

БАКАЛАВРСЬКА РОБОТА

РБ. ІП - 25.00.00.000 ІЗ

Група ІП-22-1

Угринчук Арсен Миколайович

2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Угринчук Арсен Миколайович

(прізвище, ім'я, по батькові)

УДК 004.942

(індекс)

БАКАЛАВРСЬКА РОБОТА

Розроблення програмної системи управління контентом

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121– Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня **Угринчук А.М.**

(підпис, ініціали та прізвище здобувача)

Науковий керівник **Процюк Василь Романович, к.т.н., доцент**

(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026

Івано-Франківський національний технічний університет нафти і газу
Інститут, факультет інформаційних технологій
Кафедра інженерії програмного забезпечення
Ступінь вищої освіти бакалавр
Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедрою ІІЗ
доцент. В.В. Бандура
“ ” 2026 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Угринчуку Арсену Миколайовичу

(прізвище, ім'я, по-батькові)

1. Тема проекту (роботи) " Розроблення програмної системи управління контентом "

керівник проекту (роботи) Процюк Василь Романович, к.т.н., доцент

затверджені наказом вищого навчального закладу від “ 21 ” квітня 2026 р. № 179/7

2. Строк подання студентом проекту (роботи) 10 червня 2026р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

- 1 Вступ до проблеми управління новинним контентом у цифрових каналах
- 2 Огляд існуючих концепцій та сервісів у сфері систем управління контентом і Telegram-ботів
- 3 Побудова архітектури, моделі даних та алгоритмів роботи власного рішення
- 4 Документування реалізації програмної системи управління контентом вибраними засобами

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

- 1 Архітектура програмної системи управління контетнтном (рис. 3.1, ст. 18)
- 2 Сценарій проходження новини через систему (рис. 3.2, ст. 20)
- 3 Логічна модель бази даних (рис. 3.3, ст. 21)
- 4 Головне меню Telegram-бота (рис. 4.1, ст. 24)
- 5 Панель модерації адміністратора (рис. 4.3, ст. 28)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання 10 січня 2026 р.

Керівник

Процюк В.Р.

(підпис)

Завдання прийняв до виконання Угринчук А.М.

(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обґрунтування теми	15.02.2026	виконано
2	Аналіз предметної області та джерел	01.03.2026	виконано
3	Формування вимог і постановка задачі	15.03.2026	виконано
4	Проектування архітектури та БД	05.04.2026	виконано
5	Реалізація програмного продукту	25.04.2026	виконано
6	Оформлення пояснювальної записки бакалаврської роботи завідувачем кафедри	10.06.2026	виконано

Студент-дипломник

Угринчук А.М.

(підпис)

(розшифровка підпису)

Керівник роботи

Процюк В.Р.

(підпис)

(розшифровка підпису)

АНОТАЦІЯ

Бакалаврська робота містить 67 сторінок, 19 рисунків, 8 таблиць, список використаних джерел із 39 найменувань, 1 додаток.

Метою бакалаврської роботи є проєктування та реалізація програмної системи управління новинним контентом на основі Telegram-бота, що забезпечує приймання матеріалів, модерацію, публікацію в Telegram-канал і відображення схвалених новин на вебсайті.

Об'єкт дослідження: процес приймання, модерації, публікації та вебвідображення користувацького новинного контенту.

Предмет дослідження: методи, програмні засоби та архітектурні рішення для побудови Telegram-бота, бази даних, вебчастини і сервісів публікації.

Результати дослідження: реалізовано програмну систему управління новинним контентом для приймання, модерації та публікації новин.

У першому розділі розглянуто предметну область, сучасні підходи до управління цифровим контентом і технології створення Telegram-ботів.

У другому розділі визначено ролі користувачів, сформульовано функціональні та нефункціональні вимоги і постановку задачі.

У третьому розділі спроектовано архітектуру, бізнес-процес проходження новини та логічну модель бази даних.

У четвертому розділі описано реалізацію коду, медіаальбомів, публікації та AI-допоміжних функцій.

У п'ятому розділі розглянуто тестування, впровадження, резервне копіювання, моніторинг і ризику розвитку.

Висновки: отримані результати включають працездатний прототип системи, який підтримує подання матеріалів через Telegram, модерацію, публікацію в канал, створення вебпостів, журнал аудиту та обробку медіафайлів.

КЛЮЧОВІ СЛОВА: TELEGRAM-БОТ, УПРАВЛІННЯ КОНТЕНТОМ, МОДЕРАЦІЯ, PYTHON, POSTGRESQL, SQLALCHEMY, AIОНТТР, ВЕБСАЙТ, МЕДІАФАЙЛИ, ЖУРНАЛ АУДИТУ.

ABSTRACT

Bachelor's thesis contains 67 pages, 19 figures, 8 tables, a list of references consisting of 39 sources, and 1 appendix.

The purpose of the bachelor's thesis is to design and implement a software system for managing news content based on a Telegram bot that supports user submissions, moderation, publication to a Telegram channel, and display of approved news on a website.

Object of research: the process of receiving, moderating, publishing, and displaying user-generated news content on the web.

Subject of research: methods, software tools, and architectural solutions for building a Telegram bot, database, web module, and publication services.

Research results: a software system for news content management was implemented, supporting content submission, moderation, and publication.

The first chapter examines the domain area, modern approaches to digital content management, and technologies for Telegram bots.

The second chapter defines user roles, functional and non-functional requirements, and the system design task.

The third chapter presents the architecture, news processing workflow, and logical database model.

The fourth chapter describes code implementation, media album processing, publication, and AI-assisted functions.

The fifth chapter covers testing, deployment, backup, monitoring, and development risks.

Conclusions: the obtained results include a working prototype that supports Telegram-based submissions, moderation, channel publication, web post creation, audit logging, and media file processing.

KEYWORDS: TELEGRAM BOT, CONTENT MANAGEMENT, MODERATION, PYTHON, POSTGRESQL, SQLALCHEMY, AIOHTTP, WEBSITE, MEDIA FILES, AUDIT LOG.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	8
ВСТУП	9
РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ТЕОРЕТИЧНІ ОСНОВИ ПРОЄКТУВАННЯ СИСТЕМ УПРАВЛІННЯ КОНТЕНТОМ	11
1.1. Аналіз сучасного стану проблематики	11
1.2. Основні поняття та визначення предметної області	13
1.3. Сучасні технології та тенденції, що застосовуються в даній області	15
1.4. Порівняльний аналіз підходів до побудови системи управління контентом	17
1.5. Висновок по розділу	19
РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА ПОСТАНОВКА ЗАДАЧІ	21
2.1. Збір та аналіз вимог користувачів	21
2.2. Функціональні та нефункціональні вимоги	23
2.3. Постановка задачі проєктування системи	25
2.4. Сценарії використання системи та критерії приймання результату	27
2.5. Висновки по розділу	29
РОЗДІЛ 3. ПРОЄКТУВАННЯ СИСТЕМИ УПРАВЛІННЯ КОНТЕНТОМ.....	31
3.1. Розробка архітектури програмного забезпечення	31
3.2. Моделювання бізнес-процесів та системних компонентів	33
3.3. Проєктування бази даних та обґрунтування технологій.....	35
3.4. Висновки по розділу	38
РОЗДІЛ 4. РЕАЛІЗАЦІЯ ПРОЄКТУ УПРАВЛІННЯ КОНТЕНТОМ.....	39
4.1. Структура програмного коду	39
4.2. Реалізація основних алгоритмів та структур даних	42

					БР.ІІ – 25.00.00.000 ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Угринчук А.М.			Розроблення програмної системи управління контентом Пояснювальна записка	Літ.	Арк.	Акрушів
Перевір.		Процюк В.Р.						71
Реценз.		Бандура В.В.				ІФНТУНГ ІІ-22-1		
Н. Контр.		Піх М.М.						
Затверд.		Бандура В. В.						

4.3. Реалізація інтерфейсів користувача та адміністратора	44
4.4. Реалізація публікації, AI-допоміжних функцій та release-процесу.....	46
4.5. Обробка помилок, журналювання та відновлення операцій	49
4.6. Висновки по розділу	51

РОЗДІЛ 5. ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ, СУПРОВІД ТА НАПРЯМИ РОЗВИТКУ ПРОЄКТУ..... 53

5.1. Стратегії, методи та результати тестування	53
5.2. Впровадження, release-перевірки та експлуатація системи	55
5.3. Захист даних, резервне копіювання та моніторинг.....	58
5.4. Масштабування, практична доцільність і ризики розвитку	61
5.5. Експериментальна оцінка продуктивності вебчастини	64
5.6. Висновки по розділу	66

ВИСНОВКИ 68

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 70

ДОДАТКИ

БІБЛІОГРАФІЧНА ДОВІДКА

					БР.ІІ - 25.00.00.000 ПЗ	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API – інтерфейс прикладного програмування.

БД – база даних.

ПЗ – програмне забезпечення.

СУБД – система управління базами даних.

FSM – скінченний автомат станів.

HTML – мова гіпертекстової розмітки.

JSON – текстовий формат обміну даними.

ORM – об’єктно-реляційне відображення.

URL – уніфікований показчик ресурсу.

					БР.ІІ - 25.00.00.000 ПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

У сучасному інформаційному суспільстві оперативність поширення новин відіграє важливу роль у забезпеченні громадськості актуальною та достовірною інформацією. Стрімкий розвиток цифрових технологій, соціальних мереж і месенджерів призвів до того, що значна частина новинного контенту створюється та передається безпосередньо користувачами. Сьогодні будь-яка людина може швидко повідомити про подію, надіслати фотографії чи відеоматеріали та стати джерелом інформації для медіаресурсів. Одним із найпоширеніших інструментів для такої взаємодії є Telegram, який поєднує зручність обміну повідомленнями, підтримку мультимедійного контенту та широкі можливості автоматизації через використання ботів.

Актуальність теми зумовлена необхідністю створення програмної системи управління новинним контентом на основі Telegram-бота, яка забезпечить зручний, швидкий та ефективний механізм приймання, модерації та публікації користувацьких повідомлень. Незважаючи на наявність різноманітних платформ для створення чат-ботів, систем управління контентом та сервісів публікації інформації, більшість із них орієнтовані на виконання окремих функцій і не забезпечують комплексної інтеграції процесів приймання матеріалів, редакційної перевірки, публікації в Telegram-каналі та відображення контенту на вебсайті. Особливо актуальною ця проблема є для локальних новинних проєктів, де оперативність опрацювання інформації безпосередньо впливає на якість інформаційного обслуговування аудиторії.

Метою роботи є розроблення програмної системи управління новинним контентом на основі Telegram-бота, яка забезпечує приймання матеріалів від користувачів, модерацію, збереження медіафайлів, публікацію повідомлень у Telegram-каналі та автоматичне відображення схвалених новин на вебсайті.

Завданнями дослідження є вивчення сучасних підходів до побудови систем управління контентом і Telegram-ботів, аналіз існуючих рішень для приймання та публікації новинних матеріалів, визначення потреб користувачів та

					БР.ІІ - 25.00.00.000 ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

адміністраторів, виявлення основних функціональних і нефункціональних вимог, проектування архітектури програмної системи, розробка моделі бази даних, реалізація Telegram-бота та вебчастини системи, тестування основних сценаріїв роботи та оцінювання працездатності розробленого рішення.

Об'єктом дослідження є процес приймання, модерації, публікації та вебвідображення користувацького новинного контенту.

Предметом дослідження є програмні засоби, технології та інструменти, що застосовуються на етапах життєвого циклу розробки програмної системи управління новинним контентом, зокрема аналіз і формалізація вимог користувачів, проектування архітектури програмного забезпечення, моделювання структури бази даних, реалізація функціональних можливостей Telegram-бота та вебзастосунку, тестування й оцінка працездатності системи.

Методи: аналіз предметної області, порівняльний аналіз існуючих систем управління контентом і Telegram-ботів, моделювання вимог, об'єктно-орієнтоване проектування, реляційне моделювання бази даних, функціональне тестування, аналіз сценаріїв користувача та оцінювання ефективності роботи системи. Використання цих методів дало змогу обґрунтувати структуру програмної системи, визначити її функціональні можливості та перевірити працездатність реалізованого програмного продукту.

Розроблена програмна система передбачатиме можливість подання новинних матеріалів через Telegram-бота, завантаження текстових повідомлень і медіафайлів, автоматизованого збереження інформації в базі даних, модерації контенту адміністраторами, публікації схвалених новин у Telegram-каналі, відображення матеріалів на вебсайті, ведення журналу аудиту дій користувачів та адміністраторів, а також використання допоміжних AI-функцій для підтримки процесу обробки контенту. Особлива увага приділена забезпеченню централізованого управління новинними матеріалами та інтеграції між Telegram-платформою і вебресурсом.

Бакалаврська робота містить 67 сторінок, 19 рисунків, 8 таблиць, список використаних джерел із 39 найменувань та 1 додаток.

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ТЕОРЕТИЧНІ ОСНОВИ ПРОЄКТУВАННЯ СИСТЕМ УПРАВЛІННЯ КОНТЕНТОМ

1.1. Аналіз сучасного стану проблематики

Новинний контент у цифровому середовищі характеризується високою швидкістю появи, коротким життєвим циклом актуальності та потребою у перевірці перед публікацією. Якщо раніше основним каналом взаємодії аудиторії з редакцією була електронна пошта або форма на сайті, то нині значна частина оперативних повідомлень надходить через месенджери, групові чати та приватні звернення. Для користувача месенджер є звичним середовищем: він не потребує встановлення окремого застосунку, дозволяє швидко прикріпити фото чи відео та отримати відповідь про статус звернення. Для адміністратора така модель створює іншу проблему: повідомлення з різних чатів важко систематизувати без програмного шару, який переводить їх у структуровані заявки.

У класичних системах керування контентом, таких як веборієнтовані CMS, основна увага приділяється публікації матеріалів редактором через адміністративну панель. Проте для задачі локального новинного бота важливою є саме перша ланка редакційного процесу: приймання матеріалу від зовнішнього користувача, фіксація автора, збереження прикріплених файлів, присвоєння статусу та подальше рішення адміністратора. Тому розроблювана система поєднує ознаки чат-бота, міні-CMS, медіасховища та вебвітрини. Така комбінація дозволяє розглядати її не як простий Telegram-скрипт, а як комплексне програмне забезпечення з окремими шарами відповідальності.

Для локальних медіа особливо важливо, щоб канал приймання повідомлень був максимально близьким до повсякденних звичок аудиторії. Користувачі частіше надсилають фото, відео або короткий опис події через месенджер, ніж заповнюють окрему вебформу. Тому Telegram-бот є не лише технічним інтерфейсом, а й способом зменшити бар'єр між джерелом інформації

					БР.ІІ - 25.00.00.000 ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		

та редакцією. Водночас така зручність створює додаткові вимоги до контролю якості: система повинна

відокремлювати факт надходження матеріалу від факту його публікації.

Порівняно з ручною роботою через приватні повідомлення, структурована система дає редактору кілька переваг. Кожна заявка отримує статус, автора, дату створення і набір пов'язаних медіафайлів. Це дозволяє не втрачати матеріали, уникати дублювання, переглядати історію рішень і швидше знаходити новини, які очікують модерації. Саме ця потреба пояснює доцільність створення окремої програмної системи, а не використання лише стандартних можливостей Telegram.

Суттєвим аспектом сучасних редакційних систем є контроль якості даних. Повідомлення, надіслане користувачем, не може автоматично потрапляти до публічного каналу без перевірки, оскільки воно може містити неточну інформацію, помилкові медіафайли, образливий текст або дублювати вже опублікований матеріал [11; 34]. З цієї причини модель статусів новини є базовим елементом системи. Новина проходить стани створення, очікування модерації, схвалення або відхилення. Адміністратор повинен мати змогу переглянути матеріал у повному вигляді, відредагувати текст, залишити коментар при відмові та опублікувати матеріал в обраний канал.

Окрему роль відіграє мультимедійність. У Telegram користувач часто надсилає не один файл, а групу фотографій або відео з підписом. Якщо система не враховує альбоми, частина файлів може бути опрацьована як окремі новини або втрачена. У реалізованому проєкті застосовано буферизацію медіагруп з короткою затримкою, що дає змогу зібрати всі елементи альбому й сформувані одну заявку. Це практичне рішення покращує повноту даних та робить поведінку бота ближчою до природного сценарію використання Telegram.

Проблематика також пов'язана з прозорістю редакційного процесу. Якщо адміністратор працює лише з приватними повідомленнями, складно зрозуміти, які матеріали вже переглянуті, які відхилені, які опубліковані, а які потребують доопрацювання. Наявність централізованої бази даних і журналу дій дозволяє відновити історію роботи з новиною, проконтролювати рішення адміністратора та

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		12

уникнути дублювання матеріалів. Це особливо важливо для невеликих локальних медіа, де одна людина може одночасно виконувати роль редактора, модератора і технічного адміністратора.

Таким чином, актуальність розроблення системи полягає не лише у створенні Telegram-бота як зручного інтерфейсу. Основна цінність полягає в організації повного керованого циклу: приймання, збереження, модерація, публікація, вебвідображення та подальший супровід. Саме така послідовність робить систему корисною для практичного використання, оскільки вона зменшує ручну роботу адміністратора, підвищує контроль якості та створює основу для подальшого масштабування.

1.2. Основні поняття та визначення предметної області

Telegram-бот у контексті цієї роботи розглядається як програмний агент, що взаємодіє з користувачами через Telegram Bot API та виконує дії відповідно до отриманих повідомлень і callback-запитів [14]. На відміну від звичайного чату, бот має формалізовану логіку станів, кнопки, обмеження прав доступу та інтеграцію із зовнішніми сервісами. Для реалізації асинхронної взаємодії використано бібліотеку aiogram, яка надає маршрутизатори, фільтри, middleware та FSM-механізм для багатокрокових сценаріїв [15; 29].

