КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра технологій управління

Спеціальність 122 «Комп'ютерні науки»

Освітньо-наукова програма «Управління проектами»

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему:

«Дослідження процесів управління проєктом розробки Headless CMS у вигляді веб-інтерфейсу»

Студента 2-го курсу групи УП-22

Артема ЮРЕЧКА

_____
(підпис студента)

Науковий керівник:

кандидат техн. наук, доцент
Олександр ТІМІНСЬКИЙ

_____ _____
(дата) (підпис)

Попередній захист:

_____

(Висновок: «До захисту в Екзаменаційній комісії»)

Завідувач кафедри
технологій управління _____ Віктор МОРОЗОВ _____
(підпис) (прізвище, ініціали) (дата)

Київ-2025

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра технологій управління
Освітній рівень Магістр
Спеціальність 122 Комп'ютерні науки
Освітня програма Управління проектами

ЗАТВЕРДЖУЮ
Завідувач кафедри
професор Морозов В.В.
_____
"27" листопада 2024 року

ЗАВДАННЯ
НА ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Студент: Юречко Артем Олександрович
Група: УП-22

1. Тема кваліфікаційної роботи: «Дослідження процесів управління проєктом розробки Headless CMS у вигляді веб-інтерфейсу» Затверджена протоколом № 5 від "26" листопада 2024 року.

2. Строк подання студентом готової роботи – "19" травня 2025 р.

3. Цільова установка та вихідні дані до роботи: метою роботи є дослідження та практичне застосування сучасних підходів до управління IT-проєктом на прикладі розробки Headless CMS з веб-інтерфейсом. У фокусі – вивчення процесів ініціалізації, планування, реалізації, контролю та завершення проєкту із урахуванням специфіки створення складного програмного продукту. Особлива увага приділяється вибору методології управління, формуванню структури команди, розподілу обов'язків, а також управлінню ризиками, термінами, ресурсами та бюджетом. Вихідними даними є сучасні тенденції у сфері веб-розробки, практики впровадження Headless CMS, доступні інструменти управління проєктами та попередній досвід реалізації подібних програмних рішень.

4. Зміст роботи: аналіз предметної галузі, обґрунтування доцільності реалізації проєкту, формування мети, цілей та очікуваних результатів проєкту. Проведення PEST- та SWOT-аналізу, побудова дерева проблем і дерева цілей. Визначення життєвого циклу проєкту, формування команди та розподіл ролей, розробка організаційної структури. Планування етапів реалізації проєкту, створення беклогу, календарне планування із застосуванням сучасних інструментів управління проєктами. Розробка структури та модулів програмного забезпечення, проектування інтерфейсу користувача, управління

якістю, ризиками, бюджетом та зацікавленими сторонами. Здійснення моніторингу виконання проєкту та оцінка його ефективності.

5. Перелік графічного матеріалу (слайдів): актуальність теми, постановка цілей і завдань, аналіз ринку CMS, модель організаційної структури команди, WBS-модель проєкту, діаграма Ганта, фінансовий план, архітектура системи, схема взаємодії модулів, інтерфейс користувача, оцінка ризиків, підсумкові висновки.

6. Календарний план виконання роботи:

№ з/п	**Назва частин роботи**	**Виконання роботи**
1	Вивчення літературних джерел з предмету дослідження	12.12.24-19.12.24
2	Вивчення матеріалів досліджуваної теми	20.12.24-25.12.24
3	Складання розгорнутого плану кваліфікаційної роботи	26.12.24-27.12.24
4	Ознайомлення наукового керівника з планом кваліфікаційної роботи	22.01.25
5	Підготовка розділу 1	05.02.25-26.02.25
6	Підготовка розділу 2	27.02.25-16.03.25
7	Підготовка розділу 3	17.03.25-06.04.25
8	Підготовка розділу 4	07.04.25-17.04.25
9	Оформлення кваліфікаційної роботи	22.04.25-02.05.25
10	Передача роботи науковому керівникові	03.05.25
11	Передача роботи на рецензування	06.05.25
12	Попередній захист кваліфікаційної роботи	12.05.25
12	Передача кваліфікаційної роботи на плагіат	16.05.25-25.05.25
13	Захист кваліфікаційної роботи	26.05.25-28.05.25

Дата видачі завдання: “06” листопада 2023 р.
Керівник роботи: кандидат техн. наук, доцент ТІМІНСЬКИЙ Олександр

_____
(підпис)

Завдання прийняв до виконання:
студент групи УП-22 ЮРЕЧКО Артем _____
(підпис)

ЗМІСТ

АНОТАЦІЯ **7**
ТАБЛИЦЯ СКОРОЧЕНЬ ТА ПОЯСНЕНЬ **8**
ВСТУП **10**
РОЗДІЛ 1. АНАЛІЗ ТА ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ ПРОЄКТУ **13**
1.1 Аналіз предметної галузі 13
1.1.1 Огляд сучасних систем управління контентом (CMS) 13
1.1.2 Headless CMS: переваги, недоліки, перспективи 14
1.1.3 Стан розвитку ринку Headless CMS та веб-технологій 17
1.2 Методології управління ІТ-проєктами 17
1.2.1 Роль та значення методологій в управлінні ІТ-проєктами 17
1.2.2 Огляд сучасних методологій управління ІТ-проєктами 18
1.2.3 Порівняння методологій управління ІТ-проєктами та обґрунтування вибору 21
1.3 Маркетингове обґрунтування проєкту 23
1.3.1 Аналіз конкурентного середовища у сфері CMS 23
1.3.2 STEP-аналіз зовнішнього середовища 26
1.3.3 Аналіз цільової аудиторії та зацікавлених сторін 28
1.3.4 Оцінка проєктних альтернатив 30
1.3.4.1 Визначення можливих підходів до реалізації проєкту 30
1.3.4.2 SWOT-аналіз альтернатив 31
1.3.5 Аналіз внутрішнього середовища проєкту 36
1.4 Побудова логіко-структурної моделі проєкту 38
1.4.1 Побудова дерева проблем 38
1.4.2 Побудова дерева цілей 39
1.4.3 Побудова логіко-структурної схеми проєкту 41
1.5 Інвестиційні дослідження 43

1.6 Мета, завдання та очікувані результати проєкту 48
РОЗДІЛ 2. ПРОЄКТНЕ ПЛАНУВАННЯ РОЗРОБКИ HEADLESS CMS 50
2.1 Побудова WBS: за фазами життєвого циклу, продуктами та процесами 50
2.1.1 Ієрархія за фазами життєвого циклу проєкту 50
2.1.2 Ієрархія за продуктами проєкту 52
2.1.3 Ієрархія за процесами проєкту 54
2.2 Організаційна модель компанії та проєктної команди 56
2.2.1 Організаційна структура компанії 56
2.2.2 Ролі учасників та зони відповідальності 59
2.2.3 Матриця відповідальності за напрямами робіт 60
2.3 Календарне планування етапів реалізації 63
2.4 Планування ресурсів проєкту 72
2.4.1 Людські ресурси 72
2.4.2 Матеріальні ресурси 73
2.4.3 Планування використання ресурсів у часі 74
2.5 Формалізація задачі та математичне моделювання 76
2.5.1 Визначення основних змінних та параметрів 76
2.5.2 Математичне моделювання процесу обробки запитів 77
2.5.3 Система обробки запитів 77
2.5.4 Визначення вимог до пропускної здатності 78
2.5.5 Безпека даних 79
2.5.6 Персоналізація контенту 80
2.5.7 Оптимізація роботи системи 80
РОЗДІЛ 3. ТЕХНІЧНЕ ПРОЄКТУВАННЯ, РОЗРОБКА ТА ТЕСТУВАННЯ СИСТЕМИ 82
3.1 Розробка архітектури програмного забезпечення Headless CMS 82
3.2 Проєктування бази даних для Headless CMS 84
3.2.1 Розробка концептуальної моделі бази даних 84
3.2.2 Розробка логічної моделі бази даних 88

3.2.3 Розробка фізичної моделі бази даних 89
3.3 Розробка ключових алгоритмів функціонування CMS 92
3.3.1 Алгоритм для створення та зберігання контенту 92
3.3.2 Алгоритм для отримання контенту 93
3.3.3 Алгоритм для оновлення контенту 94
3.3.4 Алгоритм для видалення контенту 95
3.4 Розробка інтерфейсу Headless CMS 96
3.5 Контроль якості розробки та тестування системи 98
РОЗДІЛ 4. ВПРОВАДЖЕННЯ ТА РОЗВИТОК HEADLESS CMS 102
4.1 Управління ризиками 102
4.2 Моніторинг виконання проєкту 110
4.3 Аналіз технічної готовності до запуску системи 114
4.4 Аналіз ефективності впровадження Headless CMS 117
4.5 Стратегія подальшого розвитку та масштабування 119
ВИСНОВКИ 122
ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ 124
ДОДАТОК А 130
ДОДАТОК Б 134
ДОДАТОК В 136

АНОТАЦІЯ

Темою даної роботи є «Дослідження процесів управління проєктом розробки Headless CMS у вигляді веб-інтерфейсу». Предметна область охоплює процеси управління IT-проєктами в контексті створення інноваційного інструменту для роботи з цифровим контентом.

Метою роботи є дослідження та впровадження ефективних підходів до управління проєктом розробки Headless CMS, що включає планування, організацію, контроль та моніторинг процесів розробки, формування стратегії реалізації проєкту, визначення його етапів, структури та складу команди, а також розробку ключових IT-компонентів. Особливу увагу приділено побудові архітектури системи, створенню інтерфейсу користувача та впровадженню засобів гнучкого управління розробкою.

Об'єктом дослідження є галузь створення систем керування контентом.

Предметом дослідження є процеси управління проєктом, зокрема управління змістом, термінами, ресурсами, ризиками, якістю, зацікавленими сторонами та програмними компонентами у межах життєвого циклу IT-проєкту.

Наукова новизна роботи полягає у поєднанні принципів розробки Headless CMS із сучасними методами управління IT-проєктами, зокрема застосуванням Scrum та інструментів гнучкого планування. У рамках роботи було створено модель процесу управління проєктом, адаптовану до специфіки розробки CMS-систем з відокремленим фронтендом.

Кваліфікаційна робота магістра складається зі вступу, основної частини, яка включає чотири розділи, висновків, списку використаних джерел та додатків.

Робота містить 138 сторінок, 23 рисунки та 11 таблиць.

ТАБЛИЦЯ СКОРОЧЕНЬ ТА ПОЯСНЕНЬ

У даній кваліфікаційній роботі використовується ряд скорочень, які наведено нижче з метою уникнення повторень та полегшення сприйняття матеріалу.

Таблиця «*Скорочень та пояснень*»

Скорочення	**Пояснення**
1	**2**
ПЗ	Програмне забезпечення
CMS	Система керування контентом (Content Management System)
Headless CMS	«Безголова» система керування контентом, у якій “тіло” (бекенд для зберігання та керування контентом) повністю відокремлене від “голови” (фронтенду для відображення контенту).
UI	Інтерфейс користувача (User Interface)
UX	Досвід користувача (User Experience)
API	Інтерфейс прикладного програмування (Application Programming Interface)
БД	База даних
SQL	Мова структурованих запитів (Structured Query Language)
HTML	Мова гіпертекстової розмітки (HyperText Markup Language)
CSS	Каскадні таблиці стилів (Cascading Style Sheets)
JS	JavaScript – мова програмування для вебінтерфейсів

Продовження табл. «*Скорочень та пояснень*»

1	2
PHP	Мова серверного програмування PHP
MVC	Архітектурний шаблон Model-View-Controller
REST	Архітектурний стиль взаємодії систем (Representational State Transfer)
JSON	Формат обміну даними (JavaScript Object Notation)
CI/CD	Безперервна інтеграція та розгортання (Continuous Integration / Continuous Deployment)
MVP	Мінімально життєздатний продукт (Minimum Viable Product)
ЖЦ	Життєвий цикл
WBS	Ієрархічна структура робіт (Work Breakdown Structure)
Git	Система контролю версій
KPI	Ключовий показник ефективності (Key Performance Indicator)

ВСТУП

Сучасний веб-простір стрімко змінюється. Бізнеси, освітні платформи, медіа та інші організації щодня продукують великий обсяг цифрового контенту, який потрібно зберігати, оновлювати та зручно управляти. З огляду на це, зростає потреба у гнучких, масштабованих та технологічно незалежних системах керування контентом. Саме таким рішенням є Headless CMS – система керування контентом, що відокремлює бекенд (керування даними) від фронтенду (відображення даних).

Завдяки архітектурі Headless CMS компанії отримують можливість реалізовувати будь-які фронтенд-рішення: вебсайти, мобільні додатки, електронні табло тощо. Це відкриває нові можливості для створення адаптивних, мультиплатформенних продуктів та підвищення ефективності команд розробки. Водночас управління проєктами створення подібних систем вимагає високого рівня організації, планування та технічної компетентності.

Метою даної роботи є розробка концепції та дослідження процесів управління проєктом розробки Headless CMS з веб-інтерфейсом, адаптованої до вимог сучасної цифрової інфраструктури.

Об'єктом дослідження є галузь створення систем керування контентом.

Предметом дослідження є процеси управління проєктом, зокрема управління змістом, термінами, ресурсами, ризиками, якістю, зацікавленими сторонами та програмними компонентами у межах життєвого циклу IT-проєкту.

Основними *завданнями* даної роботи є:

- проведення аналізу предметної галузі;
- проведення маркетингових та інвестиційних досліджень;
- виявлення проблем, визначення цілей і альтернатив проєкту;
- опис продукту, його цільового призначення та основних функцій;
- розробка життєвого циклу проєкту;
- побудова ієрархічної структури робіт;

- формування організаційної структури команди проєкту та розподіл ролей;
- календарне планування виконання проєкту;
- оцінка потреб у ресурсах та вартості проєкту;
- розробка критеріїв забезпечення якості продукту;
- визначення можливих ризиків та стратегії їх уникнення;
- планування закупівель у межах проєкту;
- визначення інтересів і взаємодії із зацікавленими сторонами;
- моніторинг реалізації проєкту з використанням методу освоєного обсягу.

На першому етапі дослідження теми було застосовано метод вивчення наявних результатів досліджень, наукових публікацій, технічної документації та прикладів реалізації подібних проєктів. Для глибокого розуміння структури майбутньої системи були використані методи аналізу та структурування: аналіз дозволив дослідити кожен компонент окремо, оцінити його функціональні можливості та обмеження, а структурування – впорядкувати отриману інформацію, встановити зв’язки між елементами та сформувати цілісне бачення системи.

На подальших етапах планування проєкту застосовувались методи моделювання, декомпозиції та побудови критичного шляху. Для побудови календарного плану, визначення залежностей між завданнями, розрахунку тривалості робіт та візуалізації структури проєкту використовувалось програмне забезпечення Merlin Project, адаптоване для macOS. Для створення беклогу, управління завданнями та командною взаємодією у межах гнучкої методології Scrum було задіяно онлайн-інструмент Trello.

Наукова новизна отриманих результатів полягає у застосуванні адаптованої моделі управління проєктом до процесу створення Headless CMS, що дозволяє забезпечити гнучкість у розробці, масштабованість архітектури та простоту інтеграції з різними каналами доставки контенту. Також у роботі проаналізовано доцільність використання саме безголової архітектури для

реалізації сучасних ІТ-рішень.

Отримані *результати* дослідження мають практичне значення для команд розробників, менеджерів проєктів та організацій, що планують створення або впровадження власної CMS. Запропонований підхід дозволяє ефективно поєднати управлінські процеси з технічними особливостями розробки, підвищити якість продукту та зменшити ризики, пов’язані з невизначеністю під час реалізації ІТ-проєктів.

РОЗДІЛ 1. АНАЛІЗ ТА ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ ПРОЄКТУ

1.1 Аналіз предметної галузі

1.1.1 Огляд сучасних систем управління контентом (CMS)

Системи управління контентом (CMS, від англ. Content Management System) стали невіддільною частиною сучасного цифрового середовища. Вони забезпечують зручне керування текстовим, візуальним та мультимедійним контентом без потреби в глибоких знаннях програмування [1]. Основна мета CMS – спростити процес створення, редагування, публікації та підтримки цифрового контенту для користувачів різного рівня технічної підготовки.

З моменту свого виникнення CMS пройшли значну еволюцію – від монолітних архітектур, де бекенд (логіка, управління контентом, зберігання даних) і фронтенд (інтерфейс користувача) були жорстко поєднані в одне ціле, – до гнучких API-орієнтованих рішень, що дозволяють відокремити логіку зберігання контенту від його відображення [2]. Завдяки цьому веб-розробники отримали більше свободи у виборі інструментів, а компанії – можливість створювати мультиплатформенні рішення.

Серед найбільш відомих традиційних CMS варто виділити:

- WordPress – найпопулярніша система, яка займає понад 40% світового ринку сайтів [3]. Вона характеризується широким вибором шаблонів і плагінів, активною спільнотою та великою кількістю готових рішень;
- Joomla – функціональна CMS із розширеними можливостями керування структурою контенту, яка потребує трохи вищої технічної підготовки;
- Drupal – система з високим рівнем гнучкості та безпеки, часто використовується для складних або державних проєктів [4];
- Magento – CMS для електронної комерції, що дозволяє створювати масштабовані інтернет-магазини зі складною логікою.

Традиційні CMS мають багато переваг: візуальний інтерфейс редагування контенту, підтримку шаблонів, розширень, SEO-інструментів, локалізацію тощо. Водночас, з розвитком цифрового середовища та появою нових каналів доставки контенту (мобільні додатки, соціальні мережі, смарт-пристрої, голосові асистенти), класичні CMS почали виявляти певні обмеження [5].

Зокрема, їхня зв’язність із шаблонним інтерфейсом ускладнює повторне використання контенту на різних платформах. Крім того, одночасне оновлення бекенду та фронтенду часто потребує змін у всій системі, що знижує гнучкість розробки.

У відповідь на ці виклики з’явився новий підхід – Headless CMS (буквально: «безголова» CMS). Такі системи дозволяють створювати, зберігати й управляти контентом у бекенді, а потім передавати його через API будь-якому фронтенду, незалежно від платформи [6]. Headless CMS значно краще підходять до сучасних мультиканальних стратегій і забезпечують вищу масштабованість та контроль над користувацьким інтерфейсом.

Таким чином, сучасна екосистема CMS охоплює як класичні монолітні рішення, що зручні для швидкого старту, так і більш гнучкі архітектури Headless, які відповідають викликам складних і технологічно вимогливих проєктів [7].

1.1.2 Headless CMS: переваги, недоліки, перспективи

На відміну від традиційних CMS, де інтерфейс користувача жорстко пов’язаний із логікою зберігання та керування контентом, Headless CMS працює за принципом повного розділення “бекенду” (де зберігаються й управляються дані) та “фронтенду” (де ці дані відображаються). Контент у таких системах передається через API, що дозволяє використовувати його на різних платформах одночасно – вебсайтах, мобільних додатках, інформаційних панелях тощо [8].

Такий підхід має низку переваг, які обумовлюють стрімке зростання

популярності Headless CMS серед сучасних команд розробки [9]:

1. забезпечення повної свободи у виборі технологій та підходів для створення інтерфейсу користувача;
2. можливість багаторазового використання контенту в різних цифрових каналах (омніканальність);
3. гнучкість і масштабованість архітектури, що дозволяє ефективно розділяти роботу між командами бекенду та фронтенду;
4. покращення продуктивності та безпеки завдяки скороченню кількості взаємодій між компонентами системи;
5. швидкість оновлення інтерфейсу без втручання в бекенд-частину.

Водночас варто враховувати й низку викликів, які можуть виникнути при впровадженні Headless CMS [10]. Такий підхід передбачає глибше технічне розуміння з боку команди, а також потребує більшої кількості ресурсів на етапі реалізації:

- складність початкового налаштування порівняно з традиційними CMS, які пропонують готові шаблони та візуальні редактори;
- необхідність у кваліфікованих фронтенд-розробниках для створення користувацького інтерфейсу з нуля;
- відсутність “з коробки” інтерфейсів для контент-менеджерів, що може ускладнити процес наповнення контентом без додаткових розробок;
- залежність від зовнішніх API та обмеження у їх роботі або швидкості.

Для кращого розуміння ключових відмінностей між традиційними та безголовими системами керування контентом доцільно провести їх порівняльний аналіз за основними критеріями. У таблиці нижче наведено основні технічні та функціональні характеристики, які відрізняють Headless CMS від класичних рішень (табл. 1.1)

Таблиця 1.1

Відмінності між CMS

Критерій	Традиційна CMS (наприклад, WordPress)	Headless CMS (наприклад, Strapi, Contentful)
Архітектура	Монолітна (зв’язаний бекенд і фронтенд)	Розділена (тільки бекенд, фронтенд окремо)
Контроль над UI/UX	Обмежений шаблонами	Повний контроль розробника
Адаптивність до платформ	Обмежена	Будь-яка платформа (веб, мобільна, IoT)
Легкість впровадження	Вища (готові теми, плагіни)	Складніше, потребує окремої фронтенд-розробки
Гнучкість	Обмежена	Максимальна
Продуктивність	Середня	Вища за рахунок розподілу навантаження
Безпека	Залежить від тем і плагінів	Вища (менше точок доступу для атак)
Візуальний редактор	Так (WYSIWYG)	Не завжди (залежить від CMS)

Таким чином, наведене порівняння дозволяє чітко побачити принципові відмінності між класичними CMS та Headless-рішеннями. Хоча впровадження Headless CMS потребує більшої технічної підготовки та ресурсоємніше на старті, саме цей підхід забезпечує гнучкість, масштабованість, технологічну нейтральність і багатоканальність – характеристики, які дедалі частіше визначають успішність сучасних цифрових продуктів.

1.1.3 Стан розвитку ринку Headless CMS та веб-технологій

Упродовж останніх п’яти років спостерігається стабільне зростання популярності Headless CMS, що зумовлено розвитком frontend-фреймворків (React, Vue, Angular), появою JAMstack-підходу, а також трендом на мікросервісну архітектуру [11].

Серед найпопулярніших рішень на ринку Headless CMS сьогодні – Contentful, Strapi, Sanity, Ghost, Prismic, Directus, а також open-source варіанти на кшталт KeystoneJS. Деякі великі компанії вже повністю перейшли на подібні рішення, зокрема Netflix, Nike, Spotify, Mozilla [12].

Згідно з прогнозами MarketsandMarkets, обсяг світового ринку Headless CMS досягне понад 1.6 млрд доларів США до 2027 року зі середнім темпом зростання понад 20% на рік [13].

Незважаючи на технічні виклики, Headless CMS демонструють впевнене зростання. Глобальний перехід на багатоплатформену взаємодію з користувачем, поява нових форматів споживання контенту (голосові помічники, розумні екрани, AR/VR), а також розвиток JAMstack-технологій сприяють активному впровадженню безголових архітектур [14].

Все більше компаній – від технологічних стартапів до великих корпорацій – переходять на Headless CMS, щоб забезпечити гнучкість, масштабованість та незалежність своєї інфраструктури. Таким чином, Headless CMS розглядається не просто як альтернатива класичним системам, а як перспективна архітектура майбутнього цифрового контенту.

1.2 Методології управління ІТ-проєктами

1.2.1 Роль та значення методологій в управлінні ІТ-проєктами

Управління ІТ-проєктами є складним процесом, що охоплює широкий спектр завдань: від планування та розподілу ресурсів до контролю виконання, управління ризиками, комунікацій та якості. Для ефективної організації всіх цих процесів використовуються методології управління проєктами – системи принципів, підходів, інструментів і практик, які забезпечують структуроване

та послідовне виконання проєктної діяльності [15].

Методологія управління проєктами – це не просто набір інструментів чи шаблонів. Вона визначає логіку організації всієї роботи над проєктом, спосіб взаємодії між учасниками команди, підхід до ухвалення рішень і контролю результатів. Застосування чітко визначеної методології дозволяє уникнути хаосу, підвищити прозорість процесів та забезпечити керованість у складних динамічних умовах [16].

Особливо актуальною наявність ефективної методології є у сфері IT, де проєкти часто реалізуються в умовах високої невизначеності, змінних вимог та обмежених строків. Відсутність структурованого підходу до управління призводить до типових проблем: порушення термінів, перевитрата бюджету, зниження якості продукту, перевантаження команди, нерозуміння між технічною та бізнес-стороною [17].

З іншого боку, використання перевірених методологій дозволяє:

1. чітко визначати цілі, етапи та результати проєкту;
2. ефективно планувати ресурси та строки;
3. підтримувати зворотний зв’язок між усіма учасниками проєкту;
4. швидко адаптуватися до змін;
5. забезпечити прозорість і контроль на кожному етапі реалізації [18].

Таким чином, кожна з описаних методологій має свої сильні сторони, і вибір конкретної залежить від типу продукту, вимог замовника, технічної складності, а також досвіду та структури команди. Далі розглядається порівняння ключових характеристик методологій та обґрунтовується доцільність застосування Scrum для реалізації проєкту з розробки Headless CMS.

1.2.2 Огляд сучасних методологій управління IT-проєктами

У сфері інформаційних технологій, де змінність вимог, швидкий розвиток технологій та висока конкуренція є нормою, вибір методології

управління проєктом відіграє вирішальну роль. Існує широкий спектр підходів до організації роботи над IT-проєктами, які відрізняються глибиною планування, рівнем гнучкості, структурою командної взаємодії та принципами прийняття рішень [19].

Waterfall – це традиційна лінійна модель управління проєктами, яка передбачає послідовне проходження фіксованих етапів:

1. Визначення вимог;
2. Проєктування;
3. Реалізація;
4. Тестування;
5. Впровадження;
6. Супровід.

Кожен етап починається лише після повного завершення попереднього, а повернення до попередньої фази зазвичай є ускладненим. Цей підхід добре підходить для проєктів з чітко визначеними вимогами, які не змінюються в процесі розробки (наприклад, державні або інфраструктурні системи). Проте в динамічних IT-проєктах, особливо пов’язаних із веброзробкою або стартапами, Waterfall втрачає ефективність. Жорстка структура обмежує можливості адаптації до змін, що часто призводить до затримок, перевитрат і невідповідності кінцевого результату очікуванням замовника [20].

Agile – це філософія управління, заснована на гнучкості, постійному зворотному зв’язку, адаптації до змін і швидкому отриманні результату. Основні цінності Agile викладено в Маніфесті гнучкої розробки програмного забезпечення (Agile Manifesto), який передбачає:

- людей та взаємодію важливішими за процеси та інструменти;
- працююче програмне забезпечення – важливіше за вичерпну документацію;
- співпрацю з замовником – важливішу за договірні умови;
- готовність до змін – важливішу за слідування початковому плану [19].

Agile не є однією конкретною методологією, а об'єднує цілу групу підходів, найвідомішим і найпоширенішим з яких є Scrum.

Scrum – це фреймворк управління проєктами в межах Agile, який особливо часто використовується у веброзробці та створенні програмного забезпечення. Scrum орієнтований на розробку продукту через серію коротких ітерацій [21]. (спринтів), що зазвичай тривають від 1 до 4 тижнів. Після кожного спринту команда представляє готовий інкремент продукту, що дає змогу замовнику оцінити прогрес і в разі потреби скорегувати напрямок роботи. Основні поняття Scrum:

- Спринт – фіксований період, протягом якого команда реалізує визначений обсяг роботи;
- Product backlog – список усіх функціональних вимог до продукту;
- Sprint backlog – перелік задач, відібраних із загального беклогу для реалізації в поточному спринті;
- Scrum-мастер – особа, яка відповідає за підтримку процесу Scrum і усунення перешкод для команди;
- Product owner – представник замовника, який формулює бачення продукту й пріоритети;
- Development team – кросфункціональна команда розробників, що виконує технічну реалізацію.

Scrum сприяє високому рівню прозорості, регулярному отриманню зворотного зв’язку, швидкій адаптації до змін і постійному вдосконаленню процесів (через ретроспективи). Це робить його надзвичайно ефективним у складних, невизначених або інноваційних проєктах – зокрема, в розробці Headless CMS, де вимоги можуть змінюватися на ходу, а кінцевий продукт формується поступово.

Kanban – ще один гнучкий підхід, який часто використовується у поєднанні зі Scrum. Він базується на візуалізації робочого процесу (найчастіше – у вигляді дошки з колонками: “To do”, “In progress”, “Done”) і обмеженні кількості одночасних задач у роботі. Kanban добре підходить для команд із

постійним потоком задач або в службах підтримки, де немає чітко окреслених ітерацій [20], [22].

Lean – методологія, що зосереджена на усуненні втрат, оптимізації процесів та підвищенні цінності для замовника. Вона була адаптована з виробництва (Toyota Production System) до IT-середовища.

Extreme Programming (XP) – орієнтована на підвищення якості коду та гнучкість розробки. XP активно використовує практики парного програмування, частого релізу, автоматизованого тестування та безперервної інтеграції. Цей підхід часто інтегрується у Scrum-команди для покращення технічної реалізації [21], [22].

У сучасній практиці управління IT-проєктами використовуються як традиційні каскадні підходи (Waterfall), так і гнучкі методології (Agile, Scrum, Kanban), кожна з яких має свої особливості, переваги й недоліки. Вибір конкретної методології залежить від типу проєкту, складу команди, цілей і вимог замовника. Далі буде представлено порівняльний аналіз сучасних методологій та обґрунтування вибору оптимального підходу до управління проєктом розробки Headless CMS.

1.2.3 Порівняння методологій управління IT-проєктами та обґрунтування вибору

Вибір методології управління є одним із ключових рішень, що визначає структуру, темп і ефективність реалізації IT-проєкту. Нижче наведено порівняльну характеристику найбільш поширених підходів – Waterfall, Scrum, Kanban – за основними критеріями управління: гнучкість, адаптивність, залучення замовника, прозорість процесів, швидкість реакції на зміни, а також релевантність для розробки складних цифрових продуктів, таких як Headless CMS.

Таблиця 1.2

Порівняння сучасних методологій управління IT-проєктами

Критерій	Waterfall	Scrum	Kanban
Структура	Лінійна, послідовна	Ітеративна, спринти	Неперервний потік завдань
Адаптація до змін	Низька	Висока	Висока
Залученість замовника	Мінімальна	Постійна (Product Owner)	Гнучка, залежно від процесу
Прозорість для команди	Низька	Висока (daily meetings, review, ретроспектива)	Середня (візуалізація на дошці)
Контроль за пріоритетами	Обмежений	Високий (беклог, пріоритети)	Високий (через зміни у потоках)
Швидкість отримання результату	Лише наприкінці проєкту	Після кожного спринту	Поступове надання результату
Придатність для CMS-проєктів	Низька (немає гнучкості)	Висока (часті оновлення, зворотний зв'язок)	Середня (ефективна підтримка, але не запуск)
Гнучкість у зміні вимог	Практично відсутня	Постійна адаптація у межах спринтів	Можлива в будь-який момент

З огляду на порівняння, найбільш ефективною методологією для реалізації проєкту розробки Headless CMS є Scrum. Такий вибір зумовлений кількома чинниками:

- Проєкт має динамічну структуру та може змінювати функціональні вимоги в процесі розробки. Scrum дозволяє легко адаптувати пріоритети завдань без втрати керованості.
- Команда проєкту є мультифункціональною, що відповідає підходу Scrum до кросфункціональних ролей та самокерованості.

- Продукт створюється поетапно, і кожен інкремент можна демонструвати замовнику, збираючи зворотний зв’язок та уточнюючи подальші вимоги.
- Наявність беклогу дає змогу ефективно планувати та оцінювати обсяг робіт у межах спринтів.
- Гнучка архітектура Headless CMS потребує регулярної перевірки інтеграцій, модулів і API, що органічно вписується в короткі ітерації Scrum.

Таким чином, впровадження Scrum забезпечує високу адаптивність, передбачуваність процесів, активну участь замовника і прозорість у досягненні проміжних результатів. Це дозволяє знизити ризики проєкту та підвищити якість реалізованого продукту.

1.3 Маркетингове обґрунтування проєкту

1.3.1 Аналіз конкурентного середовища у сфері CMS

Системи управління контентом (CMS) є основою сучасної цифрової інфраструктури для вебсайтів, онлайн-сервісів, мобільних додатків та інших цифрових платформ. Ринок CMS активно розвивається і включає як традиційні монолітні рішення (наприклад, WordPress, Joomla, Drupal), так і новіші Headless-рішення (Strapi, Contentful, Sanity та інші), орієнтовані на API-підхід і омніканальність.

Станом на 2025 рік WordPress утримує 61.3% світового ринку CMS, залишаючись найбільш популярним вибором для створення вебсайтів завдяки своїй простоті, широкому вибору шаблонів та плагінів, а також активній спільноті користувачів і розробників [24]. Проте у сегменті складних, високонавантажених і масштабованих продуктів все чіткіше простежується зміщення фокусу до Headless-платформ. Це зумовлено низкою факторів: стрімким зростанням кількості точок взаємодії з користувачем (веб, мобільні застосунки, голосові інтерфейси, IoT-пристрої), потребою в централізованому керуванні контентом для кількох платформ одночасно, підвищеними

вимогами до продуктивності та безпеки, а також бажанням мати гнучку архітектуру, яка дозволяє незалежно оновлювати фронтенд і бекенд.

Згідно з прогнозами, обсяг світового ринку Headless CMS досягне понад 22.28 млрд доларів США до 2034 року зі середнім темпом зростання понад 21.42% на рік [24]. Серед найпопулярніших рішень на ринку Headless CMS сьогодні – Contentful, Strapi, Sanity, Ghost, Prismic, Directus, а також open-source варіанти на кшталт KeystoneJS. Деякі великі компанії вже повністю перейшли на подібні рішення, зокрема Netflix, Nike, Spotify, Mozilla [25].

