

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Інформаційна система онлайн-спільноти фанатів відеоігор»

Виконав студент групи ІПЗ-22-1
МАГАЗ Артем Олександрович

Керівник роботи:
ЄФРЕМОВ Юрій Миколайович

Рецензент:
БАКАЛЮК Тетяна Анатоліївна

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«27» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **МАГАЗ Артем Олександрович**

Керівник роботи: **ЄФРЕМОВ Юрій Миколайович**

Тема роботи: **«Інформаційна система онлайн-спільноти фанатів відеоігор»**,
затверджена Наказом закладу вищої освіти від «27» лютого 2026 р., №92/с

Вихідні дані для роботи: **процес проєктування та розробки інформаційного середовища для тематичної спільноти, яке використовує сучасні архітектурні рішення для структурування ігрового контенту та зручної взаємодії фанатів відеоігор.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Єфремов Ю.М.	28.02.2026р.	28.02.2026р.
2	Єфремов Ю.М.	28.02.2026р.	28.02.2026р.
3	Єфремов Ю.М.	28.02.2026р.	28.02.2026р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з інформаційної системи онлайн-спільноти фанатів відеоігор та пояснювальної записки. Пояснювальна записка до роботи містить 84 сторінки, 57 ілюстрацій, 16 таблиць, 2 додатки, 30 джерел.

Метою роботи є розробка інформаційної системи онлайн-спільноти фанатів відеоігор, яка дозволить підвищити залученість користувачів, структурувати великі масиви ігрової інформації та забезпечить адміністраторів ефективними інструментами для керування контентом, користувачами, базою даних і сповіщеннями.

Під час розробки застосовано принципи клієнт-серверної архітектури, методи об'єктно-орієнтованого програмування та моделювання реляційних баз даних.

У результаті дослідження спроектовано та розроблено повноцінний вебзастосунок із використанням технологій ASP.NET, React та СУБД PostgreSQL. Реалізовано модулі автентифікації, управління контентом, адміністрування бази даних та систему сповіщень.

Розроблений програмний продукт рекомендовано до впровадження в якості готової інформаційної платформи для підтримки геймерських онлайн-спільнот.

КЛЮЧОВІ СЛОВА: ВЕБЗАСТОСУНОК, ВІДЕОГРА, ЛОР, СПІЛЬНОТА, ASP.NET, REACT.

					ІПЗ.КР.Б-121-26-ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Інформаційно-довідковий сайт спільноти фанатів відеоігор	Літ.	Арк.	Аркушів
Розроб.		Магаз А. О.					3	84
Перевір.		Єфремов Ю. М.				Житомирська політехніка, група ІПЗ-22-1		
Реценз.								
Н. Контр.		Тичина О. П.						
Зав.каф.		Вакалюк Т. А.						

ABSTRACT

The bachelor's degree final project consists of a web application for an information system for a video game fan community, and an explanatory note. The explanatory note accompanying the project contains 71 pages, 57 illustrations, 16 tables, 2 appendices and 30 references.

The goal of the work is to develop an information system for a community of video game fans, which will increase user engagement, organise large amounts of gaming information, and provide administrators with effective tools for managing content, users, the database and notifications.

The methods used include client-server architecture, object-oriented design, web programming and relational database modelling.

As a result of the work, a fully-fledged web application has been designed and developed using ASP.NET, React and the PostgreSQL DBMS. Modules for authentication, content management, database administration and a notification system have been implemented.

The developed software product is recommended for implementation as a ready-made information platform to support online gaming communities.

KEYWORDS: WEB APPLICATION, VIDEO GAME, LORE, COMMUNITY, ASP.NET, REACT.

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT.....	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОНЛАЙН-СПІЛЬНОТИ ФАНАТІВ ВІДЕОІГОР.....	9
1.1 Постановка задачі.....	9
1.2 Аналіз аналогів програмного продукту.....	11
1.3 Вибір архітектури інформаційної системи онлайн-спільноти фанатів відеоігор	18
1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.....	22
Висновки до першого розділу	26
РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОНЛАЙН-СПІЛЬНОТИ ФАНАТІВ ВІДЕОІГОР.....	28
2.1 Вимоги до системи	28
2.2 Проєктування бази даних.....	35
2.3 Опис узагальнених алгоритмів та взаємодії елементів системи	39
2.4 Реалізація програмного комплексу.....	43
Висновки до другого розділу.....	48
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ІНФОРМАЦІЙНОЮ СИСТЕМОЮ ОНЛАЙН-СПІЛЬНОТИ ФАНАТІВ ВІДЕОІГОР.....	49
3.1 Порядок встановлення та налаштування параметрів системи	49
3.2 Структура інтерфейсу інформаційної системи.....	53
3.3 Тестування роботи програмного комплексу	76
Висновки до третього розділу	79
ВИСНОВКИ	81
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	82
ДОДАТКИ	85

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення

API – інтерфейс прикладного програмування

БД – база даних

СУБД – система управління базами даних

CRUD – фундаментальний набір функцій для керування даними
(створення, читання, редагування, видалення)

EF – Entity Framework

		Магаз А. О.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Актуальність теми полягає у тому, що сучасний етап розвитку інформаційних технологій характеризується стрімким зростанням індустрії інтерактивних розваг та відеоігор, що супроводжується формуванням масштабних глобальних спільнот. Зі збільшенням складності ігрових механік та обсягу внутрішньоігрового контенту виникає гостра потреба у створенні спеціалізованих платформ для систематизації, збереження та обміну накопиченими знаннями. Традиційні засоби комунікації, такі як розрізнені форуми чи соціальні мережі, вже не здатні задовольнити вимоги користувачів щодо швидкості пошуку, структурованості інформації та зручності навігації. Це зумовлює високу необхідність розробки спеціалізованих веб-орієнтованих інформаційних середовищ у форматі довідкових вікі-систем, які забезпечують централізований доступ до верифікованого тематичного контенту. Розробка такого програмного рішення дозволяє не лише консолідувати розрізнені дані в єдиній структурованій базі знань, але й надати користувачам зручний інструментарій для спільної взаємодії, створення нових статей та модерації матеріалів. Враховуючи високі вимоги сучасної аудиторії до швидкодії та кросплатформності вебресурсів, використання передових технологій розробки односторінкових застосунків та надійних серверних архітектур робить дане дослідження своєчасним, практично значущим та затребуваним в умовах сучасного ринку.

Метою роботи є розробка інформаційної системи онлайн-спільноти фанатів відеоігор.

Завданням роботи є поетапне виконання наступних кроків для досягнення поставленої мети: проведення аналізу предметної області та існуючих підходів до організації тематичних веб-середовищ; вибір і детальне обґрунтування програмних засобів та технологічного стеку; проектування архітектури клієнт-серверної системи та логічної структури реляційної бази даних; безпосередня практична реалізація серверної та клієнтської частин платформи; проведення

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

комплексного тестування готового програмного забезпечення для підтвердження його стабільної та коректної роботи.

Об’єктом дослідження є процес організації та функціонування веб-орієнтованих інформаційних середовищ для взаємодії тематичних онлайн-спільнот.

Предметом дослідження є використання сучасних веб-орієнтованих технологій для розробки системи взаємодії тематичних онлайн-спільнот.

У роботі було використано методи системного аналізу для дослідження предметної області, методи об’єктно-орієнтованого проєктування для розробки архітектури програмного забезпечення, методи теорії баз даних для побудови реляційної моделі, а також сучасні методи веб-програмування та UX/UI дизайну для створення користувацького інтерфейсу.

		Магаз А. О.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОНЛАЙН-СПІЛЬНОТИ ФАНАТІВ ВІДЕОІГОР

1.1 Постановка задачі

Стрімкий розвиток ігрової індустрії та збільшення кількості активних гравців призводить до формування масштабних онлайн-спільнот. Сучасні відеоігри часто містять глибокий сюжет, складні ігрові механіки, розгалужену систему персонажів та локацій. Відповідно, у фанатів виникає постійна потреба в обміні досвідом, пошуку достовірної довідкової інформації та комунікації з іншими гравцями [24].

Головною проблемою існуючих рішень є децентралізація інформації: користувачі змушені шукати дані на розрізнених форумах, у соціальних мережах або на неструктурованих вікі-сторінках. Крім того, адміністратори таких спільнот часто стикаються з відсутністю зручних інструментів для модерації контенту, управління базою даних та розсилки сповіщень.

Головною задачею даної випускної кваліфікаційної роботи є проектування та розробка спеціалізованої веб-орієнтованої інформаційної системи для онлайн-спільноти фанатів відеоігор, яка об'єднає в собі функції структурованої бази знань та платформи для соціальної взаємодії.

Для успішного вирішення цієї задачі необхідно реалізувати програмний комплекс, який задовольнятиме вимоги різних груп користувачів. Такий підхід дозволить централізувати накопичений ігровий контент та створити безпечне середовище для його постійного оновлення. Розроблений продукт повинен гарантувати стабільну роботу та мати інтуїтивно зрозумілий інтерфейс для швидкої навігації по розділах бази знань [19].

У рамках розробки системи необхідно автоматизувати та реалізувати наступні задачі, наведені у табл.1.1:

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

Основні задачі розробки інформаційної системи

Роль	Перелік завдань
Неавторизовані користувачі	Забезпечення доступу до перегляду публічного довідкового контенту;
	Реалізація інтуїтивно зрозумілої системи пошуку та фільтрації інформації за категоріями [19];
	Забезпечення можливості реєстрації та авторизації в системі для отримання розширених прав.
Авторизовані користувачі	Керування даними власного особистого профілю;
	Публікація та видалення власних коментарів;
	Формування та надсилання повідомлень про виявлені неточності або помилки в інформаційних статтях.
Адміністратор системи	Робота із захищеною панеллю керування для повного адміністрування платформи;
	Створення, читання, редагування та видалення статей і категорій;
	Блокування доступу користувачів до системи на визначений термін;
	Видалення коментарів інших користувачів.
Технічна та архітектурна складова	Проектування нормалізованої реляційної бази даних для надійного збереження ігрового контенту [3, 15];
	Оптимізація запитів до сервера для забезпечення високої швидкості завантаження клієнтського інтерфейсу [25, 26];
	Впровадження надійних механізмів захисту даних [7, 18]

Успішна реалізація наведених вище завдань дозволить створити комплексну платформу, яка повністю задовольнятиме потреби цільової аудиторії. Спроектowana система стане надійним фундаментом для подальшого масштабування та розвитку фанатської спільноти.

1.2Аналіз аналогів програмного продукту

Для розробки інформаційно-довідкового сайту спільноти фанатів відеоігор необхідно дослідити програмні продукти конкурентів. Аналіз існуючих рішень дозволяє виявити їхні сильні та слабкі сторони, а також визначити ключові функції, які необхідно реалізувати у власному вебзастосунку [19, 24].

Як основний аналог для дослідження було обрано платформу Fandom – найбільший у світі вікі-хостинг, орієнтований на поп-культуру та відеоігри. На рисунку 1.1 зображено головну сторінку платформи Fandom.

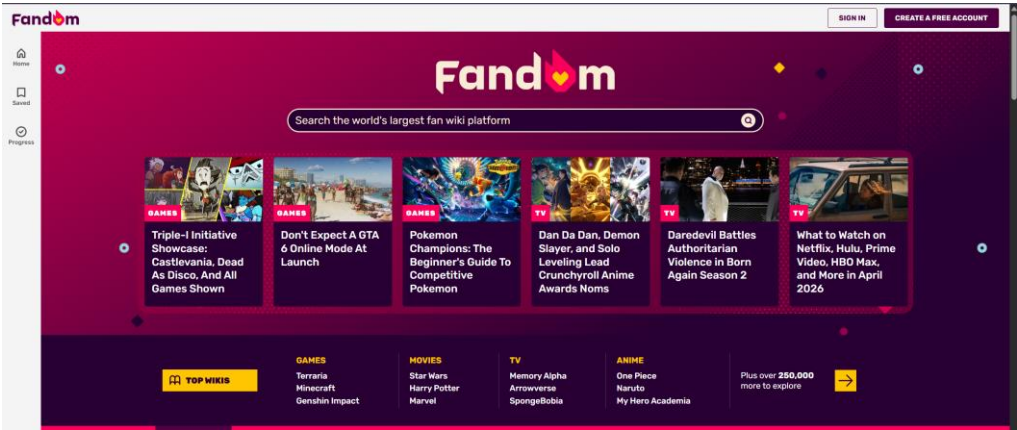


Рис. 1.1 Головна сторінка Fandom

Fandom надає користувачам інструменти для створення спеціалізованих баз знань, присвячених конкретним ігровим всесвітам. Платформа побудована на принципах краудсорсингу, де контент створюється та модерується самою спільнотою фанатів.

Головною перевагою платформи Fandom є можливість створювати зручну структуру сторінок, категорій та шаблонів. Це ідеально підходить для детального опису складного ігрового лору: сюжетів, персонажів чи локацій. Завдяки великій аудиторії платформи нові статті з'являються та оновлюються дуже швидко. Крім

того, матеріали бази знань добре оптимізовані, тому майже завжди займають перші місця в пошукових системах за ігровими запитами [8].

Проте платформа має і суттєві недоліки. Найбільшою проблемою є закритість спілкування: щоб просто почитати коментарі під статтею, користувачу обов'язково треба авторизуватися, що часто відштовхує нових відвідувачів. Ще один вагомий недолік – це надмірна кількість нав'язливої реклами та відео, які відтворюються автоматично. Це не лише псує загальне враження від сайту, але й сильно уповільнює завантаження сторінок. До того ж, адміністратори спільнот дуже обмежені в налаштуваннях, оскільки практично не можуть змінювати дизайн або розширювати функціонал своїх розділів.

MediaWiki – це вільне програмне забезпечення (вікі-рушій), на базі якого працює Вікіпедія та тисячі інших незалежних енциклопедій. Це галузевий стандарт для створення відкритих баз знань, який дозволяє розгорнути власне інформаційне середовище на незалежному сервері. На рисунку 1.2 зображено приклад статті на MediaWiki.

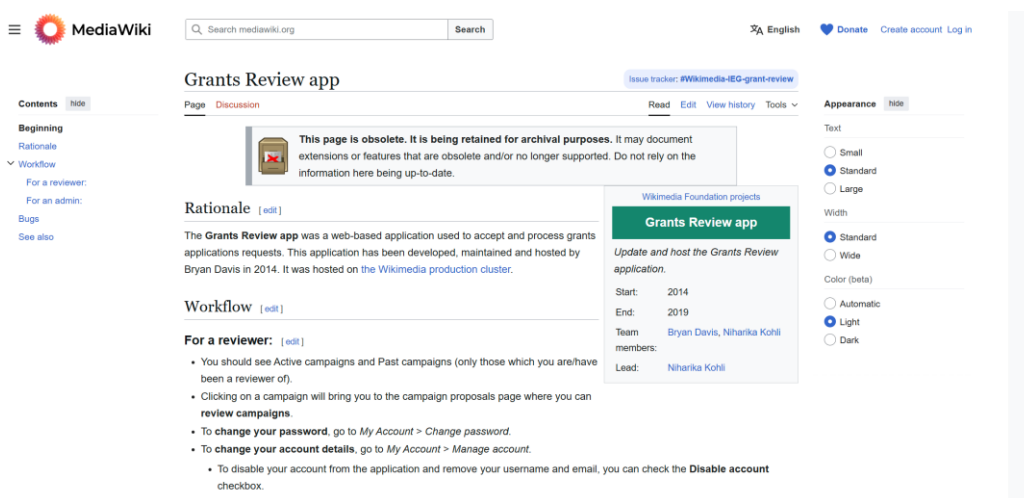


Рис. 1.2 MediaWiki

Сильною стороною MediaWiki є відкрита архітектура вихідного коду. Ця особливість відкриває широкі можливості для глибокої кастомізації платформи, розробки унікального дизайну та інтеграції зовнішніх модулів. Важливим аспектом безпеки є механізм збереження повної історії редагувань сторінок [28], що є ефективним інструментом протидії вандалізму у відкритих системах. Автономність розгортання на власному серверному обладнанні також нівелює

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		12

ризика блокування ресурсу або втрати даних через політику сторонніх провайдерів.

Серед слабких сторін рушія виділяється високий поріг входження для контентмейкерів через використання специфічної вікі-розмітки, яка ускладнює процес форматування матеріалів для користувачів без технічного бекграунду. Стандартний графічний інтерфейс системи потребує значних трудовитрат для адаптації під сучасні вимоги юзабіліті [19]. Підтримка власної інфраструктури вимагає експертизи в адмініструванні баз даних та серверних операційних систем.

Notion – це сучасний багатофункціональний інструмент для організації робочого простору, який дедалі частіше використовується для створення баз знань та публічних вікі-сторінок завдяки своїй модульній структурі. На рисунку 1.3 зображена головна сторінка Notion.

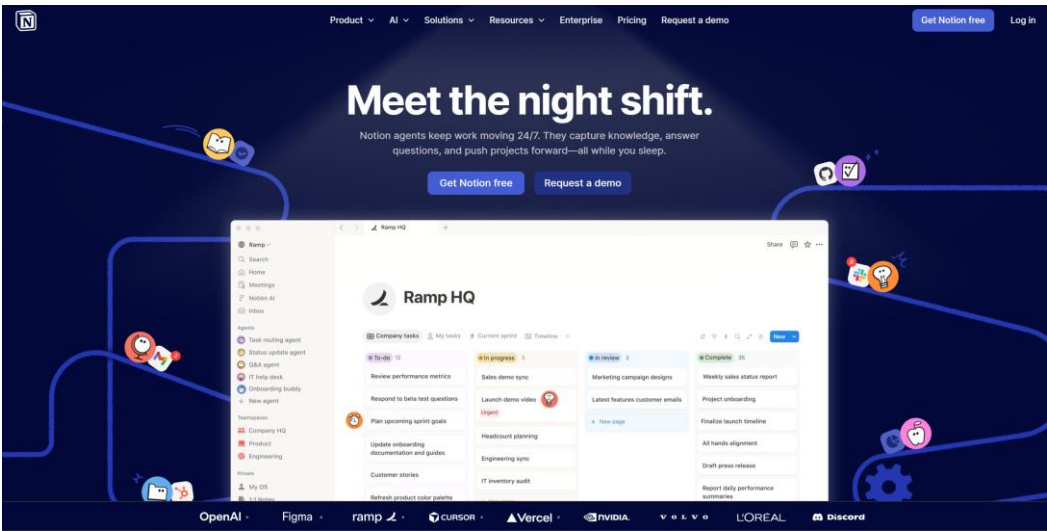


Рис. 1.3 Головна сторінка Notion

Архітектурною перевагою Notion є концепція блокового інтерфейсу, яка забезпечує високий рівень інтуїтивності під час взаємодії з контентом [19]. Система дозволяє генерувати складні візуальні компоненти, такі як галереї чи канбан-дошки з розширеною фільтрацією, що є оптимальним рішенням для каталогізації ігрових артефактів. Процес розгортання публічних матеріалів максимально спрощений і не потребує залучення сторонніх хостингових ресурсів.

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

Водночас, платформа не позиціонується як інструмент для побудови соціальних взаємодій, оскільки не має інтегрованих модулів форумів чи відкритих коментарів. Індексція публічних просторів Notion пошуковими краулерами є менш ефективною порівняно з профільними вікі-рушіями, а рендеринг сторінок з великими масивами даних може супроводжуватися затримками. Закрита екосистема компанії-розробника унеможливорює локальне розгортання системи на серверах клієнта.

Reddit – це глобальна соціальна новинна платформа, структурована за тематичними розділами, яка є одним із головних осередків спілкування геймерів у світі. На рисунку 1.4 зображена головна сторінка новинного ресурсу Reddit.

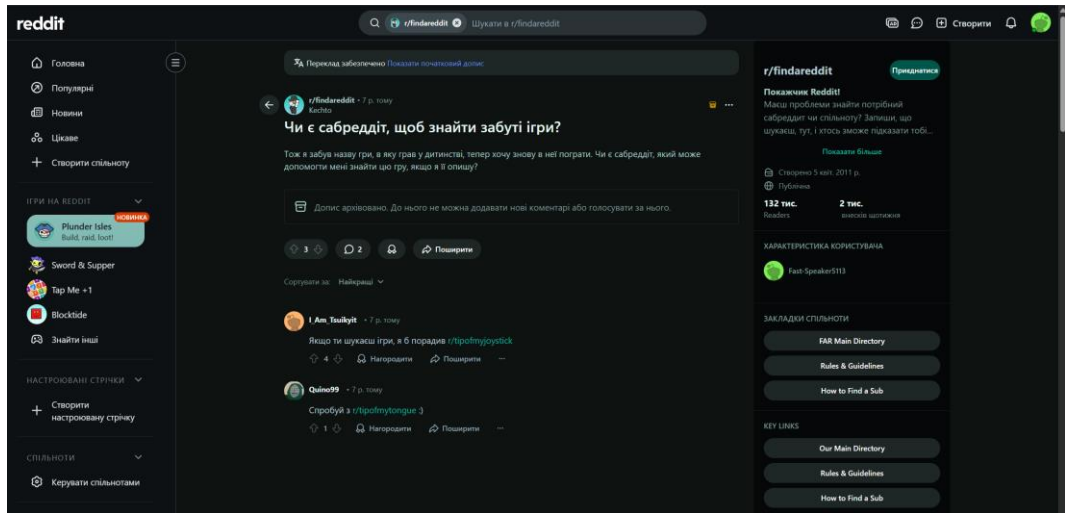


Рис. 1.4 Головна сторінка Reddit

Успіх Reddit базується на алгоритмічному ранжируванні контенту за допомогою системи оцінок, що забезпечує швидке просування найбільш релевантної інформації. Цей механізм стимулює залученість спільноти та дозволяє оперативно отримувати експертні відповіді на вузькоспеціалізовані питання. Ефективність управління сабреддітами підтримується широким спектром модераторських інструментів для контролю за дотриманням правил спільноти.

Головним недоліком архітектури Reddit у контексті вікі-систем є хронологічно-стрічковий формат відображення даних. Ця особливість призводить до швидкої втрати доступу до фундаментальних матеріалів (лорів, об'ємних гайдів), які зміщуються новими публікаціями. Обмеженість

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		14

вбудованого текстового редактора також перешкоджає створенню багаторівневих інформаційних ієрархій.

Steam Guides – це вбудована підсистема найбільшої платформи цифрової дистрибуції відеоігор Steam, яка дозволяє гравцям створювати та публікувати посібники з проходження та механік конкретних ігор.



Рис. 1.5 Головна сторінка Steam Guides

Конкурентною перевагою Steam Guides є глибока інтеграція з ігровим клієнтом, що забезпечує доступ до бази знань безпосередньо під час сесії через оверлей [24]. Прив'язка матеріалів до особистих профілів та бібліотек спрощує процес інтеграції ігрового контенту (скріншотів, досягнень) у статті. Наявність вбудованої економіки нагород додатково стимулює генерацію якісного контенту користувачами.

Однак система характеризується високим рівнем фрагментації: кожен посібник є ізольованою сутністю без можливості створення глобальних перехресних зв'язків. Доступ до інфраструктури обмежується необхідністю реєстрації в екосистемі Steam та часто вимагає наявності ліцензії на конкретний

продукт. Функціонал редагування також позбавлений можливостей для побудови складних реляційних структур або візуальних баз даних.

У табл. 1.2 наведено всі ключові переваги та недоліки усіх наведених інформаційних систем.

Таблиця 1.2

Переваги та недоліки інформаційних систем

Платформа	Переваги	Недоліки
Fandom	Потужна вікі-структура; висока популярність серед геймерів; глибока система категорій.	Неможливість читати коментарі без реєстрації; агресивна реклама; повільне завантаження сторінок.
MediaWiki	Повна незалежність даних; відкритий код; надійна система контролю версій та захисту від вандалізму.	Складний поріг входу; застарілий дизайн; необхідність самостійного технічного обслуговування.
Notion	Сучасний блоковий інтерфейс; висока швидкість створення контенту; привабливий дизайн сторінок.	Відсутність інструментів для публічних обговорень; обмежені можливості для масштабування спільноти.

Платформа	Переваги	Недоліки
Reddit	Висока динаміка спілкування; швидка реакція спільноти на події; зручна система рейтингу постів.	Складно структурувати постійну інформацію (лор); гайди швидко губляться в загальній стрічці новин.
Steam Guides	Інтеграція з ігровим клієнтом; легкість створення посібників з ачівками та скріншотами.	Закрита екосистема; ізольованість контенту (неможливо створити цілісну вікі-мережу статей).

На основі проведеного аналізу існуючих рішень можна визначити стратегію розробки власної системи. Проектована інформаційна система має поєднати кращі риси існуючих платформ, одночасно виправивши їхні критичні недоліки.

Від Fandom та MediaWiki доцільно запозичити принцип ієрархічної організації даних. Система повинна підтримувати глибоку вкладеність категорій та зв'язність статей через перехресні посилання [24]. Водночас, замість складної вікі-розмітки, необхідно реалізувати інтуїтивно зрозумілий графічний інтерфейс, подібний до Notion, що дозволить будь-якому користувачу без спеціальних технічних навичок легко наповнювати базу знань.

Ключовою відмінністю проекту має стати політика відкритості та доступності. На відміну від Fandom, у розроблюваній системі буде реалізовано можливість перегляду обговорень та коментарів для незареєстрованих користувачів. Це знизить бар'єр входу для нових учасників спільноти та підвищить загальну привабливість ресурсу. Також архітектурно буде передбачено відсутність нав'язливого рекламного контенту, що суттєво покращить швидкість роботи інтерфейсу [8].

		Магаз А. О.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		17

Таким чином, ціллю проєкту є створення вебзастосунку, який володіє структурною потужністю професійної вікі-системи, але має легкість соціальної мережі з акцентом на зручність користувача та доступність інформації.

1.3 Вибір архітектури інформаційної системи онлайн-спільноти фанатів відеоігор

Під час проєктування сучасного веб-орієнтованого інформаційного середовища для спільноти фанатів відеоігор фундаментальним та критично важливим завданням є правильний вибір архітектурного шаблону. Архітектура програмного забезпечення безпосередньо визначає вектор розвитку проєкту, його стійкість до змін, здатність до горизонтального та вертикального масштабування, а також загальну вартість подальшої підтримки та додавання нового функціоналу. Зважаючи на специфіку інформаційної системи, яка передбачає високе навантаження на читання, динамічне оновлення контенту та необхідність надійного збереження реляційних зв'язків між сутностями (персонажами, локаціями, зброєю), було детально розглянуто декілька провідних архітектурних підходів.

Першим із проаналізованих підходів є монолітна архітектура. Вона класично визначається як модель розробки програмного забезпечення, де користувацький інтерфейс, бізнес-логіка, рівень доступу до даних та всі фонові процеси об'єднані в єдину неподільну кодову базу і виконуються як один цілісний процес у межах спільного простору пам'яті. Головною і беззаперечною перевагою такого підходу є відносна простота розробки на початкових етапах життєвого циклу проєкту. Моноліт легко тестувати локально, зручно розгортати (шляхом копіювання однієї збірки на сервер), а завдяки відсутності мережових викликів між внутрішніми компонентами системи він часто демонструє високу швидкість обробки транзакцій. Проте в контексті довгострокового розвитку монолітний підхід має низку суттєвих недоліків. Найбільш критичним є висока зв'язність компонентів. Коли система розростається, зміна в одному модулі може

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		18

непередбачувано вплинути на інші частини застосунку. Це ускладнює командну розробку, проведення ізольованого модульного тестування та масштабування системи. Будь-яке оновлення функціоналу вимагає повної перекомпіляції та перезапуску всього вебзастосунку, що підвищує ризики виникнення критичних простоїв.