Новинна заявка є основною сутністю системи. Вона містить текст, автора, статус, дату створення, дату оновлення, адміністративний коментар і пов'язані медіафайли. Медіафайл описується типом, шляхом локального збереження, Telegram file_id та позицією у наборі. Така структура дозволяє відновити послідовність матеріалів, повторно використати Telegram-ідентифікатор для публікації й одночасно мати локальну копію для вебсайту. Розділення текстової заявки та медіафайлів відповідає принципам нормалізації бази даних.

Модерація у системі означає керований процес прийняття рішення щодо матеріалу. На цьому етапі адміністратор отримує попередній перегляд, може перегортати список заявок, відкривати деталі, редагувати текст, відхиляти

					БР.ІІ - 25.00.00.000 ПЗ	Арк.
						13
Змн.	Арк.	№ докум.	Підпис	Дата		

матеріал із поясненням або запускати публікацію. Важливим є те, що модерація відбувається у тому самому середовищі Telegram, тобто адміністратору не потрібно переходити до окремої вебпанелі для базових дій. Це зменшує кількість контекстних перемикачів і пришвидшує роботу.

Публікація в межах проєкту має два напрями: Telegram-канал і вебсайт. Telegram-канал забезпечує миттєве поширення матеріалу підписникам, а вебсайт створює архів, сторінки деталізації та API для отримання списку постів. Дзеркалення новини на сайт виконується після схвалення адміністратором. Для формування вебсторінки із тексту новини генерується заголовок, короткий опис і slug, що використовується як частина URL. Такий підхід робить одну редакційну дію корисною для кількох каналів поширення.

Окремим поняттям є редакційний цикл. У цій роботі він охоплює всі дії з матеріалом: отримання повідомлення, створення заявки, збереження медіа, перевірку адміністратором, зміну статусу, публікацію, фіксацію результату в базі даних і можливий подальший супровід. Такий цикл показує, що розроблена система не обмежується однією функцією надсилання повідомлення в канал. Вона описує повний шлях даних і забезпечує контроль на кожному етапі.

Система управління контентом у межах цієї роботи розглядається як програмний комплекс, що забезпечує створення, збереження, перевірку, публікацію і подальше відображення матеріалів. На відміну від класичної CMS, де основним користувачем є редактор, у розробленій системі перша дія належить зовнішньому користувачу Telegram. Тому модель повинна враховувати не лише адміністративне редагування, а й приймання неструктурованих повідомлень від аудиторії.

Важливими поняттями також є заявка, медіагрупа, статус модерації та канал публікації. Заявка фіксує намір користувача передати матеріал редакції. Медіагрупа дозволяє пов'язати кілька фото або відео з одним повідомленням. Статус модерації показує етап обробки матеріалу, а канал публікації визначає, куди саме буде передано схвалену новину. Таке уточнення понять робить предметну область чіткішою і спрощує подальше проєктування бази даних.

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						14
Змн.	Арк.	№ докум.	Підпис	Дата		

Важливою характеристикою предметної області є наявність різних ролей. Звичайний користувач очікує простого способу подати матеріал. Адміністратор очікує впорядкованої черги, кнопок керування, можливості редагування і зрозумілого зворотного зв'язку. Читач вебсайту очікує перегляду вже схвалених новин у зручному форматі. Ці ролі визначають функціональні межі системи й пояснюють, чому в архітектурі поєднано Telegram-бота, базу даних, сервісний шар і вебчастину.

1.3. Сучасні технології та тенденції, що застосовуються в даній області

Асинхронне програмування є однією з ключових технологічних основ сучасних ботів і вебсервісів [28]. Система повинна одночасно обробляти повідомлення користувачів, callback-запити адміністраторів, завантаження медіафайлів, запити до бази даних і відповіді вебсервера. Використання `async/await` у Python дозволяє ефективно працювати з операціями введення-виведення без блокування всього процесу [16; 28]. У проєкті цей підхід застосовано в `aiogram`, `SQLAlchemy` `asyncio`, `asyncpg` та `aiohhttp`.

Сучасні системи управління контентом дедалі частіше будуються за принципом `API-first`. Це означає, що дані не прив'язуються лише до одного інтерфейсу, а можуть використовуватися Telegram-ботом, вебсайтом, адміністративною панеллю або зовнішніми сервісами. У межах розробленого проєкту цю ідею реалізовано частково: вебчастина має HTML-сторінки та JSON API, а Telegram-бот працює з тією самою базою даних через сервісний і repository-рівні.

Ще однією тенденцією є використання допоміжних AI-функцій у редакційній роботі. Такі інструменти можуть скорочувати текст, пропонувати заголовки або покращувати стиль повідомлення, але остаточне рішення має залишатися за адміністратором. Для бакалаврського проєкту важливо показати саме контрольоване використання AI: модель допомагає редактору, однак не має права самостійно публікувати матеріали без модерації.

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		15

Іншою важливою тенденцією є поділ відповідальностей між шарами. Обробники Telegram-подій не повинні безпосередньо містити всю логіку роботи з базою даних, генерації вебпублікацій або збереження медіа. Тому в проєкті виділено пакети handlers, services, db, middlewares, keyboards, states та web. Така організація полегшує тестування, супровід і подальше розширення [6]. Наприклад, публікація на сайт винесена у service-рівень, а доступ до таблиць реалізовано через repository-класи.

Для зберігання даних обрано PostgreSQL, що добре підходить для транзакційних застосунків із реляційними зв'язками [8; 19]. SQLAlchemy використовується як ORM-рівень, який дозволяє описувати таблиці у вигляді Python-класів і виконувати асинхронні запити [17; 30]. Alembic забезпечує міграції схеми бази даних, тобто контрольовану зміну структури таблиць у процесі розвитку системи [18; 31]. Це особливо важливо для бакалаврського проєкту, оскільки наявність міграцій демонструє не лише прототип інтерфейсу, а й інженерний підхід до життєвого циклу даних.

У системі також передбачено інтеграцію з генеративним AI-сервісом для покращення тексту, скорочення матеріалу, створення заголовків і відповіді на запити користувача [23]. Такий модуль не є обов'язковим для базового приймання новин, однак підвищує практичну цінність продукту для редактора. Важливо, що AI-логіка винесена в окремий сервіс і не порушує базовий сценарій роботи системи: у разі розширення або заміни моделі достатньо змінити відповідний модуль без переробки обробників Telegram.

Для вебчастини використано серверний HTML-рендеринг. У межах прототипу це рішення є достатнім, оскільки сайт виконує роль публічної вітрини з картками новин, сторінками деталізації та JSON API. HTML використовується для структури сторінок, CSS - для зовнішнього вигляду, а aiohttp - для маршрутизації та обробки HTTP-запитів [21; 24; 25; 32]. Такий підхід зменшує складність порівняно з окремим frontend-фреймворком і дозволяє зберегти весь проєкт у межах одного Python-стеку.

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						16
Змн.	Арк.	№ докум.	Підпис	Дата		

З погляду сучасних практик розроблення важливими є також конфігурація, безпека і супровід. Конфігурація системи реалізована через pydantic-settings, що дозволяє типізовано описати параметри bot_token, admin_ids, database_url, gemini_api_key, media_dir, site_host, site_port і site_public_url [22; 33]. Секрети не повинні жорстко записуватися в код, а користувацький текст перед відображенням на сайті має екрануватися, щоб зменшити ризик XSS-атак [11; 34]. Таким чином, вибрані технології відповідають не лише функціональним задачам, а й вимогам підтримуваності та безпеки.

1.4. Порівняльний аналіз підходів до побудови системи управління контентом

Під час проєктування програмної системи управління контентом важливо визначити, чому для поставленої задачі недостатньо використати лише готову CMS, окремий Telegram-канал або просту форму зворотного зв'язку. Кожен із цих підходів має власні переваги, однак вони не повністю покривають потребу у прийманні користувацьких матеріалів, їх модерації, роботі з медіафайлами, публікації у Telegram-канал і подальшому відображенні на вебсайті. Саме тому доцільним є розгляд кількох можливих архітектурних варіантів і вибір такого, який відповідає умовам бакалаврського проєкту та практичним потребам локального новинного сервісу [3; 4].

Перший можливий підхід полягає у використанні класичної системи керування контентом, наприклад WordPress, Joomla або Drupal. Такі системи надають готові механізми створення сторінок, редагування матеріалів, керування користувачами, публікації записів і підключення розширень. Їхньою перевагою є зрілість екосистеми, велика кількість шаблонів і наявність адміністративної панелі. Проте класична CMS насамперед орієнтована на редактора, який самостійно створює матеріал у вебінтерфейсі. Для сценарію, коли новина надходить від звичайного користувача через месенджер,

					БР.ІІ - 25.00.00.000 ПЗ	Арк.
						17
Змн.	Арк.	№ докум.	Підпис	Дата		

така модель потребує додаткових інтеграцій, плагінів або окремого проміжного сервісу [24; 25].

Другий підхід полягає у використанні тільки Telegram-каналу та ручної модерації повідомлень. У цьому випадку користувачі можуть надсилати матеріали адміністратору в приватні повідомлення або груповий чат, а редактор вручну копіює текст і медіа в канал. Такий спосіб є простим на початку роботи, але швидко створює проблеми зі структуризацією. Повідомлення губляться серед інших діалогів, складно відстежити статус кожної заявки, немає єдиного журналу дій, а повторна публікація або помилкове відхилення стають більш імовірними. Крім того, Telegram-канал не забезпечує повноцінного вебархіву, сторінок деталізації та програмного API для отримання списку опублікованих матеріалів [14].

Третій підхід передбачає використання вебформи або no-code/low-code платформ для подання новин і створення простих ботів. Такі рішення можуть швидко закрити базові сценарії: приймання тексту, додавання файлів, налаштування кнопок і простих відповідей. Проте для даної роботи вони мають обмеження, оскільки система повинна підтримувати асинхронну обробку медіагруп, реляційну модель даних, окремі таблиці для новин, медіафайлів, публікацій, каналів і журналу аудиту, а також контроль над логікою модерації та публікації [6; 8; 24].

Порівняння цих підходів показує, що найдоцільнішим для поставленої задачі є власне програмне рішення на основі Telegram-бота, бази даних, сервісного шару та вебчастини. Telegram-бот виконує роль зручного інтерфейсу для користувача й адміністратора, база даних забезпечує стійке збереження структурованих даних, сервісний шар ізолює бізнес-логіку, а вебчастина відповідає за публічне відображення схвалених матеріалів. Така архітектура дозволяє уникнути надмірної складності великої CMS і водночас не обмежуватися ручним веденням каналу.

Важливим критерієм вибору підходу є можливість контролювати життєвий цикл новини. У готовій CMS основний цикл часто починається з дії

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						18
Змн.	Арк.	№ докум.	Підпис	Дата		

редактора, тоді як у розроблюваній системі першим кроком є повідомлення звичайного користувача. У ручному Telegram-процесі життєвий цикл існує лише умовно і залежить від дисципліни адміністратора. У власній системі кожна заявка має статус, дату створення, автора, пов'язані медіафайли, адміністративний коментар і можливий запис публікації.

Отже, порівняльний аналіз підтверджує доцільність власної системи на основі Telegram-бота. Вона краще відповідає задачі оперативного приймання новин, ніж класична CMS, забезпечує більше контролю, ніж ручна модерація в Telegram, і є гнучкішою за типові no-code рішення. Такий висновок безпосередньо підводить до формування вимог і постановки задачі, які розглядаються у другому розділі.

1.5. Висновки по розділу

У першому розділі проведено аналіз предметної області систем управління новинним контентом і визначено особливості роботи локальних цифрових медіа. Встановлено, що Telegram є зручним каналом приймання користувацьких матеріалів, оскільки користувач може швидко надіслати текст, фото або відео без встановлення окремого застосунку чи заповнення складної вебформи. Водночас ручна робота з повідомленнями не забезпечує достатнього рівня контролю, систематизації та прозорості редакційного процесу, особливо коли кількість заявок зростає.

Розглянуто основні поняття предметної області: Telegram-бот, новинна заявка, медіафайл, медіагрупа, модерація, публікація, вебпост і редакційний цикл. Показано, що кожне з цих понять має практичне значення для побудови системи. Новинна заявка дозволяє фіксувати матеріал у структурованому вигляді, медіагрупа забезпечує коректну роботу з кількома фото або відео, статус модерації показує етап обробки матеріалу, а вебпост дає змогу відображати схвалений контент на сайті.

Обґрунтовано, що розроблювана система повинна поєднувати функції чат-бота, міні-CMS, медіасховища, адміністративної модерації та вебвідображення

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						19
Змн.	Арк.	№ докум.	Підпис	Дата		

схвалених матеріалів. Така комбінація є доцільною, оскільки окремий Telegram-канал не забезпечує повного контролю над заявками, а класична CMS не завжди зручна для швидкого приймання матеріалів від користувачів через месенджер. Тому власне програмне рішення краще відповідає задачі оперативного приймання, перевірки та публікації новинного контенту.

Також проаналізовано технологічні тенденції, зокрема асинхронне програмування, серверний HTML-рендеринг, API-підхід, ORM-рівень, міграції бази даних і допоміжне використання AI. Встановлено, що асинхронний підхід є важливим для системи, яка працює з Telegram API, базою даних, медіафайлами та вебзапитами. ORM-рівень і міграції бази даних спрощують супровід структури даних, а серверний HTML-рендеринг дозволяє реалізувати вебчастину без надмірного ускладнення архітектури.

У результаті визначено, що власна програмна система на основі Telegram-бота є найбільш доцільним рішенням для поставленої задачі. Вона дозволяє поєднати зручність месенджера для користувача з контрольованим адміністративним процесом для редактора. Такий підхід створює основу для подальшого проєктування архітектури, моделі даних, функціональних вимог і сценаріїв використання.

Отже, результати першого розділу підтверджують доцільність розроблення власної програмної системи, яка забезпечує повний шлях новини від користувацького повідомлення до публікації та подальшого перегляду на вебсайті. Проведений аналіз став основою для формування вимог до системи, визначення її меж, вибору технологій і подальшого проєктування програмного рішення.

					БР.ІІ - 25.00.00.000 ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА ПОСТАНОВКА ЗАДАЧІ

2.1. Збір та аналіз вимог користувачів

Основними зацікавленими сторонами системи є звичайні користувачі Telegram, адміністратори новинного каналу та потенційні читачі вебсайту. Користувач очікує швидкого і зрозумілого способу подати новину без складної реєстрації. Адміністратор очікує впорядкованого списку заявок, зручного перегляду медіа, можливості редагування та контролю публікації. Читач вебсайту очікує перегляду вже схвалених матеріалів у зрозумілому форматі. Ці ролі утворюють межі системи та визначають пріоритети розробки.

Для уточнення вимог доцільно розглядати кожен роль через її цілі та обмеження. Звичайний користувач не повинен знати внутрішню структуру системи, тому для нього важлива простота: надіслати текст, фото або відео, підтвердити заявку і отримати зрозумілу відповідь. Адміністратор, навпаки, потребує інструментів контролю: перегляд черги, попередній перегляд, редагування, відхилення, вибір каналу і фіксація результату. Читач вебсайту взаємодіє лише з уже схваленим контентом, тому для нього важливими є доступність сторінок, коректне відображення тексту і медіа.

Такий поділ ролей дозволяє уникнути змішування вимог. Функції, потрібні адміністратору, не повинні ускладнювати користувацький сценарій, а вебчастина не повинна відкривати можливість редагування матеріалів. Завдяки цьому межі системи стають зрозумілими: Telegram-бот приймає і модерує матеріали, база даних зберігає стан процесу, а вебсайт показує тільки опубліковані записи.

Аналіз сценаріїв показує, що користувацька частина повинна бути максимально короткою. Якщо бот змушує користувача проходити багато кроків, імовірність незавершеної заявки зростає. Тому сценарій подання новини побудовано навколо трьох дій: введення тексту або надсилання медіа з підписом, додавання додаткових файлів за потреби та підтвердження. Система повинна обробляти як текстові повідомлення, так і одиночні фото, відео або медіагрупи.

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						21
Змн.	Арк.	№ докум.	Підпис	Дата		

Адміністративний сценарій має іншу логіку. Адміністратор працює з чергою матеріалів, тому потрібна пагінація, короткий список заявок і детальний перегляд конкретної новини. Після перегляду адміністратор повинен мати кнопки для схвалення, публікації в канал, публікації на сайт, редагування або відхилення. Для відхилення необхідно зберегти причину, щоб автор отримав зрозумілий зворотний зв'язок. Для схвалення необхідно зберегти факт публікації та пов'язані Telegram message_id.

Вебчастина системи не призначена для редагування. Її завданням є публічне відображення схвалених новин, формування головної сторінки, сторінок деталізації та JSON API. Це зменшує складність реалізації й одночасно створює корисний канал поширення матеріалів. Вимога до вебчастини полягає у безпечному екрануванні HTML, коректному формуванні URL-адрес і роботі з локально збереженими медіафайлами.

Під час аналізу вимог було враховано, що система має працювати як прототип для локального новинного проєкту, а не як велика промислова CMS. Тому пріоритет отримали функції, які забезпечують основний редакційний цикл: приймання заявки, збереження даних, модерація, зміна статусу, публікація й перегляд результату. Додаткові можливості, такі як складна вебадмінпанель, категорії, ролі або планування публікацій, визначено як перспективні напрями розвитку, але не як критичні вимоги першої версії.

Важливою вимогою є зрозумілість інтерфейсу. Telegram-бот повинен вести користувача через мінімальну кількість дій, а адміністратору потрібно надати кнопки, які прямо відповідають редакційним операціям. Це дозволяє зменшити кількість текстових команд, знизити ризик помилок і зробити систему придатною для використання навіть без спеціальної технічної підготовки. Саме тому в роботі приділено увагу inline-клавіатурам, callback-запитам і FSM-станам.

