ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ
КАФЕДРА ІНФОРМАТИКИ І КОМП'ЮТЕРНИХ НАУК

Пояснювальна записка

до дипломної бакалаврської роботи

на тему:

РОЗРОБКА ТА ВПРОВАДЖЕННЯ МОДУЛЬНОЇ СИСТЕМИ КЕРУВАННЯ ВЕБ-КОНТЕНТОМ НА БАЗІ DJANGO ТА WAGTAIL

Виконав: студент 4 курсу, групи 4КН
спеціальності 122 «Комп'ютерні науки»

Русскій А. В.

Керівник: к.т.н., доц. Моісеєнко С. В.

Рецензент: _____

Хмельницький - 2026 р.

Факультет	**Інформаційних технологій та дизайну**
Кафедра	**Інформатики і комп'ютерних наук**
Рівень вищої освіти	**перший (бакалаврський) рівень**
Галузь підготовки	**12 «Інформаційні технології»**
Освітньо-професійна програма	**Комп'ютерні науки**
Спеціальність	**122 «Комп'ютерні науки»**

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри ІКН,
доцент
_____ Моісеєнко С. В.
«____» _____ 2026 року

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ СТУДЕНТА

_____Русскій Андрій Вікторович_____
(прізвище, ім'я, по батькові)

1. Тема роботи: Розробка та впровадження модульної системи керування веб-контентом на базі Django та Wagtail.

Керівник роботи Моісеєнко Світлана, кандидат технічних наук, доцент кафедри інформатики і комп'ютерних наук
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом ХНТУ від 28.01. 2024 р. № 162-с

2. Строк подання студентом роботи
10.05.2026

3. Вихідні дані до роботи

Матеріали та результати, отримані під час проходження переддипломної практики, методичні вказівки, технічна література

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Вступ; 1. Постанова завдання на розробку модульної системи; 2. Проектування модульної системи керування вебконтентом; 3. Реалізація та практичне застосування модульнох системи керування вебконтентом; Висновок

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Таблиць - 8,

Формул - 7,

Рисунків - 22

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 18.02.2026

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Ознайомлення з темою дипломної роботи, визначення мети, завдань, об’єкта та предмета дослідження	18.02.2026 - 28.02.2026	Виконано
2	Дослідження предметної області керування вебконтентом офіційного сайту закладу вищої освіти	01.02.2026 - 02.03.2026	Виконано
3	Аналіз існуючих систем керування вебконтентом, зокрема WordPress, Drupal та Wagtail	03.03.2026 - 14.03.2026	Виконано
4	Формування системних, функціональних і нефункціональних вимог до модульної системи керування вебконтентом	15.03.2026 - 25.03.2026	Виконано
5	Проєктування архітектури системи, сторінкової структури, контентних блоків та UML-діаграм	26.03.2026 - 10.04.2026	Виконано
6	Реалізація програмного продукту на базі Django та Wagtail, створення моделей сторінок і блочного контенту	11.04.2026 - 24.04.2026	Виконано
7	Реалізація API-модуля подій та анонсів, налаштування маршруту, API-блоку і клієнтської JavaScript-логіки	25.04.2026 - 05.05.2026	Виконано
8	Перевірка працездатності системи, підготовка інструкції користувача, оформлення пояснювальної записки та висновків	06.05.2026	Виконано

Студент _____ Андрій РУССКІЙ

(підпис) (прізвище та ініціали)

Керівник роботи _____ Світлана МОІСЄЄНКО

(підпис) (прізвище та ініціали)

ЗМІСТ

Скорочення, термін, позначення	Пояснення
CMS	Content Management System - система керування контентом
WCMS	Web Content Management System - система керування вебконтентом
API	Application Programming Interface - програмний інтерфейс застосунку
JSON	JavaScript Object Notation - текстовий формат обміну структурованими даними
HTML	HyperText Markup Language - мова розмітки гіпертекстових документів
CSS	Cascading Style Sheets - каскадні таблиці стилів
JS	JavaScript - мова програмування для реалізації клієнтської інтерактивності
URL	Uniform Resource Locator - уніфікований покажчик ресурсу
UI	User Interface - користувацький інтерфейс
UML	Unified Modeling Language - уніфікована мова моделювання
ORM	Object-Relational Mapping - об’єктно-реляційне відображення
СУБД	Система управління базами даних
БД	База даних
Django	Вебфреймворк мовою Python для розробки серверної частини вебзастосунків

АНОТАЦІЯ

У кваліфікаційній роботі розглянуто розробку та впровадження модульної системи керування вебконтентом для сайту закладу вищої освіти на базі Django та Wagtail. Актуальність теми зумовлена потребою у створенні зручного інструменту для централізованого адміністрування сторінок, контентних блоків, подій, документів та інших інформаційних матеріалів університетського вебресурсу.

У роботі проаналізовано предметну область керування вебконтентом, розглянуто існуючі CMS, зокрема WordPress, Drupal та Wagtail, визначено їх переваги й обмеження. Обґрунтовано вибір Django та Wagtail як технологічної основи системи, оскільки вони забезпечують підтримку сторінкових моделей, блочного формування контенту, адміністративного інтерфейсу та API-взаємодії.

Під час проєктування визначено архітектуру системи, структуру сторінок, контентних блоків і механізм повторного використання даних. Для наочного подання структури програмного продукту використано UML-діаграми класів і компонентів. Практичну реалізацію виконано із застосуванням Django, Wagtail, PostgreSQL, HTML, CSS та JavaScript.

У межах системи реалізовано сторінкові моделі, набір контентних блоків, механізм формування сторінок через StreamField, а також API-модуль подій та анонсів. Модуль подій забезпечує централізоване зберігання інформації та її повторне використання на різних сторінках сайту через маршрут /api/events/ і клієнтську JavaScript-логіку.

Практичне значення роботи полягає у створенні працездатної модульної системи керування вебконтентом, яка спрощує адміністрування сайту, зменшує дублювання інформації та створює основу для подальшого розширення функціоналу.

Abstract

This thesis examines the development and implementation of a modular web content management system for a higher education institution's website based on Django and Wagtail. The relevance of the topic stems from the need to create a user-friendly tool for the centralized administration of pages, content blocks, events, documents, and other informational materials on the university's website.

The thesis analyzes the domain of web content management, reviews existing CMS platforms—including WordPress, Drupal, and Wagtail—and identifies their advantages and limitations. The choice of Django and Wagtail as the system's technological foundation is justified, as they support page models, block-based content composition, an administrative interface, and API integration.

During the design phase, the system architecture, page structure, content blocks, and data reuse mechanism were defined. UML class and component diagrams were used to visually represent the software product's structure. The practical implementation was carried out using Django, Wagtail, PostgreSQL, HTML, CSS, and JavaScript.

The system implements page models, a set of content blocks, a page generation mechanism via StreamField, as well as an API module for events and announcements. The events module provides centralized storage of information and its reuse on various pages of the site via the /api/events/ route and client-side JavaScript logic.

The practical significance of this work lies in the creation of a functional modular web content management system that simplifies site administration, reduces information duplication, and lays the foundation for further functional expansion.

ВСТУП

У сучасних умовах цифровізації діяльності закладів вищої освіти офіційний вебсайт є важливим інструментом представлення університету в інформаційному просторі, взаємодії зі здобувачами освіти, абітурієнтами, працівниками та іншими користувачами. Він забезпечує публікацію новин, оголошень, нормативних документів, відомостей про освітні програми, структурні підрозділи та інші інформаційні матеріали.

Актуальність теми полягає в необхідності створення гнучкої та зручної системи керування вебконтентом, яка б забезпечувала ефективне адміністрування структури сайту, повторне використання окремих інформаційних модулів, спрощення супроводу та можливість подальшого розширення функціоналу. Традиційні підходи до розробки вебсайтів часто призводять до дублювання логіки, складності підтримки та обмеження масштабованості. Тому доцільним є використання модульної системи керування вебконтентом, побудованої на сучасних засобах веброзробки.

Метою роботи є розробка та впровадження модульної системи керування вебконтентом для сайту закладу вищої освіти на базі фреймворку Django та CMS Wagtail, яка забезпечує гнучке адміністрування сторінок, повторне використання контентних блоків і можливість масштабування функціоналу.

Для досягнення поставленої мети необхідно розв’язати такі **завдання**:

1. Дослідити предметну область університетських вебресурсів та особливості керування вебконтентом;
2. Проаналізувати існуючі підходи та програмні засоби для побудови систем керування вебконтентом;
3. Сформувати функціональні та нефункціональні вимоги до розроблюваної системи;
4. Спроєктувати архітектуру модульної системи керування вебконтентом;
5. Визначити структуру основних сутностей, сторінок і контентних блоків системи;
6. Обґрунтувати вибір технологій Django та Wagtail;

7. Реалізувати основні програмні модулі системи;
8. Реалізувати API-взаємодію та повторне використання контентних блоків на прикладі модуля подій та анонсів;
9. Виконати перевірку працездатності системи та описати особливості її використання.

Об’єктом дослідження є процес керування вебконтентом офіційного сайту закладу вищої освіти.

Предметом дослідження є методи, моделі, програмні засоби та архітектурні рішення, що використовуються під час розробки модульної системи керування вебконтентом на базі Django та Wagtail.

Методи дослідження: аналіз предметної області та існуючих програмних рішень, методи об’єктно-орієнтованого проєктування, моделювання структури програмного продукту, порівняльний аналіз технологій, програмна реалізація засобами Python, Django, Wagtail, HTML, CSS, JavaScript, а також тестування функціональності розробленої системи.

Практичне значення одержаних результатів полягає у створенні працездатної модульної системи керування вебконтентом, яка може бути використана для підтримки та розвитку вебсайту університету. Запропоноване рішення забезпечує централізоване адміністрування сторінок і контентних блоків, зменшує дублювання даних, спрощує супровід сайту та створює основу для подальшого розширення функціоналу.

Структура роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку джерел посилання та додатків. У першому розділі розглянуто предметну область, виконано аналіз аналогів і сформульовано постановку задачі. У другому розділі описано проєктування архітектури, структури та основних технологічних рішень системи. У третьому розділі наведено особливості програмної реалізації, результати тестування та інструкцію щодо використання розробленого програмного продукту

РОЗДІЛ 1. ПОСТАНОВКА ЗАВДАННЯ НА РОЗРОБКУ МОДУЛЬНОЇ СИСТЕМИ КЕРУВАННЯ ВЕБКОНТЕНТОМ

1.1 Опис предметної області

Предметною областю даної кваліфікаційної роботи є процес керування вебконтентом офіційного сайту закладу вищої освіти. У сучасних умовах цифровізації університетський вебсайт виступає одним із головних інформаційних ресурсів, через який забезпечується представлення закладу освіти в цифровому середовищі, інформування користувачів та підтримка взаємодії з абітурієнтами, здобувачами вищої освіти, працівниками, випускниками й іншими відвідувачами.

Вебсайт закладу вищої освіти виконує не лише інформаційну, а й репрезентативну функцію, оскільки є офіційним цифровим представництвом університету для внутрішньої та зовнішньої аудиторії. У політиці вебприсутності SUNY Empire State University університетські вебсторінки прямо визначаються як офіційний голос університету для абітурієнтів, здобувачів освіти, випускників, працівників та широкої громадськості [16]. Це підкреслює, що якість структури, актуальність і керованість вебконтенту безпосередньо впливають на сприйняття закладу освіти в цифровому середовищі.

У межах цієї предметної області важливе місце займає вебконтент, до якого належать новини, події, анонси, сторінки факультетів і кафедр, описи освітніх програм, нормативні документи, контактна інформація та інші інформаційні матеріали. Такий контент характеризується різнорідністю, значним обсягом і потребою в регулярному оновленні. Саме тому ефективне створення, редагування, зберігання, структурування та публікація інформації є важливою складовою підтримки сучасного університетського вебресурсу.

Ефективність університетського вебресурсу значною мірою залежить не лише від наявності інформації, а й від способу її організації. У рекомендаціях Appalachian State University зазначається, що ефективні вебсайти повинні бути цілеспрямованими, орієнтованими на користувача та забезпечувати якісний

досвід використання на різних пристроях; при цьому важливу роль відіграють контентна стратегія, доступність, зручність користування та логічна структура інформації [17]. У контексті даної роботи це означає, що система керування вебконтентом повинна забезпечувати не просто публікацію матеріалів, а їх зручне структурування, підтримку цілісної навігації та можливість оперативного оновлення.

Для організації цих процесів застосовуються системи керування контентом. CMS розглядається як програмне забезпечення, призначене для створення, зберігання, редагування та публікації цифрового контенту. Така система поєднує інтерфейс адміністрування, шаблони відображення, програмні модулі, механізми взаємодії з базою даних і засоби формування користувацької частини вебсайту. Узагальнену схему традиційної системи керування вебконтентом наведено на рисунку 1.1 [1].

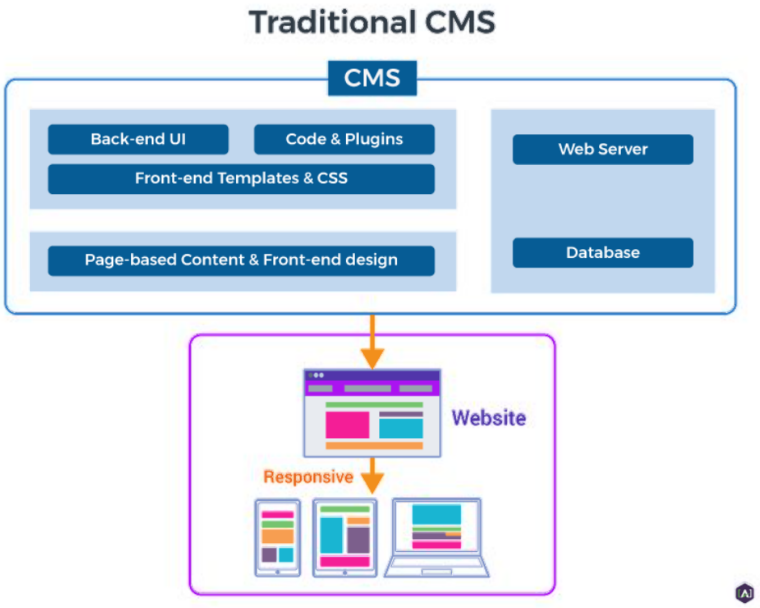

Рисунок 1.1- Узагальнена схема традиційної системи керування вебконтентом [1]

У традиційній CMS користувач працює з контентом через панель керування, після чого система обробляє введені дані, зберігає їх у базі даних і забезпечує їх виведення на вебсторінках для кінцевих користувачів. Таким чином, CMS виконує роль проміжної ланки між процесом підготовки

інформаційних матеріалів та їхнім публічним відображенням. Для університетських сайтів це має особливе значення, оскільки велика кількість розділів і постійна зміна інформації вимагають централізованого й зручного механізму адміністрування.

До важливих характеристик систем керування вебконтентом належать зручність адміністрування, підтримка різних ролей користувачів, централізоване зберігання інформації, можливість швидкого оновлення матеріалів, використання шаблонів відображення та розширення функціоналу за рахунок додаткових модулів. Для сайтів закладів вищої освіти такі можливості є особливо актуальними, оскільки вони дозволяють підтримувати логічну структуру ресурсу, зберігати єдині принципи подання інформації та зменшувати складність супроводу [1].

Разом із тим традиційні підходи до побудови CMS не завжди повною мірою відповідають сучасним вимогам до гнучкості та масштабованості. У процесі експлуатації університетського вебсайту часто виникає потреба у повторному використанні окремих інформаційних блоків, швидкому додаванні нових сторінок і секцій, розширенні функціоналу без суттєвого перепроєктування всієї системи, а також у взаємодії між різними модулями через API. Це стосується, зокрема, таких типів контенту, як новини, події, анонси чи документи, які можуть бути потрібні одночасно в декількох розділах сайту.

Додаткової актуальності досліджуваній предметній області надають сучасні процеси цифрової трансформації освіти. UNESCO наголошує, що цифрові інновації використовуються для розширення доступу до освітніх можливостей, підвищення якості навчання, розвитку освітніх систем та підтримки цифрових середовищ взаємодії [18]. За таких умов університетський вебсайт розглядається не лише як засіб оприлюднення інформації, а як компонент цифрової інфраструктури закладу вищої освіти, що повинен бути гнучким, масштабованим і придатним до інтеграції нових сервісів.

У зв’язку з цим у межах даної роботи предметна область розглядається в контексті розробки модульної системи керування вебконтентом для сайту

закладу вищої освіти. Така система повинна забезпечувати централізоване керування інформацією, можливість створення й повторного використання контентних блоків, спрощення адміністрування, масштабованість архітектури та подальше функціональне розширення вебресурсу.

Отже, предметна область дослідження охоплює процеси адміністрування, структурування, збереження та публікації вебконтенту на сайті закладу вищої освіти. Її особливості обґрунтовують доцільність розробки модульної системи керування вебконтентом, яка поєднує централізоване зберігання даних, гнучке формування сторінок і підтримку повторного використання інформаційних компонентів.

1.2 Аналіз існуючих аналогів і потенційних конкурентних переваг програмного продукту

Для визначення доцільних підходів до розробки модульної системи керування вебконтентом необхідно проаналізувати існуючі програмні рішення, що використовуються для створення та супроводу вебресурсів. Серед найбільш поширених платформ такого типу доцільно розглянути WordPress, Drupal та Wagtail, оскільки вони належать до відомих систем керування контентом, підтримують розширення функціоналу та можуть бути використані для побудови складних вебресурсів різного призначення.

Для вебресурсів закладів вищої освіти під час вибору CMS важливими є не лише базові можливості публікації контенту, а й підтримка цілісної інформаційної архітектури, актуальності матеріалів, зручності навігації та узгодженого користувацького досвіду. У дослідженні EDUCAUSE вебкерування в закладах вищої освіти розглядається як інституційно важлива задача, оскільки офіційний сайт університету одночасно є маркетинговим і інформаційним центром для різних категорій аудиторії, а його наповнення повинно бути своєчасним, навігованим і послідовним [19]. Це означає, що під час аналізу аналогів доцільно враховувати не лише популярність платформи, а й її здатність підтримувати керованість складної багаторівневої структури сайту.

Одним із найбільш поширених рішень у сфері керування вебконтентом є WordPress. Його популярність зумовлена простотою встановлення, великою кількістю готових тем оформлення, широким вибором плагінів та зручним інтерфейсом адміністрування. Платформа дозволяє створювати різні типи вебсайтів, а її функціональність може бути розширена за допомогою тем і плагінів. Крім того, WordPress підтримує вбудовану систему ролей і прав доступу, що робить його зручним рішенням для типових вебресурсів. [2; 3].

Разом із тим для проєктів, у яких необхідно реалізувати складні моделі даних, індивідуальні типи сторінок, нетипові контентні структури та глибоку інтеграцію із прикладною логікою, WordPress може виявитися менш гнучким. Значна залежність від сторонніх плагінів спрощує реалізацію базових функцій, але водночас може ускладнювати підтримку, контроль цілісності архітектури та подальше масштабування програмного продукту. Для університетського сайту, який має багаторівневу структуру сторінок, різні типи контенту та потребу в модульному повторному використанні блоків, це є суттєвим обмеженням.

Додатково для університетських сайтів суттєвим є питання стандартизації та зменшення дублювання контенту між різними підрозділами. У вебполітиці Campbell University прямо вказано на доцільність використання спільної CMS і єдиної кодової бази для всіх шкіл і департаментів, а також на потребу зменшення надлишкового контенту через продуману інформаційну архітектуру та регулярний супровід [20]. У цьому контексті використання платформи, яка значною мірою залежить від великої кількості сторонніх плагінів, може ускладнювати забезпечення єдності архітектури та довгострокової підтримки університетського вебресурсу.

Іншим поширеним аналогом є Drupal. Ця система орієнтована на створення більш складних, структурованих і масштабованих вебресурсів. Вона має розвинені механізми керування ролями та дозволами, а також гнучкі можливості побудови модульної структури сайту. Drupal традиційно вважається потужною платформою для великих вебсайтів із високими вимогами до безпеки, розмежування доступу та адміністрування [4].

Однак для реалізації програмного продукту, побудованого навколо індивідуально розробленої архітектури на Python та Django, Drupal є менш доцільним. Він використовує інший технологічний стек, має вищий поріг входу з точки зору налаштування та супроводу, а також не забезпечує природної інтеграції з Django-моделями та Python-логікою. Тому, незважаючи на його функціональну потужність, використання Drupal у межах даної роботи не є оптимальним.