Другим підходом, який набув значної популярності в сучасній ІТ-індустрії при створенні високонавантажених систем, є мікросервісна архітектура. Цей підхід кардинально відрізняється від моноліту і передбачає розбиття системи на набір слабо зв'язаних, вузькоспеціалізованих та повністю незалежних сервісів. Кожен мікросервіс відповідає за конкретну бізнес-доменну область (наприклад, окремий сервіс для авторизації, окремий для управління статтями, окремий для обробки коментарів) і взаємодіє з іншими через полегшені мережеві протоколи, такі як HTTP/REST або асинхронні брокери повідомлень (RabbitMQ, Apache Kafka). Хоча ця модель забезпечує виняткову можливість незалежного масштабування окремих модулів та дозволяє використовувати різні технологічні стеки, її імплементація супроводжується надмірною складністю інфраструктури (DevOps). Впровадження мікросервісів вимагає вирішення комплексних проблем розподілених транзакцій (відповідно до теореми CAP та патерну Saga), налаштування динамічного балансування навантаження, управління сервісними сітками та забезпечення консистентності даних. Для проєкту формату інформаційно-довідкової вікі-системи впровадження мікросервісного підходу є економічно невиправданим та технічно надлишковим, оскільки система має єдиний контекст бази знань.

Враховуючи проаналізовані недоліки монолітної (висока зв'язність) та мікросервісної (інфраструктурна перевантаженість) моделей, оптимальним та найбільш раціональним вибором для розроблюваної інформаційної системи стала архітектура Clean Architecture. Цей архітектурний шаблон, концептуалізований видатним інженером Робертом Мартіном (Robert C. Martin), спеціально створений для вирішення проблеми так званого «спагетті-коду» та побудови систем, які залишаються гнучкими протягом усього життєвого циклу.

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		19

Фундаментальною основою Clean Architecture є суворе дотримання принципу інверсії залежностей (Dependency Inversion Principle) із класичного набору принципів SOLID. Згідно з правилом залежностей (The Dependency Rule), усі залежності у вихідному коді повинні бути спрямовані виключно всередину – від зовнішніх механізмів (баз даних, фреймворків, інтерфейсів користувача) до центральних високорівневих політик (бізнес-правил). Відповідно до цього підходу, серверна частина розроблюваного додатка фізично та логічно розділена на чотири основні незалежні шари (проєкти), які взаємодіють між собою через чітко визначені інтерфейси.

Найглибшим рівнем системи, який лежить в її основі і є повністю автономним, виступає рівень предметної області (Domain Layer). Цей шар не містить жодних посилань на інші проєкти чи зовнішні пакети (наприклад, він принципово ізольований від Entity Framework чи ASP.NET Core). У ньому зосереджені фундаментальні сутності системи (Entities), такі як моделі користувачів, статей, коментарів, скарг. Також на цьому рівні визначаються об'єкти-значення (Value Objects), глобальні перелічення (Enums), абстрактні інтерфейси, які визначають контракти поведінки, та спеціалізовані класи виключень (Domain Exceptions). Завдяки такій ізоляції ядро системи залишається незмінним навіть у разі повної зміни технологічного стеку інфраструктури, що забезпечує довговічність бізнес-логіки.

Наступним рівнем, який обгортає предметну область, є рівень застосування (Application Layer). Тут зосереджена безпосередня бізнес-логіка розроблюваного продукту, яка специфічна саме для цього застосунку. Цей шар містить реалізацію сценаріїв використання (Use Cases) у вигляді сервісних класів. На рівні застосування також визначаються об'єкти передачі даних (Data Transfer Objects – DTO), які використовуються для структурування та оптимізації інформації, що передається між клієнтом та сервером, запобігаючи надмірному (Over-fetching) або недостатньому (Under-fetching) отриманню даних через мережу. Крім того, шар містить абстракції для доступу до даних (інтерфейси репозиторіїв), залишаючи деталі їхньої фізичної реалізації зовнішнім рівням.

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		20

Важливою складовою Application Layer є також механізми валідації бізнес-правил, які гарантують стовідсоткову коректність вхідних даних перед їх обробкою системою.

За практичну реалізацію абстрактних інтерфейсів, визначених у рівні застосування, відповідає рівень інфраструктури (Infrastructure Layer). Цей шар інкапсулює всю логіку взаємодії із зовнішнім світом та апаратним забезпеченням. Саме тут реалізується доступ до реляційної бази даних PostgreSQL за допомогою інструменту об'єктно-реляційного відображення (ORM) Entity Framework Core. Інфраструктурний шар містить контекст бази даних, класи-конфігурації для налаштування мапінгу сутностей за допомогою Fluent API, конкретні реалізації репозиторіїв.

Окремої уваги заслуговує реалізація доступу до даних на інфраструктурному рівні із застосуванням патерну Repository та Unit of Work. Репозиторій виступає як посередник між бізнес-логікою та базою даних, імітуючи об'єктно-орієнтовану колекцію сутностей у пам'яті. Це дозволяє приховати складні LINQ-запити та інфраструктурні особливості СУБД від сервісів застосунку. Патерн Unit of Work, у свою чергу, гарантує, що всі операції читання та запису в межах одного бізнес-сценарію виконуються в єдиній ACID-транзакції. Це унеможливорює часткове збереження даних у разі виникнення помилок на проміжних етапах виконання запиту. Infrastructure Layer також відповідає за інтеграцію зі сторонніми сервісами, такими як хмарне сховище для зображень, сервіс розсилки електронних листів та зовнішні API для автоматичного перекладу тексту.

Точкою входу в бекенд-систему виступає рівень представлення (API Layer), реалізований у вигляді RESTful API на базі контролерів. Його головним і єдиним завданням є прийом HTTP/HTTPS-запитів від клієнтського вебзастосунку, десеріалізація вхідних JSON-даних, виклик відповідних сервісів з Application Layer та формування стандартизованих HTTP-відповідей із правильними статус-кодами. На цьому рівні налаштовується конвеєр обробки запитів, реалізуються механізми CORS (Cross-Origin Resource Sharing), політики

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		21

кешування відповідей, глобальна обробка виключень (Global Exception Handling) та конфігуруються параметри безпеки, зокрема перевірка автентичності та авторизації на базі JWT-токенів.

Ефективна взаємодія всіх описаних шарів забезпечується завдяки вбудованому у фреймворк ASP.NET Core контейнеру впровадження залежностей (Dependency Injection Container). Контейнер автоматично управляє життєвим циклом об'єктів, реєструючи їх як Singleton (єдиний екземпляр для всієї програми), Scoped (один екземпляр на кожен HTTP-запит) або Transient (новий екземпляр при кожному зверненні). Використання шаблону Clean Architecture у поєднанні з DI-контейнером забезпечує високий рівень модульності. Код стає відкритим для розширення, але закритим для модифікації (Open-Closed Principle). Наприклад, у разі необхідності зміни провайдера бази даних, розробнику достатньо буде створити нову реалізацію на рівні Infrastructure Layer, не вносячи жодного рядка змін у бізнес-логіку Application Layer чи доменні моделі. Такий архітектурний підхід повністю відповідає найвищим стандартам розробки програмного забезпечення та гарантує, що розроблена інформаційна система онлайн-спільноти фанатів відеоігор буде стабільно функціонувати, легко підтримуватися та безпроблемно масштабуватися протягом багатьох років.

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Вибір інструментальних засобів для реалізації програмного комплексу є критичним етапом проєктування, що безпосередньо впливає на швидкість розробки, безпеку, подальшу вартість підтримки системи та загальний користувацький досвід. Для практичної реалізації серверної частини розроблюваної інформаційної системи було обрано платформу ASP.NET Core на базі сучасної об'єктно-орієнтованої мови програмування C# [1, 5]. Цей вибір обґрунтовується насамперед високою продуктивністю фреймворку, який

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

оптимізовано для роботи в хмарних середовищах, його можливістю розгортання на операційних системах Windows, Linux та macOS без зміни вихідного коду та потужними вбудованими механізмами впровадження залежностей (Dependency Injection) [13, 20]. Вбудований контейнер інверсії управління (IoC) дозволяє легко керувати життєвим циклом об'єктів, що ідеально поєднується з обраним патерном Clean Architecture [16] та забезпечує слабку зв'язність між архітектурними рівнями системи.

Екосистема .NET надає надійні інструменти для забезпечення багатовимірної безпеки. Зокрема, фреймворк має вбудовані підсистеми для налаштування авторизації та автентифікації на базі стандартів OAuth 2.0 та JSON Web Tokens (JWT) [7, 18], що гарантує захищений обмін даними між клієнтом та сервером без необхідності зберігання сесій у пам'яті сервера. Використання технології об'єктно-реляційного відображення Entity Framework Core [4] дозволяє максимально оптимізувати процес взаємодії серверної логіки з базою даних, забезпечуючи автоматичну генерацію безпечних SQL-запитів та надійний захист від поширених вразливостей, таких як SQL-ін'єкції.

В якості системи управління базами даних було обрано PostgreSQL [3]. Це надійна об'єктно-реляційна СУБД корпоративного рівня з відкритим вихідним кодом, яка відзначається високою стабільністю, повною відповідністю принципам ACID (Атомарність, Узгодженість, Ізольованість, Довговічність) та здатністю ефективно обробляти складні зв'язки між мільйонами записів. Оскільки специфіка розроблюваної вікі-системи передбачає наявність глибокої ієрархії категорій, великої кількості статей із перехресними посиланнями, метаданих та розгалуженої системи коментарів користувачів, вибір PostgreSQL гарантує абсолютне збереження цілісності інформації. Наявність просунутих алгоритмів побудови індексів та підтримка роботи з неструктурованими даними забезпечує високу швидкість виконання агрегованих пошукових запитів [15].

Для розробки клієнтської частини оптимальним інженерним рішенням стало використання JavaScript-бібліотеки React [2]. Її компонентний підхід дозволяє створювати незалежні, інкапсульовані та перевикористовувані

		Магаз А. О.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

елементи інтерфейсу користувача (компоненти навігації, форми автентифікації, списки статей), що суттєво пришвидшує процес розробки та полегшує подальшу підтримку великої кодової бази [14]. Завдяки інноваційному механізму віртуального DOM, React оптимізує процес оновлення браузерного інтерфейсу. Замість повного перемальовування сторінки, бібліотека обчислює мінімально необхідну кількість змін та точково застосовує їх, що забезпечує миттєве оновлення даних на екрані без необхідності повного перезавантаження сторінки клієнтом [25]. Це є критично важливим фактором для забезпечення плавного та комфортного користувацького досвіду під час навігації великими обсягами довідкового ігрового контенту.

Системні вимоги: Для забезпечення стабільної, безперебійної та швидкої роботи вебзастосунку серед широкого кола користувачів було визначено та систематизовано базові технічні характеристики апаратного забезпечення. Платформа для веб-орієнтованої системи онлайн-спільноти спроектована таким чином, щоб ефективно функціонувати навіть в умовах обмежених ресурсів клієнтської сторони. З метою охоплення максимальної аудиторії та гарантування комфортного рівня зручності використання [24], система повинна повністю задовольняти встановленим вимогам до роботи на пристроях із чітко визначеною мінімальною апаратною комплектацією.

Ці експлуатаційні вимоги були детально проаналізовані та класифіковані. Як видно з таблиці 1.3, для забезпечення стабільного рендерингу клієнтської частини на десктопних персональних комп'ютерах користувачам необхідно мати базову сучасну конфігурацію, що включає підтримку актуальних версій браузерів.

Таблиця 1.3

Вимоги для десктопних пристроїв

Характеристика					Мінімальні вимоги		
Операційна система					Windows 10 і вище, macOS 10.14 і вище, сучасні дистрибутиви Linux		
Процесор					Двоядерний з тактовою частотою від 1.6 ГГц		
		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ		
		Єфремов Ю. М.					
Змн.	Арк.	№ докум.	Підпис	Дата			

Характеристика	Мінімальні вимоги
Оперативна пам'ять	Мінімум 4 ГБ
Браузер	Останні версії Google Chrome, Mozilla Firefox, Safari, Microsoft Edge[6]

З огляду на глобальну тенденцію зростання частки мобільного трафіку, розроблений інформаційний ресурс має повністю адаптивний дизайн. У таблиці 1.4 наведено мінімальні технічні характеристики для смартфонів та планшетів, які гарантують відсутність затримок під час скролінгу об'ємних статей та взаємодії зі складними елементами інтерфейсу.

Таблиця 1.4

Вимоги для мобільних пристроїв

Характеристика	Мінімальні вимоги
Операційна система	Android 8.0 і вище, iOS 12 і вище
Оперативна пам'ять	Мінімум 2 ГБ
Браузер	Сучасні мобільні браузери з підтримкою HTML5 та CSS3[6]

Окрім апаратних потужностей пристрою, критичним фактором для функціонування будь-якої клієнт-серверної системи є стабільність каналу передачі даних та налаштування локального середовища виконання. Ці параметри, які безпосередньо впливають на процес отримання JSON-відповідей

від серверного API та збереження локальних сесій автентифікації, зведено у таблиці 1.5.

Таблиця 1.5

Мережеві та додаткові вимоги

Категорія	Опис вимог
Мережеві вимоги	Інтернет-з'єднання зі швидкістю від 2 Мбіт/с[8]
Інші вимоги	Ввімкнений JavaScript для коректної роботи інтерактивних елементів[17]
	Підтримка Cookies/Local Storage для збереження сесій користувачів та обраної мови[6]

Такий комплексний підхід до визначення апаратних та програмних лімітів гарантує коректний рендеринг клієнтської частини та швидкий обмін даними із сервером незалежно від обраної користувачем платформи.

Висновки до першого розділу

Досліджено актуальність створення спеціалізованих вікі-систем, що дозволяють систематизувати великі обсяги інформації з ігрових всесвітів. Визначено, що ключовими потребами користувачів є зручна рубрикація контенту та розвинена система комунікації.

Проведено порівняльний аналіз існуючих аналогів. Виявлено їхні основні недоліки, серед яких перевантаженість рекламою, закритість коментарів для неавторизованих користувачів та складність інтерфейсів. Це підтвердило доцільність розробки нової, більш оптимізованої та відкритої системи.

Сформовано детальний перелік функціональних та нефункціональних вимог до програмного продукту, визначено ролі користувачів та базові сценарії їхньої взаємодії з системою.

Здійснено вибір та обґрунтування технологічного стеку і архітектури. Як основу обрано шаблон Clean Architecture, що забезпечить високу масштабованість та легкість підтримки коду. Для реалізації серверної логіки

обрано платформу ASP.NET Core спільно з ORM Entity Framework Core та БД PostgreSQL. Для клієнтської частини визначено використання бібліотеки React.

		Магаз А. О.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		27

РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОНЛАЙН-СПІЛЬНОТИ ФАНАТІВ ВІДЕОІГОР

2.1 Вимоги до системи

Інформаційна система онлайн-спільноти фанатів відеоігор надає веб-інтерфейс, через який користувачі можуть шукати та переглядати інформацію про ігровий всесвіт, взаємодіяти між собою, а також брати участь у наповненні порталу контентом. Функціональність платформи залежить від рівня доступу та активності користувачів.

Усіх користувачів системи доцільно розділити на три основні групи, кожна з яких наділена відповідним рівнем прав та доступу до функціоналу. Першу групу становлять незареєстровані користувачі, які взаємодіють із платформою виключно для пошуку та ознайомлення з розміщеною інформацією, використовуючи лише базові ресурси середовища. Другою групою є зареєстровані користувачі, яким надається ширший спектр можливостей, зокрема доступ до інтерактивного функціоналу платформи. Ця категорія має змогу брати участь у житті спільноти через публікацію коментарів, залишення реакцій під статтями та внесення власних пропозицій щодо наповнення бази знань. Третя група представлена адміністраторами, які мають повні права на комплексне управління всією технічною частиною системи, модерацію контенту та контроль за діями інших користувачів.

Проектування інформаційної системи передбачає створення архітектури та детальне планування функціональних і нефункціональних вимог [23]. Для наглядної взаємодії між акторами та системою було розроблено Діаграму варіантів використання (рис. 2.1).

Відповідно до розробленої діаграми, кожен тип користувача наділений специфічним набором прецедентів, які визначають межі його взаємодії з платформою.

		Магаз А. О.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

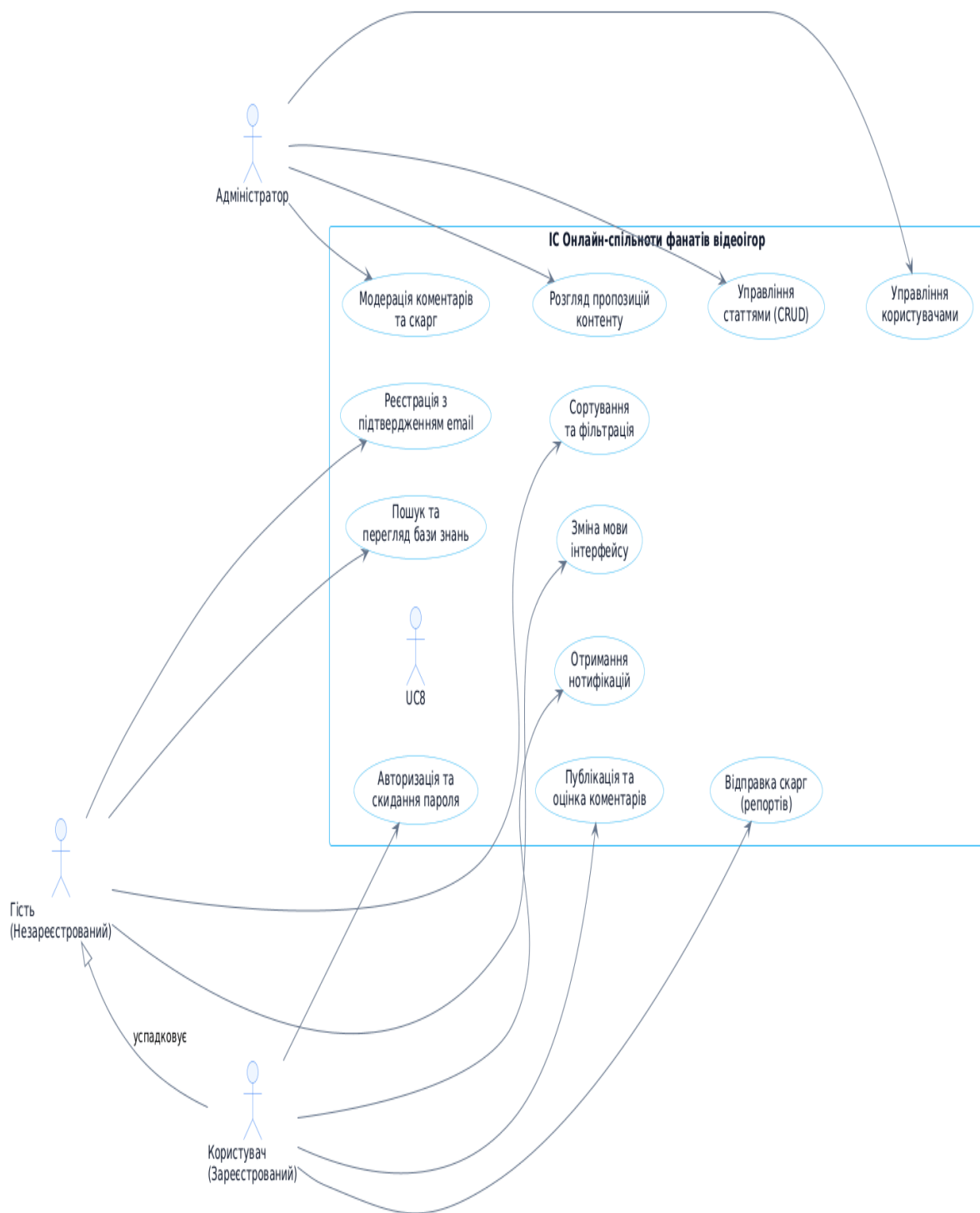


Рис. 2.1 Діаграма використання системи

У табл. 2.1 наведено опис основних прецедентів діаграми варіантів використання.

		Магаз А. О.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		29

Основні прецеденти варіантів використання інформаційної системи

ID	Назва варіанту використання	Актор	Опис	Кроки
1	Реєстрація користувача	Користувач	Створення нового облікового запису із підтвердженням пошти	Відкриває сторінку реєстрації та вводить необхідні дані
				Вводить 6-значний код підтвердження, надісланий системою на email користувача
				Система створює обліковий запис
2	Авторизація та відновлення доступу	Користувач	Вхід у систему або скидання пароля	Відкриває сторінку входу
				Вводить логін/пароль або обирає "Забули пароль"

продовж. табл. 2.1

				Для скидання: отримує код на пошту та встановлює новий
				Система надає доступ до кабінету
3	Перегляд та сортування бази знань	Гість, Користувач	Пошук інформації про персонажів, зброю, місії	Відкриває каталог бази знань
				Використовує пошук або застосовує сортування
				Переглядає детальну сторінку об'єкта
4	Взаємодія з контентом	Користувач	Залишення коментарів та реакцій	Відкриває сторінку об'єкта
				Пише коментар або ставить реакцію на коментарі інших

продовж. табл. 2.1

				Відправляє репорт на коментар, що порушує правила
5	Надсилання репортів	Користувач	Формування запиту на скаргу або пропозицію	Натискає "Повідомити про помилку"
				Заповнює спеціальне поле
				Відправляє на розгляд адміністратору
				Отримує нотифікацію про зміну статусу заявки
6	Модерація спільноти	Адміністратор	Обробка репортів та видалення порушень	Входить у панель адміністратора
				Переглядає список репортів від користувачів
				Видаляє спам або хейт- коментарі

7	Управління контентом	Адміністратор	Повний контроль над базою знань	Входить у панель адміністратора
				Розглядає пропозиції користувачів
				Додає, редагує або видаляє статті на сайті
8	Налаштування локалізації	Усі користувачі	Зміна мови інтерфейсу	Натискає на перемикач мов
				Обирає потрібну мову
				Система перекладає вміст сторінки

Основна мета розробки даної системи полягає у створенні зручного, структурованого та надійного ресурсу для фанатської спільноти, який забезпечує швидкий доступ до інформації та зручні інструменти комунікації [24].

Для досягнення цієї мети архітектура платформи була спроектована з урахуванням сучасних стандартів веброзробки, що гарантує високу продуктивність та безперебійну роботу навіть при значних навантаженнях. Особливу увагу було приділено системі розмежування прав доступу, яка дозволяє ефективно управляти контентом та забезпечувати надійну безпеку даних на всіх рівнях взаємодії з вебзастосунком [18]. Зважаючи на ці

архітектурні та системні особливості, було сформовано наступний перелік функціональних (табл. 2.2) та нефункціональних (табл. 2.3) вимог.

Таблиця 2.2

Функціональні вимоги

Рівень доступу	Перелік можливостей
Незареєстрований користувач	Пошук та перегляд інформації про ігровий всесвіт.
	Сортування списків.
	Перемикання мови інтерфейсу.
	Створення нового облікового запису з підтвердженням електронної пошти (6-значний код).
Зареєстрований користувач	Усі функції Незареєстрованого користувача
	Авторизація в системі та відновлення пароля через код на email.
	Публікація коментарів під статтями.
	Можливість ставити реакції.
	Відправка репортів на порушення правил у коментарях та неточності у статтях.
	Отримання системних нотифікацій про статус змін та відповіді.
Адміністратор	Додавання, редагування, видалення статей.
	Розгляд пропозицій від користувачів щодо зміни контенту
	Управління користувачами
	Перегляд скарг від користувачів.
	Видалення коментарів зі спамом, ненормативною лексикою чи мовою ворожнечі.

Нефункціональні вимоги

Категорія	Опис вимог
Навчання та зручність	Інтуїтивно зрозумілий інтерфейс, що не вимагає додаткової підготовки.
	Час відповіді системи для базових запитів: до 1-2 секунд.
	Загальна надійність та стійкість до збоїв.
Доступність та безперебійність	Час технічного обслуговування: не більше 5% від загального часу роботи.
	Максимальна норма помилок: не більше 1 помилки на 1000 запитів.
Продуктивність та масштабованість	Підтримка не менше 5000 одночасно активних користувачів без втрати швидкодії.
	Архітектура бази даних має підтримувати швидке масштабування при збільшенні обсягу даних.

2.2 Проєктування бази даних

Розробка бази даних для інформаційної системи онлайн-спільноти забезпечує зберігання, обробку та керування даними, необхідними для функціонування порталу. Оскільки система побудована з використанням Entity Framework Core [4] та підсистеми управління доступом ASP.NET Core Identity [18], архітектура бази даних включає як таблиці бізнес-логіки, так і системні таблиці для авторизації.

Реалізована реляційна база даних PostgreSQL [3] має назву FanWikiDb. У табл. 2.4 наведено список таблиць бази даних інформаційної системи.

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		35

Список таблиць бази даних інформаційної системи

Назва таблиці	Призначення
AspNetUsers	Інформація про облікові записи зареєстрованих користувачів
AspNetRoles	Довідник ролей користувачів у системі
Articles	Зберігання нелокалізованої інформації про статті
ArticleTranslations	Локалізований текст статей
Comments	Зберігання коментарів та ієрархії відповідей
CommentReactions	Реакції користувачів на коментарі
Reports	Скарги на порушення правил спільноти
AspNetUserRoles	Зв'язуюча таблиця між користувачами та їхніми ролями

Структура бази даних зображена на рис. 2.2:

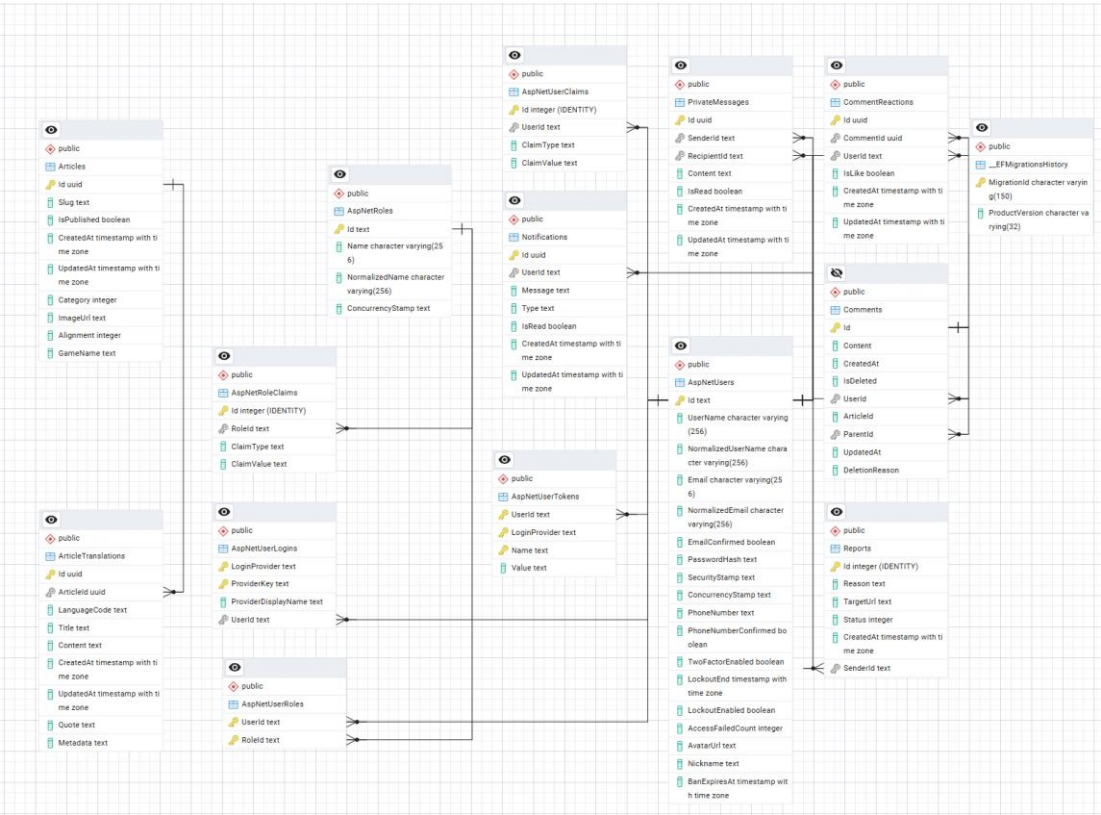


Рис. 2.2 Структура бази даних

Розглянуто основні таблиці бази даних:AspNetUsers (табл. 2.5), Articles (табл. 2.6,) ArticleTranslations (табл. 2.7), Comments (табл. 2.8), CommentReactions (табл. 2.9) та Reports (табл. 2.10).