Для кращого розуміння конкурентного середовища в таблиці 1.3 наведено порівняльний огляд популярних CMS за ключовими параметрами.

Таблиця 1.3

Порівняння популярних CMS

CMS	Тип	Переваги	Недоліки
1	**2**	**3**	**4**
WordPress	Традиційна	Найбільша спільнота, тисячі плагінів, простота встановлення	Обмежена масштабованість, вразливість до атак, складнощі при кастомізації
Joomla	Традиційна	Гнучкі налаштування, підтримка багатомовності	Менше підтримки з боку спільноти, складніше в освоєнні ніж WP
Drupal	Традиційна	Висока безпека, масштабованість, підходить для урядових сайтів	Високий поріг входу, потреба в досвідчених розробниках
Magento	eCommerce	Потужний функціонал для онлайн-магазинів	Вимогливий до ресурсів, складна підтримка

Продовження Таблиці 1.3

1	2	3	4
Strapi	Headless (Open Source)	Гнучка структура, просте API, розширюваність, працює на Node.js	Потребує розробників, менш зручний для нетехнічних користувачів
Sanity	Headless (SaaS)	Реальний час редагування, гнучкість структур даних, швидкий API	Менша популярність, потребує знання інтерфейсу Studio
Directus	Headless (Open Source)	Візуальна адмін-панель, API-first, SQL-бази підтримуються нативно	Менш розвинена спільнота, не для високонавантажених проєктів
KeystoneJS	Headless (Open Source)	Гнучкий код, підтримка GraphQL, сучасна архітектура	Менш зрілий продукт, потребує глибокої інтеграції вручну

Аналіз показує, що ринок CMS поділений на два основних сегменти: масовий сегмент із домінуванням традиційних рішень (WordPress, Joomla) та просунутий технічний сегмент, орієнтований на гнучкі, API-орієнтовані системи. Саме останній сегмент активно зростає, і саме в ньому Headless CMS демонструють найбільший потенціал.

Завдяки своїй архітектурі та незалежності від конкретного каналу доставки контенту, Headless-рішення на кшталт Strapi, Contentful, Sanity стають привабливими для компаній, що прагнуть масштабованості, омніканальності та високої адаптивності інтерфейсу [26]. Водночас їхній недолік – складність реалізації – відкриває простір для нових гравців, які можуть запропонувати кастомізоване рішення з покращеною UX для контент-менеджерів і розробників.

Саме на цей сегмент і орієнтується розробка нашої Headless CMS, яка має на меті поєднати гнучкість API-архітектури з інтуїтивною панеллю

керування та можливістю глибокого налаштування.

1.3.2 STEP-аналіз зовнішнього середовища

Для комплексної оцінки зовнішніх факторів, які можуть вплинути на реалізацію проєкту розробки Headless CMS, доцільно застосувати STEP-аналіз. Цей інструмент дозволяє врахувати політичні, економічні, соціально-культурні та технологічні зміни, що формують зовнішнє середовище функціонування проєкту [27].

У таблиці 1.4 наведено перелік основних факторів за кожною категорією, які були враховані при стратегічному аналізі.

Таблиця 1.4

STEP-фактори впливу

Політичні	Економічні	Соціальні	Технологічні
Підтримка цифрової трансформації та IT-галузі	Попит на економічно ефективні IT-рішення	Попит на швидкий доступ до контенту з різних платформ	Широке впровадження JAMstack і Headless-архітектури
Євроінтеграційний курс і вимоги до захисту даних (GDPR)	Зростання IT-ринку та частка SaaS-моделей	Зростаюча кількість digital-команд і контент-редакторів	Стрімкий розвиток frontend-фреймворків
Вплив воєнного стану та загроза кібератак	Потреба в оптимізації витрат на розробку	Очікування персоналізованого досвіду користувача	Поширення cloud-native рішень та API-first підходу
Можливі державні стимули для локального програмного забезпечення	Конкуренція з великими SaaS-рішеннями (Contentful, Sanity тощо)	Низький рівень технічної грамотності частини аудиторії	Підвищені вимоги до кібербезпеки

Зокрема, у політичному середовищі відзначається загальна підтримка цифровізації та IT-індустрії з боку держави, що створює сприятливі умови для запуску нового вітчизняного програмного забезпечення. Економічна ситуація

стимулює попит на доступні й адаптивні рішення з мінімальними витратами на підтримку. У соціальному контексті зростає кількість користувачів, які працюють із цифровим контентом, а також підвищуються очікування щодо швидкості, зручності та персоналізації. Технологічні фактори демонструють розвиток JAMstack-підходу, API-орієнтованих архітектур та сучасних фреймворків, що створює сприятливий ландшафт для реалізації Headless CMS.

Для конкретизації впливу кожного чинника на проєктну діяльність було здійснено деталізований аналіз змін у галузі, їх впливу на проєкт та відповідних управлінських дій. Результати подано у таблиці 1.5.

Таблиця 1.5

Підсумки STEP-Аналізу

Фактори	**Зміни в галузі**	**Вплив на проєкт**	**Дії в межах проєкту**
1	**2**	**3**	**4**
Політичні	Підтримка IT-продуктів українського походження; підвищення вимог до захисту персональних даних	Підвищення привабливості локальної Headless CMS; необхідність відповідності GDPR	Розробка українського продукту з підтримкою європейських стандартів
Економічні	Попит на дешевші, але гнучкі рішення; домінування підписних моделей	Необхідність конкурентної вартості; оптимізація бізнес-моделі	Реалізація гнучкої підписної моделі та мінімізація витрат на запуск MVP
Соціальні	Зростання кількості цифрових користувачів; підвищення очікувань щодо UX	Фокус на простоту використання та адаптацію для різних типів користувачів	Інтуїтивний UI, onboarding, документація для нетехнічних користувачів

Продовження таблиці 1.5

1	2	3	4
Технологічні	Розвиток JAMstack/ API-first, багатоканальностіта посилення ролі безпеки	Необхідність підтримки сучасних технологій, безпечних API, модульної архітектури	Інтеграція API-first, підтримка JAMstack, налаштування політик безпеки

Загальний результат STEP-аналізу свідчить про високий рівень зовнішньої готовності до запуску проєкту Headless CMS. Політична ситуація сприяє створенню локального ПЗ, а технологічний розвиток відкриває широкі можливості для впровадження сучасної архітектури. Водночас проєкт потребує стратегічного врахування економічних реалій (цінова конкуренція, підписні моделі) та підвищеної уваги до UX і безпеки – як ключових очікувань із боку ринку.

1.3.3 Аналіз цільової аудиторії та зацікавлених сторін

Успішна реалізація будь-якого ІТ-проєкту потребує глибокого розуміння потреб, очікувань і специфіки як кінцевих користувачів, так і всіх учасників, що прямо або опосередковано впливають на життєвий цикл продукту [28]. Детальне визначення цільової аудиторії та зацікавлених сторін дозволяє побудувати продукт, максимально адаптований до ринкових запитів, уникнути помилок у функціональності та сформувати ефективну комунікацію протягом розробки.

Визначення цільової аудиторії – ключовий етап у проєктному плануванні, оскільки саме ці користувачі взаємодіятимуть із системою на щоденній основі. У випадку з Headless CMS спектр потенційних користувачів охоплює як технічні команди, так і нетехнічних фахівців, що працюють із контентом [29]. Продукт має відповідати вимогам обох груп і забезпечувати баланс між гнучкістю та зручністю використання. До основних сегментів

цільової аудиторії належать:

- Frontend-розробники – зацікавлені в максимально гнучкому API, який легко інтегрується з сучасними фреймворками (React, Vue, Angular).
- Back-end розробники / DevOps – цінують архітектурну масштабованість, безпеку, REST/GraphQL API, розгортання на хмарних платформах.
- Контент-менеджери / редактори – очікують просту, інтуїтивно зрозумілу адмін-панель, що дозволяє ефективно керувати контентом без участі розробників.
- Агенції / SaaS-компанії – шукають можливість повторного використання контенту на багатьох платформах, масштабованість і кастомізацію.
- Продуктові команди – фокусуються на швидкому виведенні змін без зупинки системи, A/B тестуванні, інтеграції з CRM, аналітикою.

Окрім основних користувачів, важливо враховувати інтереси інших сторін, які впливають на проєкт або мають очікування щодо його результатів. Їхній вплив може стосуватися фінансування, прийняття рішень, просування продукту або забезпечення його життєздатності після запуску. Ключові зацікавлені сторони проєкту:

- Ініціатор проєкту / замовник – забезпечує фінансування, формулює стратегічні цілі продукту.
- Команда розробки – безпосередньо впроваджує технічні рішення, впливає на вибір архітектури, стеку технологій та підходів.
- Партнери (наприклад, веб-агенції) – можуть бути як користувачами, так і каналами поширення продукту.
- Кінцеві клієнти партнерів (B2B2C) – споживають контент, сформований через CMS, тому їхній досвід опосередковано впливає на вимоги до продукту.

- Маркетингова команда – формує позиціонування продукту, впливає на функціонал, пов’язаний із SEO, метаданими, багатомовністю.

Чітке визначення й продумана сегментація цільової аудиторії дозволяють адаптувати продукт під реальні потреби ринку, мінімізувати технічні бар’єри для впровадження, а також створити зрозуміле бачення функціональності, необхідної для кожної категорії користувачів. Наявність різнорівневих зацікавлених сторін вимагає гнучкої архітектури, масштабованого API, а також дружнього інтерфейсу для нетехнічних користувачів. Це закладає основу для ефективної реалізації та успішного впровадження Headless CMS на конкурентному ринку.

1.3.4 Оцінка проєктних альтернатив

1.3.4.1 Визначення можливих підходів до реалізації проєкту

На етапі початкового проєктного аналізу важливо розглянути кілька можливих підходів до реалізації програмного продукту та оцінити їхню доцільність з точки зору функціональності, гнучкості, вартості, термінів реалізації та складності підтримки. Такий підхід дозволяє не лише обґрунтувати обраний шлях, а й зменшити ризики, пов’язані з одностороннім технічним рішенням [30].

У межах проєкту розробки Headless CMS було сформовано п’ять потенційних варіантів реалізації:

1. Використання традиційної CMS із надбудовою API. Створення REST або GraphQL-шарів поверх платформи типу WordPress, Joomla або Drupal для імітації Headless-архітектури [31].

2. Інтеграція готової Headless CMS (SaaS або open-source). Підключення існуючої системи (наприклад, Contentful, Sanity, Strapi) через API для керування контентом і взаємодії з frontend-застосунками [32].

3. Повна розробка кастомної Headless CMS. Рішення створити власну CMS з нуля – для повного контролю над логікою, структурою контенту,

авторизацією, API, UI-панеллю тощо.

4. Використання існуючого open-source рішення з глибоким доопрацюванням. Базування на платформі типу Strapi, KeystoneJS або Directus з подальшим розширенням функціональності, налаштуванням панелі, авторизації, ролей [33].

5. Комбіноване рішення: власний backend + сторонній редактор. Побудова бекенду (наприклад, на Laravel або Symfony), використання візуального інтерфейсу типу Builder.io або Netlify CMS для керування контентом.

Для зручності оцінювання кожної альтернативи було складено порівняльну таблицю 1.6 з ключовими перевагами та недоліками кожного підходу.

Таблиця 1.6

Порівняльний аналіз альтернатив реалізації Headless CMS

Альтернатива	**Переваги**	**Недоліки**
Традиційна CMS із надбудовою API	Швидкий старт, знайома екосистема	Архітектурні обмеження, низька масштабованість
Інтеграція готової Headless CMS	Висока швидкість запуску, стабільне ядро, документація	Залежність від постачальника, ліцензійні витрати
Повна кастомна розробка	Повна гнучкість, адаптація під власну архітектуру	Дорого, довго, потребує великої команди
Open-source CMS + глибоке кастомне доопрацювання	Баланс між готовим рішенням і гнучкістю, контроль над кодом	Вимагає сильної технічної експертизи, можлива нестабільність
Власний backend + сторонній редактор (наприклад Builder.io)	Висока кастомізація, гнучка UI/UX, можливість масштабування	Складна інтеграція, різнорідність інструментів, підтримка декількох систем

1.3.4.2 SWOT-аналіз альтернатив

Для прийняття зваженого рішення щодо вибору оптимального підходу до реалізації Headless CMS було проведено стратегічну оцінку п’яти

альтернатив на основі SWOT-аналізу. Цей метод дозволяє виявити сильні та слабкі сторони кожного варіанту, а також зовнішні можливості й загрози, які можуть впливати на успішність проєкту.

У таблиці 1.7 наведено SWOT-аналіз п’яти варіантів реалізації системи управління контентом:

Таблиця 1.7

SWOT-аналіз проєктних альтернатив

1. Традиційна CMS з API	
Сильні сторони (Strengths): - відомі платформи - швидкий старт	Слабкі сторони (Weaknesses): - архітектурні обмеження - складнощі кастомізації
Можливості (Opportunities): - використання наявних плагінів і шаблонів	Загрози (Threats) - застаріла структура - конфлікти плагінів
2. Готова Headless CMS	
Сильні сторони (Strengths): - швидке впровадження - підтримка - документація	Слабкі сторони (Weaknesses): - обмежена гнучкість - абонплата
Можливості (Opportunities): - швидкий MVP - масштабування через хмару	Загрози (Threats) - залежність від провайдера - ліцензії
3. Повна кастомна розробка	
Сильні сторони (Strengths): - максимальна адаптація - гнучкість - контроль	Слабкі сторони (Weaknesses): - висока вартість - тривала розробка
Можливості (Opportunities): - створення унікального рішення	Загрози (Threats) - ризики перевищення бюджету та строків

Продовження таблиці 1.7

4. Open-source CMS + кастомізація	
Сильні сторони (Strengths): - відкритий код - баланс гнучкості й вартості	Слабкі сторони (Weaknesses): - потреба в досвідченій команді
Можливості (Opportunities): - швидке масштабування - контроль над розробкою	Загрози (Threats) - втрата підтримки проєкту або залежності
5. Власний backend + сторонній редактор	
Сильні сторони (Strengths): - гнучкість UI/UX - модульність	Слабкі сторони (Weaknesses): - складна підтримка гібридної архітектури
Можливості (Opportunities): - побудова конкурентної системи	Загрози (Threats) - інтеграційні ризики та технічна роз’єднаність

Для формалізованого порівняння було обрано 4 ключові критерії, за якими експерти оцінили кожен із варіантів:

- Час на розробку
- Вартість проєкту
- Актуальність системи
- Зручність використання

Оцінювання здійснювалося за трирівневою шкалою:

- 1 – незадовільне
- 2 – сумнівно
- 3 – задовільне

Ваги експертів зазначені у таблиці 1.8 та були визначені відповідно до їх досвіду в проєктному управлінні.

Таблиця 1.8

Ваговий коефіцієнт експертів

Експерт	**Ваговий коефіцієнт**
1	**2**
Експерт1	1.00

Продовження таблиці 1.8

1	2
Експерт2	0.75
Експерт3	0.50
Експерт4	0.25

Ваги експертів зазначені у таблиці 1.8 та були визначені відповідно до їх досвіду в проєктному управлінні:

Таблиця 1.9

Оцінка альтернатив

1. Традиційна CMS з API				
Експерт	**Експерт1**	**Експерт2**	**Експерт3**	**Експерт4**
Час на розробку	2	3	2	3
Вартість проєкту	2	2	2	3
Актуальність системи	1	1	1	1
Зручність використання	1	1	1	2
Оцінка експертів	6	7	6	9
Оцінка з коефіцієнтом	6	5,25	3	2,25
Підсумок	16,5			
2. Готова Headless CMS				
Експерт	**Експерт1**	**Експерт2**	**Експерт3**	**Експерт4**
Час на розробку	2	2	2	1
Вартість проєкту	2	2	2	1
Актуальність системи	2	2	2	2
Зручність використання	2	2	2	2
Оцінка експертів	8	8	8	4
Оцінка з коефіцієнтом	8	6	4	1
Підсумок	19			

Продовження таблиці 1.9

3. Повна кастомна розробка				
Експерт	**Експерт1**	**Експерт2**	**Експерт3**	**Експерт4**
Час на розробку	1	1	1	2
Вартість проєкту	1	1	1	1
Актуальність системи	3	2	2	2
Зручність використання	3	2	3	3
Оцінка експертів	8	6	6	9
Оцінка з коефіцієнтом	8	4,5	3,5	2
Підсумок	18			
4. Open-source CMS + кастомізація				
Експерт	**Експерт1**	**Експерт2**	**Експерт3**	**Експерт4**
Час на розробку	2	2	2	1
Вартість проєкту	2	2	2	1
Актуальність системи	3	3	3	3
Зручність використання	3	3	3	2
Оцінка експертів	10	10	10	4
Оцінка з коефіцієнтом	10	7,5	5	1,75
Підсумок	24,25			
5. Власний backend + сторонній редактор				
Експерт	**Експерт1**	**Експерт2**	**Експерт3**	**Експерт4**
Час на розробку	2	2	2	1
Вартість проєкту	2	2	2	1
Актуальність системи	3	3	2	2
Зручність використання	2	2	2	2
Оцінка експертів	9	10	8	4
Оцінка з коефіцієнтом	9	6,75	4	1,5
Підсумок	21,25			

На основі проведеного SWOT-аналізу та багатокритеріальної оцінки проєктних альтернатив можна зробити висновок, що альтернатива «Open-

source CMS з глибокою кастомізацією» отримала найвищу оцінку 24,25 та є найбільш збалансованою. Вона забезпечує гнучкість, контроль над функціональністю, можливість масштабування та має високу релевантність сучасним ринковим вимогам. При цьому вона не потребує такого обсягу інвестицій і часу, як повна кастомна розробка, але значно перевершує класичні CMS у плані архітектурної гнучкості. Саме тому ця альтернатива була обрана як основа для реалізації проєкту Headless CMS у вигляді веб-інтерфейсу.

1.3.5 Аналіз внутрішнього середовища проєкту

Внутрішнє середовище проєкту охоплює усі наявні ресурси, організаційні й технічні можливості, кадровий потенціал, а також ступінь підтримки та залучення з боку зацікавлених сторін. Комплексний аналіз цих чинників дозволяє сформувати реалістичне уявлення про потенціал проєкту, ідентифікувати внутрішні загрози та сформувати умови для ефективного управління [34].

Особливу увагу слід приділити кадровому складу. Команда, сформована для реалізації проєкту, включає фахівців, які мають досвід у PHP/Laravel, JavaScript, React, PostgreSQL, Git, Docker, DevOps-практиках та хмарних середовищах. Такий підхід узгоджується з сучасними рекомендаціями щодо формування мультидисциплінарних DevOps-команд у сфері розробки [35], [36]. У структурі проєкту передбачено такі ключові ролі:

- проєктний менеджер, відповідальний за загальне планування, координацію та взаємодію із зацікавленими сторонами;
- бекенд-розробник, який реалізує API, логіку взаємодії з базою даних і відповідає за архітектуру ядра;
- фронтенд-розробник, що інтегрує API із веб-інтерфейсом системи на базі JavaScript-фреймворків;
- UI/UX дизайнер, який забезпечує логічну, зручну та адаптивну візуальну частину адміністративної панелі;
- DevOps-інженер, відповідальний за CI/CD, конфігурацію

серверного середовища та моніторинг [37];

- контент-менеджер (тестовий користувач), який бере участь у валідації юзабіліті системи.

Такий склад дозволяє покрити ключові напрями розробки та впровадження Headless CMS без потреби в залученні зовнішніх підрядників [38].

Усі члени команди мають належне середовище для ефективної роботи: використовується локальна розробка у Docker-контейнерах, система контролю версій Git, редактори коду VS Code, а також CI/CD процеси на базі GitHub Actions. Для хостингу планується використання VPS-серверів із попередньо налаштованим стеком (Linux, Nginx, PostgreSQL), що забезпечує повну технічну незалежність і контроль над інфраструктурою.

З організаційної точки зору проєкт підтримується керівництвом, що підтверджується наявністю затвердженого бюджету на етап реалізації MVP. Це дозволяє команді зосередитися на функціональності ядра системи, базових сценаріях керування контентом і побудові API.

Ще одним важливим фактором є мотиваційна складова – частина учасників команди має особисту зацікавленість у створенні унікального open-source-рішення, яке може бути використане в інших проєктах компанії або представлене у професійному середовищі. Це формує додаткову відповідальність і сприяє більш високій якості реалізації. Крім того, структура обраної архітектури дозволяє реалізовувати проєкт поступово, у форматі гнучкого MVP з можливістю раннього тестування функціональних модулів.

Узагальнюючи вищезазначене, можна зробити висновок, що внутрішнє середовище проєкту є сприятливим для реалізації Headless CMS. Технічна база, професійна команда та підтримка з боку керівництва створюють надійний фундамент для досягнення цілей проєкту у визначені терміни з оптимальним використанням ресурсів.

1.4 Побудова логіко-структурної моделі проєкту

1.4.1 Побудова дерева проблем

Одним із ключових інструментів системного аналізу проєкту є побудова дерева проблем, що дозволяє виявити першопричини, окреслити центральну проблему та простежити її наслідки на стратегічному, операційному та технічному рівнях. Такий підхід допомагає структуровано осмислити проблемне поле, сформувати основу для постановки цілей та визначення шляхів їх досягнення [39].

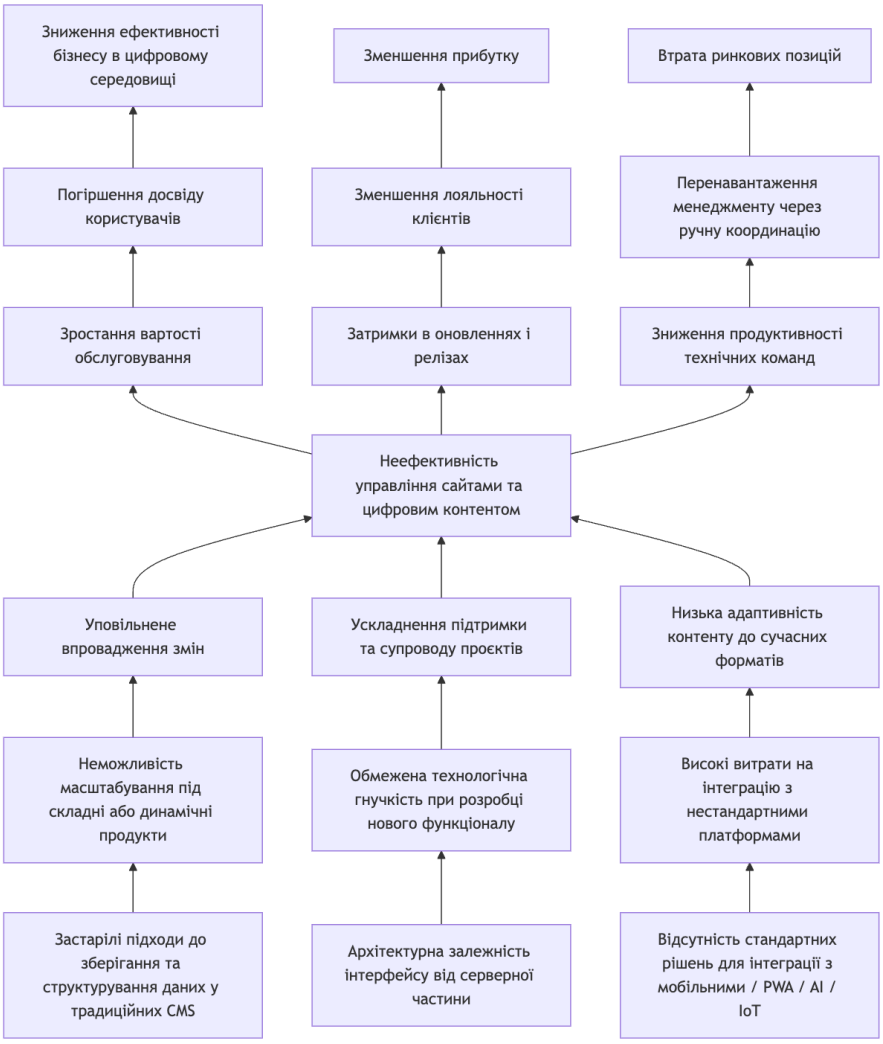


Рис. 1.1. Дерево проблем

У межах дослідження визначено, що головною проблемою є неефективність управління сайтами та цифровим контентом. Вона виникає внаслідок низки технічних та організаційних чинників, зокрема використання

застарілих монолітних CMS, що не здатні забезпечити достатній рівень гнучкості, масштабованості, а також інтеграції з сучасними технологіями розповсюдження контенту.

Серед першопричин можна виокремити обмеження в адаптації функціоналу під потреби бізнесу, високу залежність від жорстко пов’язаних інтерфейсів, а також складнощі з багатоканальним доступом до контенту. Ці фактори призводять до затримок у впровадженні змін, ускладнень у підтримці сайтів та зниження ефективності технічних команд.

Центральна проблема зумовлює низку негативних наслідків: від підвищених витрат на обслуговування та зниження продуктивності, до втрати ринкових позицій та загального зниження ефективності бізнесу.

Структурну візуалізацію цих зв’язків наведено на рис. 1.1 – Дерево проблем, де представлено повну логіку переходу від першопричин до фінальних стратегічних наслідків.

Таким чином, побудова дерева проблем дозволила систематизувати ключові виклики, що стоять перед проєктом, і визначити причинно-наслідкові зв’язки між ними. Встановлення центральної проблеми – неефективності управління сайтами – створює основу для формування цілей проєкту, які будуть спрямовані на усунення першопричин та мінімізацію негативних наслідків. Наступним кроком логічного аналізу є побудова дерева цілей, що дозволить деталізувати шляхи досягнення очікуваних результатів.

1.4.2 Побудова дерева цілей

На основі проведеного аналізу проблемного поля доцільно сформувати дерево цілей, що дозволяє перетворити ідентифіковані проблеми на бажані результати. Такий підхід є одним з найефективніших методів стратегічного планування, оскільки дозволяє чітко простежити логіку досягнення генеральної мети через досягнення взаємопов’язаних проміжних і конкретних цілей [40], [41].

У центрі дерева розташована генеральна ціль проєкту – підвищення

ефективності управління сайтами та цифровим контентом. Вона прямо пов’язана з центральною проблемою, виявленою під час побудови дерева проблем. Кожна першопричина була трансформована в одну або декілька цілей нижчого рівня, що дає змогу цілеспрямовано усунути негативні чинники, які призводили до неефективності [42].

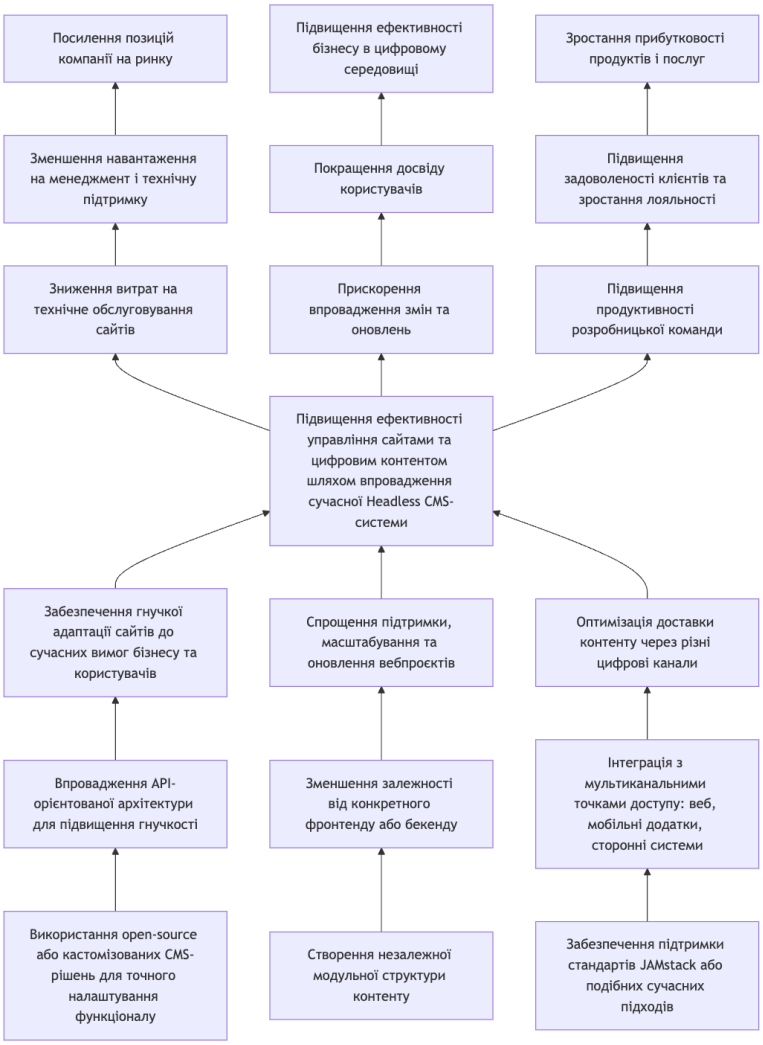


Рис. 1.2. Дерево цілей

Цілі формуються за принципом зростаючої конкретизації – від стратегічного бачення до технічних або організаційних заходів. Наприклад, такі глибинні проблеми, як застарілі підходи до структурування контенту чи обмежена інтеграція з цифровими каналами, трансформуються у цілі, спрямовані на впровадження сучасних open-source рішень, побудову модульної архітектури та використання API для мультиканальної доставки.

Досягнення цілей нижчого рівня забезпечить реалізацію проміжних результатів – підвищення гнучкості, скорочення часу оновлення системи, покращення адаптації сайтів до вимог користувачів і бізнесу. В результаті це сприятиме досягненню генеральної мети проєкту.

Водночас, як і у дереві проблем, очікувані результати мають свою ієрархію. На першому рівні – це безпосередні ефекти впровадження, зокрема зниження витрат, підвищення продуктивності команд та прискорення релізів. Другий рівень охоплює покращення користувацького досвіду й оптимізацію внутрішніх процесів, а третій – стратегічні переваги, включно з підвищенням прибутковості й посиленням позицій компанії на ринку [43].

Структурну візуалізацію цілей і очікуваних результатів наведено на рис. 1.2 – Дерево цілей, що демонструє логіку переходу від проміжних цілей до досягнення стратегічного результату.

1.4.3 Побудова логіко-структурної схеми проєкту

Після формалізації дерева проблем та дерева цілей наступним кроком є побудова логіко-структурної схеми (ЛСС), яка дозволяє структурувати цілі проєкту, окреслити очікувані результати, ресурси, показники досягнення, а також визначити ключові ризики реалізації [44].

Логіко-структурна схема (ЛСС) є важливим інструментом управління проєктами, яка допомагає зв'язати стратегічну мету з конкретними заходами, результатами та припущеннями. Такий підхід забезпечує прозорість планування, полегшує моніторинг прогресу та прийняття управлінських рішень на всіх етапах реалізації [45].

У межах проєкту розробки сучасної Headless CMS загальною метою визначено підвищення ефективності управління сайтами та цифровим контентом для компаній середнього та великого бізнесу.

Кожна з цілей пов’язана з відповідними результатами, засобами реалізації та показниками досягнення. Це дозволяє побудувати чітку логіку впровадження проєкту та забезпечити ефективний моніторинг його прогресу.

ЛСС також враховує низку припущень і ризиків, які можуть вплинути на успішність проєкту. Зокрема, передбачається наявність кваліфікованої команди, технічних ресурсів та підтримки з боку замовника. Серед потенційних ризиків – затримки в розробці, складність інтеграції з іншими системами та можливі зміни вимог у процесі реалізації.

Логіко-структурна схема представлена у таблиці 1.9 та включає загальну мету, конкретизовані цілі, очікувані результати, показники досягнення, засоби реалізації, припущення та ризики. Такий формат дозволяє узгодити стратегічні наміри із практичними етапами впровадження проєкту.

Таблиця 1.9

Логіко-Структурна Схема

Елемент	**Зміст**
1	**2**
Загальна ціль	Підвищення ефективності управління сайтами та цифровим контентом для малого і середнього бізнесу.
Конкретні цілі	1. Забезпечення гнучкості у розробці та масштабуванні сайтів. 2. Оптимізація процесів оновлення контенту. 3. Створення умов для багатоканальної доставки інформації.
Показники досягнення	1. Зменшення часу на оновлення сайту. 2. Зменшення кількості технічних проблем. 3. Підвищення задоволеності користувачів.
Вимірювачі	1. Середній час внесення змін у контент (до 1 години). 2. Кількість технічних інцидентів на місяць (менше 2). 3. Відгуки користувачів (середня оцінка більше 4 з 5).
Припущення та ризики	Можливе недотримання термінів, складність інтеграції, недостатній рівень технічної кваліфікації команди.
Результати	1. Сайти можна масштабувати без змін у CMS. 2. Контент легко адаптується під нові канали. 3. Менше технічних помилок.

Продовження таблиці 1.9

Логіко-Структурна Схема

1	2
Дії	1. Провести аудит поточної архітектури. 2. Розробити Headless CMS-рішення. 3. Підключити frontend через API. 4. Провести тестування та запуск.
Засоби	1. Команда з 1 PM, 1 аналітика, 2 backend та 2 frontend розробників, 1 тестувальник. 2. Сервери, репозиторії, API-документація.
Витрати	Оренда хостингу, оплата праці команди, вартість технічної підтримки, витрати на тестування.
Передумови	Потреба в масштабованій і гнучкій системі для управління сайтами, бажання зменшити технічне навантаження та підвищити якість продукту.

