеження, а й сформувати технічні вимоги до цільової системи з урахуванням актуальних практик розробки.

Сучасний ринок пропонує широкий спектр інструментів для побудови як frontend-, так і backend-частин додатків. До найбільш поширених frontend-бібліотек належать React, Vue.js, Angular, тоді як для серверної частини застосовуються Node.js, Laravel, Django. Системи управління контентом представлені як традиційними CMS (WordPress, Drupal), так і спеціалізованими Headless-рішеннями, на основі CMS.

Вибір архітектурної моделі безпосередньо впливає на технічні характеристики майбутньої системи, її масштабованість, швидкість розробки, можливості інтеграції з іншими сервісами та подальшу підтримку. З огляду на це, доцільно розглянути основні підходи до створення WEB-додатків, що застосовуються на практиці.

Перший із таких підходів передбачає використання вбудованого шаблонізатора в межах CMS. У цьому випадку WEB-додаток створюється безпосередньо у середовищі системи керування контентом. Розмітка сторінок, обробка даних та виведення контенту реалізуються за допомогою вбудованих шаблонів та механізмів CMS. Прикладами такого підходу є сайти на базі WordPress із використанням PHP-шаблонів, Joomla або Drupal із системою Twig-шаблонів. Такий підхід дозволяє швидко запускати проєкти невеликого та середнього масштабу, використовуючи численні готові шаблони і плагіни. Однак при спробі розширити функціонал або інтегрувати сучасний динамічний інтерфейс, ці системи часто виявляються обмеженими. Крім того, масштабування додатків у цій архітектурі зазвичай призводить до зростання складності та зниження продуктивності, особливо при великому трафіку. На рисунку 1.1 зображено схему роботи традиційних CMS.

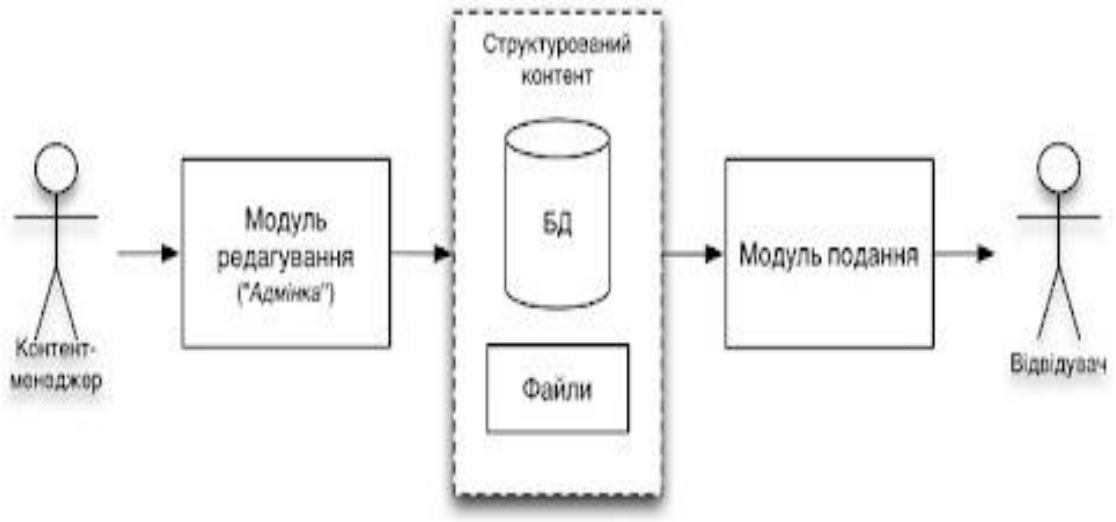


Рисунок 1.1 – Схема роботи традиційної CMS

Структура CMS визначається завданнями, які вона має виконувати для управління контентом. Зазвичай CMS включає наступні основні компоненти:

– адміністративний модуль, який відповідає за редагування контенту та виконання відповідних адміністративних функцій;

– контент, представлений у структурованому вигляді, що зазвичай містить базу даних для збереження HTML-тексту сторінок та набір організованих медіафайлів;

– модуль подання, який є ядром системи, контролює логіку відображення контенту для користувачів і забезпечує навігацію по сайту.

Інший підхід базується на повному розділенні frontend і backend-частин. У цьому випадку frontend створюється як окремий застосунок із використанням сучасних JavaScript-фреймворків, а серверна частина – незалежно, на відповідній платформі. Обмін даними між цими компонентами здійснюється через REST API або GraphQL. Водночас впровадження такого підходу потребує значних зусиль на етапі налаштування інфраструктури і вимагає глибшої технічної компетенції. Це також збільшує тривалість розробки та пов’язані з нею витрати. На рисунку 1.2 зображено схему роботи будь-якого сучасного fullstack додатку.

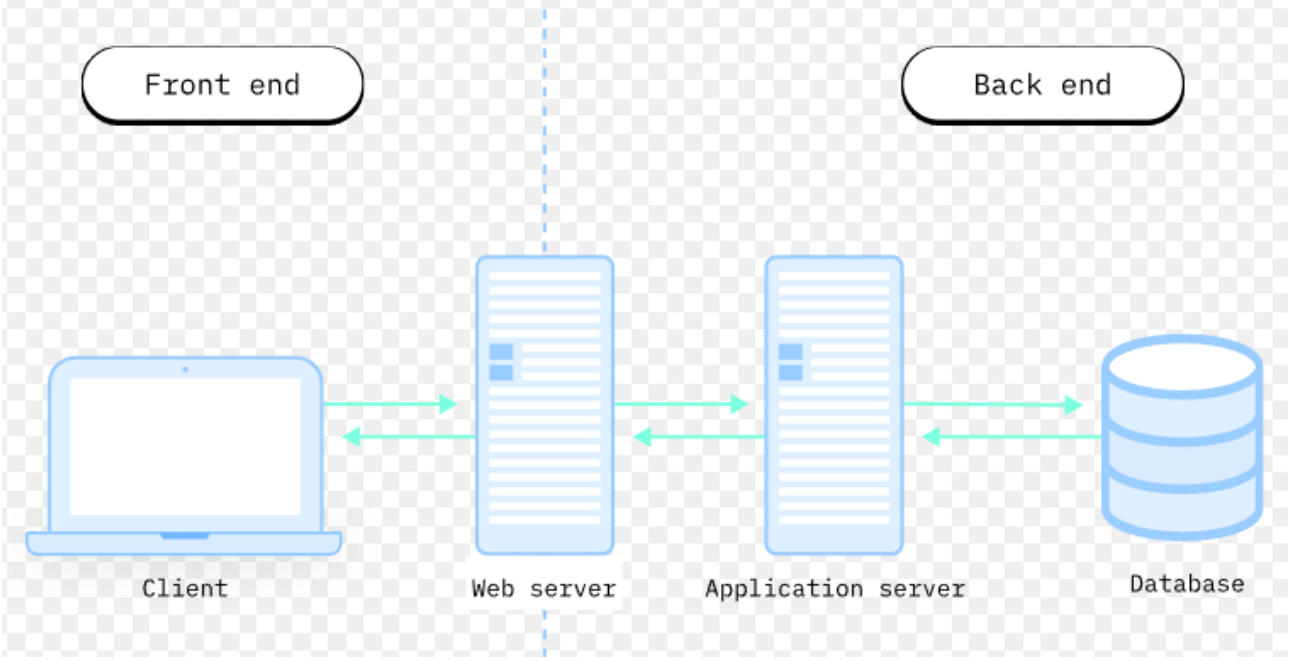


Рисунок 1.2 – Схема роботи fullstack додатку

Цей підхід передбачає розділення системи на кілька окремих компонентів: клієнтську частину (Client), WEB-сервер (WEB Server), сервер додатків (Application Server) та базу даних (Database). Клієнт надсилає запити до WEB-сервера, який відповідає за їх прийом і передачу серверу додатків для обробки бізнес-логіки. Сервер додатків взаємодіє з базою даних для збереження або отримання необхідної інформації. Після обробки дані повертаються через WEB-сервер до клієнта, що забезпечує розподіл функцій, гнучкість і масштабованість системи.

Найбільш сучасним і перспективним підходом у контексті розробки складних WEB-додатків є використання Headless-архітектури. У такій моделі CMS або інша система управління контентом функціонує виключно як backend-сервіс для зберігання та керування даними, які передаються через API. Логіка відображення інформації та взаємодії з користувачем реалізується у frontend-додатку окремо за допомогою спеціалізованих фреймворків. Це дозволяє досягти мультиплатформності: один і той самий API може обслуговувати як WEB-версію додатку, так і мобільний застосунок, а також інші клієнтські сервіси. Додатковою перевагою є те, що компоненти системи можуть масштабуватися незалежно один від одного, що суттєво полегшує обслуговування та розвиток проєкту.

Окрім цього, використання статичного рендерингу (SSG) або серверного рендерингу (SSR) дозволяє значно підвищити продуктивність системи, зменшити час завантаження сторінок та покращити загальну взаємодію користувача з вебресурсом. Завдяки попередньому рендерингу вмісту або його генерації на сервері лише за запитом, знижується навантаження на клієнтську частину та оптимізується споживання ресурсів. Важливою особливістю такого підходу є також підвищена безпека: CMS або інші backend-системи залишаються недоступними ззовні, оскільки взаємодія з ними відбувається виключно через заздалегідь налаштовані, захищені API. Це мінімізує ризики несанкціонованого доступу до адміністративної панелі або баз даних і створює додатковий захисний бар’єр у структурі додатку. На рисунку 1.3 зображено схему роботи Headless додатку.

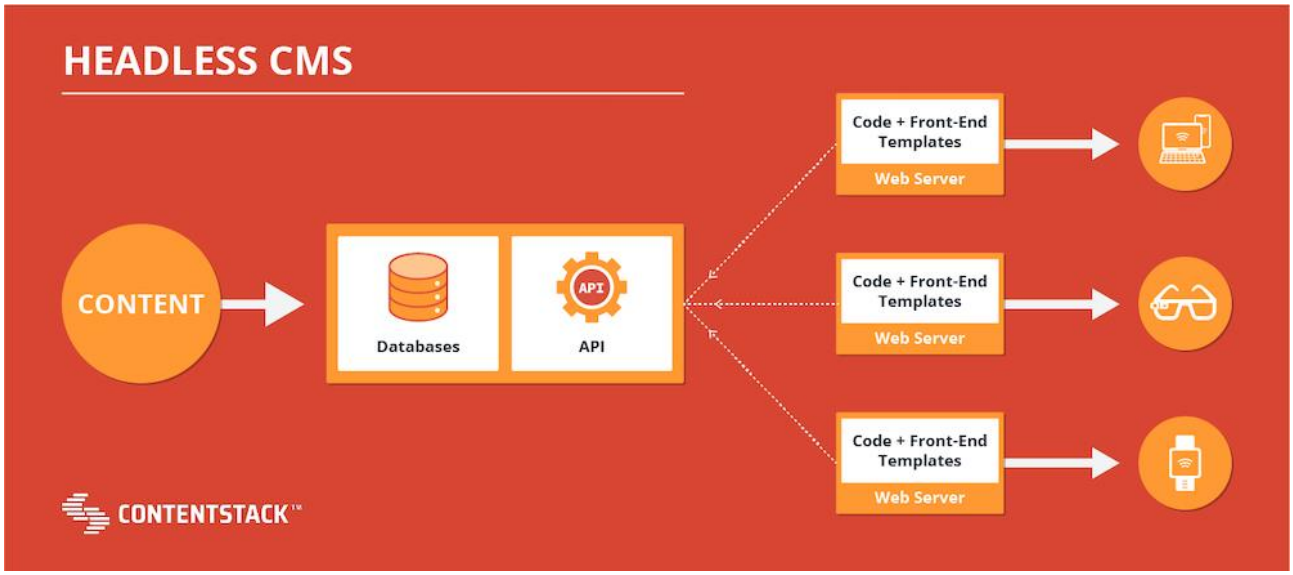


Рисунок 1.3 – Схема роботи Headless додатку

Headless-додаток базується на розділенні backend і frontend. CMS або інша система керування контентом виступає виключно як сервер даних, який через API передає контент зовнішньому frontend. Frontend, створений за допомогою сучасних JavaScript-фреймворків, відповідає за відображення інформації та взаємодію з користувачем. Така архітектура дозволяє забезпечити високу гнучкість, масштабованість і мультиплатформність.

1.3 Постановка задачі та формулювання вимог

Для успішної та якісної реалізації WEB-додатку необхідно чітко визначити його функціональні характеристики. Важливо сформулювати вимоги до продуктивності, стабільності, зручності та адаптивності інтерфейсу, а також забезпечити відповідність сучасним стандартам WEB-розробки.

Адміністративна частина додатку має включати інтерфейс для створення, редагування та публікації статей і сторінок, що дозволить реалізувати повноцінний блоговий функціонал. Має бути забезпечена можливість створення списку публікацій у вигляді окремої сторінки блогу з посторінковою навігацією, а також перегляду окремих статей на сторінці поста. Адміністратор повинен мати змогу додавати та оновлювати графічні матеріали, модерувати коментарі

користувачів, інтегрувати аналітичні системи та маркетингові інструменти без потреби змінювати програмний код. Адміністратор має також отримувати сповіщення щодо застарілого програмного забезпечення або потенційних вразливостей, що впливають на стабільність і безпеку додатку.

Для кінцевого користувача важливим є створення адаптивного інтерфейсу, який забезпечує коректне відображення на широкому спектрі пристроїв – від настільних комп'ютерів до мобільних телефонів. Інтерфейс повинен бути інтуїтивно зрозумілим і відповідати сучасним стандартам доступності, зокрема забезпечувати навігацію з клавіатури, підтримувати масштабування шрифтів і мати оптимальні контрастні режими. Окремо слід передбачити сторінку з повідомленням про помилку 404, яка допомагає користувачеві повернутися до основних розділів сайту або на головну сторінку. Необхідно також реалізувати функціонал пошуку для швидкого доступу до потрібного контенту.

До загальних вимог входить забезпечення високої швидкості завантаження сторінок, яка не повинна перевищувати дві секунди за умов нормального навантаження. При цьому додаток має залишатися стабільним і при пікових навантаженнях, підтримуючи коректну роботу при значній кількості одночасних користувачів. Крім того, важливо дотримуватися вимог SEO-оптимізації: структура заголовків, наявність мета-тегів, правильна генерація файлів sitemap.xml і robots.txt мають відповідати актуальним рекомендаціям пошукових систем.

Усі зазначені вимоги мають бути враховані у процесі розробки та підтверджені під час тестування відповідно до визначених сценаріїв.

1.4 Висновок до розділу 1

У першому розділі було проведено ґрунтовний аналіз предметної області розробки сучасних WEB-додатків, зокрема із використанням Headless-архітектури та фреймворків нового покоління. Визначено основні виклики та

проблеми, пов’язані зі швидкодією та гнучкістю WEB-інтерфейсів, які сьогодні є критично важливими факторами для забезпечення якісного користувацького досвіду.

Було розглянуто еволюцію підходів до розробки: від традиційної монолітної архітектури CMS до Headless-моделі, яка дозволяє розділити backend і frontend, забезпечуючи більшу масштабованість, продуктивність та можливість мультиплатформенного використання контенту. Окрему увагу було приділено аналізу механізмів оптимізації рендерингу DOM, зокрема завдяки використанню Virtual DOM у фреймворку React, що дає змогу мінімізувати час відгуку додатку та підвищити його інтерактивність.

В огляді аналогів були розглянуті переваги та обмеження різних підходів до розробки WEB-додатків для CMS, включно з традиційними рішеннями на базі PHP-шаблонів, повноцінними Fullstack-архітектурами та Headless-рішеннями. Показано, що саме Headless-підхід надає найбільші можливості для створення сучасних, гнучких, продуктивних та безпечних додатків.

У результаті було сформульовано чіткі функціональні вимоги до майбутнього WEB-додатку, що розроблятиметься у рамках даної роботи. Основними орієнтирами є: висока швидкодія, адаптивний інтерфейс, розширюваність системи, інтеграція з аналітичними та SEO-інструментами, а також забезпечення якісного користувацького та адміністративного досвіду.

Таким чином, аналіз предметної області та існуючих аналогів підтвердив доцільність використання Headless-архітектури у поєднанні з React для реалізації WEB-додатку. Це дозволить створити конкурентоспроможний та інноваційний продукт, що відповідатиме сучасним вимогам ринку.

2 ПРОЕКТУВАННЯ WEB-ДОДАТКУ ДЛЯ CMS З ВИКОРИСТАННЯМ HEADLESS ПІДХОДУ

2.1 Вибір архітектури WEB-додатку для CMS з використанням Headless підходу

Важливою складовою при проєктуванні WEB-додатку є вибір архітектури, на якій базуватиметься система. Саме архітектурний підхід визначає гнучкість, масштабованість, продуктивність і простоту підтримки та обслуговування проєкту. На сьогодні найбільш поширеними є три підходи: монолітна, мікросервісна та сервісоорієнтована архітектура (SOA).

При побудові архітектури використовуються архітектурні шаблони, що дозволяють використовувати сучасні та перевірені практики для вирішення різних архітектурних проблем.

Монолітна архітектура передбачає створення програмного забезпечення у вигляді єдиного цілісного блоку, у межах якого всі складові системи інтерфейс користувача, бізнес-логіка, а також система управління базами даних функціонують у спільному кодовому просторі та часто розгортаються одночасно як єдине ціле. Усі модулі взаємодіють між собою без необхідності додаткових комунікаційних протоколів, що значно спрощує їхню координацію та синхронізацію.

Такий підхід характеризується високим ступенем інтеграції між окремими частинами додатку, що забезпечує швидку, пряму та ефективну взаємодію між ними. Завдяки цьому можна досягти певної оптимізації продуктивності, особливо в умовах обмежених ресурсів, та спрощення початкового етапу розробки, коли важливо швидко реалізувати базову функціональність. Крім того, розробка та деплой монолітної системи часто є менш затратними з точки зору інфраструктури та DevOps-процесів.

Водночас, тісна взаємопов’язаність компонентів у межах монолітної архітектури знижує гнучкість системи: будь-які зміни в одному з модулів можуть мати непередбачувані наслідки для інших частин програми, що ускладнює процес оновлення та тестування. Масштабування таких систем також є нетривіальним завданням, оскільки не можна незалежно масштабувати окремі компоненти – доводиться масштабувати всю систему в цілому.

Даний підхід має низку переваг:

– чітке уявлення про всі взаємозв’язки додатку;

– маленька кількість зовнішніх залежностей;

– централізоване управління [5].

На рисунку 2.1 зображено схему роботи монолітної архітектури.

Рисунок 2.1 – Зображення схеми роботи монолітної архітектури

На рисунку зображено типовий приклад монолітної архітектури. Усі основні частини WEB-додатку – інтерфейс користувача (UI), бізнес-логіка (Business Logic) та рівень доступу до даних (Data Access Layer) – об’єднані в один єдиний програмний блок. Вони функціонують як частини одного додатку та безпосередньо взаємодіють між собою:

– UI (User Interface) відображає дані користувачу та приймає взаємодії. Всі запити з інтерфейсу обробляються у межах цього самого блоку;

– Business Logic – обробляє основні правила та сценарії роботи системи. Вона напряму пов’язана як з інтерфейсом, так і з рівнем доступу до даних;

– Data Access Layer – відповідає за зчитування, збереження та зміну інформації у базі даних. Вся робота з даними відбувається через нього, без потреби в окремих сервісах [6].

Мікросервісна архітектура (Microservices Architecture, MSA) є сучасним підходом до створення складних програмних систем, що передбачає побудову додатків на основі набору незалежних модулів – так званих мікросервісів. Кожен мікросервіс реалізує окреме бізнес-завдання або обслуговує певний функціональний блок системи, що дає змогу розглядати його як самостійний програмний компонент.

Однією з ключових характеристик мікросервісної архітектури є те, що кожен мікросервіс розробляється, тестується, оновлюється та розгортається незалежно від інших модулів. Завдяки такій автономності стає можливим внесення змін до окремих частин системи без впливу на її загальну працездатність або стабільність. Це дозволяє гнучко масштабувати окремі сервіси у відповідь на зміну навантаження, що особливо актуально для систем із високою динамікою розвитку та великою кількістю користувачів.

Крім того, мікросервіси можуть бути реалізовані різними командами з використанням різних технологічних стеків, оскільки вони взаємодіють між собою через стандартизовані інтерфейси, зазвичай на основі протоколів REST або gRPC. Це відкриває широкі можливості для організації паралельної розробки та забезпечує високу ступінь гнучкості у виборі технологічних рішень.

Мікросервісна архітектура активно використовується у розробці сучасних масштабованих WEB-додатків, оскільки дозволяє ефективно керувати складністю проєктів, полегшує впровадження нових функціональних можливостей і значно спрощує підтримку та розвиток систем у довгостроковій перспективі [7].

Даний підхід має низку переваг:

– масштабованість. Кожен сервіс в архітектурі працює незалежно, тому можливо масштабувати лише ті компоненти, які цього потребують. Це дозволяє ефективніше використовувати ресурси та знижує витрати на інфраструктуру;

– гнучкість у виборі технологій. Для кожного мікросервісу можна обрати найбільш підходящу мову програмування, базу даних або фреймворк. Це дає змогу реалізовувати найефективніші технічні рішення для конкретного завдання;

– простота підтримки та розгортання. Оскільки мікросервіси розробляються окремо, зміни в одному модулі не впливають на інші. Це дозволяє оновлювати, виправляти або перевипускати окремі частини системи без повного перезапуску всього додатку;

– підвищена надійність. Якщо один сервіс виходить з ладу, інші продовжують працювати. Це знижує ризик повного відключення системи та забезпечує кращу відмовостійкість.

На рисунку 2.2 зображено схему мікросервісної архітектури.