Таблиця AspNetUsers призначена для зберігання облікових записів. Вона згенерована підсистемою Identity та розширена кастомними полями для потреб спільноти: аватар, нікнейм, статус блокування.

Таблиця 2.5

Структура таблиці AspNetUsers

Назва	Тип даних	ПК	ЗК	Опис поля
Id	text	+	-	Ідентифікатор користувача
UserName	varchar(256)	-	-	Логін користувача
Email	varchar(256)	-	-	Електронна пошта
PasswordHash	text	-	-	Хеш пароля
Nickname	text	-	-	Відображуване ім'я на порталі
AvatarUrl	text	-	-	Посилання на зображення профілю
BanExpiresAt	timestamp	-	-	Час закінчення блокування

Таблиця Articles є основною сутністю бази знань і зберігає загальні параметри вікі-сторінок, які не залежать від мови відображення.

Таблиця 2.6

Структура таблиці Articles

Назва	Тип даних	ПК	ЗК	Опис поля
Id	uuid	+	-	Унікальний ідентифікатор статті
Slug	text	-	-	URL-ідентифікатор сторінки
IsPublished	boolean	-	-	Статус публікації
Category	integer	-	-	Категорія контенту
ImageUrl	text	-	-	Посилання на головне зображення
CreatedAt	timestamp	-	-	Дата та час створення

Таблиця ArticleTranslations забезпечує виконання вимоги щодо перекладу сторінок на інші мови. Вона зберігає безпосередній текстовий контент для кожної мовної версії.

Таблиця 2.7

Структура таблиці ArticleTranslations

Назва	Тип даних	ПК	ЗК	Опис поля
Id	uuid	+	-	Ідентифікатор локалізації
ArticleId	uuid	-	+	Ідентифікатор оригінальної статті
LanguageCode	text	-	-	Код мови
Title	Text	-	-	Заголовок відповідною мовою
Content	text	-	-	Основний текстовий вміст

Таблиця Comments призначена для забезпечення взаємодії між користувачами шляхом публікації відгуків. Завдяки полю ParentId підтримується деревоподібна структура відповідей.

Таблиця 2.8

Структура таблиці Comments

Назва	Тип даних	ПК	ЗК	Опис поля
Id	uuid	+	-	Ідентифікатор коментаря
ArticleId	uuid	-	+	Ідентифікатор статті
UserId	text	-	+	Ідентифікатор автора
Content	text	-	-	Текст коментаря
ParentId	uuid	-	+	ID батьківського коментаря (для відповідей)
IsDeleted	boolean	-	-	Статус

Таблиця CommentReactions фіксує оцінки, які користувачі ставлять коментарям інших учасників.

Таблиця 2.9

Структура таблиці CommentReactions

Назва	Тип даних	ПК	ЗК	Опис поля
Id	uuid	+	-	Ідентифікатор реакції
CommentId	uuid	-	+	Ідентифікатор оціненого коментаря
UserId	text	-	+	Ідентифікатор користувача
IsLike	boolean	-	-	Тип реакції

Таблиця Reports збирає скарги користувачів на небажаний контент для подальшого розгляду адміністратором.

Таблиця 2.10

Структура таблиці Reports

Назва	Тип даних	ПК	ЗК	Опис поля
Id	integer	+	-	Ідентифікатор скарги
SenderId	text	-	+	Ідентифікатор ініціатора скарги
TargetUrl	text	-	-	Посилання на об'єкт скарги
Reason	text	-	-	Причина
Status	integer	-	-	Статус обробки заявки

Реалізована база даних у повній мірі відповідає вимогам до реляційної моделі та забезпечує збереження, цілісність та доступ до інформації [15]. Чітке розділення системних та бізнес-даних дозволяє легко масштабувати проєкт у майбутньому.

2.3 Опис узагальнених алгоритмів та взаємодії елементів системи

Функціонування розробленої інформаційної системи онлайн-спільноти фанатів відеоігор базується на подієво-орієнтованій та клієнт-серверній взаємодії між односторінковим вебзастосунком (SPA) та розподіленим сервером додатків. Узагальнений алгоритм роботи платформи передбачає безперервне

		Магаз А. О.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		39

опрацювання вхідних HTTP-запитів від користувачів, їх сувору багатоступеневу валідацію на рівні бізнес-логіки (Application Layer) та безпечне, транзакційне збереження актуальних станів у реляційній базі даних під керуванням СУБД PostgreSQL [3, 26].

Для повного та вичерпного розкриття внутрішніх принципів роботи розробленого програмного комплексу необхідно детально проаналізувати кілька ключових системних алгоритмів, що забезпечують життєдіяльність веб-середовища: алгоритм автентифікації та розмежування прав доступу на основі криптографічних токенів, алгоритм динамічного формування контенту бази знань з оптимізацією запитів, а також репрезентативний алгоритм модерації контенту на основі скарг користувачів.

Перший базовий алгоритм визначає життєвий цикл сесії користувача та безпеку міжсерверної взаємодії. Оскільки архітектура серверної частини проектування побудована за принципом відсутності збереження стану сесії (Stateless), кожен запит до захищених кінцевих точок API повинен містити верифіковані дані про відправника. Процес ініціалізації взаємодії починається на стороні клієнтського застосунку React, де користувач вводить облікові дані (логін та пароль) у формі автентифікації. Клієнтська частина формує асинхронний POST-запит, який передається на сервер. Серверний рівень представлення (API Layer) делегує перевірку підсистемі безпеки. Алгоритм обробки на стороні бекенду виконує хешування отриманого пароля та його порівняння із збереженим бінарним хешем у базі даних PostgreSQL. У разі успішного збігу, спеціалізований сервіс генерації токенів створює об'єкт JSON Web Token (JWT), вшиваючи в його корисне навантаження (Claims) унікальний ідентифікатор користувача (ID), його електронну адресу та перелік присвоєних ролей (наприклад, «User», «Moderator», «Administrator») [7]. Токен підписується секретним симетричним ключем за допомогою алгоритму HMAC-SHA256 і повертається клієнту. Отриманий токен зберігається у локальному сховищі браузера (Local Storage) і автоматично додається HTTP-клієнтом Axios до заголовка Authorization: Bearer <token> для всіх наступних запитів, що

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		40

унеможливилює необхідність повторного введення пароля під час поточної сесії [25].

Другим важливим архітектурним алгоритмом є процес модерації контенту на основі скарг користувачів, який є найбільш репрезентативним для демонстрації взаємодії між усіма шарами Clean Architecture. Цей багатокроковий процес органічно об'єднує інтерактивну роботу зареєстрованого учасника спільноти, автоматизовані безпекові перевірки на стороні сервера та управлінські дії адміністратора платформи.

Процес взаємодії елементів системи під час надсилання скарги розпочинається з того, що авторизований користувач під час навігації сторінками інформаційного ресурсу виявляє коментар або статтю, яка порушує встановлені правила онлайн-спільноти, та ініціює його оскарження за допомогою графічного інтерфейсу. Реагуючи на дію користувача, React-компонент змінює свій локальний стан і відкриває модальне вікно, де відображається перелік чітко визначених причин оскарження (наприклад, спам, нецензурна лексика, образи, спойлери). Після вибору причини та підтвердження дії клієнтський додаток активує асинхронний метод, який формує HTTP-запит типу POST на адресу `/api/reports`. Тіло запиту містить унікальний ідентифікатор (GUID) цільового об'єкта, обраний код причини та необов'язковий текстовий коментар користувача.

Отримавши HTTP-запит, серверний рівень представлення пропускає його крізь конвеєр проміжного програмного забезпечення (Middleware Pipeline). На першому кроці конвеєра алгоритм валідації перевіряє криптографічний підпис токена, переконуючись, що він виданий саме цим сервером і не був модифікований злоумисниками. Далі запит передається на рівень застосування (Application Layer), де активований обробник команди (Command Handler) ініціює перевірку бізнес-правил за допомогою вбудованого спам-фільтра. Алгоритм фільтрації виконує швидкий запит до бази даних, щоб переконатися, що поточний користувач не надсилав дублюючу скаргу на цей самий об'єкт

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

раніше, захищаючи систему від штучного накручування скарг (DoS-атак на рівні бізнес-логіки).

Після успішного проходження всіх етапів валідації система активує рівень інфраструктури (Infrastructure Layer) через інтерфейс репозиторію. У межах поточної транзакції створюється новий запис у таблиці Reports бази даних PostgreSQL, якому автоматично присвоюється початковий статус очікування розгляду (ReportStatus.Pending), а також фіксується точний час створення (Timestamp). Сервер повертає клієнту статус-код 201 Created, а інтерфейс користувача плавно закриває модальне вікно та виводить сповіщення про успішне прийняття скарги.

На фінальному етапі цього алгоритму в дію вступає адміністратор або модератор платформи. Через спеціалізований захищений інтерфейс адміністративної панелі (Admin Dashboard) виконується GET-запит на отримання списку активних скарг. Сервер перевіряє наявність у JWT-токені відправника відповідних адміністративних прав. Після аналізу суті порушення адміністратор приймає управлінське рішення: відхилити скаргу як безпідставну або задовольнити її. Якщо приймається рішення про видалення порушення, клієнт надсилає запит на видалення об'єкта. Сервер виконує процедуру «м'якого видалення» (Soft Delete) для вказаного контенту, встановлюючи прапорець IsDeleted у значення true в таблиці бази даних, та автоматично закриває всі пов'язані скарги, оновлюючи їхній статус на ReportStatus.Resolved.

Запропонований алгоритм демонструє високу інженерну ефективність завдяки застосуванню принципів асинхронної обробки даних та ізоляції шарів логіки. Використання JWT-токенів на етапі ініціації запиту гарантує, що система обробляє звернення виключно від верифікованих джерел, що повністю унеможливорює несанкціоноване втручання злоумисників у процес модерації. Підхід із «м'яким видаленням» відіграє критично важливу роль у збереженні цілісності реляційної бази даних. Замість фізичного знищення записів з таблиць (що могло б призвести до каскадного порушення зв'язків з іншими сутностями або помилок цілісності), зміна логічного статусу об'єкта дозволяє

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		42

адміністраторам за необхідності проводити ретроспективний аналіз інцидентів, збирати статистику порушень та миттєво відновлювати помилково прихований контент без втрати даних. Подієво-орієнтована архітектура цього процесу також оптимізує загальне навантаження на сервер, оскільки важкі операції з оновлення статусів виконуються швидко і не блокують основний потік взаємодії користувача з клієнтським інтерфейсом спільноти.

Для детальної візуалізації часової послідовності передачі повідомлень, викликів методів та обміну даними між об'єктами системи під час описаного процесу модерації контенту було розроблено UML-діаграму послідовності (рис. 2.3). Вона наочно демонструє шлях запиту від ініціації на фронтенді до фізичного збереження змін у СУБД PostgreSQL.

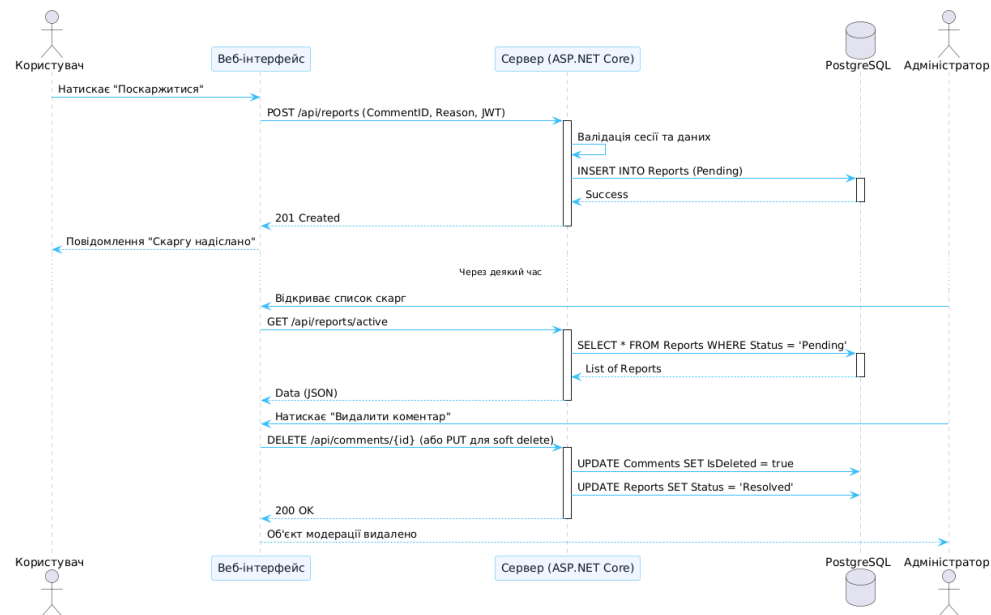


Рис. 2.3. Діаграма послідовності процесу модерації на основі скарг

2.4 Реалізація програмного комплексу

Процес безпосередньої програмної реалізації інформаційної системи онлайн-спільноти фанатів відеоігор базується на інтеграції передових технологій веб-розробки, що утворюють потужний та оптимізований стек. Програмний комплекс спроектовано та імплементовано у вигляді сучасного клієнт-серверного вебзастосунку з чітким та жорстким розділенням зон

відповідальності між клієнтською та серверною частинами, які взаємодіють між собою виключно за допомогою стандартизованого інтерфейсу прикладного програмування (API). Таке архітектурне рішення забезпечує незалежність життєвих циклів розробки обох частин системи та дозволяє у майбутньому легко інтегрувати додаткові клієнтські додатки, наприклад, нативні мобільні застосунки для платформ iOS та Android.

Серверну частину розроблено з використанням високопродуктивної, кросплатформної платформи ASP.NET Core на базі сучасної об'єктно-орієнтованої мови програмування C# [1, 5]. Відповідно до обраного патерну Clean Architecture (Чистої архітектури), який є де-факто галузевим стандартом для проектування гнучких та масштабованих систем [11, 22], вихідний код бекенду логічно та фізично структуровано на чотири ізольовані проекти. Залежності між цими проектами жорстко контролюються на рівні компіляції платформи .NET. Найглибшим і повністю автономним рівнем виступає шар предметної області (Domain Layer), де імплементовано чисті сутності бази даних. Тут визначено класи моделей користувачів (ApplicationUser), структури вікі-статей (Article, ArticleTranslations), категорій, скарг (Report) та ієрархічних коментарів (Comment). Ці класи інкапсулюють фундаментальні правила бізнесу, перелічення статусів та кастомні виняткові ситуації домену.

Над предметною областю розгортається рівень застосування (Application Layer), який виступає координатором бізнес-процесів. Цей шар містить абстрактні інтерфейси для взаємодії з базою даних, об'єкти передачі даних (DTO) для безпечного обміну інформацією по мережі, а також сервісні класи бізнес-логіки. Для забезпечення валідності вхідних даних на цьому рівні імплементовано правила перевірки за допомогою потужних інструментів валідації, які гарантують, що до бази даних потрапить виключно перевірена та безпечна інформація. За зв'язок із зовнішнім світом відповідає рівень інфраструктури (Infrastructure Layer), де міститься конкретна реалізація контексту реляційної бази даних ApplicationDbContext, розширена конфігурація сутностей за допомогою Fluent API, імплементация класів репозиторіїв та

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		44

налаштування інтеграції із зовнішніми сервісами, такими як хмарне сховище мультимедіа та сервіси автоматичного машинного перекладу текстів. Точкою входу в бекенд-систему є рівень представлення (API Layer), який реалізує набір REST-контролерів, відповідає за прийом вхідних HTTP-запитів, конфігурацію конвеєра проміжного програмного забезпечення (Middleware), обробку політик CORS для захисту від міжсайтових атак та формування структурованих JSON-відповідей.

Взаємодія бекенду з реляційною системою управління базами даних PostgreSQL реалізована через використання потужного фреймворку об'єктно-реляційного відображення (ORM) Entity Framework Core [4]. Для стандартизації всіх операцій читання, створення, оновлення та видалення (CRUD), а також для повної архітектурної ізоляції рівня бізнес-логіки від специфіки SQL-синтаксису конкретної СУБД, у проєкті успішно застосовано фундаментальний патерн проєктування Repository Pattern (Репозиторій) спільно з патерном Unit of Work [21]. Це дозволило глибоко інкапсулювати складні та багатопотокові LINQ-запити всередині інфраструктурного шару бази даних. Використання Unit of Work зі свого боку гарантує, що всі операції, які виконуються в межах одного багатоетапного бізнес-сценарію (наприклад, створення статті та одночасний запис її локалізацій), виконуються строго у межах єдиної транзакції бази даних відповідно до принципів ACID. Будь-які зміни в архітектурі та структурі бази даних контролюються виключно за допомогою вбудованого механізму міграцій (EF Core Migrations), що гарантує надійну синхронізацію кодової бази та фізичної схеми таблиць на серверах розгортання без необхідності ручного написання DDL-скриптів.

Клієнтську частину інформаційної системи реалізовано у вигляді сучасного односторінкового застосунку (Single Page Application - SPA) на базі популярної компонентної JavaScript/TypeScript бібліотеки React [2]. Використання функціональних компонентів у поєднанні з системою React Hooks, зокрема useState для управління локальними станами, useEffect для побічних ефектів та обробки життєвого циклу, а також створення кастомних

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		45

хуків для абстрагування мережових запитів, дозволило розробити високореактивний графічний інтерфейс користувача. Такий інтерфейс здатний миттєво реагувати на будь-які дії гравця без візуально помітних затримок та повного перезавантаження веб-сторінки [14, 25]. Для забезпечення високої модульності та підтримки коду кожен логічний елемент інтерфейсу, включаючи інтерактивні картки статей, ієрархічні блоки коментарів, компоненти глобальної навігації, інфобокси та форми подачі скарг, було винесено в окремі ізольовані файли-компоненти. Усі висхідні та низхідні HTTP-запити до серверного API виконуються повністю асинхронно у фоновому режимі за допомогою HTTP-клієнта Axios. Отримані JSON-дані динамічно підвантажуються у відповідні компоненти сторінки бази знань за допомогою оптимізованого алгоритму узгодження (Reconciliation) та механізму Virtual DOM, що повністю виключає необхідність перезавантаження сторінки браузера під час навігації багатосторінковим каталогом ігрового контенту [25].

Важливою особливістю програмної реалізації клієнтської частини є інтеграція потужного WYSIWYG-редактора на базі бібліотеки Tiptap, який використовується в панелі адміністратора для форматування тексту статей бази знань. Цей інструмент дозволяє модераторам застосовувати складне семантичне форматування, додавати заголовки, списки та вбудовувати медіаконтент безпосередньо у вікні браузера, генеруючи на виході чистий та безпечний HTML-код, який згодом рендериться на публічних сторінках проєкту.

Одним із ключових та найбільш критичних аспектів програмної реалізації є забезпечення багаторівневої безпеки даних та строге розмежування прав доступу між різними групами користувачів системи (гості, зареєстровані гравці, модератори, адміністратори). Для вирішення цього завдання в ядро системи інтегровано сучасний механізм криптографічної авторизації на базі токенів стандарту JWT (JSON Web Token) спільно з вбудованою підсистемою управління ідентифікацією ASP.NET Core Identity [7, 18]. Реалізацію алгоритму контролю доступу із використанням рольової моделі (Role-Based Access Control) детально опрацьовано на рівні контролерів. Наприклад, під час обробки HTTP-

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

запиту на створення скарги щодо порушення правил платформи, ідентифікатор відправника задля підвищеної безпеки принципово не приймається з клієнтської частини у тілі запиту. Натомість бекенд самостійно витягує унікальний ідентифікатор користувача безпосередньо з криптографічно підписаного JWT-токена, який передається у заголовок авторизації, за допомогою вбудованих засобів фреймворку та властивостей поточного контексту HttpContext.User. Це повністю виключає можливість підміни даних (ID Spoofing) з боку злоумисників. Для методів отримання, редагування та модерації чутливих даних застосовано жорстке обмеження доступу за допомогою атрибутів авторизації [Authorize(Roles = "Admin")]. Завдяки цьому система автоматично перевіряє наявність ролі адміністратора або модератора на етапі проходження конвеєра Middleware і миттєво блокує несанкціоновані запити від звичайних користувачів, повертаючи стандартизований статус HTTP 403 Forbidden [26].

Окрім базових механізмів обміну даними, для забезпечення комунікації в режимі реального часу (Real-time communication) програмний комплекс використовує технологію WebSockets через бібліотеку SignalR, інтегровану в ASP.NET Core. Відповідна конфігурація маршрутизації (app.MapHub<ChatHub>("/hubs/chat")) розгортає двосторонній постійний канал зв'язку між клієнтським застосунком та сервером. Це дозволяє реалізувати миттєву доставку системних сповіщень користувачам, наприклад, при зміні статусу їхньої скарги або при появі нових коментарів під їхніми статтями, що значно підвищує загальний рівень залученості та інтерактивності спільноти.

З метою максимальної оптимізації продуктивності системи та економії апаратних ресурсів сервера, під час формування великих масивів даних (наприклад, списку всіх активних скарг для панелі адміністратора) інтенсивно використовується механізм проєкції даних засобами мови інтегрованих запитів LINQ за допомогою оператора Select [13]. Ця технологія дозволяє створювати легкі анонімні об'єкти (DTO) безпосередньо на етапі трансляції запиту до бази даних PostgreSQL. Як наслідок, з фізичних таблиць витягується виключно та колонка інформації, яка безпосередньо необхідна для відображення в графічній

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		47

таблиці на стороні клієнта. Це дозволяє уникнути ресурсомісткого завантаження важких пов'язаних сутностей через механізм жадібного завантаження (Eager Loading). Такий архітектурний підхід суттєво зменшує використання оперативної пам'яті бекенд-сервера, оптимізує роботу індексів системи управління базами даних та багаторазово знижує загальне навантаження на мережу під час передачі інформації. Додатково, для автоматизації процесу тестування, відлагодження та самодокументування розроблених кінцевих точок API, у проєкт було успішно інтегровано популярний інструмент Swagger UI [10], що значно спростило та пришвидшило процес налагодження асинхронної взаємодії між клієнтом та сервером на етапі розробки програмного комплексу.

Висновки до другого розділу

Було здійснено комплексне проєктування архітектури онлайн-спільноти для фанатів відеоігор.

На базі сформованих функціональних вимог розроблено структурну модель проєкту та визначено ключові прецеденти взаємодії користувачів із платформою. Для забезпечення збереження даних спроектовано реляційну базу даних. Окрему увагу приділено алгоритмізації головних бізнес-процесів, зокрема механізмам модерації.

Також наведено детальний опис програмної реалізації базового функціоналу: реєстрації, авторизації, обробки користувацьких звернень та управління статтями на боці сервера.

		Магаз А. О.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		48

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ІНФОРМАЦІЙНОЮ СИСТЕМОЮ ОНЛАЙН-СПІЛЬНОТИ ФАНАТІВ ВІДЕОІГОР

3.1 Порядок встановлення та налаштування параметрів системи

Процес розгортання та налаштування параметрів розробленого програмного комплексу є фундаментальним багатоетапним завданням, яке вимагає чіткої послідовності дій для забезпечення стабільної, безпечної та високопродуктивної роботи інформаційної системи онлайн-спільноти. Загальна процедура встановлення складається з трьох ключових етапів: ініціалізації та конфігурації серверного середовища, проектування та розгортання реляційної бази даних, а також компіляції, оптимізації та публікації клієнтського вебзастосунку. Архітектурно серверна частина реалізована на базі сучасного кросплатформного фреймворку ASP.NET Core [1], що забезпечує гнучкість розгортання незалежно від цільової операційної системи, а в якості основної СУБД використовується надійна об'єктно-реляційна система PostgreSQL [3].

Першочерговим завданням під час локального розгортання або підготовки робочого сервера є інсталяція ядра PostgreSQL та створення порожньої бази даних, яка слугуватиме єдиним джерелом істини для всієї платформи. Оскільки взаємодія серверної бізнес-логіки з базою даних реалізована виключно через систему об'єктно-реляційного відображення (ORM) Entity Framework Core [4], розробнику не потрібно вручну писати складні SQL-скрипти для створення таблиць. Натомість структура таблиць, типи колонок, обмеження, індекси та зовнішні ключі (Foreign Keys) створюються повністю автоматизовано за допомогою вбудованого механізму міграцій на основі доменних моделей. Цей підхід реалізує концепцію «інфраструктура як код», дозволяючи версіонувати зміни схеми бази даних разом із вихідним кодом проєкту. Для ініціалізації бази в інтерфейсі командного рядка (CLI) .NET у кореневій директорії серверного проєкту виконується команда `dotnet ef database update`. Ця команда аналізує

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		49

поточний стан бази даних, генерує відповідні SQL-інструкції, застосовує всі відсутні міграції у строгій хронологічній послідовності та створює готову до роботи схему, що повністю відповідає архітектурі розробленої онлайн-спільноти.

Наступним, критично важливим кроком є налаштування глобального конфігураційного файлу серверної частини – `appsettings.json`, а також його середовище-залежних варіацій (наприклад, `appsettings.Production.json` або `appsettings.Development.json`). У цьому файлі централізовано зберігаються всі параметри, необхідні для коректного функціонування бекенду. Головним компонентом конфігурації є рядок підключення (Connection String) до створеної бази даних PostgreSQL. У ньому вказуються мережева адреса хоста (або IP-адреса віддаленого сервера), робочий порт (за замовчуванням 5432), назва цільової бази даних та облікові дані доступу. Для забезпечення вищого рівня безпеки в робочому середовищі ці дані можуть передаватися системі не через конфігураційний файл, а за допомогою безпечних змінних оточення (Environment Variables).