Разом із тим Drupal залишається поширеним вибором для сектору вищої освіти саме завдяки своїй гнучкості, масштабованості та розвинутим механізмам керування контентом. У матеріалах Pantheon підкреслюється, що Drupal традиційно використовується багатьма закладами вищої освіти через підтримку складних структур сайтів, багаторівневих дозволів та розширюваної модульної архітектури [21]. Проте переваги цієї платформи супроводжуються вищою складністю впровадження та супроводу, що для проєкту, орієнтованого на поєднання CMS-рівня з фреймворком Django, робить її менш доцільною.

Як третій аналог доцільно розглянути Wagtail. На відміну від WordPress і Drupal, Wagtail побудований на Django та використовує модельно-орієнтований підхід до опису сторінок. Кожен тип сторінки в цій системі може бути представлений окремою Django-моделлю, що дає змогу застосовувати стандартні засоби фреймворку для опису полів, логіки та зв’язків між сутностями. Додатково Wagtail підтримує StreamField для формування змішаного контенту з окремих блоків, а також вбудовані засоби API, що дозволяє організовувати повторне використання даних і взаємодію між окремими компонентами системи [5-7].

Саме Wagtail найбільше відповідає вимогам розроблюваної системи, оскільки поєднує можливості CMS із гнучкістю фреймворкового підходу. На відміну від WordPress, де структура значною мірою залежить від тем і плагінів, у Wagtail контентні типи й логіка описуються на рівні моделей, що забезпечує кращий контроль над архітектурою системи. На відміну від Drupal, Wagtail

природно інтегрується з Django, що є важливим для реалізації модульного програмного продукту на Python.

Проведений аналіз дозволяє сформулювати потенційні конкурентні переваги розроблюваного програмного продукту. По-перше, система орієнтована не на універсальне масове використання, а на конкретну предметну область - керування вебконтентом закладу вищої освіти. Це дає можливість врахувати специфіку структури університетського сайту, типів сторінок і сценаріїв адміністрування. По-друге, модульний підхід забезпечує можливість повторного використання контентних блоків у різних частинах вебресурсу. По-третє, поєднання Django та Wagtail дозволяє реалізувати як зручне керування контентом, так і власну прикладну логіку без надмірної залежності від сторонніх розширень. По-четверте, підтримка API створює основу для повторного використання даних і взаємодії між окремими модулями системи, що було практично продемонстровано на прикладі блоку подій та анонсів [5-7].

Вибір платформи для університетського вебсайту також повинен враховувати не лише технічні можливості, а й вимоги до доступності, підтримки різних ролей редакторів, узгодженості дизайну та стабільності супроводу. У вебпротоколі University of Cincinnati зазначено, що вебавтори мають проходити обов’язкове навчання з доступності й керування контентом, а зміни інформаційної архітектури повинні координуватися в межах визначеної моделі вебкерування [22]. Це додатково підтверджує доцільність використання такої платформи, яка поєднує гнучкість побудови сторінок із можливістю централізованого адміністрування та контролю структури контенту.

Отже, проведений аналіз показує, що WordPress є зручним для швидкого розгортання типових сайтів, Drupal - для складних структурованих рішень із високими вимогами до адміністрування та безпеки, а Wagtail - для реалізації гнучких модульних вебсистем на базі Django. З урахуванням цілей кваліфікаційної роботи, а також вимог до модульності, масштабованості, централізованого керування контентом і можливості реалізації API-взаємодії,

найбільш доцільною платформою для розробки програмного продукту є Wagtail у поєднанні з фреймворком Django.

1.3 Постановка задачі

У межах кваліфікаційної роботи необхідно розробити модульну систему керування вебконтентом для офіційного сайту закладу вищої освіти. Програмний продукт повинен забезпечувати централізоване адміністрування інформаційних матеріалів, підтримку різних типів сторінок, структуроване зберігання даних, повторне використання контентних блоків та можливість подальшого розширення функціоналу. З урахуванням вимог методичних рекомендацій постановка задачі має охоплювати загальний опис задачі, системні, функціональні та нефункціональні вимоги, а також очікувані результати реалізації програмного продукту.

Основне призначення розроблюваної системи полягає в автоматизації процесів створення, редагування, публікації та відображення вебконтенту університетського ресурсу. На відміну від статичного підходу до побудови сайту, у якому зміни часто потребують прямого втручання в HTML-шаблони або програмний код, модульна CMS повинна дозволяти виконувати більшість операцій через адміністративний інтерфейс. Це спрощує супровід вебресурсу, зменшує ймовірність дублювання інформації та робить систему придатною до тривалого практичного використання.

У процесі розв’язання поставленої задачі програмний продукт повинен забезпечувати роботу з основними інформаційними сутностями університетського вебсайту. До них належать сторінки загального змісту, сторінки освітніх програм, секції з документами, контентні блоки, зображення, файли, а також модулі подій та анонсів. Важливою вимогою є підтримка блочного формування контенту, за якого сторінка не обмежується жорстко фіксованою структурою, а може складатися з окремих функціональних елементів, що додаються та змінюються незалежно один від одного.

Окрему увагу в межах даної роботи доцільно приділити повторному використанню даних і блоків. Практичним прикладом такого підходу є модуль подій та анонсів, у якому інформація про події зберігається централізовано, а далі може використовуватися як на окремій сторінці подій, так і у вигляді API-блоку на інших сторінках сайту.

До вхідних даних системи належать текстові матеріали сторінок, заголовки, підзаголовки, дати, описи, параметри контентних блоків, графічні матеріали, документи, посилання та службові атрибути, що впливають на порядок і спосіб відображення інформації. Вихідними даними є сформовані вебсторінки публічної частини сайту, структуровані інформаційні блоки, API-відповіді для повторного використання контенту, а також дані адміністративної частини, що відображають стан і структуру інформаційного наповнення системи.

З урахуванням призначення розроблюваного програмного продукту доцільно окремо визначити системні, функціональні та нефункціональні вимоги до модульної системи керування вебконтентом. Системні вимоги описують програмно-технічне середовище, необхідне для розгортання та функціонування системи. Функціональні вимоги визначають основні дії, які система повинна виконувати. Нефункціональні вимоги характеризують якісні властивості програмного продукту, зокрема його продуктивність, безпеку, доступність, масштабованість і зручність супроводу. (табл. 1.1).

Таблиця 1.1 - Вимоги до модульної системи керування вебконтентом

Група вимог	Зміст вимог	Практичне значення
Системні вимоги	Використання платформи на базі Python, Django та Wagtail; підтримка роботи з реляційною СУБД PostgreSQL; можливість розгортання системи в середовищі Windows; Використання браузера для різних типів доступу	Забезпечують технічну основу для стабільної роботи системи та її подальшого розгортання
Функціональні вимоги	Створення, редагування, публікація та видалення сторінок; формування вмісту сторінок із контентних блоків; керування подіями та анонсами; завантаження зображень і документів; повторне використання даних через API; підтримка ролей адміністратора та редактора	Функціональні вимоги
Нефункціональні вимоги	Забезпечення зручності адміністрування, стабільної роботи системи, захисту адміністративної частини, адаптивності інтерфейсу, доступності контенту для користувачів	

Джерело: створено автором на основі власних досліджень

Сформульовані вимоги дають змогу конкретизувати очікувані властивості розроблюваної системи та визначити критерії, яким повинен відповідати програмний продукт у процесі реалізації. Вони також узгоджуються із загальною

метою роботи, оскільки орієнтовані на створення гнучкої, зручної та масштабованої системи керування вебконтентом для сайту закладу вищої освіти.

З урахуванням особливостей предметної області, визначених вимог та функціональних можливостей платформи Wagtail основні напрями реалізації розроблюваної модульної системи керування вебконтентом наведено в таблиці 1.2.

Таблиця 1.2 - Основні напрями реалізації модульної системи керування вебконтентом

Напрям реалізації	Зміст	Практичне значення для сайту університету	Засоби реалізації
Структуровані типи сторінок	Подання різних розділів сайту у вигляді окремих моделей сторінок	Дозволяє створювати сторінки факультетів, кафедр, подій, програм і документів із власним набором полів	Django models, Wagtail Page
Напрям реалізації	Зміст	Практичне значення для сайту університету	Засоби реалізації
Блочне формування контенту	Побудова сторінок із незалежних контентних блоків	Дає змогу повторно використовувати окремі елементи, спрощує редагування та підтримку сторінок	StreamField, StructBlock, ListBlock
Централізоване керування даними	Зберігання та адміністрування контенту через єдину систему	Забезпечує цілісність даних, зменшує дублювання та спрощує супровід ресурсу	Адмін-панель Wagtail, база даних

Джерело: створено автором на основі власних досліджень

Розроблюваний програмний продукт повинен відповідати сучасним вимогам до вебсистем, зокрема підтримувати багаторівневу організацію, мати чітко відокремлені рівні зберігання даних, доступу до даних, прикладної логіки та користувацького інтерфейсу, а також бути придатним до подальшого тестування і розвитку. Такі вимоги також узгоджуються з методичними рекомендаціями до розробки програмного продукту, де окремо наголошується на необхідності багаторівневої архітектури та використання сучасних технологій.

Отже, постановка задачі полягає у створенні модульної системи керування вебконтентом, яка забезпечує структуроване формування сторінок, централізоване адміністрування даних, повторне використання контентних блоків і можливість API-взаємодії між окремими модулями. Результатом виконання роботи має стати працездатний програмний продукт, придатний для практичного використання як основа для керування інформаційним наповненням університетського вебресурсу.

Висновок до розділу

У першому розділі було розглянуто предметну область керування вебконтентом офіційного сайту закладу вищої освіти. Визначено, що університетський вебресурс є складною інформаційною системою, яка повинна забезпечувати структуроване подання різнорідного контенту, його постійне оновлення, централізоване адміністрування та зручну взаємодію з різними категоріями користувачів. Встановлено, що для розв’язання таких задач доцільним є використання системи керування вебконтентом [1].

У процесі аналізу існуючих аналогів було розглянуто платформи WordPress, Drupal та Wagtail. Виявлено, що WordPress є зручним для швидкого розгортання типових сайтів, Drupal - для складних структурованих рішень із розвиненим адмініструванням, а Wagtail найбільш повно відповідає вимогам розроблюваного програмного продукту завдяки інтеграції з Django, підтримці сторінкових моделей, блочного контенту та API-взаємодії [2-7].

У результаті постановки задачі визначено призначення програмного продукту, його основні функції, вхідні та вихідні дані, а також ключові напрями реалізації модульної системи керування вебконтентом. Отже, результати першого розділу створюють основу для подальшого етапу проєктування архітектури, структури та технологічних рішень програмного продукту, що розглядатимуться в наступному розділі.

У межах постановки задачі також було визначено системні, функціональні та нефункціональні вимоги до розроблюваної системи. Системні вимоги окреслюють програмно-технічне середовище функціонування програмного продукту, функціональні вимоги визначають основні можливості системи, а нефункціональні вимоги характеризують її якісні властивості, зокрема зручність адміністрування, безпеку, доступність, стабільність роботи та можливість подальшого масштабування.

Окремо було визначено основні напрями реалізації модульної системи керування вебконтентом, серед яких структуровані типи сторінок, блочне формування контенту та централізоване керування даними. Це дозволило конкретизувати практичну спрямованість програмного продукту та показати, що розроблювана система має забезпечувати не лише створення й редагування сторінок, а й повторне використання інформаційних компонентів у різних частинах вебресурсу.

РОЗДІЛ 2. ПРОЄКТУВАННЯ МОДУЛЬНОЇ СИСТЕМИ КЕРУВАННЯ ВЕБКОНТЕНТОМ

2.1 Моделювання функціонування системи

На етапі проєктування програмного продукту важливим завданням є визначення загальної логіки функціонування системи, її основних учасників та характеру взаємодії між ними. Для модульної системи керування вебконтентом це дає змогу встановити, яким чином здійснюється створення, редагування, збереження, публікація та повторне використання інформації в межах вебресурсу закладу вищої освіти.

Функціонування розроблюваної системи базується на поєднанні можливостей традиційної CMS та елементів API-взаємодії. З одного боку, система має адміністративний інтерфейс, через який користувачі працюють із вмістом сторінок, контентними блоками, зображеннями, документами та іншими інформаційними об’єктами. З іншого боку, окремі типи контенту можуть повторно використовуватися через API, що дозволяє відображати їх у різних частинах сайту без дублювання даних. Саме така логіка була реалізована в модулі подій та анонсів, де централізовано створені записи подій відображаються як на окремій сторінці, так і у вигляді API-блоку на інших сторінках сайту [5-7].

Для загального розуміння логіки функціонування систем керування контентом доцільно розглянути співвідношення традиційного та headless-підходів. У традиційній CMS контент, шаблони відображення, програмні модулі та база даних тісно поєднані в межах одного середовища. У headless-підході контент керується окремо, а його подання реалізується через API для різних клієнтських інтерфейсів. Узагальнену схему такого порівняння наведено на рисунку 2.1 [8].

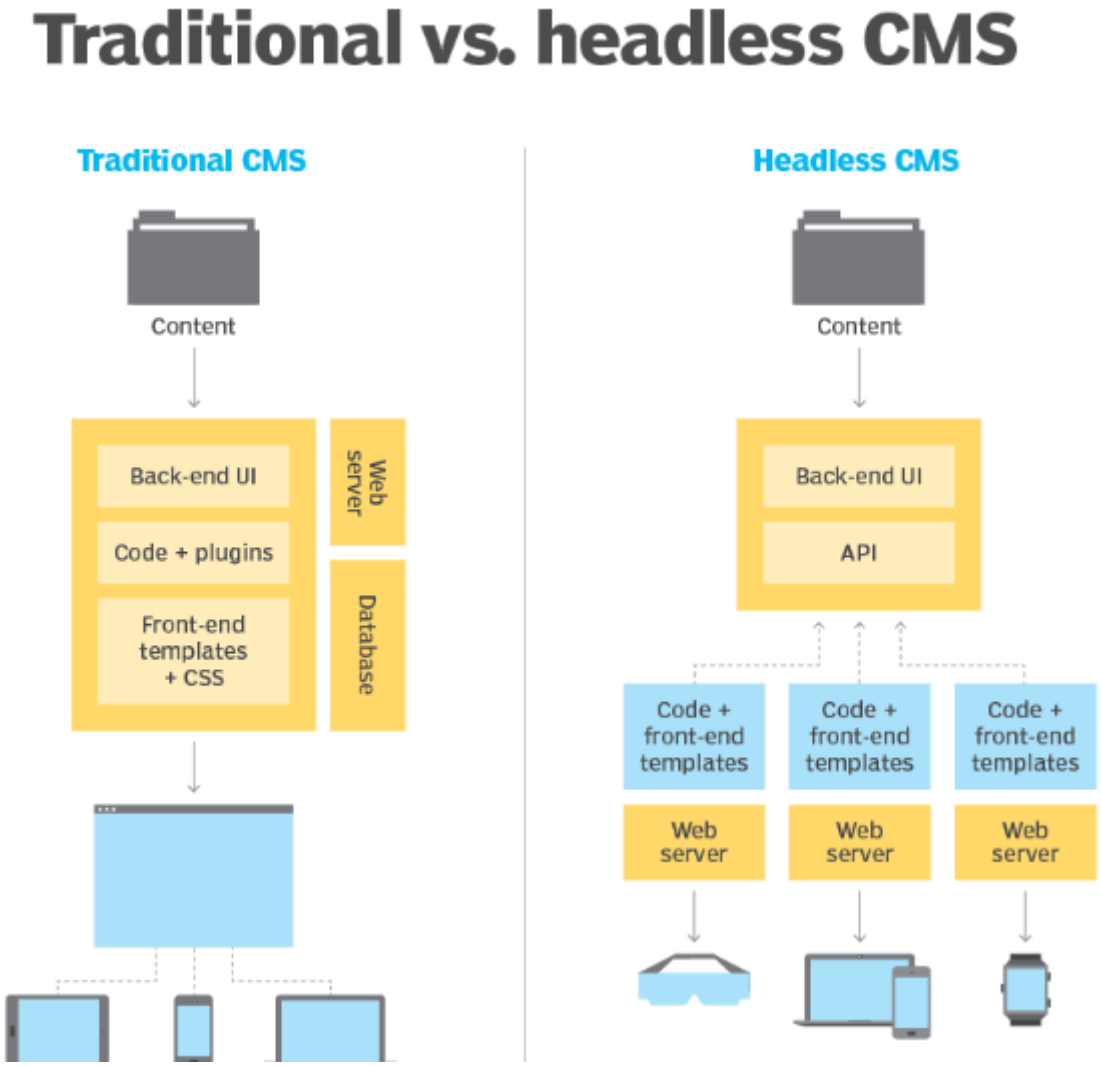

Рисунок 2.1 - Порівняння традиційної та headless CMS

У межах даної роботи реалізовано комбінований підхід: основна логіка керування контентом виконується в межах традиційної CMS на базі Wagtail, однак для окремих модулів використовується API-взаємодія. Це дозволяє поєднати зручність адміністративного керування сторінками із можливістю повторного використання структурованих даних у різних частинах системи.

Функціонування розроблюваної системи передбачає участь кількох основних категорій користувачів. До них належать адміністратор, редактор контенту, кінцевий користувач сайту та API-клієнт. Адміністратор відповідає за загальне налаштування системи, структуру сторінок, права доступу та технічне адміністрування. Редактор контенту працює з інформаційним наповненням сторінок, створює та оновлює матеріали. Кінцевий користувач взаємодіє лише з публічною частиною сайту та отримує доступ до опублікованого контенту. API-

клієнт виступає засобом отримання даних із системи для повторного відображення в інших модулях або інтерфейсах. У Django передбачено вбудовану систему користувачів, груп і дозволів, а Wagtail використовує та розширює ці механізми для адміністрування контенту [9; 10].

Основні учасники функціонування системи та їхні дії розглянуті в табл. 2.1.

Таблиця 2.1 - Основні учасники системи та їхні функції

Учасник системи	Основні функції	Результат взаємодії	Рівень доступу
Адміністратор	Налаштування структури сайту, керування користувачами, ролями та правами доступу	Підготовлена до роботи та налаштована CMS	Повний
Редактор контенту	Створення, редагування та публікація сторінок, блоків, подій,	Актуальне інформаційне наповнення сайту	Обмежений відповідно до ролі
Кінцевий користувач	Перегляд сторінок, новин, подій, документів та інших матеріалів	Доступ до опублікованого контенту	Публічний
API-клієнт	Отримання структурованих даних для повторного використання в інших модулях	JSON-відповіді та повторне відображення контенту	Залежить від налаштувань API

Джерело: створено автором на основі власних досліджень

Таким чином, моделювання функціонування системи показує, що розроблюваний програмний продукт є багатокомпонентною системою, у якій

поєднуються централізоване керування контентом, ролі користувачів, публічна частина вебсайту та API-рівень для повторного використання даних. Отримані результати є основою для подальшого проєктування архітектури та внутрішньої структури системи.

2.2 Проєктування архітектури та структури системи

Проєктування архітектури програмного продукту є одним із ключових етапів розробки модульної системи керування вебконтентом, оскільки саме на цьому етапі визначаються основні структурні компоненти системи, їх призначення, взаємозв’язки та загальна логіка функціонування. Для розроблюваного програмного продукту було обрано архітектурний підхід, заснований на використанні фреймворку Django та CMS Wagtail, що дозволяє поєднати модельно-орієнтоване описання структури сторінок із централізованим керуванням контентом [11; 14].

У межах розробленої системи архітектура побудована за багаторівневим принципом. На рівні зберігання даних використовуються моделі Django, що описують сторінки, інформаційні сутності та зв’язки між ними. На рівні прикладної логіки реалізуються механізми обробки контенту, API-взаємодії та правила доступу. Рівень представлення включає HTML-шаблони, стилі оформлення та клієнтські сценарії, що відповідають за відображення даних у публічній та адміністративній частинах системи. Така організація забезпечує розмежування відповідальності між окремими складовими програмного продукту та спрощує його супровід [11; 14].

Ключовою особливістю структури системи є використання сторінкової моделі Wagtail. У документації платформи зазначається, що кожен тип сторінки в Wagtail є окремою Django-моделлю, представлено в базі даних окремою таблицею і може мати власний набір полів [11]. Це дозволяє подати різні розділи сайту у вигляді структурованих сутностей із власною логікою та параметрами. У

межах даної роботи такий підхід було використано для реалізації головної сторінки, контентних сторінок, сторінки списку подій і окремої сторінки події.