1.5 Інвестиційні дослідження

Інвестування в розробку Headless CMS є обґрунтованим рішенням у відповідь на зростаючий попит на гнучкі та масштабовані системи керування контентом. На відміну від традиційних монолітних CMS, Headless-архітектура дозволяє розділяти фронтенд і бекенд, забезпечуючи вищу швидкість, безпеку та технологічну свободу для розробників. Згідно з дослідженням Forrester, саме ці переваги стали ключовими драйверами зростання сегменту Headless CMS у сфері цифрової трансформації компаній [46].

Комерційні Headless-рішення часто є дорогими або складними для глибокої кастомізації, що створює попит на доступні open-source альтернативи. Запропонований проєкт орієнтований на створення такого продукту – з сучасною технологічною базою, відкритим кодом, зручним веб-інтерфейсом та можливістю впровадження SaaS-моделі. Цей підхід відповідає загальним ринковим трендам – наприклад, у 2022 році компанія Kontent.ai, що спеціалізується на Headless CMS, залучила $40 мільйонів інвестицій, що підтверджує високий інтерес до подібних рішень з боку венчурних фондів [47].

Таким чином, інвестиції в проєкт дозволяють створити конкурентоспроможний цифровий продукт із високим потенціалом монетизації та масштабування в середньо- та довгостроковій перспективі.

Загальна тривалість реалізації проєкту становить 12 місяців, що є обґрунтованим строком для створення MVP-версії продукту, її технічного вдосконалення та початку комерційного використання. Проєкт реалізується у дві фази, кожна з яких має чітке функціональне спрямування.

Перші сім місяців присвячено розробці програмного забезпечення. На цьому етапі здійснюється проєктування архітектури системи, вибір відповідного технологічного стеку, створення моделі бази даних, реалізація ядра CMS та ключових функціональних модулів. Паралельно розробляється зручний веб-інтерфейс для керування контентом, інтегруються API, реалізуються системи кешування, логування та інші компоненти сучасної мікросервісної архітектури. Впроваджуються інструменти безперервної інтеграції та доставки, проводиться початкове тестування. Завершальним етапом фази є публікація MVP-версії, яку можна використовувати для внутрішнього тестування або залучення перших бета-користувачів.

Починаючи з восьмого місяця, проєкт переходить до етапу маркетингу, підтримки та комерціалізації. У цей період усуваються критичні помилки, що були виявлені під час тестування, оптимізується функціональність відповідно до зворотного зв’язку користувачів. Проводиться налаштування платіжної інфраструктури для запуску підписної моделі, створюється базова документація для клієнтів, організовується технічна підтримка. Паралельно розгортаються маркетингові активності: запуск таргетованої реклами, публікації в соціальних мережах, формування партнерських програм. Основна мета цього етапу – залучення перших користувачів, тестування бізнес-моделі та поступовий вихід на точку беззбитковості.

Для реалізації проєкту Headless CMS у межах 12-місячного періоду передбачається низка операційних та одноразових витрат. Бюджет охоплює як технічну реалізацію, так і бізнесову складову (маркетинг, офісне утримання,

ліцензії тощо). Розрахунок базується на реалістичних середньоринкових ставках, характерних для проєктів у сфері IT-розробки в Україні у 2025 році. З бюджет проєкту можна ознайомитися у таблиці 1.10.

Таблиця 1.10

Бюджет проєкту

Стаття витрат	**Щомісячна сума (грн)**	**Кількість місяців**	**Загалом (грн)**
Заробітна плата (команда)	985 000	12	1 182 0000
Оренда офісу	600 00	12	720 000
Обладнання та техніка	585 000 (одноразово)	1	585 000
Хостинг і сервери	12 000	12	144 000
Ліцензії, ПЗ, домени	6 500	12	78 000
Маркетинг і реклама	200 000	5	1 000 000
Інші витрати	8 000	12	96 000
РАЗОМ			**14 443 000**

Основною статтею витрат є фонд оплати праці – він включає зарплати ключових членів команди (розробники, дизайнер, DevOps, проєктний менеджер тощо). Вартість обладнання передбачає закупівлю ноутбуків, моніторів, периферії та частини серверної інфраструктури на старті проєкту. Усі витрати є необхідними для повноцінного запуску технічно складного цифрового продукту.

Значна частина бюджету також закладається на маркетинг, який планується активізувати на другому етапі реалізації проєкту (місяці 8–12). Це дозволить забезпечити видимість продукту, протестувати гіпотези щодо позиціонування та залучити перших клієнтів.

Фінансова модель проєкту базується на підписній системі, яка передбачає щомісячну оплату доступу до Headless CMS відповідно до обсягу

функціональності та кількості проєктів, якими може керувати користувач. Такий підхід є стандартом на ринку SaaS-продуктів у сфері керування контентом і дозволяє забезпечити стабільний дохід у довгостроковій перспективі.

Запропоновано три тарифні плани:

- *Basic* – доступ до одного проєкту, орієнтований на фрилансерів або малі бізнеси. Вартість: $190/міс (≈7 980 грн).
- *Pro* – до 10 проєктів, розширений функціонал і технічна підтримка. Орієнтований на студії та агентства. Вартість: $1 500/міс (≈63 000 грн).
- *Enterprise* – необмежена кількість проєктів, пріоритетна підтримка, SLA, кастомні інтеграції. Орієнтований на середній і великий бізнес. Вартість: $8 000/міс (≈336 000 грн).

Від восьмого місяця реалізації проєкту, після публікації MVP та підготовки SaaS-інфраструктури, розпочинається продаж підписок. Стратегія зростання передбачає поступове нарощування клієнтської бази без різких стрибків – через системну роботу з маркетингом, репутацією та зворотним зв’язком.

До 12-го місяця проєкт планує досягти щомісячного доходу 1 626 600 грн, що дозволяє не лише вийти на точку беззбитковості (операційні витрати становлять приблизно 1 300 000 грн/міс), а й розпочати часткове відшкодування раніше здійснених інвестицій. З прогнозом зростання доходів за підписною моделлю можна ознайомитися у таблиці 1.11.

Таблиця 1.11

Прогноз доходів у проєкті

Місяць	**Basic**	**Pro**	**Enterprise**	**Загальний дохід, грн**
1	**2**	**3**	**4**	**5**
1–7	0	0	0	0

Продовження Таблиці 1.11

1	2	3	4	5
8	3	1	0	87 940
9	6	2	0	175 920
10	10	3	1	468 900
11	15	5	2	1 014 750
12	20	7	3	1 626 600
Разом	–	–	–	3 374 110

Таким чином, фінансова модель демонструє реалістичний сценарій комерціалізації, при якому проєкт досягає самоокупності в межах першого року та створює передумови для подальшого масштабування і прибутковості.

Фінансова модель проєкту Headless CMS демонструє поступовий розвиток, що дозволяє вийти на операційну точку беззбитковості вже в межах першого року реалізації. Загальний обсяг інвестицій за 12 місяців становить приблизно 14,4 млн грн, більша частина з яких припадає на фонд оплати праці, технічну інфраструктуру та маркетинг.

Від восьмого місяця, після публікації MVP та запуску SaaS-платформи, проєкт починає генерувати дохід за рахунок підписної моделі. Кожного наступного місяця кількість клієнтів поступово зростає, і вже на 12-му місяці дохід досягає 1 626 600 грн, що перевищує щомісячні витрати (≈1 300 000 грн). Таким чином, проєкт виходить на точку беззбитковості до завершення першого року.

Накопичені грошові потоки за рік залишаються негативними, оскільки прибутки починають формуватись лише з другої половини проєкту. Водночас позитивна динаміка дозволяє очікувати повну окупність інвестицій протягом наступного року, за умови збереження темпів зростання та розширення клієнтської бази.

Загалом, реалізація Headless CMS як open-source SaaS-рішення є інвестиційно привабливою в середньостроковій перспективі. Поєднання

стабільної підписної моделі, гнучкої архітектури, конкурентного функціоналу та стійкого попиту на ринку створює потенціал для масштабування, зростання доходів та досягнення довгострокової фінансової ефективності.

1.6 Мета, завдання та очікувані результати проєкту

Управління контентом на сучасних вебплатформах вимагає інструментів, що забезпечують гнучкість, масштабованість і багатоканальну дистрибуцію. Традиційні CMS часто не відповідають цим вимогам через жорстку зв’язаність фронтенду й бекенду, складність адаптації до різних пристроїв і обмеження в API.

Мета проєкту полягає у створенні кастомізованої Headless CMS на основі open-source рішень, яка дозволить ефективно керувати цифровим контентом незалежно від каналу його публікації. Система повинна бути простою в адмініструванні, розширюваною, безпечною та здатною інтегруватися з будь-якими зовнішніми платформами через API.

Продуктом проєкту виступає Headless CMS із веб-інтерфейсом адміністратора, системою керування колекціями записів, ролями доступу, REST API, а також можливістю гнучкого налаштування структури даних під конкретні бізнес-завдання.

Для досягнення поставленої мети необхідно провести аналітичну роботу щодо сучасного стану CMS-систем, виявити ключові обмеження традиційних рішень та сформулювати функціональні, технічні й безпекові вимоги до майбутньої системи. Далі слід обґрунтувати вибір архітектури, спроєктувати логіку взаємодії між модулями, реалізувати REST API та розробити адаптивний інтерфейс для керування контентом. Окрему увагу потрібно приділити автентифікації, контролю доступу, механізмам резервного копіювання та масштабованості. З управлінського боку проєкт вимагає формування команди, необхідної для реалізації поставлених завдань, розробки плану-графіка виконання робіт із чітко визначеними етапами та відповідальними, а також організації процесів контролю, тестування,

документації та приймання результатів.

Очікувані результати реалізації проєкту включають створення повнофункціонального MVP з реалізованими API, панеллю адміністратора та базовими сценаріями використання. Система має забезпечити скорочення часу на публікацію та оновлення контенту щонайменше на 30%, надати зручні засоби керування сайтами без прив’язки до шаблонного інтерфейсу та бути готовою до масштабування або повторного використання в інших цифрових продуктах.

Критеріями ефективності проєкту визначено стабільну роботу системи після запуску MVP, відповідність реалізованої CMS попередньо визначеним вимогам, позитивну оцінку з боку користувачів (контент-менеджерів і розробників), готовність до розширення функціоналу без змін у ядрі, а також зменшення витрат на технічну підтримку сайтів.

Таким чином, реалізація даного проєкту дозволить організаціям ефективніше керувати контентом, оптимізувати внутрішні процеси, зменшити залежність від технічної підтримки та створити гнучке середовище для цифрової комунікації.

РОЗДІЛ 2. ПРОЄКТНЕ ПЛАНУВАННЯ РОЗРОБКИ HEADLESS CMS

2.1 Побудова WBS: за фазами життєвого циклу, продуктами та процесами

Для ефективного планування та управління проєктом розробки Headless CMS було побудовано WBS (Work Breakdown Structure) – ієрархічну структуру робіт, що декомпонує весь обсяг робіт на логічні складові. Це дозволяє сформувати чітке уявлення про масштаб проєкту, упорядкувати завдання, визначити відповідальних осіб, оцінити ресурси та спланувати часові межі для кожного етапу.

Ієрархію WBS представлено у трьох взаємопов’язаних площинах:

- за фазами життєвого циклу проєкту;
- за кінцевими продуктами;
- за основними процесами.

Таке широке представлення дозволяє врахувати як управлінські, так і технічні аспекти реалізації системи та сформувати цілісну картину виконання проєкту.

2.1.1 Ієрархія за фазами життєвого циклу проєкту

Структурування проєкту за фазами життєвого циклу дозволяє впорядкувати увесь обсяг робіт від ініціювання до завершення, забезпечуючи контроль на кожному етапі реалізації. Для Headless CMS, як комплексного IT-рішення, було сформовано WBS (Work Breakdown Structure), що охоплює всі ключові фази розробки [48].

Життєвий цикл проєкту розбито на вісім основних фаз:

1. Ініціювання – постановка цілей, аналіз доцільності, формування проєктної команди, реєстрація компанії, затвердження статуту.
2. Аналіз вимог – визначення функціональних та нефункціональних вимог, дослідження конкурентного середовища, встановлення критеріїв якості.

3. Планування – побудова структури робіт (WBS), оцінка ресурсів, формування бюджету, управління ризиками, визначення строків.
4. Проєктування – розробка архітектури системи, створення концептуальної моделі бази даних, вибір технологій, створення дизайну інтерфейсу.
5. Розробка – програмування ядра CMS, реалізація API, побудова інтерфейсу адміністратора, інтеграція з сервісами, робота з базою даних.
6. Тестування – створення тест-кейсів, ручне і автоматизоване тестування, перевірка безпеки, тестування на реальних користувачах.
7. Реліз – запуск MVP, підготовка супровідної документації, маркетингова активність, первинна підтримка користувачів.
8. Завершення – аналіз результатів, фінальний аудит, передача документації, архівація проєкту, оцінка досягнення цілей.

Кожна з цих фаз деталізується до рівня конкретних робіт і задач, що дозволяє точно оцінити обсяг виконання, розподілити відповідальність між членами команди та сформувати основу для календарного й ресурсного планування.

На етапі ініціювання передбачено визначення мети, обґрунтування проєкту, аналіз ринку, постановка цілей, формування команди менеджменту, складання статуту та реєстрація компанії. Після цього проводиться глибокий аналіз функціональних і нефункціональних вимог, оцінка конкурентів, визначення пріоритетів і критеріїв якості.

Фаза планування охоплює створення структури робіт, формування проєктної команди, розрахунок ресурсів, оцінку ризиків і формування бюджету. На етапі проєктування виконується розробка системної архітектури, вибір технологічного стеку, хостингу, доменного імені, створення дизайну інтерфейсу.

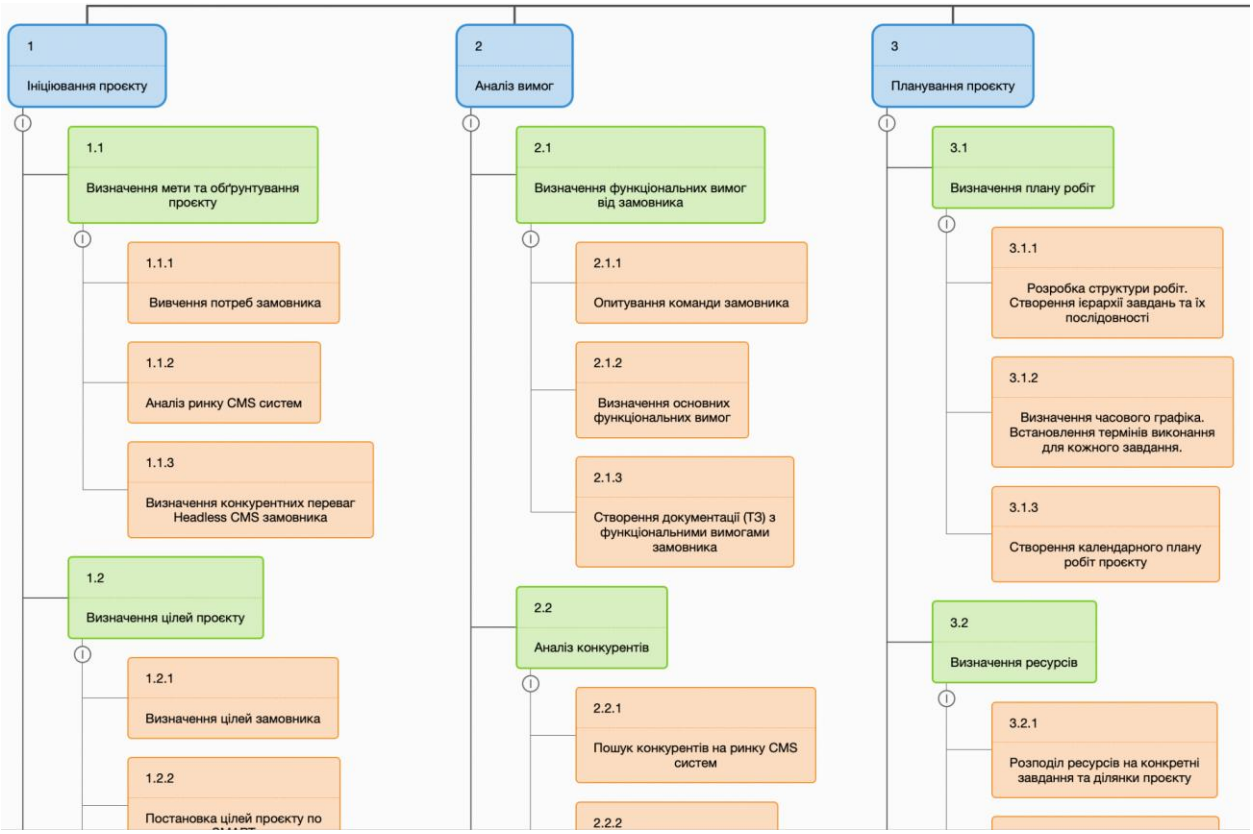


Рис. 2.1. Фрагмент WBS-структури життєвого циклу проєкту

Розробка охоплює створення CMS ядра, API, інтерфейсу користувача, структури БД, інтеграцію з сервісами та наповнення контентом. Після цього реалізується етап тестування – від розробки тест-кейсів до залучення перших реальних користувачів.

Фінальна частина проєкту включає реліз продукту, підготовку документації, маркетинг, супровід, а також завершення проєкту, аудит, архівацію та передачу проєктної документації.

Фрагмент декомпозиції за фазами 1-3 наведена на рис. 2.1 – WBS-структурі життєвого циклу проєкту.

2.1.2 Ієрархія за продуктами проєкту

Крім поділу проєкту за фазами життєвого циклу, важливою є побудова WBS за кінцевими продуктами, які має створити команда. Такий підхід дозволяє зосередитись на конкретних результатах, які необхідно досягти, і забезпечує цілісне уявлення про складові майбутньої системи.

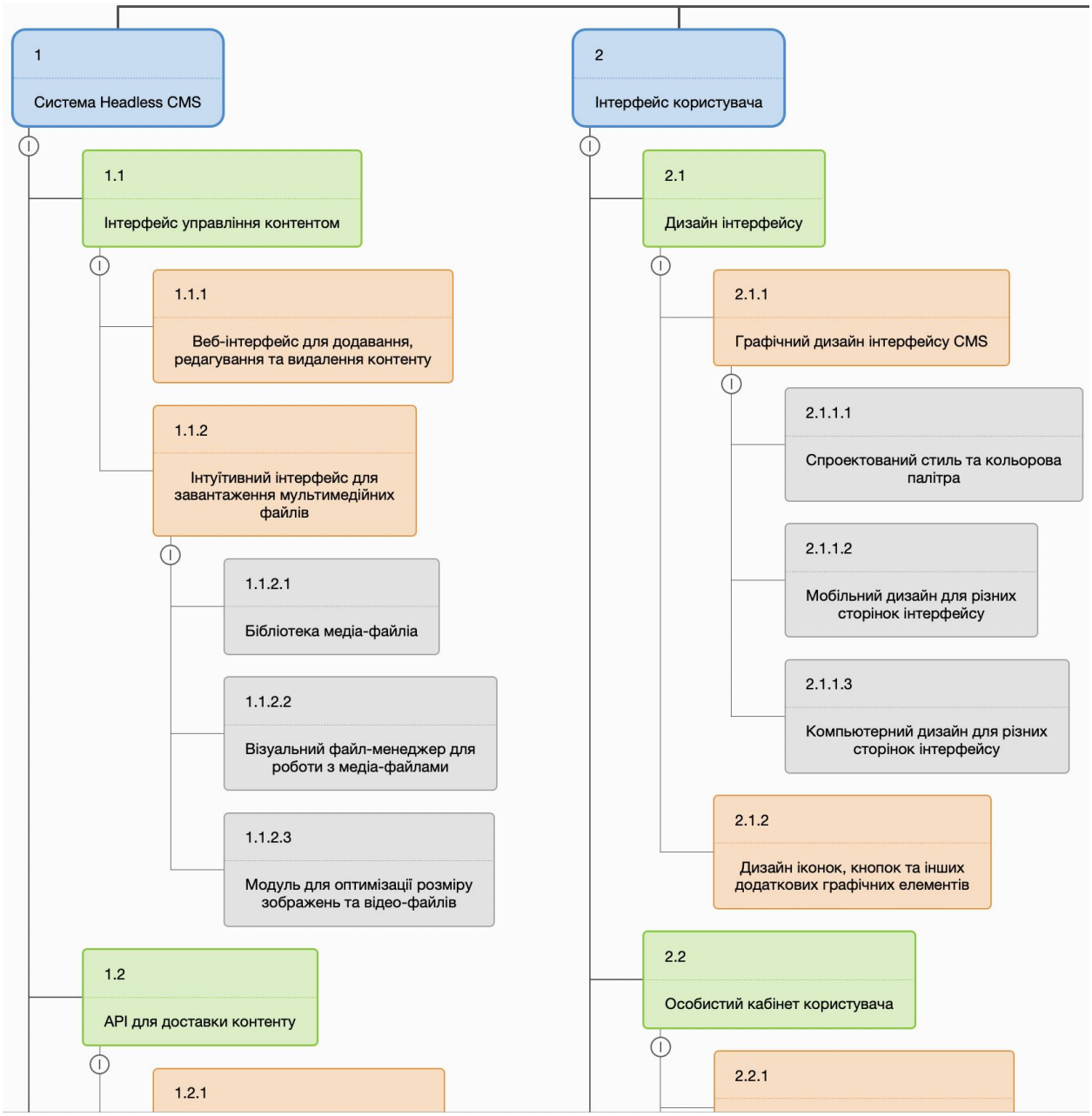


Рис. 2.2. Фрагмент WBS-структури продуктів проєкту

У межах проєкту створення Headless CMS виділено чотири головні продуктові блоки:

1. Система Headless CMS, яка включає веб-інтерфейс для керування контентом, API, автентифікацію та систему доступу, структуру зберігання даних, модуль версіювання, інтеграцію зі сторонніми сервісами, пошук, аналітику, сповіщення та керування ролями.
2. Інтерфейс користувача, який охоплює графічний дизайн, особистий кабінет користувача, інструменти редагування

контенту, адаптивність для мобільних пристроїв, багатомовність, доступність і зручність використання.

3. Інструменти для розробників, до яких належать API-документація, SDK та бібліотеки, інструкції з інтеграції, тестове середовище, система контролю версій, а також фреймворк для розширень і плагінів.
4. Документація та підтримка, яка включає інструкції для користувачів, технічну документацію, портал підтримки, навчальні матеріали, базу знань та механізми для відстеження помилок.

Кожен з цих блоків деталізується до рівня підсистем, модулів та конкретних компонентів. Наприклад, блок API включає RESTful і GraphQL-інтерфейси, механізми автентифікації, контроль доступу та документацію для інтеграції. Інтерфейс адміністратора охоплює редактор контенту, менеджер мультимедійних файлів, інструменти пошуку, навігації та аналітики.

Представлена ієрархія дозволяє не лише ефективно планувати розробку, а й визначити залежності між компонентами, спростити тестування та забезпечити модульність системи. Фрагмент WBS за 2 продуктами проєкту наведено на рис. 2.2.

2.1.3 Ієрархія за процесами проєкту

Ще одним підходом до побудови ієрархічної структури робіт є декомпозиція проєкту за ключовими процесами. Такий погляд дозволяє виділити основні напрями діяльності команди, визначити відповідальних за кожен процес, а також забезпечити управління на горизонтальному рівні – незалежно від фаз або продуктів.

У проєкті розробки Headless CMS було визначено п’ять основних процесних напрямів:

1. Процес розробки системи охоплює дизайн, програмування CMS, створення інтерфейсу, API, бази даних, інтеграцію з сервісами,

тестування та початкове наповнення контентом. Роботи в межах цього процесу зосереджені на створенні функціональної частини продукту та підготовці його до впровадження.

2. Процес забезпечення IT-інфраструктури, що включає налаштування серверів, хостингу, DNS, безпеки, резервного копіювання, а також організацію моніторингу систем. Цей блок забезпечує технічну стабільність та готовність середовища до розгортання проєкту.
3. Процес маркетингу, у межах якого відбувається аналіз цільової аудиторії, розробка маркетингової стратегії, створення промо-матеріалів, запуск рекламних кампаній і робота з підрядниками. Цей напрям є критично важливим для просування CMS на ринку або для її внутрішнього впровадження з урахуванням цільових груп користувачів.
4. Процес управління проєктом, який охоплює управління змістом, якістю, ресурсами, часом, комунікацією, а також звітування перед замовником. Всі ці елементи забезпечують координовану реалізацію проєкту відповідно до встановлених термінів, стандартів і вимог.
5. Процес бухгалтерського та фінансового супроводу, який включає облік, звітність, нарахування зарплати, бюджетування, контроль витрат і податкову звітність. Наявність цього напрямку гарантує прозорість фінансової складової проєкту та відповідність регуляторним вимогам.

Фрагмент WBS-структура за 2 процесами наведена на рис. 2.3, і є логічним доповненням до попередніх WBS-моделей. Її використання дозволяє ефективно організувати командну роботу в межах обраної гнучкої методології Scrum, де процеси планування, розробки, тестування, взаємодії зі стейкхолдерами та підтримки реалізуються циклічно в межах спрінтів. Така структура сприяє прозорості, взаєморозумінню між учасниками команди та

гнучкому реагуванню на зміни під час реалізації проєкту.

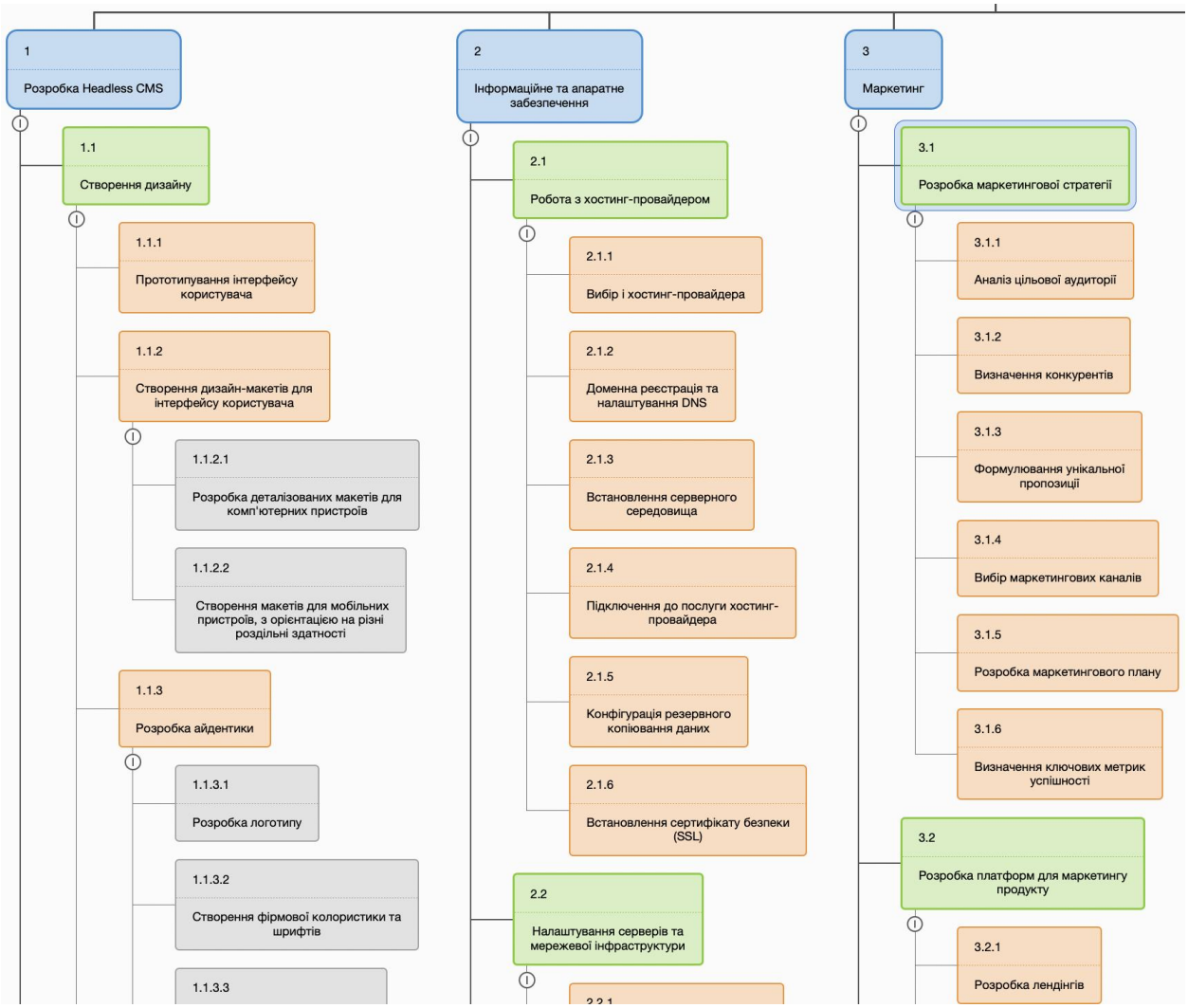


Рис. 2.3. Фрагмент WBS-структури за процесами проєкту

2.2 Організаційна модель компанії та проєктної команди

2.2.1 Організаційна структура компанії

Організаційна структура проєкту з розробки Headless CMS побудована відповідно до сучасних принципів управління IT-проєктами з акцентом на прозорість, поділ відповідальності та інтеграцію функціональних і проєктних ролей. Такий підхід дозволяє забезпечити злагоджену взаємодію між усіма учасниками та підвищити загальну ефективність реалізації проєкту. Структура представлена на Рис. 2.4.

На вищому рівні управління знаходиться генеральний директор, який визначає стратегічні цілі компанії та здійснює загальний нагляд за ключовими

ініціативами, включно з реалізацією Headless CMS. Директор українського офісу виконує функції операційного координатора, відповідального за забезпечення комунікації між центральним керівництвом і локальною командою, а також за дотримання регіональних норм і політик.

Центральною фігурою проєктної команди є керівник проєкту, який відповідає за планування, моніторинг і координацію всіх етапів розробки. Він слідкує за виконанням термінів, дотриманням бюджету, організовує щоденні стендапи й спринт-планування згідно з методологією Scrum. У межах своєї відповідальності керівник проєкту взаємодіє з усіма учасниками, забезпечуючи своєчасне прийняття рішень і вирішення ризиків.

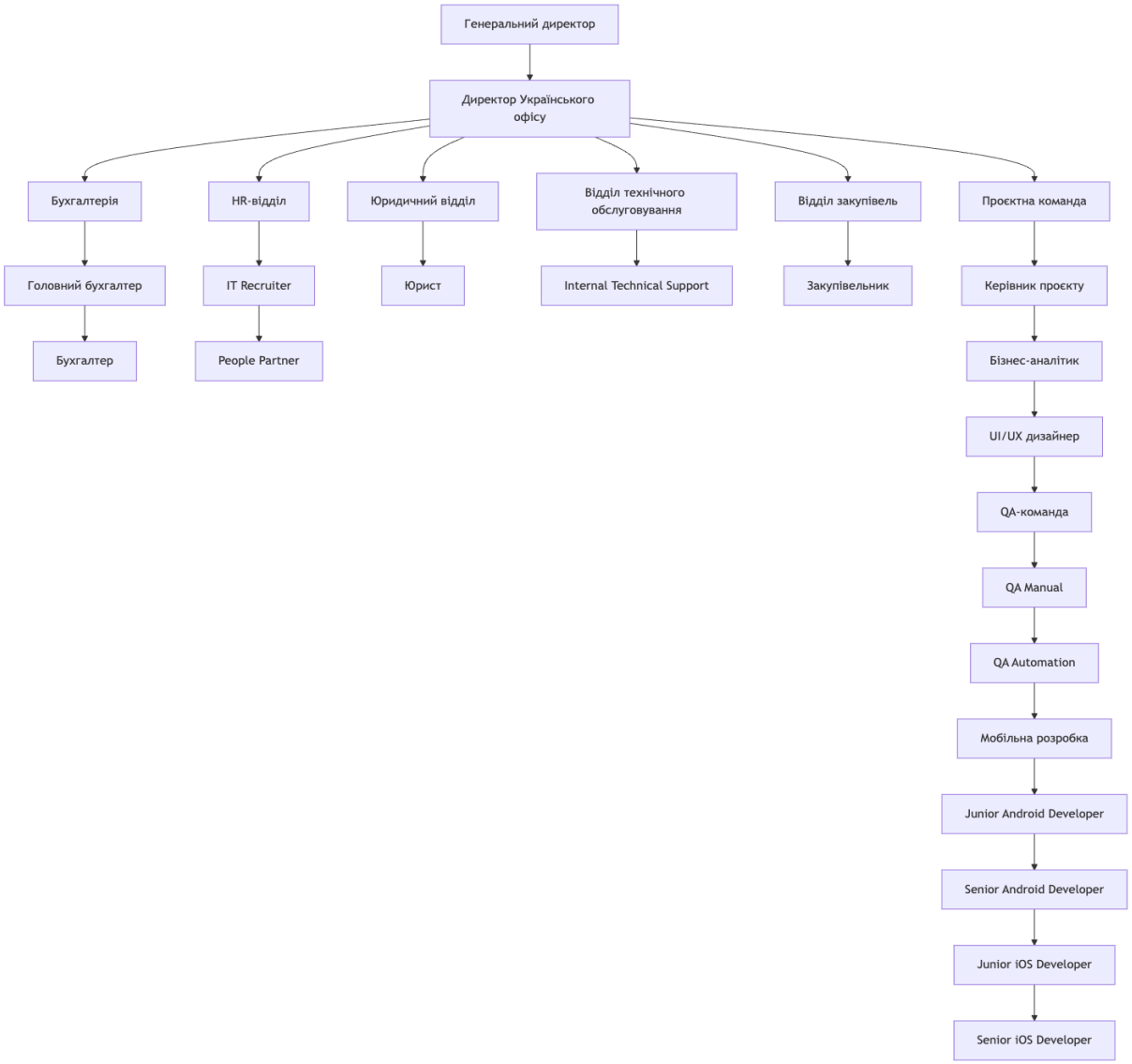


Рис. 2.4. Організаційна структура компанії

Для збирання та систематизації вимог до продукту проєкт передбачає

участь бізнес-аналітика, який проводить інтерв'ю з клієнтом, моделює функціональність, документує user stories та готує технічну документацію. Саме він виступає посередником між замовником і командою розробки, трансформуючи бізнес-потреби у чіткі технічні задачі.