Окрему увагу на етапі конфігурації серверної частини приділено налаштуванню параметрів безпеки та ідентифікації. Зокрема, у конфігураційному файлі визначаються ключові параметри для генерації та валідації JSON Web Tokens (JWT) [7, 18], які відповідають за авторизацію та автентифікацію користувачів системи. Розробник повинен вказати унікальний секретний ключ (Secret Key), який використовується алгоритмом HMAC для симетричного криптографічного підпису токенів, а також ідентифікатори видавця та цільової аудиторії споживачів. Це гарантує, що токен, згенерований даним сервером, не зможе бути підроблений зловмисниками або використаний у сторонніх системах. Також на цьому етапі налаштовується глобальна політика CORS, яка явно визначає суворий список дозволених доменів (клієнтських адрес), що мають право надсилати асинхронні HTTP-запити до серверного API, блокуючи таким чином несанкціоновані міжсайтові запити та захищаючи користувачів від атак типу CSRF.

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		50

Підготовка клієнтської частини, створеної за допомогою компонентної JavaScript-бібліотеки React [2], є окремим технологічним процесом, який розпочинається із встановлення всіх необхідних зовнішніх залежностей. Для цього використовується пакетний менеджер Node.js (команда `npm install`), який зчитує файл маніфесту `package.json` та завантажує точні версії всіх бібліотек у локальну директорію `node_modules`. Цей підхід гарантує абсолютну ідентичність середовища розробки на різних машинах.

Для забезпечення коректної маршрутизації мережових запитів між клієнтом та бекендом у корені фронтенд-проєкту створюється конфігураційний файл оточення `.env`. У ньому прописуються глобальні константи, зокрема актуальна базова адреса серверного API. Використання файлу оточення дозволяє легко перемикати клієнтський застосунок між режимами локального тестування та робочого використання без необхідності прямого втручання у вихідний код React-компонентів.

Після успішного завершення конфігурації виконується команда збірки проєкту (`npm run build`). Ця команда запускає процес компіляції та транспіляції вихідного коду (JSX та сучасного синтаксису JavaScript) у набір максимально оптимізованих, мініфікованих статичних файлів (HTML, CSS, JS). Під час цього процесу застосовуються складні алгоритми видалення мертвого коду (Tree Shaking) та розбиття коду на менші фрагменти, що суттєво мінімізує кінцевий розмір клієнтського бандлу. Це гарантує надшвидке завантаження графічного інтерфейсу в браузері кінцевого користувача навіть за умов слабкого інтернет-з'єднання, що відповідає сучасним вимогам до продуктивності вебзастосунків.

У рамках практичної реалізації системи було впроваджено передові DevOps-практики, зокрема процеси безперервного розгортання (Continuous Deployment – CD). Клієнтська частина розгортається на сучасній спеціалізованій хмарній платформі Vercel. Вибір цього сервісу обґрунтований його здатністю автоматично розповсюджувати статичні файли через глобальну мережу доставки контенту (CDN), що забезпечує мінімальну затримку для користувачів з різних географічних регіонів. Платформа автоматично інтегрується з репозиторієм

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		51

GitLab: при кожному фіксації змін (push) у головну гілку, Vercel перехоплює вебхук (Webhook), самостійно ініціює процес збірки (npm run build) та оновлює робочу версію сайту. Крім того, платформа автоматично генерує та оновлює SSL-сертифікати, забезпечуючи надійне та безпечне з'єднання за протоколом HTTPS з перших секунд роботи сайту. Безпосередній обмін даними між розгорнутим клієнтським застосунком та серверною частиною відбувається за суворим архітектурним стилем REST API [26]. Ініціатором взаємодії виступає клієнт, використовуючи сучасний Promise-орієнтований HTTP-клієнт Axios [25], який забезпечує надійне асинхронне виконання запитів, автоматичну серіалізацію JSON-даних та зручну систему перехоплення (interceptors) для ін'єкції JWT-токенів у заголовки авторизації.

Окремої уваги заслуговує реалізація аналогічного процесу безперервного розгортання для серверної частини системи. З метою мінімізації ручного втручання та забезпечення стабільності оновлень бекенд інтегровано з сучасною хмарною платформою Render. Роль основного обробника мережевих запитів тут виконує вбудований у платформу .NET високопродуктивний кросплатформний вебсервер Kestrel. Механізм автоматизації полягає у використанні вебхуків, налаштованих між GitLab та серверами Render. При виконанні команди git push в основну гілку репозиторію, Render автоматично перехоплює подію, ініціює створення ізольованого контейнерного середовища, виконує завантаження необхідних NuGet-пакетів та запускає процес компіляції проєкту командою dotnet build. У разі успішної збірки нова версія серверного API автоматично публікується в мережі Інтернет із застосуванням стратегії Zero-Downtime Deployment (розгортання без простоїв).

Такий архітектурний підхід повністю відповідає сучасним індустріальним стандартам життєвого циклу розробки програмного забезпечення (SDLC), гарантуючи, що клієнтська частина завжди взаємодіє з найактуальнішою та стабільною версією серверної логіки без необхідності ручного перенесення скомпільованих файлів. Запропонована конфігурація не вимагає використання надлишкових інфраструктурних рішень, гарантує високий рівень безпеки та

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		52

відмовостійкості, а також забезпечує простоту подальшої підтримки інформаційної системи.

3.2 Структура інтерфейсу інформаційної системи

При запуску сайту відображається головна сторінка з усіма статтями (Рис. 3.1), пошуковий рядок, кнопки для сортування статей та категоріями, випадаюче меню для сортування за алфавітом або проти алфавіту. Інтерфейс за замовчуванням українською мовою, а статті відсортовані за алфавітом.

Доступні мови інтерфейсу – українська та англійська. Мова інтерфейсу обирається користувачем за допомогою відповідного випадаючого списку зверху. При виборі користувачем англійської мови весь інтерфейс інформаційної системи перекладається на англійську мову. На рисунку 3.2 можна побачити головну сторінку інформаційної системи онлайн-спільноти фанатів відеоігор англійською мовою.

Статті можна сортувати за категоріями: Персонажі, Зброя, Локації, Гра. Персонажів також можна відсортувати за підкатегоріями: Герої та Лиходії.

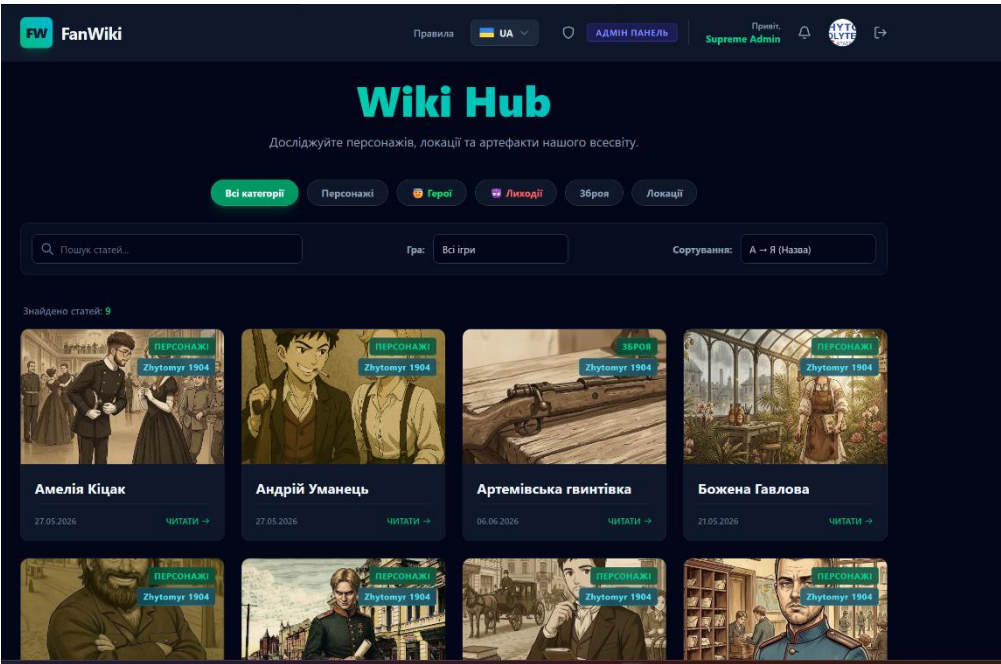


Рис. 3.1 Головна сторінка інформаційної системи

		Магаз А. О.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		53

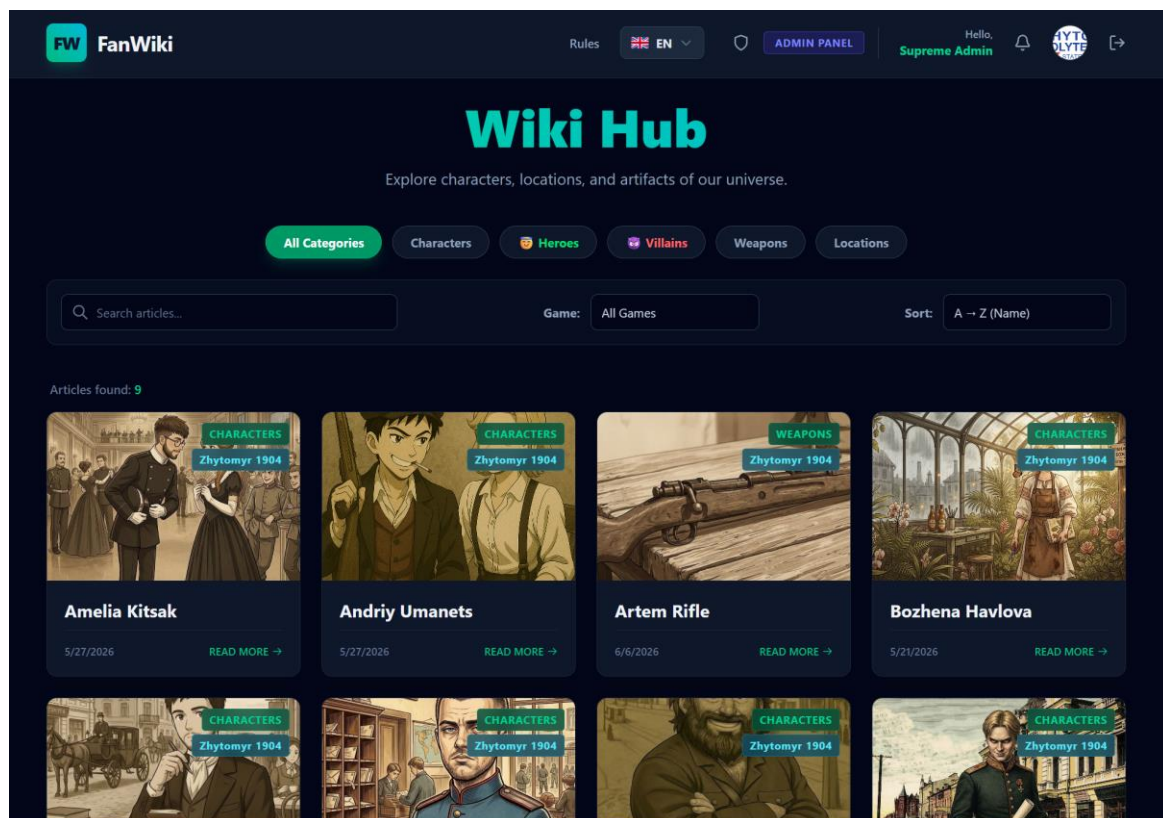


Рис. 3.2 Головна сторінка інформаційної системи англійською мовою

При натисненні кнопки «Персонажі» показуються лише статті з категорії «Персонажі», але не відсортовані за підкатегорією. На рисунку 3.3 наведено всі статті з категорії «Персонажі».

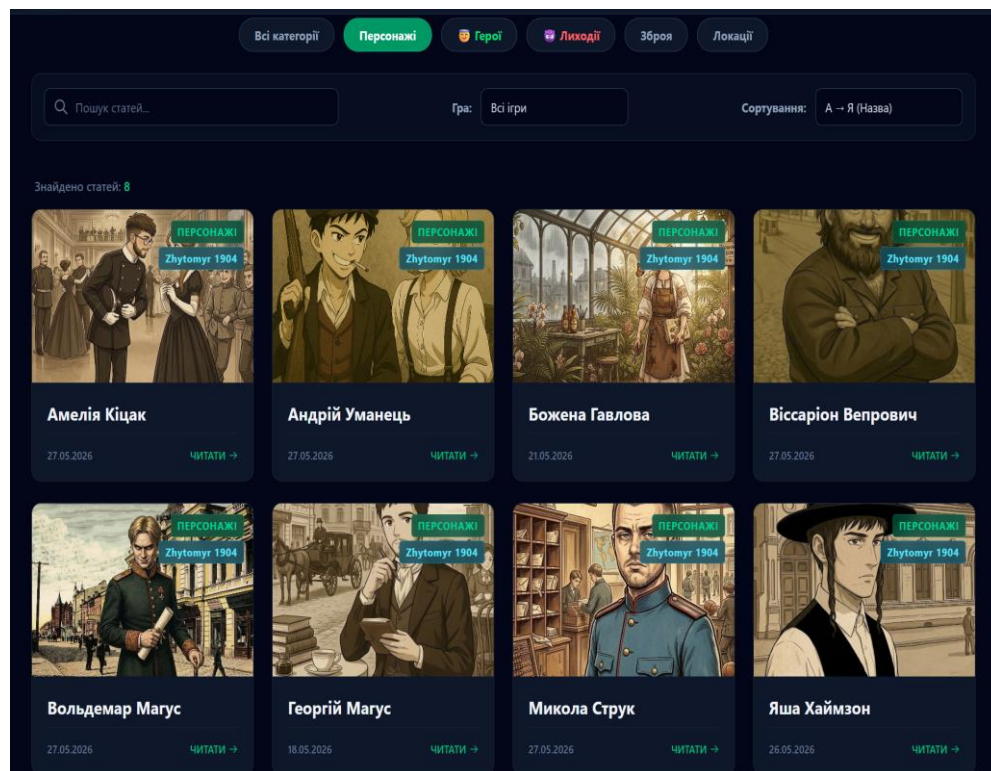


Рис. 3.3 Всі статті з категорії «Персонажі»

		Магаз А. О.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		54

Враховуючи специфіку сюжетних ліній у відеоіграх, категорію персонажів додатково розширено системою підкатегорій для розподілу героїв за їхнім моральним спрямуванням. Застосування фільтра «Герої» дозволяє виокремити протагоністів та позитивних персонажів (Рис. 3.4), що значно спрощує навігацію для користувачів, які цікавляться конкретною сюжетною аркою.

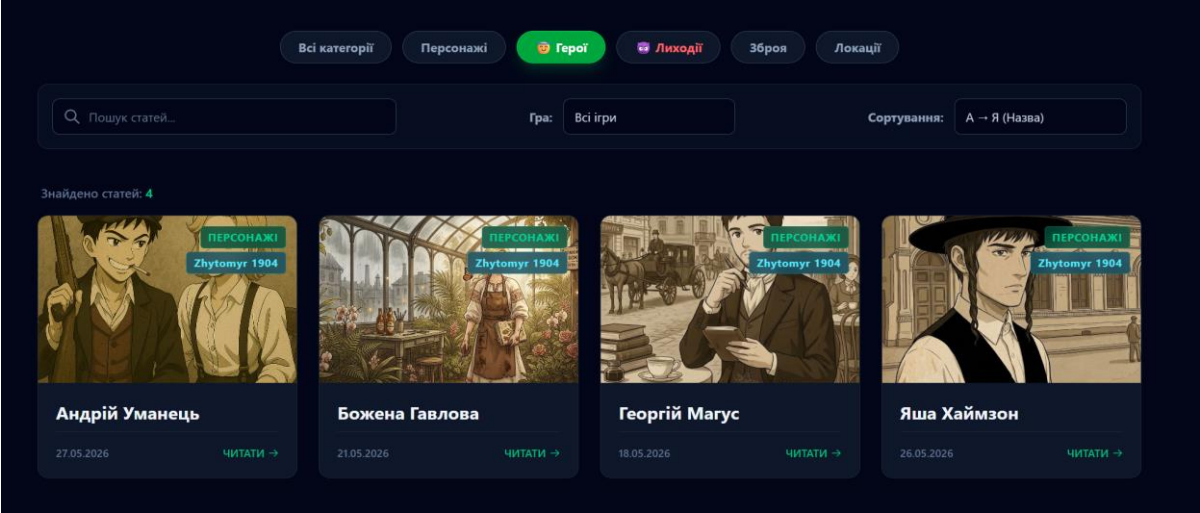


Рис. 3.4 Герої

Аналогічним чином реалізовано фільтрацію негативних персонажів (Рис. 3.5). Такий логічний поділ на протагоністів та антагоністів не лише полегшує пошук, але й допомагає новим гравцям швидше розібратися у складних взаємовідносинах всередині ігрового всесвіту. Візуальне розмежування цих підкатегорій робить інтерфейс бази знань більш інтуїтивно зрозумілим.

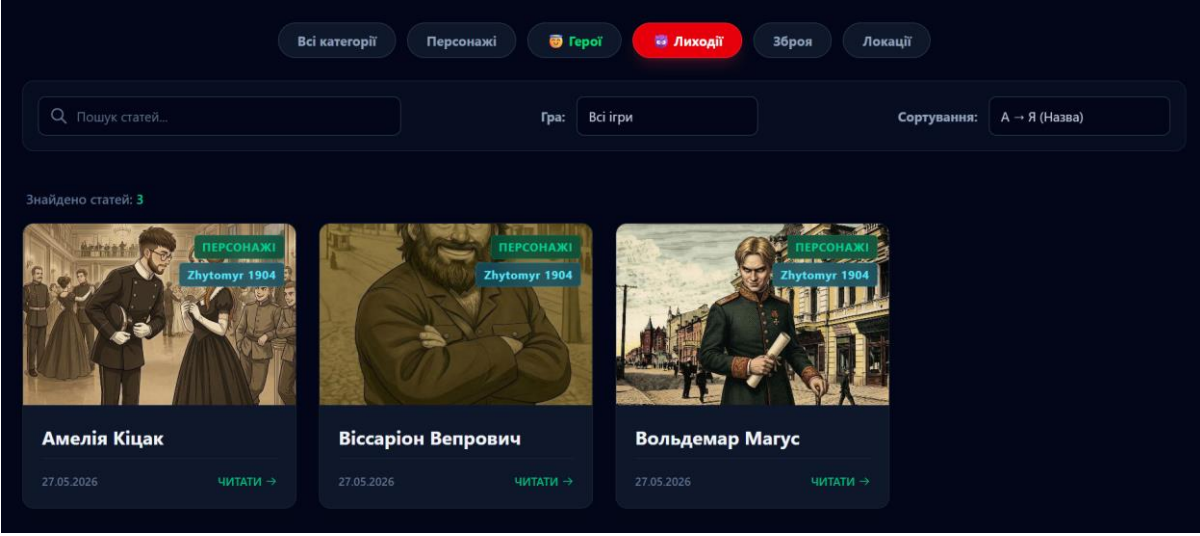


Рис. 3.5 Лиходії

При виборі категорії «Зброя» вбудований алгоритм фільтрації приховує нерелевантний контент, відображаючи лише ті статті, що описують різноманітне ігрове озброєння та його характеристики. На рисунку 3.6 наочно зображено сторінку розробленої інформаційної системи, яка містить сформовану вибірку статей, що належать виключно до категорії «Зброя».

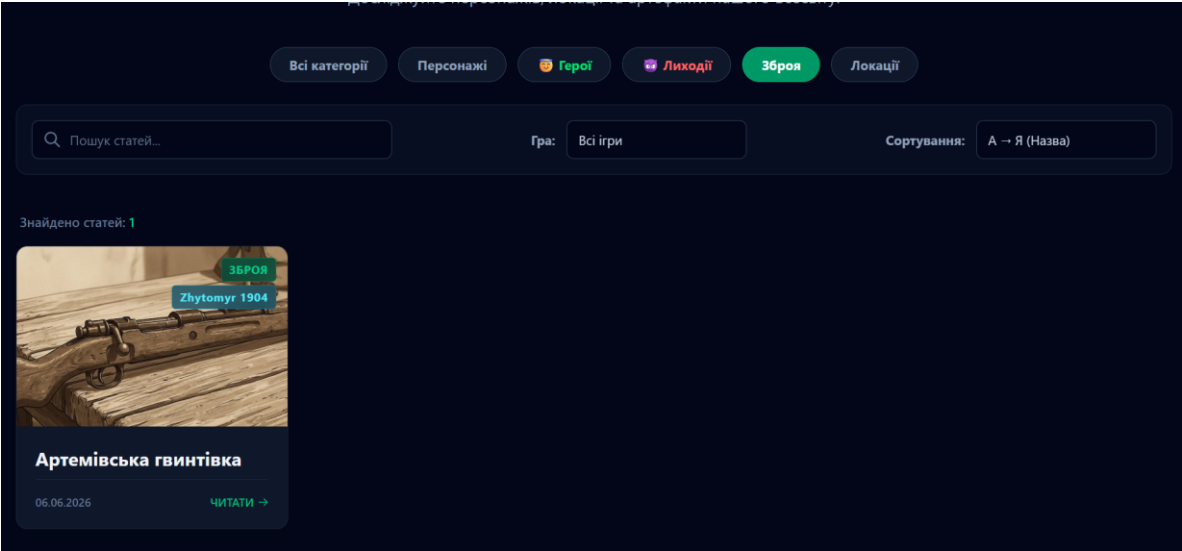


Рис. 3.6 Зброя

Якщо користувач вибере категорію «Локації», то йому відповідно відобразяться лише статті з локаціями ігор. На рисунку 3.7 зображено лише локації.

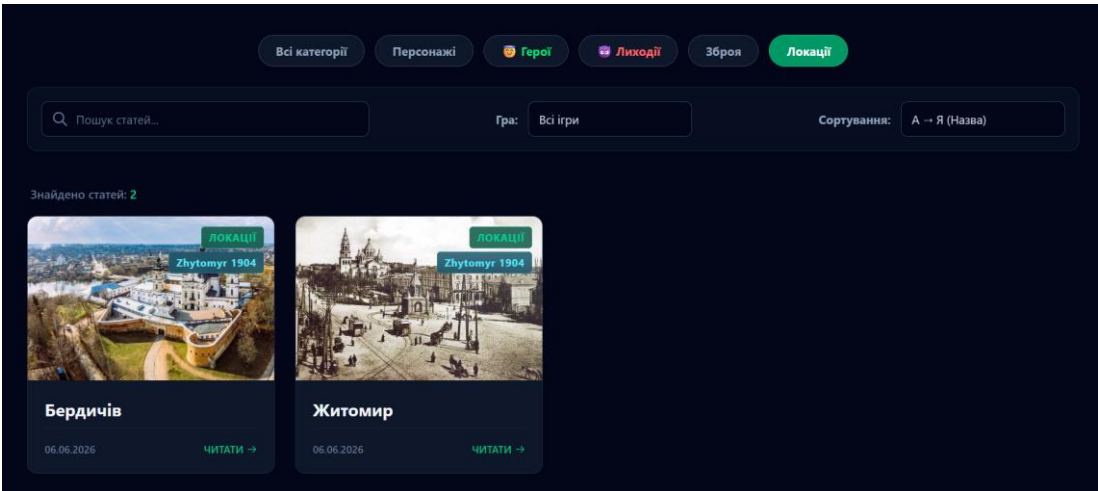


Рис. 3.7 Локації

Інформаційна система онлайн-спільноти фанатів відеоігор надає можливість своїм користувачам фільтрувати та сортувати статті за належністю до конкретної відеогри. У верхній частині інтерфейсу передбачено спеціальне

текстове поле «Гра», де користувач може обрати назву бажаного ігрового проєкту. Після введення відповідного запиту система миттєво опрацьовує дані, і на сторінці відображаються виключно ті картки статей, які безпосередньо стосуються вказаної гри. На рисунку 3.8 статті відсортовано за конкретною грою.

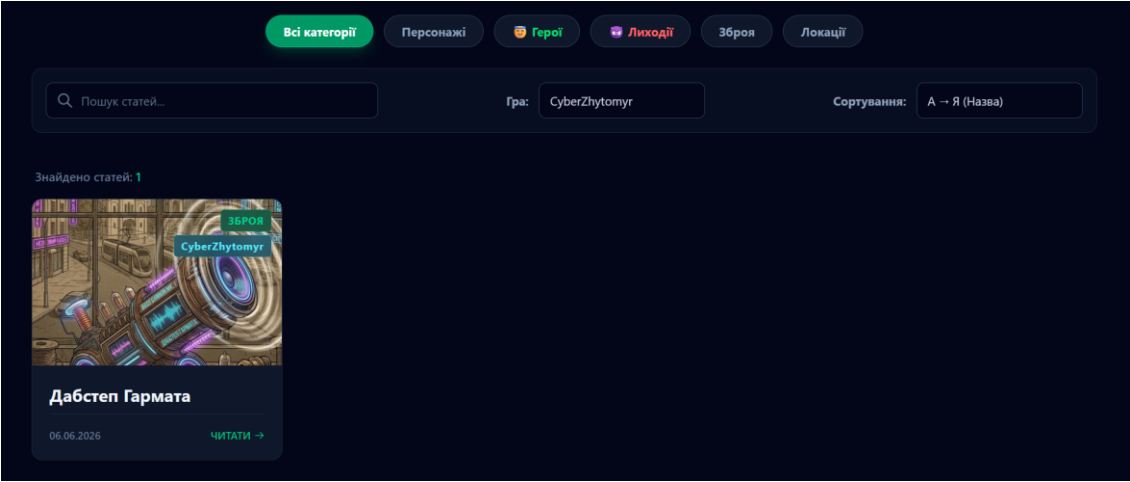


Рис. 3.8 Статті, відсортовані за назвою гри

При натисненні лівою кнопкою миші на інтерактивну картку система ініціює маршрутизацію користувача на сторінку детального перегляду обраної статті. Інтерфейс цієї сторінки спроектовано з урахуванням необхідності зручного структурування великих обсягів текстової інформації: біографія та деталізоване досьє персонажа. На рисунку 3.9 наочно продемонстровано приклад інтерфейсу такої розгорнутої статті.

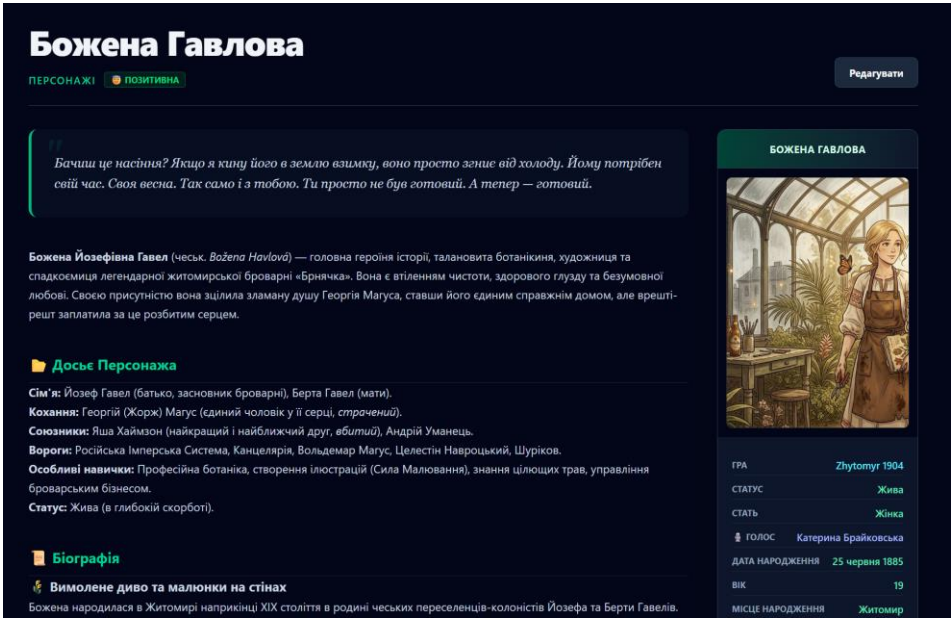


Рис. 3.9 Приклад статті

Важливим елементом структуризації даних на сторінці статті є інфобокс (Рис. 3.10), який традиційно розташовується у правій частині інтерфейсу. Цей компонент акумулює ключові метадані про об'єкт у вигляді зручної таблиці: належність до гри, поточний статус, дату та місце народження персонажа, акторів озвучення, а також інтерактивні посилання на пов'язаних сутностей (родина, друзі, вороги).