2.2. Функціональні та нефункціональні вимоги

Функціональні вимоги визначають, які дії система повинна виконувати.

					БР.ІІ - 25.00.00.000 ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

Для користувача це запуск бота, вибір мови, подання новини, додавання фото або відео, підтвердження заявки, перегляд власних новин і звернення до адміністратора. Для адміністратора це перегляд панелі, робота з новинами на модерації, редагування тексту, відхилення із коментарем, вибір каналу, публікація, керування каналами, перегляд повідомлень і статистики. Для вебчастини це відображення головної сторінки, деталізації новини, медіафайлів, статичних стилів і endpoint `/api/posts`. Приклад реалізації функціональних вимог адміністратора, пов'язаних з управлінням опублікованим контентом, наведено на рисунку 2.1 та на UML діаграмі варіантів використання на рисунку 2.2.

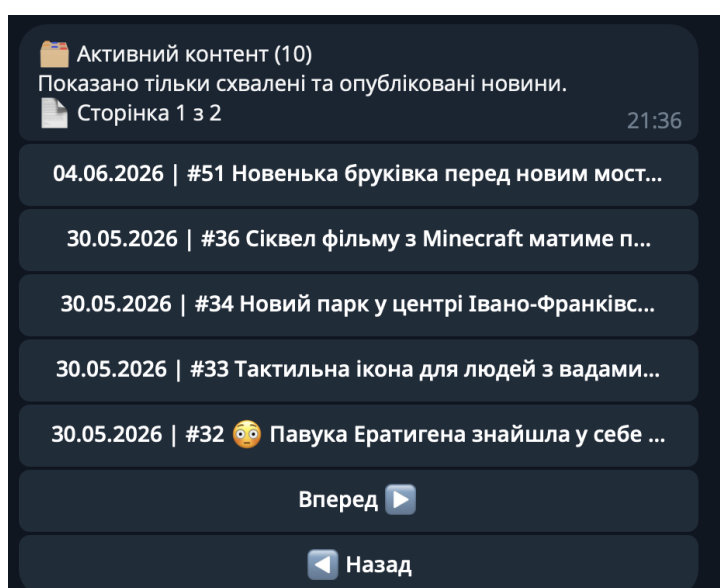


Рисунок 2.1 – Приклад адміністративного управління контентом у Telegram-боті

До нефункціональних вимог належать надійність, підтримуваність, безпека, продуктивність і зручність використання. Надійність забезпечується транзакційним збереженням даних, обробкою помилок під час публікації та журналюванням адміністративних дій. Підтримуваність забезпечується модульною структурою коду, розділенням репозиторіїв і сервісів, використанням міграцій Alembic. Безпека реалізується через фільтр адміністратора, екранування HTML у вебчастині, зберігання конфігурації в змінних середовища та обмеження частоти повідомлень.

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		23

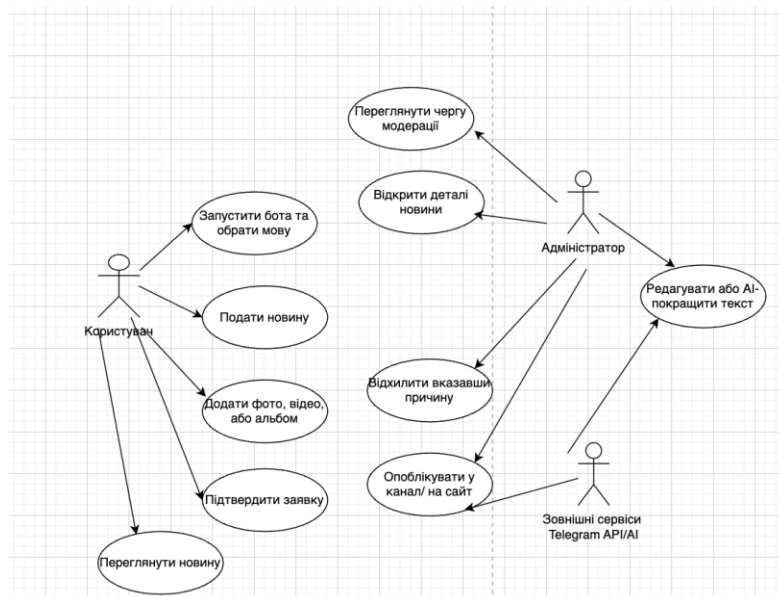


Рисунок 2.2 – UML-діаграма варіантів використання системи управління новинним контентом.

Продуктивність системи оцінюється з урахуванням асинхронної природи застосунку. Основними потенційно повільними операціями є запити до Telegram API, завантаження медіафайлів, звернення до бази даних і генерація HTML-відповідей. Оскільки ці операції виконуються через асинхронні бібліотеки, бот може обслуговувати кількох користувачів без створення окремого потоку на кожне повідомлення.

Вимоги до зручності передбачають мінімальну кількість текстових команд. Користувач повинен взаємодіяти переважно через кнопки, а адміністратор повинен бачити зрозумілі дії у кожному стані. Також система повинна очищувати службові повідомлення у чаті, щоб не засмічувати діалог. У коді це реалізовано через допоміжні функції `safe_delete`, `safe_delete_by_id` та `cleanup_state_messages`, які застосовуються в користувацьких і адміністративних сценаріях. Для систематизації функціональних і нефункціональних вимог до системи сформовано перелік критеріїв їх перевірки, наведений у таблиці 2.1.

Вимоги до системи та критерії їх перевірки

Тип вимоги	Зміст вимоги	Критерій перевірки
Функціональні	Подання новини користувачем, модерація адміністратором, публікація в Telegram-канал і на сайт	Заявка проходить шлях від створення до публікації або відхилення
Надійність	Транзакційне збереження даних, журналювання дій і коректна обробка помилок публікації	При помилці Telegram API статус не змінюється на approved
Безпека	Доступ до адміністративних дій лише для Telegram ID зі списку admin_ids, екранування HTML	Звичайний користувач не бачить і не виконує admin-кнопки
Зручність	Взаємодія через кнопки, короткі сценарії та очищення службових повідомлень у чаті	Діалог залишається читабельним після завершення сценарію

2.3. Постановка задачі проєктування системи

Постановка задачі полягає у створенні програмного продукту, який забезпечує повний цикл роботи з користувацькою новиною: від надсилання матеріалу до його публічного відображення. Вхідними даними є Telegram-повідомлення користувачів, медіафайли, callback-команди адміністраторів і конфігураційні параметри середовища. Вихідними даними є записи у базі даних, повідомлення у Telegram-каналі, вебсторінки новин, JSON-відповіді API, повідомлення авторам і записи журналу аудиту.

Окремо задача передбачає підтримку двох взаємопов'язаних сценаріїв: користувацького та адміністративного. Користувацький сценарій повинен бути максимально простим і складатися з подання тексту або медіафайлів, підтвердження заявки та отримання повідомлення про результат. Адміністративний сценарій, навпаки, має забезпечувати повний контроль над матеріалом: перегляд черги, відкриття деталей новини, редагування тексту, відхилення із зазначенням причини, вибір каналу та публікацію. Такий поділ сценаріїв дозволяє узгодити зручність для звичайного користувача з потребою адміністратора контролювати якість контенту [3; 4; 10].

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						25
Змн.	Арк.	№ докум.	Підпис	Дата		

Система повинна мати межі відповідальності. Telegram-бот відповідає за діалог із користувачами та адміністраторами; база даних відповідає за стійке збереження сутностей; сервіс публікації відповідає за коректне надсилення повідомлень у канал; вебсервіс відповідає за відображення схвалених матеріалів. Такий поділ дозволяє змінювати окремі частини без повної перебудови проєкту. Наприклад, у майбутньому можна замінити локальне зберігання медіа на зовнішнє сховище, не змінюючи користувацький сценарій подання новини.

До обмежень проєкту належить використання polling-режиму для бота, локального файлового сховища медіа та серверного HTML-рендерингу без окремого JavaScript-фреймворку. Ці рішення обрано свідомо, оскільки вони зменшують складність прототипу й дозволяють зосередитися на редакційному циклі. У майбутньому систему можна перевести на webhook-режим, додати хмарне сховище для медіа та розгорнути окрему адміністративну вебпанель [14; 39].

У постановці задачі також потрібно врахувати припущення, в межах яких працює прототип. Передбачається, що кількість адміністраторів є обмеженою, а їхні Telegram ID відомі заздалегідь. Система орієнтована на локальний новинний канал, тому не потребує складної багаторівневої редакційної ієрархії у першій версії. Медіафайли зберігаються локально, що спрощує реалізацію, але вимагає резервного копіювання директорії media.

Обмеження першої версії не знижують практичної цінності системи, оскільки основний редакційний цикл уже реалізується повністю. Водночас вони формують підстави для майбутнього розвитку: додавання ролей, категорій, пошуку, планування публікацій, webhook-режиму, контейнеризації та зовнішнього сховища медіа. Таким чином постановка задачі охоплює не тільки поточну реалізацію, а й напрям подальшого масштабування.

Критеріями успішного виконання задачі є можливість створити заявку з текстом і медіа, переглянути її адміністратором, змінити статус, опублікувати в Telegram-канал, створити вебпост, отримати сторінку новини та зафіксувати дії у журналі. Додатковим критерієм є зрозумілість структури коду: нові обробники,

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						26
Змн.	Арк.	№ докум.	Підпис	Дата		

репозиторії або сервіси повинні додаватися без порушення вже реалізованих сценаріїв.

Задача проєктування також передбачає визначення структури даних. Потрібно зберігати не лише текст новини, а й автора, статус, адміністративний коментар, пов'язані медіафайли, канали публікації, `message_id` після публікації та вебверсію матеріалу. Якщо всі ці дані зберігати в одній таблиці, структура швидко стане незручною. Тому в роботі використано реляційну модель, де різні сутності винесено в окремі таблиці, а зв'язки між ними описано через SQLAlchemy-моделі.

У результаті постановка задачі зводиться до побудови системи, яка поєднує простий інтерфейс для користувача і достатньо керований адміністративний цикл. Користувач повинен швидко подати матеріал, адміністратор - отримати всі потрібні інструменти для перевірки, а система - забезпечити збереження, публікацію, аудит і подальше відображення новини. Саме ці вимоги визначили архітектуру, модель даних і набір технологій, описані в наступному розділі.

2.4. Сценарії використання системи та критерії приймання результату

Для деталізації постановки задачі доцільно описати систему через основні сценарії використання. Такий підхід дозволяє пов'язати загальні вимоги з конкретними діями користувачів і перевірити, чи справді реалізована система покриває повний редакційний процес. У межах даного проєкту ключовими є сценарії подання новини, адміністративної модерації, публікації матеріалу та обробки помилкових ситуацій [3; 4].

Перший сценарій стосується подання новини користувачем. Користувач запускає Telegram-бота, надсилає текст, фото, відео або медіаальбом, після чого система формує заявку та переводить її у стан очікування підтвердження. Якщо матеріал містить кілька фото або відео, вони повинні бути збережені як одна заявка з пов'язаними медіафайлами. Очікуваним результатом є створена, але ще не опублікована заявка [14; 15].

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						27
Змн.	Арк.	№ докум.	Підпис	Дата		

Другий сценарій описує адміністративну модурацію. Адміністратор відкриває чергу заявок, переглядає текст і медіафайли, за потреби редагує матеріал, після чого приймає рішення про схвалення або відхилення. У разі відхилення система повинна зберегти причину, змінити статус заявки та не показувати матеріал серед активного контенту. Критерієм успішності є коректна зміна статусу та наявність зрозумілого результату для адміністратора й автора [10; 11].

Окремо у сценарії модурації враховується потреба адміністратора швидко орієнтуватися в черзі заявок. Якщо матеріалів багато, система повинна показувати їх у впорядкованому вигляді, з короткою інформацією про автора, дату створення, текст і наявність медіафайлів. Це зменшує час пошуку потрібної заявки та дозволяє адміністратору приймати рішення без ручного перегляду приватних повідомлень.

Для Telegram-публікації система зберігає `message_id`, а для вебпублікації створює вебпост із заголовком, коротким описом, `slug` і зв'язком з оригінальною заявкою. Критерієм приймання є наявність матеріалу в обраному каналі, коректний статус у базі даних і доступність вебсторінки схваленої новини [14; 21; 24].

Четвертий сценарій охоплює помилкові та обмежувальні ситуації. Якщо Telegram API повертає помилку під час публікації, статус новини не повинен безпідставно змінюватися на опублікований. Якщо користувач без адміністративних прав натискає службову кнопку, дія не повинна виконуватися. Якщо текст містить HTML або спеціальні символи, вебчастина повинна відобразити його без виконання небажаного коду. Такі перевірки показують поведінку системи не лише в успішних, а й у небезпечних або некоректних ситуаціях [10; 11; 14].

Таким чином, сценарії використання уточнюють постановку задачі та перетворюють загальні вимоги на перевірювані умови. Вони формують основу для тестування, оскільки кожен сценарій можна перевірити окремо: від створення заявки до модурації, публікації або відхилення. Саме такі критерії дозволяють

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						28
Змн.	Арк.	№ докум.	Підпис	Дата		

оцінити відповідність системи практичним потребам управління новинним контентом [3; 4; 10].

2.5. Висновки по розділу

У другому розділі виконано аналіз вимог до програмної системи управління новинним контентом. Визначено основні ролі системи: користувач Telegram, адміністратор і читач вебсайту. Для кожної ролі описано очікувані сценарії взаємодії та функції, які система повинна забезпечувати.

Сформульовано функціональні вимоги до подання новини, додавання медіафайлів, обробки медіаальбомів, модерації, редагування, відхилення, публікації в Telegram-канал, створення вебпоста і перегляду матеріалів на сайті.

Також визначено нефункціональні вимоги до надійності, безпеки, продуктивності, підтримуваності та зручності використання.

Окремо розглянуто сценарії використання системи, які уточнюють послідовність роботи з новинним матеріалом. Описано подання заявки користувачем, адміністративну модерацію, публікацію схваленої новини та поведінку системи в помилкових або обмежувальних ситуаціях. Це дозволило перетворити загальні вимоги на практичні критерії, які можна перевірити під час тестування.

Важливим результатом розділу є визначення меж системи. Telegram-бот відповідає за взаємодію з користувачами й адміністраторами, база даних - за збереження заявок, статусів, медіафайлів і журналу дій, а вебчастина - за відображення тільки схвалених матеріалів. Такий поділ відповідальностей зменшує складність реалізації та полегшує подальший супровід проєкту.

Під час формування вимог окрему увагу приділено збереженню цілісності даних. Новина, автор, медіафайли, статус модерації, запис публікації та журнал дій повинні бути логічно пов'язані між собою. Це дає змогу уникнути ситуацій, коли матеріал опубліковано без відповідного запису в базі даних або коли медіафайли існують окремо від новинної заявки.

					БР.ІІ - 25.00.00.000 ПЗ	Арк.
						29
Змн.	Арк.	№ докум.	Підпис	Дата		

Також визначено вимоги до адміністративного контролю. Адміністратор повинен мати можливість швидко переглядати чергу заявок, відкривати деталі матеріалу, редагувати текст, відхиляти новину із зазначенням причини та обирати напрям публікації. Такі вимоги є важливими, оскільки саме адміністративний сценарій забезпечує якість і безпечність контенту перед його публічним поширенням.

Окремо враховано вимогу до зручності користувацького сценарію. Користувач не повинен виконувати складні дії або вводити технічні команди для подання новини. Основна взаємодія має відбуватися через кнопки Telegram-бота, а процес створення заявки повинен залишатися коротким і зрозумілим навіть для користувача без спеціальної підготовки.

Сформовані вимоги також дозволили визначити пріоритети першої версії системи. До основних функцій віднесено приймання новин, роботу з медіафайлами, модерацію, публікацію та вебвідображення схваленого контенту. Додаткові можливості, такі як розширена вебадмінпанель, рольова модель, категорії, пошук і планування публікацій, визначено як перспективні напрями подальшого розвитку.

Також у розділі уточнено обмеження першої версії системи. До них належать використання polling-режиму Telegram-бота, локальне зберігання медіафайлів і відсутність повноцінної вебадміністративної панелі розвитку системи після завершення базової реалізації.

У результаті постановка задачі проєктування зведена до створення модульної системи, яка поєднує простий інтерфейс подання матеріалів із контрольованим адміністративним процесом.

					БР.ІІ - 25.00.00.000 ПЗ	Арк.
						30
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ПРОЄКТУВАННЯ СИСТЕМИ УПРАВЛІННЯ КОНТЕНТОМ

3.1. Розробка архітектури програмного забезпечення

Архітектура системи побудована як модульний асинхронний застосунок мовою Python. Основний процес Telegram-бота запускається через модуль `bot.__main__`, де створюються Bot, Dispatcher, сховище станів, middleware та підключаються маршрутизатори. Окремо реалізовано вебзастосунок на aiohttp, який може запускатися як незалежний процес і працювати з тією самою базою даних. Такий підхід дозволяє розділити інтерактивну частину бота і публічний вебінтерфейс, але не дублювати бізнес-сутності.

На етапі проєктування можна було розглянути кілька архітектурних варіантів. Перший варіант - монолітний скрипт Telegram-бота, у якому обробники, запити до бази даних, збереження медіа і публікація знаходяться в одному файлі. Такий підхід простий для початку, але швидко ускладнює підтримку. Другий варіант - окрема мікросервісна система, де бот, вебчастина, публікація і медіасховище працюють як незалежні сервіси. Для бакалаврського прототипу це надмірно складно.

Обраний варіант можна визначити як модульний моноліт з окремим вебпроцесом. Він дозволяє зберегти простоту розгортання, але водночас розділяє відповідальність між пакетами `handlers`, `services`, `db`, `web` і `middlewares`. Така архітектура відповідає масштабу задачі: система не потребує складної інфраструктури, але має достатню внутрішню структуру для супроводу і розширення.