Зовнішній вигляд та логіка інтерфейсу визначаються UI/UX дизайнером. Його завдання – розробка інтуїтивно зрозумілої структури екранів, побудова макетів, створення інтерактивних прототипів, а також участь у тестуванні дизайну з користувачами для забезпечення високого рівня зручності.

Усі етапи тестування функціоналу виконує QA-команда, що складається з інженерів ручного тестування (QA Manual) та спеціалістів з автоматизованого тестування (QA Automation). Вони розробляють тест-кейси, виявляють помилки, проводять регресійне тестування та забезпечують відповідність продукту очікуванням користувача.

Команда розробників виконує основну технічну реалізацію продукту. До її складу входять як junior-, так і senior-розробники, що дозволяє розподіляти задачі відповідно до рівня складності та досвіду. Команда може включати спеціалістів за напрямками front-end, back-end, API, бази даних та інтеграцій. Завдяки гнучкому складу, команда здатна масштабуватися залежно від потреб спринтів або окремих модулів системи.

Паралельно з основною командою функціонують підрозділи підтримки: HR-відділ, який відповідає за підбір, адаптацію та мотивацію персоналу; бухгалтерія – за фінансовий супровід; юридичний відділ – за правове забезпечення; технічний відділ – за обслуговування інфраструктури; а також відділ закупівель, що забезпечує ресурсну підтримку.

Завдяки чітко структурованій ієрархії організаційної моделі проєкт отримує усі необхідні ресурси – як людські, так і процесні – для успішної реалізації на кожному етапі життєвого циклу. Така структура не тільки відповідає стандартам управління проєктами, а й забезпечує гнучкість, необхідну для швидкої адаптації до змін ринку та вимог замовника.

2.2.2 Ролі учасників та зони відповідальності

Розподіл ролей у межах проєкту є ключовим чинником успішної реалізації функціоналу, дотримання термінів та забезпечення якості кінцевого продукту. Всі учасники команди мають чітко окреслені зони відповідальності, що відповідають їх компетенціям, рівню прийняття рішень і внеску у досягнення цілей проєкту. Така модель дозволяє не лише уникнути дублювання функцій, а й забезпечити ефективну взаємодію в умовах гнучкої методології Scrum.

Керівник проєкту виконує функції координатора всіх етапів реалізації, відповідає за планування спринтів, організацію командної взаємодії, контроль виконання задач, а також за управління ризиками, термінами та ресурсами. Він є основною ланкою між замовником і командою, забезпечуючи узгодження очікувань з технічними можливостями.

Бізнес-аналітик забезпечує трансформацію потреб клієнта у формалізовані вимоги, створює user stories та технічні специфікації, бере участь у пріоритизації задач та перевіряє, наскільки реалізовані функції відповідають очікуванням користувача. Його зона відповідальності охоплює етапи збору, аналізу, уточнення та валідації вимог.

UI/UX дизайнер розробляє макети інтерфейсів, прототипи, елементи візуального оформлення та архітектуру взаємодії з системою. Він забезпечує відповідність між очікуваним користувацьким досвідом і функціональною частиною системи, працює над адаптивністю, доступністю та інтуїтивністю взаємодії.

Тестувальники відповідають за якість продукту на всіх рівнях. QA Manual проводить ручне тестування функціоналу, готує чек-листи, тест-кейси та звіти про баги. QA Automation створює та підтримує автоматизовані тести, що дозволяють перевіряти стабільність роботи системи після кожного оновлення. Разом вони забезпечують постійний зворотний зв’язок про якість продукту.

Розробники реалізують усі функціональні компоненти системи

відповідно до архітектури та технічного завдання. Вони розробляють backend-логіку CMS, реалізують API, створюють frontend-інтерфейси та здійснюють інтеграцію з базою даних, сервісами аналітики, пошуку, керування файлами тощо. До обов'язків розробників також належить написання технічної документації, рефакторинг коду та участь у code review.

Допоміжні ролі – HR-фахівець, бухгалтер, юрист, внутрішній технічний спеціаліст та спеціаліст із закупівель – не залучені безпосередньо до технічної реалізації, проте забезпечують необхідну адміністративну, юридичну, кадрову та фінансову підтримку. Їхня діяльність дозволяє команді зосередитися на ключових завданнях розробки без додаткових операційних навантажень.

Ролі в команді взаємодіють на щоденній основі, беручи участь у плануванні, оцінюванні задач, спільному тестуванні та презентації результатів. Такий розподіл функціоналу сприяє гнучкості процесів, швидкому прийняттю рішень та мінімізації ризиків, пов'язаних з неузгодженістю або дублюванням обов'язків.

2.2.3 Матриця відповідальності за напрямами робіт

Для забезпечення прозорості взаємодії між членами команди проєкту, чіткого розподілу функцій та уникнення дублювання завдань було складено матрицю відповідальності у форматі RACIS. Цей інструмент дозволяє зіставити ключові етапи проєкту з відповідними ролями, закріпивши рівень участі кожного спеціаліста у тій чи іншій діяльності.

Формат RACIS є одним із найпоширеніших у проєктному менеджменті. Його суть полягає в позначенні ступеня залученості кожної ролі за допомогою п'яти букв:

- R (Responsible) – відповідальний за виконання завдання або напряму роботи; саме ця роль виконує основну частину діяльності;
- A (Accountable) – відповідальний за прийняття рішень та кінцевий результат; несе фінальну відповідальність;

- C (Consulted) – консультує або надає важливий експертний внесок у процес;
- I (Informed) – інформується про хід або результати виконання, але не бере участі безпосередньо;
- S (Support) – технічно або адміністративно підтримує процес.

Такий підхід дозволяє швидко зчитувати, хто що робить на кожному етапі, і водночас формує основу для побудови відповідальності, внутрішнього контролю та ефективного планування навантаження на команду.

Для скорочення тексту в заголовках таблиці 2.1 використано умовні позначення ролей:

- C1 – Менеджер проєкту
- C2 – Бізнес-аналітик
- C3 – Дизайнер (UI/UX)
- C4 – Розробники
- C5 – Тестувальники
- C6 – Юрист
- C7 – Закупівельник

У таблиці 2.1 наведено детальну матрицю відповідальності RACIS для основних етапів реалізації проєкту розробки Headless CMS. Вона охоплює як стратегічні, так і технічні, юридичні та адміністративні процеси, від ініціації до завершення.

Таблиця 2.1

Матриця відповідальності RACIS

Етапи проєкту / Виконавці	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Управління проєктом	R A	I	I	I	I	I	I

Продовження таблиці 2.1

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Ініціювання проєкту	R A	C	I	I	I	I	I
Формування команди	R A	C	I	I	I	I	I
Збір та аналіз вимог	C	R A	C	I	I	I	I
Формалізація бізнес-вимог	I	R A	I	I	I	I	I
Постановка задач у вигляді технічного ТЗ	C	R A	C	C	I	I	I
Розробка концептуальної моделі бази даних	I	C	I	R C A	I	I	I
Проєктування логічної структури бази даних	I	C	I	R A	I	I	I
Побудова фізичної моделі бази даних	I	C	I	R C A	I	I	I
Архітектура Headless CMS	C	C	I	R A	I	I	I
Закупівля серверного ПЗ, хостингу або ліцензій	I	I	I	C	I	I	R A
Створення API для управління контентом	C	I	I	R A	I	I	I
Розробка візуального інтерфейсу (UI)	I	I	R A	C	C	I	I
UX-дизайн CMS-адмінки	I	I	R A	I	C	I	I
Адаптація під мобільні/настільні екрани	I	I	R A	C	I	I	I
Програмна реалізація бекенду	C	I	I	R A	C	I	I
Розробка фронтенду (інтеграція CMS із UI)	I	I	C	R A	C	I	I
Модульне та інтеграційне тестування	C	I	I	C	R A	I	I
Перевірка юридичної відповідності ПЗ	I	I	I	I	I	R A	I
Завершення проєкту, аудит та здача	R A	C	C	C	C	C	I

2.3 Календарне планування етапів реалізації

Календарне планування є одним з найважливіших інструментів ефективного управління IT-проєктом, оскільки дозволяє забезпечити логічну послідовність виконання робіт, оптимізувати завантаження ресурсів, мінімізувати часові втрати та контролювати ризики затримок. В умовах гнучкої методології управління (Scrum), де проєкт розбивається на короткі ітерації (спринти), наявність детального календарного плану дозволяє підтримувати прозору координацію між учасниками команди та сприяє досягненню ключових контрольних точок у задані строки.

Процес побудови календарного плану ґрунтується на результатах попередніх етапів – структуризації завдань у WBS-модель, формуванні матриці відповідальності, аналізі вимог до системи та проектному плануванні. Кожне завдання було описано з точки зору очікуваного результату, тривалості виконання, а також залежності від попередніх робіт. Таким чином, ми отримали повний перелік задач, які відображають як технічну, так і організаційну динаміку реалізації проєкту.

Для створення календарного плану та візуалізації зв’язків між задачами було використано програмне забезпечення ProjectLibre – безкоштовний аналог MS Project, сумісний з macOS. Даний інструмент дозволяє будувати діаграми Ґанта, мережеві графіки, розраховувати критичний шлях, а також формувати календарний план із прив’язкою до конкретних ресурсів. Його використання є оптимальним для невеликих IT-команд з обмеженим бюджетом, які працюють у середовищі з різними операційними системами.

У процесі планування застосовано логіку зв’язків типу «фініш-старт» (Finish-to-Start, FS) – найбільш поширену у проєктному менеджменті. Вона означає, що наступна задача може розпочатися лише після повного завершення попередньої. Це дозволяє будувати лінійно-логічну послідовність дій, виявляти критичні точки затримки та наочно відображати проєкт у вигляді діаграми Ґанта, мережевої моделі або інтегрованого календаря.

Особливістю планування цього проєкту є висока деталізація на рівні

конкретних задач (до 120), що охоплюють ініціацію, планування, аналіз, розробку, тестування, реліз, супровід MVP і підготовку до наступних фаз розвитку CMS. Це дозволяє не лише здійснювати контроль прогресу, але й швидко адаптуватися до змін у вимогах, навантаженні або зовнішніх умовах.

Перелік робіт по проєкту з вказанням тривалості, дати початку та логічного попередника (зв'язок «фініш-старт») наведено у таблиці 2.2.

Таблиця 2.2

Перелік робіт за проєктом

№	Назва роботи	Попередні роботи
1	**2**	**3**
1	Ініціалізація проєкту	
2	Формування проєктної команди	1
3	Призначення керівника проєкту	2
4	Визначення цілей і KPI проєкту	3
5	Проведення вступної наради з командою	4
6	Визначення вимог до CMS на основі інтерв'ю	5
7	Аналіз ринку та конкурентів	6
8	Аналіз ризиків	6
9	Вибір методології управління (Scrum)	7,8
10	Створення статуту проєкту	9
11	Визначення зацікавлених сторін	10
12	Побудова WBS-структури	11
13	Розробка матриці відповідальності	12
14	Побудова організаційної структури проєкту	12
15	Планування спринтів	13,14
16	Складання ризик-реєстру	15
17	Створення комунікаційного плану	15
18	Розробка календарного плану	15
19	Планування бюджету	15

Продовження таблиці 2.2

1	2	3
20	Інтерв’ю з цільовими користувачами	18
21	Проведення опитування для збору вимог	20
22	Формалізація вимог	21
23	Побудова user-flow діаграм	22
24	Визначення MVP	22
25	Узгодження функціональних вимог	24
26	Узгодження нефункціональних вимог	24
27	Розробка технічного завдання для дизайнера	25,26
28	Розробка технічного завдання для бекенду	25,26
29	Розробка технічного завдання для API	25,26
30	Розробка технічного завдання для адмінпанелі	25,26
31	Узгодження ТЗ з командою	27,28,29,30
32	Розробка інформаційної архітектури	31
33	Розробка структури CMS (контентні типи, модулі)	31
34	Створення вайрфреймів (Low-fidelity)	32
35	Створення UI-концепції	34
36	Розробка UI для публічної частини	35
37	Розробка UI для адмін-панелі	35
38	Створення адаптивних макетів	36,37
39	Збір і затвердження дизайн-системи	38
40	Прототипування у Figma	38,39
41	Презентація дизайну команді	40
42	Налаштування репозиторію	41
43	Ініціалізація проєкту	42
44	Підготовка шаблону CMS (Open-source база)	43
45	Розробка структури бази даних	44
46	Налаштування середовища розробки	44
47	Підключення авторизації користувачів	45,46
48	Створення модулю "Користувачі"	47
49	Створення модулю "Категорії"	48

Продовження таблиці 2.2

1	2	3
50	Створення модулю "Матеріали"	49
51	Вивід контенту через REST API	50
52	Розробка фільтрації та пошуку матеріалів	50
53	Додавання підтримки тегів	50
54	Налаштування управління ролями	47,48
55	Реалізація редактора контенту	51
56	Реалізація медіабібліотеки	55
57	Завантаження та зберігання зображень	56
58	Інтеграція CKEditor або аналога	55
59	Додавання історії змін / ревізій	55
60	Базова аналітика використання CMS	54,55
61	Налаштування доступу через токени	54
62	Написання тестових сценаріїв	55,56
63	Проведення модульного тестування	62
64	Тестування UI	58
65	Перевірка відповідності функціональним вимогам	63
66	Усунення виявлених помилок	65
67	Оцінка продуктивності	63
68	Аналіз помилок логування	63
69	Тестування API на відповідність документації	51,63
70	Перевірка адаптивності інтерфейсу	64
71	Інтеграція з email-сервісом	51
72	Підключення SEO-модулю	51
73	Додавання JSON-LD для структурованих даних	72
74	Написання документації для користувачів	66
75	Написання документації для розробників	66
76	Формування API-специфікацій	51
77	Опис архітектури CMS	45,51
78	Формування README для GitHub	75,76

Продовження таблиці 2.2

1	2	3
79	Створення інструкції для деплойменту	74,75
80	Написання changelog	75
81	Підготовка серверного середовища	77
82	Створення docker-образів	81
83	Налаштування CI/CD	82
84	Перенесення даних на staging	82
85	Проведення smoke-тестування	84
86	Перенесення на production	83,85
87	Проведення фінального тестування	86
88	Презентація готового рішення	87
89	Прийомка від замовника	88
90	Створення резервної копії	86
91	Збір фідбеку від замовника	89
92	Фікс критичних помилок	91
93	Проведення оновлення безпеки	92
94	Аналіз навантаження на сервер	87
95	Налаштування масштабованості	94
96	Моніторинг продуктивності	94
97	Підтримка користувачів (1 лінія)	89
98	Визначення зони для покращень	91
99	Оновлення документації	92,98
100	Аналіз успішності впровадження	91,97
101	Оформлення технічного звіту	100
102	Створення фінального презентаційного матеріалу	100
103	Проведення фінальної зустрічі з командою	101,102
104	Фінальний аналіз досягнень по KPI	101
105	Оцінка командної взаємодії	103
106	Архівація проєкту	103
107	Передача прав та документації	103

Продовження таблиці 2.2

1	2	3
108	Закриття облікових записів	107
109	Відключення test/stage середовищ	108
110	Завершення проєкту	106,107,109
111	Аналіз відгуків користувачів	97
112	Ініціація технічного боргу	111
113	Оптимізація бази даних	112
114	Запуск контентної кампанії	100
115	Додавання нових модулів	112
116	Планування наступної фази розвитку	110,115
117	Підготовка до open-source публікації	113,115
118	Оцінка повторного використання коду	117
119	Звіт про економічну ефективність	104,118
120	Остаточний аналіз проєкту	110,11

Побудова переліку робіт здійснювалася з урахуванням принципів класичного життєвого циклу управління IT-проєктами, адаптованого до особливостей створення Headless CMS з глибокою кастомізацією. При розробці плану було використано ітеративно-інкрементальний підхід, характерний для гнучких методологій (зокрема Scrum), однак з формальним поділом на фази, що дозволяє зберегти структурованість і керованість реалізації.

Загальна структура календарного плану включає 12 логічно послідовних фаз, кожна з яких об’єднує групу взаємопов’язаних завдань. Такий поділ дає змогу забезпечити:

- чітке розмежування відповідальності;
- контроль за динамікою розробки;
- логічну послідовність реалізації залежних етапів;
- зручність для побудови візуальних моделей (Ґант, PERT);
- можливість масштабування та адаптації у разі змін.

У таблиці 2.2 роботи впорядковані відповідно до цієї структури, а логіка їх формування відображає поетапний підхід до реалізації IT-проєкту з розробки Headless CMS.

Перший етап – Ініціація проєкту (завдання 1–11) – охоплює стартові дії: формулювання ідеї, визначення ключових ролей, затвердження цілей і метрик ефективності, а також створення хартії та ідентифікацію зацікавлених сторін. Ці кроки закладають фундамент для усієї подальшої роботи.

Далі реалізується етап планування (завдання 12–19), у межах якого формуються основні артефакти управління: WBS-структура, матриця відповідальності (RACIS), організаційна структура, календарний план, бюджет, а також плани ресурсів і ризиків. Саме тут проєкт набуває чіткої структурної рамки.

Наступною логічною фазою є збір і формалізація вимог (завдання 20–31). У цій частині фіксуються функціональні та нефункціональні вимоги до CMS, проводиться визначення MVP, а також формується технічна документація для різних підкоманд (дизайн, бекенд, API).

Дизайнерська фаза (завдання 32–41) включає створення інформаційної архітектури, вайрфреймів, адаптивних макетів, дизайн-системи та прототипів. Цей етап критично важливий для подальшого користувацького досвіду, що формуватиметься на базі розробленого інтерфейсу.

Розробка (спринт 1) – завдання 42–54 – фокусується на бекенд-компоненті системи: структури бази даних, модулях управління контентом, авторизації та базовому API. У спринті 2 (завдання 55–61) – реалізується інтерфейсна частина, редактор, медіабібліотека, пошук і інші інтерактивні елементи.

Після цього відбувається фаза тестування (завдання 62–70), де команда перевіряє відповідність розроблених компонентів ТЗ, проводить модульне й інтеграційне тестування, аналіз продуктивності та стійкість до помилок.

На етапі інтеграції та документації (завдання 71–80) команда впроваджує сторонні сервіси (email, SEO), створює внутрішню документацію,

готує інструкції та забезпечує проектну спадкоємність.

Релізна фаза (завдання 81–90) передбачає розгортання staging- та production-версій, перевірку стабільності, CI/CD, smoke-тестування, презентацію замовнику та резервне копіювання.

Післярелізна підтримка (завдання 91–100) містить аналіз фідбеку, оновлення, виправлення критичних помилок, масштабування та технічну підтримку користувачів.

Останні два етапи – завершення (завдання 101–110) та розвиток (завдання 111–120) – передбачають підбиття підсумків, формування звітності, передачу документації, оцінку економічної ефективності та підготовку до наступної фази розвитку проєкту або публікації CMS як open-source рішення.

Такий поетапний підхід дозволив забезпечити послідовність, логіку та керованість усіх дій у проєкті, враховуючи як технологічні, так і організаційні аспекти.

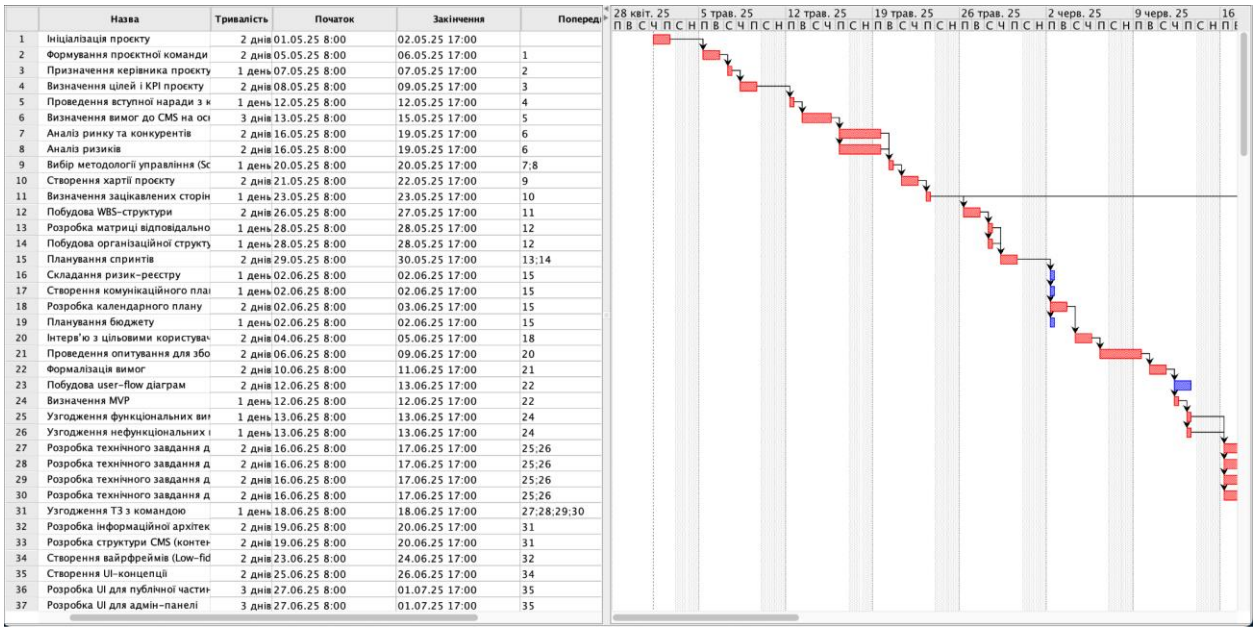
Для візуалізації календарного плану, встановлення логічних зв’язків між завданнями, аналізу строків і виявлення критичних точок у реалізації проєкту застосовано два класичні інструменти управління проєктами: діаграму Ґанта та мережеву діаграму (PERT/CPM). Обидві моделі були побудовані в програмному середовищі ProjectLibre, яке є зручним і доступним інструментом для команд, що працюють з macOS.

Діаграма Ґанта (рис. 2.5) демонструє усі 120 завдань проєкту, їхню тривалість, дату початку, а також зв’язки типу «фініш-старт». Горизонтальні смуги в діаграмі відповідають конкретним задачам і наочно ілюструють планову тривалість кожної фази. Для полегшення навігації завдання згруповані відповідно до фаз (ініціація, планування, аналітика, розробка тощо), що дозволяє бачити загальний прогрес як у межах окремих блоків, так і по всьому проєкту.

Додатково візуально позначено критичний шлях – послідовність взаємозалежних задач, затримка в яких напряму впливає на загальні строки завершення проєкту. Саме ці задачі мають нульовий резерв часу та потребують

найбільшої уваги в процесі реалізації. Використання діаграми Ґанта дає змогу не лише планувати, а й оперативно реагувати на відставання чи перенавантаження ресурсів.

Разом із цим, для глибшого розуміння логіки виконання робіт та визначення критичних вузлів у структурі задач була побудована мережева діаграма (рис. 2.6). Вона ґрунтується на алгоритмах PERT/CPM і дозволяє аналізувати завдання у вигляді графа з вузлами та дугами. Такий формат особливо корисний для візуалізації залежностей, знаходження найкоротших і найдовших шляхів реалізації, а також розрахунку ранніх та пізніх строків початку і завершення кожної задачі.



	Назва	Тривалість	Початок	Закінчення	Попереді
1	Ініціалізація проєкту	2 днів	01.05.25 8:00	02.05.25 17:00	
2	Формування проєктної команди	2 днів	05.05.25 8:00	06.05.25 17:00	1
3	Призначення керівника проєкту	1 день	07.05.25 8:00	07.05.25 17:00	2
4	Визначення цілей і KPI проєкту	2 днів	08.05.25 8:00	09.05.25 17:00	3
5	Проведення вступної наради з к	1 день	12.05.25 8:00	12.05.25 17:00	4
6	Визначення вимог до CMS на ос	3 днів	13.05.25 8:00	15.05.25 17:00	5
7	Аналіз ринку та конкурентів	2 днів	16.05.25 8:00	19.05.25 17:00	6
8	Аналіз ризиків	2 днів	16.05.25 8:00	19.05.25 17:00	6
9	Вибір методології управління (Sc	1 день	20.05.25 8:00	20.05.25 17:00	7;8
10	Створення хартії проєкту	2 днів	21.05.25 8:00	22.05.25 17:00	9
11	Визначення зацікавлених сторін	1 день	23.05.25 8:00	23.05.25 17:00	10
12	Побудова WBS-структури	2 днів	26.05.25 8:00	27.05.25 17:00	11
13	Розробка матриці відповідально	1 день	28.05.25 8:00	28.05.25 17:00	12
14	Побудова організаційної структу	1 день	28.05.25 8:00	28.05.25 17:00	12
15	Планування спринтів	2 днів	29.05.25 8:00	30.05.25 17:00	13;14
16	Складання ризик-реєстру	1 день	02.06.25 8:00	02.06.25 17:00	15
17	Створення комунікаційного пла	1 день	02.06.25 8:00	02.06.25 17:00	15
18	Розробка календарного плану	2 днів	02.06.25 8:00	03.06.25 17:00	15
19	Планування бюджету	1 день	02.06.25 8:00	02.06.25 17:00	15
20	Інтерв'ю з цільовими користува	2 днів	04.06.25 8:00	05.06.25 17:00	18
21	Проведення опитування для збо	2 днів	06.06.25 8:00	09.06.25 17:00	20
22	Формалізація вимог	2 днів	10.06.25 8:00	11.06.25 17:00	21
23	Побудова user-flow діаграм	2 днів	12.06.25 8:00	13.06.25 17:00	22
24	Визначення MVP	1 день	12.06.25 8:00	12.06.25 17:00	22
25	Узгодження функціональних ви	1 день	13.06.25 8:00	13.06.25 17:00	24
26	Узгодження нефункціональних	1 день	13.06.25 8:00	13.06.25 17:00	24
27	Розробка технічного завдання д	2 днів	16.06.25 8:00	17.06.25 17:00	25;26
28	Розробка технічного завдання д	2 днів	16.06.25 8:00	17.06.25 17:00	25;26
29	Розробка технічного завдання д	2 днів	16.06.25 8:00	17.06.25 17:00	25;26
30	Розробка технічного завдання д	2 днів	16.06.25 8:00	17.06.25 17:00	25;26
31	Узгодження ТЗ з командою	1 день	18.06.25 8:00	18.06.25 17:00	27;28;29;30
32	Розробка інформаційної архітек	2 днів	19.06.25 8:00	20.06.25 17:00	31
33	Розробка структури CMS (контен	2 днів	19.06.25 8:00	20.06.25 17:00	31
34	Створення вайрфреймів (Low-fid	2 днів	23.06.25 8:00	24.06.25 17:00	32
35	Створення UI-концепції	2 днів	25.06.25 8:00	26.06.25 17:00	34
36	Розробка UI для публічної части	3 днів	27.06.25 8:00	01.07.25 17:00	35
37	Розробка UI для адмін-панелі	3 днів	27.06.25 8:00	01.07.25 17:00	35

Рис. 2.5. Фрагмент календарного плану в середовищі ProjectLibre

На відміну від діаграми Ґанта, мережева модель фокусує увагу не лише на часовій шкалі, а й на структурі логічних зв’язків між задачами, що робить її надзвичайно корисною для виявлення слабких місць у плані. Мережа дозволяє визначити критичний шлях аналітично та переконатися в обґрунтованості календарного планування.

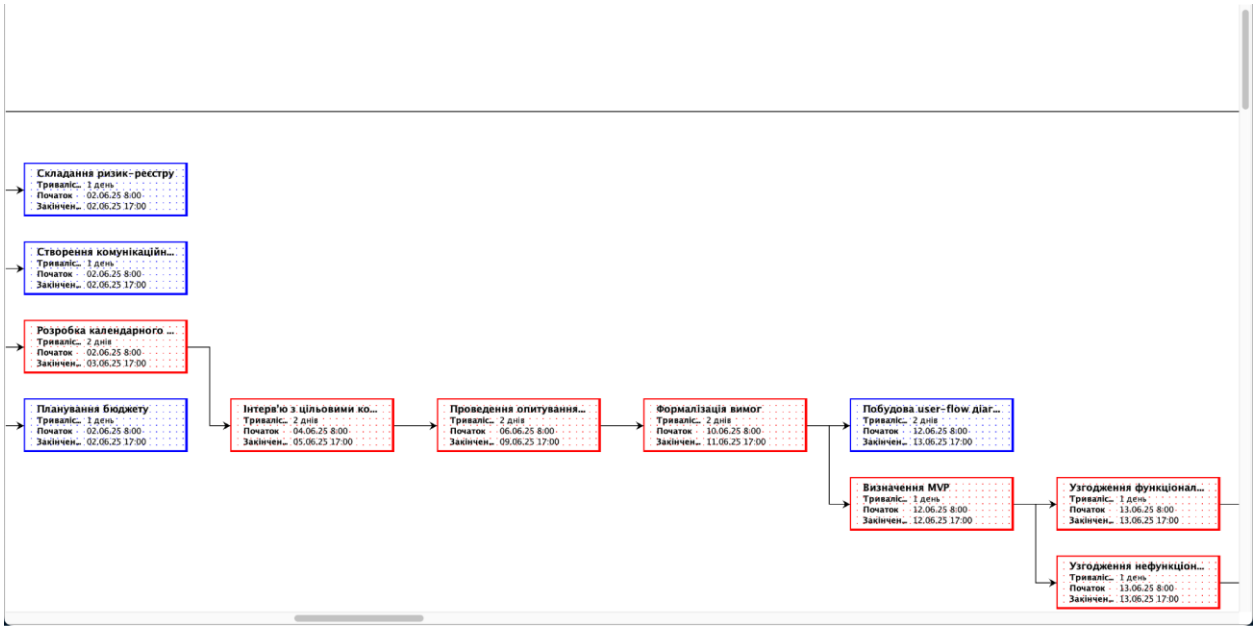


Рис. 2.6. Фрагмент мережевої діаграми в середовищі ProjectLibre

Таким чином, поєднання діаграми Ґанта та мережевої моделі в межах одного плану забезпечує комплексне бачення проєкту: як у часовому вимірі, так і в структурному. Це дає змогу з високою точністю планувати реалізацію, забезпечувати контроль над критичними етапами, ефективно розподіляти ресурси й оперативно реагувати на ризики.

2.4 Планування ресурсів проєкту

2.4.1 Людські ресурси

Планування людських ресурсів ґрунтується на організаційній структурі компанії та проєктної команди (рис. 2.4). Кожна роль має чітке функціональне призначення, яке охоплює всі етапи життєвого циклу проєкту – від аналізу вимог до запуску Headless CMS.

До основної проєктної команди входять менеджер проєкту, бізнес-аналітик, UI/UX дизайнер, розробники та тестувальники. Вони беруть участь у безпосередній розробці системи, включно з підготовкою технічної документації, створенням функціоналу, дизайну інтерфейсів та перевіркою якості продукту.

До ролей підтримки належать юрист, закупівельник, фахівці з

технічного обслуговування та HR-відділу. Вони забезпечують юридичну, кадрову й технічну підтримку, а також відповідають за придбання обладнання.

Весь перелік людських ресурсів для реалізації проєкту наведено у таблиці 2.3.

Таблиця 2.3

Перелік людських ресурсів

№	Посада	Кількість	Коментар
1	**2**	**3**	**4**
1	Менеджер проєкту	1	Управління командою, контроль термінів та бюджету
2	Бізнес-аналітик	1	Збір та аналіз вимог, документація
3	UI/UX дизайнер	1	Розробка інтерфейсів, прототипування
4	Розробники	4	Full-stack команда для frontend і backend
5	Тестувальники	2	Мануальне та автоматизоване тестування
6	Юрист	1	Юридичний супровід, ліцензії, договори
7	Закупівельник	1	Закупівля техніки та ПЗ
8	IT Support	1	Підтримка інфраструктури
9	HR	2	Формування та супровід команди

2.4.2 Матеріальні ресурси

Матеріальні ресурси є критично важливим компонентом, що забезпечує стабільну та ефективну роботу проєктної команди. До них належать комп’ютерна техніка, сервери, ліцензії на програмне забезпечення, інструменти для командної роботи та тестування.

Усі ресурси закуповуються відповідно до графіку виконання робіт. Закупівельник відповідає за своєчасне постачання обладнання, а відділ технічного обслуговування – за встановлення, налаштування й технічну підтримку.