Для наочного подання взаємозв’язків між основними сторінковими моделями та контентними блоками розроблюваної системи доцільно використати UML-діаграму класів. Вона відображає ключові сутності програмного продукту, зокрема базовий клас сторінки, головну сторінку, універсальну контентну сторінку, сторінку-розділ подій, окрему сторінку події та стандартний набір контентних блоків. Така діаграма дозволяє показати, що структура системи побудована на основі успадкування від базового класу Wagtail Page, а вміст окремих сторінок формується за допомогою набору незалежних блоків StreamField.

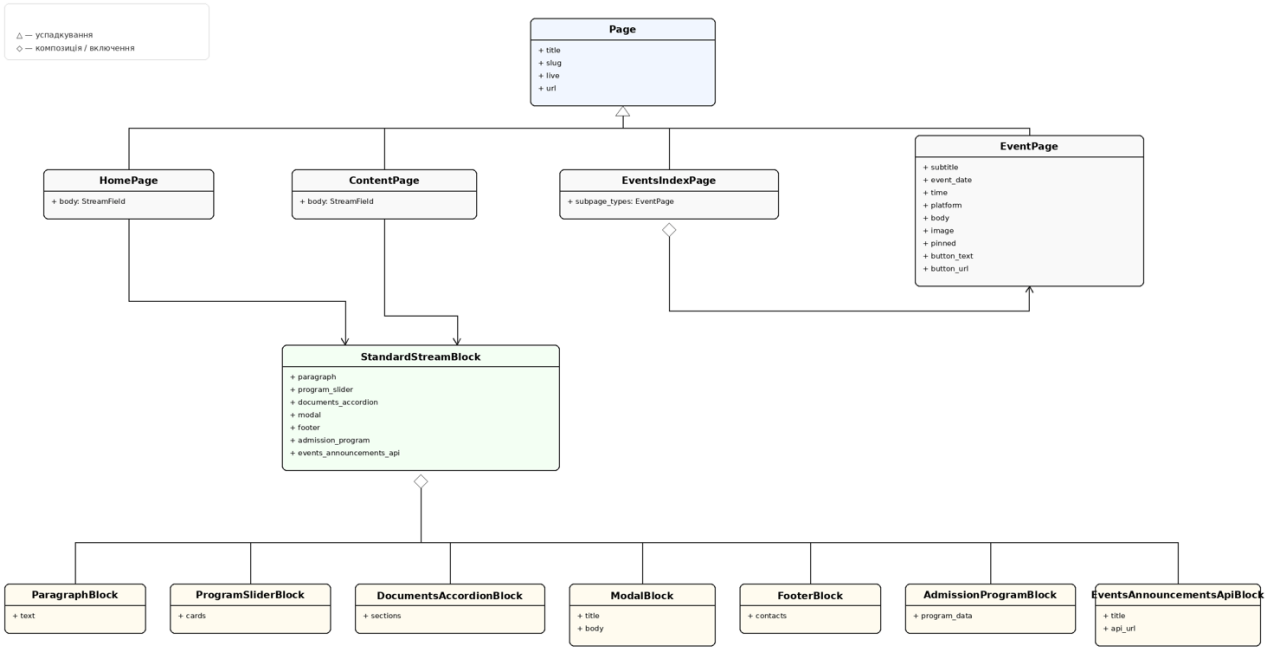

Рисунок 2.2 - UML-діаграма класів модульної системи керування вебконтентом

Як видно з рисунка 2.2, основні сторінкові моделі HomePage, ContentPage, EventsIndexPage та EventPage успадковуються від базового класу Page. Моделі HomePage і ContentPage використовують поле body на основі StreamField, що забезпечує формування сторінок із набору контентних блоків. Сторінка EventsIndexPage виконує роль контейнера для дочірніх сторінок EventPage, які

містять структуровані дані про окремі події. Окремо на діаграмі показано StandardStreamBlock, який об’єднує основні блоки системи, зокрема ParagraphBlock, ProgramSliderBlock, DocumentsAccordionBlock, ModalBlock, FooterBlock, AdmissionProgramBlock та EventsAnnouncementsApiBlock.

Для забезпечення гнучкого формування вмісту сторінок у системі використано механізм StreamField. Wagtail визначає StreamField як модель редагування контенту, що підходить для сторінок без жорстко фіксованої структури, де вміст може складатися з різних блоків, які повторюються та комбінуються в довільному порядку [12]. Саме цей підхід був покладений в основу побудови сторінок у розробленій системі, де окремі інформаційні елементи реалізовано як незалежні блоки з власними шаблонами відображення. Це дає змогу редагувати структуру сторінки через адміністративний інтерфейс без зміни програмного коду.

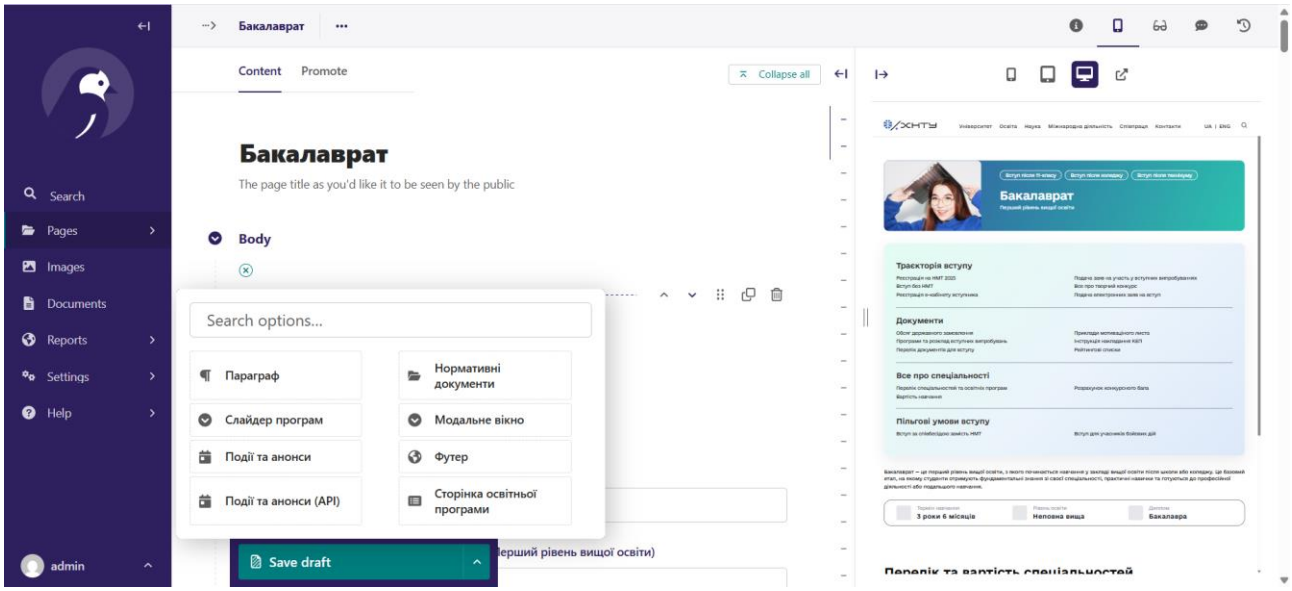

Рисунок 2.2 - Формування структури сторінки в адміністративній частині модульної системи керування вебконтентом

Окреме значення в архітектурі системи має модульний підхід до побудови контентних компонентів. У межах програмного продукту це реалізовано через набір блоків, що входять до стандартної структури сторінки. До них належать текстові блоки, слайдери, секції документів, блоки освітніх програм, модальні вікна та блоки подій і анонсів. Кожен такий блок має власну внутрішню

структуру, параметри та шаблон відображення, що дозволяє забезпечити повторне використання компонентів на різних сторінках сайту. Такий підхід підвищує гнучкість системи та спрощує її розширення в майбутньому [12].

Для модуля подій та анонсів у роботі реалізовано окремий архітектурний підхід. Якщо початково такий блок міг існувати як набір елементів усередині конкретної сторінки, то в межах удосконалення структури системи було запроєктовано централізовану модель подій із власною сторінкою-розділом, сторінками окремих подій та API-рівнем. У документації Wagtail API вказано, що платформа має вбудовані засоби для налаштування публічного API, який може використовуватися для надання контенту зовнішнім клієнтам або іншим компонентам системи [13]. Це дозволило реалізувати API-блок подій, який повторно використовує централізовані дані на різних сторінках сайту без дублювання інформації.

Важливим складником архітектури є також організація доступу до системи. Wagtail використовує механізми прав доступу та розширює підходи Django для адміністрування контенту, що дозволяє організувати роботу адміністраторів і редакторів з різними ролями [15]. У межах розроблюваного програмного продукту це дає змогу відокремити технічне адміністрування системи від операційного керування контентом, що є важливим для практичного використання CMS в умовах університетського вебресурсу.

Для узагальнення основних елементів архітектури розроблюваної системи доцільно виокремити її ключові структурні компоненти, які забезпечують зберігання, обробку, відображення та повторне використання вебконтенту. До таких компонентів належать моделі сторінок, контентні блоки, шаблони відображення та API-рівень. Кожен із них виконує окрему функцію в межах загальної архітектури програмного продукту, а їхня взаємодія забезпечує цілісність і гнучкість системи. Основні структурні компоненти модульної системи керування вебконтентом (табл 2.2)

Таблиця 2.2 - Основні структурні компоненти модульної системи керування вебконтентом

Компонент системи	Призначення	Приклад реалізації в проєкті	Технологічна основа
Моделі сторінок	Описують типи сторінок і їх поля	HomePage, ContentPage, EventsIndexPage, EventPage	Django models, Wagtail Page
Контентні блоки	Формують вміст сторінок із незалежних елементів	ParagraphBlock, EventsAnnouncementsApiBlock, FooterBlock	StreamField, StructBlock, ListBlock
Шаблони відображення	Забезпечують візуальне представлення даних	home_page.html, event_page.html, events_announcements_api.html	Django Templates, HTML, CSS, JavaScript
API-рівень	Повторне використання будь-чого	/api/events/, API-блок подій та анонсів	Wagtail API, Django views, JSON

Джерело: створено автором на основі власних досліджень

Крім класової структури, архітектуру програмного продукту доцільно подати у вигляді UML-діаграми компонентів. Така діаграма відображає основні програмні складові системи та взаємодію між ними. У межах розроблюваної модульної CMS до основних компонентів належать адміністративна частина Wagtail, сторінкові моделі Django та Wagtail, база даних PostgreSQL, шаблони Django, публічна частина сайту, API-модуль подій та клієнтська JavaScript-логіка.

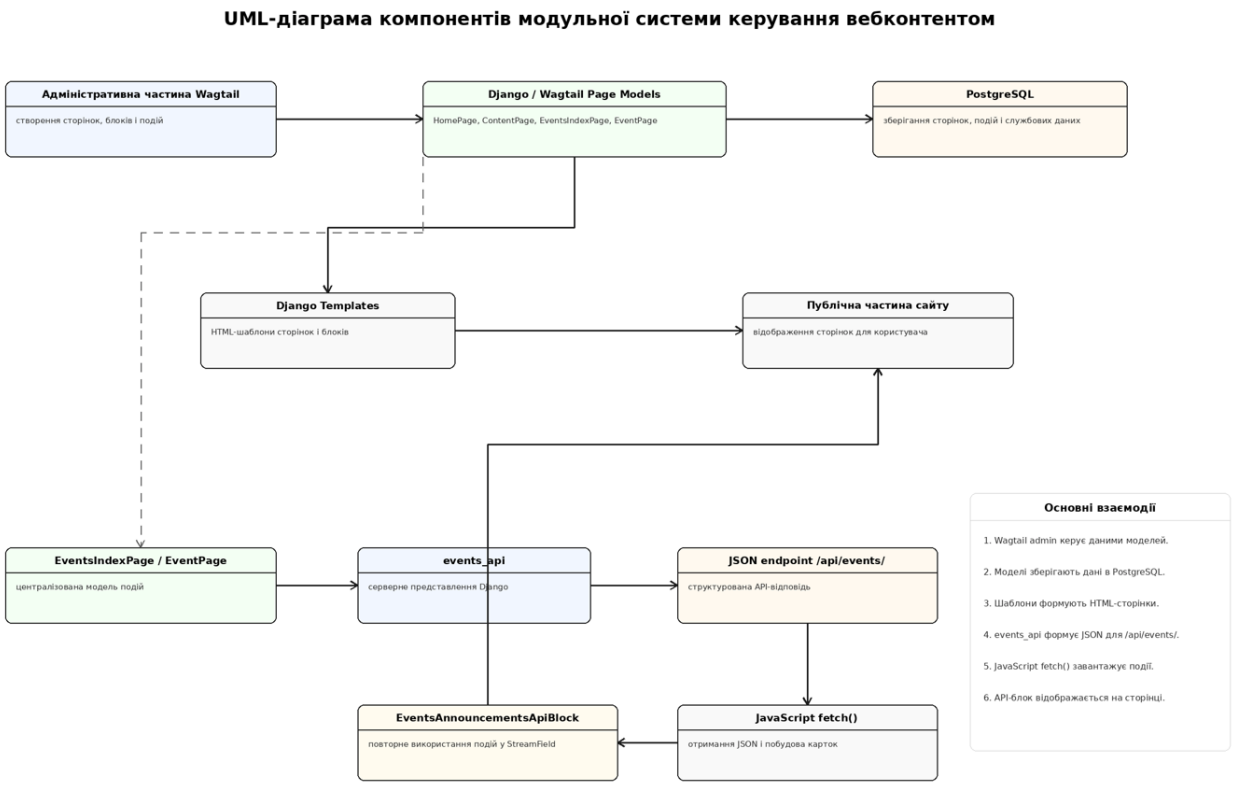

Рисунок 2.3 - UML-діаграма компонентів модульної системи керування вебконтентом

На рисунку 2.3 показано загальну взаємодію компонентів розроблюваної системи. Адміністративна частина Wagtail використовується редактором або адміністратором для створення сторінок, подій і контентних блоків. Дані зберігаються у моделях Django та Wagtail, а фізичне збереження інформації забезпечується базою даних PostgreSQL. Для формування публічної частини сайту моделі передають дані до шаблонів Django, які генерують HTML-сторінки для кінцевого користувача. Окремо на діаграмі показано API-ланцюг роботи модуля подій: дані зі сторінок EventsIndexPage та EventPage обробляються представленням events_api, передаються через endpoint /api/events/ у форматі JSON, після чого JavaScript-код отримує ці дані за допомогою fetch() і відображає їх у блоці EventsAnnouncementsApiBlock.

Приклад реалізації структурного підходу до формування сторінки в адміністративній частині системи наведено на рисунку 2.2. Поданий інтерфейс демонструє, що вміст сторінки формується з окремих незалежних блоків, які можуть додаватися, редагуватися та змінювати порядок розташування без

втручання у програмний код. Такий підхід забезпечує гнучкість побудови сторінок і спрощує подальший супровід системи.

Отже, у результаті проєктування архітектури та структури системи було визначено основні компоненти модульної CMS, їх призначення та взаємозв'язки. Побудована структура поєднує сторінкові моделі, блочний підхід до формування контенту, шаблонний рівень представлення, механізми API-взаємодії та розмежування доступу. Це створює основу для подальшої реалізації програмного продукту та забезпечує його гнучкість, масштабованість і зручність супроводу [11-15]

2.3 Обґрунтування вибору технологій та опис алгоритмів роботи системи

Під час розробки модульної системи керування вебконтентом важливим етапом є вибір технологій, інструментів і підходів, які забезпечують не лише поточну працездатність програмного продукту, а й можливість його подальшого розширення, супроводу та адаптації до нових вимог. Для реалізації системи у межах даної кваліфікаційної роботи було обрано стек технологій, основу якого становлять мова програмування Python, вебфреймворк Django та CMS Wagtail.

Вибір Python як базової мови програмування зумовлений її придатністю до швидкої та читабельної реалізації серверної логіки вебзастосунків. В офіційній документації Python підкреслюється, що мова надає базові засоби, необхідні для ефективної розробки прикладних програм, а її синтаксис і структура сприяють зрозумілості коду [23]. Для даного проєкту це має практичне значення, оскільки система містить як моделі даних, так і шаблони, представлення та допоміжну логіку, що повинні залишатися підтримуваними в процесі подальшого розвитку.

Використання Django зумовлене тим, що цей фреймворк надає цілісні засоби для побудови вебзастосунків, зокрема модель даних, маршрутизацію, шаблонізацію, механізми роботи з базою даних та вбудовані засоби автентифікації й адміністрування. У документації Django зазначається, що

модель є основним джерелом інформації про дані та визначає поля і поведінку сутностей, які зберігаються в системі [11]. Це робить Django придатним для побудови структурованої системи, у якій різні типи сторінок, подій та контентних сутностей описуються на рівні окремих моделей.

Додатковою перевагою Django є чітке розмежування між маршрутизацією запитів, логікою представлень і шаблонним рівнем. Офіційна документація вказує, що URL dispatcher забезпечує зіставлення шляхів URL із відповідними представленнями, а шаблонна система дозволяє відокремити програмну логіку від рівня візуального подання даних [25]. У межах даної роботи це дозволило організувати окремо маршрути для сторінок і API, серверні представлення для формування HTML та JSON-відповідей, а також шаблони для сторінок і контентних блоків.

Вибір Wagtail як CMS-рівня обґрунтовується його тісною інтеграцією з Django та підтримкою модельно-орієнтованого підходу до побудови сайту. Wagtail використовує сторінки як окремі типи моделей, успадковані від базового класу Page, що дозволяє описувати різні розділи сайту у вигляді незалежних структурованих сутностей [12]. Для розроблюваного програмного продукту це є важливою перевагою, оскільки дозволяє окремо реалізувати головну сторінку, контентні сторінки, сторінку списку подій та сторінки окремих подій, зберігаючи при цьому цілісність загальної структури вебресурсу.

Ще однією причиною вибору Wagtail є підтримка StreamField, який призначений для створення сторінок зі змішаним контентом, де окремі блоки можуть комбінуватися у довільному порядку [13]. Для університетського сайту такий підхід є доцільним, оскільки сторінки часто мають різну структуру й можуть містити текстові секції, списки документів, інформаційні блоки, події, оголошення та інші елементи. Саме тому в межах проєкту було реалізовано набір власних блоків, які формують зміст сторінок у межах StreamField. Це дало змогу забезпечити модульність системи, повторне використання блоків та спрощення редагування контенту через адміністративний інтерфейс.

Для зберігання даних у проєкті використовується реляційна база даних PostgreSQL, яка добре інтегрується з Django ORM та забезпечує надійну роботу із структурованими сутностями системи. На рівні представлення інформації використовуються HTML-шаблони, CSS та JavaScript. HTML і CSS забезпечують структурне та візуальне подання контенту, а JavaScript застосовується для реалізації динамічних елементів, зокрема API-блоку подій і модального вікна.

Використання PostgreSQL є доцільним не лише з погляду сумісності з Django ORM, а й завдяки підтримці сучасних типів даних і засобів роботи зі структурованою інформацією. В офіційній документації PostgreSQL зазначено, що система підтримує типи json і jsonb, а також функції й оператори для роботи з JSON-даними [28; 29]. Хоча в межах цього проєкту API-відповідь формується на рівні Django-представлення, підтримка структурованих даних у СУБД загалом відповідає логіці побудови сучасних вебсистем, де важливими є гнучкість моделі даних і придатність до обміну інформацією у форматі JSON.

Основні технології, використані під час розробки системи, та їх роль у проєкті наведено в таблиці 2.3.

Таблиця 2.3 - Основні технології, використані для реалізації системи

Технологія	Призначення	Причина вибору	Приклад використання в проєкті
Python	Основна мова програмування	Високий рівень абстракції, широка підтримка веброзробки	серверна логіка, моделі, представлення

Продовження таблиці 2.3

Технологія	Призначення	Причина вибору	Приклад використання в проєкті
PostgreSQL	Система керування базою даних	надійність, сумісність із Django, масштабованість	зберігання сторінок, подій, службових даних
HTML / CSS	Формування та оформлення інтерфейсу	стандартні засоби представлення вебконтенту	шаблони сторінок і блоків
JavaScript	Клієнтська інтерактивність	підтримка динамічної взаємодії без перезавантаження сторінки	API-блок подій, модальне вікно
JSON / API	Обмін структурованими даними	повторне використання контенту між компонентами	endpoint /api/events/
Wagtail	CMS-рівень системи	підтримка Page models, StreamField, адмінки та API	CMS-структура сторінок, блоки, керування контентом
Django	Серверний вебфреймворк	ORM, маршрутизація, шаблони, система доступу	моделі сторінок, views, налаштування проєкту

Джерело: створено автором на основі власних досліджень

Особливе значення в реалізації системи має використання сторінкової моделі Wagtail. Кожен тип сторінки описується як окрема модель, що дозволяє чітко розмежувати різні сутності системи. Наприклад, у проєкті окремо реалізовано головну сторінку, універсальні контентні сторінки, сторінку списку подій і окрему сторінку події. Такий підхід забезпечує гнучке моделювання структури сайту та дозволяє задавати для різних типів сторінок власні поля, логіку вкладеності та правила відображення.