Пакет `handlers` містить сценарії взаємодії з Telegram. Користувацькі обробники відповідають за подання новин, перегляд власних заявок, чат з адміністратором та загальні команди. Адміністративні обробники реалізують модерацію, публікацію, редагування матеріалів, розсилки, канали, статистику й відповіді користувачам. Пакет `keyboards` відповідає за inline-клавіатури, а пакет `states` описує FSM-стани для багатокрокових дій. Така організація робить логіку

					БР.ІІ - 25.00.00.000 ПЗ	Арк.
						31
Змн.	Арк.	№ докум.	Підпис	Дата		

навігації явною.

Шар доступу до даних реалізовано через SQLAlchemy-моделі та repository-класи [5; 17]. Моделі описують користувачів, канали, новинні заявки, медіафайли, опубліковані пости, вебпости, повідомлення та журнал аудиту. Репозиторії інкапсулюють типові запити: створення новини, додавання медіа, отримання черги модерації, оновлення статусу, пошук вебпоста за slug тощо. Це зменшує дублювання SQL-логіки в обробниках і полегшує зміну способу зберігання даних.

Сервісний шар містить операції, які не належать безпосередньо до Telegram-обробників. Наприклад, publisher.py відповідає за надсилання матеріалів у канал, site_publisher.py формує заголовок, excerpt, slug і запис вебпоста, media.py зберігає файли, а ai.py інкапсулює звернення до AI-моделі. Архітектура системи наведена на рисунку 3.1.

Архітектура програмної системи управління контентом

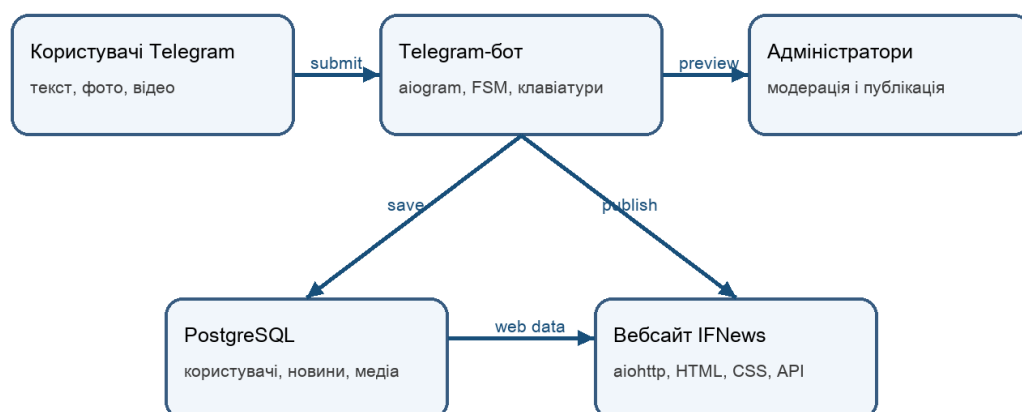


Рисунок 3.1 – Архітектура програмної системи управління контентом

Використання модульної архітектури важливе для підтримки системи. Якщо логіку Telegram-бота, запити до БД, роботу з медіафайлами та генерацію вебсторінок розмістити в одному файлі, код буде складно тестувати й розширювати. У розробленому проєкті кожен пакет має власну відповідальність.

Це відповідає підходам інженерії програмного забезпечення, де система розглядається як набір взаємопов'язаних, але відокремлених компонентів [3; 4].

Окрему роль відіграє middleware. Він дозволяє додавати спільну логіку для різних обробників, наприклад створення сесії бази даних, передачу конфігурації або обмеження частоти повідомлень. Завдяки цьому обробники не повинні вручну відкривати підключення до БД чи повторювати однакові перевірки. Такий підхід зменшує дублювання, а також робить код більш передбачуваним і зручним для супроводу.

3.2. Моделювання бізнес-процесів та системних компонентів

Головний бізнес-процес починається з користувача, який надсилає новину через Telegram-бота. Якщо повідомлення містить текст, бот переходить до стану очікування медіа; якщо користувач одразу надсилає фото або відео з підписом, система формує заявку з медіафайлом. Для альбомів застосовується буфер, який збирає всі елементи однієї media_group_id протягом короткого інтервалу. Після підтвердження заявка зберігається у базі даних зі статусом new, а адміністратори отримують повідомлення.

Адміністративний бізнес-процес починається з відкриття списку новин на модерації. Система рахує кількість pending-заявок і показує сторінку з обмеженою кількістю елементів. При відкритті конкретної заявки бот формує попередній перегляд: текст, автора, дату, кількість медіа та кнопки дій. Якщо матеріал містить кілька фото або відео, вони надсилаються як media group, після чого додається окреме повідомлення з керуючими кнопками. Це відповідає обмеженням Telegram щодо caption і кнопок для альбомів.

Процес публікації має транзакційну логіку на рівні застосунку. Спочатку система надсилає матеріал у Telegram-канал або створює вебпост, залежно від вибраної дії. Після успішної операції оновлюється статус новини, створюється запис про опублікований пост, зберігаються ідентифікатори повідомлень і створюється запис аудиту. Якщо публікація не вдалася, адміністратор отримує

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						33
Змн.	Арк.	№ докум.	Підпис	Дата		

повідомлення про помилку, а статус новини не змінюється на схвалений.

Системні компоненти взаємодіють через чіткі контракти. Обробник отримує Telegram-подію і сесію бази даних через middleware; repository повертає ORM-об'єкти; сервіс публікації приймає текст, канал і медіа; вебсервіс читає тільки опубліковані записи. На рисунку 3.2 подано послідовність проходження новини через систему.

Сценарій проходження новини через систему



Рисунок 3.2 – Сценарій проходження новини через систему

Описаний бізнес-процес дозволяє відокремити приймання матеріалу від публікації. Це важливо, тому що користувач не повинен напряму впливати на публічний канал. Його дія створює лише заявку, а остаточне рішення приймає адміністратор. Такий підхід зменшує ризик випадкового поширення неперевіреної інформації та відповідає редакційній логіці, де між джерелом повідомлення і публікацією існує етап перевірки.

Важливою особливістю цього бізнес-процесу є поетапність обробки матеріалу. Кожна дія користувача або адміністратора не одразу змінює публічний стан системи, а проходить через проміжні етапи: створення заявки, попередній перегляд, модерацію, вибір способу публікації та фіксацію результату. Така послідовність робить роботу системи більш контрольованою і прозорою. Адміністратор бачить, на якому етапі перебуває кожна новина, а збереження статусів, повідомлень і записів аудиту дозволяє відстежити історію обробки матеріалу у разі помилки або спірної ситуації.

Крім основного сценарію, система повинна обробляти допоміжні випадки: відсутність медіа, наявність кількох файлів, відмову від публікації, помилку

Telegram API, недоступність AI-сервісу або відсутність вебпублікації. Для цього у проєкті передбачено статуси, службові повідомлення, обробку винятків і журналювання дій адміністратора. Завдяки цьому система не зводиться до одного ідеального сценарію, а враховує типові ситуації, які можуть виникнути під час реального використання.

Саме тому у проєкті важливо розділяти стани користувацького діалогу і статуси новини в базі даних. FSM-стани описують тимчасовий процес спілкування з ботом, а статуси news_suggestions фіксують довготривалий стан заявки. Таке розділення робить систему стійкішою до перерваних сценаріїв і дозволяє адміністратору бачити реальний стан матеріалу після кожної дії.

3.3. Проєктування бази даних та обґрунтування технологій

База даних системи містить набір таблиць, які відображають основні сутності редакційного процесу: користувачів, новинні заявки, медіафайли, канали публікації, опубліковані матеріали, вебпости та журнал адміністративних дій. Центральною таблицею є news_suggestions, оскільки саме вона зберігає новинну заявку, її автора, статус модерації та службові дані. Пов'язані з нею медіафайли винесено в окрему таблицю news_media, що дозволяє зберігати кілька фото або відео для однієї новини.

Логічна модель даних наведена на рисунку 3.3, а основні сутності бази даних, їх призначення та ключові поля узагальнено в таблиці 3.1. Вона демонструє, що система не зберігає весь матеріал в одній таблиці, а розділяє сутності відповідно до їхньої ролі. Це зменшує дублювання, спрощує запити і дозволяє розширювати систему.

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						35
Змн.	Арк.	№ докум.	Підпис	Дата		

Логічна модель даних

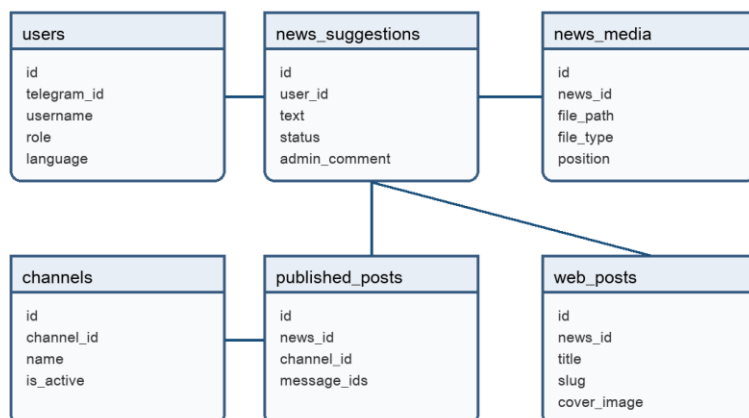


Рисунок 3.3 – Логічна модель бази даних

Таблиця 3.1

Основні сутності бази даних

Сутність	Призначення	Ключові поля
User	Зберігання користувачів Telegram та ролей	telegram_id, username, role, language
NewsSuggestion	Новинні заявки користувачів	user_id, text, status, admin_comment
NewsMedia	Медіафайли, пов'язані з новиною	news_id, file_path, file_type, position
Channel	Telegram-канали для публікації	channel_id, name, is_active
PublishedPost	Факт публікації в Telegram	news_id, channel_id, message_ids
WebPost	Матеріал для вебсайту	title, slug, excerpt, cover_image
AuditLog	Журнал адміністративних дій	admin_id, action, details

Окрему увагу під час проєктування бази даних приділено цілісності інформації. Зв'язки між таблицями дозволяють уникнути ситуацій, коли медіафайл, запис про публікацію або запис журналу існує без відповідної новинної заявки чи користувача. Така структура робить схему зрозумілою для підтримки та зменшує ризик дублювання даних [17; 31].

Для подальшої роботи системи важливими є не лише самі таблиці, а й поля, за якими найчастіше виконуються запити. До таких полів належать `status` і `created_at` у `news_suggestions`, `telegram_id` у `users`, `slug` та `is_published` у `web_posts`. У прототипі кількість записів є невеликою, однак при накопиченні новин індексація цих полів може суттєво пришвидшити чергу модерації, пошук користувачів і формування вебсторінок.

Також варто врахувати життєвий цикл медіафайлів. Запис у `news_media` повинен бути пов'язаний із реально збереженим файлом або `Telegram file_id`. Якщо файл видалено з диска, але запис залишився в базі даних, вебпублікація може втратити зображення. Тому резервне копіювання БД і директорії `media` має виконуватися узгоджено. Це поєднує проєктування бази даних із подальшою експлуатацією системи.

Python обрано як основну мову реалізації завдяки підтримці асинхронного програмування та наявності бібліотек для Telegram-ботів, баз даних і вебсервісів [16]. Крім того, для цієї мови існує велика кількість готових рішень, документації та прикладів, що зменшує час розробки та полегшує пошук помилок.

SQLAlchemy у версії 2.x використовується як ORM і query builder [17; 30]. Його перевагою є можливість описати модель предметної області в коді, застосувати relationship-зв'язки і використовувати selectinload для контрольованого завантаження пов'язаних сутностей. asyncpg обрано як асинхронний драйвер PostgreSQL, оскільки він забезпечує ефективну роботу з базою даних у асинхронному середовищі [20]. Alembic застосовується для міграцій, що дозволяє фіксувати еволюцію схеми БД у вигляді версійованих файлів і безпечно змінювати структуру бази під час розвитку проєкту [18; 31].

Вебчастина реалізована на aiohttp [21; 32]. Для цього проєкту немає необхідності у важкому frontend-фреймворку, оскільки сайт виконує роль публічної вітрини з картками новин, сторінками деталізації та API. Серверний HTML-рендеринг дозволяє швидко віддавати сторінки, а CSS-файл відповідає за зовнішній вигляд [24; 25]. Для безпеки всі користувацькі тексти екрануються

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						37
Змн.	Арк.	№ докум.	Підпис	Дата		

через `html.escape`, що знижує ризик відображення небезпечного HTML-коду на сторінках сайту [34].

Конфігурація системи реалізована через `pydantic-settings` [22; 33]. Це дозволяє описати параметри типізовано: `bot_token`, `admin_ids`, `database_url`, `gemini_api_key`, `media_dir`, `site_name`, `site_host`, `site_port` і `site_public_url`. Такий спосіб зручний для локального запуску і подальшого розгортання, оскільки секрети не потрібно жорстко записувати в код. Замість цього вони можуть зберігатися у файлі середовища або передаватися через змінні оточення, що підвищує безпеку та гнучкість налаштування системи [13].

3.4. Висновки по розділу

У третьому розділі спроектовано архітектуру програмної системи управління контентом. Обґрунтовано вибір модульної структури, у якій Telegram-обробники, сервісний шар, `repository`-рівень, база даних, `middleware` і вебчастина мають окремі зони відповідальності. Такий підхід дозволяє уникнути надмірної складності мікросервісної архітектури, але зберегти підтримуваність коду.

Описано бізнес-процес проходження новини через систему: подання матеріалу користувачем, збереження заявки, модерація адміністратором, публікація в Telegram-канал або на вебсайт і фіксація результату в журналі аудиту. Розроблено логічну модель бази даних, у якій користувачі, заявки, медіафайли, канали, публікації, вебпости і журнал адміністративних дій винесені в окремі сутності.

Також обґрунтовано вибір технологій реалізації: Python, `aiogram`, `SQLAlchemy`, `asyncpg`, `Alembic`, `PostgreSQL`, `aiohttp` і `pydantic-settings`. Результати проєктування стали основою для практичної реалізації програмного продукту, описаної у четвертому розділі.

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						38
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ПРОЄКТУ УПРАВЛІННЯ КОНТЕНТОМ

4.1. Структура програмного коду

Репозиторій проєкту має модульну структуру. Кореневий пакет bot містить усі основні частини застосунку. Файл bot/__main__.py є точкою входу Telegram-бота: він створює об'єкт Bot, Dispatcher, підключає middleware і реєструє маршрутизатори. Пакет handlers поділено на admin та user, що відповідає ролям системи. Пакети keyboards, states, filters, middlewares, services, db, i18n і web мають окремі зони відповідальності, що узагальнено в таблиці 4.1.

Користувацький сценарій подання новини реалізовано у файлі bot/handlers/user/submit_news.py. У ньому визначено допоміжні функції для отримання file_id фото й відео, буферизації альбомів, додавання медіа до заявки та фінального підтвердження. Ключова особливість полягає у роботі з media_group_id: бот очікує короткий проміжок часу, збирає елементи альбому й лише після цього формує повідомлення підтвердження. Це запобігає створенню кількох заявок з одного альбому. Головне меню Telegram-бота, з якого користувач починає взаємодію із системою, наведено на рисунку 4.1.

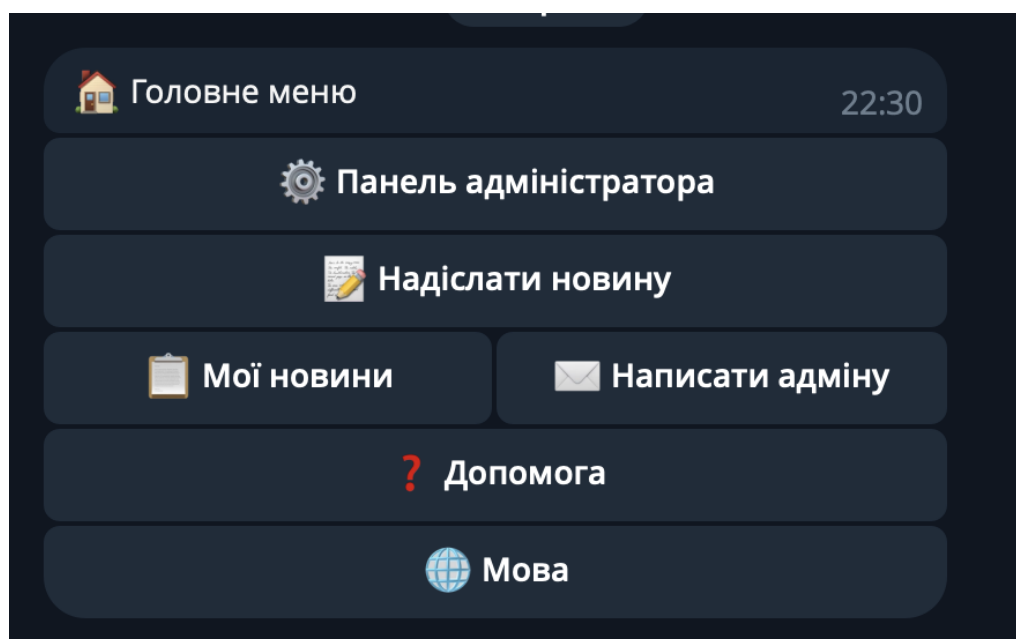


Рисунок 4.1 – Головне меню Telegram-бота

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						39
Змн.	Арк.	№ докум.	Підпис	Дата		

Адміністративну модуерацію реалізовано у файлі bot/handlers/admin/moderate.py. Тут є пагінація заявок, повний перегляд новини, очищення попередніх повідомлень, відхилення з причиною та редагування тексту (рис. 4.2-4.3).