Нижче наведено перелік ключових матеріальних ресурсів, необхідних

для реалізації проєкту (табл. 2.4).

Таблиця 2.4

Перелік матеріальних ресурсів

№	Найменування	Кількість	Призначення
1	**2**	**3**	**4**
1	Ноутбуки для членів команди	10	Робочі місця розробників, аналітиків, QA
2	Сервери (фізичні або віртуальні)	2	Хостинг бази даних, API, вебінтерфейсу
3	Ліцензії на IDE (PHPStorm, VSCode)	4	Розробка та налагодження програмного коду
4	Ліцензія Figma	1	Дизайн інтерфейсів
5	GitHub (Pro або Team)	1	Сховище коду, CI/CD
6	Jira/Notion	1	Керування задачами, документація
7	Хмарні сервіси (AWS або DigitalOcean)	1	Хостинг тестових і продуктивних середовищ
8	Тестові мобільні пристрої	2	Тестування інтерфейсу під Android та iOS
9	Засоби відеозв’язку	1	Проведення щоденних мітингів та ретро

2.4.3 Планування використання ресурсів у часі

Ефективне планування використання ресурсів у часі є критичним компонентом управління IT-проєктом. Воно дозволяє забезпечити збалансоване навантаження на команду, оптимізувати використання матеріальних засобів та уникнути ризиків простою або перевантаження окремих учасників. У рамках цього проєкту ресурсне планування реалізовано з використанням середовища ProjectLibre, яке надає широкі можливості для деталізації графіку роботи кожного ресурсу.

У програмі створено повний перелік ресурсів, включаючи як людські (проєктна команда, фахівці супровідних відділів), так і матеріальні

(обладнання, програмне забезпечення, ліцензії, інфраструктура). Ці дані структуровано у таблиці з основними характеристиками, такими як тип ресурсу, ініціали, групування, стандартна ставка, максимальне завантаження та календар використання. Повна візуалізація наведена на рис. 2.7, який демонструє приклад заповненого списку ресурсів.

	Назва	Тип	Од.вим. матеріалів	Ініціалы	Групувати	Максимальне використ...	Стандартна ставка	Нараховувати	Основний календар
1	Менеджер проєкту	Робота		PR1		100%	428 грн./година	Пропорційно	П'ятиденка
2	Бізнес-аналітик	Робота		PR2		100%	357 грн./година	Пропорційно	П'ятиденка
3	UI/UX дизайнер	Робота		PR3		100%	285 грн./година	Пропорційно	П'ятиденка
4	Розробник Full Stack Senior	Робота		PR4.1		100%	2141 грн./година	Пропорційно	П'ятиденка
5	Розробник Full Stack Middle	Робота		PR4.2		100%	1200 грн./година	Пропорційно	П'ятиденка
6	Розробник Middle Web Dev	Робота		PR4.3		100%	857 грн./година	Пропорційно	П'ятиденка
7	Розробник Middle Web Dev	Робота		PR4.4		100%	857 грн./година	Пропорційно	П'ятиденка
8	Тестувальники	Робота		PR5		100%	214 грн./година	Пропорційно	П'ятиденка
9	Юрист	Робота		PR6		100%	178 грн./година	Пропорційно	П'ятиденка
10	Закупівельник	Робота		PR7		100%	200 грн./година	Пропорційно	П'ятиденка
11	IT Support	Робота		PR8		100%	171 грн./година	Пропорційно	П'ятиденка
12	HR (People Partner, Recruiter)	Робота		PR9		100%	221 грн./година	Пропорційно	П'ятиденка
13	Ноутбуки для членів команди	Матеріал		MR1	Робочі місця розробн...		500000 грн.	Пропорційно	
14	Сервери (фізичні або віртуальні)	Матеріал		MR2	Хостинг бази даних, ...		5000 грн.	Пропорційно	
15	Ліцензії на IDE (PHPStorm, VSCod	Матеріал		MR3	Розробка та налагод...		2000 грн.	Пропорційно	
16	Ліцензія Figma	Матеріал		MR4	Дизайн інтерфейсів		1200 грн.	Пропорційно	
17	GitHub (Pro або Team)	Матеріал		MR5	Сховище коду, CI/CD		800 грн.	Пропорційно	
18	Jira/Notion	Матеріал		MR6	Керування задачами, ...		800 грн.	Пропорційно	
19	Хмарні сервіси (AWS або DigitalC	Матеріал		MR7	Хостинг тестових і пр...		3000 грн.	Пропорційно	
20	Тестові мобільні пристрої	Матеріал		MR8	Тестування інтерфейс...		4500 грн.	Пропорційно	
21	Засоби відеозв'язку	Матеріал		MR9	Проведення щоденни...		6000 грн.	Пропорційно	

Рис. 2.7. Фрагмент таблиці ресурсів у ProjectLibre

Навантаження кожного ресурсу прив’язане до конкретних завдань з календарного плану (табл. 2.2), із точним зазначенням обсягу часу, що витрачається на виконання. На рис. 2.8 зображено фрагмент завантаження ресурсів у динаміці – наприклад, для Менеджера проєкту та Бізнес-аналітика – із денним розподілом години праці по задачах. Це дозволяє легко оцінити ефективність використання людських ресурсів на різних етапах реалізації.

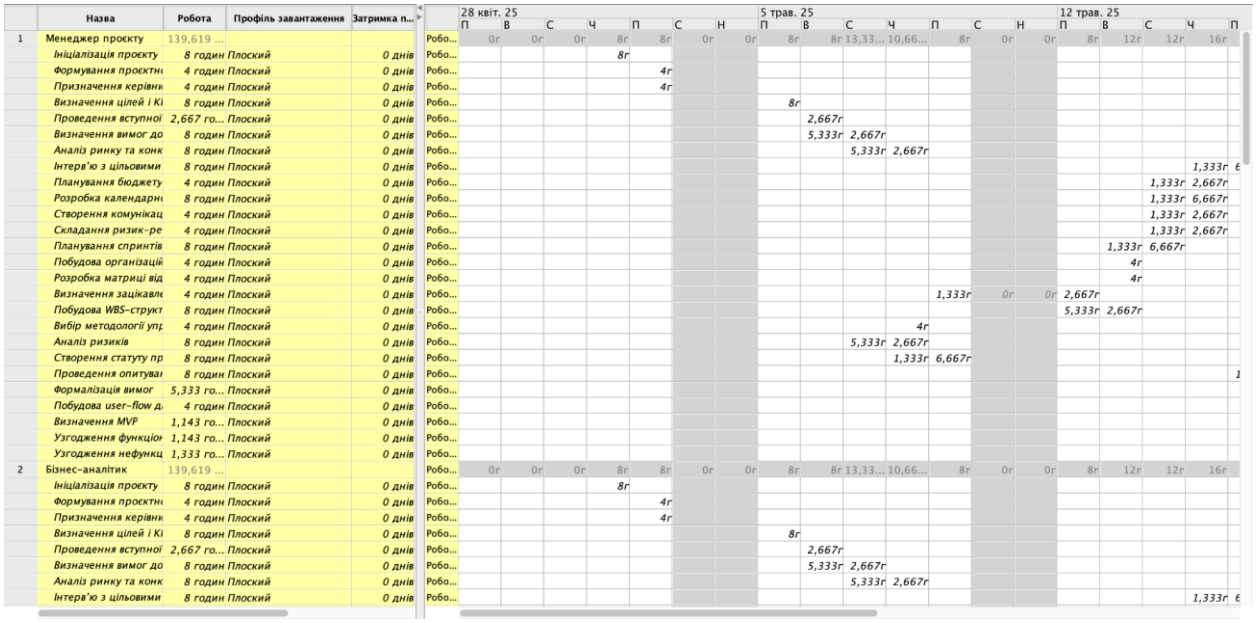


Рис. 2.8. Приклад завантаження ресурсів у часі (ProjectLibre)

Такий підхід забезпечує прозорість планування, своєчасне виявлення конфліктів у завантаженні персоналу та дає змогу гнучко реагувати на зміни у розкладі. Водночас, прив’язка витрат до графіка робіт дозволяє вести оперативний контроль за використанням бюджету, адаптувати витрати до реального перебігу подій, а також формувати фінансову звітність безпосередньо з плану проєкту.

Таким чином, використання ProjectLibre як основного інструмента для управління ресурсами дозволяє інтегрувати часові, кадрові та матеріальні аспекти проєкту в єдину систему, що підвищує загальну ефективність управління реалізацією Headless CMS з глибокою кастомізацією.

2.5 Формалізація задачі та математичне моделювання

Для створення ефективної системи керування сайтами на основі підходу Headless CMS необхідно чітко визначити ключові параметри та змінні, що впливають на її роботу. Математичне моделювання дозволяє формалізувати основні процеси системи – зокрема зберігання, обробку та доставку контенту, забезпечення безпеки, взаємодію з користувачем і інтеграцію з зовнішніми сервісами.

Побудова математичної моделі дає змогу кількісно оцінити продуктивність системи, визначити вузькі місця та ефективно планувати ресурси. Це, у свою чергу, забезпечує гнучкість, стабільність та масштабованість рішення – критично важливі для задоволення потреб сучасного бізнесу та кінцевих користувачів.

2.5.1 Визначення основних змінних та параметрів

Для побудови математичної моделі системи необхідно виявити ключові змінні та параметри, що характеризують її функціонування. Ці змінні дозволяють описати поведінку користувачів, навантаження на систему, а також ефективність її роботи в умовах реального використання. Далі наведено

перелік основних параметрів, які будуть використані для подальшої формалізації та аналізу.

Позначимо наступні основні змінні та параметри системи:

- U – кількість користувачів системи.
- C – кількість запитів до контенту за одиницю часу.
- S – розмір збереженого контенту (в байтах).
- R – кількість запитів, оброблених системою за одиницю часу.
- T – час відповіді системи на запит.
- λ – інтенсивність запитів до системи.
- μ – інтенсивність обробки запитів системою.
- B – пропускна здатність каналу зв'язку.

2.5.2 Математичне моделювання процесу обробки запитів

Одним із ключових аспектів формалізації системи є моделювання процесу обробки запитів користувачів. Для цього доцільно використовувати апарат теорії масового обслуговування, що дозволяє оцінити ефективність функціонування системи за умов навантаження.

Припустимо, що надходження запитів до системи підкоряється пуассонівському розподілу з інтенсивністю λ, що є характерним для випадкових подій у часі. У свою чергу, час обробки запиту системою можна змоделювати експоненціальним розподілом із параметром μ, який описує середню продуктивність системи.

Такий підхід дозволяє оцінити ймовірність виникнення черг, час очікування відповіді та загальну пропускну здатність, що критично важливо для побудови масштабованої та стабільної Headless CMS.

2.5.3 Система обробки запитів

Процес обробки запитів у системі може бути змодельований за допомогою класичних моделей черг, зокрема систем типу $M/M/1$ або $M/M/n$. Позначення означають наступне:

- Перша M вказує, що інтервали між надходженням запитів розподілені за експоненційним законом, тобто надходження моделюється пуассонівським процесом.
- Друга M вказує, що час обробки також має експоненційний розподіл.
- Число 1 або n означає кількість серверів у системі: $M/M/1$ – одна черга до одного сервера, $M/M/n$ – одна черга до n серверів.

Для моделі $M/M/1$ можна використовувати наступні основні співвідношення:

- Середня кількість запитів у системі:

$$[L = \frac{\lambda}{\mu - \lambda}] \tag{2.1}$$

- Середній час перебування запиту в системі:

$$[W = \frac{1}{\mu - \lambda}] \tag{2.2}$$

Для моделі $M/M/n$ (модель з n серверами) ймовірність того, що всі n серверів зайняті, можна обчислити за формулою Ерланга:

$$P_0 = \left(\sum_{k=0}^{n-1} \frac{(\lambda/\mu)^k}{k!} + \frac{(\lambda/\mu)^n}{n!} \cdot \frac{1}{1 - \lambda/(n\mu)} \right)^{-1} \tag{2.3}$$

Середня кількість запитів у системі для $M/M/n$ моделі:

$$L = \frac{(\lambda/\mu)^n \cdot \lambda/(n\mu)}{n! \cdot (1 - \lambda/(n\mu))^2} \cdot P_0 + \frac{\lambda}{\mu} \tag{2.4}$$

2.5.4 Визначення вимог до пропускної здатності

Для забезпечення стабільної та швидкої доставки цифрового контенту користувачам системи необхідно правильно оцінити пропускну здатність

каналу зв’язку. Вона має бути достатньою для передачі всього обсягу запитуваного контенту в межах допустимого часу відгуку.

Це означає, що пропускна здатність B повинна відповідати нерівності:

$$B \geq \frac{S \cdot C}{T} \tag{2.5}$$

де:

- S – розмір одиниці контенту (в байтах),
- C – кількість запитів за одиницю часу,
- T – допустимий час відповіді системи.

Ця умова гарантує, що система здатна ефективно обробляти всі запити, не виходячи за межі встановленого SLA (Service Level Agreement). Недотримання цього критерію призводить до затримок, зниження продуктивності та погіршення користувацького досвіду, що є критично важливим у сучасних високонавантажених системах управління контентом.

2.5.5 Безпека даних

Забезпечення належного рівня безпеки є ключовим аспектом під час розробки сучасної системи керування контентом. Для цього необхідно впровадити ефективні механізми шифрування, дешифрування та автентифікації, які не лише гарантують конфіденційність, цілісність і доступність даних, але й не створюють суттєвих затримок при обробці запитів.

Позначимо:

- E– швидкість шифрування даних;
- D – швидкість дешифрування даних.

Щоб забезпечити безперервну обробку запитів і не перевищити допустиме навантаження на систему, ці параметри повинні відповідати наступній умові:

$$E, D \geq \mu - \lambda \tag{2.6}$$

де λ – інтенсивність надходження запитів, а μ – інтенсивність їх обробки.

Таким чином, система повинна забезпечити таку продуктивність процесів шифрування та дешифрування, щоб вони не ставали "вузьким місцем" у загальній архітектурі та не впливали негативно на швидкість і стабільність роботи. Це критично важливо для дотримання сучасних стандартів кібербезпеки та захисту даних кінцевих користувачів.

2.5.6 Персоналізація контенту

Для забезпечення персоналізованого користувацького досвіду система повинна інтегруватися з аналітичними сервісами, які збирають, обробляють та інтерпретують дані про поведінку користувачів у режимі реального часу. На основі цієї інформації формуються індивідуальні рекомендації, адаптивний контент і покращені сценарії взаємодії з користувачем.

Позначимо:

- A – інтенсивність запитів до аналітичних сервісів;
- P – швидкість обробки даних аналітичними сервісами.

Щоб система могла своєчасно формувати персоналізовані відповіді, необхідно, щоб продуктивність аналітичних сервісів задовольняла наступну умову:

$$P \geq A \cdot T \tag{2.7}$$

де T – максимально допустимий час відповіді системи.

Це означає, що аналітичні модулі повинні бути достатньо потужними, аби обробляти усі вхідні запити в межах заданого часового вікна, забезпечуючи безперебійну генерацію персоналізованого контенту. Такий підхід є критично важливим для збереження високої якості взаємодії з користувачами та задоволення їхніх очікувань.

2.5.7 Оптимізація роботи системи

Для забезпечення ефективного функціонування системи управління контентом необхідно сформулювати чіткі цілі оптимізації, які спрямовані на досягнення максимальної продуктивності, швидкодії та безпеки. У процесі

проєктування системи важливо досягти балансу між навантаженням, пропускною здатністю, швидкістю відповіді та рівнем захисту даних.

Основними напрямами оптимізації є:

- Максимізація пропускної здатності системи – $\max B$ забезпечує здатність системи обробляти великий обсяг запитів без затримок.
- Мінімізація часу відповіді на запити – $\min T$ визначає швидкість реакції системи на дії користувачів, що критично для забезпечення якісного досвіду взаємодії.
- Підтримка належного рівня безпеки обробки даних – $\max E, D$ передбачає забезпечення високих швидкостей шифрування та дешифрування інформації без втрати продуктивності.

Досягнення цих оптимізаційних показників дозволяє системі відповідати вимогам сучасного ринку, забезпечуючи стабільність, масштабованість та безпечне керування цифровим контентом.

РОЗДІЛ 3. ТЕХНІЧНЕ ПРОЄКТУВАННЯ, РОЗРОБКА ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Розробка архітектури програмного забезпечення Headless CMS

Архітектура програмного забезпечення відіграє ключову роль у створенні гнучкої, надійної та масштабованої Headless CMS, здатної забезпечити стабільну роботу навіть за умов високого навантаження, інтеграції з численними зовнішніми платформами та частих оновлень контенту. Саме правильно побудована архітектура визначає, наскільки система зможе відповідати вимогам бізнесу, гнучко масштабуватись, ефективно розгортатись у різних середовищах і при цьому залишатися безпечною та стабільною.

З огляду на сучасні тенденції у сфері розробки CMS, для реалізації цього проєкту було обрано мікросервісну архітектуру, яка дозволяє розділити функціональність системи на автономні логічні блоки. Кожен мікросервіс відповідає за чітко визначений набір функцій, взаємодіє з іншими сервісами через API та може масштабуватись незалежно. Це не лише спрощує підтримку й модернізацію системи, а й дозволяє паралелізувати процеси розробки різними командами, зменшуючи загальний час реалізації.

В основі архітектурного підходу – принцип поділу системи на три рівні: інтерфейс користувача (frontend), сервіс взаємодії (API Gateway) та внутрішні бізнес-компоненти (мікросервіси + база даних). Така багаторівнева структура дозволяє розподіляти навантаження, покращити керованість коду та забезпечити модульність розгортання. Система підтримує взаємодію не лише з адміністративною панеллю, але й з будь-якими зовнішніми застосунками – сайтами, мобільними клієнтами, зовнішніми CRM або eCommerce-платформами.

Загальна логіка побудови програмної архітектури зображена на рис. 3.1, де відображено взаємозв’язки між модулями, рівнями відповідальності та зовнішніми точками взаємодії. Це дозволяє візуально оцінити глибину

структуризації проєкту та логіку організації даних потоків у системі.

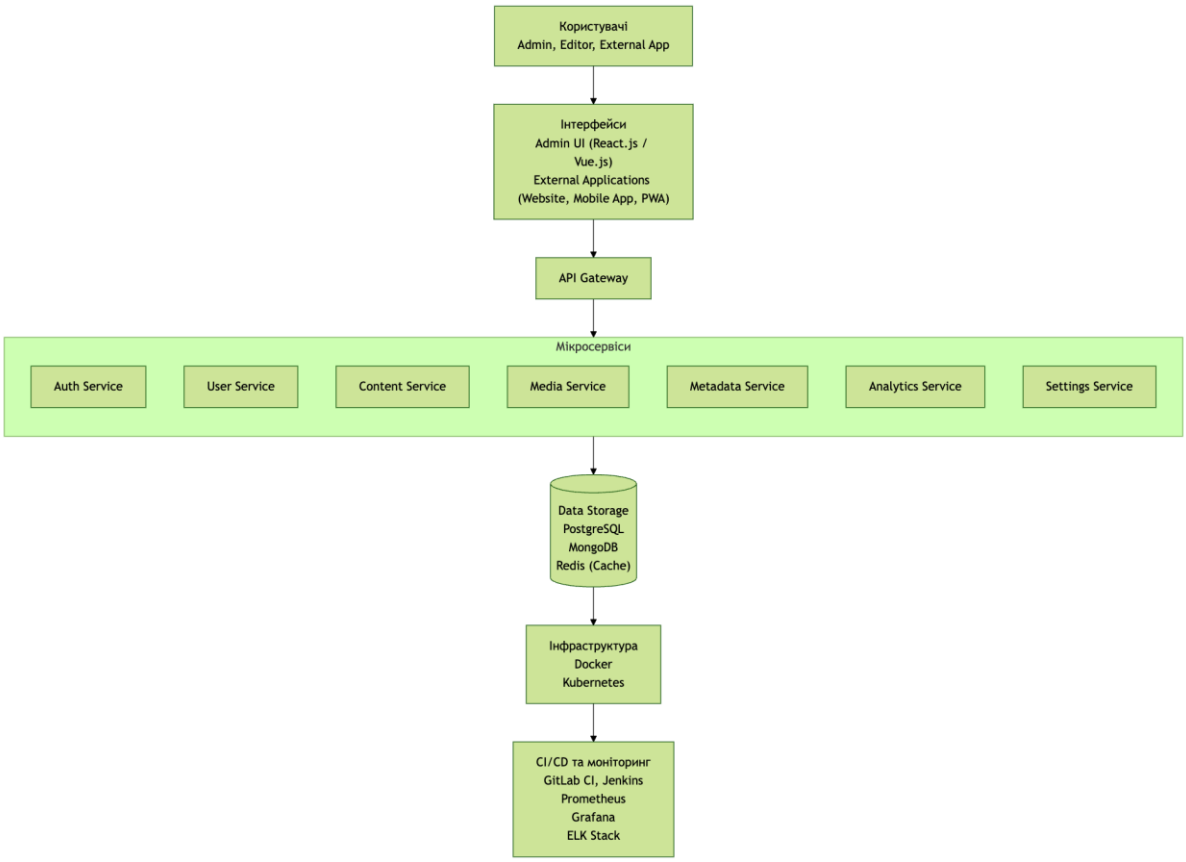


Рис. 3.1. Загальна логіка архітектури Headless CMS

У межах цієї архітектури frontend частина реалізується на JavaScript із використанням React або Vue, що забезпечує створення динамічного, адаптивного та зручного інтерфейсу для адміністраторів і контент-менеджерів. Серверна логіка побудована на Node.js з Express.js, що дозволяє швидко обробляти запити, керувати сесіями, ролями та політиками доступу.

Всі зовнішні запити надходять через API Gateway, який виступає єдиною точкою входу. Саме тут відбувається маршрутизація до відповідних мікросервісів, первинна авторизація, кешування та базова аналітика. Далі запити розподіляються до спеціалізованих мікросервісів: User Service – для обробки користувачів, Content Service – для CRUD-операцій з контентом, Media Service – для роботи з медіафайлами, Metadata Service – для SEO, локалізації та структурної інформації, а також аналітичні й сервісні модулі для технічного адміністрування.

В якості сховищ даних використовуються дві бази – реляційна PostgreSQL для роботи з критично важливими структурованими об’єктами (користувачі, ролі, публікації) та документоорієнтована MongoDB для неструктурованих або динамічних блоків, включаючи JSON-структури, мультимедіа та кастомні шаблони. Для підвищення продуктивності використовується Redis, що забезпечує кешування часто запитуваних даних, сесій і токенів.

Уся система контейнеризована за допомогою Docker і управляється через Kubernetes, що дозволяє динамічно масштабувати окремі сервіси, оперативно розгортати нові версії та забезпечувати безперервну доступність. CI/CD-процеси автоматизовані за допомогою GitLab CI або Jenkins, а моніторинг виконання і логування реалізовано через Prometheus, Grafana та стек ELK (Elasticsearch, Logstash, Kibana).

Такий архітектурний підхід формує надійну основу для подальшої розробки й масштабування Headless CMS, відкриваючи можливості як для впровадження у малих проектах, так і для інтеграції у великі організації з розгалуженою контентною інфраструктурою.

3.2 Проєктування бази даних для Headless CMS

3.2.1 Розробка концептуальної моделі бази даних

Концептуальне проєктування бази даних є першим і критично важливим етапом створення стабільної й масштабованої системи зберігання інформації для Headless CMS. На цьому рівні моделювання визначаються ключові сутності, їх атрибути та взаємозв’язки без урахування специфіки реалізації в конкретній СУБД. Побудована модель має відображати логіку предметної області – керування контентом, користувачами, метаданими, медіаресурсами та зовнішньою взаємодією через API.

При розробці концептуальної моделі враховувалися специфіка Headless-архітектури, багатоканальний характер доставки контенту, а також необхідність забезпечення ефективного пошуку, масштабованості й

безперервного відстеження змін у системі. Основні сутності, які формують ядро моделі, наведено нижче:

Користувачі (Users) – зберігають ключову інформацію про зареєстрованих осіб у системі: унікальні ідентифікатори, логіни, email-адреси, хешовані паролі, дату реєстрації та останній вхід. Це дозволяє організовувати контроль доступу, автентифікацію та відстеження активності.

Контентні елементи (ContentItems) – основна сутність, що містить структуру контенту, включаючи заголовки, текст або HTML, тип матеріалу, автора, дату створення та статус (чернетка, опубліковано тощо). Взаємозв'язок з користувачами дозволяє ідентифікувати авторів.

Метадані (Metadata) – реалізовані як гнучка структура "ключ-значення", прив'язана до конкретного контенту. Модель дозволяє зберігати SEO-дані, ключові слова, опис, категорії та інші атрибути, що можуть змінюватись або розширюватись без зміни схеми основної таблиці.

Медіаресурси (MediaResources) – окрема сутність, яка містить інформацію про завантажені медіафайли: назву, тип, шлях до збереження, дату додавання та зв'язок із контентними елементами. Такий підхід дає змогу централізовано керувати всіма мультимедійними об'єктами.

API-запити (APIRequests) – сутність, що зберігає технічну інформацію про взаємодію із зовнішніми клієнтами через API. Включає дату запиту, точку доступу, статус відповіді, час відповіді, а також зв'язок з користувачем, який ініціював запит. Це дозволяє відстежувати продуктивність, виявляти помилки та оптимізувати API.

Для кожної сутності визначено базовий набір атрибутів, що наведено у таблицях нижче:

Users:

- UserID – Унікальний ідентифікатор
- UserName – Ім'я користувача
- Email – Електронна пошта
- PasswordHash – Хешований пароль

- RegistrationDate – Дата реєстрації
- LastLogin – Дата останнього входу

ContentItems:

- ContentID – Унікальний ідентифікатор контенту
- Title – Заголовок
- Body – Текст або HTML
- ContentType – Тип контенту (стаття, блог тощо)
- AuthorID – Зв’язок з користувачем
- CreatedAt / PublishedAt – Час створення та публікації
- Status – Статус контенту

Metadata:

- MetadataID – Унікальний ідентифікатор
- ContentID – Зв’язок із ContentItems
- Key – Назва метаатрибуту
- Value – Значення

MediaResources:

- MediaID – Ідентифікатор
- ContentID – Зв’язок із контентом
- FileName – Назва файлу
- FilePath – Шлях
- FileType – Тип файлу
- UploadedDate – Дата завантаження

APIRequests:

- RequestID – Ідентифікатор запиту
- UserID – Користувач
- Endpoint – Точка доступу
- RequestDate – Час
- StatusCode – Код відповіді
- ResponseTime – Час відповіді в мс

Концептуальна модель охоплює ключові процеси CMS: керування користувачами, створення та організація контенту, інтеграція з мультимедіа, обробка метаінформації та взаємодія через API. Такий підхід дозволяє забезпечити узгодженість даних, масштабованість і гнучкість системи у подальшій реалізації.

Структурна візуалізація взаємозв'язків між сутностями представлена на рис. 3.2, де показано повну концептуальну модель бази даних для Headless CMS.

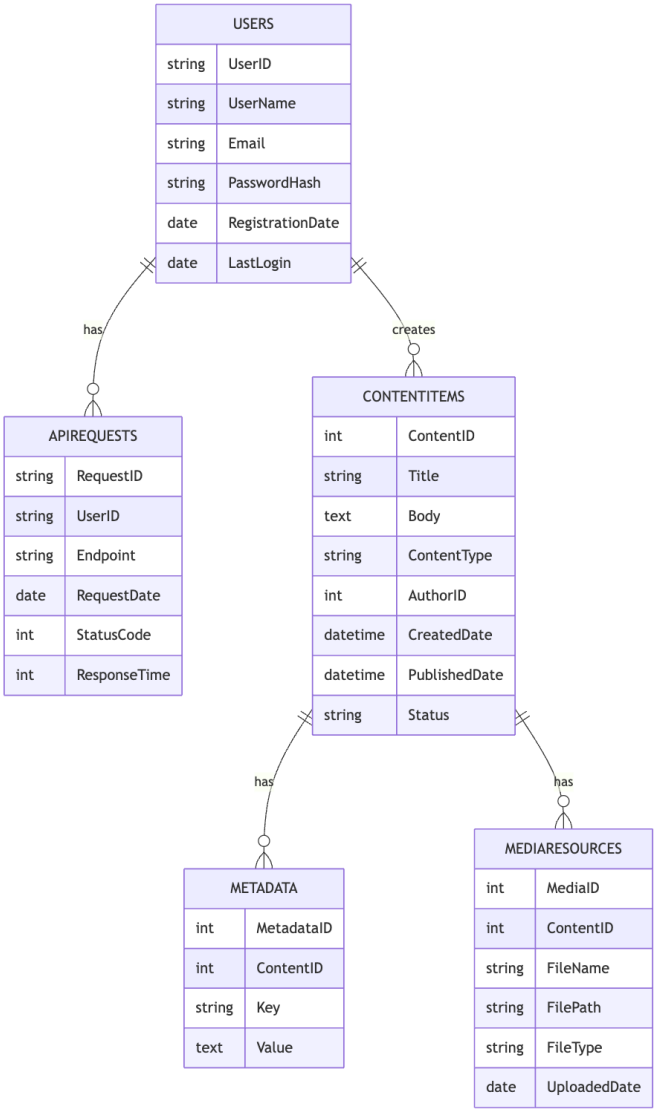


Рис. 3.2. Концептуальна модель бази даних

3.2.2 Розробка логічної моделі бази даних

Логічне проєктування бази даних є ключовим етапом у процесі побудови інформаційної структури системи. На цьому рівні відбувається формалізація структури даних без урахування особливостей реалізації у конкретній СУБД. Метою логічної моделі є визначення того, які саме таблиці буде створено, які зв'язки між ними будуть встановлені, а також які обмеження (constraints) діятимуть у базі даних.

Логічна модель базується на попередньо сформованій концептуальній моделі, де кожна сутність трансформується у таблицю, а її атрибути – у відповідні поля. Усі зв'язки між таблицями реалізуються за допомогою зовнішніх ключів, що забезпечують узгодженість даних і правильну логіку їх обробки.

Для кожної таблиці визначаються:

- первинний ключ (Primary Key) – для однозначної ідентифікації кожного запису;
- зовнішні ключі (Foreign Keys) – для встановлення зв'язків між таблицями;
- обмеження цілісності – наприклад, NOT NULL, UNIQUE, або ON DELETE CASCADE.

Важливою частиною логічного проєктування є нормалізація – процес структурування таблиць таким чином, щоб мінімізувати надлишковість даних і забезпечити їх цілісність. У межах даної системи реалізовано принаймні три перші нормальні форми (1NF–3NF), що дозволяє уникати повторення атрибутів, логічних аномалій при оновленнях і забезпечити ефективну структуру для масштабування.

Нормалізація особливо актуальна в контексті Headless CMS, де дані можуть бути часто змінюваними, гнучко структуруватися, інтегруватися з різними джерелами та одночасно використовуватися в декількох каналах. Наприклад, відокремлення медіа-даних, SEO-метаданих та користувацьких атрибутів у окремі таблиці дозволяє більш ефективно будувати запити,

забезпечити розширюваність системи та гнучкість у змінах структури даних без потреби в глобальній перебудові бази.

Логічна модель забезпечує повний опис майбутньої бази даних на абстрактному рівні – із чіткою структурою таблиць, типами полів, ключами та зв’язками. Вона також слугує базою для створення фізичної моделі, яка вже враховує специфіку конкретної системи управління базами даних, обмеження середовища розгортання та оптимізації під продуктивність.

На рис. 3.3 представлено логічну модель бази даних для Headless CMS, сформовану на основі концептуального проєкту. Вона включає таблиці користувачів, контенту, метаданих, медіаресурсів, API-запитів, а також типи зв’язків між ними.

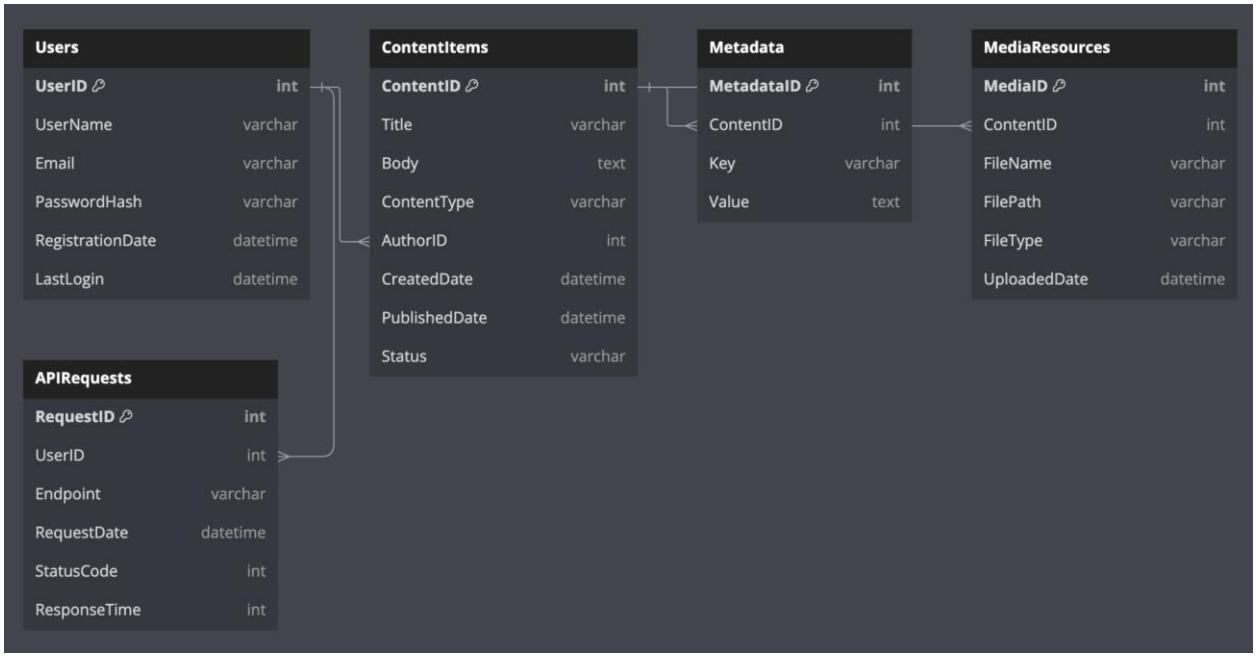


Рис. 3.3. Логічна модель бази даних

3.2.3 Розробка фізичної моделі бази даних

Фізичне проєктування бази даних є фінальним етапом моделювання, у межах якого логічна структура перетворюється у реальні об’єкти СУБД. У межах даного проєкту як основну систему управління базами даних було обрано MySQL – популярну платформу з відкритим кодом, що поєднує високу продуктивність, надійність і широку підтримку сучасних функцій безпеки.