Для формування вмісту сторінок використано механізм StreamField, який є одним із ключових інструментів Wagtail. Його застосування дозволило відмовитися від жорстко зафіксованої структури сторінки та перейти до модульного принципу побудови контенту. У межах проєкту сторінки формуються з незалежних блоків, серед яких текстові секції, слайдери, секції документів, футерні блоки, блоки освітніх програм, модальні вікна та модулі подій. Це значно підвищує гнучкість редагування контенту та полегшує повторне використання компонентів.

Одним із найважливіших результатів розробки стало впровадження API-підходу для модуля подій та анонсів. Спочатку блок подій був реалізований як локальний контентний блок усередині сторінки, однак такий підхід обмежував можливості повторного використання даних. У процесі вдосконалення архітектури було реалізовано окрему модель події, сторінку-розділ подій, представлення API та API-блок для відображення подій на сторінках сайту. Це дозволило перейти від локального зберігання подій у межах окремої сторінки до централізованої моделі даних.

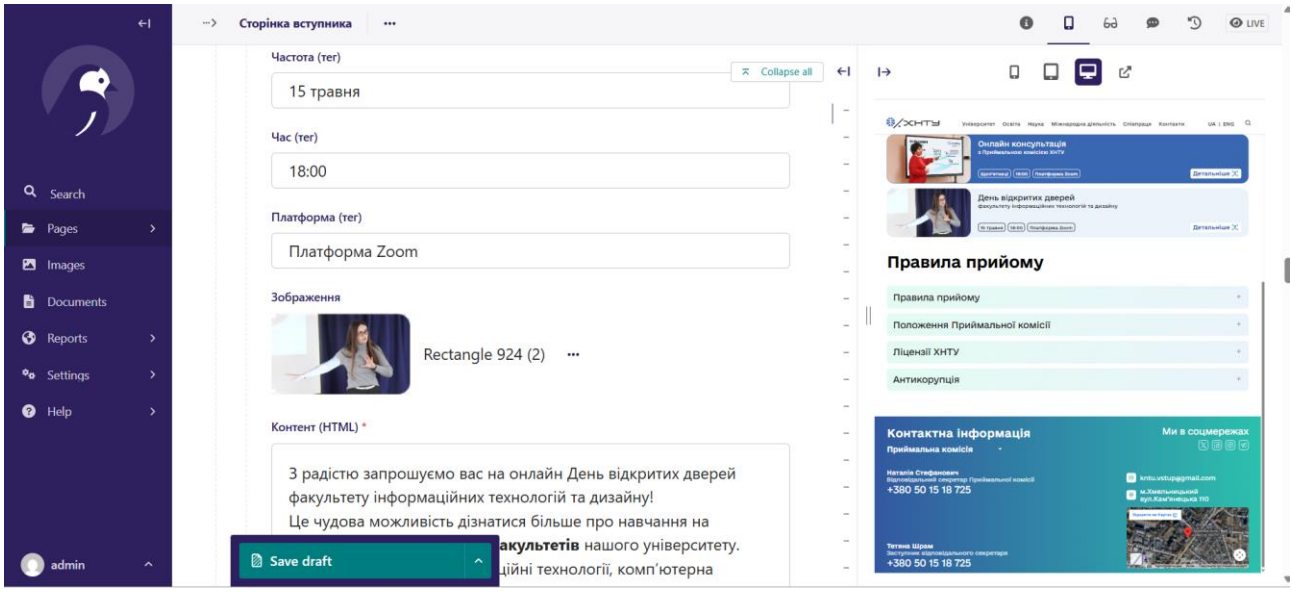

Рисунок 2.3 - Редагування контенту в адміністративній частині модульної системи керування вебконтентом

Крім вибору технологій, важливим є опис алгоритмів роботи основних модулів системи. У даній роботі доцільно виділити загальний алгоритм формування сторінки, алгоритм роботи блочного контенту та алгоритм API-взаємодії для модуля подій.

Загальний алгоритм функціонування системи можна подати так:

1. Адміністратор або редактор створює чи редагує контент у CMS;
2. Дані зберігаються у відповідних моделях бази даних;
3. Під час запиту до сторінки система звертається до моделей і шаблонів;
4. Контент збирається з окремих блоків та передається в шаблон відображення;
5. Користувач отримує сформовану сторінку у браузері.

Алгоритм роботи модуля подій та анонсів має дещо складнішу структуру:

1. У системі створюється сторінка-розділ подій;
2. Редактор додає окремі події як дочірні сторінки відповідного типу;
3. Події зберігаються централізовано в базі даних;
4. API-представлення повертає перелік подій у форматі JSON;
5. API-блок звертається до endpoint і отримує структуровані дані;
6. Отримані події динамічно відображаються на потрібній сторінці;
7. При натисканні на кнопку детальніше відкривається модальне вікно з повною інформацією.

На клієнтському рівні взаємодія з API реалізується за допомогою механізму fetch(). У документації MDN зазначається, що Fetch API надає інтерфейс для отримання ресурсів через мережу, а метод Response.json() дозволяє прочитати тіло відповіді та перетворити його на JavaScript-об’єкт [30].

У межах розроблюваного програмного продукту це дало змогу реалізувати завантаження подій без перезавантаження сторінки та динамічне формування карток і модального вікна на клієнтському рівні.

Основні алгоритми роботи розроблюваної системи, їх вхідні дані, ключові дії та результати наведено в таблиці 2.4.

Таблиця 2.4 - Основні алгоритми роботи модульної системи

Алгоритм	Вхідні дані	Основні дії	Результат
Формування сторінки	структура сторінки, блоки, шаблон	зчитування даних з моделі, передача в шаблон, рендеринг	готова HTML-сторінка
Робота блочного контенту	набір блоків StreamField	послідовна обробка блоків і підключення шаблонів	гнучка структура сторінки
Отримання подій через API	запит до /api/events/	вибірка подій, формування JSON-відповіді	структуровані дані для повторного використання
Відображення API-блоку подій	JSON-дані, шаблон, JS	завантаження даних, побудова карток, відкриття модалки	інтерактивний блок подій на сторінці

Джерело: створено автором на основі власних досліджень

Наведені в таблиці 2.4 алгоритми відображають загальну логіку функціонування ключових модулів розроблюваної системи та показують послідовність обробки даних на різних рівнях її роботи. Практична реалізація одного з таких алгоритмів, а саме взаємодії з API модуля подій та анонсів, проілюстрована на рисунку 2.4.

Рисунок 2.4 - Приклад результату API-взаємодії модуля подій та анонсів

Таким чином, обраний набір технологій забезпечує побудову модульної, розширюваної та придатної до супроводу системи керування вебконтентом. Використання Django та Wagtail дозволяє поєднати переваги серверного фреймворку та повноцінної CMS, а реалізовані алгоритми роботи сторінок, блоків і API-модулів формують цілісну основу для функціонування програмного продукту. Результати цього етапу проєктування створюють безпосередню основу для практичної реалізації системи, що буде розглянута в наступному розділі.

Математична формалізація процесу формування контентних блоків

Для більш формалізованого опису логіки роботи розроблюваної системи доцільно подати процес формування сторінки на основі контентних блоків із використанням елементів теорії множин та алгоритмічного підходу. Такий опис дозволяє показати, що сторінка в межах системи не є фіксованою структурою, а формується як впорядкована сукупність окремих незалежних компонентів, кожен із яких має власний тип, набір параметрів і шаблон відображення.

Нехай множина всіх доступних типів контентних блоків у системі визначається так:

$$B=\{b_1,b_2,b_3,\ldots,b_n\}$$

де B - множина типів блоків, а b_i - окремий тип блоку, наприклад текстовий блок, слайдер, блок документів, футерний блок або API-блок подій.

Тоді конкретну сторінку можна подати у вигляді впорядкованої послідовності блоків:

$$P=\langle x_1,x_2,x_3,\ldots,x_m\rangle$$

де P - сторінка, а $x_i \in B$ - блок певного типу, розміщений на відповідній позиції в межах сторінки. На відміну від звичайної множини, тут важливим є саме порядок елементів, оскільки розташування блоків впливає на підсумкове представлення сторінки для користувача.

Кожен блок може бути описаний як пара

$$x_i = (t_i, A_i)$$

де t_i - тип блоку, а A_i - множина його атрибутів. Наприклад, для текстового блоку до множини атрибутів можуть входити текст, вирівнювання, розмір шрифту та стилі оформлення, а для API-блоку подій - заголовок секції та адреса endpoint.

Процес побудови сторінки можна подати у вигляді функції рендерингу

$$R(P) = r(x_1) \oplus r(x_2) \oplus \ldots \oplus r(x_m)$$

де $R(P)$ - результат формування сторінки, $r(x_i)$- результат відображення окремого блоку через відповідний шаблон, а символ $\oplus$ позначає операцію послідовного об'єднання HTML-фрагментів у єдину вебсторінку.

Для API-блоку подій та анонсів математичну модель можна подати окремо. Нехай множина подій визначається як

$$E = \{e_1, e_2, e_3, \ldots, e_k\}$$

де E - множина подій, а e_j - окрема подія з її атрибутами, такими як назва, підзаголовок, дата, час, платформа, опис і зображення.

Тоді серверне API-представлення можна розглядати як відображення

$$F: E \to J$$

де F - функція перетворення множини подій у JSON-подання, а J - структурований набір даних, придатний до передачі клієнтській частині системи. На стороні клієнта цей результат далі перетворюється у множину візуальних елементів сторінки:

$$V(J) \rightarrow UI$$

де V - функція побудови інтерфейсу на основі отриманих JSON-даних, а UI - сформований блок подій у користувацькому інтерфейсі.

З алгоритмічної точки зору процес формування сторінки з блоків можна подати так:

1. Зчитати впорядкований список блоків сторінки;
2. Для кожного блоку визначити його тип;
3. Вибрати відповідний шаблон відображення;
4. Передати атрибути блоку в шаблон;
5. Отримати HTML-фрагмент для кожного блоку;
6. Послідовно об’єднати отримані фрагменти в єдину сторінку.

Аналогічно алгоритм роботи API-блоку подій включає:

1. Отримання множини подій із бази даних;
2. Перетворення подій у JSON-структуру;
3. Передавання JSON-відповіді клієнтській частині;
4. Обробку відповіді JavaScript-кодом;
5. Побудову карток подій;
6. Відображення детальної інформації в модальному вікні за запитом користувача.

Подана формалізація підтверджує, що процес побудови вебсторінки в межах модульної системи керування вебконтентом може бути описаний не лише на рівні програмної реалізації, а й у вигляді впорядкованих множин, функцій відображення та послідовності алгоритмічних дій. Це відповідає вимозі застосування математичних методів і алгоритмічних принципів у процесі моделювання та розробки програмного продукту.

Висновок до розділу 2

У другому розділі було виконано проєктування модульної системи керування вебконтентом та визначено основні підходи до її побудови. У підрозділі 2.1 змодельовано функціонування системи, встановлено основних учасників взаємодії з програмним продуктом та описано загальну логіку поєднання традиційного CMS-підходу з API-взаємодією. Це дозволило окреслити основні сценарії роботи системи та роль окремих користувачів у процесі керування вебконтентом [8-10].

У підрозділі 2.2 було спроєктовано архітектуру та структуру системи. Визначено, що програмний продукт доцільно будувати на основі сторінкових моделей Wagtail, блочного підходу до формування контенту, шаблонного рівня представлення, API-компонентів та механізмів розмежування доступу. У результаті встановлено основні структурні компоненти системи, їх призначення та взаємозв’язки, що створює основу для модульної та масштабованої архітектури [11-15].

У підрозділі 2.3 обґрунтовано вибір основних технологій реалізації програмного продукту, серед яких Python, Django, Wagtail, PostgreSQL, HTML, CSS, JavaScript та JSON. Також було описано алгоритми формування сторінок, роботи блочного контенту та API-модуля подій і анонсів. Додатково подано математичну формалізацію процесу формування сторінки на основі контентних блоків із використанням елементів теорії множин і функцій відображення. Отже, результати другого розділу визначають технологічну, архітектурну та формалізовану основу програмного продукту, а також забезпечують логічний перехід до практичної реалізації модульної системи керування вебконтентом, що розглядається в наступному розділі. [11-15].

У межах проєктування також було використано UML-діаграми класів і компонентів, що дозволило наочно подати внутрішню структуру програмного продукту та взаємодію його основних складових. UML-діаграма класів відображає зв’язки між сторінковими моделями, моделлю події та контентними

блоками, а UML-діаграма компонентів демонструє взаємодію адміністративної частини Wagtail, моделей Django, бази даних PostgreSQL, шаблонів, API-модуля та клієнтської JavaScript-логіки. Це дало змогу більш формалізовано описати архітектуру системи та показати її модульний характер.

Окрему увагу в розділі приділено механізму повторного використання контенту через API. На прикладі модуля подій та анонсів було показано, що інформація може зберігатися централізовано в межах окремих моделей, а потім передаватися до інших компонентів системи у форматі JSON. Такий підхід зменшує дублювання даних, спрощує супровід вебресурсу та створює основу для подальшого розширення функціоналу системи без суттєвого перепроєктування її архітектури.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ПРАКТИЧНЕ ЗАСТОСУВАННЯ МОДУЛЬНОЇ СИСТЕМИ КЕРУВАННЯ ВЕБКОНТЕНТОМ

3.1 Реалізація моделей даних і сторінкової структури системи

На етапі реалізації програмного продукту одним із головних завдань стало практичне втілення спроєктованої архітектури у вигляді моделей даних, сторінкової структури та адміністративних засобів керування вебконтентом. У межах даної кваліфікаційної роботи реалізація системи виконувалася на основі фреймворку Django та CMS Wagtail, що дозволило поєднати можливості модельно-орієнтованої розробки із засобами зручного адміністрування контенту [11; 12].

Як було визначено в попередньому розділі, основою структури системи є сторінкові моделі Wagtail. У документації платформи зазначено, що кожен тип сторінки реалізується як окрема Django-модель, яка успадковується від базового класу Page, а отже може мати власні поля, правила вкладеності та спосіб відображення [12]. У межах розробленого програмного продукту це рішення було використано для побудови ієрархічної структури сайту, де кожен розділ поданий у вигляді окремого типу сторінки.

На першому етапі реалізації було створено базові сторінкові моделі системи. До них належать головна сторінка HomePage, універсальна контентна сторінка ContentPage, сторінка-розділ подій EventsIndexPage та сторінка окремої події EventPage. Кожна з цих моделей виконує окрему функцію в межах загальної структури вебресурсу. Головна сторінка виконує роль точки входу до системи й містить набір інформаційних блоків, з яких формується основний вміст сайту. Контентні сторінки використовуються для побудови типових інформаційних розділів. Сторінка-розділ подій забезпечує зберігання та логічне групування сторінок окремих подій. Сторінка події призначена для подання

конкретного запису з відповідними параметрами, такими як дата, підзаголовок, час, платформа, опис та зображення.

У системі Django модель визначає структуру даних і поведінку сутності, а Wagtail надає додаткові механізми для інтеграції такої моделі в дерево сторінок та адміністративну панель [11; 12]. Це дало змогу не лише зберігати дані в базі, а й одразу організувати їх у вигляді ієрархічної структури вебсайту. Таким чином, структура контенту формується не лише логічно, а й фізично в межах дерева сторінок, що є важливим для систем керування вебконтентом.

Практична реалізація моделі HomePage та ContentPage базується на використанні поля StreamField, у якому розміщується основний вміст сторінки. Це поле дозволяє формувати сторінки з окремих незалежних блоків, що робить структуру контенту гнучкою та придатною до подальшого розширення [13]. У межах проєкту таке рішення дало змогу відмовитися від жорстко фіксованого макета сторінок і забезпечити можливість редагування структури сторінки без зміни програмного коду. З практичної точки зору це є важливою перевагою, оскільки редактор контенту може самостійно змінювати композицію сторінки, додаючи або вилучаючи окремі блоки через адміністративний інтерфейс.

Для сторінки події EventPage було реалізовано окремий набір полів, орієнтований на збереження структурованої інформації про подію. До таких полів належать ознака закріплення події, частота, час, платформа проведення, підзаголовок, основний опис, текст кнопки, посилання та дата події. Крім того, до моделі додано зв’язок із зображенням через ForeignKey на модель зображень Wagtail. Такий підхід забезпечує цілісне представлення події як окремої інформаційної сутності та створює основу для її повторного використання у шаблонах і API-відповідях.

Реалізація сторінкової структури системи тісно пов’язана з налаштуванням правил вкладеності сторінок. У Wagtail це здійснюється через атрибути parent_page_types та subpage_types, які визначають, які сторінки можуть бути дочірніми або батьківськими відносно інших [12]. У межах даної роботи було встановлено, що головна сторінка може містити контентні сторінки та сторінку-

розділ подій, а сторінка-розділ подій, у свою чергу, може містити лише сторінки окремих подій. Це дозволило забезпечити логічно впорядковану навігаційну структуру сайту та обмежити помилки під час створення нових сторінок в адміністративній частині системи.

Приклад реалізації сторінкової структури та дерева сторінок в адміністративній частині системи наведено на рисунку 3.1.

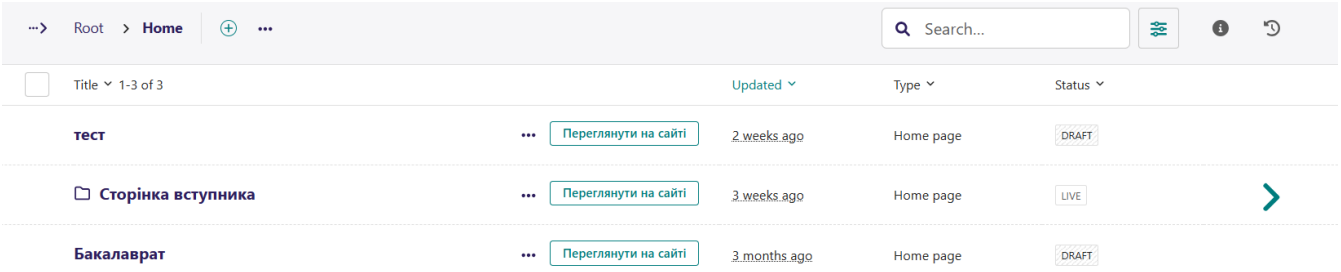

Рисунок 3.1 - Дерево сторінок у адміністративній частині модульної системи керування вебконтентом

Окремого розгляду потребує реалізація адміністративного інтерфейсу редагування сторінок. У Wagtail для цього використовуються панелі редагування, за допомогою яких визначається, які саме поля відображаються редактору під час роботи зі сторінкою. У межах проєкту для сторінок подій було налаштовано панелі редагування, що включають текстові поля, поле дати, зображення, логічні ознаки та інші параметри, необхідні для повноцінного опису події. Це дозволило сформувати зрозумілий інтерфейс, у межах якого редактор може послідовно заповнювати всі необхідні властивості сторінки.

Приклад реалізації інтерфейсу редагування окремої події наведено на рисунку 3.2.