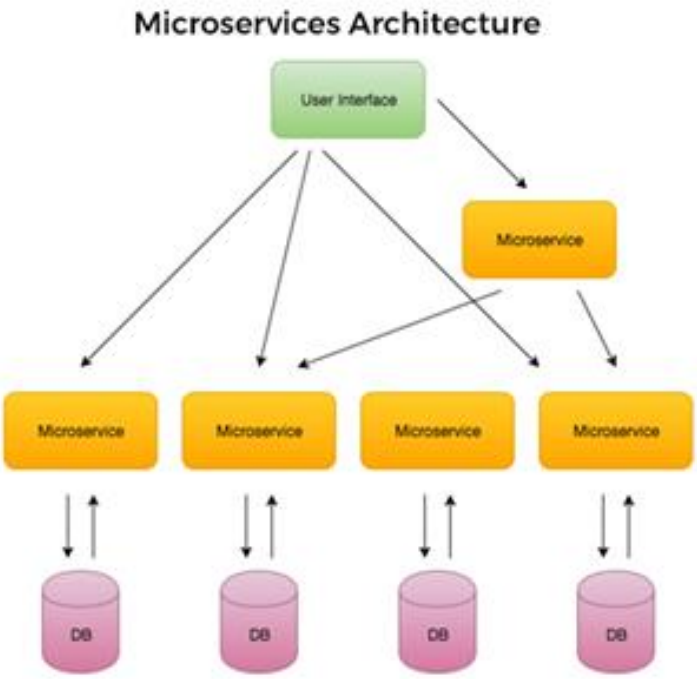


Рисунок 2.2 – Зображення схеми роботи мікросервісної архітектури

На схемі зображено мікросервісну архітектуру, в якій інтерфейс користувача взаємодіє з окремими мікросервісами, кожен з яких виконує свою вузьку функцію. Кожен мікросервіс має власну окрему базу даних і працює незалежно від інших. Взаємозв’язки між сервісами дозволяють обмінюватися даними або координувати дії, але при цьому всі частини системи залишаються

автономними. Така побудова дає змогу масштабувати систему поступово, оновлювати чи замінювати окремі сервіси без зупинки роботи всього застосунку.

Оскільки підхід CMS базується на монолітній архітектурі, саме її ми обираємо для подальшого розгляду.

Для подальшого планування та впровадження WEB-додатку необхідно створити UML-діаграму, яка надасть чітке уявлення про структуру системи та взаємодію її компонентів. Це дозволить детально відобразити бізнес-логіку, розмежувати ролі та функції, а також оптимізувати процес розробки за рахунок візуалізації основних процесів і потоків даних у системі. На рисунку 2.3 зображено UML-діаграму компонентів додатку.

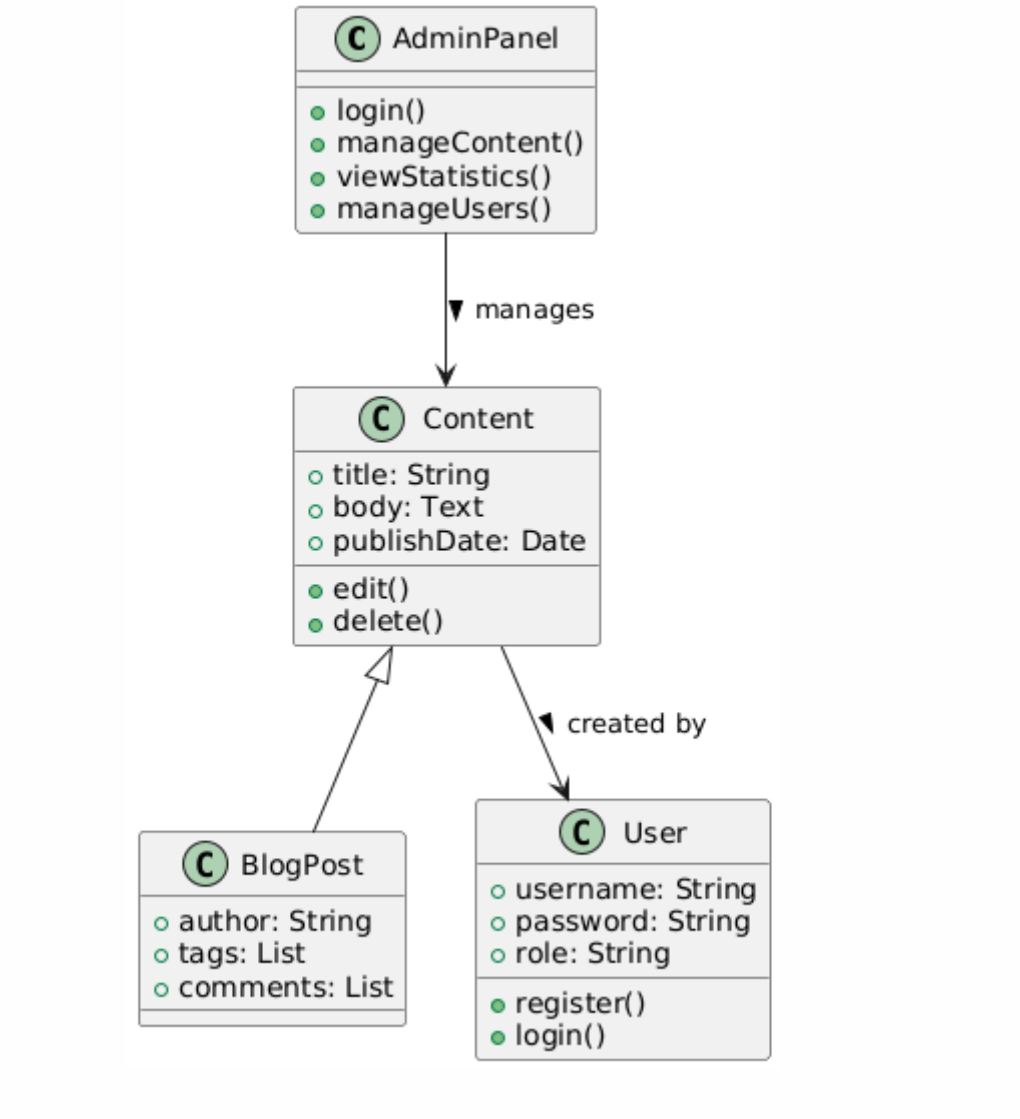


Рисунок 2.3 – Зображення UML-діаграми компонентів додатку

AdminPanel – це компонент, який відповідає за управління WEB-додатком з боку адміністратора. Тут реалізовано функції для авторизації, редагування контенту, моніторингу активності сайту та налаштування безпеки. Через цей блок адміністратор має змогу контролювати публікації, керувати користувачами та оновлювати інформацію, забезпечуючи стабільну роботу системи.

Content – основний блок, що описує структуру контенту на сайті. Він містить базові елементи, такі як заголовок, текст, дату публікації, і слугує шаблоном для більш спеціалізованих типів інформації. Цей клас відповідає за збереження і обробку даних, які відображаються у WEB-додатку.

BlogPost – розширення базового класу Content, яке адаптоване під специфіку блогових матеріалів. Окрім основних атрибутів, цей блок включає інформацію про автора, ключові слова для категоризації, а також механізми роботи з коментарями користувачів. Це дозволяє реалізувати повноцінний функціонал блогу в системі.

User – компонент, що описує користувачів WEB-додатку, їх облікові дані та ролі. Він відповідає за процеси реєстрації, входу до системи та контроль доступу до різних частин ресурсу. Завдяки цьому блоку забезпечується безпека та персоналізація взаємодії користувачів із сайтом.

У межах представленої архітектури кожен компонент виконує специфічні функції, що реалізуються через відповідні методи. Компонент AdminPanel відповідає за адміністративне управління системою. Він включає метод login(credentials), який здійснює перевірку облікових даних адміністратора, createContent(data) для створення нового контенту, editContent(id, data) для редагування наявного контенту, deleteContent(id) для його видалення, viewStatistics() для отримання статистичних даних сайту та manageUsers() для адміністрування користувачів і керування їхніми ролями.

Компонент Content призначений для роботи з контентом сайту. До його функцій належать метод getContent(), що повертає дані для відображення, updateContent(data) для оновлення інформації, publish() для публікації контенту та archive() для його архівування.

Компонент BlogPost, який є підкласом Content, розширює його функціональність додатковими методами. Метод addComment(comment) реалізує додавання користувацьких коментарів, getComments() забезпечує отримання списку коментарів до публікації, а tagPost(tags) дозволяє додавати ключові слова для категоризації вмісту.

Компонент User відповідає за взаємодію із зареєстрованими користувачами. Метод register(data) забезпечує реєстрацію нового користувача, login(credentials) – вхід до системи, logout() – вихід, updateProfile(data) – оновлення профільної інформації користувача, а метод checkPermissions() дозволяє перевірити рівень доступу до ресурсів системи.

2.2 Аналіз та вибір технологій розробки

Під час розробки будь-якого продукту важливим етапом є правильний вибір технологій. Саме від обраного стеку залежить стабільність роботи системи, її продуктивність, масштабованість а також комфорт безпосердньо розробника. Правильно обравши стек та архітектуру, можна заощадити чимало часів та ресурсів, а також уникнути багатьох проблем, що можуть бути пов’язані з масштабуванням або інтеграцією зі сторонніми сервісами.

Проте, варто звернути увагу на використання трендових, але нестабільних технологій, що знаходяться у ранньому доступі. Часто функціонал даних технологій не є в повній мірі добре задокументованими, що спричинить потребу у більшій кількості часу та зусиль для входження у технології. Приховані технічні обмеження можуть проявитись після запуску. Аби уникнути даних проблем, потрібно орієнтуватись перевірені інструменти, які активно підтримуються компанією-власником чи спільнотою, мають широку кількість користувачів та гарну документацію.

Вибір технологій розпочнемо з backend-частини, адже саме там зосереджується основна бізнес-логіка системи, обробка запитів, керування

контентом та інтеграція зі сторонніми сервісами. На цьому етапі необхідно визначитися з мовою програмування, що стане основою для розробки.

Для обґрунтованого вибору слід провести детальний аналіз популярних мов програмування, їх можливостей, продуктивності, екосистеми та відповідності вимогам проєкту. Згідно зі статистикою порталу kinsta.com [8], PHP використовується приблизно на 80% усіх WEB-сайтів, що свідчить про його широку поширеність, наявність великої кількості готових рішень та розвинену спільноту. Водночас, серед серверних мов програмування варто виділити Java, яка є однією з найпопулярніших у корпоративному середовищі. Java відома своєю стабільністю, масштабованістю та високою продуктивністю, що робить її оптимальним вибором для складних, масштабних WEB-додатків із великим навантаженням. Крім того, Java має потужну екосистему з численними фреймворками та інструментами, які полегшують розробку та підтримку проєктів різного рівня складності.

PHP – мова програмування загального призначення, що використовується для розробки WEB-орієнтованого програмного забезпечення. На сьогодні PHP є лідером серед мов програмування, що використовуються для створення динамічних WEB-сайтів. PHP використовується для розробки проектів різних масштабів – від найпростіших сайтів до великих багатофункціональних WEB-сервісів. Ця мова програмування має ряд значних переваг, які роблять її основним інструментом для розробки серверної частини додатків та сайтів.

PHP залишається однією з найпоширеніших мов програмування у сфері WEB-розробки, що зумовлено низкою технічних і економічних переваг. Однією з ключових причин популярності цієї мови є її широка підтримка серед сучасних хостинг-провайдерів. У більшості випадків PHP-підтримка надається за замовчуванням, що дозволяє зменшити вартість розгортання WEB-додатку або сайту. Такий фактор суттєво впливає на зниження бар’єру входження для розробників та підприємців, які працюють із обмеженим бюджетом.

Ще однією істотною перевагою PHP є розвинена екосистема систем управління контентом (CMS), створених на основі цієї мови. Серед них варто

відзначити такі універсальні CMS, як WordPress, Joomla, Drupal, які підходять для створення сайтів загального призначення, а також OpenCart і Magento – рішення, орієнтовані на електронну комерцію. Зазначені системи охоплюють лише найбільш поширену частину доступного програмного забезпечення, тоді як загальна кількість CMS на основі PHP, як безкоштовних, так і комерційних, є значною. Це забезпечує розробникам широкий вибір інструментів та шаблонів, що спрощує створення та підтримку WEB-додатків різної складності.

Окрім широкої підтримки та багатої екосистеми, сучасні версії PHP демонструють високу швидкість виконання серверних скриптів. Показники продуктивності у нових релізах значно покращилися, що забезпечує ефективну роботу додатків навіть при зростанні навантаження. Така продуктивність у поєднанні з простотою використання та доступністю інструментів робить PHP привабливим вибором для реалізації як малих, так і масштабованих WEB-проєктів.

Натомість Java – це об’єктно-орієнтована мова програмування загального призначення з простим і зрозумілим синтаксисом, яка підходить для різних платформ. Найчастіше нею пишуть backend (серверну частину софту). Ось ключові особливості Java:

– жорстка типізація. У Java всі змінні мають бути оголошені із зазначенням їхнього типу. Це запобігає помилкам типізації, дає змогу писати зрозумілий код і знаходити баги на етапі компіляції, а не виконання.

– автоматичне керування пам’яттю. У мові програмування Java реалізовано автоматичне збирання сміття. Вона звільняє пам’ять, зайняту об’єктами, що більше не використовуються. Розробникам не потрібно робити це власноруч.

– об’єктно-орієнтованість. Java ґрунтується на концепції об’єктно-орієнтованого програмування. Це означає, що все в Java є об’єктом, який має свої властивості та методи. Об’єктно-орієнтований підхід дає можливість Java-розробникам створювати модульні, гнучкі та безпечні застосунки [9].

На таблиці 2.1 зображена порівняльна характеристика мов програмування Java та PHP.

Таблиця 2.1 – Порівняльна характеристика мов програмування Java та PHP.

Критерій	**PHP**	**Java**
Призначення	Спеціалізована серверна мова для вебу	Загального призначення, універсальна
Простота освоєння	Відносно проста, швидкий старт	Складніша, потребує глибшого розуміння ООП
Розробка веб-сайтів	Оптимізована під веб, інтеграція з HTML	Потребує додаткових фреймворків (Spring, etc.)
Екосистема та інструменти	Величезна кількість CMS, фреймворків, бібліотек (WordPress, Laravel)	Багато фреймворків, але складніші у налаштуванні
Продуктивність	Достатня для більшості веб-додатків	Вища продуктивність у великих системах
Хостинг та доступність	Широка підтримка у будь-якого хостинг-провайдера, дешевший	Менш поширений, вимагає спеціалізованих серверів
Вартість розробки	Нижча завдяки великій кількості готових рішень та доступним розробникам	Вища через складність і потребу в кваліфікованих кадрах
Гнучкість та масштабованість	Добре підходить для малого та середнього бізнесу, але має обмеження для дуже великих систем	Відмінно масштабуються, підходить для великих корпоративних систем
Підтримка спільноти	Дуже велика та активна спільнота	Також велика, але більше орієнтована на корпоративний сектор
Безпека	Вимагає обережності через історію вразливостей, але сучасні фреймворки допомагають	Вбудовані механізми безпеки на високому рівні

Враховуючи наведені порівняння, PHP є оптимальним вибором для більшості WEB-проєктів, особливо коли мова йде про швидку розробку та гнучке управління контентом. Його спеціалізація саме на серверній стороні WEB, простота у вивченні та величезна екосистема готових рішень значно

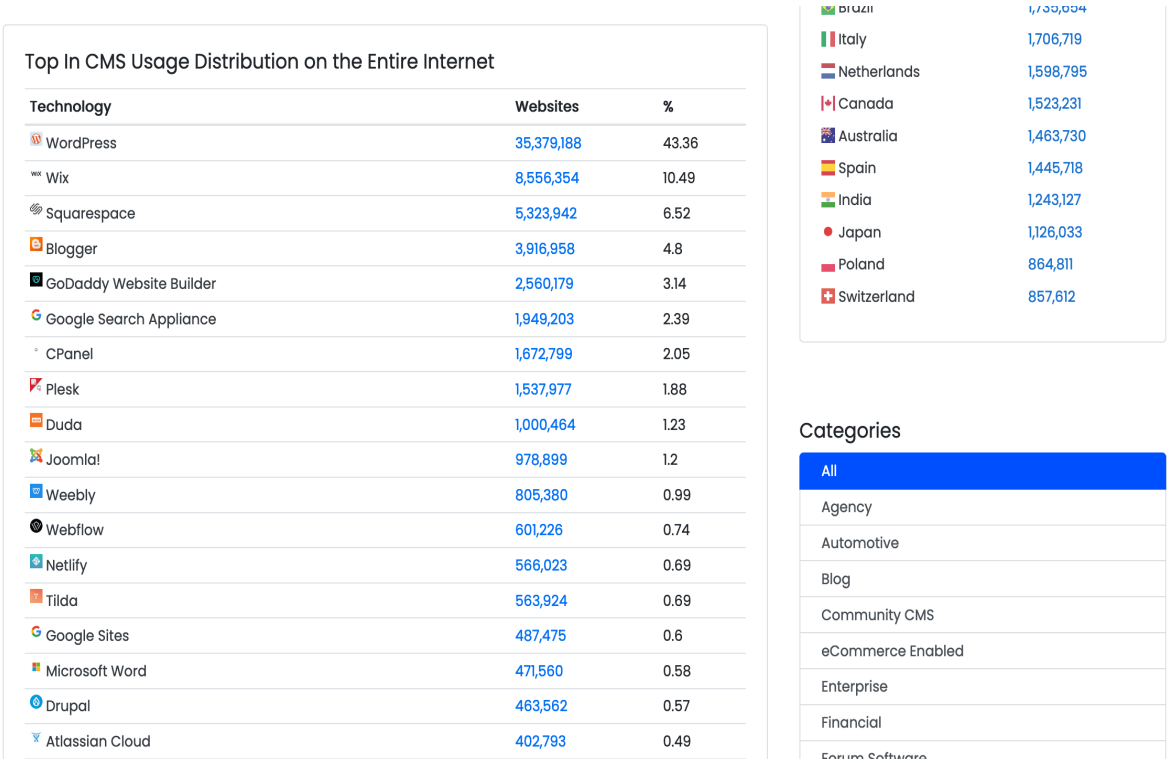
скорочують час на запуск та подальшу підтримку додатків. Доступність великої кількості фреймворків і систем управління контентом, зокрема WordPress – найбільш популярної CMS у світі – дозволяє розробникам ефективно використовувати вже перевірені інструменти.

В той час як Java відома своєю високою продуктивністю, безпекою та масштабованістю, її складність, дорожчі ресурси розробників та необхідність налаштування складних середовищ роблять її менш привабливою для невеликих та середніх проєктів. Java найкраще підходить для великих корпоративних систем з високими вимогами до надійності та масштабування, але для багатьох типових веб-додатків це надмірний інструмент.

З огляду на те, що проєкт передбачає створення WEB-додатку з функціоналом управління контентом, блогом, адаптивним інтерфейсом і можливістю швидкої кастомізації, вибір PHP стає очевидним. Це рішення забезпечує оптимальне співвідношення продуктивності, швидкості розробки та вартості підтримки. Саме тому для реалізації системи керування контентом ми обираємо PHP, який також слугує базою для популярної CMS WordPress, що є перевіреним інструментом із багатою функціональністю та широкою підтримкою спільноти.

WordPress – система керування вмістом з відкритим кодом, яка через свою простоту в установленні та використанні широко застосовується для конструювання у WEB. Сфера використання – від блогів до складних WEB-додатків. Вбудована система тем і плагінів у поєднанні з вдалою архітектурою дозволяє конструювати на основі WordPress практично будь-які проєкти [2].

За даними агентства BuiltWith CMS WordPress є найпоширенішою у світі та налічує близько 82 мільйонів сайтів, що наведено на рисунку 2.4.



Top In CMS Usage Distribution on the Entire Internet

Technology	Websites	%
WordPress	35,379,188	43.36
Wix	8,556,354	10.49
Squarespace	5,323,942	6.52
Blogger	3,916,958	4.8
GoDaddy Website Builder	2,560,179	3.14
Google Search Appliance	1,949,203	2.39
CPanel	1,672,799	2.05
Plesk	1,537,977	1.88
Duda	1,000,464	1.23
Joomla!	978,899	1.2
Weebly	805,380	0.99
Webflow	601,226	0.74
Netlify	566,023	0.69
Tilda	563,924	0.69
Google Sites	487,475	0.6
Microsoft Word	471,560	0.58
Drupal	463,562	0.57
Atlassian Cloud	402,793	0.49

Italy	1,706,719
Netherlands	1,598,795
Canada	1,523,231
Australia	1,463,730
Spain	1,445,718
India	1,243,127
Japan	1,126,033
Poland	864,811
Switzerland	857,612

Categories

All
Agency
Automotive
Blog
Community CMS
eCommerce Enabled
Enterprise
Financial

Рисунок 2.4 – Розподіл CMS за даними агентства BuiltWith

Обрана CMS вигідно вирізняється серед аналогів завдяки двом ключовим перевагам: відкритому вихідному коду та потужній спільноті розробників. Це створює надійну основу для розробки, оскільки більшість технічних проблем уже були виявлені й описані іншими користувачами. У більшості випадків готові рішення або обговорення можна знайти на спеціалізованих форумах, у репозиторіях чи в офіційній документації.

Відкритість коду, у свою чергу, забезпечує повну свободу кастомізації. Це дозволяє розширювати базову функціональність CMS під конкретні потреби проєкту, інтегрувати нестандартні модулі та реалізовувати унікальну бізнес-логіку. Такий підхід є основою для ефективного масштабування та адаптації системи в майбутньому.

До інших переваг WordPress можна додати:

– модулі для підключення (плагіни) з унікально простою системою їх взаємодії з кодом; можливість автоматичного встановлення та оновлення версії безпосередньо з панелі адміністратора;

– наявність ЛЗУ (людино-зрозумілий URL);

– SEO-оптимізована система [2];

– простота установки та перенесення.

– WordPress використовує мову програмування PHP що дозволяє не купувати спеціальний виділений хостинг. Досить будь-якого PHP хостингу з хорошим рейтингом. Тому, що цей движок повністю безкоштовний вам не потрібно буде платити за його використання [3].