Вибір MySQL зумовлений її стабільною роботою з великими обсягами даних, оптимізацією під складні запити, розгалуженою спільнотою та підтримкою інструментів для резервного копіювання, реплікації та масштабування. Це дає змогу збудувати надійне, швидке й масштабоване сховище для Headless CMS, яке легко адаптується до змін структури або розширення системи.

На основі попередньо розробленої концептуальної та логічної моделі було створено відповідні таблиці в СУБД MySQL. Для кожного атрибута було підібрано оптимальний тип даних (наприклад, INT, VARCHAR, TEXT, DATETIME, JSON), що забезпечує цілісність, точність і мінімальне використання пам’яті. Первинні ключі (PRIMARY KEY) гарантують унікальність записів, а зовнішні ключі (FOREIGN KEY) реалізують логічні зв’язки між таблицями, відповідно до сутностей, визначених на концептуальному рівні.

Особливу увагу було приділено безпеці та продуктивності. Для таблиць, що обробляють особисту інформацію або облікові дані, реалізовано обмеження доступу та хешування критичних полів (наприклад, PasswordHash). Додатково, у таблицях з великим обсягом даних та високою частотою доступу було здійснено індексацію ключових полів – таких як UserID, ContentID, Email, Endpoint. Це дозволило зменшити час виконання запитів та покращити загальну швидкодію системи.

Паралельно було виконано нормалізацію структури бази даних, що дозволила усунути дублювання даних та забезпечити логічну незалежність таблиць. Застосовано принципи першої, другої та третьої нормальних форм (1NF–3NF), що дає змогу зберігати інформацію у компактному й структурованому вигляді. Такий підхід значно підвищив гнучкість системи, спростив обслуговування та подальшу модернізацію.

Усі SQL-запити для створення таблиць були реалізовані вручну – із чітким контролем за типами, ключами, індексами та правилами зв’язків. Наприклад, SQL-запит на створення таблиці Users наведено на рис. 3.4, а

графічне відображення фізичної моделі цієї сутності – на рис. 3.5. Структури інших таблиць і відповідні SQL-інструкції представлені в Додатку А.

```sql
CREATE TABLE Users (
    UserID INT AUTO_INCREMENT PRIMARY KEY,
    UserName VARCHAR(255) NOT NULL,
    Email VARCHAR(255) NOT NULL,
    PasswordHash VARCHAR(255) NOT NULL,
    RegistrationDate DATETIME NOT NULL,
    LastLogin DATETIME
);
```

Рис. 3.4. SQL запит для створення сутності Users

Рис. 3.5. Сутність Users у СУБД phpMyAdmin

Таким чином, розроблена фізична модель бази даних стала надійною основою для реалізації CMS-системи, що відповідає принципам Headless-архітектури. Вона забезпечує стабільне зберігання, швидкий доступ і ефективне опрацювання контенту, а також дозволяє масштабувати проєкт без суттєвих змін у структурі даних.

3.3 Розробка ключових алгоритмів функціонування CMS

Функціонування Headless CMS неможливе без чітко реалізованих алгоритмів для обробки основних дій з контентом. У межах даного проєкту було реалізовано низку ключових CRUD-операцій (створення, зчитування, оновлення та видалення контенту), які забезпечують базову взаємодію користувача з системою. Ці операції виконуються через API-запити, які обробляються сервером та інтегруються з базою даних MySQL.

Алгоритми були реалізовані з урахуванням принципів безпеки, масштабованості та розширюваності. Кожна дія супроводжується перевіркою вхідних даних, обробкою запиту та формуванням відповідного результату. Це забезпечує надійність і передбачуваність системи під час роботи з контентом на різних етапах його життєвого циклу.

Огляд кожного з ключових алгоритмів наведено у підпунктах нижче, а приклади реалізації представлені у вигляді фрагментів коду та схем взаємодії.

3.3.1 Алгоритм для створення та зберігання контенту

Процес створення нового контенту в Headless CMS починається з надсилання даних користувачем через інтерфейс адміністративної панелі. Дані, які надходять від користувача, містять обов’язкові поля: заголовок (title), основний вміст (body), тип контенту (content_type), ідентифікатор автора (author_id), дата публікації (published_date) та статус (status).

Першим етапом є перевірка наявності всіх необхідних полів за допомогою функції validateData($data). Якщо перевірка проходить успішно, викликається функція createContent($data), яка ініціалізує підключення до бази даних MySQL через об'єкт PDO.

Далі система формує SQL-запит на додавання нового запису в таблицю ContentItems, вказуючи перелік стовпців і відповідні значення. Окремо використовується функція NOW() для автоматичної фіксації дати створення запису.

Якщо запит виконується успішно, система повертає у відповідь статус "success" разом з ID створеного запису. У випадку помилки або некоректних вхідних даних повертається повідомлення про невдачу із відповідним статусом "error".

Цей алгоритм гарантує:

- валідність отриманих даних перед збереженням;
- безпечне виконання SQL-запиту через підготовлений вираз (prepare);
- коректне логування результату операції.

З базовим кодом реалізації цього алгоритму можна ознайомитися на рис. 3.6.

```php
function validateData($data) {
    return isset($data['title'], $data['body'], $data['content_type'], $data['author_id'], $data['published_date'], $data['status']);
}

function createContent($data) {
    if (validateData($data)) {
        $db = new PDO('
            mysql:host=localhost;
            dbname=headless_cms',
            'username',
            'password
        ');
        $stmt = $db->prepare("INSERT INTO ContentItems
            (Title, Body, ContentType, AuthorID, CreatedDate, PublishedDate, Status)
            VALUES (?, ?, ?, ?, NOW(), ?, ?)");
        $stmt->execute([$data['title'], $data['body'], $data['content_type'], $data['author_id'], $data['published_date'], $data['status']]);
        return ["status" => "success", "content_id" => $db->lastInsertId()];
    } else {
        return ["status" => "error", "message" => "Invalid data"];
    }
}
```

Рис. 3.13. Базовий код алгоритму для створення та зберігання контенту

3.3.2 Алгоритм для отримання контенту

Алгоритм отримання контенту реалізує гнучкий механізм вибірки даних із бази з урахуванням заданих користувачем параметрів фільтрації. Клієнт надсилає запит до API з можливими параметрами, наприклад, author_id або status. Ці параметри використовуються для формування SQL-запиту.

Спочатку викликається функція buildQuery($filters), яка будує динамічний SQL-запит на основі переданих фільтрів. Наприклад, якщо

користувач вказав ID автора або статус публікації, ці умови додаються до запиту в секції WHERE.

Далі цей запит виконується функцією getContent($filters), яка ініціалізує підключення до бази даних через PDO, виконує сформований запит і отримує масив результатів. Для повернення даних у зручному для клієнта форматі використовується функція formatResponse($results), яка конвертує результат у JSON.

Цей підхід дозволяє реалізувати масштабовану та адаптивну систему вибірки контенту, забезпечуючи швидку обробку запитів та персоналізоване відображення даних.

З базовим кодом алгоритму можна ознайомитися у додатку Б.

3.3.3 Алгоритм для оновлення контенту

Оновлення контенту є важливою функцією для забезпечення актуальності та точності інформації в системі. Алгоритм починається з ініціалізації запиту користувачем, який хоче змінити наявний запис через інтерфейс CMS. Дані, надіслані користувачем, передаються у форматі JSON до API, де відбувається перевірка їхньої коректності за допомогою функції validateData($data).

Після валідації система встановлює з’єднання з базою даних через PDO. Перед оновленням виконується перевірка існування вказаного контенту у таблиці ContentItems за допомогою функції contentExists($content_id, $db). Це дозволяє уникнути оновлення неіснуючих записів і гарантує цілісність бази даних.

Якщо контент існує, виконується підготовлений SQL-запит UPDATE, у якому змінюються основні поля запису: заголовок (Title), вміст (Body), тип контенту (ContentType), дата публікації (PublishedDate) та статус (Status). Запит виконується з використанням параметрів, що унеможливлює SQL-ін’єкції.

У випадку успішного оновлення користувачу повертається відповідь з повідомленням про успішну операцію. Якщо контенту з вказаним ідентифікатором не знайдено або передані дані некоректні – система повертає відповідне повідомлення про помилку.

Цей алгоритм забезпечує безпечне та контрольоване оновлення контенту в системі, дозволяючи підтримувати його актуальність та відповідність вимогам користувачів.

З базовим кодом алгоритму можна ознайомитися у додатку Б.

3.3.4 Алгоритм для видалення контенту

Видалення контенту є невіддільною частиною управління життєвим циклом даних у CMS. Алгоритм реалізує контрольоване вилучення контенту з бази даних та гарантує, що видалення відбувається лише у разі наявності вказаного запису.

Процес розпочинається з ініціалізації запиту користувачем через клієнтський інтерфейс. У запиті передається ідентифікатор контенту, що підлягає видаленню. Далі запит надходить на API сервер, де відбувається підключення до бази даних за допомогою PDO.

Перед фактичним видаленням виконується перевірка наявності запису у таблиці ContentItems за допомогою функції contentExists($content_id, $db). Це дозволяє уникнути спроб видалення неіснуючих записів і підвищує надійність системи.

Якщо запис існує, виконується SQL-запит DELETE, який видаляє контент із бази за відповідним ContentID. Успішне виконання операції супроводжується поверненням статусу "success", інакше – повідомленням про помилку, наприклад "Content not found".

Застосування даного алгоритму дозволяє підтримувати актуальність, чистоту та структурну цілісність бази даних, запобігаючи накопиченню застарілої інформації.

З базовим кодом алгоритму можна ознайомитися у додатку Б.

3.4 Розробка інтерфейсу Headless CMS

Інтерфейс адміністративної панелі є одним з ключових елементів Headless CMS, оскільки саме через нього здійснюється повсякденна взаємодія користувачів із системою. Зручність, інтуїтивність і логічна структура інтерфейсу безпосередньо впливають на ефективність управління контентом, медіа, користувачами та налаштуваннями.

У межах проєкту було розроблено адаптивний інтерфейс основних сторінок системи, який поєднує сучасний мінімалістичний дизайн, високий рівень юзабіліті та підтримку багатофункціональності. Основна увага приділялась послідовності у компонуванні елементів, відповідності поведінки елементів очікуванням користувачів та швидкому доступу до основних дій.

Нижче описано основні сторінки адміністративної панелі, що були реалізовані у рамках цього проєкту.

Головна сторінка (Dashboard) є центральною точкою входу в систему. Вона надає користувачеві узагальнену інформацію про поточний стан CMS: статистику відвідувань, огляд останніх змін у контенті, сповіщення та блоки швидких дій. Такий інтерфейс дозволяє адміністраторам оперативно реагувати на зміни та швидко перемикатися до ключових розділів. Дизайн сторінки наведено на рис. 3.17.

Сторінка керування контентом охоплює повний життєвий цикл роботи з матеріалами: створення, редагування, публікацію, архівацію та видалення. Для зручності реалізовано фільтрацію, сортування та пошук за ключовими параметрами. Особлива увага приділена організації простору – навіть за великої кількості записів користувач може швидко орієнтуватися та знаходити потрібний елемент. Зовнішній вигляд сторінки представлено у Додатку В.

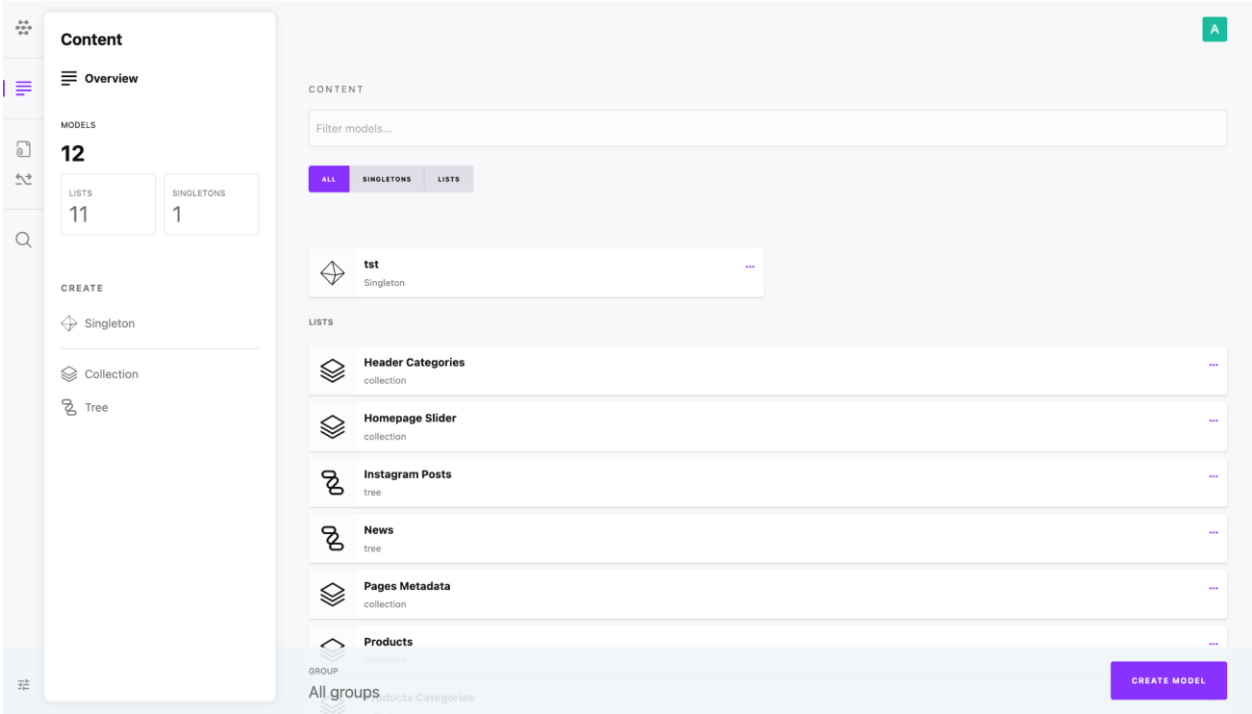


Рис. 3.17. Дизайн головної сторінки

Медіа-менеджер надає інструменти для роботи з файлами: завантаження, перегляд, категоризацію та вставлення у контент. Інтерфейс підтримує основні формати (зображення, відео, документи) та дозволяє здійснювати пошук за назвою, типом або датою завантаження. Це дозволяє створювати візуально багатий контент без зайвих складнощів. Інтерфейс сторінки також відображено у Додатку В.

Сторінка керування користувачами дає змогу створювати та редагувати облікові записи, керувати ролями й правами доступу. Це дозволяє підтримувати структурований контроль над тим, хто має доступ до тих чи інших функцій CMS. Надійна система керування користувачами критично важлива для забезпечення безпеки та внутрішньої дисципліни в адмініструванні. З дизайном сторінки можна ознайомитися у Додатку В.

Сторінка керування API відповідає за створення й налаштування API-ключів, обмеження доступу та контроль інтеграцій із зовнішніми сервісами. Передбачено налаштування за IP-адресами, терміном дії токенів і типами дозволених запитів. Така сторінка є необхідною складовою для побудови

гнучкої та масштабованої екосистеми. Інтерфейс сторінки також відображено у Додатку В.

Сторінка налаштувань системи є централізованим розділом, де адміністратор може змінювати глобальні параметри: безпеку, мови, режими доступу, інтеграцію з іншими сервісами та інше. Зовнішній вигляд сторінки наведено у Додатку В.

3.5 Контроль якості розробки та тестування системи

Якість програмного забезпечення є одним із ключових факторів успішної реалізації цифрових продуктів. У рамках розробки Headless CMS було приділено особливу увагу формуванню критеріїв якості, їх відповідності очікуванням користувачів і технічним вимогам. Контроль якості охоплював не лише тестування функціональності, але й глибоку аналітику потреб зацікавлених сторін, управління ризиками, планування витрат і забезпечення стабільності продукту в умовах реального використання.

Для забезпечення якісної реалізації системи було здійснено модульний поділ функціональності. Кожен модуль відповідає за окрему частину логіки Headless CMS, що дозволяє проводити незалежну оцінку якості, тестування та вдосконалення кожного компонента. Такий підхід підвищує керованість процесом розробки, а також спрощує майбутнє масштабування системи.

Нижче наведено опис ключових модулів продукту та визначено їхню пріоритетність згідно з важливістю для стабільного функціонування CMS:

1. Авторизація користувача. Відповідає за ідентифікацію та авторизацію користувачів. Забезпечує контроль доступу до функціональних частин системи.
2. Управління базами даних. Забезпечує зчитування, оновлення, видалення інформації, а також стабільну взаємодію з базами даних.
3. API для роботи з даними. Призначений для взаємодії з іншими системами та клієнтськими додатками. Відкриває стандартизовані

точки доступу до даних.

4. Інтерфейс користувача. Забезпечує зручне візуальне представлення функціоналу системи.
5. Документація. Містить інструкції з використання CMS для розробників та адміністраторів.
6. Налаштування сповіщень. Дозволяє конфігурувати сценарії отримання системних повідомлень.
7. Адаптація для мобільних пристроїв. Забезпечує зручне використання CMS на різних пристроях.
8. Зворотний зв’язок. Дозволяє користувачам надсилати відгуки, повідомлення про помилки або пропозиції щодо покращення.

Такий порядок пріоритетів дозволив сконцентрувати тестування і перевірку якості насамперед на критично важливих компонентах системи, що забезпечують безперебійне функціонування CMS у виробничому середовищі.

З метою забезпечення високої якості кожного окремого модуля Headless CMS було визначено вимоги до якості продукту, враховуючи очікування ключових зацікавлених сторін – кінцевих користувачів, адміністраторів, розробників і тестувальників.

Для кожного з ключових функціональних модулів системи були сформульовані чіткі, обґрунтовані вимоги до якості. Ці вимоги охоплюють широкий спектр характеристик, які критично важливі для ефективної роботи Headless CMS: продуктивність, стабільність, зручність використання, гнучкість, адаптивність до різних пристроїв, безпека зберігання та обміну даними, а також інтеграційна сумісність з іншими сервісами. Такий підхід дозволяє забезпечити не лише технічну надійність системи, але й позитивний досвід взаємодії для кінцевих користувачів та адміністраторів.

Для кожного модуля було запропоновано конкретні заходи, які необхідно реалізувати на етапах розробки та впровадження. Ці заходи охоплюють як технічні (наприклад, використання алгоритмів шифрування, резервне копіювання, балансування навантаження), так і UX-орієнтовані

(адаптація інтерфейсу, організація зворотного зв’язку тощо) рішення. Такий комплексний підхід дозволяє гарантувати досягнення бажаного рівня якості продукту в умовах обмежених ресурсів і часу. Зведену інформацію щодо вимог до якості та відповідних заходів подано у наведено у таблиці 3.1.

Таблиця 3.1

Вимоги до якості та заходи реалізації для модулів системи

Зацікавлені сторони	**Модуль продукту**	**Вимоги до якості продукту**	**Заходи необхідні для задоволення вимог**
1	**2**	**3**	**4**
Адміністратори	Авторизація користувача	- Безпека: Захист від несанкціонованого доступу. - Ефективність: Швидкий та надійний процес авторизації.	- Реалізація авторизації з сучасними алгоритмами шифрування паролів.
Розробники	Управління базами даних	- Надійність: Стійкість і відновлення БД. - Продуктивність: CRUD-операції виконуються швидко.	- Регулярне створення резервних копій. - Оптимізація запитів і індексація.
Кінцеві користувачі	Інтерфейс користувача	- Зручність: Легкість використання. - Сумісність: Підтримка різних пристроїв і браузерів.	- Проведення юзабіліті-тестів і адаптація під різні екрани.
Розробники, користувачі	Документація	- Зрозумілість: Легка та повна документація для розробників і користувачів.	- Створення зрозумілої й структурованої документації.

Продовження Таблиці 3.1

1	2	3	4
Адміністратори	Налаштування сповіщень	- Гнучкість: Можливість налаштування сповіщень для різних сценаріїв.	- Панель налаштувань для зміни параметрів сповіщень адміністратором.
Розробники	Адаптація для мобільних пристроїв	- Адаптивність: Зручність і ефективність інтерфейсу на мобільних пристроях.	- Технології адаптивного дизайну для зменшених екранів.
Користувачі, розробники	Зворотний зв'язок	- Ефективність: Швидке надсилання. - Взаємодія: Обговорення і вирішення проблем.	- Форми зворотного зв'язку в інтерфейсі та створення служби підтримки.

Застосування зазначених заходів дозволяє системно контролювати якість розробки на всіх етапах життєвого циклу Headless CMS.

Завдяки поєднанню технічних рішень (оптимізовані запити, шифрування, балансування навантаження) з UX-орієнтованим підходом (зрозумілий інтерфейс, адаптивний дизайн, гнучке налаштування) забезпечується не лише відповідність системи функціональним вимогам, а й високий рівень задоволеності користувачів.

РОЗДІЛ 4. ВПРОВАДЖЕННЯ ТА РОЗВИТОК HEADLESS CMS

4.1 Управління ризиками

Успішна реалізація проєктів у сфері інформаційних технологій вимагає не лише чіткого планування та технічної експертизи, але й ефективного управління ризиками. Ризики можуть виникати на всіх етапах життєвого циклу проєкту – від початкової розробки до масштабування готового продукту. Їх вчасне виявлення, оцінка та нейтралізація є ключовими факторами, що впливають на стабільність і результативність реалізації технічного рішення.

Проєкт створення Headless CMS передбачає низку специфічних викликів, пов’язаних із вибором архітектури, використанням open-source підходу, необхідністю гнучкої кастомізації, інтеграцією з іншими системами та відповідальністю за підтримку і розвиток платформи після впровадження. Крім того, важливими є організаційні ризики – кадрові, юридичні, фінансові й ті, що пов’язані з позиціонуванням продукту на ринку.

У цьому підрозділі буде проведено комплексний аналіз потенційних ризиків, що супроводжують проєкт впровадження Headless CMS. Методологія передбачає ідентифікацію ключових загроз, оцінку їх ймовірності та впливу, побудову матриці ризиків, а також розробку відповідних заходів реагування. Такий підхід дозволяє сформувати стратегію мінімізації наслідків у разі виникнення проблем і підвищити стійкість проєкту до зовнішніх та внутрішніх змін.

На етапі планування та реалізації проєкту впровадження Headless CMS важливо здійснити всебічну ідентифікацію можливих ризиків, що можуть вплинути на перебіг або результативність розробки. Для цього застосовується класифікаційний підхід, який дозволяє розподілити ризики за їхнім походженням, характером впливу та ступенем керованості. Такий підхід допомагає сформувати системне розуміння потенційних загроз і підготувати команду до ефективного реагування на них.

Ризики були згруповані в шість основних категорій: програмні, апаратні, внутрішні проєктні, зовнішні, форс-мажорні та ризики порушення кібербезпеки. У межах кожної групи проаналізовано характерні події, які можуть стати джерелом проблем у процесі реалізації. Для кожного з ризиків визначено силу впливу (тобто потенційний масштаб негативних наслідків для проєкту) та керованість (наскільки реально запобігти або зменшити вплив цього ризику). Це дозволяє у подальшому ефективно оцінити критичність кожного ризику та визначити першочергові пріоритети в управлінні ними.

Зведений перелік виявлених ризиків, їх класифікацію та оцінку представлено в таблиці 4.1. Ця таблиця є основою для подальшого побудови матриці ризиків та визначення протиризикових заходів.

Таблиця 4.1

Ідентифікація ризиків

№	Тип ризику	Ризикова подія	Сила впливу	Керованість
1	**2**	**3**	**4**	**5**
1	Програмні ризики	Вибір нестабільного або недостатньо підтримуваного стеку технологій	Висока	Середня
2	Програмні ризики	Критичні помилки у роботі API або втрати сумісності з frontend-застосунками	Висока	Середня
3	Програмні ризики	Низька продуктивність CMS при масштабуванні кількості проєктів	Середня	Висока
4	Апаратні ризики	Втрата доступу до хмарної інфраструктури через технічні збої або DDoS	Висока	Низька
5	Апаратні ризики	Недостатня продуктивність серверів для обробки запитів на піковому навантаженні	Середня	Середня
6	Апаратні ризики	Пошкодження або втрата даних у результаті збоїв обладнання	Висока	Середня

Продовження Таблиці 4.1

1	2	3	4	5
7	Внутрішні ризики проєкту	Втрата ключового розробника або архітектора системи	Висока	Низька
8	Внутрішні ризики проєкту	Відсутність чіткої комунікації між учасниками команди	Середня	Середня
9	Внутрішні ризики проєкту	Відставання від графіку спринтів або зрив дедлайнів	Середня	Висока
10	Зовнішні ризики проєкту	Зміни у законодавстві, які обмежують зберігання або обробку даних	Висока	Низька
11	Зовнішні ризики проєкту	Високий рівень конкуренції з боку великих міжнародних CMS-рішень	Середня	Низька
12	Зовнішні ризики проєкту	Зниження довіри до open-source продуктів у цільовій аудиторії	Середня	Середня
13	Форс мажори	Масштабні перебої з електроенергією або зв’язком (блекаут, кібератака)	Висока	Низька
14	Форс мажори	Тимчасова недоступність GitHub чи іншої інфраструктури розгортання	Середня	Низька
15	Форс мажори	Військово-політична нестабільність у регіоні розробки	Висока	Низька
16	Ризики порушення кібербезпеки	Несанкціонований доступ до адміністративної панелі CMS	Висока	Середня
17	Ризики порушення кібербезпеки	SQL-ін’єкції, XSS або інші критичні вразливості в API	Висока	Середня
18	Ризики порушення кібербезпеки	Витік даних клієнтів через неправильне зберігання або логування	Висока	Середня

Для подальшого аналізу та ранжування ризиків, ідентифікованих у межах проєкту, необхідно перевести якісні оцінки в умовно-кількісні. Це дозволяє систематизувати ризики, здійснити їх порівняльний аналіз, а також використовувати формалізовані методи для розрахунку рівня загрози. З цією метою застосовується шкала квазі-кількісного оцінювання ризиків, представлена в таблиці 4.2.

Таблиця 4.2

Шкала оцінювання ризиків

Проста якісна оцінка	**Деталізована якісна оцінка**	**Шифр оцінки**	**Відповідна квазі-кількісна оцінка**
–	Відсутній	немає	0
Низький	Низько-низький	НН	1
Низький	Низько-середній	НС	2
Низький	Низько-високий	НВ	3
Середній	Середньо-низький	СН	4
Середній	Середньо-середній	СС	5
Середній	Середньо-високий	СВ	6
Високий	Високо-низький	ВН	7
Високий	Високо-середній	ВС	8
Високий	Високо-високий	ВВ	9
Високий	Катастрофічний	К	10

Шкала базується на трирівневому підході: низький, середній, високий ризик, – кожен із яких деталізується підкатегоріями з відповідними шифрами (НН, СН, ВВ тощо). Кожному варіанту присвоюється числове значення від 0 до 10, яке можна використовувати у формулі Risk Score = Ймовірність × Сила впливу. Такий підхід дозволяє уникнути суб’єктивності при оцінюванні загроз і забезпечує більшу точність у прийнятті управлінських рішень щодо поводження з ризиками.

Застосування цієї шкали забезпечує уніфікацію оцінки всіх видів

ризиків, що розглядаються у проєкті. Надалі ці значення будуть використані для побудови матриці ризиків, де кожна подія буде відображена з урахуванням її імовірності та впливу на проєкт.

Для формування цілісного уявлення про ступінь загрози кожного з ризиків, виявлених на етапі ідентифікації, доцільно провести їх комплексну оцінку. Такий аналіз враховує не лише ймовірність настання події, але й масштаб її потенційних наслідків – як у часовому, так і у фінансовому вимірах, а також частоту виникнення у межах життєвого циклу проєкту.

У таблиці 4.3 представлено структуровану оцінку кожної ризикової події за чотирма ключовими критеріями:

- затримки у часі
- фінансові втрати
- ймовірність
- частота виникнення

Для кожного параметра подано як якісну оцінку (Я) – у вигляді умовного опису або шифру (наприклад, СН, К, ВВ), так і кількісну оцінку (К) – у вигляді числового еквівалента від 0 до 10.

Ці значення дозволяють обчислити інтегральний показник важливості ризику, що відображає загрозу для проєкту. Отримане значення нормалізовано до шкали від 0 до 100, що забезпечує зручне візуальне порівняння та ранжування ризиків за критичністю.

Такий підхід дає змогу визначити найбільш пріоритетні загрози, які потребують розробки конкретних протиризикових заходів, і дозволяє фокусувати зусилля команди на потенційно найнебезпечніших аспектах проєкту.

Для мінімізації наслідків реалізації критичних ризиків проєкту було розроблено цільові протиризикові заходи, які охоплюють усі ключові етапи життєвого циклу ризику – від профілактики до реагування. Такий підхід забезпечує системну готовність проєктної команди до потенційних загроз, скорочує час реагування та зменшує вплив інцидентів на загальну стабільність

проєкту.

Таблиця 4.3

Оцінювання ризиків

№	Ризикова подія	Затримки у часі		Фінансові втрати		Ймовірність		Частота (за проєкт)		Важливість (компл. показник)
		Я	К	Я	К	Я	К	Я	К	Я
2	2	3	4	5	6	7	8	9	10	11
1	Вибір нестабільного або недостатньо підтримуваного стеку технологій	СН	4	СВ	6	СВ	6	СС	5	80
2	Критичні помилки у роботі API або втрати сумісності з frontend	СВ	5	СВ	6	СВ	6	СС	5	100
3	Низька продуктивність CMS при масштабуванні	СС	4	СВ	5	СН	4	СН	4	35
4	Втрата доступу до хмарної інфраструктури через DDoS	СВ	6	ВВ	9	СВ	5	СН	3	93
5	Недостатня продуктивність серверів на піковому навантаженні	СС	5	СВ	6	СН	4	СН	4	53
6	Пошкодження або втрата даних через збій обладнання	СВ	6	ВВ	9	СН	4	СН	4	96
7	Втрата ключового розробника або архітектора	СВ	6	СВ	6	СН	4	СН	3	48
8	Відсутність чіткої комунікації в команді	СС	5	СВ	6	СС	5	СС	4	66

Продовження Таблиці 4.3

2	2	3	4	5	6	7	8	9	10	11
9	Відставання від графіку спринтів	СН	4	СС	5	СС	5	СС	4	44
10	Зміни у законодавстві щодо зберігання/обробки даних	СВ	6	ВВ	9	СВ	5	СН	3	90
11	Високий рівень конкуренції з боку міжнародних рішень	СС	5	СВ	6	СН	4	СН	3	40
12	Зниження довіри до open-source у клієнтів	СС	5	СС	5	СН	4	СН	3	33
13	Блекаут або масштабні перебої зв’язку	ВВ	8	К	10	СН	3	НС	2	53
14	Недоступність GitHub чи іншої інфраструктури CI/CD	СС	5	СВ	6	СС	5	СН	3	50
15	Військово-політична нестабільність у регіоні	ВВ	9	К	10	СВ	5	НС	2	100
16	Несанкціонований доступ до адмін-панелі CMS	СВ	6	ВВ	9	СВ	5	СН	3	90
17	SQL-ін’єкції або XSS вразливості	СВ	6	ВВ	9	СВ	5	СН	3	85
18	Витік даних клієнтів (неправильне зберігання/логування)	СН	4	К	10	ВН	6	НС	2	53

У таблиці 4.4 представлено комплекс дій для шести найбільш важливих ризиків, які мають найвищі показники важливості за результатами попередньої оцінки. Для кожного ризику визначено:

- ПРЗ 1 (профілактика) – запобіжні заходи, що знижують

ймовірність реалізації події;

- Симптом (рання ознака) – ознаки, які свідчать про початок прояву ризику;
- ПРЗ 2 (при симптомі) – дії для нейтралізації загрози на ранньому етапі;
- ПРЗ 3 (при проблемі) – кризові заходи на випадок повної реалізації ризику.