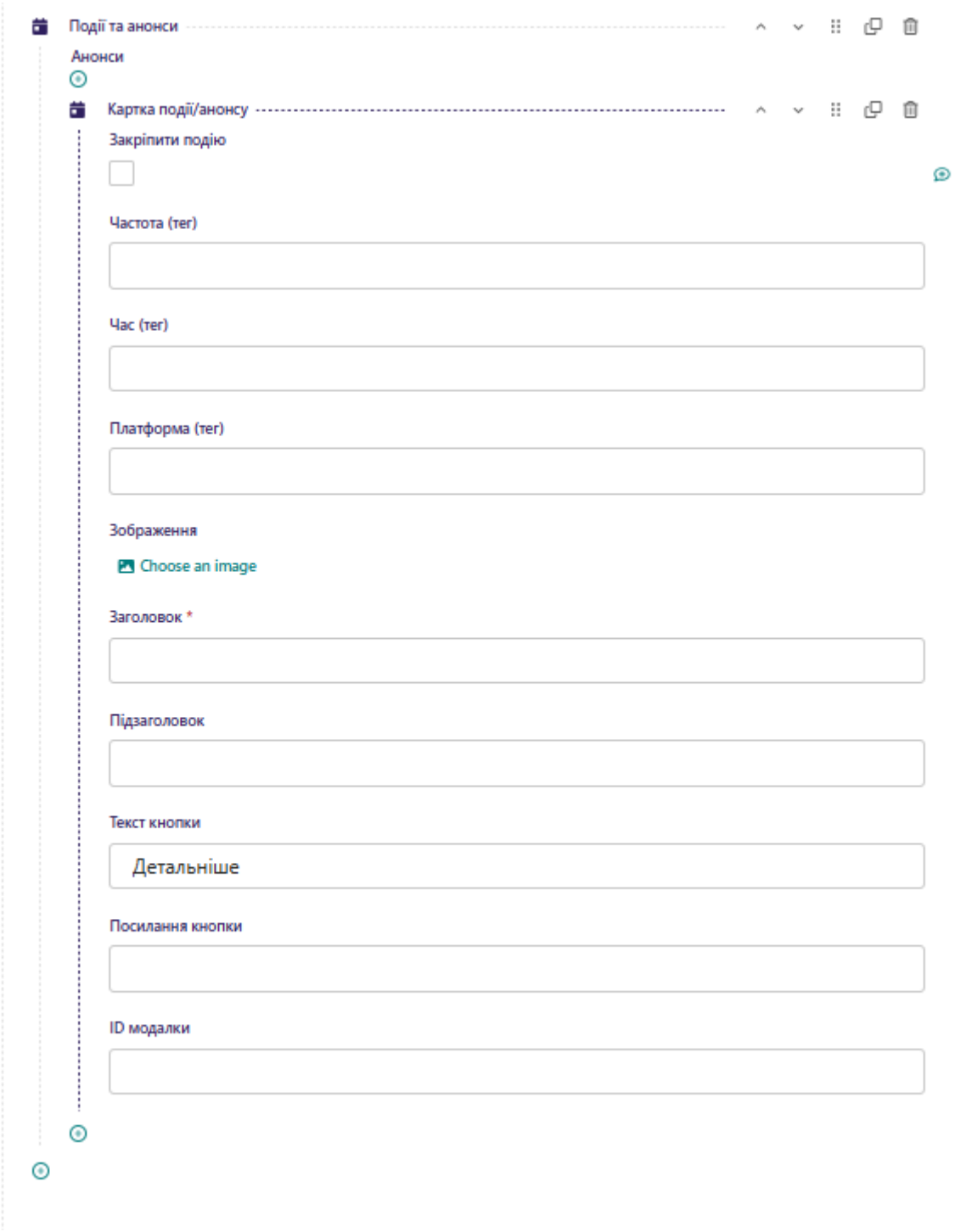

Рисунок 3.2 - Інтерфейс редагування сторінки події в адміністративній частині системи

Важливим аспектом реалізації стало також формування шаблонів відображення сторінок. Для кожного типу сторінки було створено відповідний HTML-шаблон, який визначає спосіб візуального представлення даних. Наприклад, для сторінки списку подій створено шаблон events_index_page.html,

що формує перелік наявних подій, а для окремої сторінки події - шаблон event_page.html, який відображає детальну інформацію про конкретний запис. Такий підхід відповідає стандартній логіці Django, де модель відповідає за дані, а шаблон - за представлення [11].

У межах реалізації було також забезпечено зв’язок між моделями сторінок і шаблонами блоків. Для сторінок, які використовують StreamField, вміст формується за рахунок окремих шаблонів блоків, що підключаються залежно від типу елемента. Це дозволяє не лише гнучко формувати вміст сторінки, а й підтримувати єдині стилістичні та структурні підходи до представлення контенту в межах усього вебресурсу.

Таблиця 3.1 - Реалізовані сторінкові моделі та їх призначення

Сторінкова модель	Призначення	Основні поля / особливості	Роль у системі
HomePage	Формування головної сторінки сайту	body (StreamField)	Точка входу до вебресурсу
ContentPage	Побудова універсальних інформаційних сторінок	body (StreamField)	Подання типових розділів сайту
EventsIndexPage	Групування та відображення списку подій	дочірні сторінки типу EventPage	Централізований розділ подій
EventPage	Зберігання та подання окремої події	title, subtitle, event_date, time, platform, body, image, button_url	Самостійна інформаційна сутність події

Джерело: створено автором на основі власних досліджень

З таблиці 3.1 видно, що реалізація сторінкових моделей забезпечує чітке розмежування функцій між окремими частинами системи. Такий підхід дозволяє

створити логічно впорядковану структуру вебресурсу, у межах якої кожна сторінка має власне призначення та набір властивостей. Крім того, використання окремих моделей сторінок позитивно впливає на підтримуваність системи, оскільки зміни в одному типі сторінки не потребують втручання в інші частини структури.

Ще одним важливим результатом реалізації є забезпечення зв’язку між адміністративною та публічною частинами системи. Дані, що вводяться та редагуються через адміністративну панель, автоматично використовуються для формування сторінок публічної частини сайту. Завдяки цьому редактор працює не з технічними файлами або кодом, а безпосередньо з інформаційними сутностями, що значно спрощує процес керування вебконтентом.

Для демонстрації результату реалізації сторінкової структури та її відображення в публічній частині сайту доцільно навести рисунок 3.3.

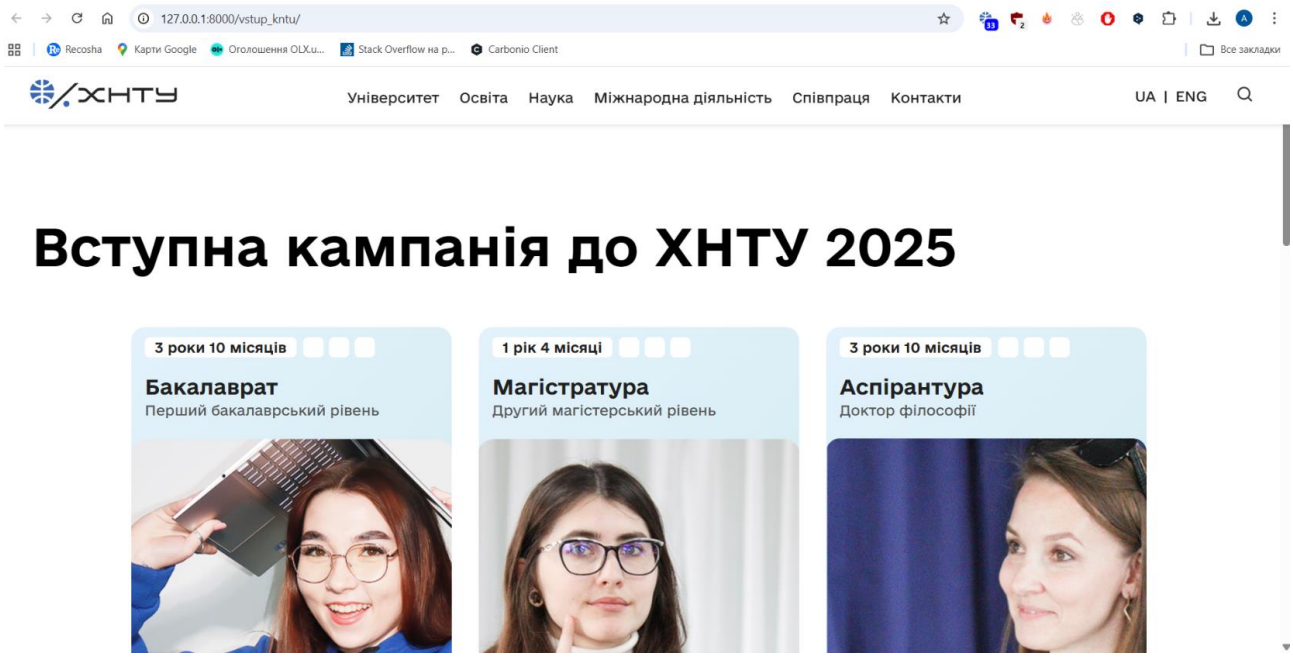

Рисунок 3.3 - Приклад відображення сторінки вступної кампанії в публічній частині системи

Отже, у межах цього етапу реалізації було створено основні моделі даних і сторінкову структуру модульної системи керування вебконтентом. Реалізований підхід забезпечує централізоване зберігання інформації, підтримку

ієрархії сторінок, зручне адміністрування та формування публічної частини вебресурсу на основі структурованих сутностей. Отримані результати створюють основу для подальшої реалізації блочного контенту, API-механізмів та інших функціональних можливостей системи.

3.2 Реалізація блочного підходу до формування вебконтенту

Одним із ключових етапів практичної реалізації програмного продукту стало впровадження блочного підходу до формування вмісту сторінок. Такий підхід забезпечує гнучке керування структурою сторінки, дозволяє повторно використовувати окремі елементи інтерфейсу та спрощує адміністрування вебконтенту. У межах розробленої системи цей механізм реалізовано за допомогою StreamField і набору власних блоків Wagtail, що описують окремі типи контенту [13].

На відміну від сторінок із жорстко зафіксованою структурою, де всі поля заздалегідь визначені в моделі, блочний підхід дає змогу формувати сторінку з незалежних функціональних елементів. Кожен блок має власну структуру, параметри та шаблон відображення, а редактор у адміністративному інтерфейсі може самостійно визначати склад сторінки, порядок розташування блоків та їх вміст. Це особливо важливо для вебресурсу закладу вищої освіти, де сторінки різних розділів часто відрізняються за наповненням, але водночас мають використовувати спільні інформаційні компоненти.

Практична реалізація такого підходу в проєкті базується на створенні класу StandardStreamBlock, у якому зібрано основні типи блоків, доступні для використання на сторінках. До нього входять текстові блоки, слайдери, секції документів, модальні вікна, футерні елементи, блок сторінки освітньої програми, а також блоки подій та анонсів. Саме цей клас використовується в моделях сторінок як основа для поля body, що формує головний вміст сторінки.

```
722
723     # 9) Основний StreamBlock
724     class StandardStreamBlock(blocks.StreamBlock):  3 usages
725         paragraph = ParagraphBlock()
726         program_slider = ProgramSliderBlock()
727         events_announcements = EventsAnnouncementsBlock()
728         events_announcements_api = EventsAnnouncementsApiBlock()
729         documents_accordion = DocumentsAccordionBlock()
730         modal_window = ModalBlock()
731         footer = FooterBlock()
732         admission_program = AdmissionProgramBlock()
733
734         class Meta:
735             icon = "placeholder"
736             label = "Основний контент"
```

Рисунок 3.4 - Фрагмент реалізації класу StandardStreamBlock у середовищі PyCharm

Важливою перевагою блочного підходу є можливість описувати складні контентні елементи як окремі структуровані блоки. Наприклад, у межах проєкту реалізовано блоки для параграфів, модальних вікон, списків документів, слайдерів програм і сторінки освітньої програми. Кожен із них має власний набір полів, що визначає логіку його заповнення й відображення. Це дозволяє створити бібліотеку повторно використовуваних компонентів, з яких надалі формуються сторінки різних типів.

Реалізація блоків у Wagtail здійснюється через класи StructBlock, ListBlock і ChoiceBlock, що дає змогу описувати як прості, так і вкладені структури даних [13]. У результаті окремий блок може містити текстові поля, зображення, посилання, списки вкладених елементів та службові параметри. Такий підхід був використаний у межах усіх ключових контентних компонентів системи, що забезпечило єдині принципи побудови й редагування вебсторінок.

Ще одним важливим етапом реалізації стало створення шаблонів для відображення блоків. Для кожного типу блоку було підготовлено відповідний HTML-файл у структурі шаблонів проєкту. Завдяки цьому один і той самий блок

має не лише визначену структуру даних, а й власний спосіб візуального подання. Наприклад, блок подій та анонсів має окремий шаблон, який формує список карток подій, тоді як блок сторінки освітньої програми формує більш складну багатосекційну сторінку з кнопками, списками та інформаційними картками.

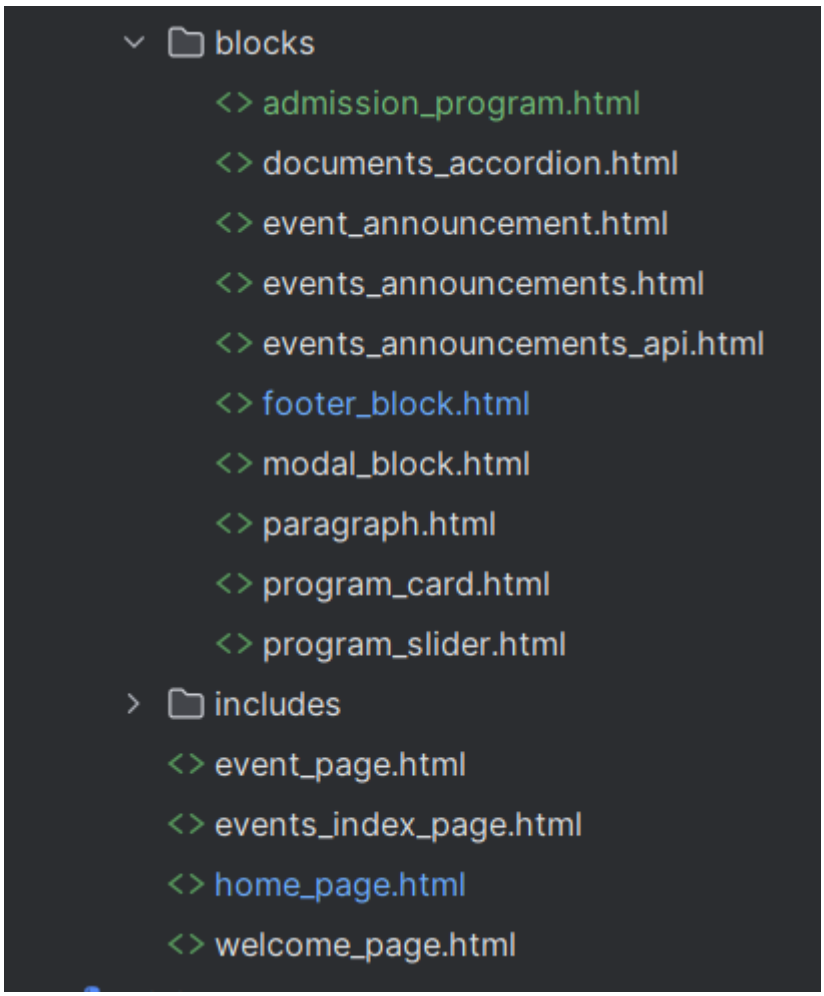

Рисунок 3.5 - Структура шаблонів блоків у проєкті в середовищі PyCharm

Одним із найбільш показових прикладів реалізації блочного підходу став блок подій та анонсів. Спочатку в системі використовувався звичайний блок подій, у якому дані задавалися безпосередньо в межах сторінки. Однак такий підхід мав обмеження, оскільки одна й та сама інформація не могла зручно повторно використовуватися на різних сторінках. У зв’язку з цим у межах удосконалення системи було реалізовано окремий блок «Події та анонси (API)», який отримує дані не локально, а через API-механізм із централізованого джерела.

Реалізація цього блоку включала дві складові: опис самого блоку в blocks.py та шаблон events_announcements_api.html, що забезпечує завантаження

даних із endpoint і побудову інтерфейсу на клієнтському рівні. У результаті сторінка може містити API-блок, який автоматично отримує актуальні дані про події та формує картки без ручного дублювання контенту. Це є прикладом переходу від локального зберігання інформації в межах сторінки до модульного повторного використання структурованих даних.

```
     <script>
document.addEventListener("DOMContentLoaded", function () {
  const section = document.querySelector(".events-announcements-api");
  if (!section) return;

  const apiUrl = section.dataset.apiUrl;
  const container = document.getElementById("events-api-list");

  const modal = document.getElementById("api-event-modal");
  const modalImageWrap = document.getElementById("api-modal-image-wrap");
  const modalImage = document.getElementById("api-modal-image");
  const modalTitle = document.getElementById("api-modal-title");
  const modalSubtitle = document.getElementById("api-modal-subtitle");
  const modalTags = document.getElementById("api-modal-tags");
  const modalContent = document.getElementById("api-modal-content");
  const modalFooter = document.getElementById("api-modal-footer");
  const modalButton = document.getElementById("api-modal-button");
  const modalButtonText = document.getElementById("api-modal-button-
text");

  let eventsData = [];

  function openModal(eventId) {
    const event = eventsData.find(item => String(item.id) ===
String(eventId));
    if (!event) return;

    if (event.image_url) {
      modalImage.src = event.image_url;
      modalImage.alt = event.title || "";
      modalImageWrap.style.display = "";
    } else {
      modalImage.src = "";
      modalImage.alt = "";
      modalImageWrap.style.display = "none";
    }

    modalTitle.textContent = event.title || "";

    if (event.subtitle) {
      modalSubtitle.textContent = event.subtitle;
      modalSubtitle.style.display = "";
    } else {
```

```
      modalSubtitle.textContent = "";
      modalSubtitle.style.display = "none";
    }

    const tags = [];
    if (event.frequency) tags.push(event.frequency);
    if (event.time) tags.push(event.time);
    if (event.platform) tags.push(event.platform);

    if (tags.length) {
      modalTags.innerHTML = tags.map(tag => `<span
class="modal__tag">${tag}</span>`).join("");
      modalTags.style.display = "";
    } else {
      modalTags.innerHTML = "";
      modalTags.style.display = "none";
    }

    modalContent.innerHTML = event.body_html || "";

    if (event.button_url) {
      modalButton.href = event.button_url;
      modalButtonText.textContent = event.button_text || "Детальніше";
      modalFooter.style.display = "";
    } else {
      modalButton.href = "#";
      modalButtonText.textContent = "Детальніше";
      modalFooter.style.display = "none";
    }

    modal.classList.add("visible");
    document.body.style.overflow = "hidden";
  }

  function closeModal() {
    modal.classList.remove("visible");
    document.body.style.overflow = "";
  }

  fetch(apiUrl)
    .then(response => response.json())
    .then(data => {
      eventsData = data;

      container.innerHTML = data.map(event => `
        <div class="event-card ${event.pinned ? 'pinned' : ''}">
          ${event.image_url ? `
            <div class="event-card__image">
              <img src="${event.image_url}" alt="${event.title}">
            </div>
          ` : ''}
```

```
          <div class="event-card__content">
            <div class="event-card__top">
              <h2 class="event-card__title">${event.title}</h2>
              ${event.subtitle ? `<div class="event-
card__subtitle">${event.subtitle}</div>` : ''}
            </div>

            <div class="event-card__actions">
              <div class="event-card__tags">
                ${event.frequency ? `<div class="event-
card__tag">${event.frequency}</div>` : ''}
                ${event.time ? `<div class="event-
card__tag">${event.time}</div>` : ''}
                ${event.platform ? `<div class="event-
card__tag">${event.platform}</div>` : ''}
              </div>

              <div class="event-card__btn">
                <a href="#"
                   class="js-open-api-modal"
                   data-event-id="${event.id}">
                  ${event.button_text}
                  <img src="/static/images/icon-details.svg"
alt="Іконка">
                </a>
              </div>
            </div>
          </div>
        </div>
      `).join("");

      container.querySelectorAll(".js-open-api-modal").forEach(btn => {
        btn.addEventListener("click", function (e) {
          e.preventDefault();
          openModal(this.dataset.eventId);
        });
      });
    })
    .catch(error => {
      container.innerHTML = "<p>Не вдалося завантажити події.</p>";
      console.error(error);
    });

  document.querySelectorAll(".js-api-modal-close").forEach(el => {
    el.addEventListener("click", closeModal);
  });

  document.addEventListener("keydown", function (e) {
    if (e.key === "Escape" && modal.classList.contains("visible")) {
      closeModal();
    }
  });
```

```
});
</script>
```

На рівні адміністративного інтерфейсу переваги блочного підходу проявляються в тому, що редактор може додавати потрібний тип блоку до сторінки через стандартне меню вибору елементів. Система дозволяє формувати сторінку як послідовність незалежних компонентів, змінювати їх порядок, редагувати вміст і відразу переглядати результат у preview-режимі. Це суттєво спрощує роботу з контентом і робить сторінки менш залежними від жорстко заданого шаблону.