Рис. 3.10 Інфобокс

Взаємодія аудиторії реалізується через систему коментування, інтегровану безпосередньо під основним текстом кожної статті. Проте, з метою захисту від автоматизованого спаму та підтримки культури спілкування, функціонал коментування має жорсткі обмеження прав доступу. Неавторизовані користувачі можуть лише переглядати існуючі обговорення, тоді як поле введення тексту замінюється системним повідомленням із проханням увійти до облікового запису (Рис. 3.11).

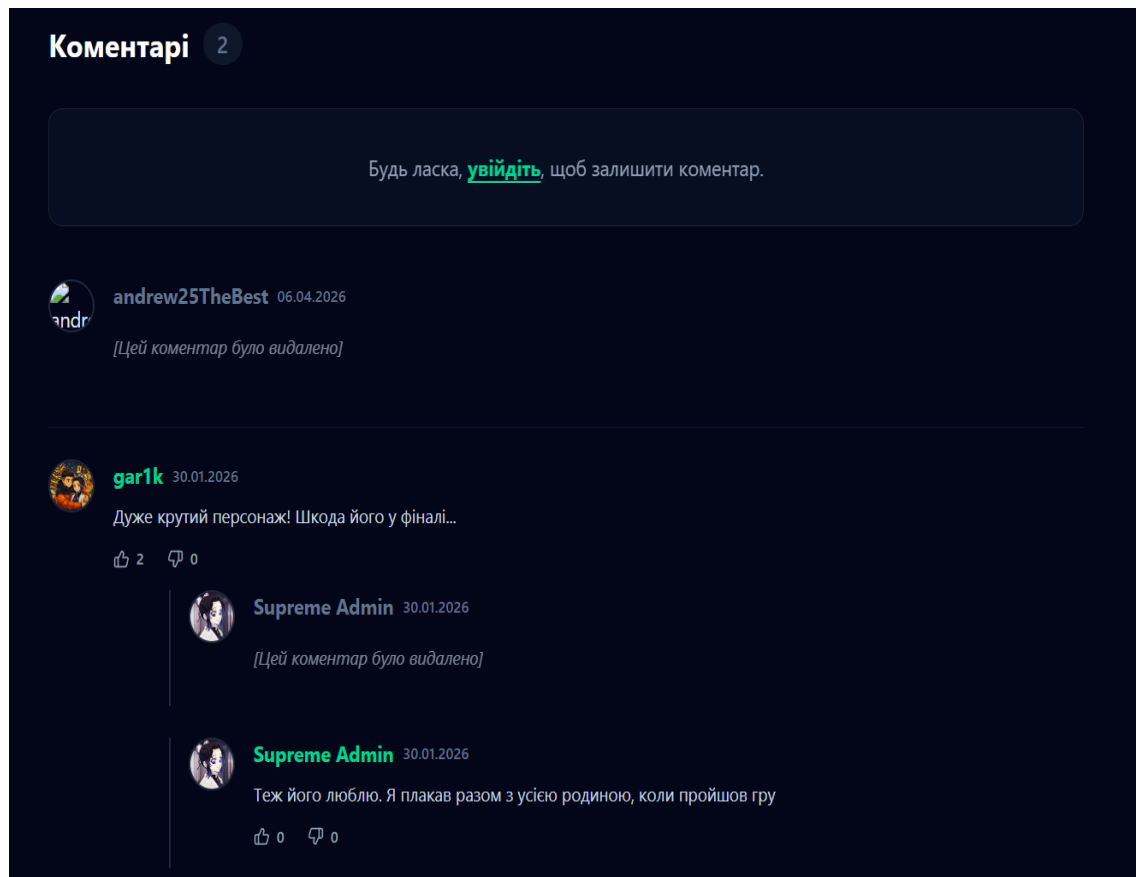


Рис. 3.11 Секція з коментарями та прохання увійти в обліковий запис для можливості коментування

Аналогічна політика безпеки застосовується і до модуля модерації контенту. Спроби незареєстрованого відвідувача ініціювати надсилання скарги на статтю чи коментар автоматично перехоплюються системою. Користувачу виводиться модальне вікно сповіщення, яке інформує про необхідність проходження процедури авторизації для виконання цієї дії (Рис. 3.12).

		Магаз А. О.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		59

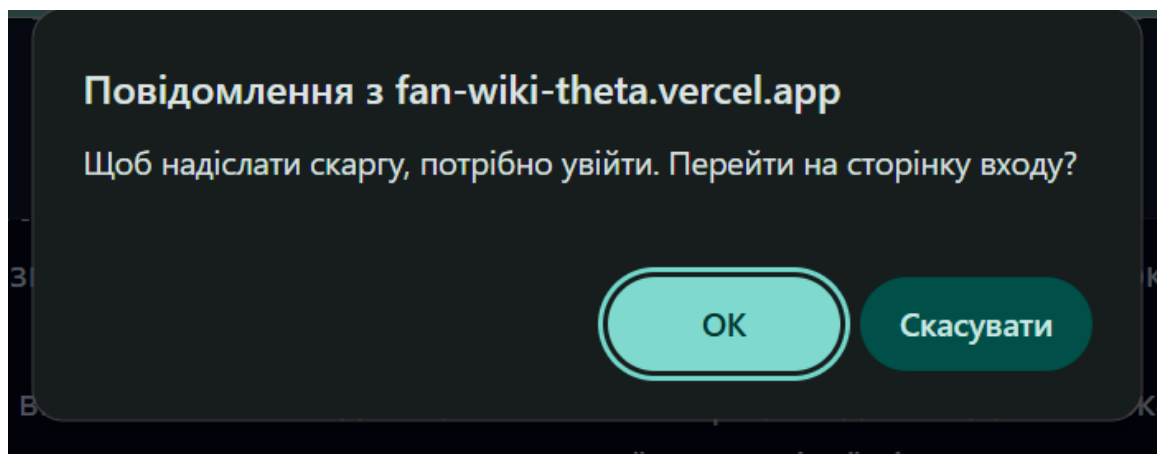


Рис. 3.12 Відсутня можливість надсилання репортів через відсутність авторизації

Для створення власного облікового запису необхідно пройти реєстрацію. Для цього здійснюється перехід за допомогою кнопки «Реєстрація», після чого відкривається форма реєстрації (Рис. 3.13). Користувачу необхідно ввести логін, електронну пошту, нікнейм та пароль. Якщо виникають труднощі зі створенням надійного пароля, передбачено використання спеціальної функції для його автоматичної генерації.

Рис. 3.13 Заповнена форма реєстрації

		Магаз А. О.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Єфремов Ю. М.				60
Змн.	Арк.	№ докум.	Підпис	Дата		

Далі нам необхідно підтвердити свою електронну пошту за допомогою шестизначного коду, який має прийти на пошту (Рис. 3.14). На рисунку 3.15 зображено код підтвердження, який прийшов на пошту користувача.

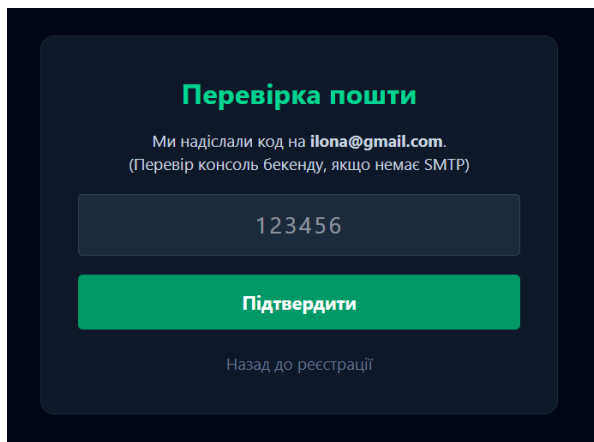


Рис. 3.14 Перевірка пошти

```
Body: <h1>Вітаємо!</h1><p>Ваш код підтвердження: <b>351645</b></p>
```

Рис. 3.15 Код підтвердження

Після введення коду підтвердження, який прийшов на пошту, обліковий запис був успішно створений та тепер ми можемо увійти в нього. Якщо користувач забув свій пароль, він може його відновити. Для цього необхідно натиснути на кнопку «Забули пароль?». Далі користувача переносить на сторінку для відновлення пароля (Рис. 3.16). Щоб відновити пароль, необхідно надіслати адресу своєї електронної пошти.

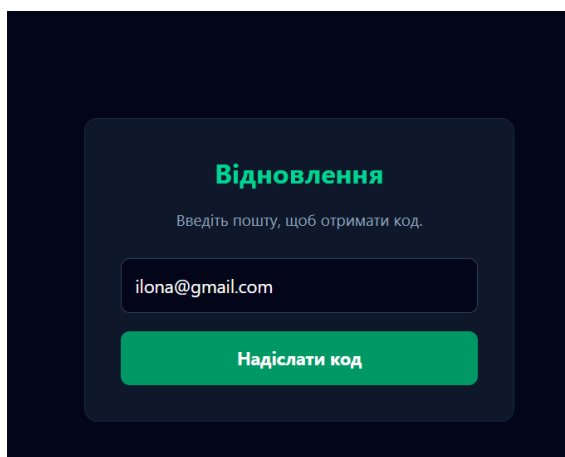


Рис. 3.16 Введення електронної пошти для відновлення пароля

Після цього система генерує та надсилає код для відновлення. Отриманий код та новий пароль вводяться у відповідні поля форми (Рис. 3.17).

		Магаз А. О.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		61

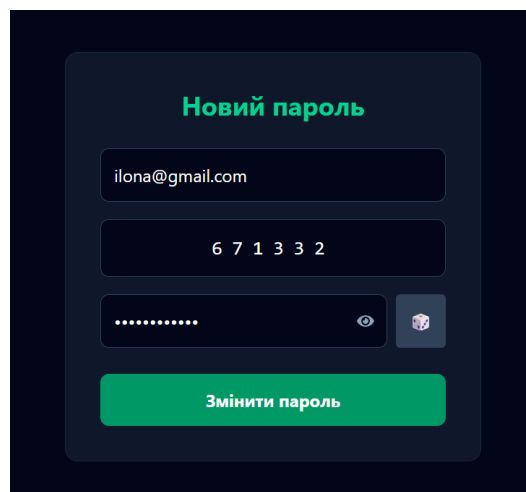


Рис. 3.17 Код відновлення та новий пароль

Про успішну зміну пароля система інформує за допомогою відповідного графічного сповіщення (Рис. 3.18).

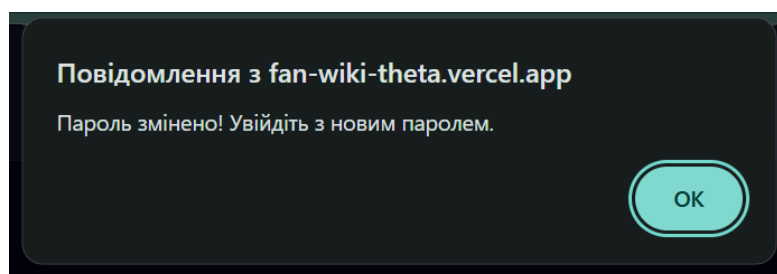


Рис. 3.18 Успішна зміна пароля

За замовчуванням аватар користувача відсутній, проте функціонал системи дозволяє його встановити. Для цього здійснюється клік по іконці біля нікнейма, що ініціює перехід на сторінку «Профіль користувача» (Рис. 3.19).

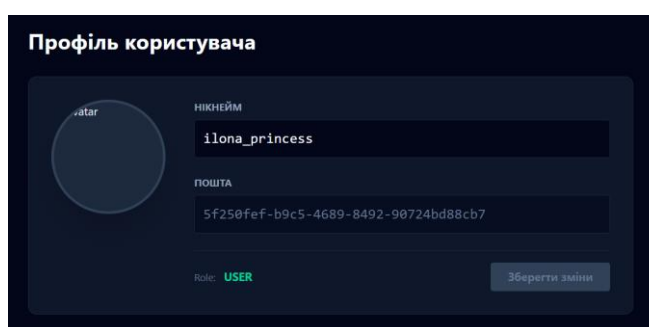


Рис. 3.19 Профіль користувача

На цій сторінці надається можливість зміни нікнейма або аватара. Збереження нових даних відбувається за допомогою кнопки «Зберегти зміни». Завантаження нового аватара здійснюється шляхом натискання на відповідну графічну область з подальшим вибором файлу з локального сховища (Рис. 3.20).

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

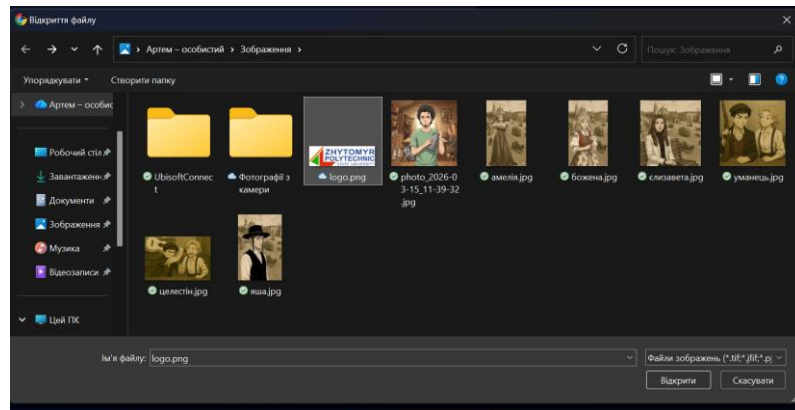


Рис. 3.20 Вибір файлу

Після успішного завантаження зображення на хмарний сервер та зміни текстових даних, оновлена інформація миттєво відображається в інтерфейсі профілю. Користувач має змогу візуально оцінити внесені зміни перед їх остаточним закріпленням у базі даних за допомогою кнопки збереження (Рис. 3.21).

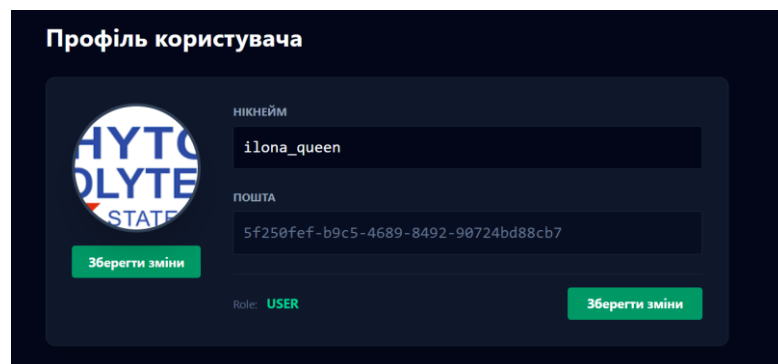


Рис. 3.21 Новий нікнейм та аватарка

Після застосування змін оновлені персональні дані автоматично синхронізуються по всьому застосунку. Новий аватар та нікнейм починають відображатися на глобальній панелі навігації, а також у всіх попередніх та майбутніх коментарях користувача, створюючи єдиний персоналізований досвід.

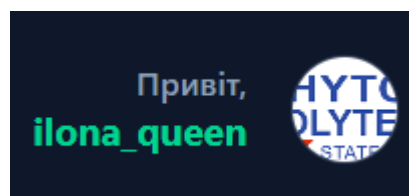


Рис. 3.22 Результат виконання

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				63
Змн.	Арк.	№ докум.	Підпис	Дата		

Наявність верифікованого облікового запису відкриває повний доступ до соціального інструментарію платформи. Користувач може брати активну участь у дискусіях, публікуючи розгорнуті коментарі під статтями. Текстове поле підтримує базове форматування та автоматично адаптується під обсяг введеного тексту (Рис. 3.23).

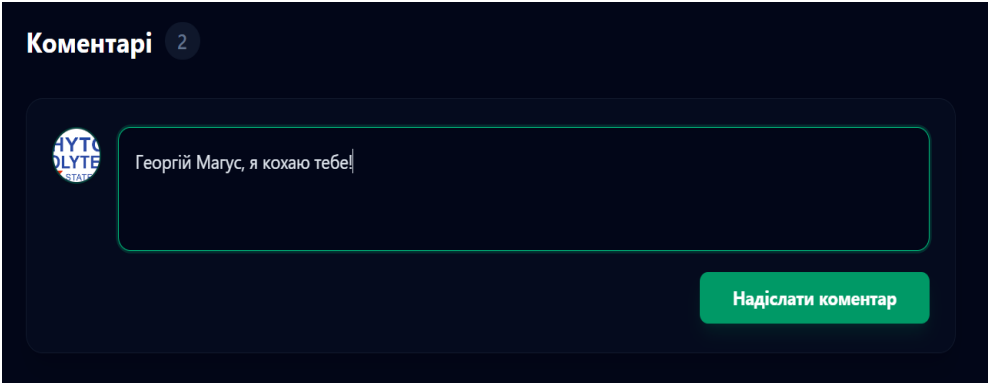


Рис. 3.23 Процес написання коментаря

Окрім прямого коментування, реалізовано механізми нелінійної взаємодії. Користувачі мають змогу формувати деревоподібні гілки обговорень, відповідаючи на конкретні дописи інших учасників. Для швидкого вираження підтримки або незгоди запроваджено систему реакцій, які відображаються у вигляді лічильника під кожним коментарем (Рис. 3.24).

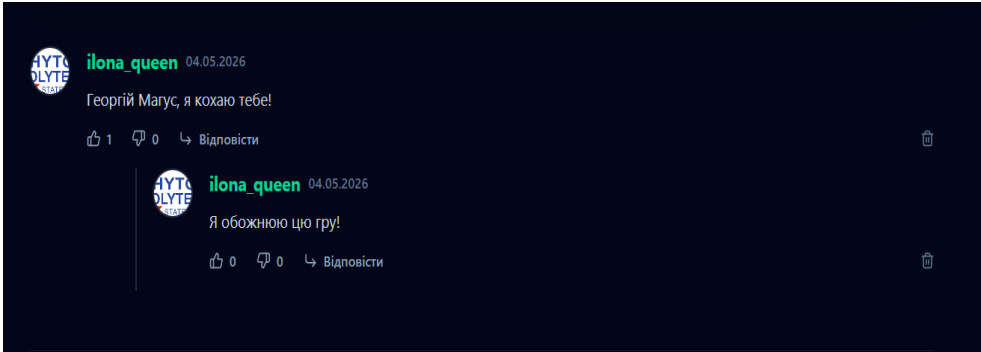


Рис. 3.24 Надісланий коментар, реакція та відповідь до нього

З метою забезпечення контролю над власним цифровим слідом користувачам надано можливість видалення своїх публікацій. При ініціації видалення система застосовує підхід «м'якого видалення»: фізично текст приховується з публічного доступу, а на його місці з'являється інформаційна заглушка про те, що коментар було видалено автором (Рис. 3.25).

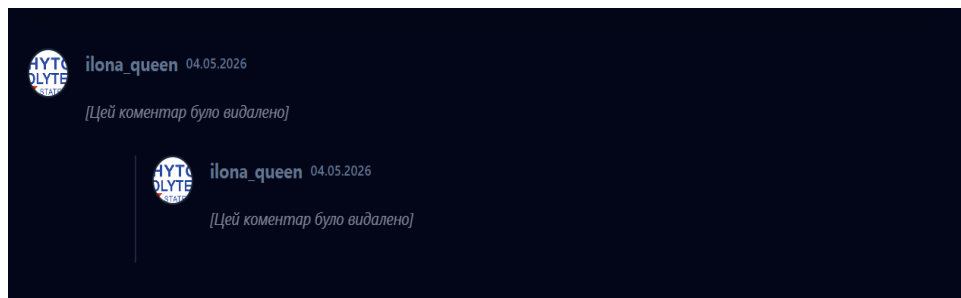


Рис. 3.25 Видалені коментарі

Для підтримання високої якості контенту та здорової атмосфери у спільноті розроблено модуль зворотного зв'язку. Кожен авторизований користувач може ініціювати надсилення репорту на будь-яку статтю чи коментар. У модальному вікні необхідно детально описати виявлену проблему, наприклад, вказати на фактичні помилки у лорі або поскаржитися на мову ворожнечі (Рис. 3.26).

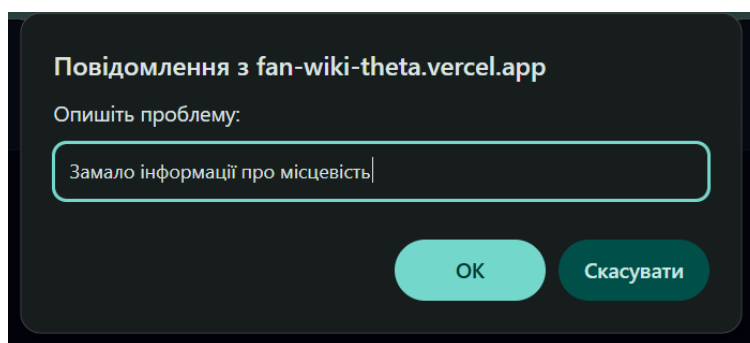


Рис. 3.26 Процес написання та надсилення репорту

Після відправки форми скарги клієнтський застосунок асинхронно передає дані на сервер, де вони фіксуються в окремій таблиці бази даних. Користувач миттєво отримує візуальне підтвердження того, що його звернення успішно зареєстровано та передано на розгляд адміністрації платформи.

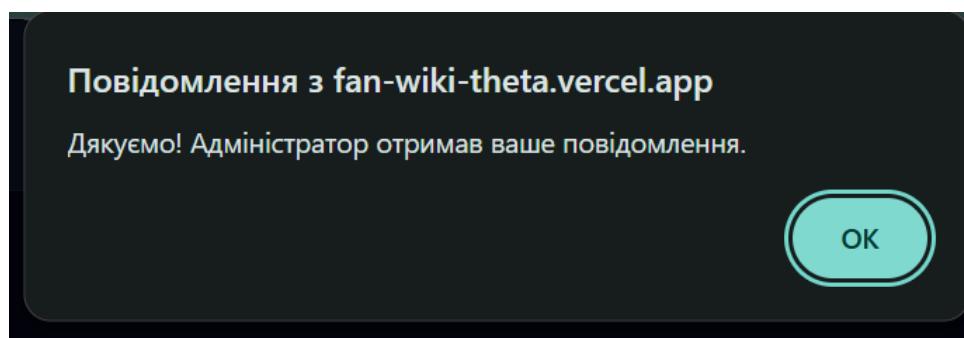


Рис. 3.27 Результат виконання

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		65

Аналізуючи роботу адміністратора платформи, варто зазначити, що на відміну від звичайного користувача, ця роль має доступ до спеціалізованої панелі управління та загального списку надісланих скарг (Рис. 3.28).

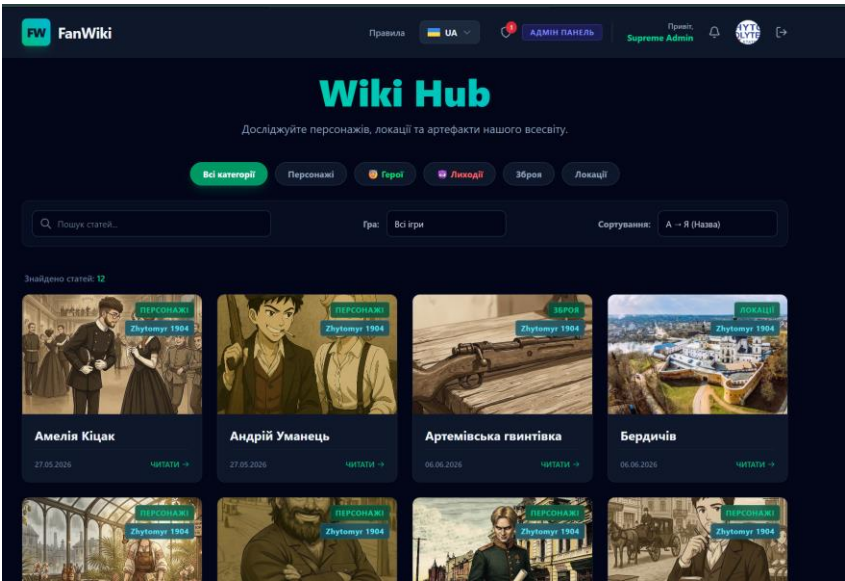


Рис. 3.28 Головна сторінка для адміністратора

Завдяки панелі адміністратора (Рис. 3.29) можна проглядати, додавати, редагувати та видаляти статті.

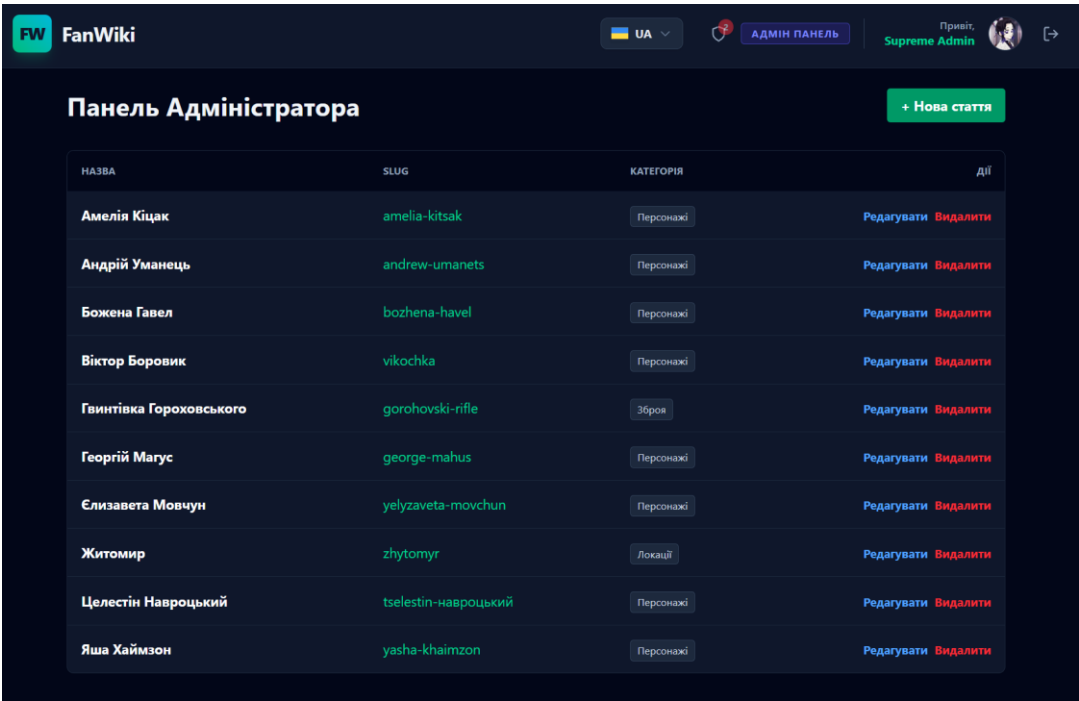


Рис. 3.29 Панель адміністратора

Щоб додати нову статтю необхідно натиснути на кнопку «Нова стаття». Адміністратора перенесе до форми для створення статті. Архітектура системи

передбачає гнучку структуру даних, яка адаптується залежно від обраного типу контенту.