Таблиця 4.4

Розробка протиризикових заходів

№	Ризикова подія	ПРЗ 1	Симптом	ПРЗ 2	ПРЗ 3
1	2	3	4	5	6
1	Критичні помилки у роботі API або втрати сумісності з frontend	Покриття коду unit-тестами, CI/CD перевірка сумісності на кожному коміті	Нестабільна робота компонентів, збої при інтеграції нових модулів	Тимчасове відкатування оновлень, запуск fallback-версій API	Глибока переробка архітектури API, оновлення документації та сумісних модулів
2	Військово-політична нестабільність у регіоні	Міграція серверів у хмарну інфраструктуру за кордоном, віддалена структура команди	Збої зв’язку, перебої в онлайн-доступі до робочих середовищ	Активація резервних каналів зв’язку, перехід у режим мінімальної підтримки	Релокація ключових працівників та офісу в стабільні регіони
3	Пошкодження або втрата даних через збій обладнання	Налаштування щоденного резервного копіювання, дублювання баз даних	Повільна відповідь серверів, недоступність окремих даних	Активація резервних копій, переключення на резервну інфраструктуру	Переоснащення серверного середовища, відмова від локального зберігання
4	Втрата доступу до хмарної інфраструктури через DDoS	Використання CDN, балансувальників навантаження та засобів захисту від DDoS	Різке зниження швидкодії сайту, перебої в авторизації	Обмеження доступу для підозрілих IP, переведення порталу в режим read-only	Міграція на більш захищену хмарну платформу, посилення SLA

Продовження Таблиці 4.4

1	2	3	4	5	6
5	Зміни у законодавстві щодо зберігання/обробки даних	Аналіз юридичних ризиків на етапі проектування, дотримання GDPR та локальних норм	Юридичні запити, скарги клієнтів, зміна правил у законодавстві	Тимчасове призупинення обробки окремих типів даних, аудит систем	Рефакторинг логіки зберігання даних, укладення додаткових угод із користувачами
6	Несанкціонований доступ до адмін-панелі CMS	Використання 2FA, ведення логів дій, обмеження доступів за IP	Підозріла активність у логах, несанкціоновані зміни у налаштуваннях	Зміна паролів, блокування облікових записів, відкат змін	Оновлення безпекової політики, перехід на Zero Trust модель доступу

4.2 Моніторинг виконання проєкту

Ефективне управління будь-яким IT-проєктом неможливе без системного моніторингу його виконання. На практиці це означає постійне відстеження прогресу, своєчасне виявлення відхилень від плану, а також оцінку досягнення запланованих результатів. Вибір інструментів та підходів до моніторингу повинен відповідати характеру проєкту, його тривалості, структури та меті. У випадку розробки Headless CMS, де проект реалізується в умовах відносно стислих термінів, високої динаміки завдань та переважання якісних результатів над кількісними, критично важливо застосовувати просту, але показову модель контролю.

З цієї причини у межах проєкту було обрано підхід milestone tracking – тобто контроль виконання робіт через досягнення ключових контрольних точок. Така модель дозволяє зберігати стратегічну видимість прогресу, не перевантажуючи процес управління зайвими метриками, які характерні для класичних фінансово-орієнтованих моделей на кшталт Earned Value Management (EVM).

У рамках моніторингу виконання проєкту впровадження Headless CMS обрано підхід відстеження контрольних точок (milestone tracking) замість використання системи Earned Value Management (EVM). Це рішення ґрунтується на специфіці проєкту, зокрема:

1. Невеликій тривалості та обмеженій кількості фаз. Проєкт триває 12 місяців із чітким розподілом на фази планування, розробки, впровадження та підтримки. У таких умовах надмірна деталізація затрат і трудових ресурсів у форматі EVM є необґрунтовано складною.
2. Домінування нематеріальних результатів, таких як реліз MVP, запуск SaaS-інфраструктури, публікація документації або досягнення стабільної версії системи. Ці результати зручніше вимірювати через досягнення ключових точок, а не у грошовому еквіваленті виконаних робіт.
3. Наявність гнучкого плану, де частина задач виконується в динамічному середовищі. Метод milestone tracking дозволяє зафіксувати лише дійсно важливі результати, без втрати гнучкості.
4. Фокус на якість і стабільність, а не на витрати. Показники успіху цього проєкту – не точність бюджету, а досягнення технічної цілі: створення кастомізованої, open-source Headless CMS.

З огляду на зазначене, контроль виконання проєкту базується на досягненні чітко визначених контрольних точок, які відображають фактичний поступ у реалізації ключових завдань. Щоб забезпечити прозорість та оперативність моніторингу, було сформовано перелік основних віх проєкту з фіксацією запланованих і фактичних дат, статусу виконання та коротких коментарів щодо причин відхилень. Такий підхід дозволяє своєчасно виявляти збої у плануванні, коригувати розклад і зберігати контроль над загальною динамікою проєкту.

У таблиці 4.5 наведено поточний стан реалізації контрольних точок проєкту. Незважаючи на низку локальних затримок, пов’язаних із

погодженням вимог, доопрацюванням технічних завдань і уточненням функціональності, команда демонструє послідовний прогрес. Ключові етапи – підготовка до запуску та фінальне впровадження системи – реалізовані вчасно, що свідчить про ефективну організацію процесів та здатність гнучко адаптуватися до змін у межах методології Scrum.

Таблиця 4.5

Контрольні точки проєкту

Контрольна точка	**Запланована дата**	**Фактична дата**	**Статус**	**Коментар**
1	**2**	**3**	**4**	**5**
Початок проєкту	01.05.2025	01.05.2025	Виконано	
Формування проєктної команди	03.05.2025	03.05.2025	Виконано	
Проведення вступної наради з командою	08.05.2025	10.05.2025	Затримка	Відтерміно-вана через графік учасників
Визначення вимог до CMS на основі інтерв’ю	09.05.2025	20.05.2025	Затримка	Інтерв’ю проводились із запізненням
Аналіз ринку та конкурентів	12.05.2025	12.06.2025	Затримка	Очікування результатів попереднього етапу
Інтерв’ю з цільовими користувачами	31.05.2025	15.06.2025	Затримка	Проведення опитування розпочалось із затримкою
Формалізація вимог	05.06.2025	25.06.2025	Затримка	Потрібно було додатково узагальнити результати опитування

Продовження Таблиці 4.5

1	2	3	4	5
Узгодження функціональних вимог	10.06.2025	04.07.2025	Затримка	Необхідно було внести уточнення після обговорення з командою
Узгодження нефункціональних вимог	11.06.2025	10.07.2025	Затримка	Частина вимог була оновлена за результатами технічного аудиту
Розробка технічного завдання для бекенду	14.06.2025	14.07.2025	Затримка	Очікувалось завершення всіх вимог перед початком роботи
Узгодження ТЗ з командою	20.06.2025	16.07.2025	Затримка	Окремі технічні рішення вимагали уточнень
Прототипування у Figma	09.07.2025	26.08.2025	Затримка	Затримка через погодження дизайн-системи
Презентація дизайну команді	12.07.2025	06.09.2025	Затримка	Затримка прототипування вплинула на презентацію
Завершення розробки основних модулів CMS	20.08.2025	22.10.2025	Затримка	Реалізація редактора і медіабібліотеки вимагала більше часу

Продовження Таблиці 4.5

1	2	3	4	5
Завершення тестування	31.08.2025	18.11.2025	Затримка	Частина сценаріїв виявилась неактуальною і потребувала оновлення
Підготовка до запуску	10.09.2025	25.11.2025	Затримка	Попередні етапи виконувались із затримкою
Фінальне впровадження системи	20.09.2025	7.12.2025	Затримка	Попередні етапи виконувались із затримкою

Таким чином, використання моделі milestone tracking забезпечило збалансоване поєднання стратегічного контролю й гнучкого управління, що дозволяє зберігати цілісність графіку, мінімізувати ризики зриву термінів і поступово просуватись до успішного завершення впровадження системи Headless CMS.

4.3 Аналіз технічної готовності до запуску системи

Аналіз технічної готовності до запуску системи є важливим етапом завершення циклу розробки програмного забезпечення, оскільки дозволяє всебічно оцінити відповідність створеного продукту поставленим технічним вимогам, рівень стабільності, безпеки, продуктивності та інтеграційної готовності. Цей аналіз є запорукою зниження ризиків на етапі розгортання та забезпечення безперервності бізнес-процесів.

У межах оцінки технічної готовності було проаналізовано кожен із ключових модулів Headless CMS. Зокрема, API Gateway протестовано на стійкість до високих навантажень та несанкціонованих запитів – реалізовано логування, обробку помилок і захист від DDoS. Контентний модуль (Content

Service) успішно виконує CRUD-операції, зберігає контент відповідно до заданої структури, підтримує типізацію та зв’язки між сутностями. У модулі користувачів реалізовано повноцінну аутентифікацію, авторизацію, ролі та права доступу, а також проведено тестування на спроби SQL-ін’єкцій та XSS-атаки. Модуль медіафайлів забезпечує стабільне збереження, масштабування та отримання медіаресурсів у різних форматах і розширеннях. Система моніторингу та логування на основі Prometheus + Grafana успішно збирає дані про стан сервісів, навантаження та помилки, з можливістю відправлення нотифікацій.

Функціональне тестування проводилося відповідно до технічних вимог та user stories. Кожна функція оцінювалася на відповідність очікуваному результату. Для критичних модулів застосовувалися unit-тести, а API-запити перевірялися вручну через Postman. Усі ключові сценарії – реєстрація, додавання контенту, редагування, фільтрація, видалення, пошук – пройшли повну перевірку без критичних відхилень.

Оцінка продуктивності проводилася за допомогою Apache JMeter. За умов 500 одночасних запитів до API система демонструвала стабільний час відповіді в межах 150–250 мс. Всі сервіси зберігали працездатність без втрати даних або серйозних збоїв. Горизонтальне масштабування забезпечується використанням Kubernetes, що дозволяє гнучко адаптуватися до зростання навантаження.

Безпекова перевірка охоплювала як ручне тестування, так і автоматизовані сканери, зокрема OWASP ZAP. Було виявлено кілька незначних недоліків у CORS-політиках і логіці реєстрації, які усунуто до завершення тестування. Додатково впроваджено шифрування паролів за допомогою bcrypt, обмеження доступу до адмін-панелі за IP-адресою, валідацію даних на сервері та захист від CSRF і XSS.

На момент аудиту було підготовлено повний пакет документації: інструкцію для адміністраторів, Swagger-документацію для API, гайд з налаштування середовища для розробників, а також документацію з

моніторингу та логування. Повна наявність актуальної технічної документації є передумовою для запуску системи у продуктивне середовище.

Інтеграційна готовність підтверджена через тестові з'єднання з зовнішніми системами: реалізовано webhook-увідомлення для frontend-редакторів, інтеграцію з телеграм-ботом для надсилання повідомлень про новий контент, а також підключення до хмарного сховища для резервного копіювання. Архітектура системи підтримує відкритість і розширюваність без модифікації ядра.

Щоб краще візуалізувати технічну архітектуру проєкту та стан готовності основних компонентів до розгортання, було побудовано структурну схему, яка відображає взаємозв'язки між модулями, проведені перевірки та результати тестування. На ній представлено головні функціональні блоки системи – API Gateway, контентний модуль, модуль користувачів, роботу з медіафайлами, систему моніторингу та безпеки, а також документацію й інтеграційні можливості. Кожен із компонентів доповнений позначками про проходження ключових перевірок, включно з навантажувальним тестуванням, валідацією функціональності, аналізом безпеки та готовністю до масштабування. Це дозволяє візуально оцінити цілісність рішення та підтвердити високу ступінь його технічної зрілості. Схема технічної архітектури та готовності компонентів наведена на рис. 4.1.

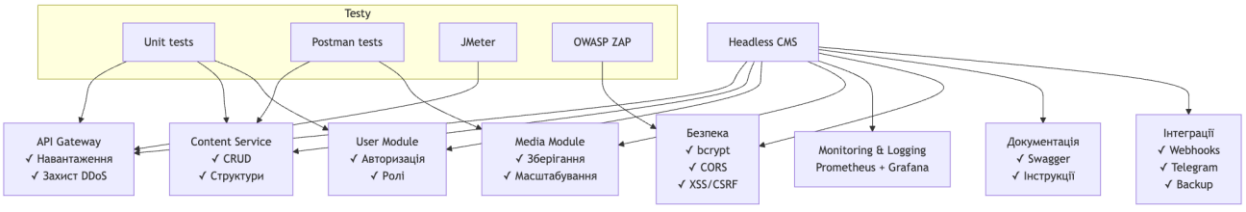


Рис. 4.1. Архітектура Headless CMS та перевірка технічної готовності ключових компонентів до розгортання

Результати технічного аудиту та тестування підтверджують високу ступінь готовності системи до запуску у продуктивне середовище. Усі

критичні компоненти пройшли перевірку на функціональність, безпеку, масштабованість та відповідність документації. Система має резервний план відновлення, а також забезпечує зручні засоби адміністрування та контролю. Таким чином, Headless CMS готова до першого розгортання, а основні ризики пов’язані переважно з адаптацією у конкретному середовищі замовника.

4.4 Аналіз ефективності впровадження Headless CMS

Впровадження Headless CMS у межах організації стало ключовим кроком у цифровій трансформації процесів управління контентом. Система була інтегрована у наявну інфраструктуру, адаптована під потреби відділів маркетингу, IT та аналітики, а співробітники пройшли базове навчання щодо користування адміністративною панеллю та API-інтерфейсами. Це забезпечило швидкий перехід до нової моделі роботи без суттєвих збоїв чи перешкод у поточній діяльності. Одразу після запуску відзначено скорочення часу на публікацію та оновлення контенту приблизно на 35% завдяки централізованій структурі зберігання даних, можливості попереднього перегляду матеріалів, автоматизованим сценаріям публікації та відсутності потреби у залученні frontend-фахівців на кожному етапі.

Завдяки API-орієнтованій архітектурі вдалося досягти єдиного джерела правди для контенту: дані синхронізуються між вебсайтом, мобільним додатком, e-mail-розсилками та внутрішнім порталом організації. Це не лише спростило управління даними, але й забезпечило узгодженість комунікації на різних каналах. Зміни до контенту тепер вносяться через інтерфейс адміністратора і одразу відображаються у всіх точках дотику з користувачем. Унаслідок цього зменшилося навантаження на технічні ресурси: участь IT-відділу у повсякденному супроводі знизилася, а значна частина завдань була делегована до контент-команд, дизайнерів та менеджерів, які можуть працювати безпосередньо з платформою.

Попри початкові інвестиції у розробку, налаштування, навчання персоналу та оновлення інфраструктури, вже протягом перших місяців

спостерігався позитивний вплив на бюджет. Витрати на підтримку скоротилися за рахунок зменшення потреби у сторонніх підрядниках та внутрішніх ресурсах для усунення технічних несправностей або реалізації нових функцій. Водночас зменшення простоїв при публікації матеріалів позитивно вплинуло на маркетингові кампанії та операційну стабільність.

Система виявилася стабільною при масштабуванні – навіть за умов підвищеного навантаження, пов’язаного з імпортом масивів даних або публікацією великої кількості змін, продуктивність залишалася в межах норми. Інтеграція Prometheus для моніторингу та налаштування автоматичних сповіщень дозволили вчасно реагувати на нештатні ситуації, зокрема – у разі помилок API або уповільнень на стороні бази даних. Крім того, модульна архітектура Headless CMS дозволила безболісно впроваджувати нові функції без порушення стабільності існуючих рішень.

Для об’єктивного відображення змін у внутрішніх процесах після впровадження Headless CMS, нижче наведено таблицю 4.2 з порівнянням ключових показників до та після впровадження проєкту.

Таблиця 4.6

Порівняльна таблиця ключових показників організації до та після реалізації проєкту

№	Показник	Було	Стало
1	**2**	**3**	**4**
1	Час публікації нового контенту	~2–3 дні з участю кількох відділів	До 1 робочого дня без залучення frontend
2	Кількість технічних звернень у місяць	15–20	3–5
3	Канали, що обслуговуються окремо	Вебсайт, мобільний додаток, портал – кожен вручну	Централізований API-доступ до всіх каналів
4	Участь IT-відділу в оновленнях контенту	Постійна	Мінімальна

Продовження Таблиці 4.6

1	2	3	4
5	Залежність від зовнішніх підрядників	Висока (підтримка CMS, верстка, інтеграції)	Знижена – більшість задач вирішуються внутрішньою командою
6	Гнучкість масштабування	Обмежена, потребує значних змін у коді	Висока – масштабування через модулі та API
7	Збої або простої через помилки системи	1–2 значущі події на місяць	Практично відсутні, контроль через моніторинг
8	Середній час реакції на помилки	4–6 годин	30–60 хвилин завдяки сповіщенням
9	Можливість повторного використання контенту	Частково (копіювання вручну)	Повна (контент у форматі блоків і API)
10	Підготовка нового сайту/лендінгу	2–3 тижні	3–5 днів завдяки шаблонам і повторному використанню секцій

В узагальнення можна констатувати, що система повністю виправдала очікування з боку організації. Вона не лише оптимізувала контент-менеджмент і спростила технічне адміністрування, а й створила гнучку основу для майбутнього масштабування цифрової екосистеми. Headless CMS показала себе як стратегічне рішення, яке вже сьогодні приносить вимірювану ефективність і має потенціал для подальшого розвитку, у тому числі – через інтеграцію з аналітичними платформами, CRM-системами або персоналізованими каналами комунікації.

4.5 Стратегія подальшого розвитку та масштабування

Після успішного впровадження Headless CMS у межах організації постає завдання формування стратегії її подальшого розвитку. Йдеться не лише про технічне масштабування, а й про розширення функціональних можливостей,

залучення нових груп користувачів, інтеграцію з іншими цифровими платформами, а також комерціалізацію рішення в довгостроковій перспективі.

Архітектура системи, заснована на мікросервісному підході, дає змогу масштабувати окремі модулі без шкоди для цілісності рішення. У разі зростання навантаження передбачене горизонтальне масштабування API-сервісів через Kubernetes, інтеграція з CDN для швидкої доставки контенту, а також перехід на хмарні сервери або інфраструктуру з вищими обчислювальними потужностями. У наступних версіях заплановано впровадження підтримки GraphQL API, що забезпечить ще більшу гнучкість при формуванні запитів і зменшить обсяг переданих даних, покращуючи загальну продуктивність системи.

З точки зору функціональності, пріоритетами на найближчий період є додавання багатомовної підтримки, створення візуального редактора для побудови складних сторінок, інтеграція аналітики ефективності контенту та інтерфейсів для динамічного шаблонування. Додатково розглядається можливість застосування AI-інструментів для автоматичного створення текстів, генерації SEO-підказок і семантичного аналізу контенту, що суттєво спростить роботу контент-команд.

Поступове розширення сфери використання CMS охоплює все більше відділів організації: від маркетингу й служби підтримки – до HR і внутрішніх комунікацій. Це створює потребу в деталізації ролей користувачів, налаштуванні політик доступу, оновленні навчальних матеріалів та створенні внутрішньої служби підтримки. Такий підхід гарантує контрольований перехід до масштабнішого використання системи без ризику перевантаження або втрати зручності.

Окремий напрям – комерційне масштабування системи. Планується відкриття доступу до Headless CMS для зовнішніх партнерів у форматі white-label-рішення, розвиток SaaS-моделі з гнучкими тарифами, а також просування open-source-версії з опцією платного хостингу та технічної підтримки. Це дозволить перетворити внутрішній інструмент на ринковий

продукт, що здатен конкурувати з міжнародними аналогами.

Загалом подальший розвиток Headless CMS базується на поєднанні технічної гнучкості, потреб реальних користувачів і стратегічного бачення. Система має високий потенціал масштабування як за кількістю даних і користувачів, так і в плані географічного або бізнес-розширення. Завдяки цьому вона може стати ключовою платформою для цифрової трансформації організації в середньостроковій і довгостроковій перспективі.

ВИСНОВКИ

У межах даної кваліфікаційної роботи було комплексно досліджено, спроєктовано та реалізовано систему Headless CMS – гнучке, масштабоване та сучасне рішення для управління контентом у цифровому середовищі. Реалізація проєкту базувалася на актуальних потребах малого та середнього бізнесу, освітніх установ, а також внутрішніх корпоративних платформ, які все частіше стикаються з обмеженнями традиційних CMS. Отримані результати підтверджують доцільність обраної архітектури, ефективність застосованих технологій та потенціал подальшого розвитку розробленого рішення.

Аналіз предметної галузі на початковому етапі дозволив чітко сформулювати проблеми, притаманні класичним CMS: низька гнучкість у роботі з нестандартними типами контенту, ускладнене масштабування, жорстка прив’язка до конкретного інтерфейсу користувача та обмежені можливості інтеграції з іншими системами. У відповідь на ці виклики було розроблено концепцію Headless CMS, побудовану на основі мікросервісної архітектури, з чітким поділом frontend- і backend-частин та відкритим API для взаємодії з будь-якими зовнішніми каналами.

У процесі реалізації проєкту було розроблено технічну архітектуру системи, спроєктовано базу даних, побудовано API Gateway, реалізовано окремі сервіси для роботи з контентом, користувачами, медіафайлами, а також інтегровано систему моніторингу й логування. Було впроваджено повний набір CRUD-операцій, механізми автентифікації та авторизації користувачів, а також забезпечено безпечну та стабільну роботу системи при навантаженні. Усі технічні модулі пройшли етапи тестування: функціонального, навантажувального й безпекового, що дозволило переконатися у відповідності рішення встановленим вимогам.

Окрему увагу приділено питанням управління проєктом. Побудована організаційна структура команди, визначено ролі й відповідальності, застосовано підхід Scrum для управління задачами у динамічному середовищі.

Було розроблено структуру WBS, сформовано календарний план реалізації, виконано оцінку ресурсів, ризиків та ефективності впровадження. Всі ключові аспекти управління проєктом – планування, моніторинг, контроль якості, управління ризиками – були реалізовані відповідно до найкращих практик.

Інвестиційні дослідження, проведені у рамках роботи, продемонстрували фінансову життєздатність проєкту. Побудована модель монетизації на основі підписки дозволяє досягти точки беззбитковості вже у перший рік функціонування, а завдяки гнучкій тарифній сітці CMS здатна задовольнити потреби як малих команд, так і великих організацій. Розрахунки враховували реалістичні витрати на розробку, інфраструктуру, маркетинг і підтримку, а також прогнозований темп зростання клієнтської бази.

З погляду замовника, система продемонструвала позитивний вплив на організаційні та операційні процеси. Було досягнуто скорочення часу на публікацію контенту, зменшення навантаження на IT-відділ, підвищення якості взаємодії між структурними підрозділами та пришвидшення запуску нових цифрових продуктів. Результати впровадження підтверджують, що система готова до масштабування, а її подальший розвиток відкриває нові можливості як для внутрішнього використання, так і для комерціалізації у форматі SaaS-платформи або open-source-рішення.

Отже, усі цілі та завдання, поставлені на початку роботи, були досягнуті у повному обсязі. Розроблена Headless CMS є актуальним, технологічно сучасним і функціонально зрілим рішенням, що відповідає потребам цифрового ринку. Вона може бути легко адаптована під різні категорії користувачів, інтегрована в існуючі інформаційні системи та слугувати основою для подальшої цифрової трансформації. Завдяки поєднанню технічної гнучкості, економічної доцільності та високої якості реалізації, система має значний потенціал до масштабування та довгострокового використання в реальних умовах.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Boiko B. Content Management Bible. Wiley Publishing, 2011. 1176 p.

2. Nadareishvili I., Mitra R., McLarty M., Amundsen M. Microservice Architecture: Aligning Principles, Practices, and Culture. O'Reilly Media, 2016. 280 p.

3. W3Techs. Usage statistics of content management systems [Електронний ресурс]. URL: https://w3techs.com/technologies/overview/content_management/all (дата звернення: 12.05.2025).

4. Buytaert D. Drupal for Web Developers. Packt Publishing, 2009. 300 p.

5. Netlify. What's a Headless CMS? The Beginner's Guide [Електронний ресурс]. URL: https://www.netlify.com/blog/complete-guide-to-headless-cms/ (дата звернення: 12.05.2025).

6. Strapi. 10 reasons to use a headless CMS [Електронний ресурс]. URL: https://strapi.io/blog/10-reasons-headless-cms (дата звернення: 12.05.2025).

7. Wikipedia. Headless content management system [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Headless_content_management_system (дата звернення: 12.05.2025).

8. Brightspot. Headless CMS: Pros and Cons You Need to Know about Headless Content Management Systems [Електронний ресурс]. URL: https://www.brightspot.com/cms-architecture/headless-cms/headless-cms-pros-and-cons (дата звернення: 12.05.2025).

9. ButterCMS. Pros and Cons of Implementing a Headless CMS into modern websites and web-applications [Електронний ресурс]. URL: https://buttercms.com/blog/headless-cms-advantages/ (дата звернення: 12.05.2025).

10. SoftKraft. Should I Use Headless CMS? Use Cases, Pros and Cons [Електронний ресурс]. URL: https://www.softkraft.co/headless-cms-pros-cons/ (дата звернення: 12.05.2025).

11. MarketsandMarkets. Headless CMS Market by Component, Application, Organization Size, Industry, and Region - Global Forecast to 2027 [Електронний ресурс]. URL: https://www.marketsandmarkets.com/Market-Reports/headless-cms-market-125113778.html (дата звернення: 12.05.2025).

12. Netlify. The Rise of Headless CMS: Trends and Adoption [Електронний ресурс]. URL: https://www.netlify.com/blog/the-rise-of-headless-cms-trends-and-adoption/ (дата звернення: 12.05.2025).

13. Strapi. Why Companies Are Switching to Headless CMS [Електронний ресурс]. URL: https://strapi.io/blog/why-companies-are-switching-to-headless-cms (дата звернення: 12.05.2025).

14. JAMstack.org. What is JAMstack? [Електронний ресурс]. URL: https://jamstack.org/what-is-jamstack/ (дата звернення: 12.05.2025).

15. 9. The Standard for Project Management and a Guide to the Project Management Body of Knowledge (PMBOK® Guide) – Seventh Edition / USA. – Project Management Institute, 2021. – 250 с.

17. Highsmith J. Agile Project Management: Creating Innovative Products. Addison-Wesley Professional, 2009. 432 p.

18. Фендьо О. Управління IT-проєктами з використанням гнучких методологій Agile, Scrum, Kanban [Електронний ресурс] // Distance Education in Ukraine: Innovative Normative-Legal Pedagogical Aspects. 2023. №1. С. 419-425. URL: https://www.researchgate.net/publication/374224543_UPRAVLINNA_IT-PROEKTAMI_Z_VIKORISTANNAM_GNUCKIH_METODOLOGIJ_AGILE_SCRUM_KANBAN (дата звернення: 20.04.2025).

19. Храпкін О.М., Кіндрат О.В., Чопей Р.О. Управління проєктами в IT-галузі: методики, інструменти та керування ризиками [Електронний ресурс] // Економіка та суспільство. 2023. №55. С. 110-115. URL: https://www.researchgate.net/publication/375536748_UPRAVLINNA_PROEKTAMI_V_IT-GALUZI_METODIKI_INSTRUMENTI_TA_KERUVANNA_RIZIKAMI (дата

звернення: 07.05.2025).

20. Project Management Institute. A Guide to the Project Management Body of Knowledge (PMBOK® Guide). 6th ed. Project Management Institute, 2017. 756 p.

21. Kniberg H., Skarin M. Kanban and Scrum Making the Most of Both. InfoQ, 2009. 122 p.

22. Tarawneh A.S., Alrfou K. A Comparative Study of Agile Methods: XP versus SCRUM [Електронний ресурс] // ResearchGate. 2015. URL: https://www.researchgate.net/publication/277813128_A_Comparative_Study_of_Agile_Methods_XP_versus_SCRUM (дата звернення: 12.05.2025).

23. Saleh M., Huq S. Comparative Study within Scrum, Kanban, XP Focused on Their Practices [Електронний ресурс] // ResearchGate. 2019. URL: https://www.researchgate.net/publication/332240953_Comparative_Study_within_Scrum_Kanban_XP_Focused_on_Their_Practices (дата звернення: 12.05.2025).

24. Future Market Insights. Headless CMS Software Market Size & Trends 2025-2035. URL: https://www.futuremarketinsights.com/reports/headless-cms-software-market (дата звернення: 12.05.2025).

25. Market Research Future. Headless CMS Software Market Size, Global Report - 2034. URL: https://www.marketresearchfuture.com/reports/headless-cms-software-market-34090 (дата звернення: 12.05.2025).

26. Search Engine Journal. The 10 Best Headless CMS Platforms To Consider. 2024. URL: https://www.searchenginejournal.com/best-headless-cms/522674/ (дата звернення: 12.05.2025).

27. Бленда Н. О., Коротєєв М. А., Соковніна Д. М., Соколюк С. Ю., Жарун О. В. Стратегічний аналіз зовнішнього середовища –основа визначення стратегічного напряму розвитку підприємницьких структур. Науковий вісник Уманського національного університету садівництва. – 2021. – № 99(2). – С. 124-131.

28. Asana. How to conduct a stakeholder analysis (+ free template). URL: https://asana.com/resources/project-stakeholder (дата звернення: 12.05.2025).

29. Wikipedia. Persona (user experience). URL: https://en.wikipedia.org/wiki/Persona_(user_experience) (дата звернення: 12.05.2025).

30. Webstacks. Contentful vs Builder.io: A Comparison Guide for B2B Companies. URL: https://www.webstacks.com/blog/contentful-vs-builder-io (дата звернення: 12.05.2025).

31. Net Solutions. Strapi vs Contentful: Choosing the Right Headless CMS Platform. URL: https://www.netsolutions.com/insights/strapi-vs-contentful/ (дата звернення: 12.05.2025).

32. Kombee. Sanity vs Strapi vs Contentstack vs Contentful vs Builder.io - Comprehensive Comparison. URL: https://www.kombee.com/blogs/sanity-vs-strapi-vs-contentstack-vs-contentful-vs-builderio-comprehensive-comparison (дата звернення: 12.05.2025).

33. SourceForge. Builder.io vs. Contentful vs. Strapi Comparison. URL: https://sourceforge.net/software/compare/Builder.io-vs-Contentful-vs-Strapi/ (дата звернення: 12.05.2025).

34. Garcia, F. A. Z., & Russo, R. F. S. M. (2019). Leadership and Performance of the Software Development Team: Influence of the Type of Project Management. Review of Business Management, 21(5), 970–1005.

35. Shahin, M., & Babar, M. A. (2020). On the Role of Software Architecture in DevOps Transformation: An Industrial Case Study. arXiv preprint arXiv:2003.06108.

36. Farooqui, S. M. (2018). Enterprise DevOps Framework: Transforming IT Operations. Apress.

37. Davis, J., & Daniels, R. (2016). Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling at Scale. O'Reilly Media.

38. Stasiukynas, A., & Valickas, A. (2021). Team Effectiveness in Software Development: The Role of Personality and Work Factors. Business: Theory and Practice, 22(1), 1–10.

39. Дяченко Л. М., Завадська І. В. Управління проєктами: навчальний

посібник. Київ: Центр учбової літератури, 2020. 312 с.

40. Dettmer H. William. The Logical Thinking Process: A Systems Approach to Complex Problem Solving. Milwaukee: ASQ Quality Press, 2007. 452 p.

41. Dwyer T., Hogan A., Timms G. Goal Tree Modelling for Project Planning and Risk Analysis. Systems Engineering, Wiley, 2018. Vol. 21(3). P. 250-263.

42. Завадська І. В., Гринчук Н. В. Методи і моделі прийняття управлінських рішень: навч. посіб. Львів: ЛНУ ім. І. Франка, 2020. 284 с.

43. Анікіна І. Д., Малєва Т. М. Стратегічне управління проєктами: теорія і практика. Київ: КНЕУ, 2019. 318 с.

44. NORAD. The Logical Framework Approach: Handbook for Objectives-Oriented Project Planning. Norwegian Agency for Development Cooperation. – 1999. – 88 p.

45. Schmidt T. Strategic Project Management Made Simple: Solution Tools for Leaders and Teams. – 2nd ed. Hoboken: Wiley, 2021. – 272 p.

46. Forrester Research. The Rise of the Headless Content Management System. Forrester Report, 2022. URL: https://go.forrester.com/blogs/the-rise-of-the-headless-cms/ (дата звернення: 12.05.2025).

47. Wikipedia. Kontent.ai. URL: https://en.wikipedia.org/wiki/Kontent.ai (дата звернення: 12.05.2025).

48. Бородін І. О., Савицька Л. А. Концепція ієрархічної структури робіт (WBS) у проєктному менеджменті. Вісник Національного університету "Львівська політехніка". Серія: Менеджмент та підприємництво в Україні, 2020. №868. С. 52-58. URL: https://www.researchgate.net/publication/346035085 (дата звернення: 12.05.2025).

49. Голуб О. С., Ляшенко В. І. Календарне планування ІТ-проєктів з використанням інструментів візуалізації (Gantt Chart, Network Diagram) // Інформаційні технології і засоби навчання. 2021. №6 (86). С. 123-132. URL: https://journal.iitta.gov.ua/index.php/itlt/article/view/4732 (дата звернення: 12.05.2025).

50. Бровко Н. А., Дяченко Л. М. Застосування програмного забезпечення

ProjectLibre у процесі управління освітніми проєктами // Вісник Житомирського державного університету імені Івана Франка. 2022. №4 (108). С. 210-215. URL: https://visnyk.zu.edu.ua/article/view/274274 (дата звернення: 10.05.2025).