Саме такий механізм було реалізовано в розробленій системі для формування сторінок вступної кампанії, освітніх програм та інших розділів вебресурсу. У практичному аспекті це дозволяє адаптувати сторінки під конкретні потреби без зміни моделі даних або втручання в серверну логіку.

Для демонстрації роботи редактора з блоками доцільно використати скріншот адміністративної панелі Wagtail із меню вибору доступних блоків і preview-режимом сторінки.

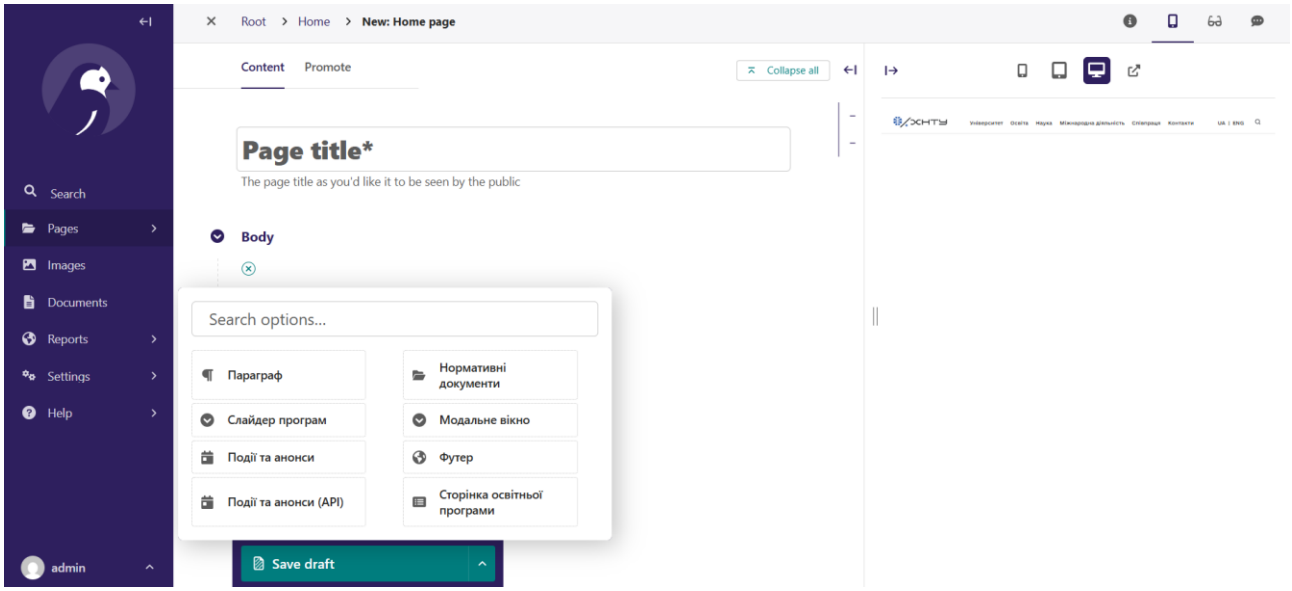

Рисунок 3.6 - Додавання та вибір блоків у адміністративній частині системи

Реалізовані в межах проєкту блоки відрізняються за структурою, функціональним призначенням та характером використання, однак у сукупності формують єдину систему компонентів для побудови вебсторінок. Частина блоків

орієнтована на подання текстової та довідкової інформації, інші забезпечують роботу з документами, освітніми програмами, модальними вікнами або подіями. З огляду на це доцільно окремо узагальнити основні блоки системи, їх призначення та практичне застосування.

Таблиця 3.2 - Основні блоки, реалізовані в системі

Назва блоку	Призначення	Основні елементи	Практичне використання
ParagraphBlock	Відображення текстового вмісту	текст, стилі, вирівнювання	інформаційні сторінки, описи
ProgramSliderBlock	Відображення карток освітніх програм	список карток, зображення, кнопки	сторінки вступної кампанії
DocumentsAccordionBlock	Відображення груп документів	секції, назви документів, файли	нормативні документи, вступ
ModalBlock	Відображення модального вікна	заголовок, опис, теги, кнопка	інформаційні блоки з деталізацією
FooterBlock	Формування нижньої частини сторінки	контакти, адреса, карта	футер сайту
AdmissionProgramBlock	Побудова сторінки освітньої програми	кнопки, секції, опис, інфо-картки	сторінки програм

Джерело: створено автором на основі власних досліджень

З таблиці 3.2 видно, що блочний підхід дозволив реалізувати набір спеціалізованих компонентів, кожен з яких виконує окрему функцію в межах системи. Така організація є зручною як для розробника, оскільки спрощує підтримку й розширення функціоналу, так і для редактора контенту, який працює з готовими структурованими елементами без необхідності втручання в код.

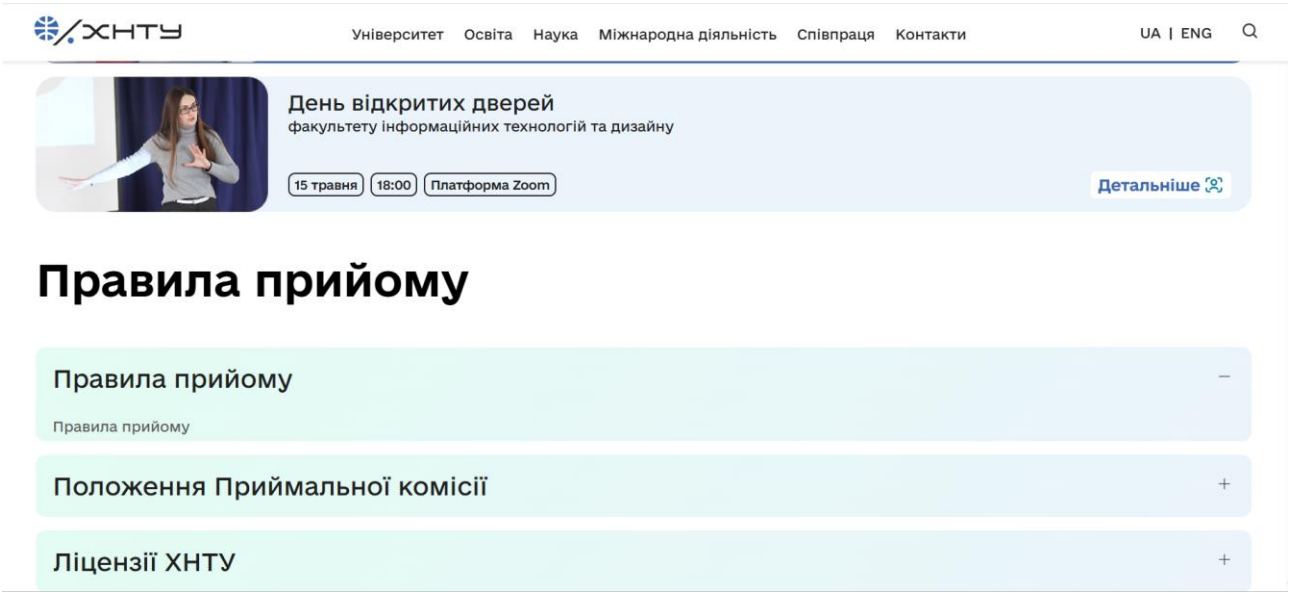

Рисунок 3.7 - Приклад сторінки, сформованої з використанням кількох контентних блоків

Отже, у межах реалізації блочного підходу до формування вебконтенту було створено набір структурованих компонентів, які використовуються для побудови сторінок різного призначення. Використання StreamField, власних блоків і шаблонів відображення забезпечило гнучке керування структурою сторінок, повторне використання елементів інтерфейсу та спрощення адміністрування вебресурсу. Практичне впровадження API-блоку подій та анонсів показало, що блочний підхід може бути поєднаний із централізованим джерелом даних, що підвищує цілісність і масштабованість системи.

3.3 Реалізація API-модуля подій, перевірка його роботи та інструкція користувача

Одним із найважливіших етапів практичної реалізації програмного продукту стало створення API-модуля подій та анонсів, який забезпечує централізоване зберігання інформації про події та її повторне використання в різних частинах вебресурсу. Необхідність реалізації такого модуля була зумовлена тим, що початковий підхід, за якого блок подій існував лише як

локальний елемент у межах конкретної сторінки, не дозволяв ефективно повторно використовувати дані й ускладнював супровід системи.

Для усунення цього обмеження в межах проєкту було реалізовано окрему сторінку-розділ подій EventsIndexPage та окрему модель події EventPage. Такий підхід дозволив зберігати інформацію про події у вигляді самостійних сутностей, що мають власні поля, адміністративний інтерфейс редагування та окреме відображення в публічній частині системи. У результаті всі події були винесені в централізовану структуру, а їх використання на сторінках почало здійснюватися не через копіювання контенту, а через звернення до єдиного джерела даних.

На рівні серверної логіки для реалізації API-модуля було створено окреме представлення events_api у файлі views.py. Це представлення виконує вибірку опублікованих подій, упорядковує їх за ознакою закріплення та датою публікації, а потім формує JSON-відповідь із необхідним набором полів. До цієї відповіді включаються ідентифікатор події, заголовок, підзаголовок, частота, час, платформа, текст кнопки, посилання, ознака закріплення, шлях до зображення, а також повний і скорочений опис події. Такий набір даних є достатнім як для формування картки події, так і для заповнення модального вікна при взаємодії користувача з API-блоком.

Фрагмент реалізації серверної логіки API-модуля наведено нижче.

```
from django.http import JsonResponse
from django.utils.html import strip_tags
from .models import EventPage


def events_api(request):
    events = EventPage.objects.live().public().specific().order_by("-pinned", "-first_published_at")

    data = []
    for event in events:
        image_url = ""
```

```
            if event.image:
                image_url = event.image.file.url

            body_html = event.body if event.body else ""
            body_text = strip_tags(body_html) if body_html else ""

            data.append({
                "id": event.id,
                "title": event.title,
                "subtitle": event.subtitle,
                "frequency": event.frequency,
                "time": event.time,
                "platform": event.platform,
                "button_text": event.button_text or "Детальніше",
                "button_url": event.url,
                "pinned": event.pinned,
                "image_url": image_url,
                "body_html": body_html,
                "body_text": body_text,
            })

        return JsonResponse(data, safe=False,
json_dumps_params={"ensure_ascii": False})
```

З практичної точки зору важливим є те, що API-представлення не лише повертає структуровані дані, а й виконує попередню підготовку інформації для подальшого використання на клієнтському рівні. Це дозволяє зменшити обсяг логіки в шаблоні та спростити побудову інтерфейсу, оскільки сторінка вже отримує готовий набір полів, придатних до безпосереднього відображення.

Після реалізації представлення було виконано підключення окремого маршруту в конфігурації urls.py. Це дало змогу організувати доступ до даних за endpoint /api/events/, який став точкою взаємодії між серверною частиною системи та API-блоком, що використовується на сторінках.

Фрагмент підключення API-маршруту наведено нижче.

```
from home.views import events_api


urlpatterns = [
    path('api/events/', events_api, name='events_api'),
]
```

Рисунок 3.8 - Результат роботи endpoint /api/events/ у форматі JSON

Наступним етапом стала реалізація блоку «Події та анонси (API)» у системі керування контентом. Для цього в blocks.py було створено окремий блок EventsAnnouncementsApiBlock, який містить заголовок секції та адресу API. Такий підхід дозволяє редактору додавати API-блок на сторінку як звичайний контентний елемент, не змінюючи серверний код і не дублюючи інформацію про події вручну.

Фрагмент опису API-блоку наведено нижче.

```
class EventsAnnouncementsApiBlock(blocks.StructBlock):
    title = blocks.CharBlock(
        required=False,
        default="Події та анонси",
        label="Заголовок секції"
```

```
    )
    api_url = blocks.CharBlock(
        required=False,
        default="/api/events/",
        label="API URL"
    )

    class Meta:
        template = "home/blocks/events_announcements_api.html"
        icon = "date"
        label = "Події та анонси (API)"
```

Після цього блок було додано до складу StandardStreamBlock, завдяки чому він став доступним для використання в адміністративній частині системи разом з іншими блоками. Це забезпечило інтеграцію API-модуля в загальний механізм формування сторінок через StreamField.

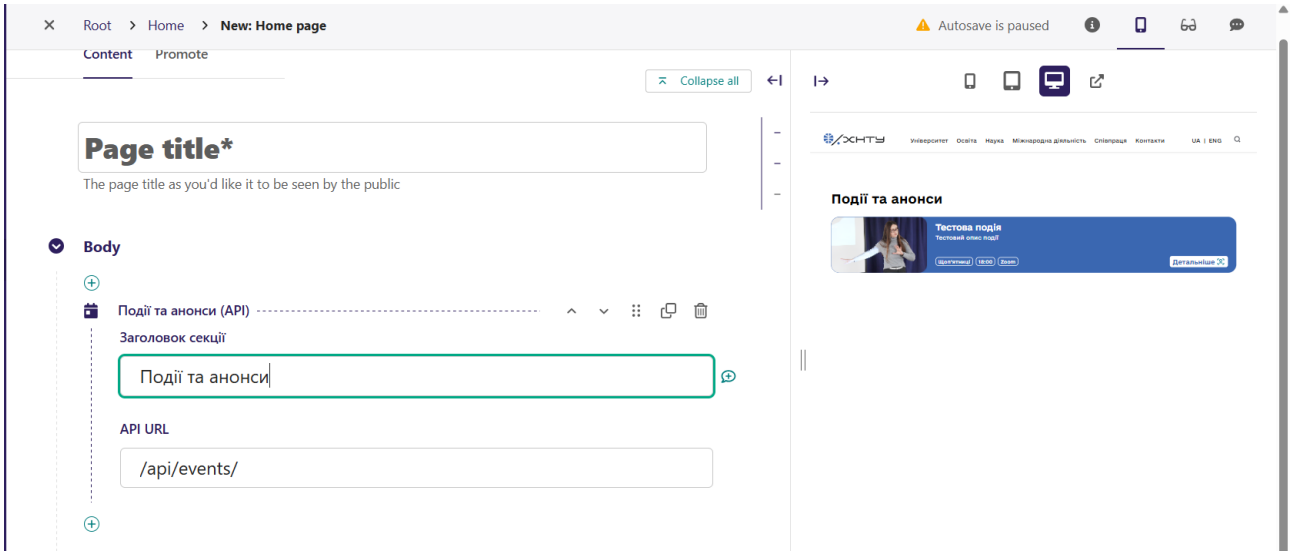

Рисунок 3.9 - Додавання блоку «Події та анонси (API)» у адміністративній частині системи

На клієнтському рівні робота API-модуля реалізується в шаблоні events_announcements_api.html. Після відкриття сторінки JavaScript-код виконує HTTP-запит до зазначеної адреси API, отримує JSON-відповідь і будує на її основі набір карток подій. Для кожної картки виводяться зображення, заголовок,

підзаголовок, теги часу та платформи, а також кнопка для відкриття детальної інформації.

Приклад JavaScript-логіки завантаження подій через API наведено у фрагменті коду.

```
fetch(apiUrl)
  .then(response => response.json())
  .then(data => {
    eventsData = data;

    container.innerHTML = data.map(event => `
      <div class="event-card ${event.pinned ? 'pinned' : ''}">
        ${event.image_url ? `
          <div class="event-card__image">
            <img src="${event.image_url}" alt="${event.title}">
          </div>
        ` : ''}
      </div>
    `).join("");
  })
```

Важливою складовою реалізації стало також модальне вікно, яке відкривається після натискання на кнопку «Детальніше». Для цього в шаблоні було створено контейнер модального вікна та JavaScript-функції openModal() і closeModal(). Під час відкриття модалки система знаходить потрібний запис події в уже завантаженому масиві eventsData, після чого заповнює заголовок, підзаголовок, теги, зображення та повний опис події. Завдяки цьому користувач отримує розгорнуту інформацію без переходу на окрему сторінку та без повторного запиту до сервера.

Такий підхід дозволяє поєднати централізоване зберігання даних із сучасною інтерактивною формою їх відображення. У практичному аспекті це підвищує зручність використання системи, оскільки події можуть відображатися

в компактному вигляді на сторінці, але за необхідності розкриватися в повному форматі через модальне вікно.

Таблиця 3.3 - Основні складові реалізації API-модуля подій та анонсів

Складова модуля	Призначення	Реалізація в проєкті	Результат
Модель події	Централізоване зберігання даних про події	EventPage	окрема структурована сутність події
Сторінка-розділ	Групування та організація подій	EventsIndexPage	логічна структура модуля подій
API-представлення	Надання даних у форматі JSON	events_api у views.py	endpoint /api/events/
API-маршрут	Доступ до API через URL	path('api/events/', events_api, ...)	зв’язок клієнта з сервером
API-блок	Відображення подій на сторінці	EventsAnnouncementsApiBlock	повторне використання подій
JavaScript-логіка	Завантаження та побудова карток	fetch(apiUrl) і обробка відповіді	динамічне формування блоку
Модальне вікно	Виведення детальної інформації	openModal() / closeModal()	інтерактивна взаємодія з подією

Джерело: створено автором на основі власних досліджень

Наведені в таблиці 3.3 складові відображають повний ланцюг реалізації API-модуля - від моделі даних і серверного представлення до клієнтського відображення та взаємодії користувача з інтерфейсом. Практичний результат роботи цього механізму доцільно проілюструвати прикладами з публічної частини системи.

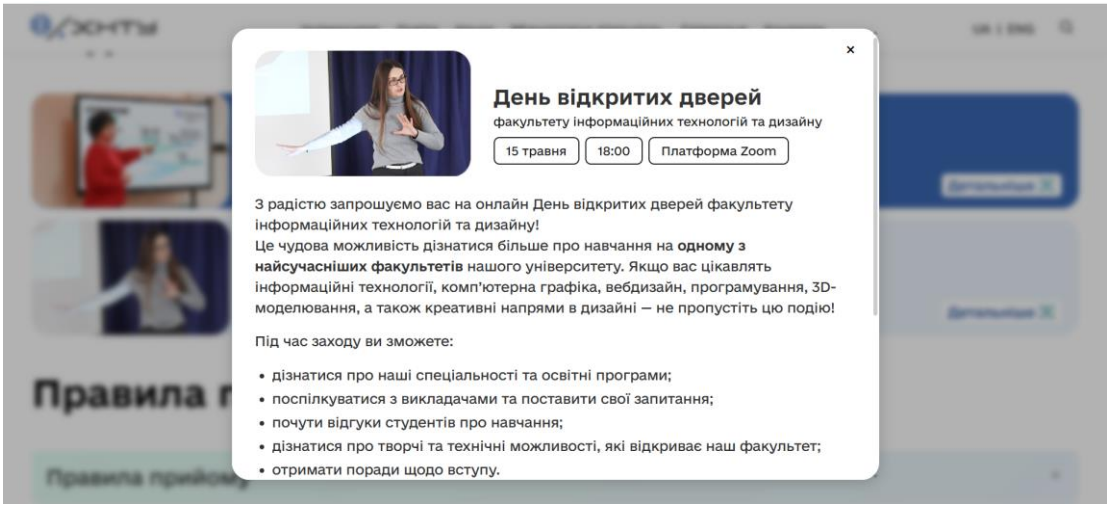

Рисунок 3.12 - Відображення детальної інформації про подію в модальному вікні

Інструкція користувача щодо роботи з модулем подій та анонсів

Для практичного використання реалізованого модуля подій та анонсів користувач із правами адміністратора або редактора повинен виконати послідовність дій в адміністративній частині Wagtail. Робота з модулем не потребує внесення змін до програмного коду, оскільки створення подій, їх редагування, публікація та додавання API-блоку на сторінку виконуються через графічний інтерфейс системи.

Першим кроком користувач переходить до адміністративної частини Wagtail і відкриває розділ сторінок. У дереві сторінок необхідно знайти сторінку-розділ подій, яка відповідає за групування записів типу EventPage. Після вибору цього розділу користувач може додати нову дочірню сторінку, що буде використовуватися як окрема подія або анонс.