Оскільки WordPress розроблений мовою програмування PHP, доцільно розглянути її переваги. Дана мова програмування створювалась виключно для WEB-розробки і довела свою доцільність у використанні на мільйонах проєктів.

Наступним важливим етапом став вибір бази даних, яка забезпечуватиме зберігання, обробку та взаємодію з усіма типами контенту. Оскільки в якості серверної частини було обрано WordPress, що історично працює на стеку LAMP (Linux, Apache, MySQL, PHP), – найбільш логічним і сумісним рішенням є використання реляційної бази даних MySQL або її форку MariaDB.

Це рішення зумовлене не лише сумісністю, а й практичністю: MySQL є стабільною, добре масштабованою системою з широкою підтримкою інструментів для резервного копіювання, міграції, моніторингу та оптимізації. Крім того, більшість плагінів, тем і інструментів для WordPress побудовані саме під структуру MySQL, що спрощує інтеграцію та знижує ризик конфліктів на рівні даних.

Таким чином, вибір MySQL як бази даних для цього проєкту забезпечує повну сумісність з обраною CMS, стабільність у роботі, легкість у масштабуванні та перевірену часом надійність.

Обираючи frontend інструмент потрібно враховувати, що сучасний WEB-додаток повинен підтримувати підхід SPA.

Single Page Application (SPA) – це тип WEB-додатку, в якому вся інтерактивність і відображення даних відбувається на одній WEB-сторінці, без необхідності завантаження додаткових сторінок або перезавантаження браузера;

У SPA всі компоненти, зазвичай керуються через JavaScript-фреймворки та бібліотеки, як-от React, Angular, Vue.js та інші. Ці фреймворки забезпечують

модульну архітектуру і можливість створення компонентів, які можна використовувати повторно [4]. На рисунку 2.5 наведено відмінність між класичної архітектурою взаємодії клієнт-серверних підходів та SPA.

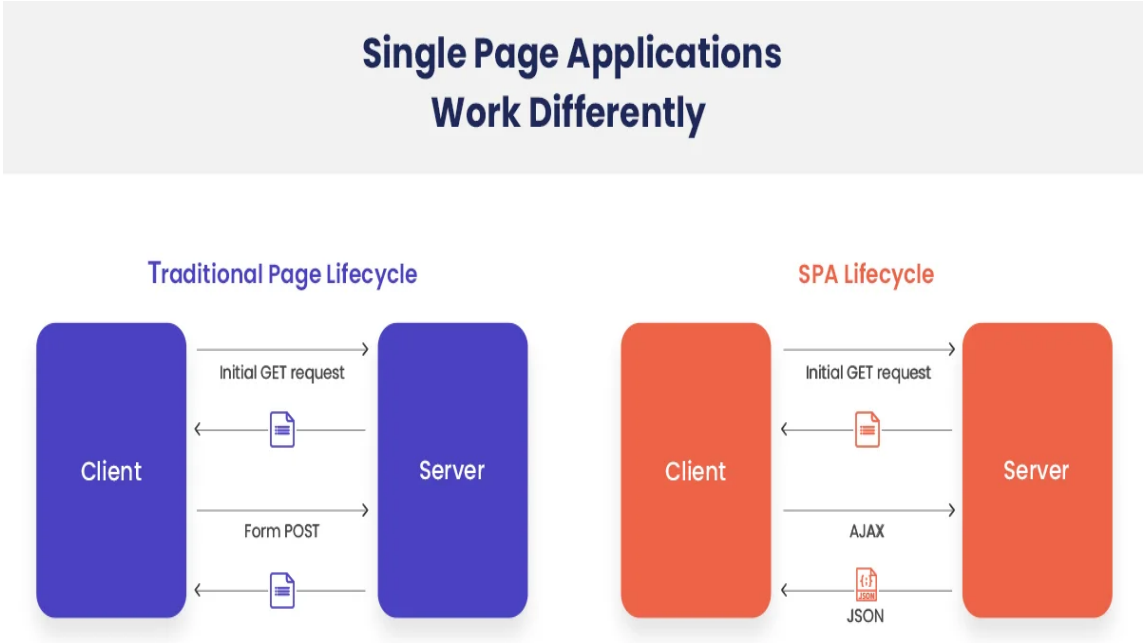


Рисунок 2.5 – відмінність між класичної архітектурою взаємодії клієнт-серверних підходів та SPA

2.3 Висновок до розділу 2

У даному розділі було проведено огляд існуючих технологій і засобів розробки, які застосовуються для створення систем керування контентом. Розглянуто різні підходи до архітектури CMS, зокрема класичні монолітні системи та Headless CMS, а також проаналізовано їх переваги та недоліки. Особливу увагу приділено серверній частині системи та вибору мови програмування й фреймворків. За результатами аналізу обґрунтовано доцільність використання монолітної архітектури із застосуванням PHP як основної мови розробки, та CMS WordPress що забезпечить необхідну функціональність, гнучкість та ефективність майбутньої системи.

3 РОЗРОБКА WEB-ДОДАТКУ ДЛЯ CMS З ВИКОРИСТАННЯМ HEADLESS ПІДХОДУ

3.1 Розробка backend частини

Безпосередню розробку WEB-додатку розпочнемо із підготовки CMS та її встановлення. Завдяки своїй поширеності, CMS WordPress можна встановлювати завдяки попердньо підготовленим скриптам (тригерам), що автоматично завантажують останню стабільну версію мови програмування PHP, СУБД MySQL та конфігурацію серверу Nginx. Дані інструменти йдуть на будь-якому хостингу “із коробки”, тож у розробника не виникає труднощів з їх подальшою конфігурацією.

Оскільки додаток на початковому етапі розроблятиметься локально і лише після ретельного тестування буде перенесений на хостинг та опублікований у мережі Інтернет, необхідно обрати локальний сервер – середовище, що імітує роботу WEB-сервера. Для цього було обрано додаток Local**.** Він популярний завдяки високій швидкодії, можливості швидкої зміни версій PHP та підтримці SSL-сертифікатів.

SSL-сертифікат – це засіб захисту особистої інформації користувачів в інтернеті. Якщо на сайті є SSL-сертифікат, в адресному рядку WEB-переглядача з’явиться зелений замок і протокол HTTPS. У роботі SSL-сертифіката бере участь два типи шифрування: симетричне й асиметричне [10].

Симетричне – це коли один ключ зашифровує та розшифровує повідомлення. Асиметричне – коли є два різних ключі: публічний і приватний. Публічний лише зашифровує повідомлення, його бачить кожний WEB-переглядач. Приватний лише розшифровує та зберігається в таємниці на сервері.

Щоразу, коли на сайт заходить відвідувач, WEB-переглядач генерує унікальний симетричний ключ, зашифровує його публічним ключем і надсилає на сервер. Сервер звіряє приватний ключ із публічним і розшифровує

повідомлення. Цей процес займає кілька секунд [10]. Таким чином вирішується проблема безпеки додатку, адже зловмисник не зможе будь-яким чином підробити сертифікат та відповідно отримати доступ до кореневого каталогу сайту.

Крім того, наявність SSL-сертифіката ще на етапі локальної розробки полегшує подальше перенесення сайту в мережу Інтернет. Наприклад, алгоритми пошукової системи Google зчитують інформацію про наявність захищеного з'єднання (HTTPS) вже під час індексації. Якщо сайт спочатку не передбачає використання SSL-сертифіката, це може негативно вплинути на його позиції в пошуковій видачі. У деяких випадках такі сайти навіть позначаються як небезпечні**,** що суттєво знижує довіру користувачів і трафік. На рисунку 3.1 позначено процес розгортання CMS на локальному сервері у додатку Local.

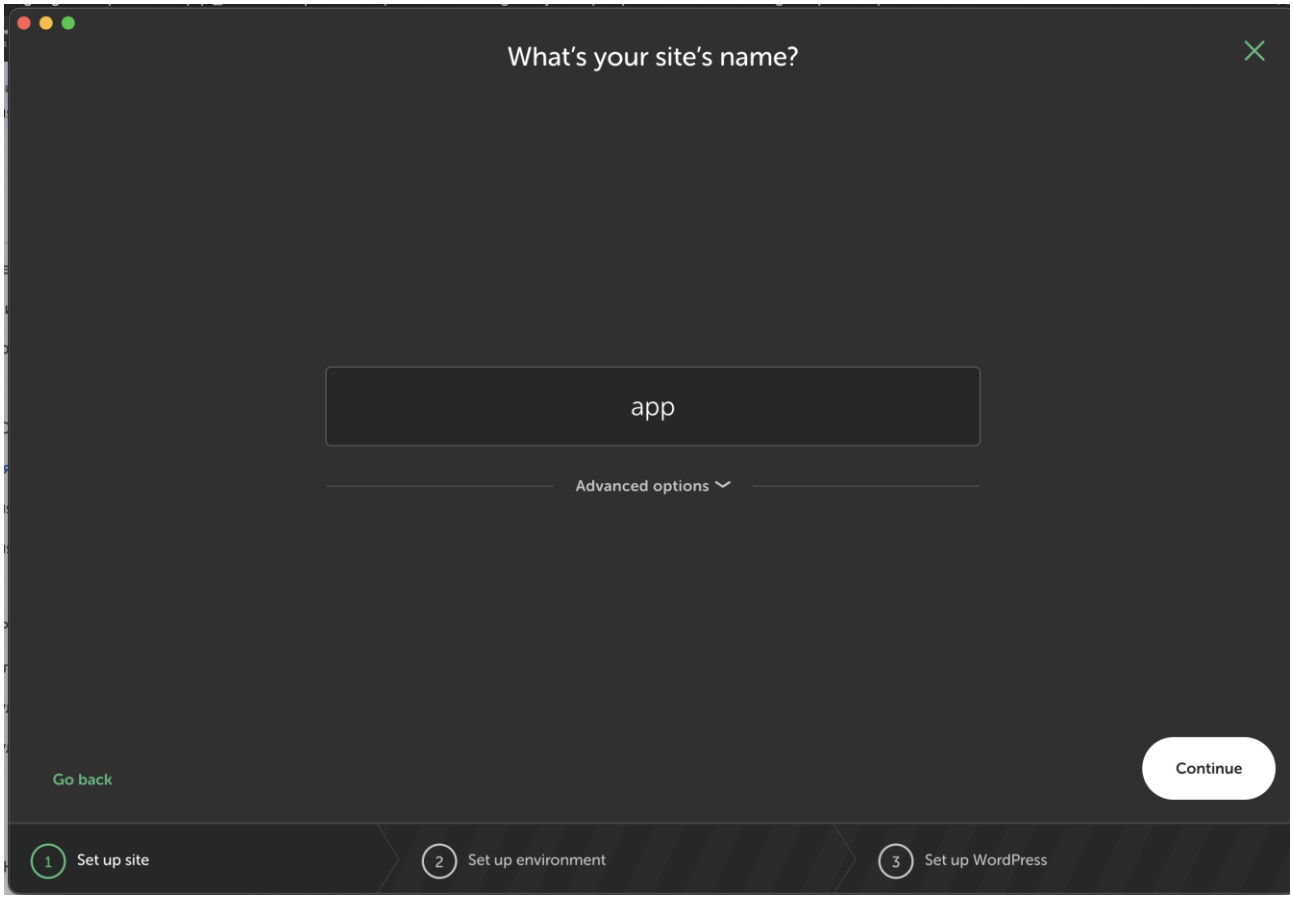


Рисунок 3.1 – Процес розгортання CMS WordPress на локальному сервері у додатку Local

Безпосередньо у першому випадаючому вікні розробнпку лише потрібно обрати назву сайту, що стане доменом. Саме завдяки домену можна буде потрапити на локальний сайт. У нашому випадку домен буде складатись із слова app та приписки local, що означатиме, що сайт розміщено локально. Отже, локальна адреса сайту – app.local.

Наступним кроком потрібно обрати серверне оточення, безпосередньо у розробників це часто позначається словом “environment”. На рисунку 3.2 зображено процес можливості вибору оточення: ручний, та автоматичний.

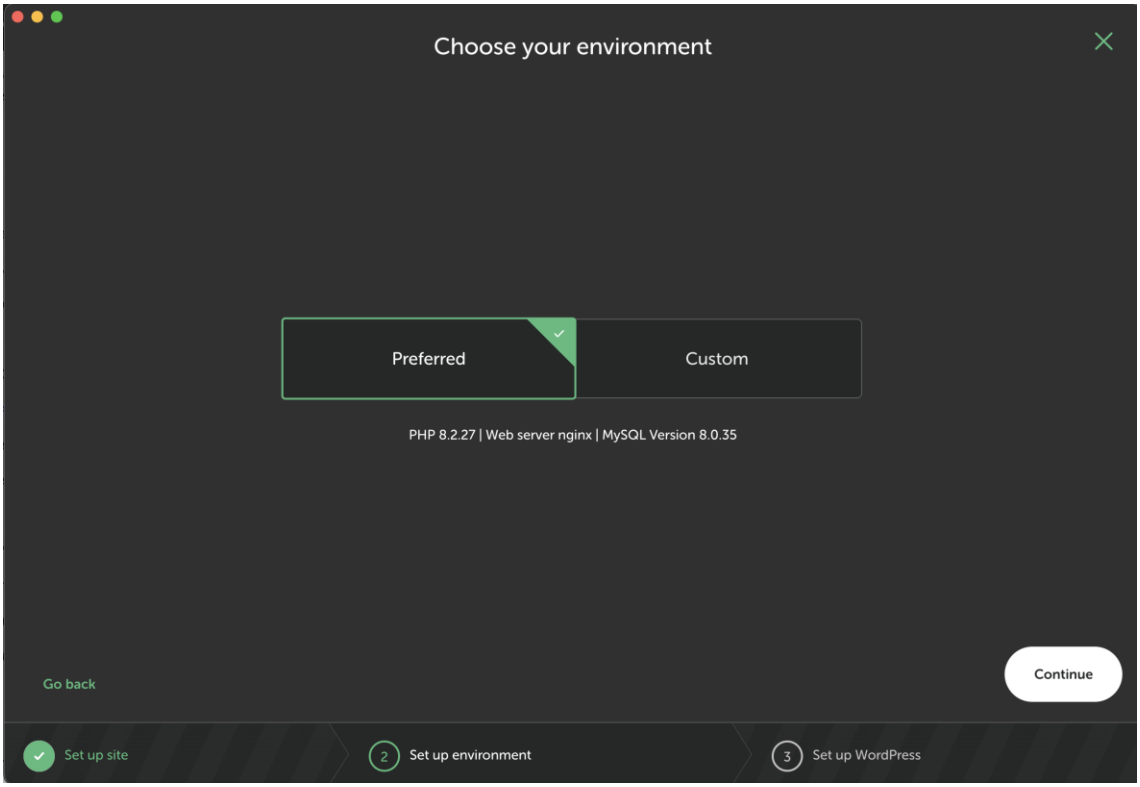


Рисунок 3.2 – Процес можливості вибору серверного оточення

Оскільки, при розробці додатку, ми будемо використовувати останні стабільні актуальні версії мови програмування PHP та СУБД MySQL, нам потрібно обрати ручний режим налаштування та вибрати потрібні версії. В нашому випадку мажорна версія PHP – 8.4 та MySQL – 8.0.35.

Останнім етапом встановлення є створення привілейованого користувача WordPress. Для цього потрібно створити username та password. Це зображено на рисунку 3.3.

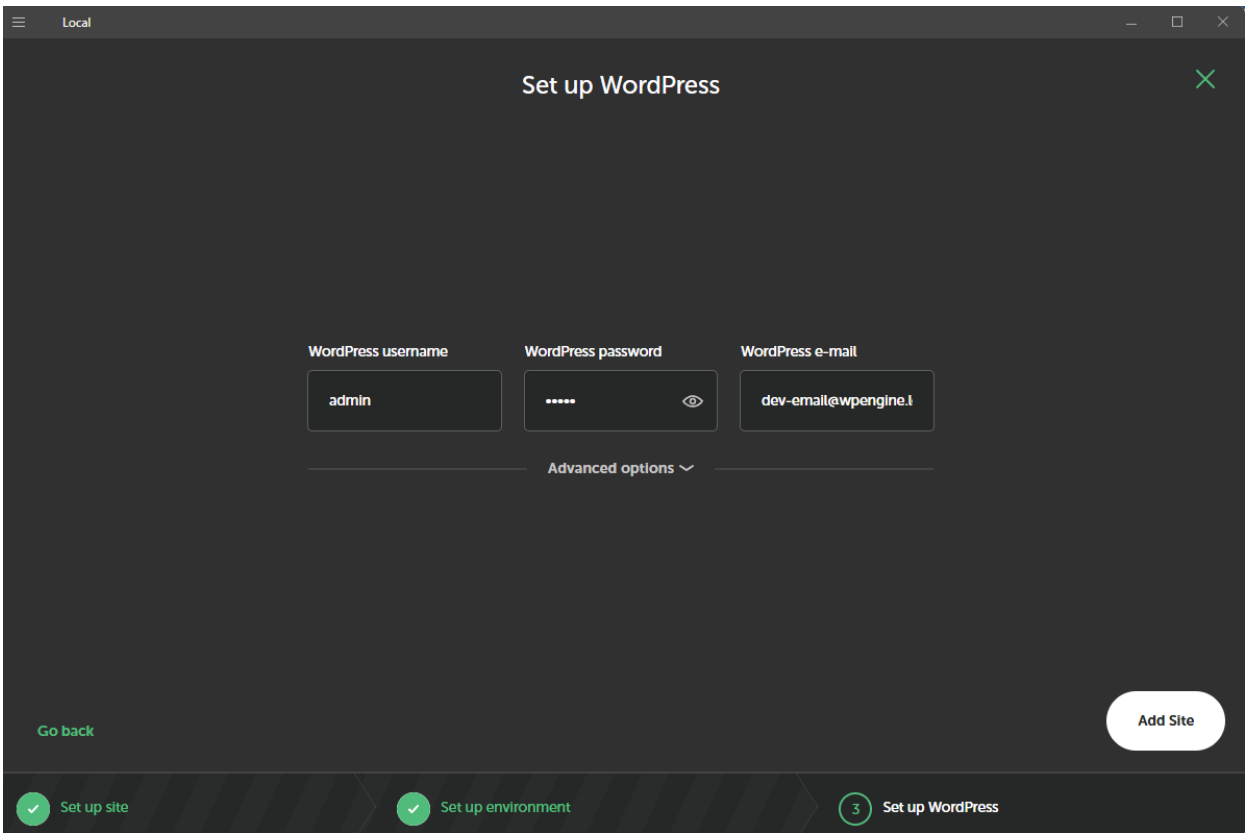


Рисунок 3.3 – Процес створення привілейованого користувача WordPress

Щоб переконатися, що конфігурація виконана правильно, потрібно відкрити в браузері локальну адресу, яка була задана на попередніх етапах. У результаті ми повинні побачити стартову сторінку з активованою останньою актуальною темою, розробленою спільнотою WordPress. Це зображено на рисунку 3.4.

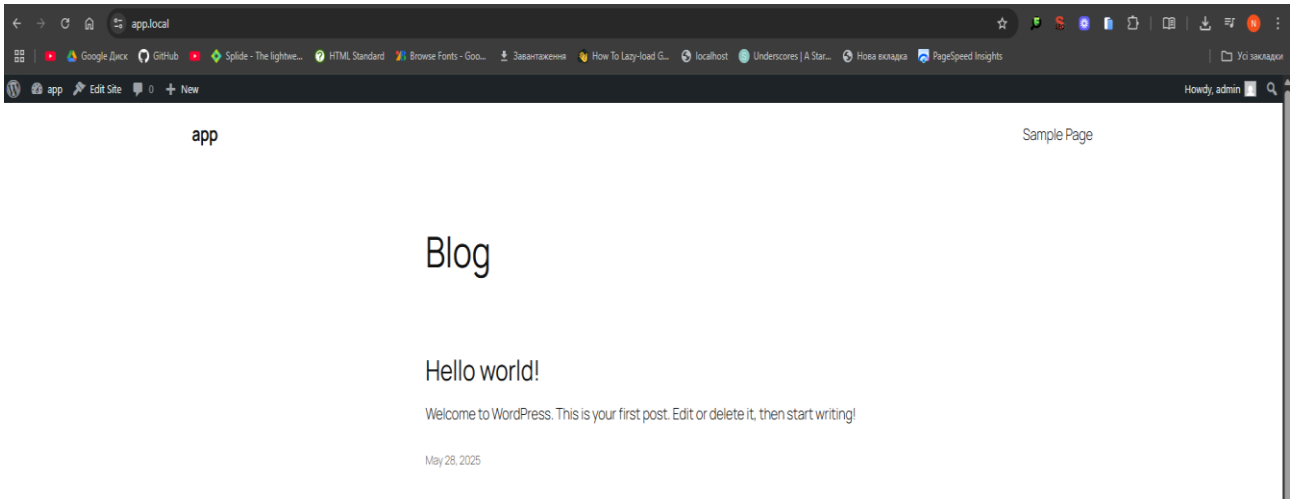


Рисунок 3.4 – Зображення успішно встановленої CMS WordPress

Будь-який WEB-додаток у WordPress базується на шаблонах. Шаблони (Templates) – це файли, які визначають, як саме буде відображатися ваш сайт WordPress у WEB-браузері [11]. Ці файли отримують інформацію з бази даних MySQL, після чого генерують HTML-код, який і передається на сторону клієнта (користувача).

Завдяки потужній системі тем (Theme system), WordPress надає можливість створювати як мінімальну кількість шаблонів, так і розгалужену структуру файлів, залежно від потреб розробника. Усі шаблони згруповані в межах однієї теми, проте кожен файл шаблону може бути призначений для певного сценарію використання: відображення окремої статті, архіву, сторінки пошуку, категорій тощо.

Створення шаблонів є важливою частиною процесу розробки теми, адже саме вони забезпечують взаємозв’язок між даними в базі та візуальним представленням контенту для користувача.

Кодекс WordPress строго описує як повинен бути побудований кожен шаблон, адже рушій CMS має власну велику систему ієрархії кожного шаблону, що зображено на рисунку 3.5 [12].