Незалежно від категорії, кожна стаття містить набір базових атрибутів: назву, семантичний ідентифікатор (slug), коротку цитату, прив'язку до конкретної гри, основний текстовий контент та головне зображення.

Окрім базових полів, форма динамічно генерує додаткові параметри специфічні для кожної сутності: для сутності «Персонаж» (Рис. 3.30) передбачено введення розширених метаданих: тип персонажа, життєвий статус, стать, актор озвучування, а також реляційні зв'язки (родина, союзники, вороги). Для сутності «Зброя» заповнюються тактико-технічні характеристики: показник шкоди, рік випуску, тип набоїв та темп стрільби. Для сутності «Локація» фіксуються географічні та демографічні дані: регіон, рік заснування об'єкта та кількість населення.

The screenshot shows a web form titled "Створити статтю" (Create Article) in a dark theme. The form is for creating an article about a character, indicated by the "Інфо (УК)" tab. The fields are organized as follows: "НАЗВА" (Name) with the value "Користувач Звичайний" and "SLUG" with "user-simple"; "ЦИТАТА" (Quote) with "Щось я там казав"; "КАТЕГОРІЯ" (Category) set to "Персонажі" and "ТИП ПЕРСОНАЖА (ХАРАКТЕР)" (Character Type) set to "Позитивний (Герой)"; "ГРА" (Game) set to "CyberZhytomyr"; "СТАТУС" (Status) set to "Живий" and "СТАТЬ" (Gender) set to "Чоловік"; and "АКТОР ОЗВУЧУВАННЯ" (Voice Actor) set to "Актор Якийсь".

Рис. 3.30 Приклад створення статті про персонажа

Залежно від обраної категорії, форма створення автоматично рендерить додаткові поля метаданих. Для персонажів це інформація про вік, дату народження, статус, а також складні реляційні зв'язки, які дозволяють

встановлювати перехресні посилання на сторінки родичів, союзників чи ворогів поточного героя (Рис. 3.31).

ДАТА НАРОДЖЕННЯ: 15 вересня 2004

ВІК: 21

МІСЦЕ НАРОДЖЕННЯ: Житомир

СІМ'Я: + Додати

- Мама Користувача
- Тато Користувача

СОЮЗНИКИ: + Додати

- Друг 1
- Друг 2

ВРОГИ: + Додати

- Ворог 1
- Ворог 2

ТАКОЖ ВІДОМИЙ ЯК: + Додати

- Просто юзер

Рис. 3.31 Реляційні зв’язки персонажа, його вік та дата народження

Основний текстовий масив статті формується за допомогою інтегрованого WYSIWYG-редактора (Рис. 3.32). Він надає адміністратору широкий набір інструментів для семантичного форматування тексту: виділення жирним шрифтом, курсивом, створення заголовків різного рівня, списків та інтеграцію гіперпосилань, що дозволяє створювати професійно зверстані матеріали.

КОНТЕНТ

В I S H1 H2 T 16

Цей персонаж - він дуже цікавий! А як Ви б до нього ставились? Я лише позитивно!

Цей персонаж - він дуже цікавий! А як Ви б до нього ставились? Я лише позитивно!

~~Цей персонаж - він дуже цікавий! А як Ви б до нього ставились? Я лише позитивно!~~

Рис. 3.32 Контент та панель інструментів

Програма може створювати автоматичний переклад статті на англійську мову. Для цього існує відповідна форма та кнопка «Авто-переклад» (Рис. 3.33).

Переклад (Англійська) Авто-переклад

НАЗВА (EN)
Standard User

ЦИТАТА (EN)
I said something like that

CHARACTER FIELDS (EN)

АКТОР ОЗВУЧУВАННЯ (EN) ДАТА НАРОДЖЕННЯ (EN)
An Actor September 15, 2004

МІСЦЕ НАРОДЖЕННЯ (EN)
Zhytomyr

СІМ'Я (EN)
The User's Mother
The User's Dad

СОЮЗНИКИ (EN)
Friend 1
Friend 2

Рис. 3. 33 Форма з перекладеним вмістом

Після заповнення всіх полів та завантаження графічних матеріалів система генерує фінальну сторінку. Результатом успішного збереження є повноцінна стаття, яка одразу стає доступною для перегляду всім користувачам платформи у вибраній мовній версії, зберігаючи задане форматування та структуру.

FanWiki Правила UA АДМІН ПАНЕЛЬ Supreme Admin

Користувач Звичайний

ПЕРСОНАЖІ ПОЗИТИВНИЙ Редагувати

Щось я там казав

Цей персонаж - він дуже цікавий! А як Ви б до нього ставились? Я лише позитивно!
Цей персонаж - він дуже цікавий! А як Ви б до нього ставились? Я лише позитивно!
Цей персонаж - він дуже цікавий! А як Ви б до нього ставились? Я лише позитивно!

ПОВІДОМИТИ ПРО ПОМИЛКУ

КОРИСТУВАЧ ЗВИЧАЙНИЙ

ГРА CyberZhytomyr
СТАТУС Живий
СТАТЬ Чоловік
ГОЛОС Актор Яхор
ДАТА НАРОДЖЕННЯ 15 вересня 2004
ВІК 21
МІСЦЕ НАРОДЖЕННЯ Житомир

Рис. 3.34 Успішно створена стаття про персонажа

Завдяки модулю локалізації користувачі, які обрали англійську мову інтерфейсу, автоматично перенаправляються на англomовну версію щойно створеної статті. Система динамічно підтягує перекладені метадані та основний текст із відповідних таблиць бази даних.

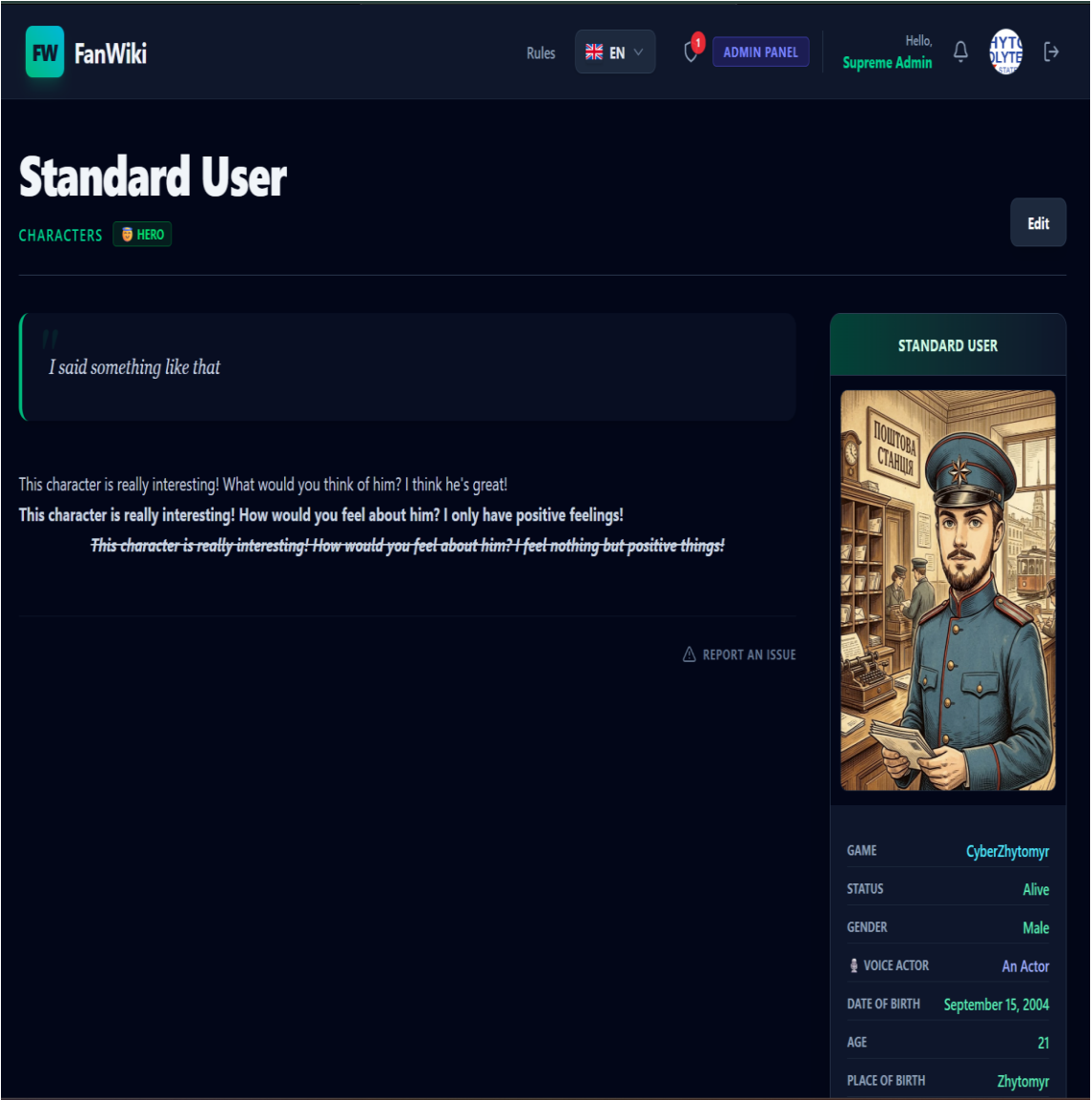


Рис. 3.35 Стаття, перекладена на англійську мову

Процес наповнення бази знань матеріалами про ігровий арсенал має свою специфіку. При виборі категорії «Зброя» форма введення адаптується, пропонуючи адміністратору вказати тактико-технічні характеристики об'єкта: показник базової шкоди, місткість магазину, темп стрільби та рік створення зразка у межах ігрового всесвіту (Рис. 3.36).

Створити статтю

■ Інфо (UK)

НАЗВА

Дабстеп Гармата

SLUG

dubstep-gun

ЦИТАТА

Бум-бум-бум...

КАТЕГОРІЯ

Зброя

ГРА

Космічний Кухар

ПОКАЗНИК ШКОДИ

100

RPK

2012

НАБОЇ

60

RPM

1337

КОНТЕНТ

<

>

B

I

S

H₁

H₂

T

16

≡

≡

≡

≡

≡

≡

🔍

📷

🔗

Неймовірно крута зброя. Хочете таку?

Рис. 3.36 Процес створення статті про вид зброї

Згенерована стаття про зброю візуально відрізняється від сторінок персонажів завдяки специфічному інфобоксу (Рис. 3.37). Він чітко відображає введені бойові характеристики, що дозволяє гравцям швидко порівнювати різні види озброєння між собою під час вивчення тактичних можливостей гри.

Дабстеп Гармата

Зброя

Редагувати

Бум-бум-бум...

Неймовірно крута зброя. Хочете таку?

Повідомити про помилку

ДАБСТЕП ГАРМАТА

ГРА	Космічний Кухар
ПОКАЗНИК ШКОДИ	100
НАВОЇ	60
ВРМ	1337
РІК	2012

Рис. 3.37 Успішно створена стаття про зброю

		Магаз А. О.		
		Єфремов Ю. М.		
Змн.	Арк.	№ докум.	Підпис	Дата

ИЗ.КР.Б-121-26-ИЗ

Арк.

71

Для документування ігрового простору використовується форма створення локації (Рис. 3.38). Вона вимагає введення географічних та статистичних даних, таких як назва регіону, кількість внутрішньоігрового населення та рік заснування поселення чи об'єкта згідно з офіційним лором гри.

Рис. 3.38 Процес створення статті про локацію

Опублікована сторінка локації містить відповідну панорамну ілюстрацію та структурований інфобокс із географічними даними (Рис. 3.39). Такий підхід забезпечує уніфікацію подачі інформації незалежно від того, описується невелика будівля чи ціле місто в ігровому світі.

Рис. 3.39 Успішно створена стаття про локацію

Окрім створення нових матеріалів, панель адміністратора дозволяє редагувати та видаляти вже існуючі статті.

Для редагування статті необхідно обрати її у загальному списку. При переході до форми всі поля автоматично заповнюються поточними даними з бази. Це дозволяє адміністратору зручно оновити текстовий контент, змінити характеристики персонажа чи зброї, або завантажити нове зображення. Після збереження змін оновлена інформація відразу відображається на платформі.

Функція видалення передбачена для очищення системи від неактуального контенту. Щоб запобігти випадковому видаленню важливої інформації через людський фактор, система обов'язково запитує додаткове підтвердження у модальному вікні. Тільки після підтвердження запис та пов'язані з ним дані видаляються з бази.

Для прикладу змінимо назву, ідентифікатор, характер та контент нашої новоствореної статті про персонажа (Рис. 3.40).

The screenshot shows a web form titled 'Редагувати' (Edit) in green text. The form is for editing a character article and is organized into several sections:

- Інфо (UK)**: A red square icon followed by the text 'Інфо (UK)'.
- НАЗВА**: A text input field containing 'Користувач Політехнічний'.
- SLUG**: A text input field containing 'user-politech'.
- ЦИТАТА**: A text input field containing 'Щось я там казав'.
- КАТЕГОРІЯ**: A dropdown menu with 'Персонажі' selected.
- ТИП ПЕРСОНАЖА (ХАРАКТЕР)**: A dropdown menu with 'Негативний (Лиходій)' selected.
- ГРА**: A text input field containing 'CyberZhytomyr'.
- СТАТУС**: A dropdown menu with 'Живий' selected.
- СТАТЬ**: A dropdown menu with 'Чоловік' selected.
- АКТОР ОЗВУЧУВАННЯ**: A text input field containing 'Актор Якийсь'.

Рис. 3.40 Процес редагування статті про персонажа

Після збереження змін оновлений матеріал без затримок замінює попередню версію на сервері. Усі правки в тексті, зміна статусу або оновлення реляційних зв'язків негайно відображаються на публічній сторінці статті, забезпечуючи спільноту найсвіжішою інформацією.

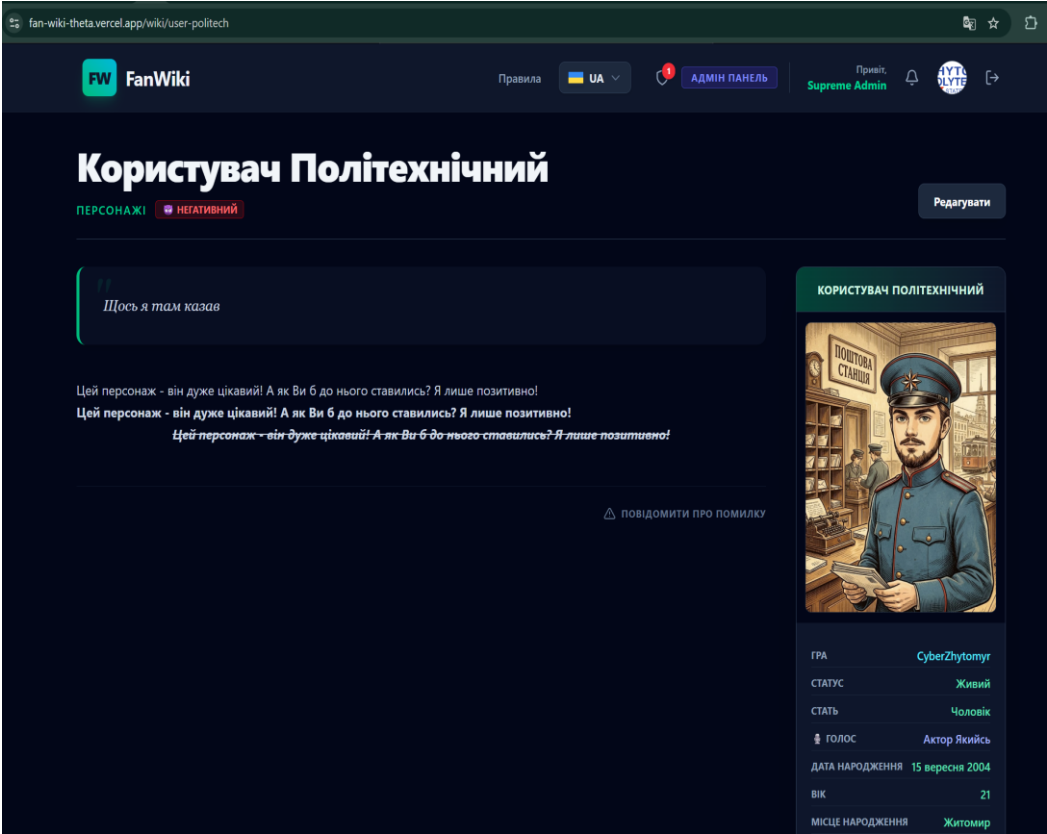


Рис. 3.41 Результат редагування

Внесені зміни автоматично рендеряться на всіх рівнях навігації. Якщо було змінено обкладинку або підзаголовки статті, ці оновлення миттєво відобразяться на прев'ю-картках головної сторінки та у відповідних розділах каталогу.

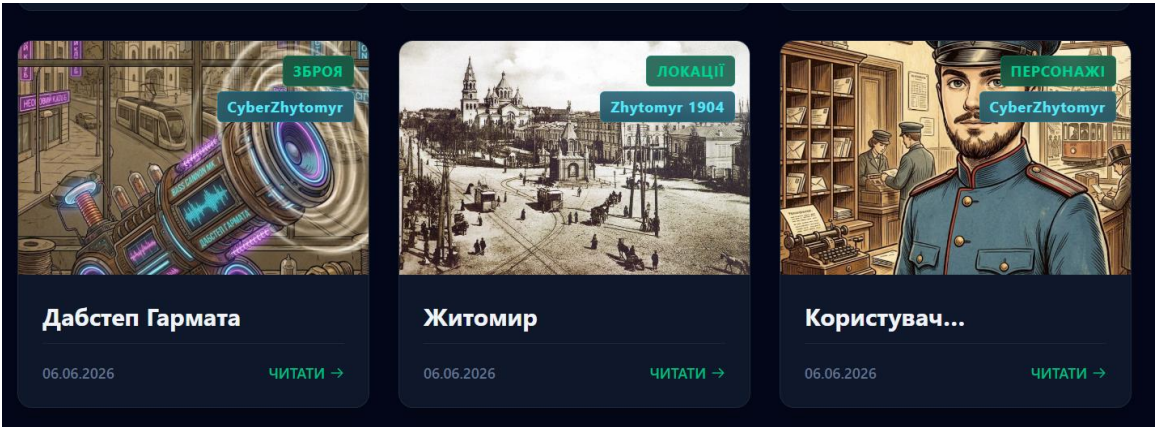


Рис. 3.42 Відредагована стаття на головній сторінці

Для видалення статті заходимо в панель адміністратора, знаходимо необхідну статтю та натискаємо кнопку «Видалити». Система вимагає підтвердження даної дії (Рис. 3.43).

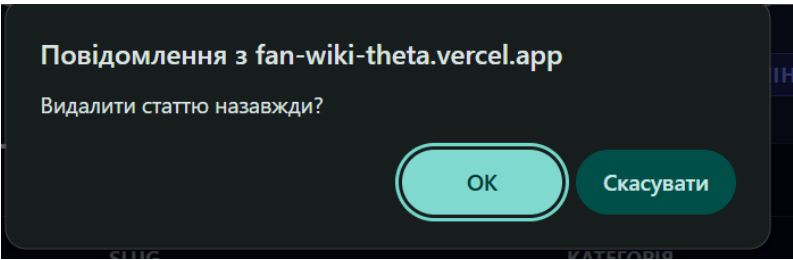


Рис. 3.43 Підтвердження видалення статті

Для забезпечення ефективного зворотного зв'язку з користувацькою спільнотою в системі реалізовано модуль «Центр скарг та пропозицій» (Рис. 3.44). Цей інструмент дозволяє адміністраторам здійснювати централізований моніторинг звернень щодо якості контенту або порушень правил поведінки на платформі.

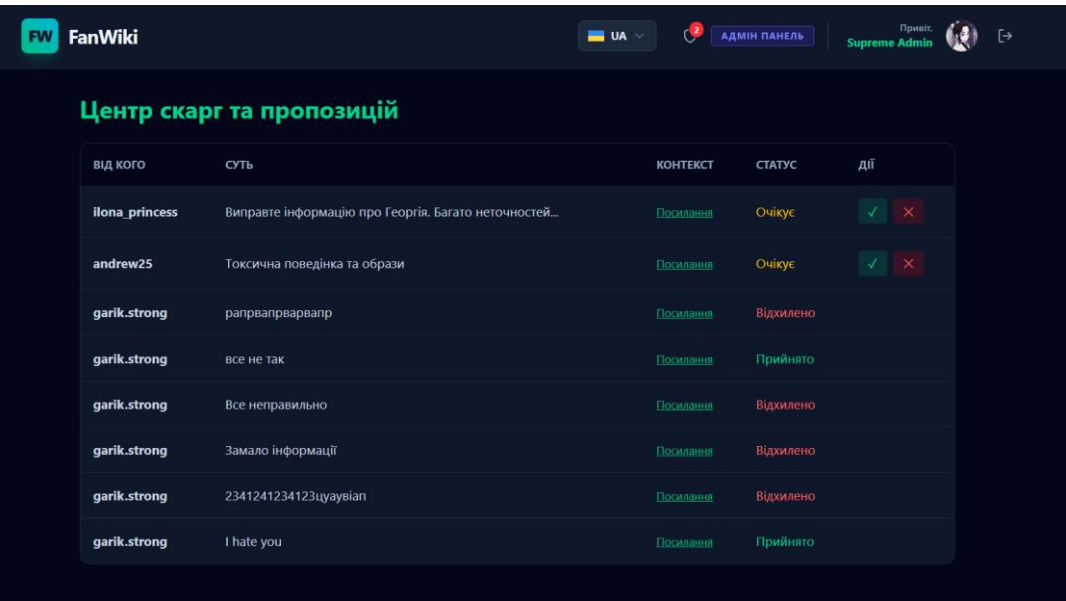


Рис. 3.44 Центр скарг та пропозицій

Життєвий цикл кожного запиту керується через систему статусів: «Очікує» (потребує розгляду адміністратором), «Прийнято» (схвалено до виконання) та «Відхилено» (Рис. 3.45). Для оперативного інформування адміністратора про надходження нових скарг, у глобальну навігаційну панель інтегровано систему візуальних сповіщень – графічний маркер із динамічним лічильником необроблених запитів.

Центр скарг та пропозицій				
ВІД КОГО	СУТЬ	КОНТЕКСТ	СТАТУС	ДІЇ
ilona_princess	Виправте інформацію про Георгія. Багато неточностей...	Посилання	Прийнято	
andrew25	Токсична поведінка та образи	Посилання	Відхилено	

Рис. 3.45 Прийнята та відхилена скарги

3.3 Тестування роботи програмного комплексу

Тестування програмного забезпечення є критично важливим етапом життєвого циклу розробки, який гарантує відповідність реалізованого функціонала початковим вимогам, а також забезпечує стабільність роботи системи під час реальної експлуатації [30]. Враховуючи сучасні підходи до розробки веб-орієнтованих застосунків, тестування проводилося паралельно з етапами імплементації нового функціонала.

Для розробленого інформаційного середовища було визначено наступні нефункціональні вимоги, які підлягають перевірці (табл. 3.1):

Таблиця 3.1

Список нефункціональних вимог

Характеристика	Значення та детальні вимоги
Інтуїтивність інтерфейсу	Час адаптації нового користувача до базового функціоналу не перевищує 15–20 хвилин; для роботи з панеллю адміністрування – до 1 години.
Час відгуку системи	На стандартні клієнтські запити – не більше 2–3 секунд; для комплексних запитів – до 5 секунд.
Рівень доступності	Очікуваний рівень доступності системи складає не менше 99%.
Технічне обслуговування	Мінімізовано завдяки використанню сучасних CI/CD практик; не перевищує 1% від загального часу експлуатації.

Характеристика	Значення та детальні вимоги
Стійкість до відмов	Допустима норма критичних помилок при взаємодії із сервером – не більше 0,5% від загальної кількості запитів.
Обробка навантаження	Стабільна обробка одночасних запитів від мінімум 20 активних користувачів без деградації часу відповіді.
Гнучкість архітектури	Архітектура передбачає можливість горизонтального масштабування при зростанні навантаження на базу даних та серверну частину.

Для комплексної перевірки розробленого програмного забезпечення було проведено два основні види тестування: тестування швидкості завантаження та навантажувальне тестування серверної частини [9].

Тестування продуктивності клієнтського інтерфейсу проводилося за допомогою інструменту Google PageSpeed Insights. Цей сервіс дозволяє об'єктивно оцінити ключові веб-показники, які безпосередньо впливають на користувацький досвід та ранжування сайту в пошукових системах [8]. Тестування проводилося окремо для десктопної (рис. 3.46) та мобільної (рис. 3.47) версій застосунку.

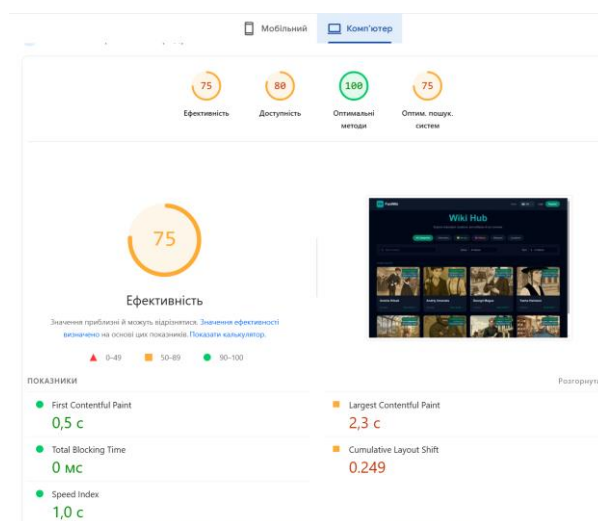


Рис. 3.46 Тестування швидкості завантаження на комп'ютері

		Магаз А. О.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		77

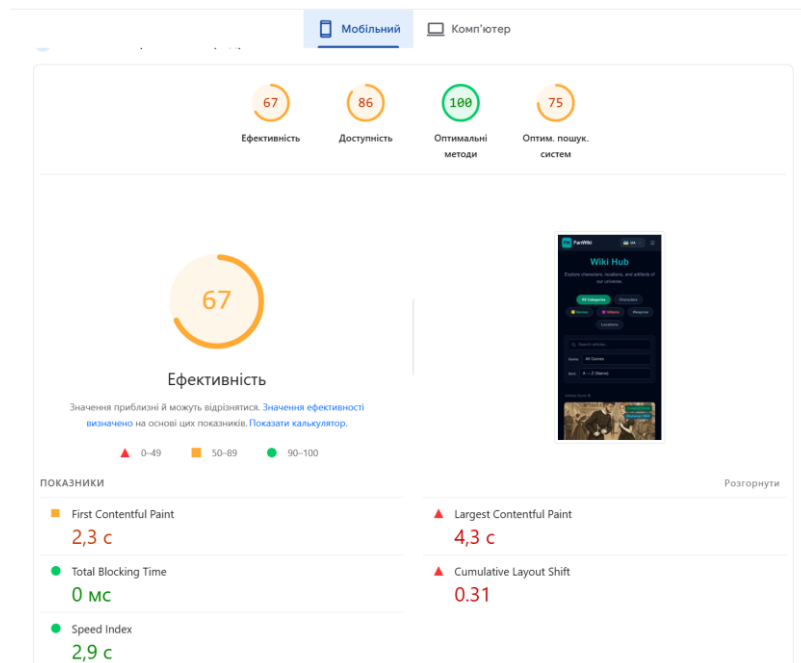


Рис. 3.47 Тестування швидкості завантаження на мобільному пристрої

Клієнтський застосунок отримав 100 балів за критерієм «Оптимальні методи» як на десктопних, так і на мобільних пристроях. Це підтверджує суворе дотримання сучасних стандартів веброзробки, відсутність використання застарілих браузерних API та коректне налаштування клієнтського оточення. Показники доступності (86–88 балів) свідчать про коректну семантику та зручність інтерфейсу. Загальна ефективність (67–75 балів) є прогнозованою та цілком прийнятною для інформаційної системи з великою кількістю графічного контенту. Найважливішим є те, що метрика Total Blocking Time дорівнює 0 мс – це гарантує відсутність зависань та миттєву реакцію застосунку на взаємодію користувача з інтерфейсом.