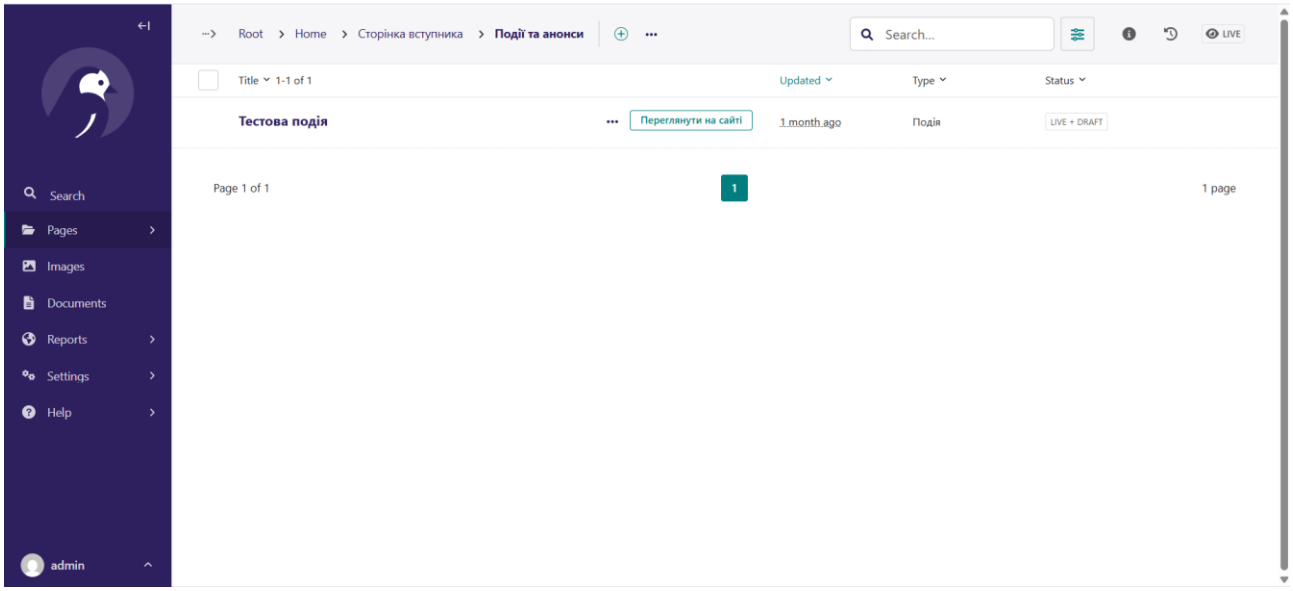

Рисунок 3.13 - Вибір сторінки-розділу подій в адміністративній частині системи

Після створення нової сторінки події система відкриває форму редагування, у якій користувач заповнює основні поля. До них належать назва події, підзаголовок, дата, час, платформа проведення, опис, текст кнопки, посилання та зображення. За потреби редактор може встановити ознаку закріплення події, що впливає на порядок її відображення в API-блоці. Після заповнення всіх необхідних полів сторінку потрібно зберегти як чернетку або опублікувати.

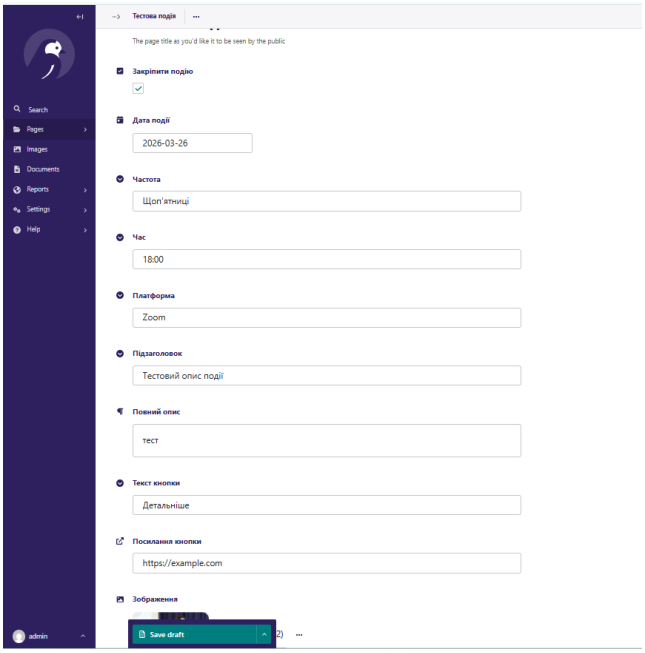

Рисунок 3.14 - Заповнення форми редагування події

Після публікації події вона автоматично стає доступною для використання в API-відповіді за маршрутом /api/events/. Це означає, що редактору не потрібно дублювати інформацію про подію на різних сторінках сайту. Достатньо один раз створити та опублікувати запис у відповідному розділі, після чого система може повторно використовувати ці дані в інших частинах вебресурсу.

Наступним кроком користувач може додати блок «Події та анонси (API)» на потрібну сторінку сайту. Для цього необхідно відкрити сторінку в режимі редагування, перейти до поля формування вмісту сторінки та вибрати відповідний блок зі списку доступних елементів. У налаштуваннях блоку можна вказати заголовок секції та адресу API. За замовчуванням використовується шлях /api/events/, тому в більшості випадків додаткове налаштування адреси не потрібне.

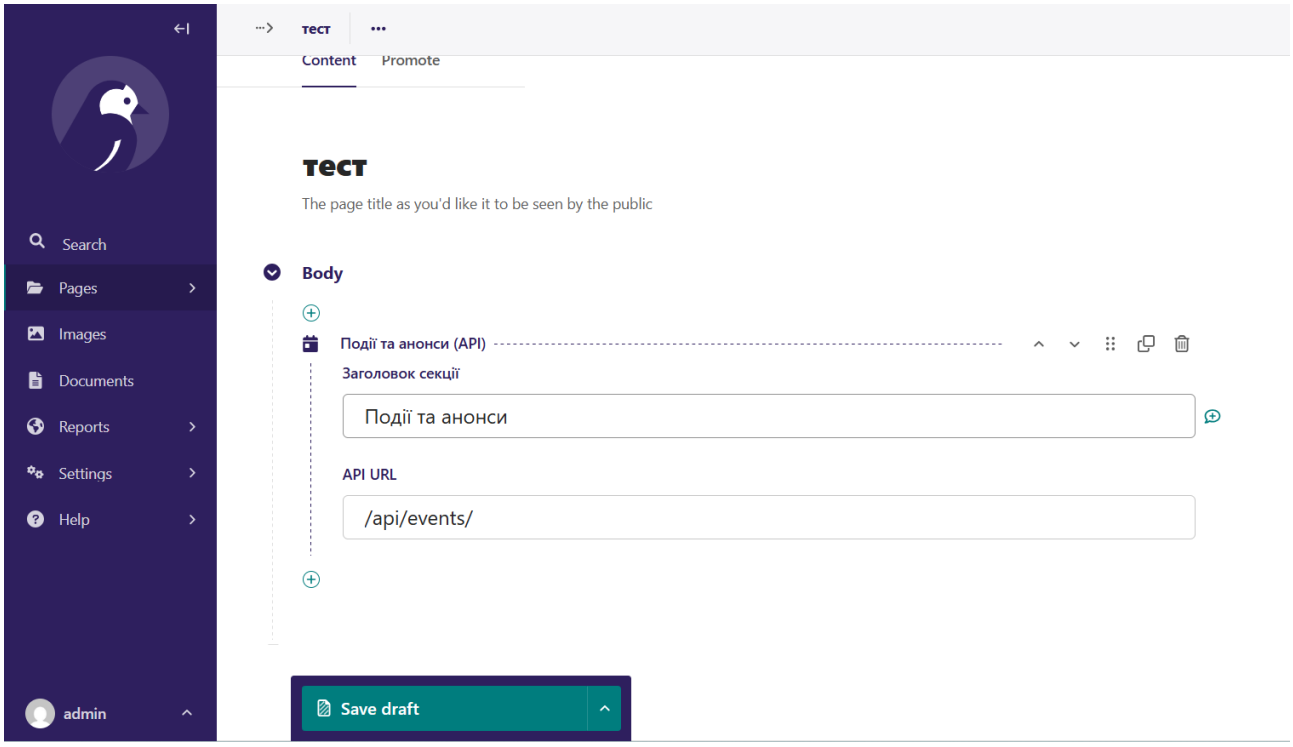

Рисунок 3.15 - Додавання блоку «Події та анонси (API)» на сторінку

Після додавання блоку сторінку необхідно зберегти та опублікувати. У публічній частині сайту API-блок автоматично завантажує актуальний перелік подій із централізованого джерела даних і формує картки подій. На картці

відображаються назва, підзаголовок, зображення, час, платформа проведення та кнопка для перегляду детальної інформації.

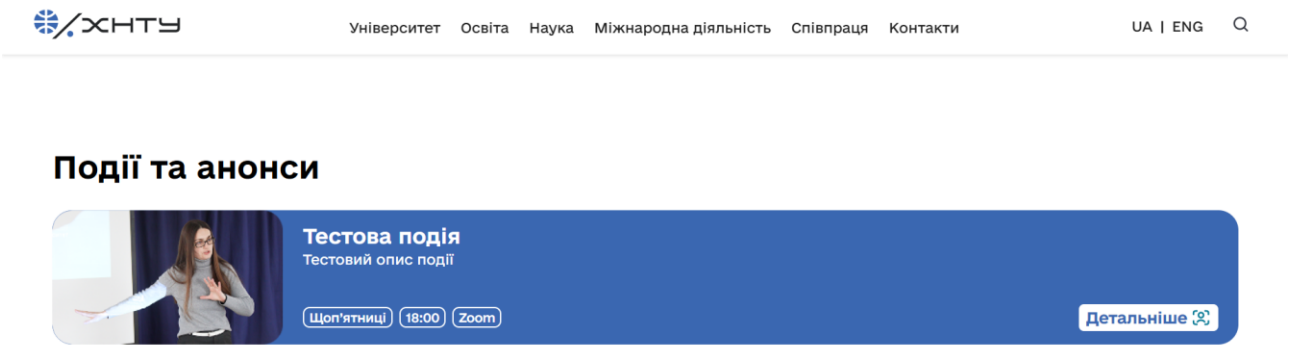

Рисунок 3.16 - Відображення API-блоку подій у публічній частині сайту

Для перегляду повної інформації про подію кінцевий користувач натискає кнопку «Детальніше». Після цього відкривається модальне вікно, у якому відображаються зображення, назва, підзаголовок, службові позначки, повний опис події та додаткове посилання. Закрити модальне вікно можна за допомогою відповідної кнопки або клавіші Escape.

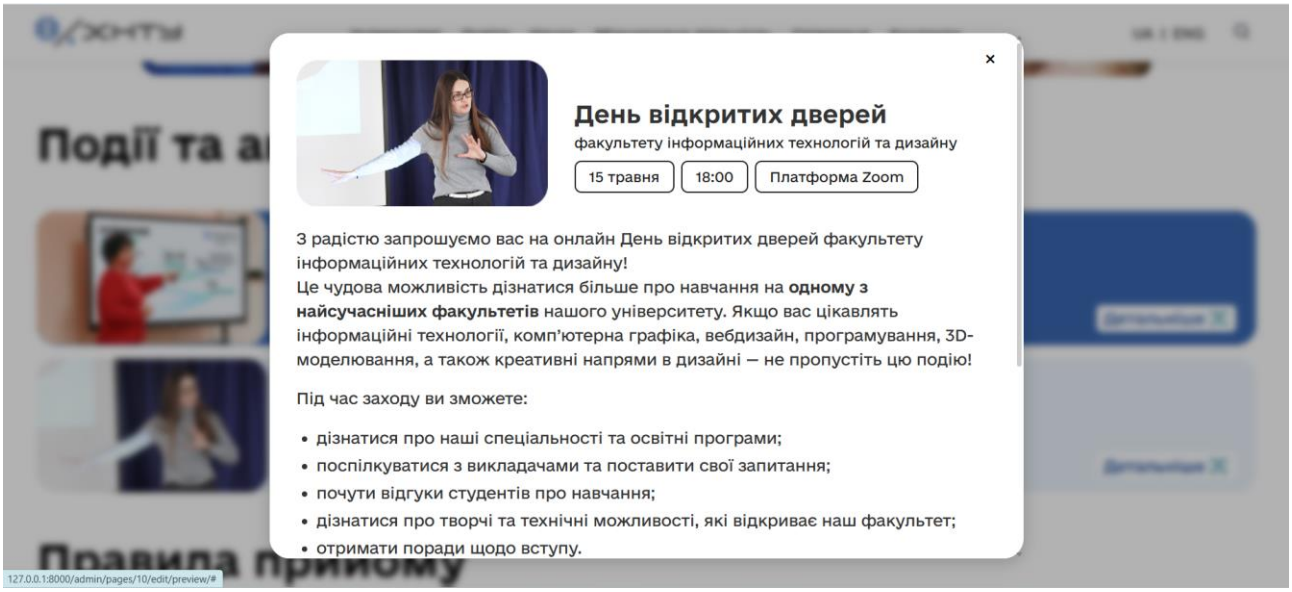

Рисунок 3.17 - Перегляд детальної інформації про подію в модальному вікні

Таким чином, робота з модулем подій та анонсів виконується через адміністративний інтерфейс Wagtail і є зрозумілою для користувача з правами редактора. Основна послідовність дій включає вибір розділу подій, створення

або редагування події, публікацію запису, додавання API-блоку на потрібну сторінку та перевірку результату в публічній частині сайту. Це підтверджує, що реалізований модуль є не лише технічно працездатним, а й придатним до практичного використання під час адміністрування вебресурсу закладу вищої освіти.

Після опису користувацького сценарію було виконано перевірку працездатності API-модуля шляхом поетапного тестування всіх його складових. Спочатку було перевірено коректність формування JSON-відповіді за адресою /api/events/. Після цього перевірено правильність підключення API-блоку в адміністративній частині системи та коректність завантаження даних на клієнтському рівні. Окремо було протестовано побудову карток подій, роботу кнопки відкриття модального вікна, заповнення полів модалки та відображення повного опису події. Результати перевірки показали, що реалізований модуль функціонує коректно та може бути використаний як повторно використовуваний компонент у різних частинах вебресурсу.

Отже, у межах реалізації API-модуля подій та анонсів було створено повноцінний механізм централізованого зберігання, отримання та повторного використання інформації про події. Розроблене рішення поєднує моделі сторінок Wagtail, серверну обробку даних у Django, API-взаємодію у форматі JSON та клієнтське динамічне відображення подій. Окремо описана інструкція користувача демонструє послідовність практичної роботи з модулем через адміністративний інтерфейс Wagtail і публічну частину сайту.

Це підтверджує практичну придатність модульного підходу до побудови системи керування вебконтентом і демонструє можливість розширення функціоналу вебресурсу без дублювання даних.

Висновок до розділу

У третьому розділі було розглянуто практичну реалізацію модульної системи керування вебконтентом на базі Django та Wagtail. У межах підрозділу 3.1 реалізовано основні моделі даних і сторінкову структуру системи, зокрема головну сторінку, контентні сторінки, сторінку-розділ подій та сторінку окремої події. Це дозволило побудувати впорядковану ієрархію вебресурсу та забезпечити централізоване зберігання контенту.

У підрозділі 3.2 було реалізовано блочний підхід до формування вебконтенту. На основі механізму StreamField і набору власних блоків створено гнучку систему побудови сторінок, у якій окремі компоненти можуть повторно використовуватися в різних частинах сайту. Такий підхід забезпечив спрощення адміністрування, підвищення модульності системи та зручність редагування вмісту через адміністративний інтерфейс.

У підрозділі 3.3 реалізовано API-модуль подій та анонсів, який став прикладом практичного використання централізованої моделі даних і повторного використання контенту. Було створено серверне API-представлення, маршрут доступу до даних, клієнтську логіку завантаження інформації та механізм відображення подій у вигляді інтерактивного блоку з модальним вікном. Проведена перевірка показала коректну роботу розробленого модуля та підтвердила доцільність використання API-підходу в межах модульної системи керування вебконтентом.

Отже, у третьому розділі було практично підтверджено можливість реалізації модульної системи керування вебконтентом, яка забезпечує структуроване зберігання даних, гнучке формування сторінок, повторне використання блоків і інтеграцію API-механізмів. Отримані результати свідчать про працездатність розробленого програмного продукту та його придатність до використання як основи для керування вебконтентом сайту закладу вищої освіти.

ВИСНОВОК

У кваліфікаційній роботі вирішено практичну проблему розробки та впровадження модульної системи керування вебконтентом для сайту закладу вищої освіти на базі фреймворку Django та CMS Wagtail. Актуальність теми зумовлена зростанням ролі офіційного вебсайту університету як основного цифрового каналу комунікації з абітурієнтами, здобувачами освіти, працівниками, випускниками та іншими категоріями користувачів. У сучасних умовах вебресурс закладу вищої освіти повинен забезпечувати не лише публікацію інформації, а й оперативне оновлення контенту, зручне адміністрування, логічну структуру сторінок, підтримку різних типів інформаційних матеріалів та можливість подальшого розширення функціоналу. Саме тому розробка модульної системи керування вебконтентом є доцільною та практично значущою.

У процесі виконання роботи було досліджено предметну область керування вебконтентом офіційного сайту закладу вищої освіти. Встановлено, що університетський вебсайт є складною інформаційною системою, яка поєднує в собі функції офіційного представництва закладу в цифровому середовищі, інформаційної підтримки освітнього процесу, представлення нормативних документів, публікації новин, анонсів, відомостей про освітні програми, структурні підрозділи та інші інформаційні об’єкти. Визначено, що особливостями цієї предметної області є значний обсяг різнорідного контенту, потреба в його регулярному оновленні, необхідність централізованого адміністрування та забезпечення логічної структури подання інформації. На основі проведеного аналізу обґрунтовано доцільність використання системи керування вебконтентом як технологічної основи для побудови сучасного університетського вебресурсу.

У роботі було проведено аналіз існуючих аналогів і потенційних конкурентних переваг програмного продукту. Розглянуто можливості таких платформ, як WordPress, Drupal та Wagtail. У результаті аналізу встановлено, що

WordPress є зручним рішенням для швидкого розгортання типових сайтів завдяки великій кількості готових тем і плагінів, однак його використання для складних модульних систем може ускладнювати підтримку архітектурної цілісності та сприяти надмірній залежності від сторонніх розширень. Drupal, у свою чергу, характеризується високою гнучкістю, потужними механізмами адміністрування та придатністю до реалізації складних вебресурсів, проте має вищий поріг входу, інший технологічний стек і є менш зручним для інтеграції з архітектурою на базі Python та Django. У результаті встановлено, що Wagtail найбільш повно відповідає потребам розроблюваної системи, оскільки поєднує переваги повноцінної CMS із гнучкістю фреймворкового підходу, підтримує сторінкові моделі, блочне формування контенту, централізоване адміністрування та API-взаємодію. На цій підставі зроблено висновок про доцільність використання Wagtail у поєднанні з Django як технологічної основи дипломного проєкту.

У межах першого розділу також було сформульовано постановку задачі на розробку модульної системи керування вебконтентом. Визначено мету, завдання, об’єкт і предмет дослідження, а також сформовано вимоги до майбутнього програмного продукту. Встановлено, що система повинна забезпечувати централізоване зберігання даних, підтримку різних типів сторінок, можливість побудови сторінок із окремих контентних блоків, використання API для повторного застосування даних, підтримку ролей користувачів і подальшу розширюваність архітектури. Таким чином, у першому розділі було створено теоретичну та постановочну основу для подальшого проєктування і реалізації системи.

У другому розділі виконано проєктування модульної системи керування вебконтентом. На етапі моделювання функціонування системи визначено основних учасників взаємодії з програмним продуктом: адміністратора, редактора контенту, кінцевого користувача та API-клієнта. Описано загальну логіку роботи системи, яка поєднує можливості традиційної CMS із елементами API-взаємодії. Встановлено, що розроблювана система повинна підтримувати як

керування сторінками через адміністративний інтерфейс, так і повторне використання структурованих даних у межах різних компонентів сайту.

У межах проєктування архітектури та структури системи обґрунтовано багаторівневий підхід до побудови програмного продукту. Виділено рівень зберігання даних, представлений моделями Django; рівень прикладної логіки, що містить механізми обробки контенту, маршрутизацію, API-представлення та правила доступу; а також рівень представлення, що включає HTML-шаблони, CSS-оформлення та клієнтські сценарії JavaScript. Встановлено, що така архітектура є доцільною, оскільки забезпечує розмежування відповідальності між окремими компонентами системи, покращує її підтримуваність і створює основу для подальшого масштабування. Окрему увагу приділено сторінковим моделям Wagtail, механізму StreamField, шаблонам контентних блоків і API-рівню як ключовим елементам архітектури розроблюваної системи.