Рисунок 3.5 – Ієрархія шаблонів CMS WordPress

Безпосередньо для роботи власної теми, побудованої на шаблонах, потрібно лише три файли:

– style.css – основний файл стилів, у якому також міститься службова інформація про тему;

– index.php – головний шаблон, який WordPress використовує за замовчуванням, якщо не знайдено інші відповідні шаблони. Файл називається так історично, адже будь-який WEB-сервер першочергово налаштований на пошук файлу з назвою index;

– functions.php – файл для підключення скриптів, стилів та оголошення основних можливостей теми (реєстрація меню, підтримка зображень тощо).

Цих трьох файлів цілком достатньо для базового функціонування власноїтеми в WordPress.

Вони забезпечують мінімально необхідну структуру, щоб система розпізнала тему та могла її активувати. У такому випадку WordPress автоматично використовує файл index.php для виведення будь-якого вмісту, а стилі з файлу style.css застосовуються до всієї сторінки.

Проте, у реальних умовах розробки WEB-сайтів кількість шаблонних файлів зазвичай значно розширюється. Це дозволяє краще контролювати вигляд і логіку виводу різних типів контенту, таких як окремі записи, сторінки, архіви, результати пошуку, категорії, сторінки помилок тощо. Крім того, додаткові шаблони значно покращують гнучкість і масштабованість теми, дозволяючи створити унікальний дизайн для кожного окремого сценарію використання.

Спільнота WordPress розробила безліч інструментів, які значно спрощують процес створення власних тем. Одним із найпопулярніших і найзручніших таких інструментів є стартова тема Underscores (_s), розроблена командою Automattic – тією ж компанією, яка стоїть за самим WordPress.

Underscores є базовим «скелетом» теми, який включає в себе всі основні шаблонні файли, структуру директорій, базовий CSS, а також необхідні функції у файлі functions.php. Головна перевага використання _s полягає в тому, що

розробнику не потрібно створювати структуру теми з нуля – усе вже готово до кастомізації.

Цей інструмент ідеально підходить для тих, хто хоче створити повністю унікальну тему, не покладаючись на готові рішення або сторонні візуальні редактори. За допомогою Underscores можна зосередитися саме на дизайні, логіці виводу контенту та функціональності, а не витрачати час на базові налаштування.

Щоб скористатися стартовою темою, достатньо перейти на офіційний сайт https://underscores.me, вказати назву майбутньої теми та завантажити архів з уже згенерованою основою. Далі її можна адаптувати під потреби конкретного проєкту: змінити шаблони, додати стилі, підключити шрифти, розширити функціональність тощо.

Таким чином, Underscores є надійною відправною точкою для розробників, які прагнуть створити власну, гнучку і водночас легку тему для WordPress з дотриманням усіх стандартів кодування [13].

На рисунку 3.6 наведено інтерфейс інструменту Underscores. Потрібно увести назву теми та її згенерувати.

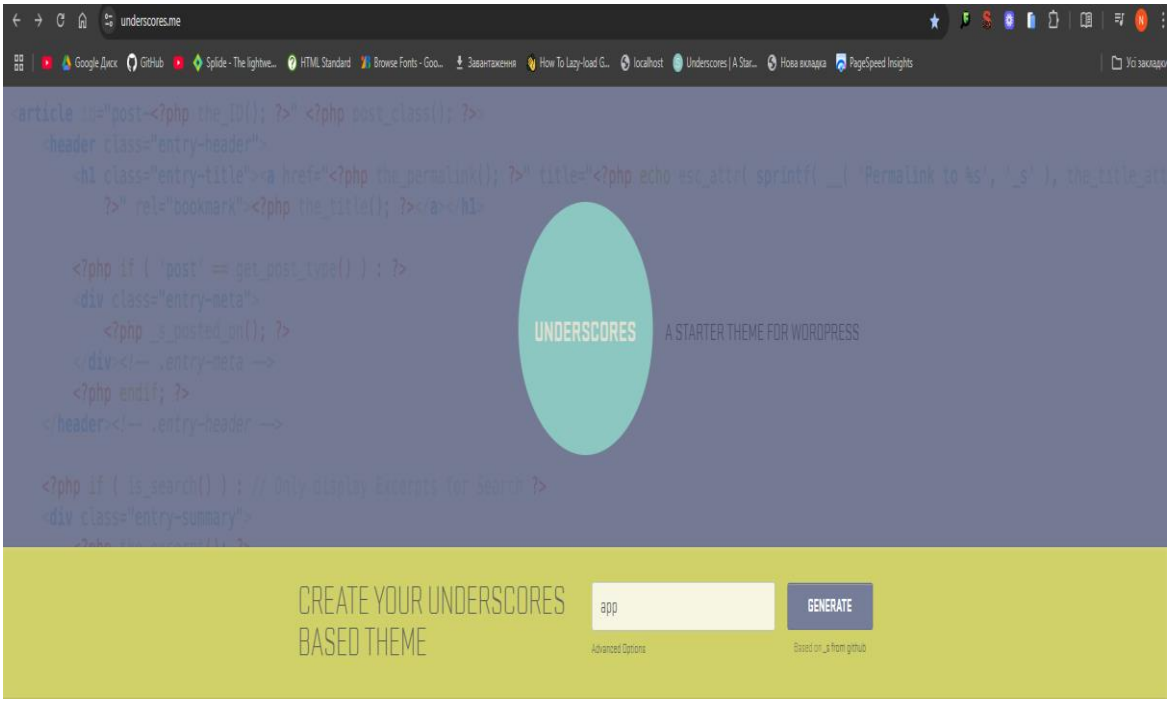


Рисунок 3.6 – Зображення інтерфейсу інструменту Underscores

Особливою перевагою Underscores є те, що цей інструмент генерує тему відповідно до офіційних стандартів кодексу WordPress (WordPress Coding Standards) [14].
Це означає, що структура, іменування, форматування коду, використані функції та шаблони повністю відповідають актуальним вимогам системи. Завдяки цьому ми уникаємо багатьох потенційних проблем, пов’язаних із використанням застарілих методів, несумісного коду чи неефективних практик.

Крім того, це гарантує максимальну сумісність з ядром WordPress та популярними плагінами, а також спрощує подальшу підтримку й масштабування проєкту. Такий підхід особливо важливий для тих розробників, які прагнуть не лише створити функціональний продукт, а й дотримуватися професійної етики кодування, що є важливою частиною роботи в команді або під час розробки для замовника.

Таким чином, Underscores є не просто шаблоном, а потужною відправною точкою, що дозволяє будувати якісні, сучасні та безпечні теми без ризику виникнення проблем, пов’язаних з порушенням стандартів.

Отже, ми отримаємо згенеровану тему, наступним кроком її потрібно встановити до CMS. Будь-яка тема генерується у архів формату zip. Щоб встановити шаблон, потрібно зайти у панель адміністрування WordPress за посиланням app.local/wp-admin та ввести згенеровані раніше дані привілейованого користувача. Після успішної авторизації – ми побачимо майстерню CMS, що зображено на рисунку 3.7. Вбудований рушій CMS WordPress автоматично обробляє завантажений архів теми: виконує його розпакування, розміщення у відповідному каталозі (wp-content/themes/) та зчитування основних службових файлів, необхідних для активації теми. Одним із ключових елементів структури теми є файл style.css, який відіграє не лише роль таблиці стилів, а й містить службову метаінформацію, що використовується WordPress для ідентифікації теми в адміністративній панелі. У верхній частині цього файлу має бути розміщений спеціальний блок коментарів, де зазначаються назва теми, її версія, автор, опис, ліцензія тощо. Наявність цього

блоку є обов'язковою умовою для того, щоб система розпізнала архів як повноцінну тему. Якщо файл style.css відсутній або не містить необхідних метаданих, WordPress не зможе активувати тему, а в інтерфейсі адміністратора з'явиться відповідне повідомлення про помилку.

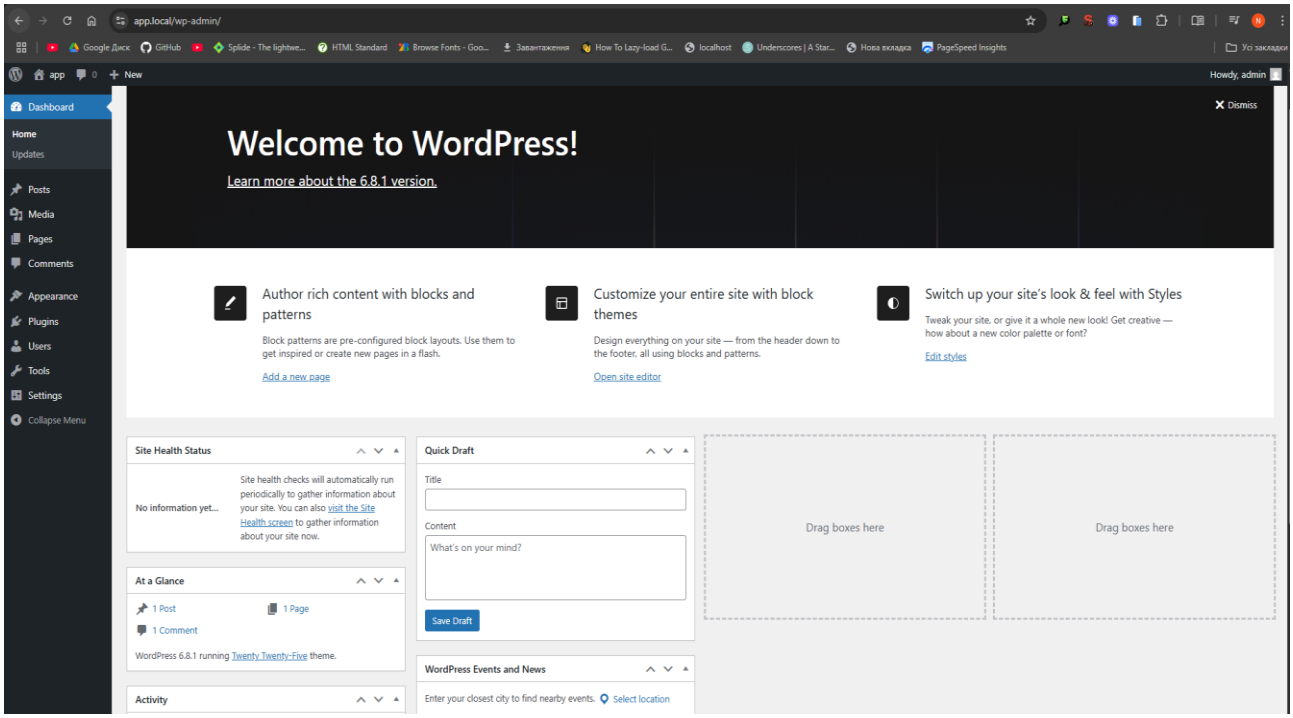


Рисунок 3.7 – Зображення майстерні WordPress

На наступному етапі необхідно перейти у вкладку "Appearance" (Зовнішній вигляд) в адміністративній панелі WordPress, або скористатися прямим посиланням app.local/wp-admin/themes.php, що веде до відповідного розділу керування темами. На цій сторінці слід завантажити попередньо підготовлений архів із темою, яка була згенерована на етапі розробки. Після завантаження архіву система автоматично розпізнає його вміст і надасть можливість активувати тему для подальшого використання у WEB-додатку. Даний крок є необхідним для інтеграції кастомізованого інтерфейсу з backend-частиною сайту та забезпечення коректного відображення контенту відповідно до розробленої структури та дизайну. Після успішної активації тема з'явиться у списку, що проілюстровано на рисунку 3.8.

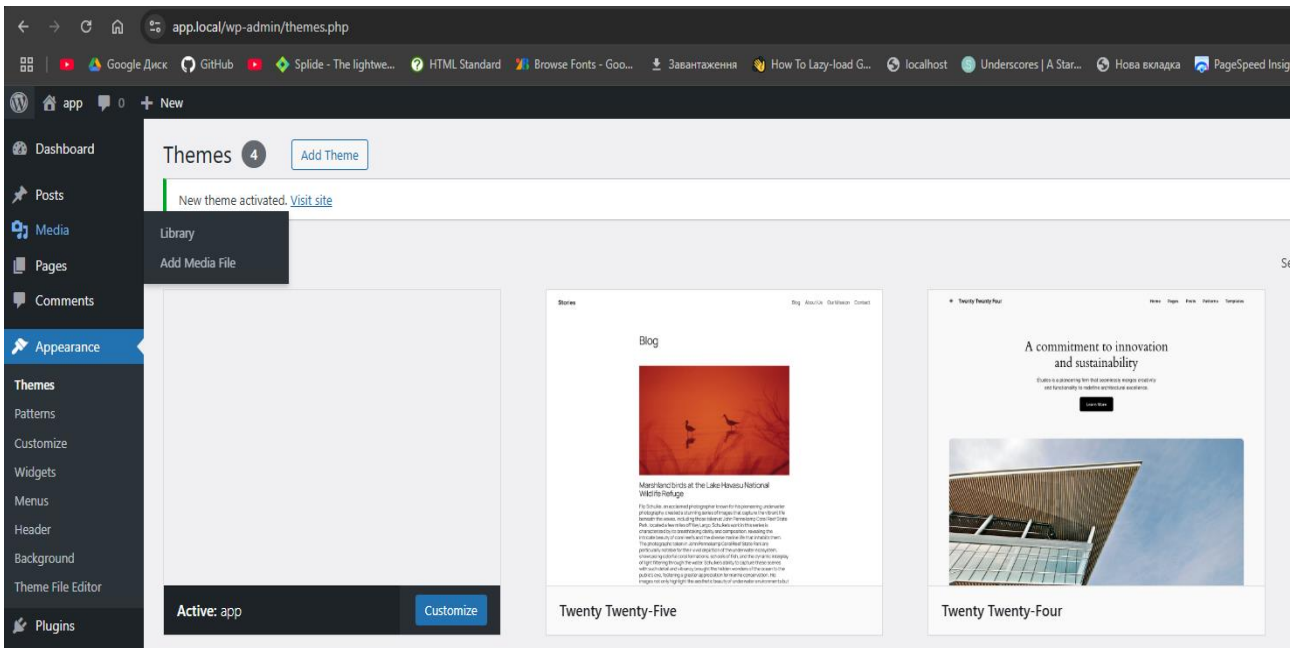


Рисунок 3.8 – Зображення успішно встановленої теми

Після успішного встановлення теми всі підготовчі етапи можна вважати завершеними, що створює необхідні передумови для подальшої розробки WEB-додатку. На цьому етапі доцільно перейти до детального ознайомлення зі структурою каталогу теми, її основними файлами та функціональними компонентами. Це дозволить забезпечити цілісне розуміння принципів побудови теми та організувати подальшу роботу відповідно до вимог системи керування вмістом WordPress.

Структура CMS WordPress містить велику кількість системних файлів та внутрішніх каталогів, кожен з яких виконує чітко визначену роль у функціонуванні платформи. Як зображено на рисунку 3.9, до складу стандартної інсталяції входять каталоги wp-admin, wp-includes, wp-content та ряд конфігураційних файлів, що забезпечують роботу ядра системи, інтерфейсу адміністратора, взаємодію з базою даних, а також підтримку плагінів і тем.

Незважаючи на складність і масштабність загальної структури, у процесі створення користувацького інтерфейсу та реалізації функціоналу майбутнього WEB-додатку безпосередній інтерес становить лише один каталог - wp-content/themes/app. Саме цей каталог є кореневим середовищем для розробки

кастомної теми, що поєднуватиме шаблони, стилі, скрипти та функціональні можливості, адаптовані під вимоги конкретного проекту.

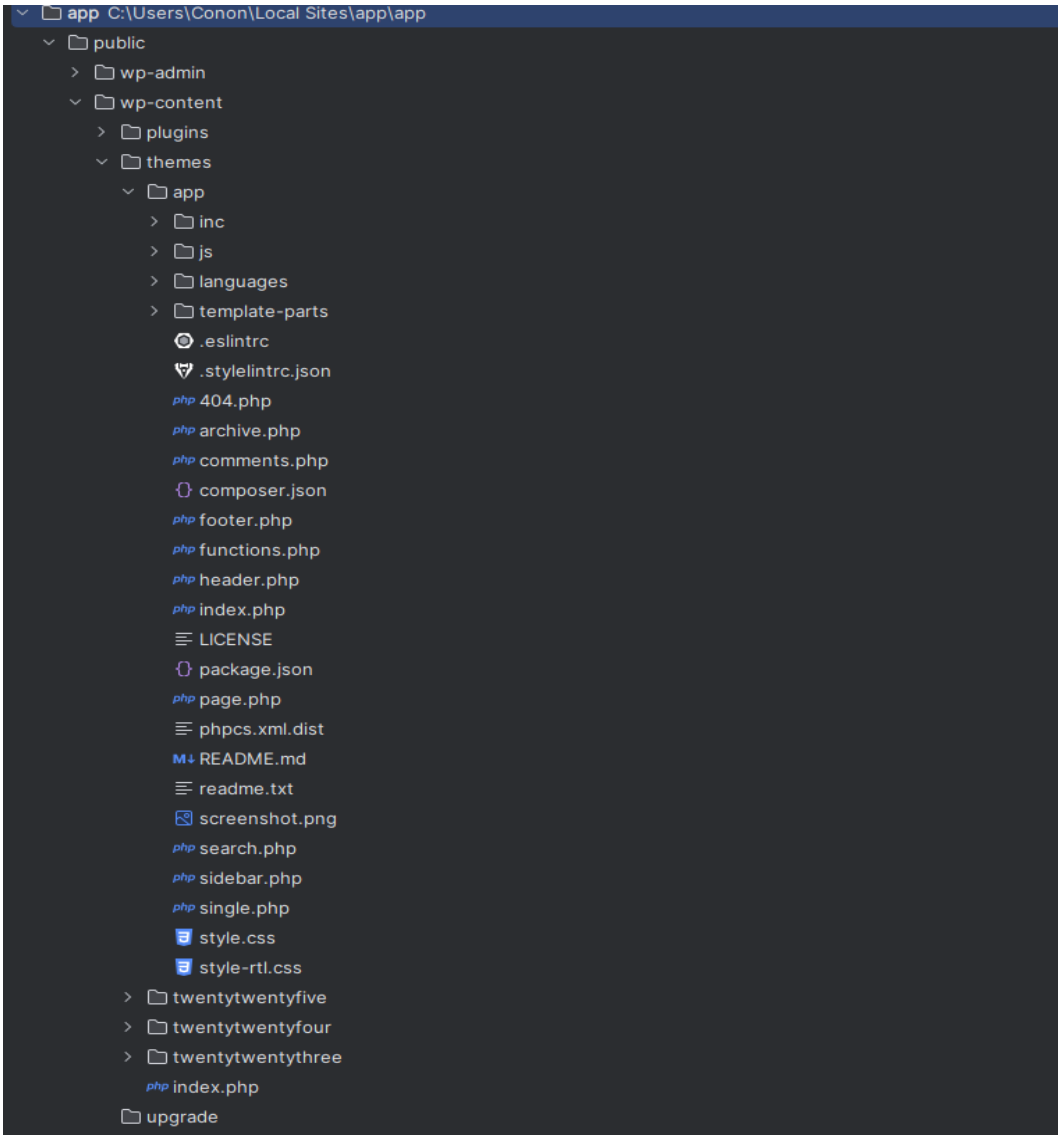


Рисунок 3.9 – Загальна структура файлів CMS WordPress

Після базового налаштування середовища розробки та структури теми наступним кроком є перевірка працездатності REST API [15], оскільки у даному проєкті реалізується Headless-підхід до побудови WEB-додатку. Цей підхід передбачає розділення логіки клієнтської та серверної частин: WordPress у цьому випадку виконує роль backend-системи керування вмістом, а вся взаємодія з даними відбувається через API-запити.

Для підтвердження того, що взаємодія з API відбувається коректно, необхідно здійснити тестовий запит до одного з базових ендпоінтів WordPress REST API – а саме: https://app.local/wp-json/wp/v2/posts

У відповідь має бути повернений JSON-об’єкт із переліком наявних публікацій, що свідчить про правильне налаштування API та успішну інтеграцію Headless-архітектури.

JSON (або JavaScript Object Notation) – це формат даних, який використовується для обміну даними між різними додатками. Він має текстовий формат, що дозволяє зручно обмінюватися даними між різними мовами програмування [16].

JSON формат складається з колекції пар ключ-значення, які зазвичай використовуються для представлення об’єктів. В об’єкті може бути присутніми інші об’єкти, масиви, числа, рядки та булеві значення.

API (або Application Programming Interface) – це інтерфейс, який дозволяє різним додаткам спілкуватися один з одним. У WEB-розробці API зазвичай використовуються для передачі даних між клієнтом та сервером.

JSON дуже поширений в API, оскільки він дуже зрозумілий і оброблюється багатьма мовами програмування. В API, як правило, клієнт запитує сервер за певними даними і отримує їх у відповідь у вигляді JSON-об’єкта. Після цього клієнт може обробити ці дані та використовувати їх в своєму додатку.

Цей крок є ключовим для подальшої реалізації клієнтської частини, яка буде динамічно отримувати та відображати контент, використовуючи JavaScript-фреймворк або іншу технологію frontend. На рисунку 3.10 зображено відповідь сервера після звертання до нього використовуючи REST API.