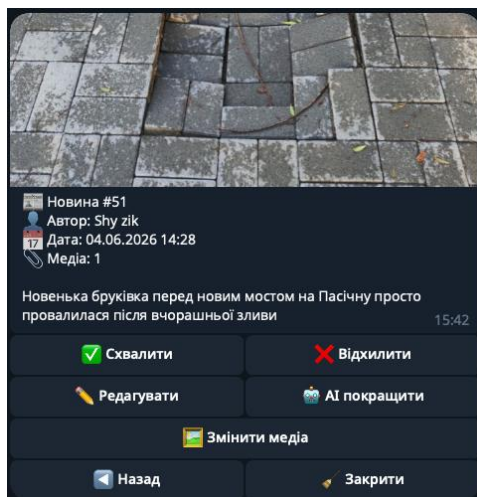


Рисунок 4.2 – Детальний перегляд новини адміністратором

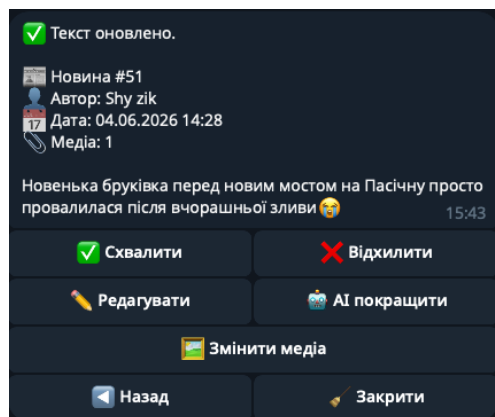


Рисунок 4.3 – Редагування тексту новини адміністратором

Публікація реалізована у bot/handlers/admin/publish.py, де адміністратор обирає канал і може опублікувати матеріал у Telegram, на сайт або в обидва напрями. Успішна публікація супроводжується оновленням статусу, створенням запису про публікацію та журналом аудиту.

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						40
Змн.	Арк.	№ докум.	Підпис	Дата		

Вебчастина знаходиться у пакеті bot/web. Файл app.py створює aiohttp-застосунок, маршрути для головної сторінки, деталізації новини, API, healthcheck, статичних файлів і медіа. HTML формується серверно з екрануванням змінних [34]. Це просте рішення добре підходить для прототипу, оскільки не потребує окремого build-процесу та зберігає весь код у межах одного Python-проєкту.

Окрему роль у структурі проєкту відіграє конфігураційний шар. Параметри запуску, такі як токен бота, список адміністраторів, адреса бази даних, шлях до медіафайлів і налаштування вебсервера, винесені з основної логіки застосунку. Завдяки цьому один і той самий код можна використовувати в різних середовищах: локально під час розробки, на тестовому сервері або під час реального розгортання.

Такий підхід також спрощує супровід, оскільки зміна параметрів не потребує редагування бізнес-логіки. Важливо, що модулі проєкту не дублюють відповідальність один одного. Обробники Telegram-подій відповідають за приймання команд і повідомлень, репозиторії працюють із базою даних, а сервіси виконують прикладну логіку, наприклад публікацію матеріалів або підготовку вебпостів. Це робить код більш передбачуваним: під час пошуку помилки можна швидше визначити, у якому шарі вона виникла. Крім того, така структура полегшує подальше розширення системи, наприклад додавання нових адміністративних дій, нових типів публікацій або окремих API-методів.

Таблиця 4.1

Структура основних пакетів проєкту

Пакет	Призначення
bot/handlers	Обробники користувацьких та адміністративних Telegram-сценаріїв
bot/db	Моделі SQLAlchemy, engine та repository-класи
bot/services	Сервіси публікації, медіа, вебпостів та AI-функцій
bot/web	aiohttp-застосунок для публічного сайту IFNews
bot/middlewares	Сесії БД, інтернаціоналізація та обмеження частоти повідомлень

Розділення коду на пакети має практичне значення. Якщо необхідно змінити

текст кнопок або структуру меню, зміни вносяться до пакета клавіатур. Якщо потрібно змінити запити до БД, коригуються репозиторії. Якщо потрібно змінити спосіб публікації, редагується сервісний шар. Такий підхід зменшує ризик випадково пошкодити інші сценарії й робить проєкт зрозумілішим для супроводу.

4.2. Реалізація основних алгоритмів та структур даних

Першим важливим алгоритмом є збирання медіаальбому. Telegram надсилає елементи альбому як окремі оновлення з однаковим `media_group_id`. Якщо обробляти кожне оновлення негайно, система може створити кілька неповних заявок. У проєкті використано словники `_album_buffers` і `_album_tasks`, ключем яких є комбінація `chat_id`, `user_id` та `media_group_id`. При надходженні елемента він додається до буфера, попередня задача фіналізації скасовується, а нова запускається із затримкою. Коли нові елементи більше не надходять, система формує одну заявку.

Алгоритм обробки медіаальбому є критичним, оскільки Telegram не надсилає альбом як одну готову сутність. Кожне фото або відео надходить окремим оновленням, але має спільний `media_group_id`. Якщо система одразу створюватиме заявку після першого елемента, інші файли будуть відокремлені від основної новини. Тому застосування короткої затримки є виправданим компромісом між швидкістю відповіді і повнотою збереження даних.

Другим алгоритмом є формування `slug` для вебпоста. Сервіс `site_publisher.py` очищує заголовок, транслітерує українські літери в латиницю, замінює неалфавітно-цифрові символи дефісами, обрізає довжину та додає ідентифікатор новини. Додавання `id` у кінець `slug` зменшує ризик колізій, навіть якщо кілька новин мають схожі заголовки. Цей алгоритм простий, але практичний для локального новинного сайту (рис 4.4).

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						42
Змн.	Арк.	№ докум.	Підпис	Дата		

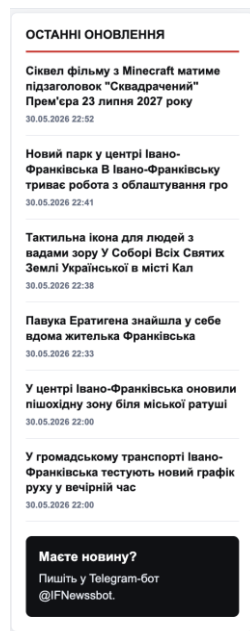


Рисунок 4.4 – Останні новини та спосіб подання новини

Третім важливим механізмом є middleware для бази даних. DbSessionMiddleware створює асинхронну сесію для кожної Telegram-події, відкриває транзакцію, передає session в data і після виконання handler виконує commit. Завдяки цьому обробники не створюють підключення вручну, а вся логіка роботи з БД відбувається в єдиному стилі. Такий підхід також зменшує ризик забути закрити сесію або підтвердити транзакцію.

Четвертим механізмом є обмеження частоти повідомлень. ThrottleMiddleware зберігає час останнього текстового повідомлення користувача у словнику і блокує надто часті повторні звернення. При цьому для фото, відео та медіагруп обмеження не застосовується, щоб не зламати сценарій надсилання альбому. Це приклад практичного компромісу між захистом від спаму і коректною підтримкою мультимедійного сценарію.

Окремо слід зазначити роботу зі статусами. Статус new означає, що заявка очікує модерації. Статус approved вказує, що матеріал схвалено або опубліковано. Статус rejected використовується, коли адміністратор відхилив матеріал і залишив пояснення. Таке представлення спрощує фільтрацію черги, формування статистики та відображення історії для користувача. Статуси також допомагають уникнути ситуації, коли один і той самий матеріал випадково публікується кілька

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		43

разів.

Важливою структурою є зв'язок між новиною і медіафайлами. Текстова заявка зберігається окремо від файлів, а кожен файл має власний запис у таблиці news_media. Завдяки цьому система може працювати як з текстовими новинами, так і з матеріалами, що містять кілька фото або відео. Позиція файлу зберігається окремо, щоб під час публікації або перегляду порядок матеріалів відповідав тому, як їх надіслав користувач.

4.3. Реалізація інтерфейсів користувача та адміністратора

Користувацький інтерфейс бота побудовано на inline-клавіатурах і коротких повідомленнях. Основне меню містить дії для подання новини, перегляду власних матеріалів і зв'язку з адміністратором. Під час подання новини бот послідовно веде користувача через стани, але не вимагає складних команд. Якщо користувач надсилає фото або відео з підписом, система одразу розуміє це як потенційну новину, що робить сценарій природним.

Адміністративний інтерфейс також реалізовано в Telegram. Панель адміністратора містить доступ до черги модерації, каналів, повідомлень, статистики та інших службових дій. Кнопки в повідомленнях формують callback_data з ідентифікаторами новин або каналів, що дозволяє швидко виконувати операції без введення текстових команд. Для безпеки адміністративні маршрути захищені фільтром IsAdmin, який перевіряє Telegram ID користувача.

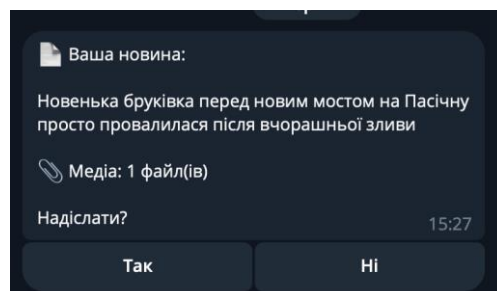


Рисунок 4.5 – Подання новини користувачем

Особливістю реалізації є очищення службових повідомлень. Під час

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						44
Змн.	Арк.	№ докум.	Підпис	Дата		

модерації бот може надсилати медіагрупу, текстовий опис і панель кнопок. Якщо адміністратор переходить до іншої заявки, попередні повідомлення потрібно прибрати, щоб чат залишався читабельним. Для цього у стані зберігаються `preview_msg_ids`, `prompt_msg_id` і `panel_msg_id`, а функція `_cleanup_preview` послідовно видаляє відстежені повідомлення.

Вебінтерфейс має головну сторінку з виділеною останньою новиною, списком останніх оновлень, сіткою матеріалів і сторінкою деталізації. Якщо в новини є `cover_image`, вона показується як зображення; якщо немає, використовується текстовий `placeholder`. Це дозволяє сайту коректно працювати навіть для текстових новин. Приклади реалізованих користувацьких, адміністративних і вебінтерфейсів системи наведено на рисунках 4.5-4.8.

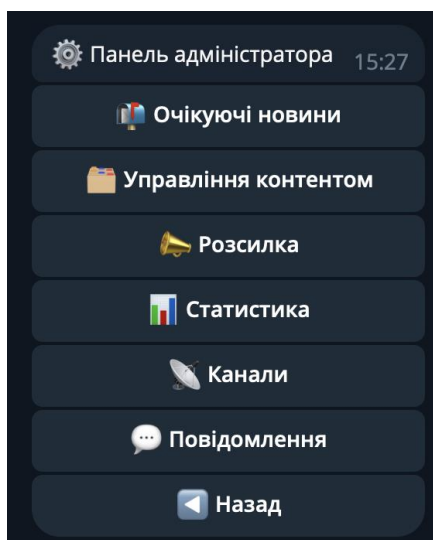


Рисунок 4.6 – Панель модерації адміністратора

Після перегляду адміністратор може відредагувати текст, відхилити матеріал із причиною або перейти до публікації. Під час редагування тексту важливо зберігати зміст новини, але виправляти граматичні помилки, прибрати зайві емоційні формулювання, уточнювати заголовок і робити матеріал придатним для публічного каналу (рис. 4.7).

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						45
Змн.	Арк.	№ докум.	Підпис	Дата		

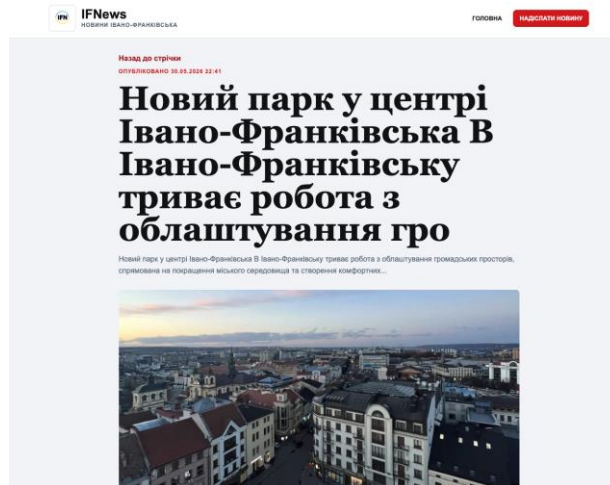


Рисунок 4.7 – Вебсторінка опублікованих новин

Після схвалення адміністратор обирає канал публікації. Система підтримує Telegram-публікацію, вебпублікацію або комбінований варіант. Якщо матеріал надсилається у Telegram-канал, у базі зберігаються message_id або список message_ids для медіаальбому (рис. 4.8). Якщо матеріал публікується на сайті, створюється запис web_posts із заголовком, slug, excerpt і cover_image. Таким чином одна адміністративна дія формує керований слід у БД.

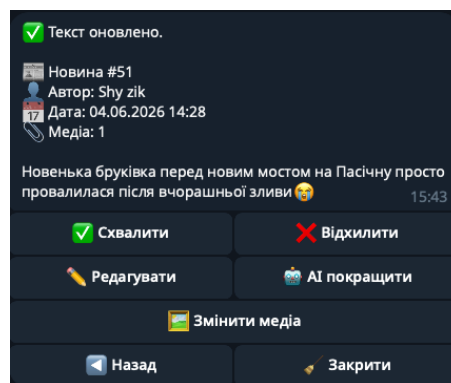


Рисунок 4.8 – Редагування тексту новини адміністратором

4.4. Реалізація публікації та AI-допоміжних функцій

Публікація у Telegram-канал виконується через сервісний метод, який приймає bot, channel_id, текст і список медіа. Якщо матеріал не містить медіа, надсилається звичайне повідомлення. Якщо є одне фото або відео, воно

					БР.ІІ - 25.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		46

надсилається з caption. Якщо медіа кілька, система формує media group, а ідентифікатори всіх повідомлень зберігаються у полі message_ids. Це важливо для подальшого аудиту або можливого редагування опублікованих матеріалів.

Публікація на сайт виконується після схвалення матеріалу. Сервіс publish_to_site перевіряє, чи існує вже вебпост для цієї новини, формує заголовок, короткий опис, slug, основний текст і cover_image. Якщо запис існує, він оновлюється; якщо ні, створюється новий. Такий підхід дозволяє повторно запускати публікацію без створення дублікатів у вебчастині.

AI-допоміжні функції винесено в окремий модуль bot/services/ai.py [23]. Він містить операції покращення тексту, скорочення тексту, генерації заголовка і відповіді в чаті. Всі ці операції звертаються до зовнішньої моделі через google-gemini, а ключ API береться з конфігурації. Винесення AI-логіки в окремий модуль є правильним архітектурним рішенням, оскільки модель, постачальник або prompt можуть змінюватися незалежно від решти системи.

У практичному редакційному сценарії AI не замінює адміністратора, а виконує допоміжну роль. Остаточне рішення про публікацію приймає людина, тому система не створює ризику автоматичного поширення некоректного тексту. Такий підхід відповідає принципу human-in-the-loop, коли генеративний інструмент підвищує продуктивність, але не прибирає контроль якості.

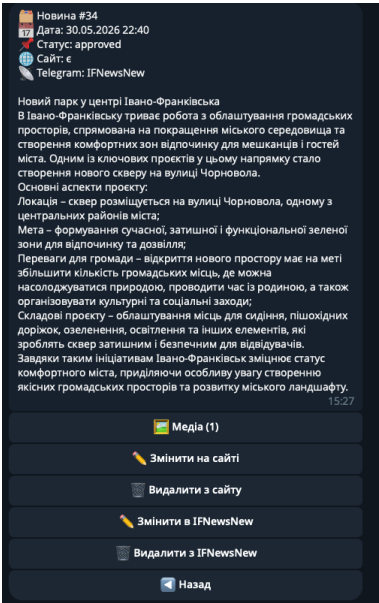


Рисунок 4.10 – Статистика та службові дії адміністратора

Під час інтеграції AI-функцій важливо врахувати можливість помилки зовнішнього сервісу або перевищення квоти. У такому випадку основний редакційний процес не повинен зупинятися: адміністратор має мати змогу опублікувати матеріал без автоматичного покращення тексту. Тому AI-сервіс у проєкті розглядається як допоміжний компонент, а не як обов'язкова умова публікації.

Такий підхід підвищує надійність системи. Якщо зміниться постачальник AI-моделі або формат відповіді API, достатньо адаптувати окремий модуль bot/services/ai.py. Обробники модерації та публікації залишаються незалежними від конкретної моделі. Це відповідає загальному принципу модульності, який застосовано в архітектурі всієї системи.

Життєвий цикл розроблення програмної системи управління контентом доцільно розглядати не як одноразове написання коду, а як послідовність взаємопов'язаних етапів SDLC: аналіз вимог, проєктування, реалізація, тестування, реліз, експлуатація та супровід [3; 4]. Для розробленого проєкту це особливо важливо, оскільки система працює з реальними користувацькими повідомленнями, медіафайлами зображено на рисунку 4.9, Telegram API, базою даних і публічним вебсайтом. Помилка на одному етапі може призвести не лише до технічного збою, а й до некоректної публікації контенту.

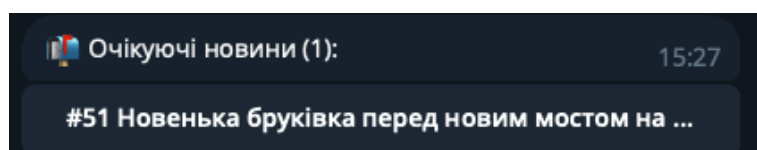


Рисунок 4.10 – Список новин на модерації

Release-процес для цього проєкту повинен містити підготовку змін, перевірку міграцій БД, запуск автоматичних або ручних тестів, оновлення конфігурації, перезапуск сервісів і післярелізний контроль логів [7]. Окремо потрібно враховувати, що Telegram-бот працює у режимі polling, тому перед запуском нової версії попередній процес має бути коректно зупинений. Для вебчастини необхідно перевірити порт, доступність healthcheck та коректність

статичних файлів і медіамаршрутів.