Для перевірки стабільності роботи серверної частини інформаційного середовища та її відповідності заявленим нефункціональним вимогам (зокрема, підтримка одночасної роботи користувачів), було проведено навантажувальне тестування за допомогою інструментарію Apache JMeter [9].

Тестування проводилося в умовах симуляції одночасного доступу 50 віртуальних користувачів, що в 2,5 раза перевищує мінімальну вимогу системи (20 користувачів).

		Магаз А. О.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		78

Summary Report

Name:Summary Report

Comments:

Write results to file / Read from file

Filename

Browse...

Log/Display Only:

Errors

Successes

Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	50	448	280	1113	190.36	0.00%	4.9/sec	322.70	0.66	67297.5
TOTAL	50	448	280	1113	190.36	0.00%	4.9/sec	322.70	0.66	67297.5

Рис. 3.48 Результати тестування продуктивності

Аналіз отриманого зведеного звіту (Summary Report) дозволяє зробити висновки щодо високої продуктивності розробленого програмного комплексу. Зокрема, під час тестування при обробці 50 одночасних запитів відсоток помилок склав 0.00%, оскільки сервер успішно обробив усі звернення. Це безпосередньо свідчить про надійність обраної архітектури та повну відсутність втрат пакетів під навантаженням. Щодо часових показників, середній час відповіді сервера становив 448 мс, при цьому мінімальна затримка дорівнювала 280 мс, а максимальна у пікові моменти сягала 1113 мс. Отримані результати повністю задовольняють початкову експлуатаційну вимогу, згідно з якою час відгуку системи для стандартних запитів не повинен перевищувати трьох секунд. Крім того, в рамках заданого тестового вікна система продемонструвала стабільну пропускну здатність, успішно обробляючи близько 4.9 запитів на секунду та генеруючи мережевий потік даних обсягом понад 322 КБ/с.

Висновки до третього розділу

Було здійснено практичне розгортання, конфігурацію та комплексну верифікацію працездатності розробленого програмного забезпечення.

На базі сучасних принципів UI/UX-дизайну розроблено структуру адаптивного користувацького інтерфейсу інформаційної системи та визначено ергономічне компонування елементів для забезпечення зручної взаємодії користувачів із платформою. Для забезпечення стабільного функціонування веб-застосунку в реальних умовах хостингу регламентовано чіткий порядок встановлення й налаштування конфігураційних параметрів серверної та клієнтської частин. Окрему увагу приділено нефункціональним показникам

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		79

системи, зокрема алгоритмам оптимізації швидкодії та стійкості архітектури до пікових навантажень.

Також наведено детальний опис та аналіз результатів проведеного тестування за допомогою інструментів Google PageSpeed Insights та Apache JMeter, який підтвердив високу швидкість завантаження інтерфейсу, відмовостійкість серверної частини та нульовий рівень помилок при симуляції одночасної роботи активних користувачів.

		Магаз А. О.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		80

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра було спроектовано та реалізовано інформаційну систему онлайн-спільноти фанатів відеоігор. Для досягнення цієї мети було вирішено наступні завдання:

Проведено дослідження існуючих платформ та аналогів, таких як Fandom, MediaWiki, Notion, Reddit та Steam Guides. Аналіз виявив їхні ключові недоліки, що дозволило визначити концепцію нової системи: поєднання потужної ієрархічної бази знань з відкритим та зручним середовищем для соціальної взаємодії фанатів.

Для реалізації було обрано підхід чистої архітектури, що гарантує високу масштабованість та легкість підтримки коду. Серверна частина розроблена на платформі ASP.NET Core з використанням ORM Entity Framework Core, клієнтська частина імплементована за допомогою бібліотеки React, а для надійного збереження даних обрано СУБД PostgreSQL.

Спроектовано реляційну базу даних, що складається з оптимізованих таблиць, основними з яких є AspNetUsers, Articles, ArticleTranslations, Comments та Reports. Розроблено логіку взаємодії елементів системи, проілюстровану UML-діаграмами варіантів використання та послідовності.

Система забезпечує ефективну роботу різних рівнів користувачів: Гостей, Зареєстрованих учасників та Адміністраторів. Реалізовано функціонал багатомовної локалізації контенту, надійну систему автентифікації, ієрархічну систему коментарів та реакцій, а також повноцінну панель адміністрування з модулем модерації контенту.

Проведено комплексне тестування розробленого програмного комплексу. Перевірка клієнтської частини за допомогою інструменту Google PageSpeed Insights підтвердила високу швидкодію та безпеку застосунку. Навантажувальне тестування серверної частини в Apache JMeter довело здатність бекенду стабільно обробляти запити від 50 одночасно активних віртуальних користувачів із нульовим відсотком помилок.

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		81

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ASP.NET Core Documentation [Електронний ресурс] // Microsoft Learn. – 2025. – Режим доступу: <https://learn.microsoft.com/en-us/aspnet/core/> (дата звернення: 12.05.2026).
2. React Official Documentation [Електронний ресурс] // Meta Platforms. – 2026. – Режим доступу: <https://react.dev/> (дата звернення: 15.05.2026).
3. PostgreSQL Documentation [Електронний ресурс] // PostgreSQL Global Development Group. – 2025. – Режим доступу: <https://www.postgresql.org/docs/> (дата звернення: 18.05.2026).
4. Entity Framework Core Documentation [Електронний ресурс] // Microsoft Learn. – 2025. – Режим доступу: <https://learn.microsoft.com/en-us/ef/core/> (дата звернення: 20.05.2026).
5. C# Programming Guide [Електронний ресурс] // Microsoft Learn. – 2025. – Режим доступу: <https://learn.microsoft.com/en-us/dotnet/csharp/> (дата звернення: 22.05.2026).
6. MDN Web Docs: Web Technologies [Електронний ресурс] // Mozilla. – 2026. – Режим доступу: <https://developer.mozilla.org/> (дата звернення: 25.05.2026).
7. JSON Web Tokens (JWT) Introduction [Електронний ресурс] // Auth0. – 2025. – Режим доступу: <https://jwt.io/introduction> (дата звернення: 01.06.2026).
8. Web Vitals: Essential metrics for a healthy site [Електронний ресурс] // web.dev. – 2024. – Режим доступу: <https://web.dev/vitals/> (дата звернення: 10.06.2026).
9. Apache JMeter User's Manual [Електронний ресурс] // Apache Software Foundation. – 2025. – Режим доступу: <https://jmeter.apache.org/usermanual/> (дата звернення: 12.06.2026).
10. Swagger API Documentation [Електронний ресурс] // SmartBear Software. – 2025. – Режим доступу: <https://swagger.io/docs/> (дата звернення: 05.06.2026).

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		82

11. Clean Architecture in ASP.NET Core [Електронний ресурс] // Microsoft Architecture E-books. – 2024. – Режим доступу: <https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/> (дата звернення: 28.05.2026).
12. Freeman A. Pro ASP.NET Core 7 / A. Freeman. – New York : Apress, 2023. – 864 p.
13. Price M. J. C# 12 and .NET 8 – Modern Cross-Platform Development / M. J. Price. – Birmingham : Packt Publishing, 2023. – 822 p.
14. React Patterns and Best Practices [Електронний ресурс] // React.dev Learn. – 2026. – Режим доступу: <https://react.dev/learn> (дата звернення: 14.06.2026).
15. PostgreSQL 16 Administration Guide [Електронний ресурс] // PostgreSQL Global Development Group. – 2024. – Режим доступу: <https://www.postgresql.org/docs/16/admin.html> (дата звернення: 08.05.2026).
16. Architecting Cloud Native .NET Applications [Електронний ресурс] // Microsoft Docs. – 2025. – Режим доступу: <https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/> (дата звернення: 08.06.2026).
17. JavaScript. The Modern Tutorial [Електронний ресурс] // JavaScript.info. – 2025. – Режим доступу: <https://javascript.info/> (дата звернення: 02.06.2026).
18. Security and Identity in ASP.NET Core [Електронний ресурс] // Microsoft Learn. – 2025. – Режим доступу: <https://learn.microsoft.com/en-us/aspnet/core/security/> (дата звернення: 09.06.2026).
19. User Interface Design Guidelines [Електронний ресурс] // Nielsen Norman Group. – 2024. – Режим доступу: <https://www.nngroup.com/articles/> (дата звернення: 11.06.2026).
20. Мартін Р. Чистий код. Створення, аналіз і рефакторинг / Р. Мартін. – Харків : Фабула, 2019. – 416 с.
21. Гамма Е. Патерни об'єктно-орієнтованого проектування / Е. Гамма, Р. Хелм, Р. Джонсон, Дж. Вліссідес. – Київ : Діалектика, 2022. – 368 с.

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		83

22. Мартін Р. Чиста архітектура. Мистецтво розробки програмного забезпечення / Р. Мартін. – Харків : Фабула, 2021. – 384 с.
23. Richards M. Fundamentals of Software Architecture / M. Richards, N. Ford. – O'Reilly Media, 2020. – 416 p.
24. Usability Engineering in Web Development [Електронний ресурс] // Interaction Design Foundation. – 2025. – Режим доступу: <https://www.interaction-design.org/literature/topics/usability> (дата звернення: 03.06.2026).
25. Axios: Promise based HTTP client [Електронний ресурс] // Axios. – 2026. – Режим доступу: <https://axios-http.com/docs/intro> (дата звернення: 10.06.2026).
26. RESTful API Design Guidelines [Електронний ресурс] // Microsoft Open Source. – 2023. – Режим доступу: <https://github.com/microsoft/api-guidelines> (дата звернення: 07.06.2026).
27. Banks A. Learning React: Modern Patterns for Developing React Apps / A. Banks, E. Porcello. – 2nd ed. – O'Reilly Media, 2020. – 350 p.
28. Git Official Documentation [Електронний ресурс] // Git-SCM. – 2025. – Режим доступу: <https://git-scm.com/doc> (дата звернення: 04.06.2026).
29. CI/CD for ASP.NET Core Apps [Електронний ресурс] // GitHub Actions Documentation. – 2025. – Режим доступу: <https://docs.github.com/en/actions> (дата звернення: 06.06.2026).
30. Майєрс Г. Мистецтво тестування програмного забезпечення / Г. Майєрс. – Київ : Діалектика, 2021. – 272 с.

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		84

ДОДАТКИ

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		85

Program.cs:

```

using System.Text;
using FanWiki.API.Services;
using FanWiki.Application.Services;
using FanWiki.Domain.Entities;
using FanWiki.Infrastructure;
using FanWiki.Infrastructure.Data;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddInfrastructure(builder.Configuration);

builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowAll", policy =>
        policy.WithOrigins("https://fan-wiki-theta.vercel.app")
            .AllowAnyMethod()
            .AllowAnyHeader()
            .AllowCredentials());
});

builder.Services.AddIdentity<ApplicationUser, IdentityRole>(options =>
{
    options.Password.RequireDigit = false;
    options.Password.RequiredLength = 6;
    options.Password.RequireNonAlphanumeric = false;
    options.Password.RequireUppercase = false;
    options.User.RequireUniqueEmail = true;
})

.AddEntityFrameworkStores<AppDbContext>()

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		86

```

        .AddDefaultTokenProviders();

var jwtKey = "SuperSecretKey1234567890_MakeItLonger_MustBe32Chars";
builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme =
JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme =
JwtBearerDefaults.AuthenticationScheme;
}))
.AddJwtBearer(options =>
{
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuer = false,
        ValidateAudience = false,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtKey))
    };
    options.Events = new
Microsoft.AspNetCore.Authentication.JwtBearer.JwtBearerEvents
    {
        OnMessageReceived = context =>
        {
            var accessToken = context.Request.Query["access_token"];
            var path = context.HttpContext.Request.Path;
            if (!string.IsNullOrEmpty(accessToken) &&
path.StartsWithSegments("/hubs"))
            {
                context.Token = accessToken;
            }
            return Task.CompletedTask;
        }
    }
}

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		87

```

        }
    };
});

builder.Services.AddScoped<IWikiService, WikiService>();
builder.Services.AddSingleton<IImageService, CloudinaryImageService>();
builder.Services.AddHttpClient("DeepL", client =>
{
    client.Timeout = TimeSpan.FromSeconds(30);
});
builder.Services.AddScoped<ITranslationService,
DeepLTranslationService>();

builder.Services.AddSignalR();

builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;
    try
    {
        {
            var context = services.GetRequiredService<AppDbContext>();
            await context.Database.MigrateAsync();

            await DbInitializer.SeedAsync(services);
        }
        catch (Exception ex)
        {
            var logger = services.GetRequiredService<ILogger<Program>>();

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		88

```

        logger.LogError(ex, "An error occurred while seeding the
database.");
    }
}

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseCors("AllowAll");

app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();
app.MapHub<FanWiki.API.Hubs.ChatHub>("/hubs/chat");

app.Run();

```

FanWiki.Infrastructure/DependencyInjection.cs:

```

using FanWiki.Application.Interfaces;
using FanWiki.Domain.Interfaces;
using FanWiki.Infrastructure.Data;
using FanWiki.Infrastructure.Repositories;
using FanWiki.Infrastructure.Services;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		89

```

namespace FanWiki.Infrastructure;

public static class DependencyInjection
{
    public static IServiceCollection AddInfrastructure(this
IServiceCollection services, IConfiguration configuration)
    {
        var connectionString =
configuration.GetConnectionString("DefaultConnection");

        services.AddDbContext<AppDbContext>(options =>
            options.UseNpgsql(connectionString));

        services.AddScoped<IArticleRepository, ArticleRepository>();

        services.AddTransient<IEmailSender, SmtpEmailSender>();

        return services;
    }
}

```

FanWiki.API.Controllers/WikiController.cs:

```

using FanWiki.Application.DTOS;
using FanWiki.Application.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace FanWiki.API.Controllers;

[ApiController]
[Route("api/[controller]")]
public class WikiController(IWikiService wikiService, IImageService
imageService) : ControllerBase
{

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		90

```

[HttpGet("{slug}")]
public async Task<IActionResult> Get(string slug, [FromQuery] string
lang = "en", CancellationToken ct = default)
{
    var article = await wikiService.GetArticleAsync(slug, lang, ct);
    if (article is null) return NotFound();
    return Ok(article);
}

```

```

[HttpGet]
public async Task<IActionResult> GetAll(
    [FromQuery] string? category,
    [FromQuery] string? alignment,
    [FromQuery] string sort = "az",
    [FromQuery] string lang = "en",
    [FromQuery] string? game = null,
    CancellationToken ct = default)
{
    var articles = await wikiService.GetAllArticlesAsync(lang,
category, alignment, sort, game, ct);
    return Ok(articles);
}

```

```

[HttpGet("{id:guid}")]
public async Task<IActionResult> GetById(Guid id, [FromQuery] string
lang = "en", CancellationToken ct = default)
{
    var article = await wikiService.GetArticleByIdAsync(id, lang,
ct);
    if (article is null) return NotFound();
    return Ok(article);
}

```

```

[HttpPost]

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		91

```

[Authorize(Roles = "Admin")]
public async Task<IActionResult> Create([FromForm] CreateArticleDto
dto, CancellationToken ct = default)
{
    try
    {
        string? imageUrl = await UploadToCloudinaryAsync(dto.Image,
"images", ct);
        var id = await wikiService.CreateArticleAsync(dto, imageUrl,
ct);
        return CreatedAtAction(nameof(Get), new { slug = dto.Slug },
new { id });
    }
    catch (Exception ex)
    {
        return BadRequest(new { message = ex.Message });
    }
}

[HttpPut("{id:guid}")]
[Authorize(Roles = "Admin")]
public async Task<IActionResult> Update(Guid id, [FromForm]
CreateArticleDto dto, CancellationToken ct = default)
{
    try
    {
        string? imageUrl = await UploadToCloudinaryAsync(dto.Image,
"images", ct);
        await wikiService.UpdateArticleAsync(id, dto, imageUrl, ct);
        return Ok(new { message = "Article updated" });
    }
    catch (Exception ex)
    {
        return BadRequest(new { message = ex.Message, stack =

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		92

```

ex.ToString() });
    }
}

[HttpDelete("{id:guid}")]
[Authorize(Roles = "Admin")]
public async Task<IActionResult> Delete(Guid id, CancellationToken ct
= default)
{
    try
    {
        await wikiService.DeleteArticleAsync(id, ct);
        return Ok(new { message = "Article deleted" });
    }
    catch (Exception ex)
    {
        return BadRequest(new { message = ex.Message });
    }
}

[HttpPost("upload-image")]
[Authorize(Roles = "Admin")]
public async Task<IActionResult> UploadArticleImage(IFormFile file,
CancellationToken ct = default)
{
    if (file == null || file.Length == 0) return BadRequest("No
file");

    using var stream = file.OpenReadStream();
    var url = await imageService.UploadImageAsync(stream,
file.FileName, "articles", ct);

    if (url == null) return StatusCode(500, "Failed to upload
image");
}

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		93

```

        return Ok(new { url });
    }

    private async Task<string?> UploadToCloudinaryAsync(IFormFile? image,
string folder, CancellationToken ct)
    {
        if (image == null) return null;

        using var stream = image.OpenReadStream();
        return await imageService.UploadImageAsync(stream,
image.FileName, folder, ct);
    }
}

```

Додаток Б

client/src/api/axios.ts:

```

import axios from 'axios';
import i18n from '../i18n';

export const api = axios.create({
    baseURL: `${import.meta.env.VITE_API_URL}/api`
});

api.interceptors.request.use((config) => {
    const token = localStorage.getItem('token');
    if (token) {
        config.headers.Authorization = `Bearer ${token}`;
    }

    const separator = config.url?.includes('?') ? '&' : '?';
    config.url = `${config.url}${separator}lang=${i18n.language}`;

    return config;
});

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		94

```

}, (error) => {
    return Promise.reject(error);
});

```

client/sec/features/admin/ArticleEditor.tsx:

```

import { useEffect, useState } from 'react';
import { useNavigate, useParams } from 'react-router-dom';
import { useTranslation } from 'react-i18next';
import { useEditor, EditorContent } from '@tiptap/react';
import StarterKit from '@tiptap/starter-kit';
import { TextStyle } from '@tiptap/extension-text-style';
import TextAlign from '@tiptap/extension-text-align';
import Link from '@tiptap/extension-link';
import { ArticleLinkModal } from './ArticleLinkModal';

import { api } from '../../api/axios';
import type { Article } from '../../types/article';
import { EditorToolbar } from './EditorToolbar';
import { FontSize } from '../../extensions/FontSize';
import { CustomImage } from '../../extensions/CustomImage';
import { extractApiError } from '../../utils/apiError';

type LinkedItem = { name: string; slug?: string };
type ListField = 'family' | 'allies' | 'enemies' | 'alsoKnownAs';

interface ArticleMetadata {
    status?: string;
    gender?: string;
    voiceActor?: string;
    causeOfDeath?: string;
    family?: LinkedItem[];
    allies?: LinkedItem[];
    enemies?: LinkedItem[];
}

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		95

```

    alsoKnownAs?: string[];
    birthDate?: string;
    birthYear?: string | number;
    birthPlace?: string;
    age?: string | number;

    damage?: string | number;
    year?: string | number;
    ammo?: string | number;
    fireRate?: string | number;
    region?: string;
    population?: string | number;
    founded?: string | number;

    [key: string]: string | number | string[] | LinkedItem[] | undefined;
}

```

```

interface EnData {
    title: string;
    quote: string;
    voiceActor: string;
    birthPlace: string;
    birthDate: string;
    causeOfDeath: string;
    familyNames: string[];
    alliesNames: string[];
    enemiesNames: string[];
    alsoKnownAs: string[];
}

```

```

const buildExtensions = () => [
    StarterKit,
    TextStyle,
    FontSize,

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		96

```

    CustomImage,
    TextAlign.configure({ types: ['heading', 'paragraph', 'image'] }),
    Link.configure({
      openOnClick: false,
      HTMLAttributes: { class: 'text-emerald-400 underline hover:text-emerald-300 cursor-pointer' },
    }),
  ];

const editorBaseClass =
  'prose prose-invert prose-sm sm:prose-base max-w-none focus:outline-none min-h-[300px] p-4 text-slate-300 leading-relaxed ' +
  ' [>img]:rounded-xl [>img]:border [>img]:border-slate-700 [>img]:inline-block ' +
  ' [&_figure]:my-4 [&_figure]:mx-auto [&_figure_img]:rounded-xl [&_figure_img]:border [&_figure_img]:border-slate-700 ' +
  ' [&_figcaption]:text-sm [&_figcaption]:text-slate-400 [&_figcaption]:text-center [&_figcaption]:mt-2 [&_figcaption]:italic';

export const ArticleEditor = () => {
  const { id } = useParams();
  const navigate = useNavigate();
  const isEdit = !!id;
  const { t, i18n } = useTranslation();

  const [formData, setFormData] = useState({
    title: '',
    slug: '',
    quote: '',
    content: '',
    category: 'Character',
    alignment: '',
    gameName: '',
    languageCode: 'uk'
  });

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		97

```

});

const [metadata, setMetadata] = useState<ArticleMetadata>({
  status: 'Alive',
  gender: 'Unknown',
  family: [],
  allies: [],
  enemies: [],
  alsoKnownAs: []
});

const [linkingField, setLinkingField] = useState<ListField |
null>(null);

const [linkingIndex, setLinkingIndex] = useState<number |
null>(null);

const [enData, setEnData] = useState<EnData>({
  title: '',
  quote: '',
  voiceActor: '',
  birthPlace: '',
  birthDate: '',
  causeOfDeath: '',
  familyNames: [],
  alliesNames: [],
  enemiesNames: [],
  alsoKnownAs: []
});

const [isTranslating, setIsTranslating] = useState(false);
const [translateError, setTranslateError] = useState<string |
null>(null);

const [existingImage, setExistingImage] = useState<string |
null>(null);

const [file, setFile] = useState<File | null>(null);

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				98
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const editor = useEditor({
  extensions: buildExtensions(),
  content: '',
  editorProps: { attributes: { class: editorBaseClass } },
  onUpdate: ({ editor }) => {
    setFormData(prev => ({ ...prev, content: editor.getHTML()
})));
  },
});

const enEditor = useEditor({
  extensions: buildExtensions(),
  content: '',
  editorProps: { attributes: { class: editorBaseClass + ' min-h-
[200px]' } } },
});

useEffect(() => {
  if (isEdit && id) {
    api.get<Article>(`/Wiki/${id}?lang=uk`).then(res => {
      const data = res.data;
      setFormData({
        title: data.title,
        slug: data.slug,
        quote: data.quote || '',
        content: data.content,
        category: data.category,
        alignment: data.alignment || '',
        gameName: data.gameName || '',
        languageCode: data.languageCode
      });
      setExistingImage(data.imageUrl || null);
    });
  }
});

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		99

```

        if (data.metadata) {
            try {
                const parsed = JSON.parse(data.metadata);
                const normalizeLinked = (arr: any[]):
LinkedItem[] =>
                    (arr || []).map((x: any) => typeof x ===
'string' ? { name: x } : x);
                setMetadata({
                    ...parsed,
                    birthDate: parsed.birthDate ||
(parsed.birthYear ? String(parsed.birthYear) : undefined),
                    family: normalizeLinked(parsed.family),
                    allies: normalizeLinked(parsed.allies),
                    enemies: normalizeLinked(parsed.enemies),
                    alsoKnownAs: parsed.alsoKnownAs || []
                });
            } catch { setMetadata({}); }
        }
        editor?.commands.setContent(data.content);
    });

    api.get<Article>(`/Wiki/${id}?lang=en`).then(res => {
        if (res.data.languageCode !== 'en') return;

        let enMeta: ArticleMetadata = {};
        if (res.data.metadata) {
            try { enMeta = JSON.parse(res.data.metadata); } catch
{ /* ignore */ }
        }
        const normalizeLinked = (arr: any[]): LinkedItem[] =>
            (arr || []).map((x: any) => typeof x === 'string' ? {
name: x } : x);
        const enFamily = normalizeLinked((enMeta.family as
any[]) || []);

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				100
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        const enAllies = normalizeLinked((enMeta.allies as
any[]) || []);
        const enEnemies = normalizeLinked((enMeta.enemies as
any[]) || []);

        setEnData({
            title:      res.data.title,
            quote:      res.data.quote || '',
            voiceActor: (enMeta.voiceActor as string) || '',
            birthPlace: (enMeta.birthPlace as string) || '',
            birthDate:  (enMeta.birthDate as string) || '',
            causeOfDeath: (enMeta.causeOfDeath as string) || '',
            familyNames: enFamily.map(x => x.name),
            alliesNames: enAllies.map(x => x.name),
            enemiesNames: enEnemies.map(x => x.name),
            alsoKnownAs: (enMeta.alsoKnownAs as string[]) || []
        });
        enEditor?.commands.setContent(res.data.content);
    }).catch(() => { /* no English translation yet */ });
}
}, [id, isEdit, editor, enEditor, i18n.language]);

const handleAutoTranslate = async () => {
    setTranslateError(null);
    setIsTranslating(true);
    try {
        const htmlContent = editor?.getHTML() || formData.content;
        const familyNames = ((metadata.family as LinkedItem[]) ||
[]).map(x => x.name).filter(Boolean);
        const alliesNames = ((metadata.allies as LinkedItem[]) ||
[]).map(x => x.name).filter(Boolean);
        const enemiesNames = ((metadata.enemies as LinkedItem[]) ||
[]).map(x => x.name).filter(Boolean);
        const alsoKnownAs = ((metadata.alsoKnownAs as string[]) ||