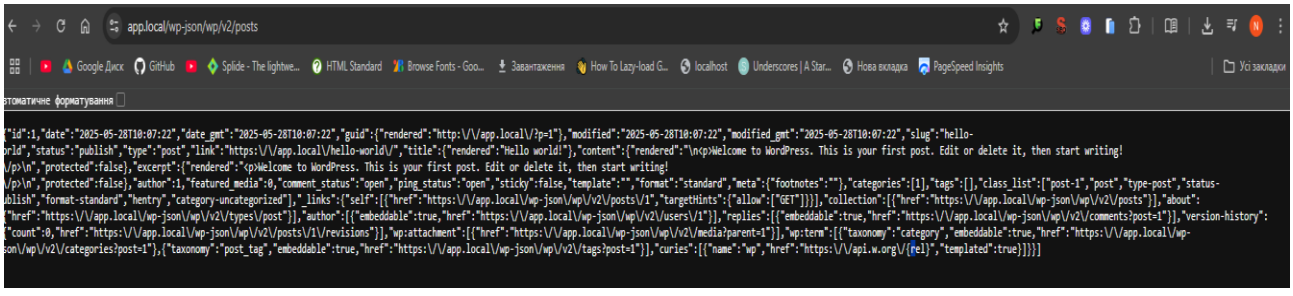


Рисунок 3.10 – Відповідь повернута сервером у форматі JSON

Таким чином, етап налаштування серверної частини WEB-додатку, який охоплював встановлення CMS WordPress, конфігурацію теми, структурування файлової системи та перевірку роботи REST API, успішно завершено. На цьому етапі створено всі необхідні умови для подальшої розробки клієнтської частини застосунку, яка відповідатиме за динамічне відображення вмісту, взаємодію з користувачем та побудову інтерфейсу відповідно до сучасних вимог до WEB-дизайну та зручності користування. Перехід до розробки frontend-компонентів дозволить реалізувати головну мету Headless-підходу – чітке розділення логіки представлення даних та системи управління контентом.

3.2 Розробка frontend частини

Оскільки, для реалізації frontend частини була вибрана реактивна JavaScript бібліотека React, то розробку слід почати із пояснення, що таке термін “реактивності”.

У контексті програмування реактивність – це підхід, де система автоматично реагує на зміни даних. Замість того, щоб вручну оновлювати DOM або викликати функції щоразу, коли щось змінюється, ми створюємо залежності між даними та «ефектами» (наприклад, оновленням UI або виведенням у консоль). Коли дані змінюються, залежні від них ефекти спрацьовують автоматично.

Щоб реалізувати базову реактивність, нам знадобляться:

– State (стан). Це дані, за якими ми спостерігаємо. Наприклад, рахунок гри або ім’я користувача на сайті. Це значення, які можуть змінюватися;

– трекінг. Потрібно мати механізм для відстеження. Наприклад, якщо функція оновлює текст на екрані, вона має знати, яке значення (наприклад, ім’я користувача) потрібно показати. Якщо ці дані змінюються, ми повинні оновити відповідну частину інтерфейсу;

– реакція на зміни. Коли дані змінюються (наприклад, оновлюється лічильник), має спрацювати механізм оповіщення [17].

Розробку frontend-частини WEB-додатку, що базується на бібліотеці React, доцільно розпочати з локального розгортання середовища розробки, аналогічно до етапів налаштування backend-частини на базі WordPress. Такий підхід дозволяє організувати ефективний процес розробки в ізольованому середовищі, де можна тестувати функціонал без потреби взаємодії з публічним сервером.

У межах початкового етапу відбувається налаштування структури React-проєкту, підключення до REST API WordPress, перевірка отримання та відображення даних через маршрут https://app.local/wp-json/wp/v2/posts, а також створення базових компонентів для інтерфейсу користувача. Такий підхід забезпечує гнучкість у розробці, полегшує відлагодження логіки взаємодії з backend-частиною, а також дозволяє зосередитися на побудові сучасного, швидкого та зручного інтерфейсу відповідно до концепції Headless-архітектури.

Для розгортання Frontend додатку є чимало способів, але гарною практикою вважається користування білдерами, наприклад: Vite. Ці інструменти власноруч завантажують всі потрібні актуальні пакети та значно полегшить процес завантаження додатку на публічний сервер.

Vite – це інструмент збірки, метою якого є забезпечення швидшого та спрощеного процесу розробки сучасних WEB-проектів. Він складається з двох основних частин:

– сервер розробки, який надає багато функцій порівняно з нативними модулями ES, наприклад, надзвичайно швидку заміну модулів Hot Module Replacement (HMR).

– команда збірки, яка об'єднує ваш код із Rollup, попередньо налаштованим для виведення високооптимізованих статичних ресурсів для продакшену.

Vite має чіткі налаштування та постачається з розумними налаштуваннями за замовчуванням [18].

Для встановлення Vite потрібно виконати команду npm create vite@latest, після чого система ініціалізує процес створення нового проєкту та виведе відповідне оповіщення в терміналі. У рамках виконання цієї команди

користувачеві буде запропонован надати дозвіл на встановлення додаткових залежностей, що зображено на рисунку 3.11.

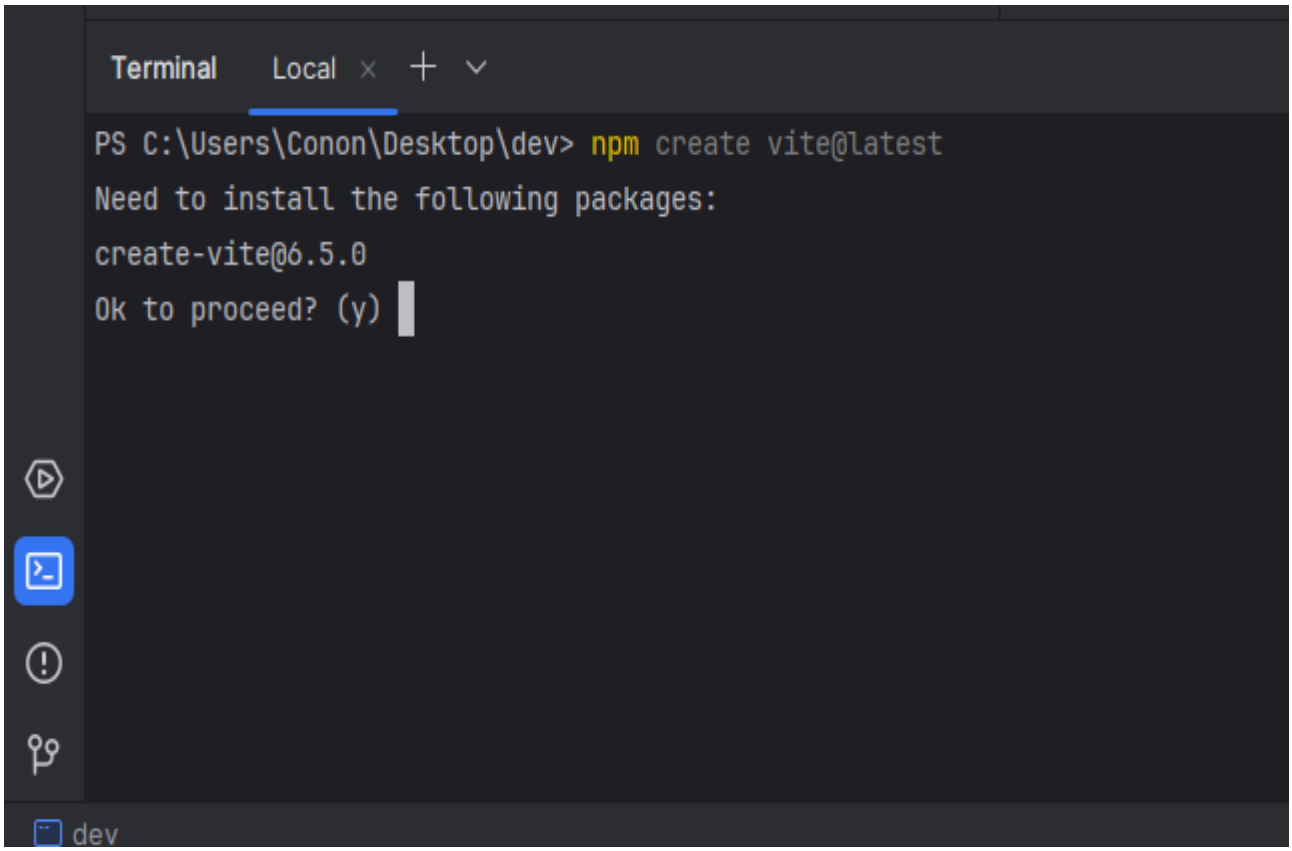


Рисунок 3.11 – Встановлення залежностей Vite

Після успішного встановлення та запуску проєкту за допомогою інструменту збирання Vite, у терміналі буде згенеровано локальну IP-адресу, яка дозволяє отримати доступ до WEB-додатку через браузер. Ця адреса зазвичай має вигляд http://localhost:5173 або подібний, залежно від конфігурації середовища. Саме за цією адресою можна перевірити працездатність проєкту, протестувати базовий функціонал та розпочати подальшу розробку інтерфейсу користувача. Подібне локальне розгортання є стандартною практикою при розробці сучасних WEB-застосунків і забезпечує ефективність та безпеку на ранніх етапах роботи. На рисунку 3.12 зображено запущений додаток.

Для того, щоб перейти до наступних етапів, що будуть описані у розділі 3.4 залишилось протестувати зв’язок WEB-додатку та backend частини створеній на CMS Wordpress. Для цього необхідно внести відповідні зміни до

компонента App.jsx. Саме цей компонент зазвичай вважається точкою входу до усього frontend-додатку, що відповідає поняттю “root-компонент”. Його основне призначення полягає у загальному структурному впорядкуванні внутрішніх компонентів, маршрутизації та базовій ініціалізації логіки інтерфейсу. Враховуючи це, безпосереднє опрацювання запитів до API або зчитування/обробка даних у межах App.jsx у більшості випадків вважається не найкращою практикою.

Компонент React – це фрагмент коду, який можна використовувати повторно, і відповідає за рендеринг частини користувацького інтерфейсу. Ці компоненти можуть бути як простими, як кнопка, так і складними, як ціла форма. Модульна природа компонентів React дозволяє розробникам розбивати свої програми на керовані та повторно використовувані фрагменти, сприяючи створенню чистої та організованої кодової бази [19].

Однак у контексті початкового етапу тестування функціональності взаємодії frontend-додатку з backend-системою (WordPress REST API) доцільним є використання спрощеного підходу, що передбачає реалізацію базового запиту безпосередньо у відповідному компоненті клієнтської частини. Такий підхід дозволяє на початковому етапі швидко перевірити працездатність налаштованого API, а також переконатися у правильності та повноті відповідей, що надходять із сервера. Це є важливим кроком для виявлення можливих технічних проблем, пов'язаних із налаштуванням маршрутизації, конфігурацією прав доступу або форматом даних. Проведення такого тестування забезпечує впевненість у стабільності базової інтеграції між frontend- та backend-частинами системи ще до розгортання повноцінної логіки обробки та відображення отриманого контенту.

Запит буде надсилатися на відповідний маршрут REST API, описаний у попередніх розділах роботи, а саме: https://app.local/wp-json/wp/v2/posts. Цей ендпоінт забезпечує доступ до публікацій (записів) у системі керування контентом WordPress у форматі JSON, що дозволяє frontend-додатку отримувати актуальні дані для відображення.

На рисунку 3.12 представлено оновлений компонент App.jsx, у якому було внесено відповідні зміни для реалізації базового механізму взаємодії з даним API. Завдяки цій інтеграції компонент отримує дані про записи безпосередньо з backend-системи та демонструє базову функціональність відтворення контенту на стороні клієнта. Такий підхід є першим кроком у побудові повноцінного зв'язку між frontend- і backend-частинами розроблюваного WEB-додатку.

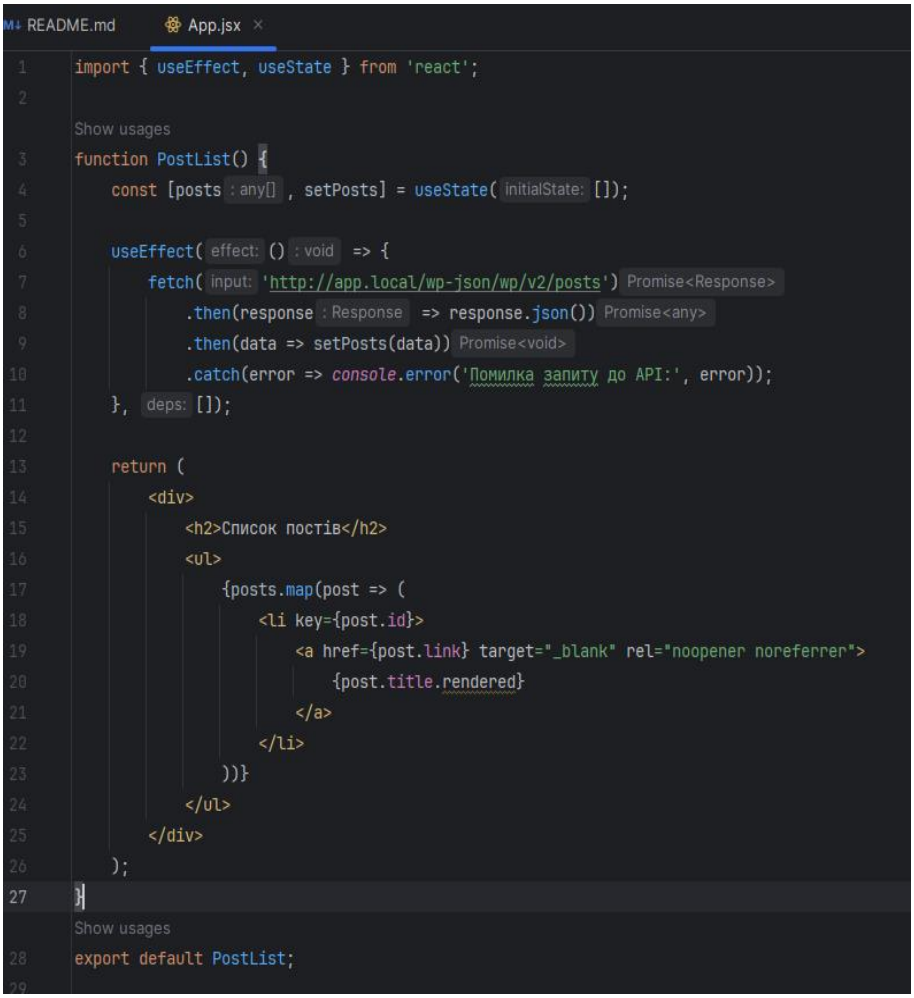


Рисунок 3.12 – Змінений компонент App.jsx

Як зображено на рисунку 3.12, компонент App звертається за допомогою методу fetch на маршрут https://app.local/wp-json/wp/v2/posts. Отримавши відповідь у форматі JSON, дані декодуються та потім записуються у масив функцію setPosts.

Компонент безпосередньо повертає частину HTML-розмітки, яка динамічно формується на основі отриманих з API даних. Зокрема, створюється елемент списку <ul>, до якого за допомогою методу .map() додаються окремі елементи <li>. Кожен з них містить посилання з назвою відповідного допису. Назва поста підтягується із поля post.title.rendered, що є частиною JSON-об’єкта, отриманого з REST API WordPress. Завдяки використанню методу map() досягається гнучка генерація контенту на основі динамічних даних.

Після написання і збереження даного коду, його було перевірено через браузер. У результаті виконання можна побачити коректно сформований список публікацій із клікабельними заголовками – відповідний приклад виводу представлено на рисунку 3.13.

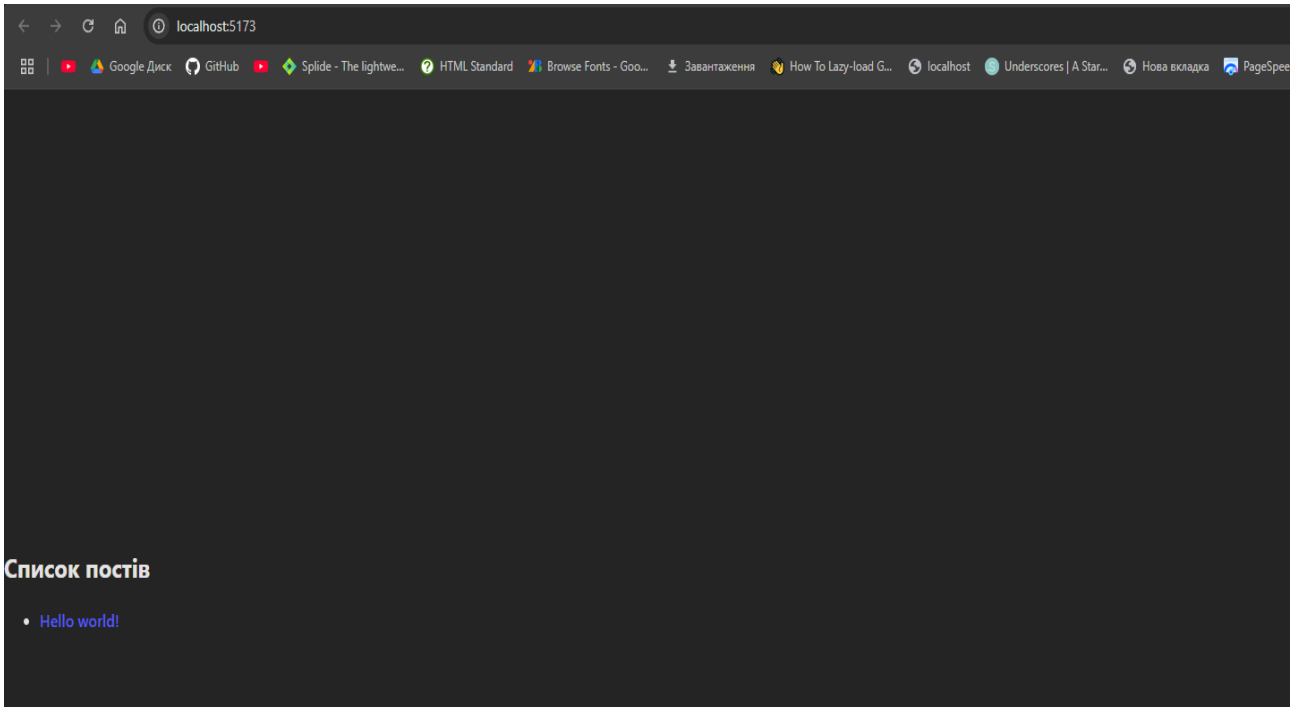


Рисунок 3.13 – Результат виконання коду

На рисунку 3.13 зображено, що WEB-додаток отримав дані про єдиний запис із назвою “Hello world!”, що є базовим записом у CMS WordPress. До того ж, посилання на цей пост сформовано правильно: https://app.local/hello-world/. Це свідчить, що frontend та backend частина додатку працює правильно і можна переходити до наступного розділу.

3.3 Реалізація Headless підходу

Для того, щоб показати гнучкість та масштабованість Headless підходу достатньо відтворити функціонал будь-якого сучасного WEB-додатку: сторінка 404, сторінка посту, сторінка блогу, головна сторінка.

Згаданий вище функціонал буде реалізовуватись за допомогою функціональних компонентів, що дозволить реалізувати структуру, аналогічну повноцінному корпоративному додатку, забезпечуючи при цьому модульність, принцип DRY та швидке масштабування.

Принцип «Не повторюйся» (DRY) – це основоположна концепція в розробці програмного забезпечення, яка допомагає створювати чистіший та зручніший у підтримці код. Запропонований Енді Хантом та Дейвом Томасом у книзі «Прагматичний програміст», DRY виступає за зменшення дублювання коду шляхом забезпечення єдиного, однозначного та авторитетного представлення кожного фрагмента знань або логіки в системі [20].

У структурі вебзастосунку ключову роль відіграє компонент App.jsx, який відповідає за організацію маршрутизації та початкову ініціалізацію інтерфейсу. У цьому компоненті визначається загальна архітектура навігації, підключаються глобальні стилі та спільні елементи інтерфейсу, що забезпечує узгоджене візуальне оформлення всіх сторінок.

Компонент Header.jsx реалізує шапку сайту, до складу якої входить логотип і навігаційне меню. Для його побудови використовується компонент Navbar із бібліотеки Bootstrap 5, що дозволяє забезпечити адаптивність і коректне відображення на пристроях із різними розширеннями екрана. Компонент Footer.jsx виконує функцію підвалу сайту. Він містить контактну інформацію, посилання на соціальні мережі, а також текстовий блок із копірайтом. Цей елемент є завершальним візуальним акцентом на кожній сторінці та сприяє зручності користувача при пошуку додаткових відомостей.

Головна сторінка застосунку реалізована у вигляді компонента Home.jsx. Вона містить загальну інформацію про проєкт або компанію, а також

інтерактивні елементи, зокрема слайдер або інформаційний банер, що привертає увагу користувача до ключових повідомлень.

Компонент Company.jsx орієнтований на взаємодію з контентом, розміщеним у CMS WordPress. Він надсилає запит до відповідної сторінки за унікальним ідентифікатором (наприклад, slug: "about-us") та виводить її вміст у frontendi. Такий підхід забезпечує гнучкість в управлінні корпоративною інформацією без необхідності вносити зміни до коду інтерфейсу. Для відображення списку блогових публікацій застосовується компонент Blog.jsx, який звертається до API WordPress (зокрема, до маршруту wp-json/wp/v2/posts).

Будь-який WEB-додаток або WEB-сайт містить так званий “header” – верхню частину інтерфейсу, яка виконує функцію шапки сторінки. У header зазвичай розміщується навігаційне меню, логотип компанії чи продукту, а також інші важливі елементи, що забезпечують користувачу швидкий доступ до основних розділів сайту. Цей компонент є незамінним елементом структури WEB-додатку, оскільки сприяє зручності користування та покращує орієнтацію відвідувачів. На рисунку 3.14 зображено вміст компонента Header.jsx

```jsx
// components/Header.jsx
import { Link } from 'react-router-dom';

Show usages
function Header() {
    return (
        <header className="bg-white shadow-md">
            <nav className="container mx-auto px-4 py-4 flex justify-between items-center">
                <Link to="/" className="text-xl font-bold">MySite</Link>
                <ul className="flex gap-4">
                    <li><Link to="/" className="hover:text-blue-600">Home</Link></li>
                    <li><Link to="/company" className="hover:text-blue-600">Company</Link></li>
                    <li><Link to="/blog" className="hover:text-blue-600">Blog</Link></li>


                </ul>
            </nav>
        </header>
    );
}
Show usages
export default Header;
```

Рисунок 3.14 – Вміст компонента Header.jsx

Окрім звичної для будь-якого WEB-додатку розмітки, компонент також містить елементи React Router. React Router – це бібліотека для обробки маршрутів і навігації в програмах React JS. Це дозволяє здійснювати навігацію в односторінковій програмі (SPA) без оновлення всієї сторінки [21].