ДОДАТОК А

Структури таблиць та SQL-запити для реалізації фізичної моделі БД

```sql
CREATE TABLE ContentItems (
  ContentID INT AUTO_INCREMENT PRIMARY KEY,
  Title VARCHAR(255) NOT NULL,
  Body TEXT NOT NULL,
  ContentType VARCHAR(50) NOT NULL,
  AuthorID INT,
  CreatedDate DATETIME NOT NULL,
  PublishedDate DATETIME,
  Status VARCHAR(50) NOT NULL,
  FOREIGN KEY (AuthorID) REFERENCES Users(UserID)
);
```

Рис. А.1. SQL запит для створення сутності ContentItems

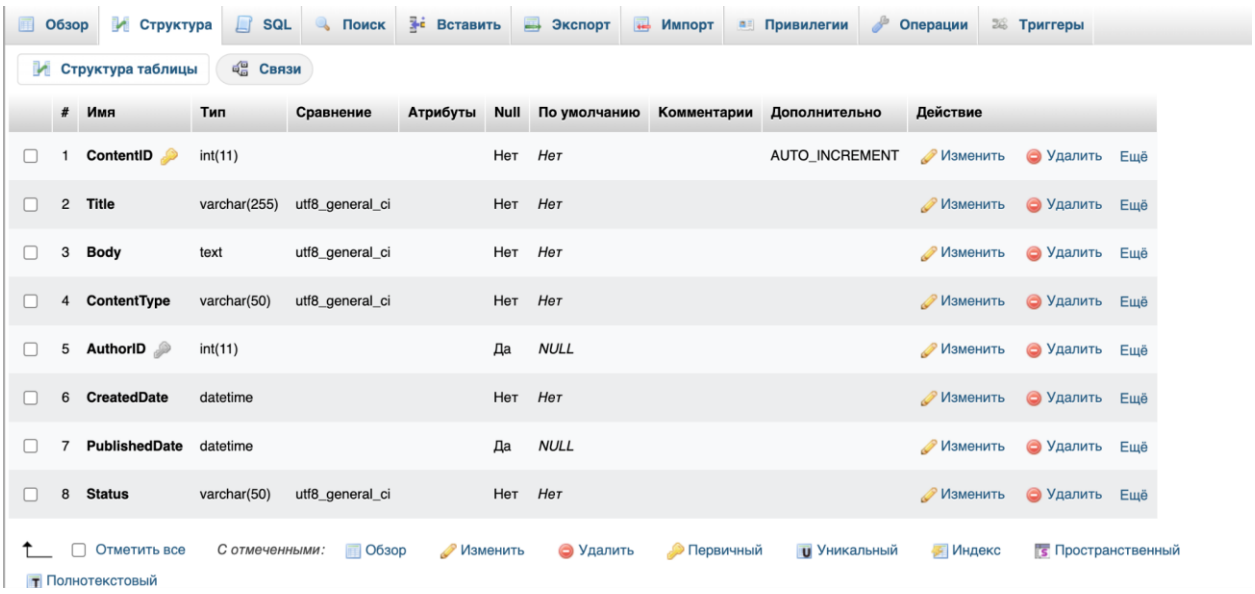


Рис. А.2. Сутність ContentItems у СУБД phpMyAdmin

```sql
CREATE TABLE Metadata (
  MetadataID INT AUTO_INCREMENT PRIMARY KEY,
  ContentID INT,
  Key VARCHAR(255) NOT NULL,
  Value TEXT NOT NULL,
  FOREIGN KEY (ContentID) REFERENCES ContentItems(ContentID)
);
```

Рис. А.3. SQL запит для створення сутності Metadata

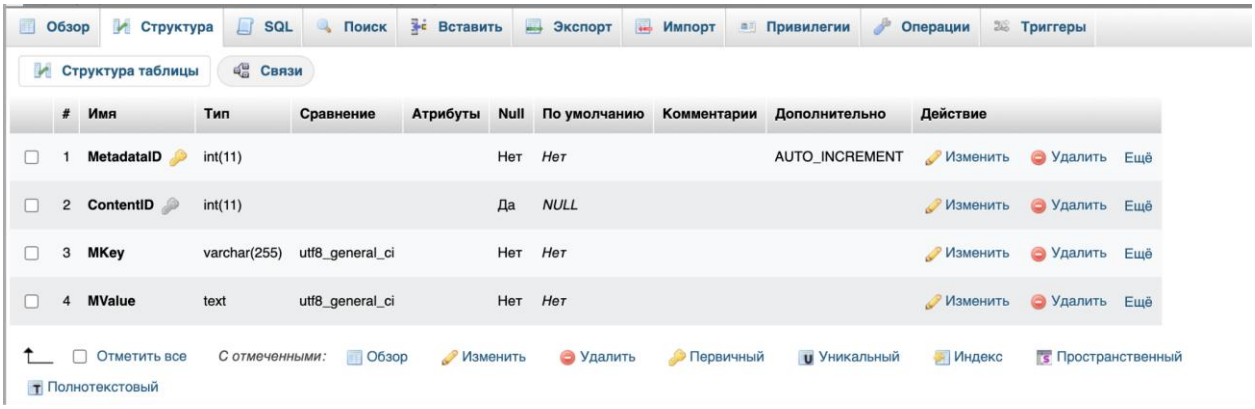
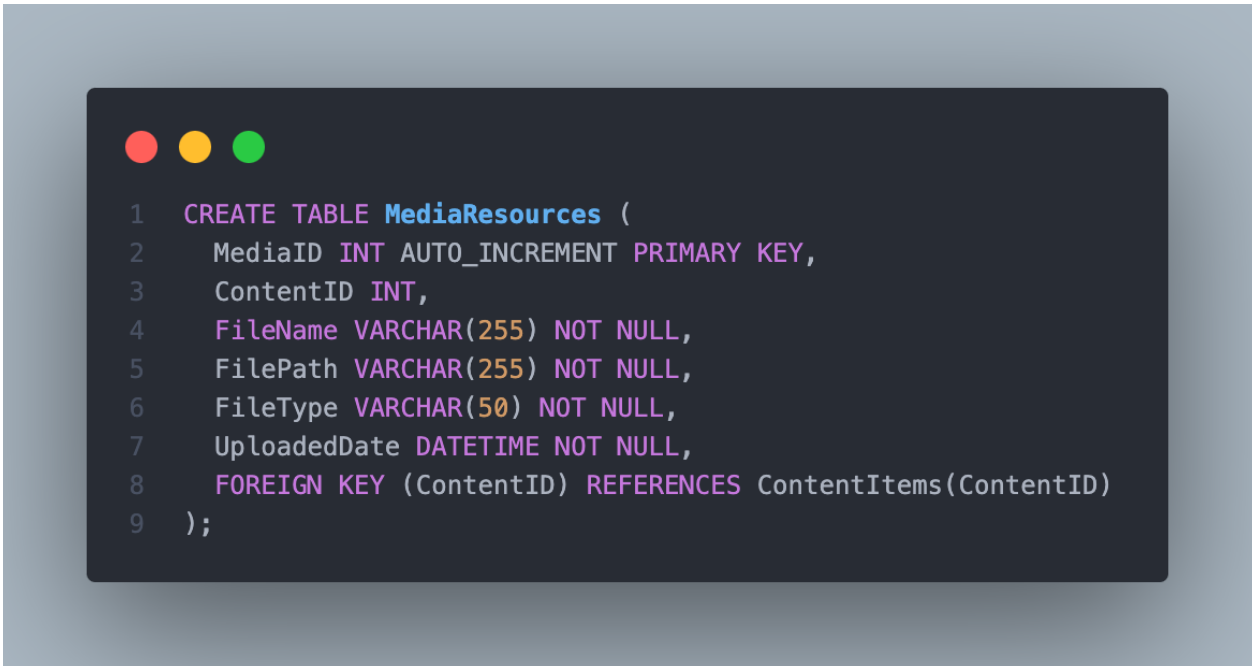
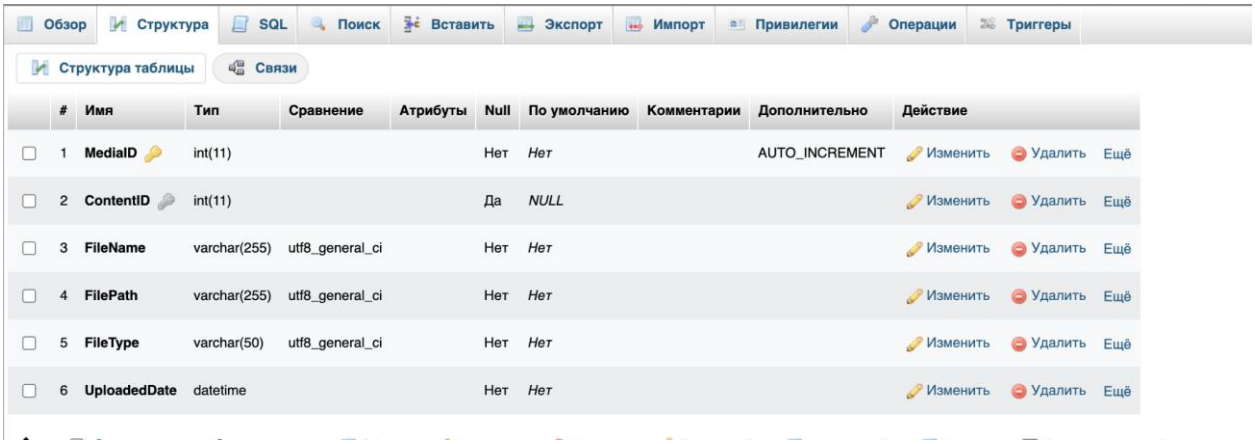


Рис. А.4. Сутність Metadata у СУБД phpMyAdmin



```sql
CREATE TABLE MediaResources (
  MediaID INT AUTO_INCREMENT PRIMARY KEY,
  ContentID INT,
  FileName VARCHAR(255) NOT NULL,
  FilePath VARCHAR(255) NOT NULL,
  FileType VARCHAR(50) NOT NULL,
  UploadedDate DATETIME NOT NULL,
  FOREIGN KEY (ContentID) REFERENCES ContentItems(ContentID)
);
```

Рис. А.5. SQL запит для створення сутності MediaResources

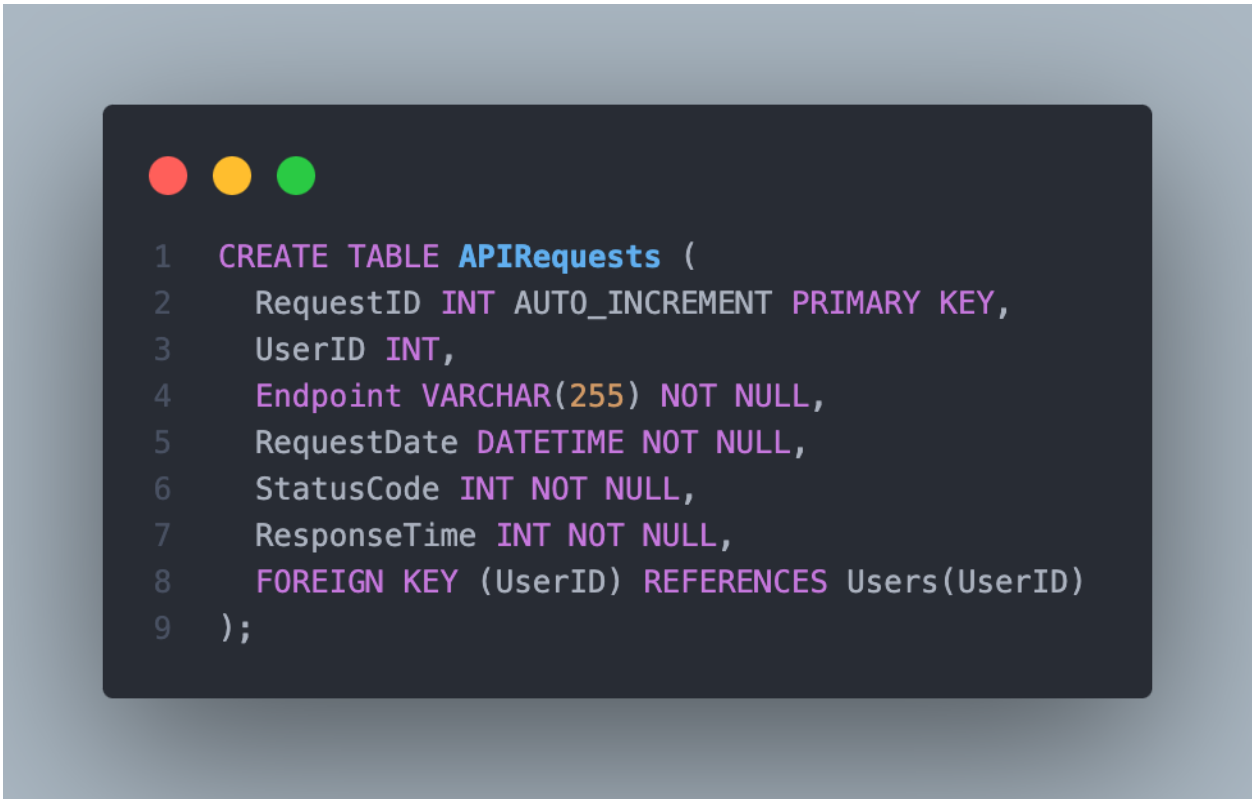


Обзор | Структура | SQL | Поиск | Вставить | Экспорт | Импорт | Привилегии | Операции | Триггеры

Структура таблицы | Связи

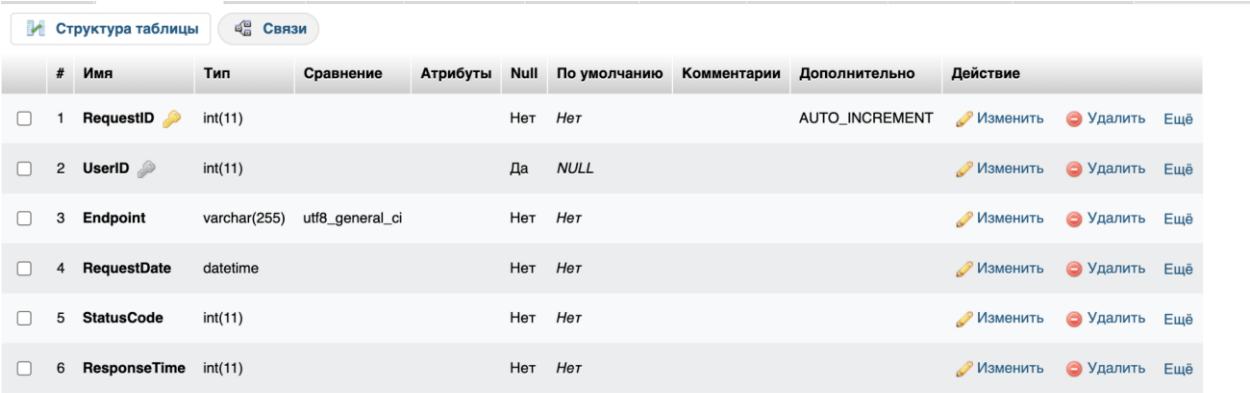
	#	Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Комментарии	Дополнительно	Действие
☐	1	MediaID	int(11)			Нет	*Нет*		AUTO_INCREMENT	Изменить Удалить Ещё
☐	2	ContentID	int(11)			Да	*NULL*			Изменить Удалить Ещё
☐	3	FileName	varchar(255)	utf8_general_ci		Нет	*Нет*			Изменить Удалить Ещё
☐	4	FilePath	varchar(255)	utf8_general_ci		Нет	*Нет*			Изменить Удалить Ещё
☐	5	FileType	varchar(50)	utf8_general_ci		Нет	*Нет*			Изменить Удалить Ещё
☐	6	UploadedDate	datetime			Нет	*Нет*			Изменить Удалить Ещё

Рис. А.6. Сутність MediaResources у СУБД phpMyAdmin



```sql
CREATE TABLE APIRequests (
  RequestID INT AUTO_INCREMENT PRIMARY KEY,
  UserID INT,
  Endpoint VARCHAR(255) NOT NULL,
  RequestDate DATETIME NOT NULL,
  StatusCode INT NOT NULL,
  ResponseTime INT NOT NULL,
  FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

Рис. А.7. SQL запит для створення сутності APIRequests



Структура таблицы | Связи

	#	Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Комментарии	Дополнительно	Действие
☐	1	RequestID	int(11)			Нет	*Нет*		AUTO_INCREMENT	Изменить Удалить Ещё
☐	2	UserID	int(11)			Да	*NULL*			Изменить Удалить Ещё
☐	3	Endpoint	varchar(255)	utf8_general_ci		Нет	*Нет*			Изменить Удалить Ещё
☐	4	RequestDate	datetime			Нет	*Нет*			Изменить Удалить Ещё
☐	5	StatusCode	int(11)			Нет	*Нет*			Изменить Удалить Ещё
☐	6	ResponseTime	int(11)			Нет	*Нет*			Изменить Удалить Ещё

Рис. А.8. Сутність APIRequests у СУБД phpMyAdmin

ДОДАТОК Б

Код алгоритмів для отримання, оновлення та видалення контенту

```php
function buildQuery($filters) {
    $query = "SELECT * FROM ContentItems WHERE 1=1";
    if (isset($filters['author_id'])) {
        $query .= " AND AuthorID = " . intval($filters['author_id']);
    }
    if (isset($filters['status'])) {
        $query .= " AND Status = '" . htmlspecialchars($filters['status']) . "'";
    }
    return $query;
}
function formatResponse($results) {
    return json_encode($results);
}
function getContent($filters) {
    $db = new PDO('mysql:host=localhost;dbname=headless_cms', 'username', 'password');
    $query = buildQuery($filters);
    $stmt = $db->query($query);
    $results = $stmt->fetchAll(PDO::FETCH_ASSOC);
    return formatResponse($results);
}
```

Рис. Б.1. Базовий код алгоритму для отримання контенту

```php
function contentExists($content_id, $db) {
    $stmt = $db->prepare("SELECT COUNT(*) FROM ContentItems WHERE ContentID = ?");
    $stmt->execute([$content_id]);
    return $stmt->fetchColumn() > 0;
}
function updateContent($content_id, $data) {
    if (validateData($data)) {
        $db = new PDO('mysql:host=localhost;dbname=headless_cms', 'username', 'password');
        if (contentExists($content_id, $db)) {
            $stmt = $db->prepare("UPDATE ContentItems SET Title = ?, Body = ?, ContentType = ?, PublishedDate = ?, Status = ? WHERE ContentID = ?");
            $stmt->execute([$data['title'], $data['body'], $data['content_type'], $data['published_date'], $data['status'], $content_id]);
            return ["status" => "success"];
        } else {
            return ["status" => "error", "message" => "Content not found"];
        }
    } else {
        return ["status" => "error", "message" => "Invalid data"];
    }
}
```

Рис. Б.2. Базовий код алгоритму для оновлення контенту

```php
function deleteContent($content_id) {
    $db = new PDO('mysql:host=localhost;dbname=headless_cms', 'username', 'password');
    if (contentExists($content_id, $db)) {
        $stmt = $db->prepare("DELETE FROM ContentItems WHERE ContentID = ?");
        $stmt->execute([$content_id]);
        return ["status" => "success"];
    } else {
        return ["status" => "error", "message" => "Content not found"];
    }
}
```

Рис. Б.3. Базовий код алгоритму для видалення контенту

ДОДАТОК В

Дизайн сторінок Headless CMS

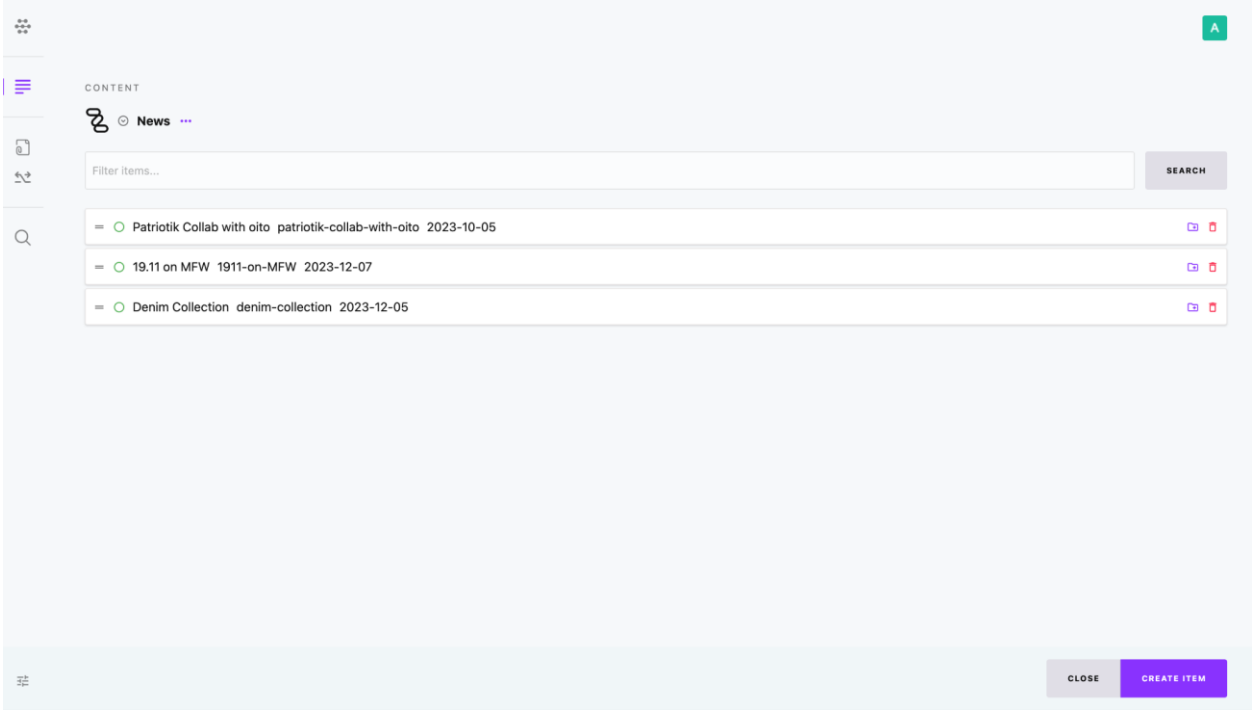


Рис. В.1. Дизайн сторінки управління контентом

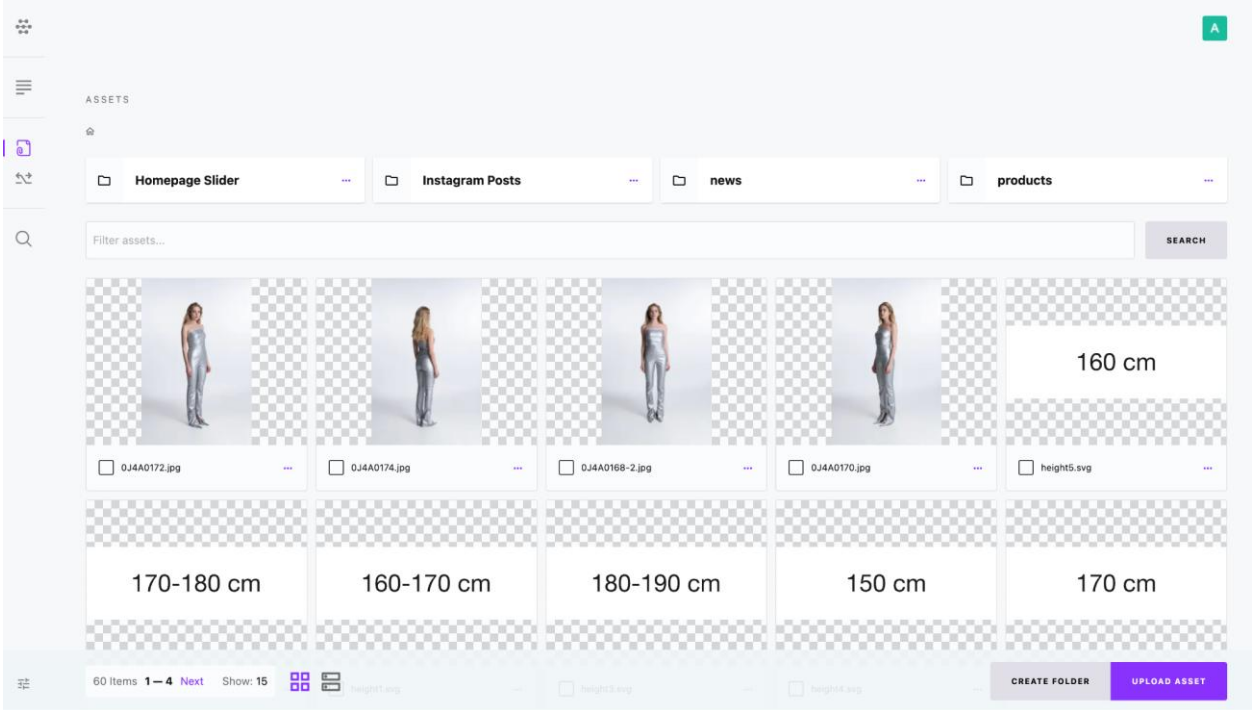


Рис. В.2. Дизайн сторінки медіа-менеджера

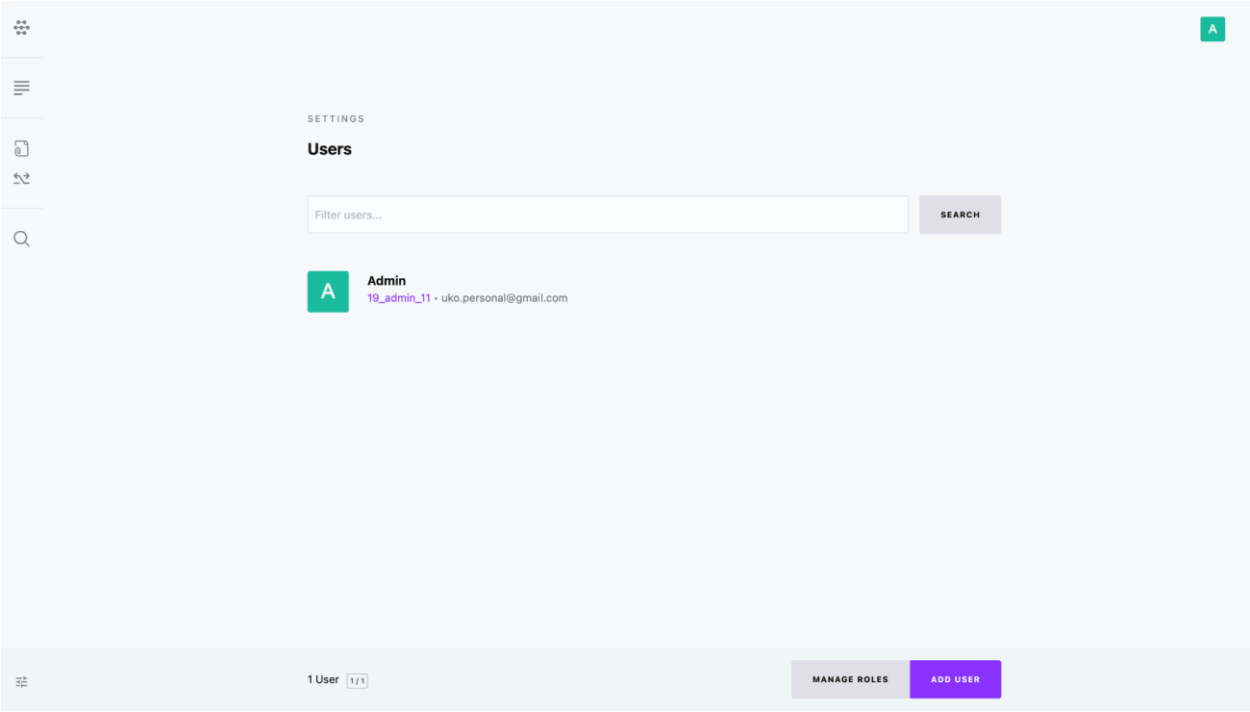


Рис. В.3. Дизайн сторінки управління користувачами

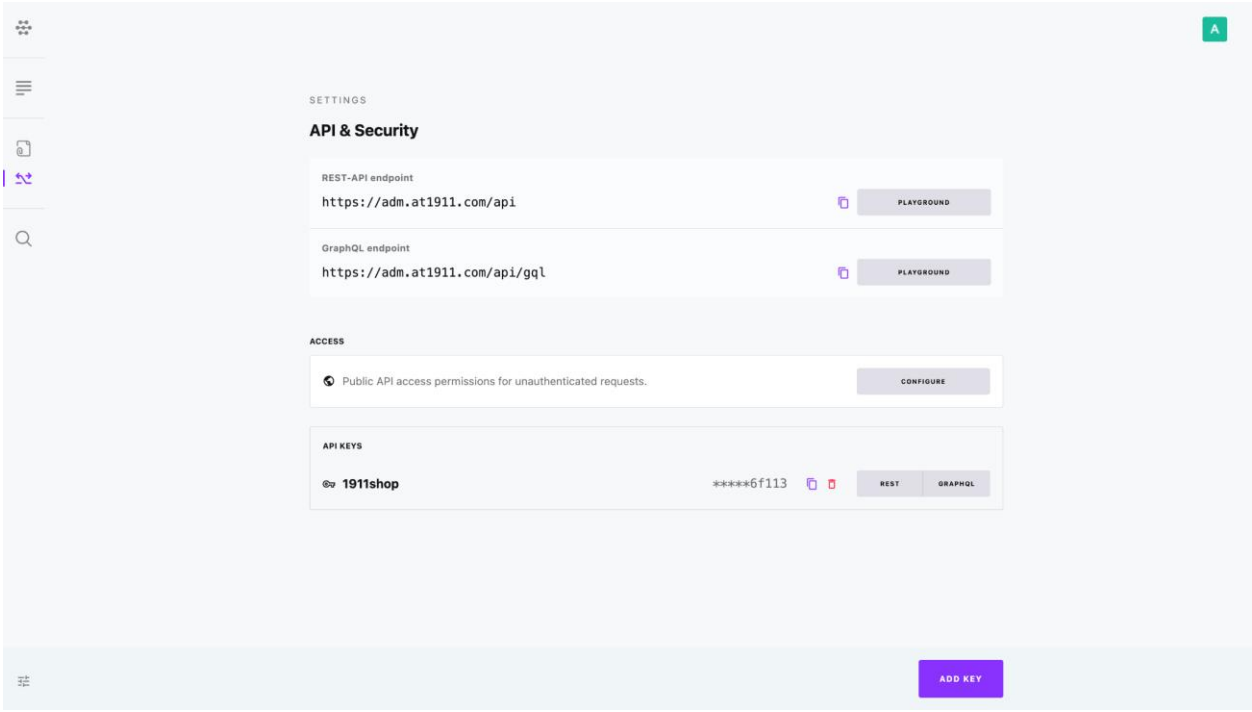


Рис. В.4. Дизайн сторінки налаштування API

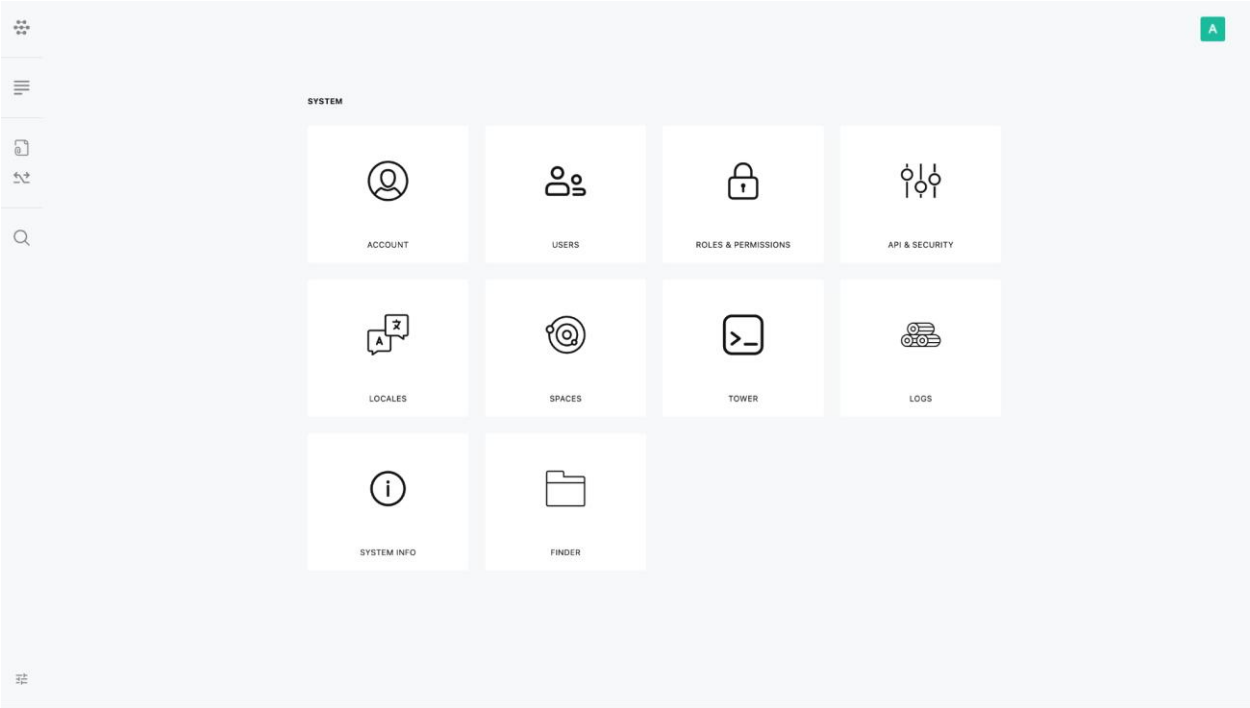


Рис. В.5. Дизайн сторінки системних налаштувань