Версіонування релізів доцільно вести за схемою Semantic Versioning у форматі MAJOR.MINOR.PATCH [12]. Розгортання системи може виконуватися вручну або через CI/CD-пайплайн. У ручному варіанті адміністратор оновлює код, встановлює залежності, застосовує Alembic-міграції, перевіряє доступність PostgreSQL, запускає вебсервіс і бота. У CI/CD-варіанті ці дії можна автоматизувати через GitHub Actions або інший інструмент, додавши перевірки форматування, компіляції, тестів і збірки артефакту [37]. Для production-середовища важливо зберігати секрети в змінних середовища, а не в репозиторії [13]. Застосування етапів SDLC у розробленій системі наведено в таблиці 4.2.

Таблиця 4.2

Застосування SDLC у розробленій системі

Етап SDLC	Зміст робіт у проєкті	Результат
Аналіз	Визначення ролей користувача, адміністратора і читача сайту	Сформовані вимоги та сценарії
Проектування	Архітектура бота, БД, сервісів і вебчастини	Модель компонентів і даних
Реалізація	Розробка handlers, repositories, services, web app	Працездатний прототип
Тестування	Перевірка подання, модерації, публікації та API	Підтверджена працездатність
Release	Міграції, запуск сервісів, smoke test, healthcheck	Готова версія для використання
Супровід	Логи, резервні копії, контроль медіа, виправлення помилок	Стабільна експлуатація

4.5. Обробка помилок, журналювання та відновлення операцій

Надійність системи управління новинним контентом залежить не лише від успішного виконання основних сценаріїв, а й від передбачуваної поведінки під час збоїв. Помилки можуть виникати при зверненні до Telegram Bot API, роботі з PostgreSQL, завантаженні медіафайлів, формуванні вебпоста або неправильній послідовності дій користувача.

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						49
Змн.	Арк.	№ докум.	Підпис	Дата		

Критичною є операція публікації новини. Статус новини не повинен змінюватися на опублікований до того, як Telegram API поверне підтвердження ідентифікатора повідомлення. Якщо зовнішній сервіс недоступний або відхиляє запит, заявка залишається у контрольованому стані, а адміністратор може повторити дію після усунення причини. Такий порядок запобігає ситуації, коли база даних містить інформацію про успішну публікацію, але повідомлення фактично відсутнє у каналі.

Сервіс `publish_to_site` спочатку формує заголовок, короткий опис, `slug`, текст і шлях до обкладинки, після чого створює або оновлює запис `web_posts`. Зміни мають виконуватися в межах керованої сесії `SQLAlchemy`. У разі помилки транзакцію потрібно відкотити, щоб частково створений вебпост не відображався читачам. Повторне виконання операції не повинно породжувати дублікати, тому перед створенням запису перевіряється наявний вебпост, пов'язаний із `news_id` [17; 19; 30].

Основні помилкові ситуації та передбачену реакцію системи узагальнено в таблиці 4.3.

Таблиця 4.3

Основні помилкові ситуації та реакція системи

Ситуація	Можливий наслідок	Реакція системи
Помилка Telegram API	Матеріал не з'явився у каналі	Не змінювати статус; повідомити адміністратора; дозволити повтор
Помилка транзакції БД	Частково збережені дані	Відкотити транзакцію і записати помилку
Збій завантаження медіа	Відсутній файл або неповний альбом	Зберегти текст заявки; повторити завантаження без дублікатів
Некоректні права доступу	Виконання адміністративної дії користувачем	Заблокувати дію через <code>IsAdmin</code> і зафіксувати спробу
Помилка HTML-рендерингу	Некоректне відображення сторінки	Екранувати текст і повернути контрольовану відповідь

Повторне виконання операцій потребує ідемпотентності: перед повторною публікацією система перевіряє `published_posts`, `web_posts` і поточний статус

заявки. Відновлення після збою спирається на резервну копію БД, контрольовані Alembic-міграції та копію медіакаталогу. Після перезапуску підтверджені матеріали залишаються доступними, оскільки їх основні дані зберігаються в PostgreSQL.

Приклад контрольованого повідомлення, яке користувач отримує в разі недоступності AI-сервісу, наведено на рисунку 4.11.

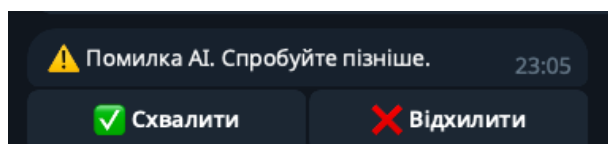


Рисунок 4.11 - Повідомлення про помилку AI-сервісу

Практична перевірка охоплює недоступний зовнішній сервіс, некоректні права, збій БД і повторне натискання кнопки. Для кожного сценарію перевіряються повідомлення, статус новини та записи журналу [10; 11; 34].

Таким чином, обробка помилок у розробленій системі базується на відкладеній зміні статусу, транзакційному збереженні, перевірці прав, розділенні технічного журналу й аудиту та можливості безпечного повторення операцій. Такий підхід не усуває зовнішні збої, але не дозволяє їм непомітно пошкодити стан новинного матеріалу.

4.6. Висновки по розділу

У четвертому розділі описано практичну реалізацію програмної системи управління контентом. Розглянуто структуру репозиторію, основні пакети проєкту, точку входу Telegram-бота, користувацькі й адміністративні обробники, сервісний шар, repository-рівень, middleware та вебчастину.

Окрему увагу приділено алгоритмам, які мають найбільше практичне значення: буферизації медіаальбомів, формуванню slug для вебпостів, створенню асинхронної сесії БД через middleware, обмеженню частоти повідомлень і роботі

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						51
Змн.	Арк.	№ докум.	Підпис	Дата		

зі статусами новинних заявок. Також описано реалізацію інтерфейсів користувача, адміністратора і вебсайту.

Результатом реалізації є працездатний прототип, який забезпечує повний цикл роботи з новиною: приймання матеріалу, збереження медіа, модерацію, редагування, відхилення, публікацію в Telegram-канал, створення вебпоста, журналювання дій і базові release-перевірки.

Важливою перевагою реалізації є те, що основні частини системи не залежать жорстко одна від одної. Telegram-обробники відповідають за взаємодію з користувачем, сервіси виконують прикладну логіку, repository-рівень працює з базою даних, а вебчастина відображає тільки вже підготовлені та опубліковані матеріали. Завдяки такому поділу зміни в одному компоненті не потребують повного переписування всієї системи. Наприклад, можна змінити спосіб публікації на сайт або додати новий канал поширення без зміни логіки подання новини користувачем.

Також у процесі реалізації було враховано особливості роботи Telegram як платформи. Система підтримує не лише прості текстові повідомлення, а й фото, відео та медіагрупи, що є важливим для реального новинного сценарію. Для адміністратора реалізовано попередній перегляд матеріалу, керуючі кнопки, редагування тексту, відхилення із коментарем та вибір напрямку публікації. Це дозволяє виконувати основні редакційні дії без переходу до окремої складної адміністративної панелі.

Отже, реалізований програмний продукт відповідає поставленим вимогам і демонструє практичну придатність обраної архітектури. Система може використовуватися як основа для локального новинного сервісу, а її модульна структура створює можливості для подальшого розвитку.

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						52
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 5. ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ, СУПРОВІД ТА НАПРЯМИ РОЗВИТКУ ПРОЄКТУ

5.1. Стратегії, методи та результати тестування

Тестування системи виконано на кількох рівнях: перевірка окремих функцій, інтеграційне тестування сценаріїв Telegram-бота, перевірка роботи з базою даних, тестування вебмаршрутів і ручне приймальне тестування. Основні сценарії тестування системи наведено в таблиці 5.1. Для бакалаврського проєкту найважливішим є підтвердити, що ключові сценарії виконуються без втрати даних: створення заявки, збереження медіа, модерація, публікація і відображення на сайті [15].

Таблиця 5.1

Основні сценарії тестування системи

Сценарій	Очікуваний результат	Статус
Подання текстової новини	Створено запис news_suggestions зі статусом new	успішно
Подання фото з підписом	Збережено текст і один запис news_media	успішно
Подання медіаальбому	Елементи одного media_group_id об'єднано в одну заявку	успішно
Відхилення адміністратором	Статус rejected, автор отримав причину	успішно
Публікація в канал	Статус approved, збережено message_ids	успішно
Публікація на сайт	Створено web_posts і доступну сторінку /news/{slug}	успішно

Модульне тестування можна застосувати до чистих функцій, які не залежать від Telegram API. До таких функцій належать make_title, make_excerpt, make_slug, media_url, format_date, а також окремі методи репозиторіїв при використанні тестової бази. Для функцій генерації slug потрібно перевірити транслітерацію, обрізання довгих заголовків і додавання ідентифікатора новини.

Для media_url потрібно перевірити, що система не віддає шлях поза дозволеною директорією media.

Інтеграційне тестування Telegram-сценаріїв може виконуватися вручну або з використанням мок-об'єктів Bot і Message. Ручна перевірка є прийнятною для прототипу, оскільки Telegram API має зовнішню залежність і вимагає реального токена. Основні сценарії: надсилання текстової новини без медіа, надсилання фото з caption, надсилання альбому, підтвердження заявки, перегляд заявки адміністратором, редагування тексту, відхилення та публікація (рис. 5.1-5.3).

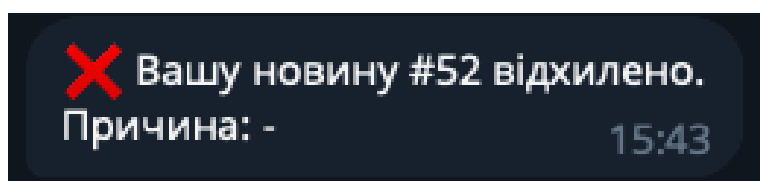


Рисунок 5.1 – Відхилення новини із зазначенням причини

Для більшої повноти тестування доцільно виділити позитивні та негативні сценарії. До позитивних належать успішне подання текстової новини, фото з підписом, медіаальбому, схвалення, публікація в канал і створення вебпоста. До негативних сценаріїв належать відхилення заявки, спроба доступу до адміністративної панелі без прав, помилка Telegram API, відсутність медіафайлу на диску та некоректний slug. Такий поділ показує, що система перевіряється не лише в ідеальному сценарії.

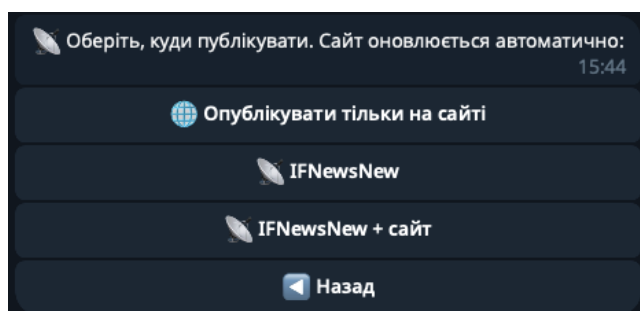


Рисунок 5.2 – Вибір каналу для публікації новини

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						54
Змн.	Арк.	№ докум.	Підпис	Дата		

Вебчастина тестується через HTTP-запити до маршрутів /, /news/{slug}, /api/posts і /health. Очікуваними результатами є код 200 для наявних сторінок, 404 для відсутньої новини, валідний JSON для API та status: ok для healthcheck. Також потрібно перевірити коректність екранування HTML, щоб користувацький текст не виконувався як код на сторінці [11; 34].

Для перевірки працездатності було сформовано набір тестових сценаріїв, що охоплює основні ролі системи та наведений у таблиці 5.1. Тестові дані включали короткі новини, довгі тексти, фото, відео, альбоми з кількох зображень, відхилені заявки та публікації на сайт. Очікуваним результатом для кожного сценарію була зміна відповідних записів у базі даних, коректні повідомлення користувачу й адміністратору та відсутність необроблених винятків у логах.

Для адміністративного сценарію перевіряються кнопки переходів, пагінація, очищення попередніх preview-повідомлень, редагування тексту та відхилення із причиною (рис. 5.1-5.2).

Після відхилення статус новини повинен змінитися на rejected, admin_comment повинен містити текст причини, а автор повинен отримати повідомлення. Після публікації статус змінюється на approved, створюється запис published_posts або web_posts, а в audit_log додається дія адміністратора (рис. 5.3-5.4).

5.2. Впровадження, release-перевірки та експлуатація системи

Ефективність системи оцінюється не лише технічно, а й організаційно. До впровадження подібного інструмента адміністратор може отримувати новини у приватні повідомлення, копіювати текст вручну, завантажувати медіа окремо та переносити матеріал у канал або сайт. У розробленій системі ці операції об'єднано в один керований цикл. Це зменшує кількість ручних дій, знижує ризик втрати файлів і дозволяє бачити статус кожної заявки.

З точки зору продуктивності, асинхронний стек дозволяє системі обробляти типове навантаження локального новинного каналу без складної

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						55
Змн.	Арк.	№ докум.	Підпис	Дата		

інфраструктури. Більшість операцій є I/O-bound, тобто пов'язані з очікуванням відповіді Telegram API, БД або файлової системи. Використання `async/await` дає змогу не блокувати весь застосунок під час цих очікувань [16; 28]. Для масштабування можна розділити бот і вебсайт на окремі процеси, винести медіа в об'єктне сховище та перейти на `webhook`-режим.

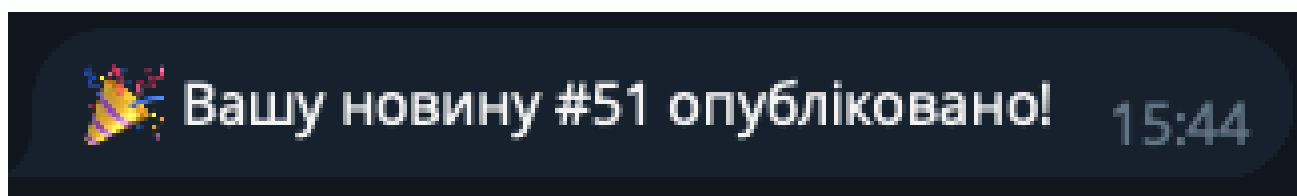


Рисунок 5.3 – Успішна публікація новини в Telegram-канал

Окрім функціонального тестування, для системи управління контентом важливими є `release`-перевірки, які виконуються безпосередньо перед запуском нової версії. Вони підтверджують, що зміни не порушують критичні сценарії: приймання новини, модерацію, публікацію, відображення на сайті та доступ до БД. Для невеликого проєкту `release`-чекліст може бути коротким, але він повинен виконуватися послідовно перед кожним оновленням [7].

Першим кроком є перевірка середовища: доступність PostgreSQL, коректність `.env`, наявність Telegram bot token, `admin_ids`, `database_url`, `media_dir` і `site_public_url`. Другим кроком є застосування Alembic-міграцій і перевірка, що схема БД відповідає ORM-моделям [18; 31]. Третім кроком є синтаксична перевірка коду, наприклад `python -m compileall bot`, а за наявності тестів - запуск `unit` та `integration tests`.

Після запуску сервісів перевіряється вебчастина. `Healthcheck /health` має повертати статус `ok`, головна сторінка повинна відкриватися без помилок, а `/api/posts` має повертати список лише опублікованих матеріалів [21; 32]. Для Telegram-бота перевіряється старт `polling`, відсутність критичних `traceback`-помилки і можливість виконати короткий `smoke test`: відкрити меню, подати тестову новину, переглянути її в адміністративній панелі та скасувати або видалити тестовий матеріал.

					БР.ІІ - 25.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		56

Експлуатація програмної системи управління контентом починається з підготовки середовища, у якому одночасно працюють Telegram-бот, база даних PostgreSQL, файлове сховище медіаматеріалів і вебсервіс. На відміну від простого навчального скрипта, така система має кілька точок відмови: токен Telegram API, підключення до БД, права доступу до директорії media, доступність порту вебсайту та коректність змінних середовища. Тому експлуатаційна документація повинна описувати не лише команду запуску, а й повний порядок перевірки робочого стану.

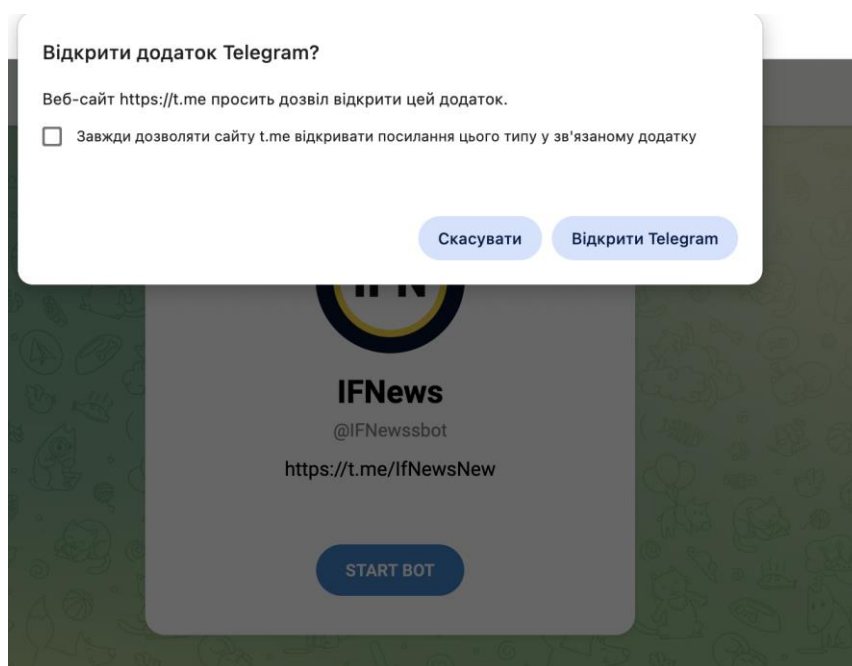


Рисунок 5.4 – Перехід в Telegram-канал

У production-умовах доцільно оформити запуск через systemd, Docker Compose або інший менеджер процесів [38]. Це дозволить автоматично перезапускати сервіс після збою, зберігати лог-файли, обмежувати права процесу та контролювати змінні середовища. Release-чекліст перед запуском нової версії системи наведено в таблиці 5.2. Навіть якщо в межах бакалаврської роботи система запускається вручну, опис такої перспективи важливий, оскільки демонструє готовність архітектури до реальної експлуатації.