Наступним компонентом буде Footer.jsx – “footer” WEB-додатку. Цих два елементи з’являтимуться безпосередньо на кожній сторінці WEB-додатку і саме підхід використання реактивної бібліотеки дозволяє просто перевикористовувати потрібний компонент та викликати його на потрібній сторінці чи навіть всередині потрібного компоненту, як наведено на рисунку 3.15.

```
function App() {
    return (
        <Router>
            <div className="flex flex-col min-h-screen">
                <Header />
                <main className="flex-grow">
                    <Routes>
                        <Route path="/" element={<Home />} />
                        <Route path="/company" element={<Company />} />
                        <Route path="/blog" element={<Blog />} />
                        <Route path="/post/:slug" element={<Post />} />
                        <Route path="*" element={<NotFound />} />
                    </Routes>
                </main>
                <Footer />
            </div>
        </Router>
    );
}
```

Рисунок 3.15 – Зображення перевикористання компонентів Header та Footer всередині кореневого компонента App

Логіка компонентів Home, Blog та Post є типовою для сучасних frontend-додатків, які базуються на JavaScript-бібліотеках, таких як React. Основна ідея цих компонентів полягає у використанні підходу, який дозволяє отримувати дані

з зовнішніх джерел (API), обробляти їх та динамічно відображати у вигляді інтерфейсу користувача.

По-друге, отримані дані піддаються попередній обробці. Наприклад, для постів блогу це може бути форматування дат, вилучення потрібних зображень або фрагментів тексту, а також очищення HTML-коду для безпечного відображення.

Завдяки такому підходу, компоненти стають ізольованими, повторно використовуваними та масштабованими. Вони виконують чітко визначені функції: один відповідає за відображення списку постів (Blog), інший – за показ окремого посту (Post), а третій – за домашню сторінку (Home) з відповідним контентом. Це сприяє зрозумілості коду, легкості його підтримки та подальшого розвитку.

Таким чином, логіка компонентів Home, Blog і Post ілюструє суть сучасних WEB-технологій – побудову адаптивних, динамічних і зручних для користувачів інтерфейсів на основі якісного, структурованого та актуального контенту. На рисунку 3.16 зображено вміст компонента Post.jsx

```jsx
import { useEffect, useState } from 'react';
import { useParams } from 'react-router-dom';

Show usages
function Post() {
    const { slug : string } = useParams();
    const [post, setPost] = useState( initialState: null);

    useEffect( effect: () : void => {
        fetch( input: `https://app.local/wp-json/wp/v2/posts?slug=${slug}` ) Promise<Response>
            .then(res : Response => res.json()) Promise<any>
            .then(data => {
                if (data.length) setPost(data[0]);
            });
    }, deps: [slug]);

    if (!post) return <div className="text-center py-10">Loading...</div>;

    return (
        <article className="container mx-auto px-4 py-10">
            <h1 className="text-3xl font-bold mb-4">{post.title.rendered}</h1>
            <p className="text-gray-600 text-sm mb-4">{new Date(post.date).toLocaleDateString()}</p>
            <div dangerouslySetInnerHTML={{ __html: post.content.rendered }} />
        </article>
    );
}

Show usages
export default Post;
```

Рисунок 3.16 – Зображення вмісту компонента Post.jsx

Хук useEffect в React використовується для управління побічними ефектами у функціональних компонентах. У нашому випадку він відповідає за відправку запиту на backend, яким виступає CMS WordPress. Після рендерингу компонента useEffect виконується один раз (або за залежностями) та ініціює асинхронний HTTP-запит до API WordPress для отримання даних.

Отримана відповідь приходить у форматі JSON, що є стандартним форматом обміну даними між клієнтом і сервером. Ці дані можуть містити різноманітну інформацію, наприклад, список постів, деталі окремого запису, метадані, зображення та інше.

Після отримання та обробки JSON-відповіді, компонент оновлює свій стан за допомогою React-хука useState, зберігаючи отримані дані у внутрішній змінній стану. Далі React автоматично виконує повторний рендер компоненту, відображаючи оновлений контент.

У нашому прикладі ці дані виводяться всередині тега article, що відповідає за семантичну розмітку основного контенту сторінки. Це дозволяє не лише красиво та структуровано показати інформацію користувачу, але й підвищує SEO-оптимізацію сайту, оскільки пошукові системи краще розуміють зміст сторінки.

Таким чином, хук useEffect реалізує ключову роль у динамічному отриманні, обробці та відображенні контенту з WordPress API в сучасному React-додатку.

3.4 Висновок до розділу 3

У третьому розділі було здійснено повний цикл розробки WEB-додатку із застосуванням архітектури Headless CMS. На етапі серверної частини було створено та налаштовано інформаційну систему на базі WordPress, реалізовано структуру контенту та організовано доступ до даних через REST API. Особливу увагу було приділено забезпеченню гнучкості та масштабованості серверної

складової, що дозволяє ефективно використовувати наявні інформаційні ресурси.

У межах реалізації клієнтської частини було розроблено інтерфейс користувача з використанням фреймворку React. Було впроваджено механізми отримання та обробки динамічних даних із backend-системи, реалізовано маршрутизацію сторінок та інтерактивні елементи, що підвищують зручність використання додатку.

У результаті виконаних робіт розроблено функціональний WEB-додаток, що відповідає сучасним вимогам до веб-розробки. Застосована Headless-архітектура забезпечила чіткий розподіл відповідальностей між клієнтською та серверною частинами, що сприяє підвищенню продуктивності, гнучкості та спрощенню подальшої підтримки та розширення програмного продукту.

4 ТЕСТУВАННЯ WEB-ДОДАТКУ ДЛЯ CMS З ВИКОРИСТАННЯМ HEADLESS ПІДХОДУ

4.1 Тестування головного модуля реалізованого додатку

У процесі виконання бакалаврської кваліфікаційної роботи було розроблено WEB-додаток, який функціонує у зв’язці з системою керування контентом (CMS) WordPress, використовуючи сучасну архітектуру типу Headless. Такий підхід передбачає повне розділення frontend-частини (користувацького інтерфейсу) та backend-системи керування контентом, що дає змогу забезпечити більшу гнучкість, масштабованість та зручність у подальшій підтримці застосунку.

Розробка програмного забезпечення здійснювалась з урахуванням загальноприйнятих стандартів WEB-програмування, що гарантує відповідність проєкту сучасним вимогам до якості, безпеки та масштабованості. У процесі реалізації було використано актуальні технології, зокрема фреймворк React, який виконує роль клієнтської частини додатку, та REST API WordPress, що слугує джерелом контенту та забезпечує динамічну взаємодію з серверною частиною. Використання React дало змогу реалізувати гнучку компонентну структуру інтерфейсу користувача, що значно спрощує підтримку та розвиток проєкту в майбутньому. Завдяки віртуальному DOM і ефективним механізмам оновлення стану компонентів досягнуто високої швидкодії інтерфейсу, зменшено час відгуку системи та покращено загальне враження від користування додатком. Інтеграція з REST API WordPress забезпечила зручне отримання, обробку та виведення контенту, даючи можливість розмежувати відповідальність між frontendom і backendom. Такий підхід дозволив створити модульну, гнучку архітектуру, яка легко масштабується та адаптується під зміни бізнес-вимог.

Особливу увагу було приділено автоматизації процесу керування контентом. Завдяки реалізації інтеграції з REST API WordPress, усі текстові та графічні матеріали, а також елементи навігації можуть редагуватись

безпосередньо з адміністративної панелі CMS. Це мінімізує потребу у програмному втручанні при зміні структури сайту, оновленні вмісту або додаванні нових сторінок. Такий підхід не лише підвищує ефективність роботи редакторів контенту, але й знижує витрати часу на технічну підтримку з боку розробників. На рисунку 4.1 зображено загальний вигляд головної сторінки додатку

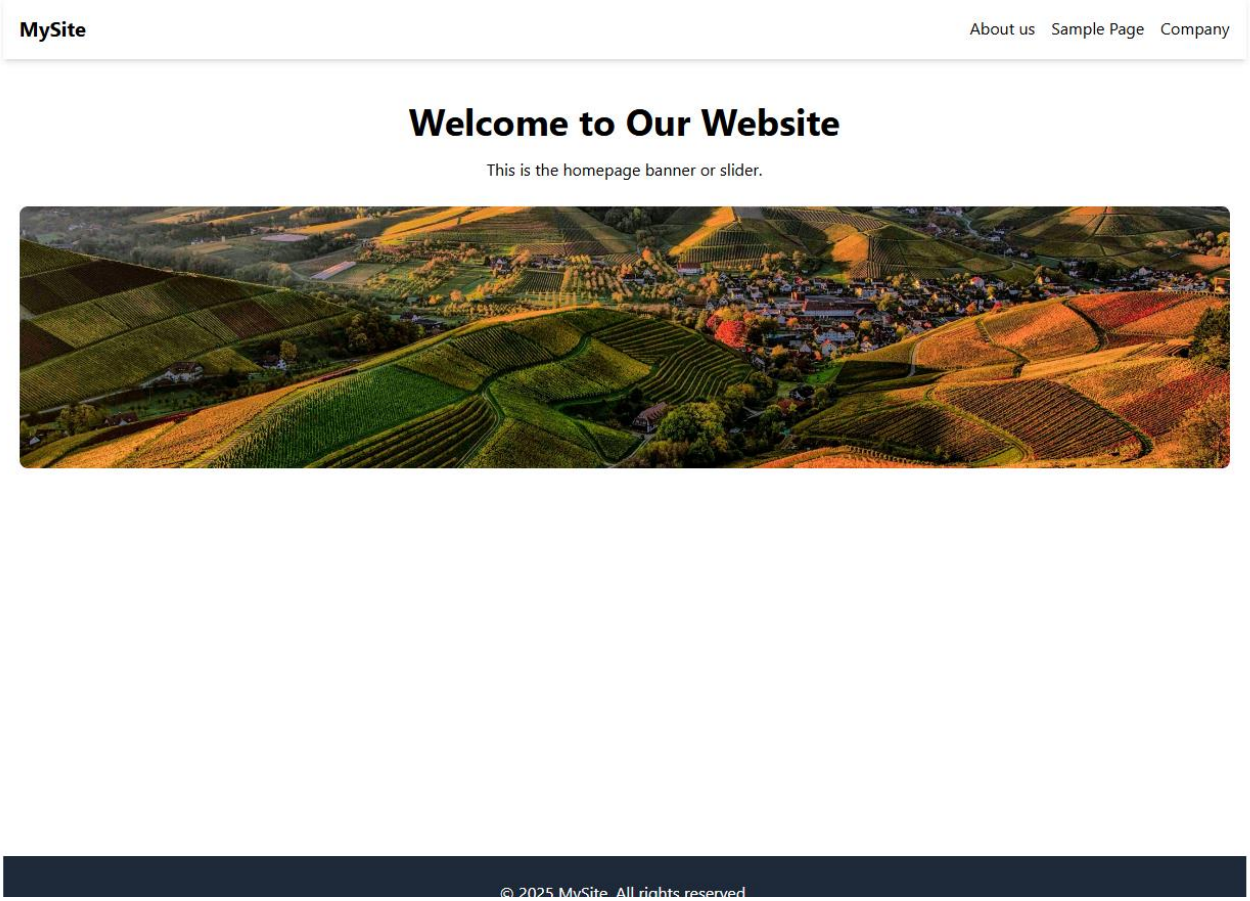


Рисунок 4.1 – Зображення головної сторінки додатку

Розроблений додаток є повністю масштабованим і орієнтованим на подальше розширення функціоналу без необхідності внесення змін у програмний код. Завдяки гнучкій структурі реалізації та використанню Headless CMS, процес додавання нових сторінок до WEB-додатку зводиться до мінімальних дій з боку адміністратора. Зокрема, для створення нової сторінки достатньо обрати відповідний шаблон (загальний компонент сторінки) у межах frontend-додатку, після чого за допомогою CMS WordPress додати новий пункт у

навігаційному меню та наповнити сторінку необхідним контентом через адміністративну панель.

Таким чином, створення та публікація нових розділів сайту не вимагає участі розробника, що значно підвищує ефективність управління контентом. Це є особливо важливим для сайтів з динамічно змінюваною інформацією, де оперативне оновлення структури ресурсу відіграє ключову роль. Зазначена функціональність демонструє практичну перевагу Headless-підходу, що забезпечує високий рівень автоматизації, зручність розширення та підтримки WEB-додатку в умовах реального використання. На рисунку 4.2 зображено процес створення нової сторінки Безпосередньо з адміністративної панелі CMS WordPress.

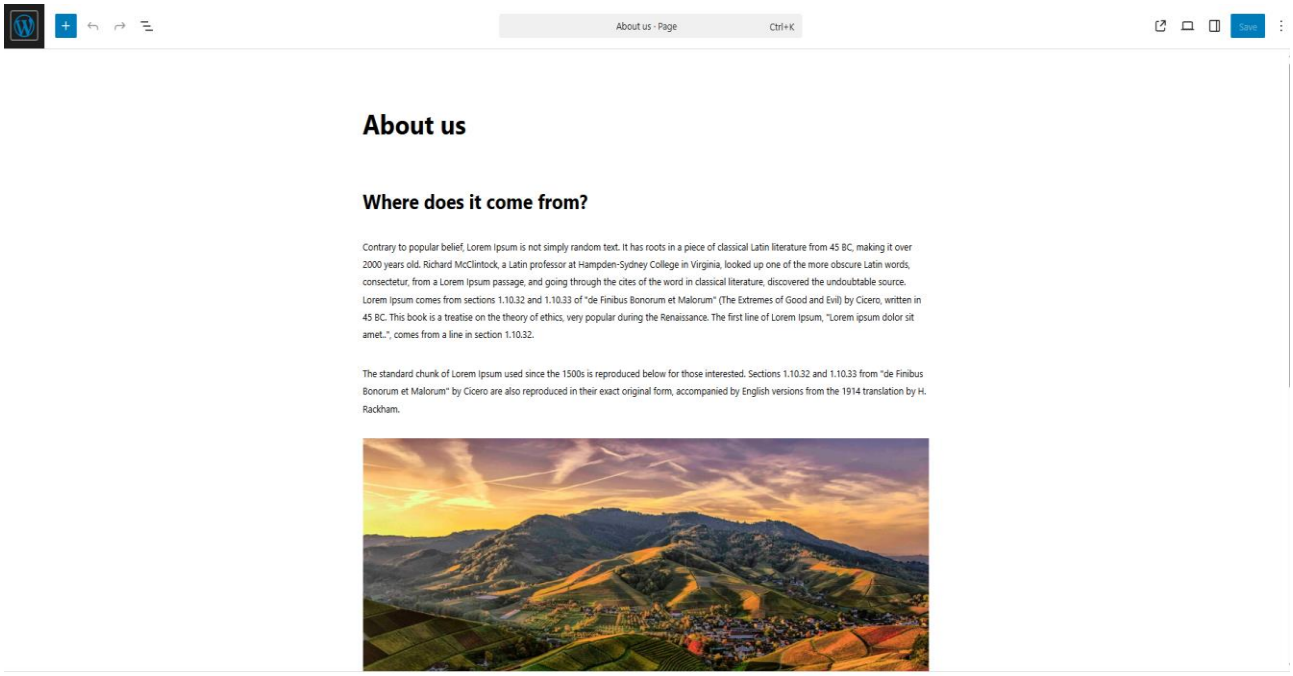


Рисунок 4.2 – Процес створення нової сторінки з адміністративної панелі CMS WordPress

Однією з важливих функціональних можливостей розробленого WEB-додатку є підтримка динамічного формування навігаційного меню, що повністю управляється через адміністративну панель CMS WordPress. Завдяки інтеграції з REST API WordPress стало можливим передавати дані меню з backend-

середовища до frontend-застосунку, створеного на основі React, без потреби у внесенні змін до програмного коду.

Створення меню у WordPress здійснюється через стандартний інтерфейс керування зовнішнім виглядом. Користувач (зазвичай контент-менеджер або адміністратор) формує нове меню, додає до нього необхідні елементи, встановлює їх послідовність і зберігає зміни. Щоб надати можливість frontend звертатися до цього меню, до WordPress додається відповідний плагін (наприклад, WP REST API Menus), який відкриває новий ендпоінт API. Далі цей ендпоінт використовується на стороні React-додатку для отримання масиву об’єктів, які представляють пункти меню.

Таким чином, реалізовано повноцінний механізм взаємодії між CMS і інтерфейсом користувача, що повністю відповідає принципам Headless-архітектури. Це дозволяє адміністраторам самостійно додавати нові сторінки, розділи чи пункти меню, не залучаючи програміста, що значно спрощує процес підтримки й масштабування застосунку. Це зображено на рисунку 4.13

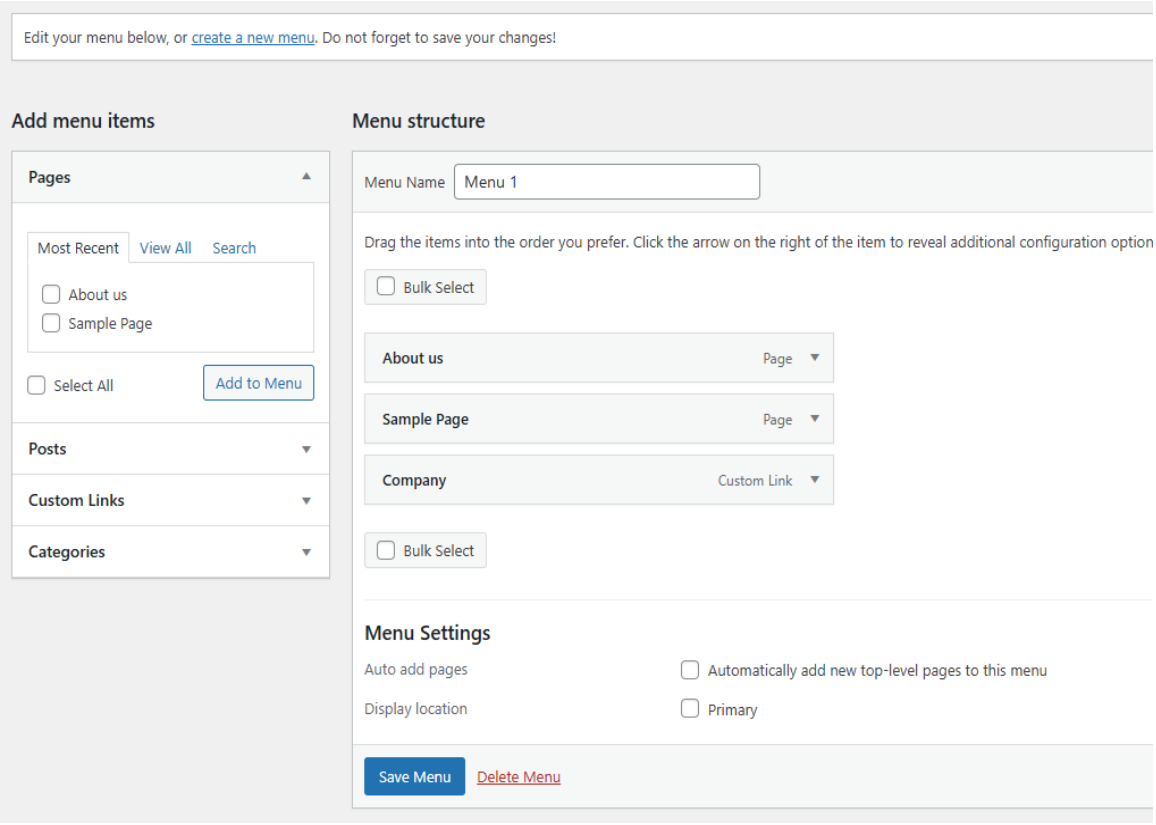


Рисунок 4.3 – Вікно створення або редагування навігаційний пунктів додатку

4.2 Тестування функціоналу модулю блогу

Блог є важливим елементом будь-якого сучасного WEB-додатку, що прагне досягти високих результатів у SEO-оптимізації. Він виступає ключовим інструментом для збільшення органічного трафіку, поліпшення рейтингу в пошукових системах і створення позитивного іміджу вебресурсу. Завдяки публікації якісного та релевантного контенту, який відповідає інтересам цільової аудиторії, блог сприяє довгостроковому зростанню залучення користувачів.

Створення та підтримка блогу дає змогу розширити семантичне ядро WEB-сайту шляхом додавання нових ключових слів у текстах, покращити зв’язність сторінок через внутрішнє посилання, а також залучити додаткові зворотні посилання з інших джерел. Одночасно блог сприяє покращенню поведінкових факторів, таких як середня тривалість перебування користувача на сайті, кількість відвіданих сторінок за одну сесію та зменшення показника відмов [22]. На рисунку 4.4 зображено процес додавання та виведення записів у WEB-додатку.