Під час обґрунтування вибору технологій та опису алгоритмів роботи системи було встановлено, що обраний технологічний стек повною мірою відповідає завданням дипломної роботи. Python обрано як основну мову програмування завдяки його виразності, читабельності й широкій підтримці веброзробки. Django використано як серверний вебфреймворк, що забезпечує моделі даних, ORM, маршрутизацію, систему шаблонів, механізми доступу та інші необхідні засоби побудови вебзастосунків. Wagtail використано як CMS-рівень системи, що дозволяє реалізувати ієрархію сторінок, блочне формування контенту та адміністративний інтерфейс. Для зберігання даних використано PostgreSQL, а для клієнтського рівня - HTML, CSS та JavaScript. У межах другого розділу також описано алгоритми формування сторінок, роботи блочного контенту та API-взаємодії для модуля подій та анонсів. Таким чином, у другому розділі було сформовано архітектурну та технологічну основу майбутньої реалізації.

У третьому розділі виконано практичну реалізацію розробленої модульної системи керування вебконтентом. На першому етапі було створено основні моделі даних і сторінкову структуру системи. Реалізовано моделі HomePage,

ContentPage, EventsIndexPage та EventPage, що забезпечили побудову ієрархії сторінок і логічне розділення функцій між різними частинами вебресурсу. Головна сторінка використовується як точка входу до сайту, контентні сторінки - для формування типових інформаційних розділів, сторінка-розділ подій - для централізованої організації подій, а сторінка окремої події - для подання детальної інформації про конкретний запис. Було також реалізовано правила вкладеності сторінок, що дозволило забезпечити логічну структуру дерева сайту та запобігти помилкам у процесі адміністрування.

Наступним важливим етапом стала реалізація блочного підходу до формування вебконтенту. У межах роботи створено клас StandardStreamBlock, у якому зібрано основні типи блоків, доступні для використання на сторінках. Реалізовано блоки для параграфів, модальних вікон, слайдерів, секцій документів, футерних компонентів, сторінок освітніх програм, а також блок подій та анонсів через API. Для кожного блоку створено власну структуру полів і окремий шаблон відображення. Використання StreamField дало змогу відмовитися від жорстко зафіксованої структури сторінок і перейти до гнучкого конструювання вмісту з незалежних компонентів. Це забезпечило повторне використання елементів, спростило редагування сторінок через адміністративний інтерфейс та підвищило модульність системи в цілому.

Одним із найбільш важливих результатів роботи стало створення API-модуля подій та анонсів. Початково події могли існувати як локальний блок у межах конкретної сторінки, однак у процесі розвитку проєкту було встановлено, що такий підхід не забезпечує достатньої гнучкості та призводить до дублювання даних. Для вирішення цієї проблеми реалізовано окрему модель події EventPage, сторінку-розділ EventsIndexPage, серверне представлення events_api, маршрут доступу до API /api/events/, блок EventsAnnouncementsApiBlock і клієнтську логіку завантаження та відображення подій через JavaScript. У результаті події почали зберігатися централізовано, а їхнє відображення на сторінках сайту стало здійснюватися через єдине джерело даних. Це дозволило реалізувати повторне

використання контенту без дублювання, а також побудувати інтерактивний інтерфейс із модальним вікном для перегляду повної інформації про подію.

У процесі практичної реалізації було підтверджено, що поєднання Django та Wagtail дозволяє створити гнучку модульну CMS, адаптовану до потреб університетського вебресурсу. Система забезпечує зручне адміністрування сторінок, централізоване зберігання контенту, підтримку різних типів сторінок і блоків, можливість повторного використання інформаційних компонентів та інтеграцію API-рівня для динамічного завантаження даних. Перевірка працездатності реалізованих модулів показала коректне функціонування системи, зокрема правильне формування сторінок, роботу блоків, API-відповіді, відображення карток подій та відкриття модального вікна.

Таким чином, у роботі досягнуто поставленої мети - розроблено та впроваджено модульну систему керування вебконтентом для сайту закладу вищої освіти на базі Django та Wagtail. Усі визначені в роботі завдання виконано: досліджено предметну область, проаналізовано існуючі аналоги, сформульовано постановку задачі, спроєктовано архітектуру системи, обґрунтовано вибір технологій, реалізовано моделі даних, сторінкову структуру, блочний підхід до формування вебконтенту та API-модуль подій і анонсів.

Практичне значення одержаних результатів полягає в тому, що створений програмний продукт може бути використаний як основа для підтримки й подальшого розвитку вебсайту закладу вищої освіти. Розроблена система забезпечує централізоване керування інформаційним наповненням, спрощує внесення змін до структури сторінок, зменшує дублювання контенту та створює умови для поступового розширення функціоналу. Особливу цінність має реалізований модуль подій та анонсів, який демонструє можливість поєднання традиційного CMS-підходу з елементами API-взаємодії без руйнування загальної структури системи.

Одержані результати можуть бути використані не лише в межах конкретного університетського вебресурсу, а й як основа для розробки подібних систем у споріднених предметних областях, де важливими є модульність,

централізоване керування контентом і повторне використання даних. Запропонований підхід може бути адаптований для сайтів факультетів, кафедр, освітніх програм, інформаційних порталів або інших вебсистем, які мають складну ієрархію сторінок і потребують гнучкого механізму адміністрування.

Перспективи подальшого розвитку програмного продукту полягають у розширенні кількості спеціалізованих модулів, удосконаленні механізмів пошуку та фільтрації контенту, розвитку ролей і прав доступу, покращенні засобів візуального редагування сторінок, а також у подальшому використанні API для інтеграції з іншими цифровими сервісами закладу вищої освіти. Крім того, перспективним напрямом є розширення клієнтської частини системи, удосконалення адаптивності інтерфейсу та подальше підвищення зручності роботи редакторів контенту в адміністративній панелі.

Отже, результати дипломної роботи підтверджують, що розроблена модульна система керування вебконтентом є працездатним, структурованим і перспективним програмним продуктом, який відповідає поставленим вимогам та може бути використаний як основа для створення й супроводу сучасного вебресурсу закладу вищої освіти.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Devopedia. Content Management System. URL: https://devopedia.org/content-management-system (дата звернення: 05.04.2026).

2. WordPress.org. Features. URL: https://wordpress.org/about/features/ (дата звернення: 05.04.2026).

3. WordPress.org. Roles and Capabilities. URL: https://wordpress.org/documentation/article/roles-and-capabilities/ (дата звернення: 05.04.2026).

4. Drupal.org. Roles and Permissions. URL: https://www.drupal.org/docs/roles-and-permissions (дата звернення: 05.04.2026).

5. Wagtail Documentation. Page models. URL: https://docs.wagtail.org/en/stable/topics/pages.html (дата звернення: 05.04.2026).

6. Wagtail Documentation. How to use StreamField for mixed content. URL: https://docs.wagtail.org/en/stable/topics/streamfield.html (дата звернення: 05.04.2026).

7. Wagtail Documentation. Wagtail API v2 configuration guide. URL: https://docs.wagtail.org/en/stable/advanced_topics/api/v2/configuration.html (дата звернення: 05.04.2026).

8. TechTarget. Traditional vs. headless CMS. URL: https://www.techtarget.com/searchcontentmanagement/definition/headless-CMS-headless-content-management-system (дата звернення: 05.04.2026).

9. Django Documentation. Using the Django authentication system. URL: https://docs.djangoproject.com/en/6.0/topics/auth/default/ (дата звернення: 05.04.2026).

10. Wagtail Documentation. Permissions. URL: https://docs.wagtail.org/en/stable/topics/permissions.html (дата звернення: 05.04.2026).

11. Django Documentation. Models. URL: https://docs.djangoproject.com/en/6.0/topics/db/models/ (дата звернення: 13.04.2026).

12. SUNY Empire State University. Web Presence and Publishing. URL: https://catalog.sunyempire.edu/undergraduate/academic-policies-procedures/web-presence-publishing-policy/ (дата звернення: 14.04.2026).

13. Appalachian State University. Guides for Effective Websites. URL: https://uc.appstate.edu/strategy-planning/websites/support/guides (дата звернення: 14.04.2026).

14. UNESCO. Digital learning and transformation of education: what you need to know. URL: https://www.unesco.org/en/digital-education/need-know (дата звернення: 14.04.2026).

15. EDUCAUSE. Web Content Management. URL: https://www.educause.edu/ecar/research-publications/enterprise-application-market-in-higher-education/2016/web-content-management (дата звернення: 14.04.2026).

16. Campbell University. Web Policy. URL: https://www.campbell.edu/university-communications/web/web-policy/ (дата звернення: 14.04.2026).

17. Pantheon. Drupal’s Role in Higher Ed Tech. URL: https://pantheon.io/learning-center/drupal/drupal-for-higher-education (дата звернення: 14.04.2026).

18. University of Cincinnati. Web Governance Protocol. URL: https://www.uc.edu/about/marketing-communications/web/protocol.html (дата звернення: 14.04.2026).

19. Python Documentation. The Python Tutorial. URL: https://docs.python.org/3/tutorial/index.html (дата звернення: 14.04.2026).

20. Django Documentation. URL dispatcher. URL: https://docs.djangoproject.com/en/6.0/topics/http/urls/ (дата звернення: 14.04.2026).

21. Django Documentation. Writing views. URL: https://docs.djangoproject.com/en/6.0/topics/http/views/ (дата звернення: 14.04.2026).

22. Django Documentation. The Django template language. URL: https://docs.djangoproject.com/en/6.0/ref/templates/language/ (дата звернення: 14.04.2026).

23. PostgreSQL Documentation. JSON Types. URL: https://www.postgresql.org/docs/current/datatype-json.html (дата звернення: 14.04.2026).

24. PostgreSQL Documentation. JSON Functions and Operators. URL: https://www.postgresql.org/docs/current/functions-json.html (дата звернення: 14.04.2026).

25. MDN Web Docs. Using the Fetch API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch (дата звернення: 14.04.2026).

ДОДАТОК 1

Реалізація головної сторінки

website/home/models.py

```
from django.db import models
from wagtail.models import Page
from wagtail.fields import StreamField, RichTextField
from wagtail.admin.panels import FieldPanel
from wagtail.search import index

from .blocks import StandardStreamBlock



class HomePage(Page):
    body = StreamField(
        StandardStreamBlock(),
        blank=True,
        use_json_field=True,
    )

    content_panels = Page.content_panels + [
        FieldPanel("body"),
    ]
```

```
        subpage_types = ["home.ContentPage", "home.EventsIndexPage"]
```

Реалізація універсальної контентної сторінки

website/home/models.py

```
class ContentPage(Page):
    body = StreamField(
        StandardStreamBlock(),
        blank=True,
        use_json_field=True,
    )

    content_panels = Page.content_panels + [
        FieldPanel("body"),
    ]

    parent_page_types = ["home.HomePage"]
```

Реалізація сторінки-розділу подій

website/home/models.py

```
class EventsIndexPage(Page):
    max_count = 1
    parent_page_types = ["home.HomePage"]
    subpage_types = ["home.EventPage"]
```

```
    class Meta:

        verbose_name = "Сторінка подій"
```

Реалізація моделі окремої події

website/home/models.py

```
class EventPage(Page):

    parent_page_types = ["home.EventsIndexPage"]

    subpage_types = []



    pinned = models.BooleanField(default=False, verbose_name="Закріпити
подію")

    frequency = models.CharField(max_length=255, blank=True,
verbose_name="Частота")

    time = models.CharField(max_length=255, blank=True,
verbose_name="Час")

    platform = models.CharField(max_length=255, blank=True,
verbose_name="Платформа")

    subtitle = models.CharField(max_length=500, blank=True,
verbose_name="Підзаголовок")

    body = RichTextField(blank=True, verbose_name="Повний опис")

    button_text = models.CharField(

        max_length=100,

        blank=True,

        default="Детальніше",

        verbose_name="Текст кнопки",

    )
```

```
    button_url = models.URLField(blank=True, verbose_name="Посилання
кнопки")

    event_date = models.DateField(null=True, blank=True,
verbose_name="Дата події")



    image = models.ForeignKey(

        "wagtailimages.Image",

        null=True,

        blank=True,

        on_delete=models.SET_NULL,

        related_name="+",

        verbose_name="Зображення",

    )
```

Налаштування панелей редагування події

website/home/models.py

```
content_panels = Page.content_panels + [

    FieldPanel("pinned"),

    FieldPanel("event_date"),

    FieldPanel("frequency"),

    FieldPanel("time"),

    FieldPanel("platform"),

    FieldPanel("subtitle"),

    FieldPanel("body"),

    FieldPanel("button_text"),
```

```
    FieldPanel("button_url"),
    FieldPanel("image"),
]

search_fields = Page.search_fields + [
    index.SearchField("subtitle"),
    index.SearchField("body"),
    index.SearchField("frequency"),
    index.SearchField("platform"),
]
```

Реалізація текстового блоку

website/home/blocks.py

```
from wagtail import blocks
from wagtail.images.blocks import ImageChooserBlock
from wagtail.documents.blocks import DocumentChooserBlock


class ParagraphBlock(blocks.StructBlock):
    text = blocks.RichTextBlock(
        required=True,
        label="Текст параграфа",
        features=["bold", "italic", "link", "ul", "ol", "blockquote",
"hr"]
    )
    font_size_value = blocks.IntegerBlock(
        required=False,
        default=16,
        label="Розмір шрифту (число)",
```

```
        help_text="Наприклад: 16"
    )
    font_size_unit = blocks.ChoiceBlock(
        choices=[("px", "px"), ("rem", "rem")],
        default="px",
        label="Одиниця виміру"
    )
    alignment = blocks.ChoiceBlock(
        choices=[
            ("left", "Вирівняно ліворуч"),
            ("center", "По центру"),
            ("right", "Вирівняно праворуч"),
        ],
        default="left",
        label="Вирівнювання"
    )

    class Meta:
        template = "home/blocks/paragraph.html"
        icon = "pilcrow"
        label = "Параграф"
```

Реалізація блоку модального вікна

website/home/blocks.py

```
class ModalBlock(blocks.StructBlock):
    modal_id = blocks.CharBlock(
        required=True,
        help_text="Унікальний ID для відкриття модалки"
    )
    title = blocks.CharBlock(required=True, label="Заголовок")
    subtitle = blocks.CharBlock(required=False, label="Підзаголовок")
    frequency = blocks.CharBlock(required=False, label="Частота (тег)")
```

```
    time = blocks.CharBlock(required=False, label="Час (тег)")
    platform = blocks.CharBlock(required=False, label="Платформа (тег)")
    image = ImageChooserBlock(required=False, label="Зображення")
    content = blocks.RichTextBlock(
        required=True,
        label="Контент (HTML)",
        features=["bold", "italic", "link", "ul", "ol", "hr"]
    )
    button_text = blocks.CharBlock(
        required=False,
        default="Приєднатися",
        label="Текст кнопки"
    )
    button_url = blocks.URLBlock(required=False, label="Посилання
кнопки")


    class Meta:
        template = "home/blocks/modal_block.html"
        icon = "placeholder"
        label = "Модальне вікно"
```

Реалізація блоку подій та анонсів через API

website/home/blocks.py

```
class EventsAnnouncementsApiBlock(blocks.StructBlock):
    title = blocks.CharBlock(
        required=False,
        default="Події та анонси",
        label="Заголовок секції"
    )
    api_url = blocks.CharBlock(
        required=False,
        default="/api/events/",
```

```
        label="API URL"
    )

    class Meta:
        template = "home/blocks/events_announcements_api.html"
        icon = "date"
        label = "Події та анонси (API)"
```

Об’єднання блоків у StandardStreamBlock

website/home/blocks.py

```
class StandardStreamBlock(blocks.StreamBlock):
    paragraph = ParagraphBlock()
    program_slider = ProgramSliderBlock()
    events_announcements = EventsAnnouncementsBlock()
    events_announcements_api = EventsAnnouncementsApiBlock()
    documents_accordion = DocumentsAccordionBlock()
    modal_window = ModalBlock()
    footer = FooterBlock()
    admission_program = AdmissionProgramBlock()

    class Meta:
        icon = "placeholder"
        label = "Основний контент"
```

Реалізація API-представлення подій

website/home/views.py

```
from django.http import JsonResponse
from django.utils.html import strip_tags

from .models import EventPage
```

```
def events_api(request):
    events = EventPage.objects.live().public().specific().order_by(
        "-pinned",
        "-first_published_at"
    )

    data = []
    for event in events:
        image_url = ""
        if event.image:
            image_url = event.image.file.url

        body_html = event.body if event.body else ""
        body_text = strip_tags(body_html) if body_html else ""

        data.append({
            "id": event.id,
            "title": event.title,
            "subtitle": event.subtitle,
            "frequency": event.frequency,
            "time": event.time,
            "platform": event.platform,
            "button_text": event.button_text or "Детальніше",
            "button_url": event.url,
            "pinned": event.pinned,
            "image_url": image_url,
            "body_html": body_html,
            "body_text": body_text,
        })

    return JsonResponse(
```

```
        data,
        safe=False,
        json_dumps_params={"ensure_ascii": False}
    )
```

Налаштування маршруту API подій

website/website/urls.py

```
from django.urls import include, path

from search import views as search_views
from home.views import events_api
from .api import api_router

urlpatterns = [
    path('django-admin/', admin.site.urls),
    path('accounts/', include('allauth.urls')),

    path('admin/', include(wagtailadmin_urls)),
    path('documents/', include(wagtaildocs_urls)),
    path('images/', include(wagtailimages_urls)),

    path('search/', search_views.search, name='search'),

    path('api/v2/', api_router.urls),
    path('api/events/', events_api, name='events_api'),

    path('', include(wagtail_urls)),
]
```

Налаштування Wagtail API

website/website/api.py

```
from wagtail.api.v2.router import WagtailAPIRouter
from wagtail.api.v2.views import PagesAPIViewSet
from wagtail.images.api.v2.views import ImagesAPIViewSet
from wagtail.documents.api.v2.views import DocumentsAPIViewSet

api_router = WagtailAPIRouter("wagtailapi")

api_router.register_endpoint("pages", PagesAPIViewSet)
api_router.register_endpoint("images", ImagesAPIViewSet)
api_router.register_endpoint("documents", DocumentsAPIViewSet)
```

Шаблон API-блоку подій

website/home/templates/home/blocks/events_announcements_api.html

```
{% load static %}

<section class="events-announcements-api" data-api-url="{{ value.api_url
}}">
  <div style="padding: 40px 20px; max-width: 1850px; margin: 0 auto;">
    <h1>{{ value.title }}</h1>

    <div class="events-api-list" id="events-api-list"></div>
  </div>
</section>

<div id="api-event-modal" class="modal">
  <div class="modal__overlay js-api-modal-close"></div>
  <div class="modal__dialog">
    <button class="modal__close js-api-modal-close" aria-label="Закрити">
      &times;
    </button>
```

```
    <div class="modal__top">
      <div class="modal__thumb" id="api-modal-image-wrap" style="display:
none;">
        <img id="api-modal-image" src="" alt="">
      </div>

      <div class="modal__header">
        <h2 id="api-modal-title"></h2>
        <p class="modal__subtitle" id="api-modal-subtitle"
style="display: none;"></p>
        <div class="modal__tags" id="api-modal-tags" style="display:
none;"></div>
      </div>
    </div>

    <div class="modal__content" id="api-modal-content"></div>
  </div>
</div>
```

Отримання даних подій через JavaScript

website/home/templates/home/blocks/events_announcements_api.html

```
document.addEventListener("DOMContentLoaded", function () {
  const section = document.querySelector(".events-announcements-api");
  if (!section) return;

  const apiUrl = section.dataset.apiUrl;
  const container = document.getElementById("events-api-list");

  let eventsData = [];

  fetch(apiUrl)
    .then(response => response.json())
    .then(data => {
```

```
      eventsData = data;

      container.innerHTML = data.map(event => `
        <div class="event-card ${event.pinned ? 'pinned' : ''}">
          ${event.image_url ? `
            <div class="event-card__image">
              <img src="${event.image_url}" alt="${event.title}">
            </div>
          ` : ''}

          <div class="event-card__content">
            <h2 class="event-card__title">${event.title}</h2>
            ${event.subtitle ? `
              <div class="event-card__subtitle">${event.subtitle}</div>
            ` : ''}
          </div>
        </div>
      `).join("");
    })
    .catch(error => {
      container.innerHTML = "<p>Не вдалося завантажити події.</p>";
      console.error(error);
    });
});
```