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						57
Змн.	Арк.	№ докум.	Підпис	Дата		

Release-чекліст перед запуском нової версії

Перевірка	Очікуваний результат	Критичність
PostgreSQL доступний	Підключення до БД виконується без помилок	висока
Alembic upgrade head	Схема БД відповідає актуальним моделям	висока
compileall / тести	Код не має синтаксичних помилок, критичні тести пройдено	висока
Порт вебсайту	Сайт слухає налаштований порт, конфліктів немає	середня
/health	Повертає JSON зі статусом ok	висока
Polling Telegram-бота	Бот стартує і приймає оновлення	висока
Smoke test контенту	Тестова новина проходить модерацію або коректно скасовується	висока
Rollback-план	Є резервна копія БД і відомий попередній git-коміт	висока

5.3. Захист даних, резервне копіювання та моніторинг

Безпека системи управління контентом має кілька рівнів. Перший рівень - доступ до адміністративних функцій. У проєкті він реалізований через перевірку Telegram ID адміністратора. Це просте, але ефективне рішення для невеликої системи, де кількість редакторів обмежена. Якщо користувач не входить до списку адміністраторів, він не повинен мати доступу до модерації, публікації, каналів, статистики чи керування контентом.

Другий рівень - захист конфігураційних даних. Токен бота, ключ Gemini API, рядок підключення до бази даних і список адміністраторів не повинні зберігатися безпосередньо у коді. У проєкті використовується pydantic-settings і файл .env, що дозволяє відокремити секрети від програмної логіки [22; 33]. У production-середовищі такі значення бажано зберігати у секретному сховищі або змінних середовища сервера [13].

					БР.ІІ - 25.00.00.000 ПЗ	Арк.
						58
Змн.	Арк.	№ докум.	Підпис	Дата		

Третій рівень - захист вебчастини. Оскільки текст новини надходить від користувача, він не може напямую вставлятися у HTML без екранування. У вебзастосунку використовується `html.escape`, що запобігає виконанню HTML або JavaScript, введених користувачем. Це важливо для захисту від XSS-атак, коли зловмисник може спробувати вставити код у текст новини [11; 34].

Четвертий рівень - журналювання адміністративних дій. У системі є таблиця `audit_log`, яка фіксує дії адміністратора: публікацію, відхилення, редагування, видалення контенту або зміну медіа. Такий журнал не лише допомагає розслідувати помилки, а й дисциплінує роботу з контентом. Якщо в майбутньому адміністраторів стане більше, `audit_log` стане основою для контролю відповідальності.

Таблиця 5.3

План резервного копіювання

Об'єкт	Періодичність	Спосіб відновлення
База даних	Щоденно та перед релізом	<code>pg_restore</code> або імпорт SQL-дампу
Медіафайли	Щотижнево або після великих публікацій	Повернення директорії <code>media</code> з архіву
<code>.env</code>	Перед зміною конфігурації	Відновлення попередньої конфігурації
Код	Кожен реліз	Повернення до попереднього <code>git</code> -коміту

Резервне копіювання є обов'язковою складовою експлуатації системи, оскільки база даних містить користувачів, заявки, статуси, публікації, повідомлення та журнал аудиту. Втрата цих даних означала б втрату історії редакційного процесу. Для PostgreSQL доцільно використовувати `pg_dump` або інший штатний механізм створення резервних копій [35]. Копії потрібно зберігати окремо від основного сервера.

Окрім бази даних, необхідно резервувати директорію `media`, а загальний план резервного копіювання основних компонентів системи подано в таблиці 5.3. У ній зберігаються фото і відео, отримані від користувачів. Якщо зберегти лише БД без `media`, вебпублікації можуть залишитися без зображень, а адміністратор не

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						59
Змн.	Арк.	№ докум.	Підпис	Дата		

зможе відновити повний контекст новини. Тому резервне копіювання БД і файлового сховища повинно виконуватися узгоджено.

Моніторинг системи потрібний для раннього виявлення проблем. У проєкті вже використовується стандартне логування Python, яке показує старт бота, polling, обробку оновлень, помилки публікації, помилки AI-сервісу та інші події. Для локального прототипу цього достатньо, але для стабільної експлуатації логи потрібно зберігати у файлі або централізованій системі [32].

Основними подіями для моніторингу є недоступність БД, помилки Telegram API, помилки завантаження медіа, помилки публікації в канал, помилки генерації вебпоста і перевищення квоти AI-сервісу. На рисунку 5.5 зображено окрему сторінку з новиною та детальною інформацією.

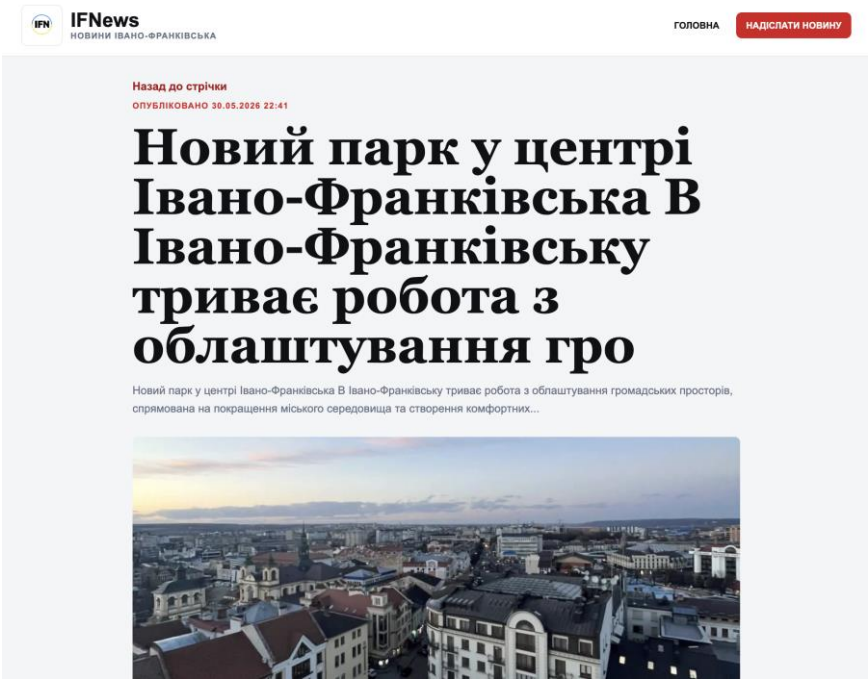


Рисунок 5.5 – Сторінка окремої новини

Для стабільної експлуатації варто визначити мінімальний набір показників, які потрібно контролювати. До них належать доступність PostgreSQL, кількість помилок Telegram API, кількість невдалих публікацій, обсяг директорії media, час відповіді маршруту /health, кількість заявок зі статусом new і наявність останньої резервної копії. Навіть без складної системи моніторингу ці показники можна періодично перевіряти через логи, healthcheck і службові команди.

Реагування на інциденти також має бути описане заздалегідь. Якщо недоступна БД, потрібно зупинити бота або перевести його в режим повідомлення про технічну помилку. Якщо заповнюється диск, необхідно тимчасово обмежити приймання великих медіафайлів і виконати cleanup або перенесення архіву. Якщо не працює AI-сервіс, основний сценарій модерації повинен залишатися доступним без функцій покращення тексту.

5.4. Масштабування, практична доцільність і ризики розвитку

Поточна реалізація є завершеним прототипом, але вона має природні обмеження. Перше обмеження полягає у використанні polling-режиму Telegram-бота. Polling простий для локального запуску і розробки, однак у production-середовищі часто доцільніше використовувати webhook. Webhook дозволяє Telegram самостійно надсилати оновлення на сервер, зменшує затримку і краще підходить для розгортання за reverse проху [14].

Друге обмеження - локальне зберігання медіафайлів. Для бакалаврського проєкту це виправдано, оскільки спрощує архітектуру і не потребує додаткових хмарних сервісів. Проте в реальній експлуатації кількість фото і відео швидко зростає. Локальний диск може заповнитися, резервне копіювання стане повільним, а масштабування на кілька серверів ускладниться. Тому перспективним є перенесення медіа в S3-сумісне сховище [39].

Масштабування системи можна розглядати у трьох напрямках: функціональне, технічне та організаційне, а потенційні функції наступних версій наведено в таблиці 5.4. Функціональне масштабування означає додавання нових можливостей: категорій, тегів, пошуку, планування публікацій, чернеток, шаблонів повідомлень і розширеної статистики. Технічне масштабування означає здатність системи обробляти більше користувачів, більше медіа і більше публікацій. Організаційне масштабування означає підтримку кількох адміністраторів, ролей і правил редакційного процесу, рисунок 5.5.

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						61
Змн.	Арк.	№ докум.	Підпис	Дата		

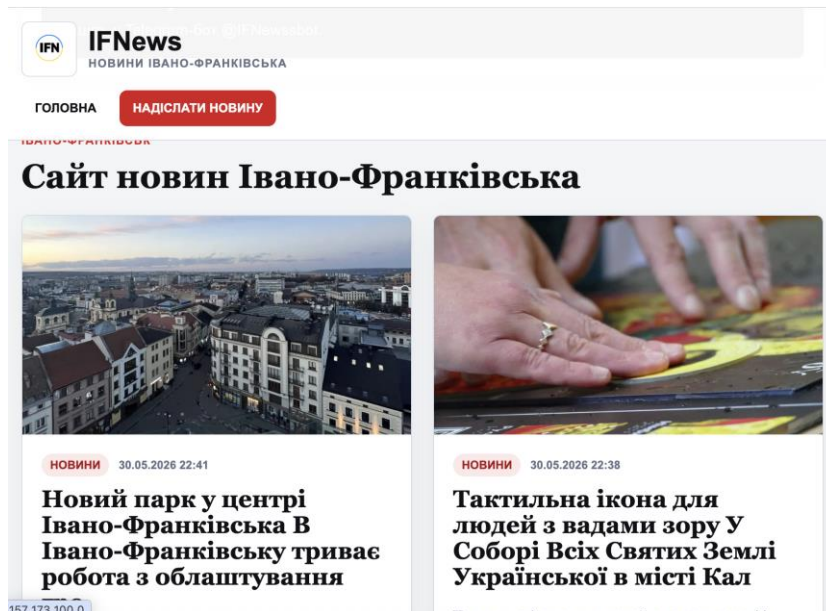


Рисунок 5.5 – Зміна розміру сайту

На рівні бази даних масштабування передбачає індексацію полів, які часто використовуються у фільтрах: `status`, `created_at`, `telegram_id`, `slug`, `is_published`. Для поточного прототипу це не критично, але зі зростанням кількості новин запити до черги модерації, списку активного контенту і вебпублікацій повинні залишатися швидкими [19; 36]. На рівні медіафайлів масштабування передбачає винесення файлів у зовнішнє сховище і збереження у БД не локального шляху, а URL або ключа об'єкта [39].

Практична доцільність розробленої системи полягає у зменшенні ручної роботи адміністратора. Без такої системи редактор отримує повідомлення у приватних чатах, копіює текст, зберігає фото, вручну переносить матеріал у канал і окремо створює публікацію на сайті. Це займає час і створює ризик помилки. Розроблена система об'єднує ці дії в один керований процес .

Економічний ефект для невеликого локального медіапроекту проявляється не у прямому прибутку, а в економії часу, зменшенні кількості помилок і підвищенні швидкості публікації. Якщо адміністратор щодня обробляє десятки повідомлень, навіть скорочення одного сценарію на кілька хвилин дає відчутну економію протягом місяця. Використання відкритих технологій також зменшує вартість реалізації, оскільки Python, aiogram, SQLAlchemy, PostgreSQL, Alembic і

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		62

aiohhttp можна використовувати без ліцензійних платежів.

Подальший розвиток системи пов'язаний із технічними й організаційними ризиками, основні з яких разом зі способами зменшення наведено в таблиці 5.5. Перший ризик - залежність від Telegram API. Якщо Telegram змінить обмеження Bot API, формат media group або правила роботи каналів, частину логіки доведеться адаптувати. Другий ризик - зростання обсягу медіафайлів, що може призвести до заповнення диска. Третій ризик - помилки адміністратора, які можна зменшити через підтвердження критичних дій, журнал аудиту і розмежування ролей. Четвертий ризик - нестабільність зовнішнього AI-сервісу, тому AI повинен залишатися допоміжною, а не критичною функцією.

Таблиця 5.4

Потенційні функції наступних версій

Функція	Призначення	Пріоритет
Категорії	Групування новин за тематикою	високий
Пошук	Швидке знаходження матеріалів у архіві	високий
Планування	Відкладена публікація	середній
Ролі	Розмежування прав редакторів	високий
Аналітика	Оцінка активності та ефективності модерації	середній

Таблиця 5.5

Ризики розвитку та способи їх зменшення

Ризик	Ймовірний вплив	Спосіб зменшення
Залежність від Telegram API	Порушення сценаріїв бота	Ізоляція API-логіки у сервісах
Зростання media	Заповнення диска	Квоти, cleanup, S3-сховище
Помилка редактора	Некоректна публікація	Підтвердження дій і audit_log
Недоступність AI	Не працює покращення тексту	AI як додаткова, а не критична функція
Ускладнення коду	Важкий супровід	Модульність і автоматизовані тести

5.5. Експериментальна оцінка продуктивності вебчастини

Для кількісної оцінки реалізованого прототипу було виконано локальну перевірку основних HTTP-маршрутів. Метою експерименту було визначення часу відповіді та стабільності вебчастини під час послідовних і паралельних звернень. Перевірялися /health, головна сторінка / та JSON API /api/posts. Healthcheck формує коротку відповідь, головна сторінка отримує вебпости з PostgreSQL і створює HTML, а API повертає структурований JSON [19; 21; 30; 32].

Основними метриками обрано середній, мінімальний і максимальний час, а також 95-й перцентиль. Успішним вважався запит зі статусом HTTP 200 і повністю отриманим тілом відповіді. Після smoke test для кожного маршруту виконано 50 послідовних запитів, результати яких наведено в таблиці 5.6.

Таблиця 5.6

Результати 50 послідовних HTTP-запитів

Маршрут	Середній час, мс	Мінімум, мс	Максимум, мс	95-й перцентиль, мс
/health	0,63	0,30	11,39	0,62
/	3,75	1,61	81,13	3,41
/api/posts	2,31	1,49	8,31	4,48

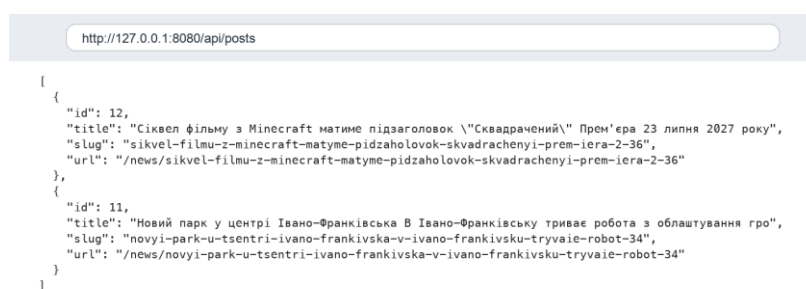


Рисунок 5.6 - Відповідь JSON API зі списком опублікованих матеріалів

Приклад фактичної відповіді маршруту /api/posts зі схваленими та опублікованими матеріалами наведено на рисунку 5.6.

Найменший середній час продемонстрував /health - 0,63 мс, що

пояснюється відсутністю запитів до бази даних. Головна сторінка мала середній час 3,75 мс, оскільки отримує вебпости, формує картки новин і повертає HTML. Маршрут /api/posts показав 2,31 мс: він також працює з PostgreSQL, але повертає компактний JSON без повного HTML-рендерингу. Одиничний максимум 81,13 мс для головної сторінки не спричинив помилки, але в production-середовищі подібні піки слід зіставляти з журналами та станом БД.

Друга частина експерименту моделювала короткий сплеск активності: для головної сторінки й API виконано по 200 запитів із паралельністю 20. Результати паралельної перевірки наведено в таблиці 5.7 [21; 28; 30].

Таблиця 5.7

Результати паралельної перевірки вебмаршрутів

Маршрут	Запити	Паралельність	Успішні	Загальний час, с	Середній час, мс	p95, мс
/	200	20	200	0,23	21,69	55,02
/api/posts	200	20	200	0,19	17,81	30,18

Усі 400 паралельних запитів завершилися зі статусом 200: застосунок не повертав помилки 500 і не зависав. Середній час головної сторінки зріс до 21,69 мс, а API - до 17,81 мс. Це закономірно, оскільки одночасні операції використовували цикл подій і з'єднання PostgreSQL. API зберіг перевагу через менший обсяг формування представлення.

Результати свідчать, що вебчастина має достатню продуктивність для поточного прототипу та невеликого локального новинного сервісу. Водночас експеримент виконувався на одному комп'ютері, без реальної мережевої затримки та з невеликим набором даних. Отримані значення не можна трактувати як гарантію для production-середовища. Після розгортання потрібно повторити тест через мережу, збільшити тривалість і контролювати CPU, пам'ять, диск та кількість з'єднань PostgreSQL.