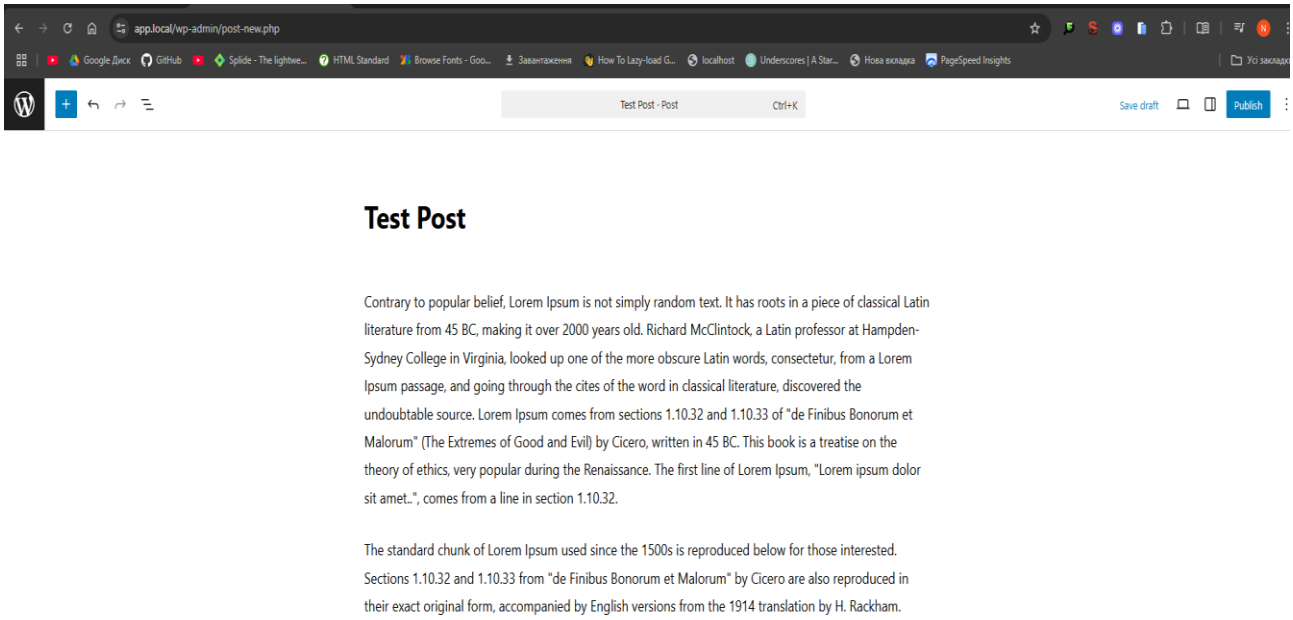


Рисунок 4.4 – Процес додавання записів у CMS WordPress

Як показано на відповідному рисунку, процес створення нового запису у системі керування контентом є інтуїтивно зрозумілим та не вимагає спеціальних технічних знань. Для додавання нового матеріалу достатньо вказати заголовок запису та заповнити його вміст у редакторі. Особливою перевагою даного підходу є можливість редагування контенту безпосередньо за допомогою вбудованих інструментів редактора, що дозволяє швидко форматувати текст, додавати зображення, вставляти посилання та інші мультимедійні елементи. Така функціональність значно спрощує процес оновлення інформаційного наповнення сайту та забезпечує зручний механізм керування контентом для редакторів та адміністраторів ресурсу.

Як видно на рисунку 4.5 – щойно створений запис успішно додався до розділу “Blog” WEB-додатку.

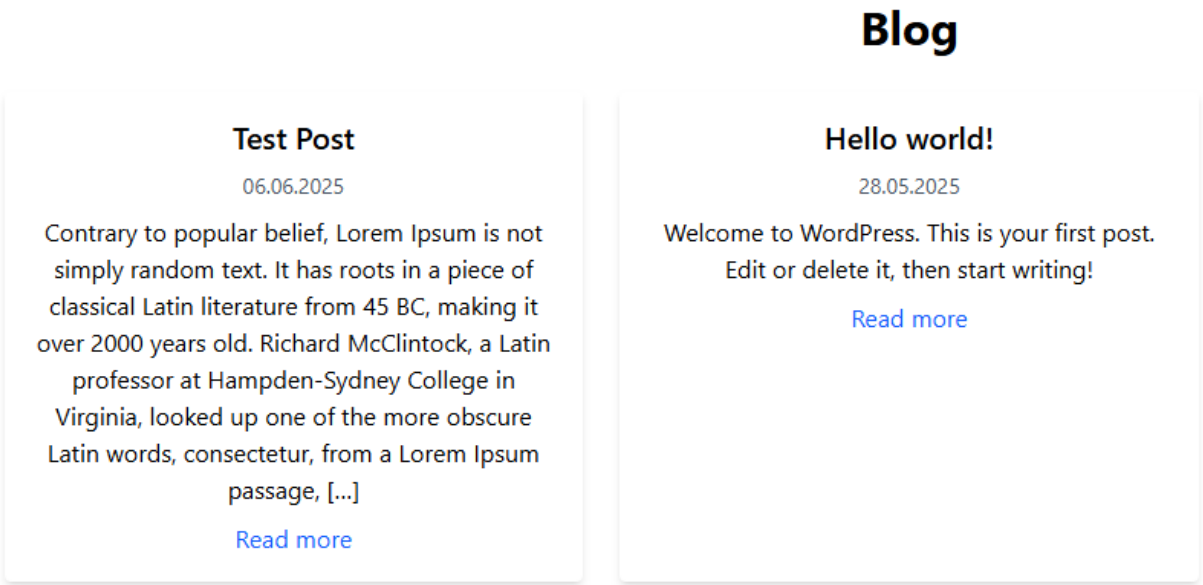


Рисунок 4.5 – Успішне додавання запису до WEB-додатку

Для комплексної оцінки вибраного підходу Headless у порівнянні з традиційним Fullstack підходом, було проведено аналіз основних технічних та функціональних характеристик обох методів розробки WEB-додатків, що наведено у таблиці 4.1. Це дозволяє визначити переваги та обмеження кожного з

підходів у контексті реалізації сучасних, масштабованих та продуктивних систем.

Таблиця 4.1 – Порівняльна характеристика Headless та Fullstack підходів

Критерій	**Headless підхід**	**Fullstack підхід**
Розділення фронтенду і бекенду	Повне (окремі сервіси)	Інтегрований
Гнучкість у виборі технологій фронтенду	Висока (будь-які фреймворки)	Обмежена технологією бекенду
Масштабованість	Висока (можливість окремого масштабування)	Обмежена (масштабування всієї системи)
Продуктивність	Вища (клієнтське кешування, API)	Залежить від серверного рендерингу
Безпека	Вища (ізоляція бекенду)	Нижча (тісна інтеграція компонентів)

На основі проведеного аналізу встановлено, що підхід Headless забезпечує вищий рівень гнучкості, кращу масштабованість та спрощує підтримку WEB-додатку, особливо в умовах швидкого розвитку проєкту та необхідності інтеграції з різноманітними сервісами. У той час як Fullstack-розробка забезпечує повний контроль над усіма аспектами додатку, її реалізація потребує значно більших часових та ресурсних витрат на розробку і підтримку.

З урахуванням зазначених особливостей, застосування Headless архітектури є доцільним рішенням, що дозволило досягти мети бакалаврської роботи – підвищення гнучкості архітектури та оптимізації процесу багатоканальної доставки контенту.

4.3 Висновок до розділу 4

У четвертому розділі було проведено тестування розробленого WEB-додатку на відповідність функціональним вимогам, визначеним на початковому етапі проєкту. В ході тестування було перевірено коректність відображення контенту, стабільність взаємодії з серверною частиною через REST API, швидкодію, адаптивність інтерфейсу для різних типів пристроїв, а також загальну зручність використання системи.

Результати тестування засвідчили, що розроблений додаток повністю відповідає встановленим функціональним вимогам:

- контент динамічно завантажується через API без необхідності перезавантаження сторінок.
- система забезпечує високу швидкість завантаження сторінок.
- здійснено успішну інтеграцію з CMS WordPress у режимі Headless.
- забезпечено стабільну роботу при зміні обсягів контенту.

ВИСНОВКИ

У процесі розробки WEB-додатку усі ключові функціональні вимоги, визначені на етапі проєктування, були повністю враховані та реалізовані. Зокрема, створено повноцінний блоговий функціонал, який передбачає можливість додавання, редагування та публікації контенту в зручній адміністративній панелі. Реалізовано окрему сторінку блогу з посторінковою навігацією, а також сторінку перегляду окремого поста, що дозволяє користувачам швидко взаємодіяти з контентом.

Особливу увагу було приділено створенню адаптивного інтерфейсу, який гарантує коректне відображення додатку на різних типах пристроїв – від десктопів до мобільних телефонів. Інтерфейс відповідає сучасним вимогам доступності, включаючи підтримку навігації з клавіатури та можливість масштабування елементів інтерфейсу.

Загальні вимоги до продуктивності та стабільності також були досягнуті. Проведене тестування показало, що додаток демонструє високу швидкість завантаження сторінок у межах встановлених нормативів і зберігає стабільність роботи навіть за умов підвищеного навантаження. Було впроваджено ключові заходи для забезпечення відповідності актуальним стандартам SEO.

Важливо зазначити, що реалізація усіх зазначених функціональних вимог дозволяє стверджувати про досягнення цілей, поставлених у межах даного проєкту. WEB-додаток відповідає сучасним стандартам розробки, є зручним у використанні для кінцевих користувачів і простим в адмініструванні.

Поставлену мету бакалаврської роботи, яка полягала у підвищенні гнучкості архітектури та оптимізації процесу багатоканальної доставки контенту, було досягнуто. Це стало можливим завдяки використанню системи керування контентом (CMS) як джерела даних, передачі структурованого контенту за допомогою REST API, а також завдяки чіткому відокремленню frontend-частини додатку від backend-логіки, що дозволило реалізувати незалежну, масштабовану і ефективну структуру WEB-додатку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кононенко Н.В., Петришин С.І., Сімончук С.В. – Розробка WEB-додатку для CMS з використанням Headless підходу // Матеріали Міжнародної науково-практичної конференції молодих науковців, аспірантів і студентів «Молодь – науці і практиці: досягнення та перспективи розвитку» (м. Вінниця, 15–16 червня 2025 р.). Вінниця: ВНТУ, 2025. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25535

2. HostIQ – Що таке WordPress та як ним користуватися? [Електронний ресурс]. – Режим доступу: https://hostiq.ua/wiki/ukr/wordpress-review/ (28.05.2025). – Загол. з екрану.

3. Esfirum – Переваги та недоліки CMS WordPress [Електронний ресурс]. – Режим доступу: https://esfirum.com/blog/pros-and-cons-cms-wordpress/ (28.05.2025). – Загол. з екрану.

4. Foxminded – Що таке SPA у програмуванні [Електронний ресурс]. – Режим доступу: https://foxminded.ua/spa-u-prohramuvanni/ (28.05.2025). – Загол. з екрану.

5. Smart Solutions – Види архітектури програмного забезпечення [Електронний ресурс]. – Режим доступу: https://smart-solutions.com.ua/archives/637 (28.05.2025). – Загол. з екрану.

6. Art of BA – Основні типи архітектури програмного забезпечення [Електронний ресурс]. – Режим доступу: https://www.artofba.com/uk/post/main-types-of-software-architecture (28.05.2025). – Загол. з екрану.

7. Wezom – Що таке мікросервісна архітектура: значення, складові, переваги [Електронний ресурс]. – Режим доступу: https://wezom.com.ua/ua/blog/scho-take-mikroservisna-arhitektura-znachennya-skladovi-perevagi (28.05.2025). – Загол. з екрану.

8. Kinsta – PHP Market Share [Електронний ресурс]. – Режим доступу: https://kinsta.com/php-market-share/ (28.05.2025). – Загол. з екрану.

9. GoIT – Що таке Java і де вона використовується? [Електронний ресурс]. – Режим доступу: https://goit.global/ua/articles/shcho-take-java-i-de-vona-vykorystovuietsia/ (28.05.2025). – Загол. з екрану.

10. HostIQ – Що таке SSL [Електронний ресурс]. – Режим доступу: https://hostiq.ua/ukr/info/what-is-ssl/ (28.05.2025). – Загол. з екрану.

11. WordPress.org – Templates [Електронний ресурс]. – Режим доступу: https://codex.wordpress.org/Templates (28.05.2025). – Загол. з екрану.

12. WordPress.org – Template hierarchy [Електронний ресурс]. – Режим доступу: https://developer.wordpress.org/themes/basics/template-hierarchy/ (28.05.2025). – Загол. з екрану.

13. Seahawk Media – How to Build a Website Using the Underscores Theme [Електронний ресурс]. – Режим доступу: https://seahawkmedia.com/wordpress/build-site-using-underscores-theme/ (28.05.2025). – Загол. з екрану.

14. WordPress.org – WordPress Coding Standards [Електронний ресурс]. – Режим доступу: https://developer.wordpress.org/coding-standards/wordpress-coding-standards/ (28.05.2025). – Загол. з екрану.

15. WordPress.org – Extending the REST API: Routes and Endpoints [Електронний ресурс]. – Режим доступу: https://developer.wordpress.org/rest-api/extending-the-rest-api/routes-and-endpoints/ (28.05.2025). – Загол. з екрану.

16. QALight – Що таке JSON? [Електронний ресурс]. – Режим доступу: https://qalight.ua/baza-znaniy/shho-take-json/ (28.05.2025). – Загол. з екрану.

17. Форум DOU [Електронний ресурс]. – Режим доступу: https://dou.ua/forums/topic/53343/ (28.05.2025). – Загол. з екрану.

18. Vite – Official Guide [Електронний ресурс]. – Режим доступу: https://vite.dev/guide/ (28.05.2025). – Загол. з екрану.

19. React Masters – What are components and types of components in React.js [Електронний ресурс]. – Режим доступу: https://medium.com/@reactmasters.in/what-are-components-and-types-of-components-in-react-js-4e2642b136a2 (28.05.2025). – Загол. з екрану.

20. Azevedo V. – DRY (Don’t Repeat Yourself): Principles and Best Practices [Електронний ресурс]. – Режим доступу: https://vitor-azevedo.medium.com/dry-dont-repeat-yourself-principles-and-best-practices-50204cf25870 (28.05.2025). – Загол. з екрану.

21. Geeks for Geeks – ReactJS Router [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/reactjs-router/ (28.05.2025). – Загол. з екрану.

22. Search Engine Journal – Головна сторінка [Електронний ресурс]. – Режим доступу: https://www.searchenginejournal.com/ (28.05.2025). – Загол. з екрану.

23. Методичні вказівки до виконання бакалаврської кваліфікаційної роботи для студентів денної та заочної форм навчання спеціальності 122 – «Комп’ютерні науки» [Електронний ресурс]. – Режим доступу: https://iq.vntu.edu.ua/method/getfile.php?fname=124285.pdf&x=1&card_id=46522&id=124285 (PDF, 55 с.) (28.05.2025). – Загол. з екрану.

ДОДАТКИ

ДОДАТОК А. ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка WEB-додатку для CMS з використанням Headless підходу

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 6,33 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

- У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Петришин С.І.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Кононенко Н.В.
(підпис) (прізвище, ініціали)

ДОДАТОК Б. ФРАГМЕНТ ЛІСТИНГУ ПРОГРАМНОГО КОДУ

```
// components/Header.jsx
import { Link } from 'react-router-dom';
import { useEffect, useState } from 'react';

function Header() {
  const [menuItems, setMenuItems] = useState([]);

  useEffect(() => {
    fetch('https://app.local/wp-json/wp-api-menus/v2/menus/2')
      .then((res) => res.json())
      .then((data) => {
        setMenuItems(data.items || []);
      })
      .catch((err) => {
        console.error('Помилка завантаження меню:', err);
      });
  }, []);

  return (
    <header className="bg-white shadow-md">
      <nav className="container mx-auto px-4 py-4 flex justify-between items-center">
        <Link to="/" className="text-xl font-bold">MySite</Link>
        <ul className="flex gap-4">
          {menuItems.map((item, index) => (
            <li key={index}>
              <Link
                to={item.url.replace('https://app.local', '')}
                className="hover:text-blue-600"
              >
                {item.title}
              </Link>
            </li>
          ))}
```

```
            </ul>
          </nav>
        </header>
    );
}

export default Header;

import { useEffect, useState } from 'react';
import { Link } from 'react-router-dom';

function Blog() {
    const [posts, setPosts] = useState([]);

    useEffect(() => {
        fetch('https://app.local/wp-json/wp/v2/posts')
            .then(res => res.json())
            .then(data => setPosts(data));
    }, []);

    return (
        <section className="container mx-auto px-4 py-10">
            <h1 className="text-3xl font-bold mb-6">Blog</h1>
            <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
                {posts.map(post => (
                    <div key={post.id} className="bg-white shadow-md p-4 rounded">
                        <h2 className="text-xl font-semibold mb-2">{post.title.rendered}</h2>
                        <p className="text-gray-600 text-sm mb-2">{new
Date(post.date).toLocaleDateString()}</p>
                        <div dangerouslySetInnerHTML={{ __html: post.excerpt.rendered }} />
                        <Link to={`/post/${post.slug}`} className="text-blue-600 hover:underline mt-2
block">Read more</Link>
                    </div>
                ))}
            </div>
        </section>
```

```
    );
}

export default Blog;

import React, { useEffect, useState } from 'react';

function Home() {
    const [featuredImage, setFeaturedImage] = useState(null);

    useEffect(() => {
        // Замініть URL на свій домен WordPress і slug/ID головної сторінки
        fetch('https://app.local/wp-json/wp/v2/pages?slug=homepage')
            .then((res) => res.json())
            .then((data) => {
                if (data.length > 0) {
                    const page = data[0];
                    const featuredMediaId = page.featured_media;
                    if (featuredMediaId) {
                        // Запит зображення по ID
                        fetch(`https://app.local/wp-json/wp/v2/media/${featuredMediaId}`)
                            .then((res) => res.json())
                            .then((media) => {
                                setFeaturedImage(media.source_url);
                            });
                    }
                }
            })
            .catch((err) => console.error('Error fetching data:', err));
    }, []);

    return (
        <section className="container mx-auto px-4 py-10">
            <h1 className="text-4xl font-bold mb-4">Welcome to Our Website</h1>
            <p className="mb-6">This is the homepage banner or slider.</p>
```

```
        {featuredImage ? (
          <img
            src={featuredImage}
            alt="Featured"
            className="w-full h-64 object-cover rounded-lg"
          />
        ) : (
          <div className="h-64 bg-blue-100 rounded-lg flex items-center justify-center">
            Slider Placeholder
          </div>
        )}
      </section>
    );
}

export default Home;

function NotFound() {
  return (
    <main className="grid min-h-full place-items-center bg-white px-6 py-24 sm:py-32 lg:px-8">
      <div className="text-center">
        <p className="text-base font-semibold text-indigo-600">404</p>
        <h1 className="mt-4 text-5xl font-semibold tracking-tight text-balance text-gray-900 sm:text-7xl">Page
          not found</h1>
        <p className="mt-6 text-lg font-medium text-pretty text-gray-500 sm:text-xl/8">Sorry, we couldn’t find
          the page you’re looking for.</p>
        <div className="mt-10 flex items-center justify-center gap-x-6">
          <a href="#"
            className="rounded-md bg-indigo-600 px-3.5 py-2.5 text-sm font-semibold text-white shadow-xs hover:bg-indigo-500 focus-visible:outline-2 focus-visible:outline-offset-2 focus-visible:outline-indigo-600">Go
            back home</a>
          <a href="#" className="text-sm font-semibold text-gray-900">Contact support <span
```

```
                    aria-hidden="true">&rarr;</span></a>
                </div>
            </div>
        </main>
    );
}

export default NotFound;



import { useEffect, useState } from 'react';
import { useParams } from 'react-router-dom';

function Post() {
    const { slug } = useParams();
    const [post, setPost] = useState(null);
    useEffect(() => {
        fetch(`https://app.local/wp-json/wp/v2/posts?slug=${slug}`)
            .then(res => res.json())
            .then(data => {
                if (data.length) setPost(data[0]);
            });
    }, [slug]);

    if (!post) return <div className="text-center py-10">Loading...</div>;
    return (
        <article className="container mx-auto px-4 py-10">
            <h1 className="text-3xl font-bold mb-4">{post.title.rendered}</h1>
            <p className="text-gray-600 text-sm mb-4">{new
Date(post.date).toLocaleDateString()}</p>
            <div dangerouslySetInnerHTML={{ __html: post.content.rendered }} />
        </article>
    );
}

export default Post;
```

ДОДАТОК В. ІНСТРУКЦІЯ КОРИСТУВАЧА

У розробленому WEB-додатку, що реалізує Headless-підхід, використовується локальне середовище на основі інструменту Local. Після запуску створеного проєкту WordPress ініціалізується серверна частина, що виступає в ролі CMS із відкритим REST API. WEB-інтерфейс WordPress доступний за адресою, вказаною при створенні сайту в середовищі Local, і дозволяє додавати та редагувати контент, який у подальшому динамічно підвантажується у клієнтську частину через запити до API.

Щоб запустити backend частину додатку, потрібно відкрити середовище Local та вибрати потрібний локальний проєкт, що використовує CMS WordPress. Це зображено на рисунку В.1.

Після запуску локального середовища автоматично активуються всі необхідні сервіси — WEB-сервер, база даних та PHP-інтерпретатор. У разі потреби можна налаштувати параметри середовища вручну, наприклад, змінити версію PHP або встановити додаткові розширення. Адмін-панель WordPress стає доступною у браузері за відповідною адресою, і саме звідти здійснюється управління контентом, що виводиться на фронтенді через REST API.

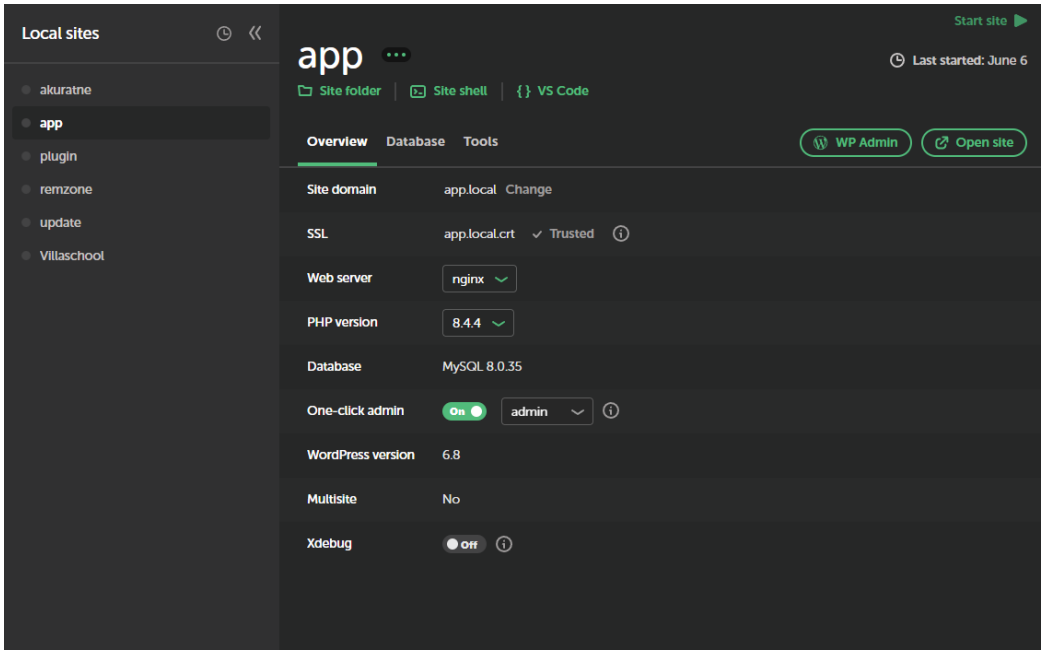


Рисунок В.1 – Процес запуску backend частини у середовищі Local

Після успішного запуску, потрібно протестувати API, що безпосередньо передає дані у форматі JSON. Для цього достатньо перейти за посиланням https://app.local/wp-json/wp/v2/posts. У разі коректної конфігурації та роботи серверної частини, backend повинен повернути валідний JSON-файл зі списком публікацій. Таким чином перевіряється функціональність REST API WordPress, який забезпечує передачу структурованих даних для подальшого відображення на стороні клієнта. На рисунку В.2 зображено успішну відповідь від серверної частини додатку.

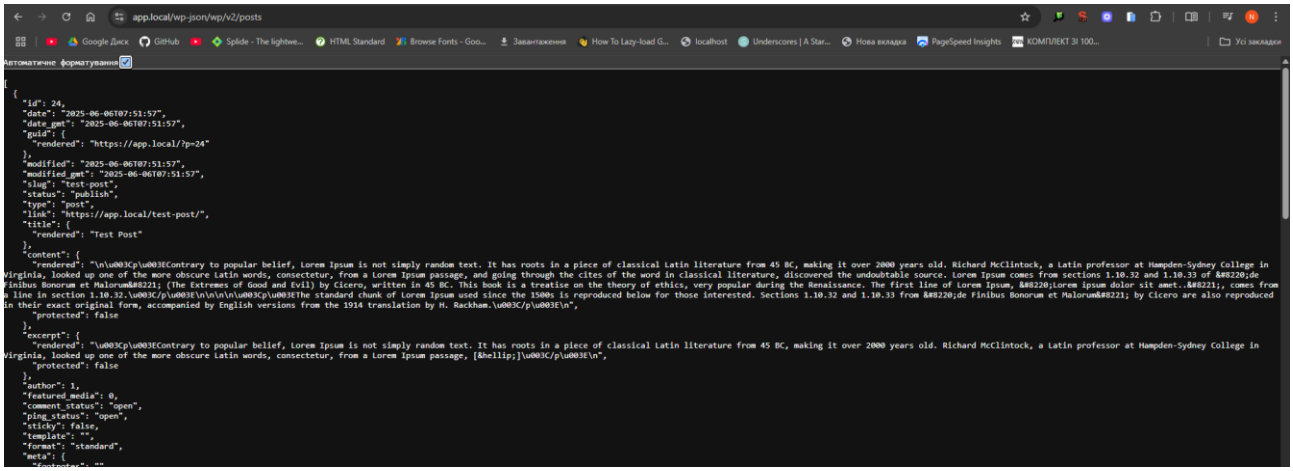


Рисунок В.2 – Успішна відповідь у форматі JSON від серверної частини додатку

Після успішного запуску серверної частини, наступним кроком є розгортання клієнтської частини WEB-додатку, реалізованої з використанням бібліотеки React. Для цього необхідно відкрити термінал, перейти в кореневу директорію frontend проєкту за допомогою команди cd, після чого здійснити встановлення всіх необхідних залежностей.

Процес ініціалізації пакунків відбувається за допомогою команди npm install, яка читає файл package.json і автоматично завантажує всі потрібні бібліотеки до локальної папки node_modules. Цей крок є обов'язковим перед будь-яким запуском додатку, оскільки без встановлених залежностей неможливо зібрати або протестувати frontend функціональність. На рисунку В.3 зображено успішне встановлення пакетів npm.

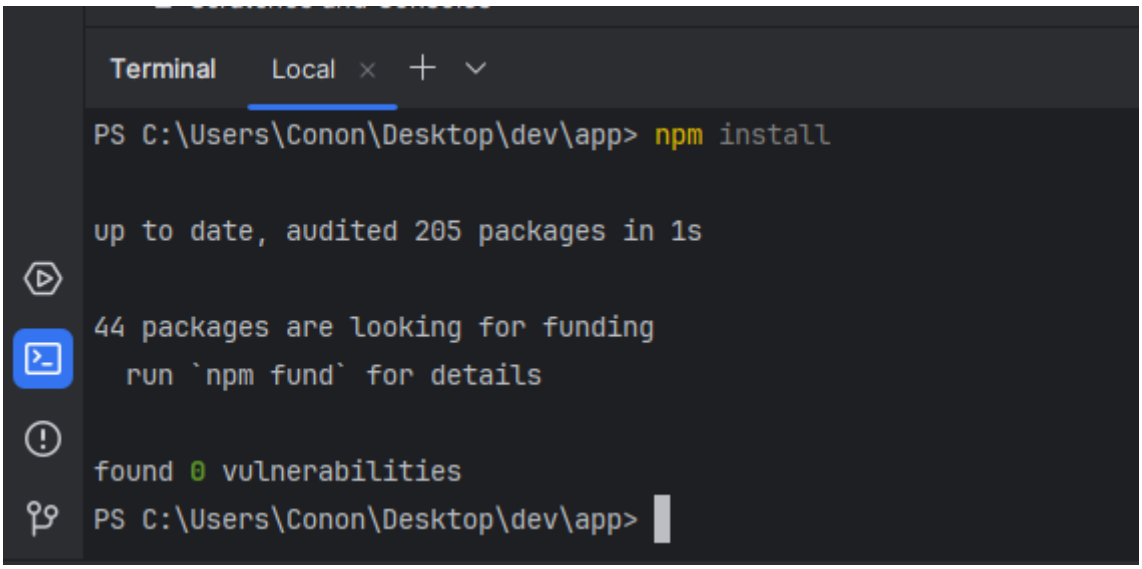


Рисунок В.3 – Успішне встановлення пакетів npm

Після завершення встановлення залежностей, для запуску frontend-серверу використовується команда npm run dev. У результаті система автоматично відкриває веб-додаток у браузері за адресою http://localhost:5173. У випадку зайнятості цього порту іншим процесом, React автоматично запропонує використання іншого доступного порту.

Якщо API з боку WordPress-сервера доступне, інтерфейс додатку завантажується коректно, і на головній сторінці відображається контент, отриманий з CMS через REST API. На рисунку В.4 зображено інтерфейс додатку після успішного підключення до API.

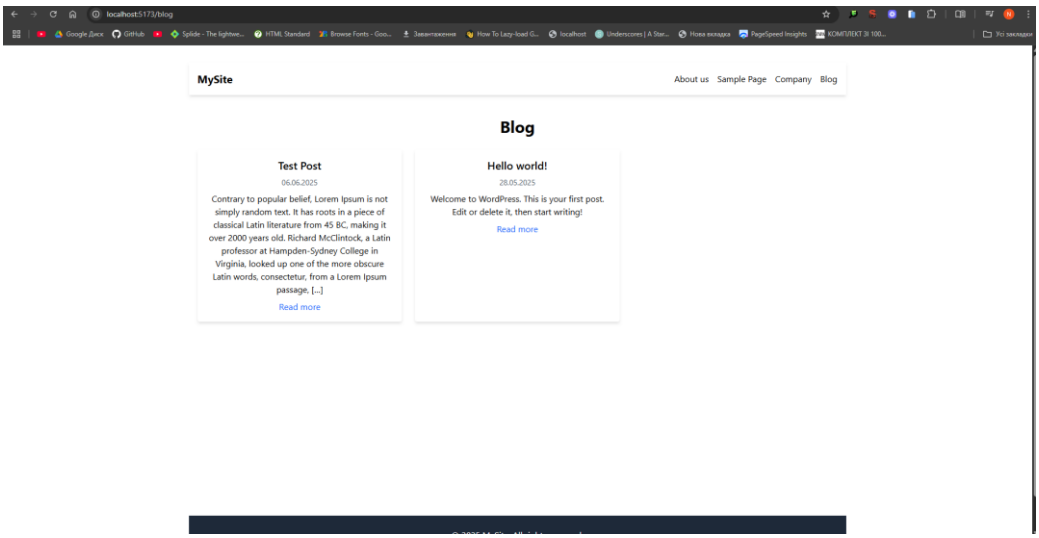


Рисунок В.4 – Успішно запущений WEB-додаток

ДОДАТОК Г. ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА WEB-ДОДАТКУ ДЛЯ CMS З ВИКОРИСТАННЯМ HEADLESS ПІДХОДУ

Виконав: студент 4 курсу, групи 2КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Назарій КОНОНЕНКО
(прізвище та ініціали)
Керівник: к.т.н, старший викладач каф. КН
Сергій ПЕТРИШИН
(прізвище та ініціали)
«03» червня 2025 р.

Вінниця ВНТУ – 2025 рік

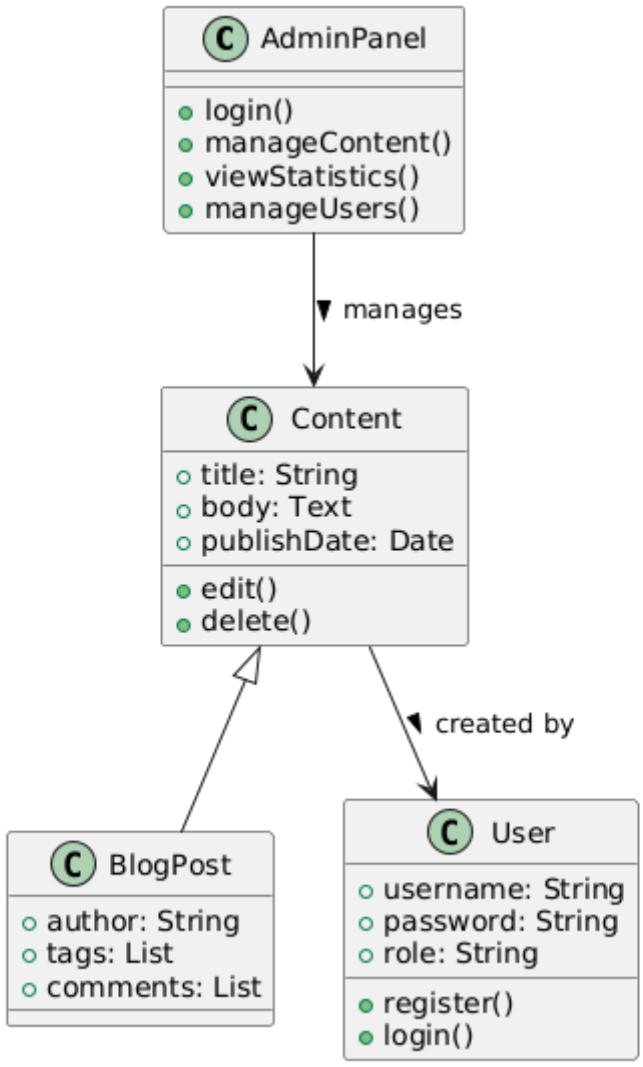


Рисунок Г.1 − Зображення UML-діаграми компонентів додатку

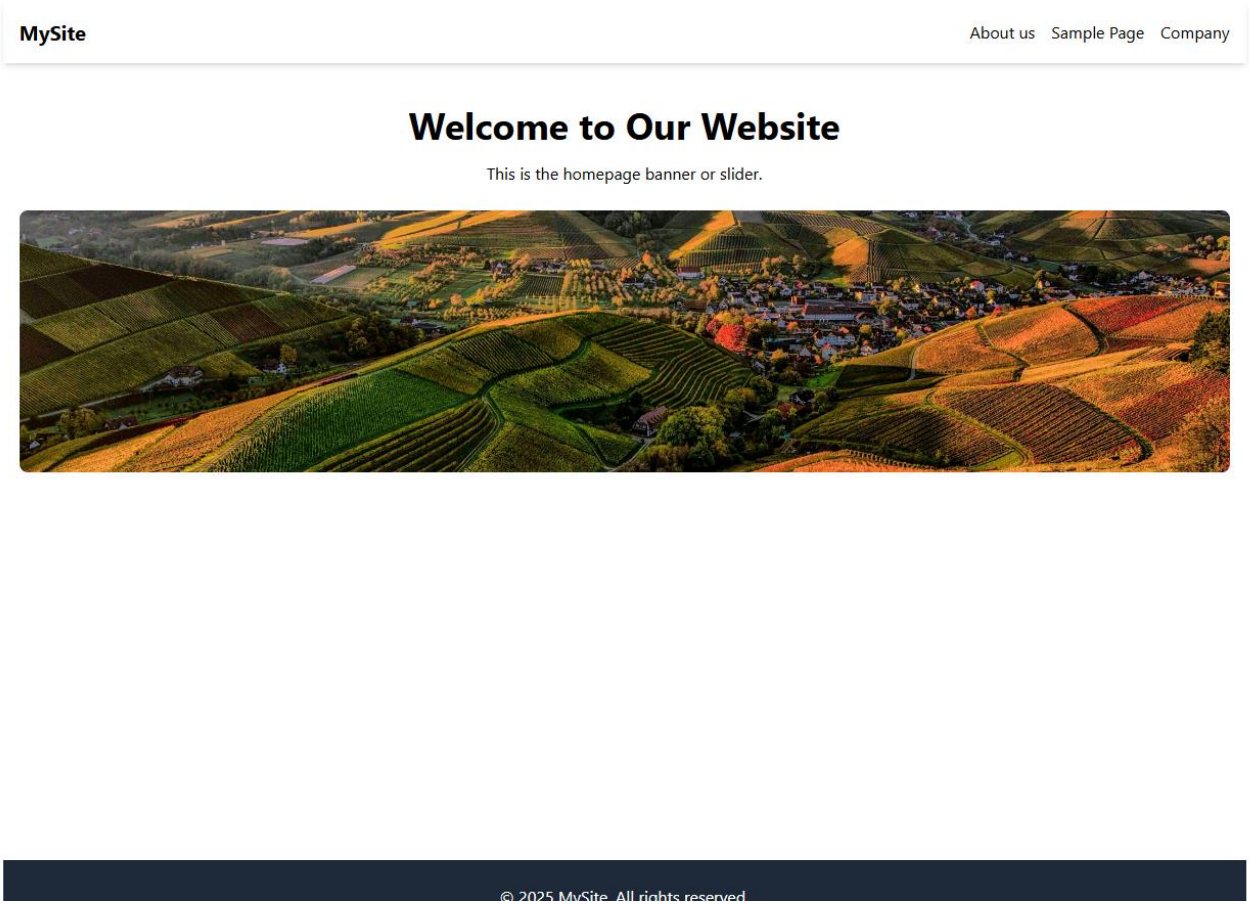


Рисунок Г.2 – Зображення головної сторінки додатку

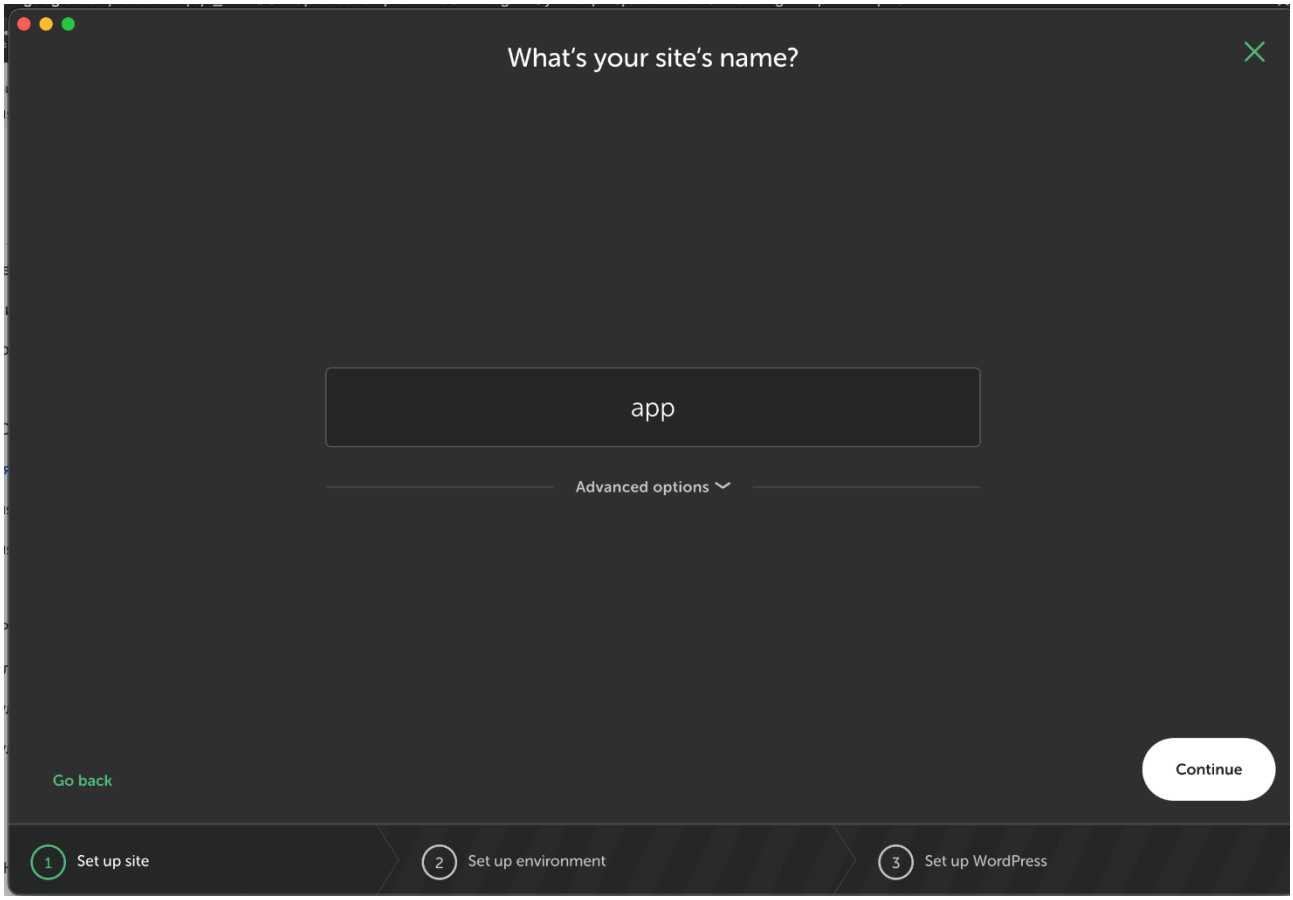


Рисунок Г.3 – Процес розгортання CMS WordPress на локальному сервері у додатку Local

Рисунок Г.4 – Ієрархія шаблонів CMS WordPress

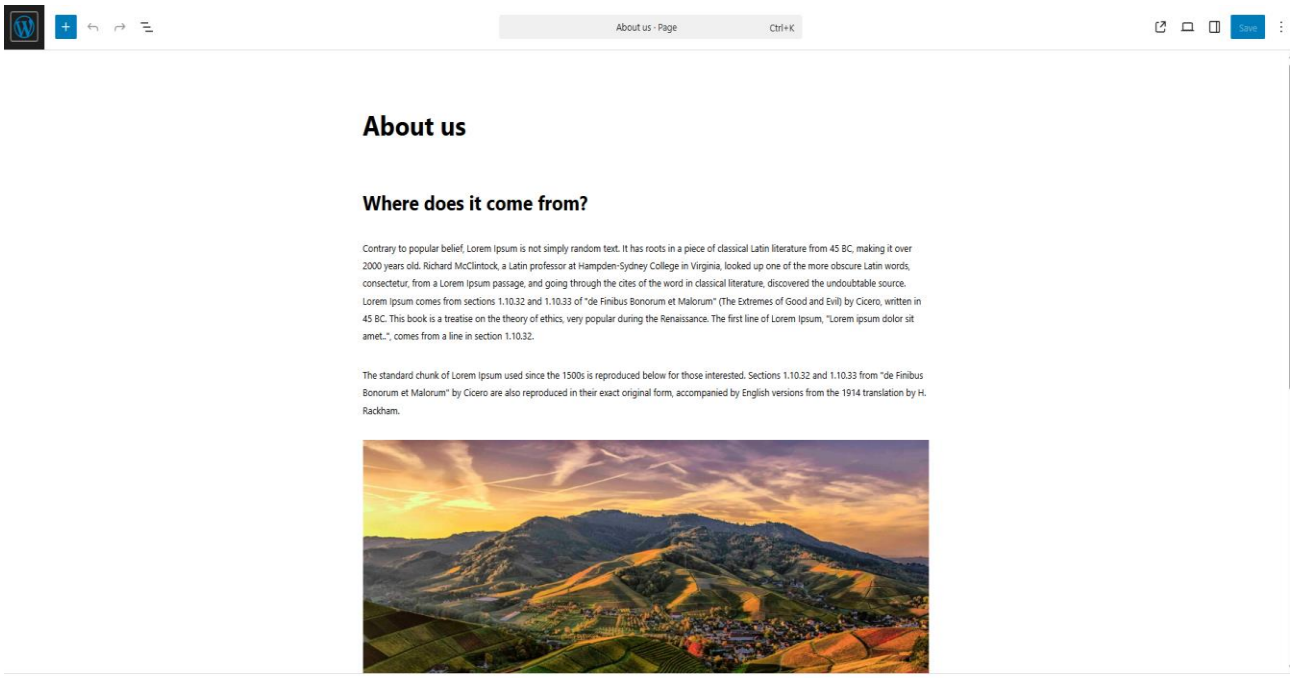


Рисунок Г.5 – Процес створення нової сторінки з адміністративної панелі CMS WordPress