Подальша оптимізація може охоплювати індекси для status, created_at, slug та is_published, пагінацію, кешування головної сторінки і винесення медіафайлів

до S3-сумісного сховища [8; 19; 39].

Таким чином, експериментальна перевірка доповнила функціональне тестування кількісними результатами. Послідовні запити виконувалися в межах кількох мілісекунд, а паралельна серія не виявила втрати запитів або аварійних відповідей. Подальші оптимізації доцільно впроваджувати на основі метрик реального використання, а не через передчасне ускладнення архітектури.

5.6. Висновки по розділу

У п'ятому розділі розглянуто тестування, впровадження, експлуатаційний супровід і напрями розвитку програмної системи управління контентом. Описано рівні перевірки: модульне тестування окремих функцій, інтеграційне тестування Telegram-сценаріїв, перевірку вебмаршрутів, healthcheck, API та ручне приймальне тестування ключових сценаріїв.

Визначено release-перевірки перед запуском нової версії: доступність PostgreSQL, застосування Alembic-міграцій, синтаксична перевірка коду, запуск Telegram-бота і вебсервісу, перевірка маршруту /health та виконання smoke test. Така послідовність дозволяє виявити критичні помилки до початку повноцінної експлуатації системи. Окремо описано питання безпеки, резервного копіювання, моніторингу та відновлення після помилок.

Під час тестування враховано не лише зовнішню поведінку системи, а й правильність зміни її внутрішнього стану. Для кожного сценарію необхідно перевірити статус новинної заявки, збереження тексту й медіафайлів, створення запису про публікацію та фіксацію адміністративної дії в журналі аудиту. Це дає змогу виявляти помилки, які можуть бути непомітними в інтерфейсі, але призводити до дублювання матеріалів, втрати даних або неправильного відображення активного контенту.

Експериментальна перевірка вебчастини підтвердила працездатність головної сторінки, маршруту перевірки стану системи та JSON API. Послідовні запити виконувалися в межах кількох мілісекунд, а під час паралельної перевірки

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		66

всі запити завершилися успішно. Отримані результати свідчать, що продуктивності поточної реалізації достатньо для прототипу та невеликого локального новинного сервісу. Водночас після розгортання на сервері доцільно повторити вимірювання з урахуванням мережевої затримки, більшого обсягу даних і реального навантаження.

Важливою складовою експлуатації є контрольована обробка помилок. Якщо зовнішній сервіс або Telegram API не повертає успішної відповіді, статус новини не повинен змінюватися на опублікований. Помилка має бути зафіксована в технічному журналі, а адміністратор повинен отримати зрозуміле повідомлення та можливість повторити операцію. Така поведінка забезпечує цілісність редакційного процесу і зменшує ризик розбіжностей між станом бази даних, Telegram-каналу та вебсайту.

Проаналізовано напрями масштабування системи: перехід із polling на webhook, контейнеризація, перенесення медіафайлів у зовнішнє сховище, додавання ролей, категорій, пошуку, планування публікацій і аналітики. Перехід до webhook може підвищити ефективність отримання оновлень у серверному середовищі, а контейнеризація — спростити розгортання та відтворення конфігурації. Використання зовнішнього сховища зменшить залежність від локального диска і полегшить масштабування вебчастини.

Отже, результати п'ятого розділу підтверджують, що розроблена система охоплює не лише основні функції приймання, модерації та публікації новин, а й необхідні процеси перевірки, запуску та подальшого супроводу. Розроблений прототип має практичну цінність і може бути основою для розвитку локального новинного сервісу з більшою кількістю користувачів, каналів і матеріалів.

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						67
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У бакалаврській роботі виконано розроблення програмної системи управління контентом. Досягнуто поставленої мети: створено прототип на основі Telegram-бота, який дозволяє користувачам надсилати новини з текстом, фото або відео, а адміністраторам - модерувати, редагувати, відхиляти чи публікувати матеріали у Telegram-канал та на вебсайт.

У процесі виконання роботи проаналізовано предметну область локальних цифрових медіа, визначено ролі користувачів і адміністраторів, сформульовано функціональні та нефункціональні вимоги. Розроблено архітектуру системи з розділенням на Telegram-обробники, сервісний шар, repository-рівень, базу даних і вебчастину. Спроектовано модель даних, що охоплює користувачів, новини, медіафайли, канали, публікації, вебпости, повідомлення та журнал аудиту.

Практична реалізація виконана мовою Python із використанням aiogram, SQLAlchemy asyncpg, Alembic, aiohttp, pydantic-settings і google-genai. Особливу увагу приділено обробці медіаальбомів, транзакційному збереженню даних, очищенню службових повідомлень, захисту адміністративних дій, генерації вебпостів і безпечному екрануванню HTML. Реалізований код має модульну структуру, що полегшує супровід і розширення.

Результати тестування підтверджують працездатність основних сценаріїв: подання новини, додавання медіа, підтвердження заявки, перегляд на модерації, редагування, відхилення, публікація у Telegram-канал, створення вебсторінки та отримання JSON API. Практичне значення роботи полягає у можливості використання системи як основи для локального новинного сервісу або внутрішнього інструмента приймання повідомлень від аудиторії.

Окремо розглянуто питання впровадження, release-перевірок, експлуатації, безпеки, резервного копіювання, відновлення, моніторингу та реагування на інциденти. Це дозволяє оцінити систему не лише як набір програмних модулів, а як приклад прикладного продукту, який можна запускати, супроводжувати, оновлювати і контролювати після впровадження.

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						68
Змн.	Арк.	№ докум.	Підпис	Дата		

У роботі також обґрунтовано доцільність використання Telegram як основного інтерфейсу взаємодії з користувачами та адміністраторами. Такий підхід зменшує потребу у створенні окремого клієнтського застосунку, оскільки більшість користувачів уже має доступ до месенджера і може швидко надіслати текст, фото або відео. Для адміністратора Telegram-інтерфейс забезпечує оперативну модерацію, перегляд заявок, керування публікаціями та взаємодію з контентом без переходу до складної зовнішньої панелі.

Важливим результатом роботи є формування цілісного редакційного процесу, у якому кожна новина має зрозумілий життєвий цикл: створення заявки, збереження у базі даних, модерація, схвалення або відхилення, публікація та подальше відображення на сайті. Наявність статусів, журналу аудиту, зв'язку між новиною і медіафайлами, а також окремих записів публікацій підвищує контрольованість системи й зменшує ризик втрати або дублювання матеріалів.

Під час виконання роботи було підтверджено, що поєднання Telegram-бота, реляційної бази даних і вебчастини є доцільним для задачі управління новинним контентом. Telegram забезпечує швидку взаємодію з користувачами та адміністраторами, база даних відповідає за збереження структурованої інформації, а вебчастина дає змогу відображати схвалені матеріали для ширшої аудиторії. Такий підхід дозволяє побудувати систему, у якій приймання, перевірка, публікація та подальше зберігання новин об'єднані в один узгоджений процес.

Подальший розвиток проєкту може включати впровадження webhook-режиму, контейнеризацію, хмарне зберігання медіафайлів, розширену вебадмінпанель, автоматизовані тести, рольову модель прав доступу, пошук по архіву та аналітику активності користувачів. У межах бакалаврської роботи реалізовано завершений прототип, який демонструє застосування сучасних методів інженерії програмного забезпечення для прикладної задачі управління новинним контентом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. Київ: ДП «УкрНДНЦ», 2016 (дата звернення: 10.03.2026).
2. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ: ДП «УкрНДНЦ», 2016 (дата звернення: 10.03.2026).
3. Sommerville I. Software Engineering. 10th ed. Pearson, 2015. 816 p. (дата звернення: 10.03.2026).
4. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. McGraw-Hill, 2019 (дата звернення: 10.03.2026).
5. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002. 560 p. (дата звернення: 10.03.2026).
6. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p. (дата звернення: 10.03.2026).
7. Humble J., Farley D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley, 2010. 512 p. (дата звернення: 10.03.2026).
8. Kleppmann M. Designing Data-Intensive Applications. O'Reilly Media, 2017. 616 p. (дата звернення: 10.03.2026).
9. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994. 395 p. (дата звернення: 10.03.2026).
10. ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models (дата звернення: 10.03.2026).
11. OWASP Foundation. OWASP Top 10:2021. URL: <https://owasp.org/Top10/> (дата звернення: 15.03.2026).
12. Semantic Versioning 2.0.0. URL: <https://semver.org/> (дата звернення: 15.03.2026).

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

13. The Twelve-Factor App. URL: <https://12factor.net/> (дата звернення: 15.03.2026).
14. Telegram Bot API Documentation. URL: <https://core.telegram.org/bots/api> (дата звернення: 25.03.2026).
15. aiogram Documentation. URL: <https://docs.aiogram.dev/> (дата звернення: 25.03.2026).
16. Python Software Foundation. Python 3 Documentation. URL: <https://docs.python.org/3/> (дата звернення: 25.03.2026).
17. SQLAlchemy Documentation. URL: <https://docs.sqlalchemy.org/> (дата звернення: 25.03.2026).
18. Alembic Documentation. URL: <https://alembic.sqlalchemy.org/> (дата звернення: 25.03.2026).
19. PostgreSQL Global Development Group. PostgreSQL Documentation. URL: <https://www.postgresql.org/docs/> (дата звернення: 25.03.2026).
20. Asyncpg Documentation. URL: <https://magicstack.github.io/asyncpg/current/> (дата звернення: 25.03.2026).
21. aiohttp Documentation. URL: <https://docs.aiohttp.org/> (дата звернення: 25.03.2026).
22. Pydantic Documentation. URL: <https://docs.pydantic.dev/> (дата звернення: 25.03.2026).
23. Google Gen AI SDK Documentation. URL: <https://googleapis.github.io/python-genai/> (дата звернення: 03.04.2026).
24. MDN Web Docs. HTML: HyperText Markup Language. URL: <https://developer.mozilla.org/docs/Web/HTML> (дата звернення: 03.04.2026).
25. MDN Web Docs. CSS: Cascading Style Sheets. URL: <https://developer.mozilla.org/docs/Web/CSS> (дата звернення: 03.04.2026).
26. Reuters Institute for the Study of Journalism. Digital News Report 2025. URL: <https://reutersinstitute.politics.ox.ac.uk/digital-news-report/2025> (дата звернення: 03.04.2026).
27. Reuters Institute for the Study of Journalism. Digital News Report 2024. URL: <https://reutersinstitute.politics.ox.ac.uk/digital-news-report/2024> (дата звернення: 03.04.2026).

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						71
Змн.	Арк.	№ докум.	Підпис	Дата		

28. Python Software Foundation. asyncio — Asynchronous I/O. URL: <https://docs.python.org/3/library/asyncio.html> (дата звернення: 03.04.2026).
29. aioogram Documentation. Finite State Machine. URL: https://docs.aioogram.dev/en/latest/dispatcher/finite_state_machine/index.html (дата звернення: 03.04.2026).
30. SQLAlchemy Documentation. Asynchronous I/O. URL: <https://docs.sqlalchemy.org/en/20/orm/extensions/asyncio.html> (дата звернення: 03.04.2026).
31. Alembic Documentation. Tutorial. URL: <https://alembic.sqlalchemy.org/en/latest/tutorial.html> (дата звернення: 03.04.2026).
32. aiohttp Documentation. Web Server Quickstart. URL: https://docs.aiohttp.org/en/stable/web_quickstart.html (дата звернення: 03.04.2026).
33. Pydantic Documentation. Settings Management. URL: https://docs.pydantic.dev/latest/concepts/pydantic_settings/ (дата звернення: 03.04.2026).
34. OWASP Foundation. Cross Site Scripting Prevention Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html (дата звернення: 03.04.2026).
35. PostgreSQL Global Development Group. pg_dump. URL: <https://www.postgresql.org/docs/current/app-pgdump.html> (дата звернення: 03.04.2026).
36. PostgreSQL Global Development Group. Full Text Search. URL: <https://www.postgresql.org/docs/current/textsearch.html> (дата звернення: 03.04.2026).
37. GitHub Docs. Writing workflows. URL: <https://docs.github.com/en/actions/how-tos/write-workflows> (дата звернення: 03.04.2026).
38. Docker Docs. Docker Compose. URL: <https://docs.docker.com/compose/> (дата звернення: 03.04.2026).
39. Amazon Web Services. What is Amazon S3? URL: <https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html> (дата звернення: 03.04.2026).

					БР.ІІІ - 25.00.00.000 ПЗ	Арк.
						72
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

Додаток А. Фрагменти інтерфейсу програмної системи

Інтерфейс програмної системи побудовано навколо Telegram-бота, через який користувач подає матеріал, а адміністратор керує редакційним процесом. У додатку наведено приклади екранів, які демонструють основні дії з новинним контентом.

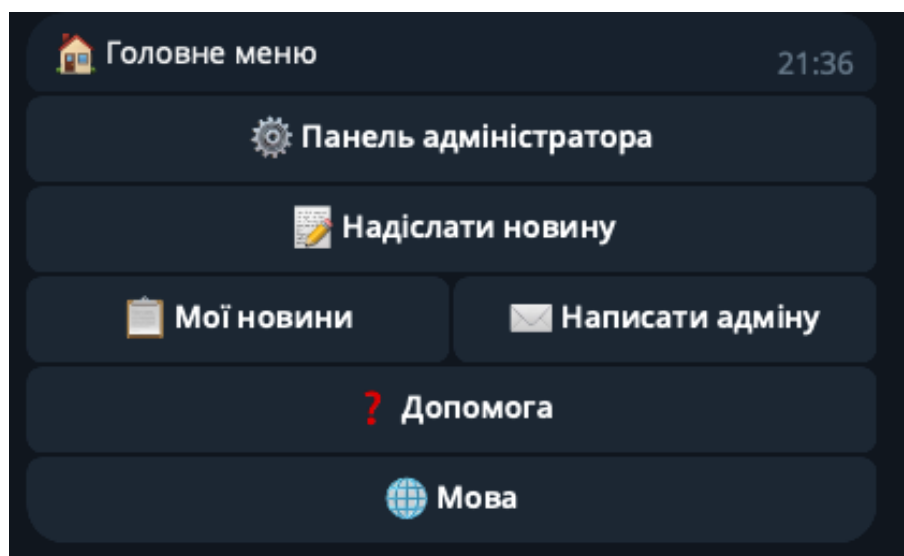


Рисунок А.1 – Головне меню користувача Telegram-бота

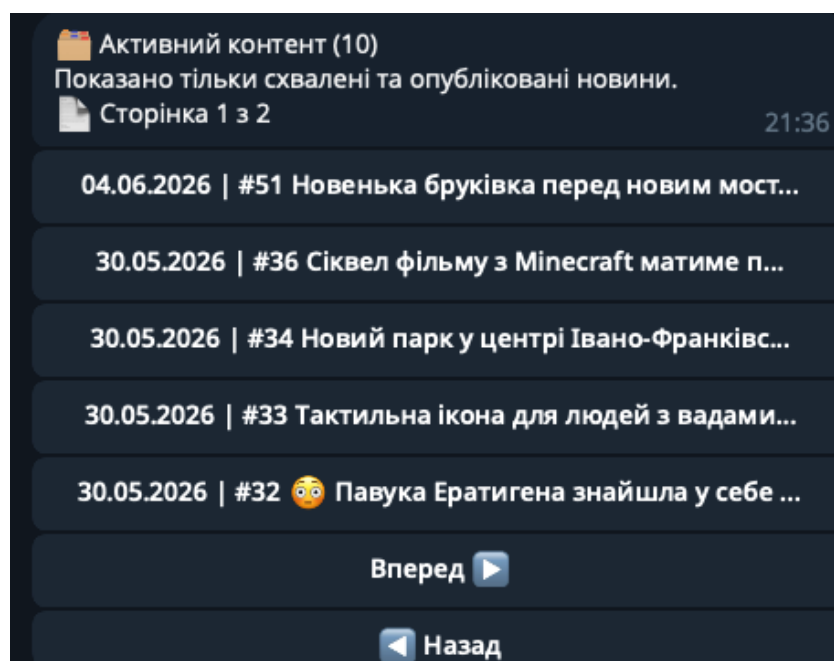


Рисунок А.2 – Розділ управління контентом адміністратора

Додаток Б. Фрагменти коду

Програмна реалізація складається з окремих пакетів для обробників Telegram-подій, сервісів, доступу до бази даних, клавіатур, станів і вебчастини. Нижче наведено компактні фрагменти структури проєкту та сервісу публікації на сайт.

Структура проєкту

```
telegram_news_bot/  
  bot/  
    handlers/  
      admin/  
      user/  
    services/  
    db/  
      repos/  
    web/  
    keyboards/  
    states/  
  alembic/  
  media/  
  requirements.txt
```

Рисунок Б.1 – Структура основних пакетів проєкту

bot/services/site_publisher.py

```
1 async def publish_to_site(session, news):  
2     repo = WebPostRepo(session)  
3     existing = await repo.get_by_news_id(news.id)  
4     title = make_title(news.text)  
5     data = {  
6         "title": title,  
7         "slug": existing.slug if existing else make_slug(title, news.id),  
8         "excerpt": make_excerpt(news.text),  
9         "text": news.text,  
10        "cover_image": first_media_path(news),  
11        "is_published": True,  
12    }  
13    if existing:  
14        return await repo.update(existing.id, **data)  
15    return await repo.create(news_id=news.id, **data)
```

Рисунок Б.2 – Фрагмент сервісу publish_to_site

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: «Розроблення програмної системи управління контентом».

Обсяг пояснювальної записки: 72 аркуші

Дата закінчення бакалаврської роботи: 10 червня 2026 р.

Підпис студента _____ Угринчук А.М.