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				101
Змн.	Арк.	№ докум.	Підпис	Дата		

```
[]).filter(Boolean);
```

```
const res = await api.post<{
  title:      string;
  quote:      string;
  content:    string;
  voiceActor: string | null;
  birthPlace: string | null;
  birthDate:  string | null;
  causeOfDeath: string | null;
  familyNames: string[];
  alliesNames: string[];
  enemiesNames: string[];
  alsoKnownAs: string[];
}>('/Translate/batch', {
  title:      formData.title,
  quote:      formData.quote,
  content:    htmlContent,
  voiceActor: (metadata.voiceActor as string) || null,
  birthPlace: (metadata.birthPlace as string) || null,
  birthDate:  (metadata.birthDate as string) || null,
  causeOfDeath: (metadata.causeOfDeath as string) || null,
  familyNames: familyNames.length > 0 ? familyNames :
null,
  alliesNames: alliesNames.length > 0 ? alliesNames :
null,
  enemiesNames: enemiesNames.length > 0 ? enemiesNames :
null,
  alsoKnownAs: alsoKnownAs.length > 0 ? alsoKnownAs :
null,
  sourceLang: 'uk',
  targetLang: 'en'
});
```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				102
Змн.	Арк.	№ докум.	Підпис	Дата		

```

enEditor?.commands.setContent(res.data.content);

setEnData({
    title:      res.data.title,
    quote:      res.data.quote,
    voiceActor: res.data.voiceActor || '',
    birthPlace: res.data.birthPlace || '',
    birthDate:  res.data.birthDate  || '',
    causeOfDeath: res.data.causeOfDeath || '',
    familyNames: res.data.familyNames || [],
    alliesNames: res.data.alliesNames || [],
    enemiesNames: res.data.enemiesNames || [],
    alsoKnownAs: res.data.alsoKnownAs || []
});
} catch (err: any) {
    console.error(err);
    const status = err?.response?.status;
    if (status === 429) {
        alert('The translation service is currently rate-limited.
Please wait a few seconds and try again.');
```

```

    } else if (status === 503) {
        alert('The translation service is temporarily
unavailable. Please try again later.');
```

```

    } else {
        alert('Translation failed. Please check your connection
and try again.');
```

```

    }
    setTranslateError(t('editor.translate_error'));
} finally {
    setIsTranslating(false);
}
};

const handleCategoryChange = (e:

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				103
Змн.	Арк.	№ докум.	Підпис	Дата		

```

React.ChangeEvent<HTMLSelectElement>) => {
    const newCategory = e.target.value;
    setFormData({ ...formData, category: newCategory });

    let defaults: ArticleMetadata = {};
    if (newCategory === 'Character') {
        defaults = { status: 'Alive', gender: 'Unknown', family: [],
allies: [], enemies: [], alsoKnownAs: [] };
    }
    setMetadata(defaults);
};

const handleListChange = (field: ListField, index: number, name:
string, slug?: string) => {
    if (field === 'alsoKnownAs') {
        const list = [...((metadata.alsoKnownAs as string[]) || [])];
        list[index] = name;
        setMetadata({ ...metadata, alsoKnownAs: list });
    } else {
        const list = [...((metadata[field] as LinkedItem[]) || [])];
        list[index] = { name, slug: slug !== undefined ? slug :
list[index]?.slug };
        setMetadata({ ...metadata, [field]: list });
    }
};

const addListItem = (field: ListField) => {
    if (field === 'alsoKnownAs') {
        const list = [...((metadata.alsoKnownAs as string[]) || [])];
list.push('');
        setMetadata({ ...metadata, alsoKnownAs: list });
    } else {
        const list = [...((metadata[field] as LinkedItem[]) || [])];
list.push({ name: '' });
    }
};

```

```

        setMetadata({ ...metadata, [field]: list });
    }
};

const removeListItem = (field: ListField, index: number) => {
    if (field === 'alsoKnownAs') {
        const list = [...((metadata.alsoKnownAs as string[]) || [])];
list.splice(index, 1);
        setMetadata({ ...metadata, alsoKnownAs: list });
    } else {
        const list = [...((metadata[field] as LinkedItem[]) || [])];
list.splice(index, 1);
        setMetadata({ ...metadata, [field]: list });
    }
};

const renderListInput = (label: string, field: ListField) => {
    const isLinked = field !== 'alsoKnownAs';
    const rawItems = metadata[field] || [];
    return (
        <div className="col-span-1 md:col-span-2 bg-slate-950/50 p-3
rounded border border-slate-700">
            <label className="text-xs text-slate-400 uppercase font-
bold flex justify-between items-center mb-2">
                {label}
                <button type="button" onClick={() =>
addListItem(field)}
                    className="text-emerald-400 hover:text-emerald-
300 text-xs px-2 py-1 bg-emerald-900/30 rounded">
                        {t('editor.add')}
                    </button>
            </label>
            <div className="space-y-2">
                {(rawItems as any[]).map((item, index) => {

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				105
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        const nameVal = isLinked ? (item as
LinkedItem).name : (item as string);
        const slugVal = isLinked ? (item as
LinkedItem).slug : undefined;
        return (
            <div key={index} className="flex gap-2 items-
center">

                <input
                    className="flex-1 bg-slate-900 p-2
rounded border border-slate-600 text-sm focus:border-emerald-500 outline-
none"
                    value={nameVal}
placeholder={` ${label} # ${index + 1} `}
                    onChange={(e) =>
handleListChange(field, index, e.target.value)}
                />
                {isLinked && (
                    <button type="button" title="Link to
article"
                        onClick={() => {
setLinkingField(field); setLinkingIndex(index); }}
                        className={` px-2 py-1 rounded
text-xs border transition ${slugVal ? 'bg-emerald-900/40 border-emerald-
700 text-emerald-400' : 'border-slate-600 text-slate-500 hover:border-
emerald-600 hover:text-emerald-400'} `}>
                            {slugVal ? '🔗' : '🔗'}
                        </button>
                    )}
                <button type="button" onClick={() =>
removeListItem(field, index)}
                    className="text-red-400 hover:text-
red-300 px-2">X</button>
            </div>
        );

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				106
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }}}
    {rawItems.length === 0 && <span className="text-
slate-600 text-xs italic">{t('editor.list_empty')}}</span>}
    </div>
  </div>
);
};

const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault();

  const data = new FormData();
  data.append('Title', formData.title);
  data.append('Slug', formData.slug);
  data.append('Quote', formData.quote);
  data.append('Content', editor?.getHTML() || formData.content);
  data.append('Category', formData.category);
  if (formData.gameName) data.append('GameName',
formData.gameName);
  if (formData.alignment) data.append('Alignment',
formData.alignment);
  data.append('LanguageCode', formData.languageCode);

  const cleanMetadata = { ...metadata };
  if (cleanMetadata.family) cleanMetadata.family =
(cleanMetadata.family as LinkedItem[]).filter(x => x.name.trim() !==
'');
  if (cleanMetadata.allies) cleanMetadata.allies =
(cleanMetadata.allies as LinkedItem[]).filter(x => x.name.trim() !==
'');
  if (cleanMetadata.enemies) cleanMetadata.enemies =
(cleanMetadata.enemies as LinkedItem[]).filter(x => x.name.trim() !==
'');

  if (cleanMetadata.alsoKnownAs) cleanMetadata.alsoKnownAs =

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				107
Змн.	Арк.	№ докум.	Підпис	Дата		

```

(cleanMetadata.alsoKnownAs as string[]).filter(x => x.trim() !== '');
delete cleanMetadata.birthYear;

data.append('Metadata', JSON.stringify(cleanMetadata));
if (file) data.append('Image', file);

const enContent = enEditor?.getHTML() || '';
const hasEnContent = enData.title.trim() !== '' &&
enContent.replace(/<[^>]*>/g, '').trim() !== '';

if (hasEnContent) {
  data.append('TitleEn', enData.title);
  data.append('ContentEn', enContent);
  if (enData.quote) data.append('QuoteEn', enData.quote);

  const enMetadata = { ...cleanMetadata };
  delete enMetadata.causeOfDeath;
  if (enData.voiceActor) enMetadata.voiceActor =
enData.voiceActor;
  if (enData.birthPlace) enMetadata.birthPlace =
enData.birthPlace;
  if (enData.birthDate) enMetadata.birthDate =
enData.birthDate;
  if (enData.causeOfDeath) enMetadata.causeOfDeath =
enData.causeOfDeath;
  if (enData.familyNames.length > 0) {
    const src = (cleanMetadata.family as LinkedItem[]) || [];
    enMetadata.family = enData.familyNames.map((name, i) =>
({ name, slug: src[i]?.slug }));
  }
  if (enData.alliesNames.length > 0) {
    const src = (cleanMetadata.allies as LinkedItem[]) || [];
    enMetadata.allies = enData.alliesNames.map((name, i) =>
({ name, slug: src[i]?.slug }));
  }
}

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				108
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }
    if (enData.enemiesNames.length > 0) {
        const src = (cleanMetadata.enemies as LinkedItem[]) ||
    [];
        enMetadata.enemies = enData.enemiesNames.map((name, i) =>
    ({ name, slug: src[i]?.slug }));
    }
    if (enData.alsoKnownAs.length > 0) enMetadata.alsoKnownAs =
    enData.alsoKnownAs;

    data.append('MetadataEn', JSON.stringify(enMetadata));
}

try {
    if (isEdit) await api.put(`/Wiki/${id}`, data);
    else await api.post('/Wiki', data);
    navigate(`/wiki/${formData.slug}`);
} catch (error: unknown) {
    console.error(error);
    alert(`${t('editor.save_error')}:
    ${extractApiError(error)}`);
}
};

const renderMetadataInputs = () => {
    switch (formData.category) {
        case 'Character':
            return (
                <div className="grid grid-cols-1 md:grid-cols-2 gap-4
    bg-slate-800 p-4 rounded border border-slate-700 animate-in fade-in">
                    <div>
                        <label className="text-xs text-slate-400
    uppercase font-bold">{t('article.status')}</label>
                        <select className="w-full bg-slate-950 p-2

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				109
Змн.	Арк.	№ докум.	Підпис	Дата		

```

rounded border border-slate-600 mt-1 outline-none focus:border-emerald-
500"

        value={metadata.status || 'Alive'}
onChange={e => setMetadata({...metadata, status: e.target.value})}
        <option
value="Alive">{t('meta_values.Alive')}</option>
        <option
value="Deceased">{t('meta_values.Deceased')}</option>
        <option
value="Unknown">{t('meta_values.Unknown')}</option>
        </select>
      </div>
      <div>
        <label className="text-xs text-slate-400
uppercase font-bold">{t('article.gender')}</label>
        <select className="w-full bg-slate-950 p-2
rounded border border-slate-600 mt-1 outline-none focus:border-emerald-
500"
        value={metadata.gender || 'Unknown'}
onChange={e => setMetadata({...metadata, gender: e.target.value})}
        <option
value="Male">{t('meta_values.Male')}</option>
        <option
value="Female">{t('meta_values.Female')}</option>
        <option
value="Unknown">{t('meta_values.Unknown')}</option>
        </select>
      </div>
      {metadata.status === 'Deceased' && (
        <div className="col-span-1 md:col-span-2
animate-in slide-in-from-top-2">
          <label className="text-xs text-red-400
uppercase font-bold">👤 {t('article.cause_of_death')}</label>
          <input

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		110

```

placeholder={t('editor.cause_placeholder')}}
                                className="w-full bg-slate-950 p-2
rounded border border-red-900/50 mt-1 outline-none focus:border-red-500"
                                value={metadata.causeOfDeath || ''}
onChange={e => setMetadata({...metadata, causeOfDeath: e.target.value})}
/>

        </div>
    )}
    <div className="col-span-1 md:col-span-2">
        <label className="text-xs text-slate-400
uppercase font-bold"> 🗣️ {t('editor.voice_actor_label')}}</label>
        <input
placeholder={t('editor.voice_actor_placeholder')}}
                                className="w-full bg-slate-950 p-2
rounded border border-slate-600 mt-1 outline-none focus:border-emerald-
500"
                                value={metadata.voiceActor || ''}
onChange={e => setMetadata({...metadata, voiceActor: e.target.value})} />
        </div>
        <div>
            <label className="text-xs text-slate-400
uppercase font-bold">{t('article.birth_date')}}</label>
            <input
placeholder={t('editor.birth_date_placeholder')}}
                                className="w-full bg-slate-950 p-2
rounded border border-slate-600 mt-1 outline-none focus:border-emerald-
500"
                                value={metadata.birthDate || ''}
onChange={e => setMetadata({...metadata, birthDate: e.target.value})} />
            </div>
            <div>
                <label className="text-xs text-slate-400
uppercase font-bold">{t('article.age')}}</label>
                <input type="number" placeholder="e.g. 35"

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		111

```

        className="w-full bg-slate-950 p-2
rounded border border-slate-600 mt-1 outline-none focus:border-emerald-
500"

        value={metadata.age || ''} onChange={e =>
setMetadata({...metadata, age: e.target.value})} />
      </div>
      <div className="col-span-1 md:col-span-2">
        <label className="text-xs text-slate-400
uppercase font-bold">{t('article.birth_place')}</label>
        <input placeholder={t('article.birth_place')}
        className="w-full bg-slate-950 p-2
rounded border border-slate-600 mt-1 outline-none focus:border-emerald-
500"
        value={metadata.birthPlace || ''}
onChange={e => setMetadata({...metadata, birthPlace: e.target.value})} />
      </div>
      {renderListInput(t('article.family'),
'family')}
      {renderListInput(t('article.allies'),
'allies')}
      {renderListInput(t('article.enemies'),
'enemies')}
      {renderListInput(t('article.also_known_as'),
'alsoKnownAs')}
    </div>
  );
  case 'Weapon':
    return (
      <div className="grid grid-cols-2 gap-4 bg-slate-800
p-4 rounded border border-slate-700 animate-in fade-in">
        <div><label className="text-xs text-slate-400
uppercase font-bold">{t('article.damage')}</label>
        <input placeholder="0" type="number"
        className="w-full bg-slate-950 p-2 rounded border border-slate-600 mt-1

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				112
Змн.	Арк.	№ докум.	Підпис	Дата		

```

outline-none focus:border-emerald-500"
                value={metadata.damage || ''} onChange={e
=> setMetadata({...metadata, damage: e.target.value})} /></div>
                <div><label className="text-xs text-slate-400
uppercase font-bold">{t('article.year')}</label>
                <input placeholder={t('article.year')}
type="number" className="w-full bg-slate-950 p-2 rounded border border-
slate-600 mt-1 outline-none focus:border-emerald-500"
                value={metadata.year || ''} onChange={e
=> setMetadata({...metadata, year: e.target.value})} /></div>
                <div><label className="text-xs text-slate-400
uppercase font-bold">{t('article.ammo')}</label>
                <input placeholder={t('article.ammo')}
type="number" className="w-full bg-slate-950 p-2 rounded border border-
slate-600 mt-1 outline-none focus:border-emerald-500"
                value={metadata.ammo || ''} onChange={e
=> setMetadata({...metadata, ammo: e.target.value})} /></div>
                <div><label className="text-xs text-slate-400
uppercase font-bold">{t('article.fire_rate')}</label>
                <input placeholder={t('editor.rate_of_fire')}
type="number" className="w-full bg-slate-950 p-2 rounded border border-
slate-600 mt-1 outline-none focus:border-emerald-500"
                value={metadata.fireRate || ''}
onChange={e => setMetadata({...metadata, fireRate: e.target.value})}
/></div>
            </div>
        );
        case 'Location':
            return (
                <div className="grid grid-cols-2 gap-4 bg-slate-800
p-4 rounded border border-slate-700 animate-in fade-in">
                    <div><label className="text-xs text-slate-400
uppercase font-bold">{t('article.region')}</label>
                    <input

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		113

```

placeholder={t('editor.region_placeholder')} className="w-full bg-slate-
950 p-2 rounded border border-slate-600 mt-1 outline-none focus:border-
emerald-500"

                                value={metadata.region || ''} onChange={e
=> setMetadata({...metadata, region: e.target.value})} /></div>
                                <div><label className="text-xs text-slate-400
uppercase font-bold">{t('article.population')}

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		114

```

<form onSubmit={handleSubmit} className="space-y-8">
  {/* — Primary language section — */}
  <div className="bg-slate-900 p-6 rounded-xl border
border-slate-800 shadow-lg space-y-6">
    <h2 className="text-xl font-bold text-slate-200 mb-4
flex items-center gap-2">
       {t('editor.info_section')}
      ({formData.languageCode.toUpperCase()})
    </h2>

    <div className="grid grid-cols-1 md:grid-cols-2 gap-
6">

      <div>
        <label className="block text-xs font-bold
text-slate-400 uppercase mb-2">{t('editor.label_title')}</label>
        <input required className="bg-slate-950 p-3
rounded border border-slate-700 w-full text-white font-bold text-lg"
          value={formData.title} onChange={e =>
setFormData({...formData, title: e.target.value})} />
      </div>
      <div>
        <label className="block text-xs font-bold
text-slate-400 uppercase mb-2">{t('editor.label_slug')}</label>
        <input required className="bg-slate-950 p-3
rounded border border-slate-700 w-full text-emerald-400 font-mono text-
sm"
          value={formData.slug} onChange={e =>
setFormData({...formData, slug: e.target.value})} />
      </div>
    </div>

    <div>
      <label className="block text-xs font-bold text-
slate-400 uppercase mb-2">{t('editor.label_quote')}</label>

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		115

```

        <textarea rows={2}
placeholder={t('editor.quote_placeholder')}
        className="bg-slate-950 p-3 rounded border
border-slate-700 w-full text-emerald-200 italic focus:border-emerald-500
outline-none resize-none"
        value={formData.quote} onChange={e =>
setFormData({...formData, quote: e.target.value})} />
    </div>

    <div className="grid grid-cols-1 md:grid-cols-2 gap-
6">

        <div>
            <label className="block text-xs font-bold
text-slate-400 uppercase mb-2">{t('article.category')}</label>
            <select className="bg-slate-950 p-3 rounded
border border-slate-700 w-full text-white"
                value={formData.category}
                onChange={handleCategoryChange}>
                <option
value="Character">{t('categories.Character')}</option>
                <option
value="Weapon">{t('categories.Weapon')}</option>
                <option
value="Location">{t('categories.Location')}</option>
                <option
value="Event">{t('categories.Event')}</option>
            </select>
        </div>
        {formData.category === 'Character' && (
            <div className="animate-in fade-in slide-in-
from-left-2">

                <label className="block text-xs font-bold
text-slate-400 uppercase mb-2">{t('editor.label_alignment')}</label>
                <select className="bg-slate-950 p-3

```

```

rounded border border-slate-700 w-full text-white focus:border-emerald-
500"

                                value={formData.alignment}
onChange={e => setFormData({...formData, alignment: e.target.value})}>
                                <option
value="">{t('editor.alignment_none')}</option>
                                <option value="Positive">😊
{t('editor.alignment_positive')}</option>
                                <option value="Negative">😬
{t('editor.alignment_negative')}</option>
                                </select>
                                </div>
                                )}
                                </div>

                                <div>
                                <label className="block text-xs font-bold text-
slate-400 uppercase mb-2">{t('article.game')}</label>
                                <input
placeholder={t('article.game_placeholder')}
                                className="bg-slate-950 p-3 rounded border
border-slate-700 w-full text-white focus:border-emerald-500 outline-none"
                                value={formData.gameName} onChange={e =>
setFormData({...formData, gameName: e.target.value})} />
                                </div>

                                {renderMetadataInputs()}
                                </div>

                                {/* — Ukrainian content editor — */}
                                <div className="bg-slate-900 rounded-xl border border-
slate-800 shadow-lg">
                                <div className="p-4 border-b border-slate-800 bg-
slate-950/50 rounded-t-xl">

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				117
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        <label className="block text-xs font-bold text-
slate-400 uppercase">{t('editor.label_content')}

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		118

```

</div>

{translateError && (
  <p className="text-red-400 text-sm bg-red-950/40
border border-red-800 px-4 py-2 rounded">{translateError}</p>
)}

<div>
  <label className="block text-xs font-bold text-
slate-400 uppercase mb-2">{t('editor.label_title_en')}</label>
  <input className="bg-slate-950 p-3 rounded border
border-slate-700 w-full text-white font-bold text-lg focus:border-cyan-
500 outline-none"
    value={enData.title} placeholder="English
title..."
    onChange={e => setEnData(prev => ({ ...prev,
title: e.target.value })))} />
</div>

<div>
  <label className="block text-xs font-bold text-
slate-400 uppercase mb-2">{t('editor.label_quote_en')}</label>
  <textarea rows={2}
    className="bg-slate-950 p-3 rounded border
border-slate-700 w-full text-cyan-200 italic focus:border-cyan-500
outline-none resize-none"
    value={enData.quote} placeholder="English
quote..."
    onChange={e => setEnData(prev => ({ ...prev,
quote: e.target.value })))} />
</div>

{/* Character metadata for English */}
{formData.category === 'Character' && (

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		119

```


|      |      |               |        |      |                     |      |
|------|------|---------------|--------|------|---------------------|------|
|      |      | Магаз А. О.   |        |      | ІІЗ.КР.Б-121-26-ІІЗ | Арк. |
|      |      | Єфремов Ю. М. |        |      |                     | 120  |
| Змн. | Арк. | № докум.      | Підпис | Дата |                     |      |


```

```

                                onChange={e => setEnData(prev => ({
...prev, birthPlace: e.target.value })))} />
                                </div>
                                {metadata.status === 'Deceased' && (
                                    <div className="col-span-full">
                                        <label className="text-xs text-red-
400 uppercase font-bold">☠ {t('article.cause_of_death')} (EN)</label>
                                        <input className="w-full bg-slate-900
p-2 rounded border border-slate-600 mt-1 text-sm focus:border-cyan-500
outline-none"
                                            value={enData.causeOfDeath}
placeholder="English cause of death..."
                                            onChange={e => setEnData(prev =>
({ ...prev, causeOfDeath: e.target.value })))} />
                                        </div>
                                    )}
                                {enData.familyNames.length > 0 && (
                                    <div className="col-span-full">
                                        <label className="text-xs text-slate-
400 uppercase font-bold">{t('article.family')} (EN)</label>
                                        <div className="space-y-1 mt-1">
                                            {enData.familyNames.map((name, i)
=> (
                                                <input key={i} className="w-
full bg-slate-900 p-2 rounded border border-slate-600 text-sm
focus:border-cyan-500 outline-none"
                                                    value={name} onChange={e
=> { const u = [...enData.familyNames]; u[i] = e.target.value;
setEnData(prev => ({ ...prev, familyNames: u })); }} />
                                                )}}
                                            </div>
                                        </div>
                                    )}
                                {enData.alliesNames.length > 0 && (

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		121

```

        <div className="col-span-full">
            <label className="text-xs text-slate-
400 uppercase font-bold">{t('article.allies')} (EN)</label>
            <div className="space-y-1 mt-1">
                {enData.alliesNames.map((name, i)
=> (
                    <input key={i} className="w-
full bg-slate-900 p-2 rounded border border-slate-600 text-sm
focus:border-cyan-500 outline-none"
                        value={name} onChange={e
=> { const u = [...enData.alliesNames]; u[i] = e.target.value;
setEnData(prev => ({ ...prev, alliesNames: u })); }} />
                    )}}
            </div>
        </div>
    )}
    {enData.enemiesNames.length > 0 && (
        <div className="col-span-full">
            <label className="text-xs text-slate-
400 uppercase font-bold">{t('article.enemies')} (EN)</label>
            <div className="space-y-1 mt-1">
                {enData.enemiesNames.map((name,
i) => (
                    <input key={i} className="w-
full bg-slate-900 p-2 rounded border border-slate-600 text-sm
focus:border-cyan-500 outline-none"
                        value={name} onChange={e
=> { const u = [...enData.enemiesNames]; u[i] = e.target.value;
setEnData(prev => ({ ...prev, enemiesNames: u })); }} />
                    )}}
            </div>
        </div>
    )}
    {enData.alsoKnownAs.length > 0 && (

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		122

```

        <div className="col-span-full">
            <label className="text-xs text-slate-
400 uppercase font-bold">{t('article.also_known_as')} (EN)</label>
            <div className="space-y-1 mt-1">
                {enData.alsoKnownAs.map((name, i)
=> (
                    <input key={i} className="w-
full bg-slate-900 p-2 rounded border border-slate-600 text-sm
focus:border-cyan-500 outline-none"
                        value={name} onChange={e
=> { const u = [...enData.alsoKnownAs]; u[i] = e.target.value;
setEnData(prev => ({ ...prev, alsoKnownAs: u })); }} />
                    )}}
            </div>
        </div>
    )}
</div>
)}

{/* — English WYSIWYG content editor — */}
<div>
    <label className="block text-xs font-bold text-
slate-400 uppercase mb-2">{t('editor.label_content_en')}</label>
    <div className="bg-slate-900 rounded-xl border
border-cyan-900/50 shadow-lg overflow-hidden">
        <EditorToolbar
            editor={enEditor}
            className="border-b border-slate-700 p-2
flex flex-wrap gap-1 bg-slate-950/80 items-center"
        />
        <EditorContent editor={enEditor} />
    </div>
</div>
</div>

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				123
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    { /* — Image — */ }
    <div className="bg-slate-900 p-6 rounded-xl border
border-slate-800 shadow-lg">
      <h2 className="text-xl font-bold text-slate-200 mb-4
flex items-center gap-2">🖼️ {t('editor.label_image')}</h2>
      <div className="border-2 border-slate-700 border-
dashed p-8 rounded-lg text-center hover:bg-slate-800/50 transition
cursor-pointer relative">
        <input type="file" className="absolute inset-0 w-
full h-full opacity-0 cursor-pointer"
          onChange={e => setFile(e.target.files?.[0] ||
null)} />
        <p className="text-lg font-medium text-slate-
300">{t('editor.click_to_upload')}</p>
      </div>
      {(file || existingImage) && (
        <div className="mt-6 p-4 bg-slate-950 rounded-lg
border border-slate-800 flex gap-4 items-center animate-in fade-in">
          {file ? (
            <div className="w-20 h-20 bg-slate-800
rounded flex items-center justify-center text-slate-500 text-
xs">{t('editor.new_file')}</div>
            ) : existingImage && (
              <img src={existingImage} alt="Current"
className="w-20 h-20 rounded object-cover border border-slate-700"/>
            )}
          <div>
            <p className="font-bold text-slate-
200">{t('editor.selected')}</p>
            <p className="text-sm text-emerald-400
break-all">{file ? file.name : existingImage}</p>
          </div>
        </div>
      )}
    </div>

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		124

```

    })
  </div>
</form>

{/* — Sticky save bar — */}
<div className="fixed bottom-0 left-0 right-0 z-30 bg-slate-
950/90 backdrop-blur-md border-t border-slate-800 p-4">
  <div className="max-w-5xl mx-auto flex justify-end">
    <button onClick={handleSubmit as any}
      className="bg-emerald-600 hover:bg-emerald-500
px-10 py-3 rounded-xl font-bold text-white transition shadow-lg shadow-
emerald-900/30">

      {t('editor.save')}
    ({formData.languageCode.toUpperCase()})
  </button>
</div>
</div>

{linkingField && linkingIndex !== null && (
  <ArticleLinkModal
    onSelect={(slug, title) => {
      handleListChange(linkingField, linkingIndex,
title, slug);

      setLinkingField(null);
      setLinkingIndex(null);
    }}
    onClose={() => { setLinkingField(null);
setLinkingIndex(null); }}
  />
)}
</div>

);
};

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		125

client/sec/features/admin/ArticleEditor.tsx:

```
import { useEffect, useState } from 'react';
import { api } from '../../api/axios';
import type { Article } from '../../types/article';
import { useNavigate } from 'react-router-dom';
import { useTranslation } from 'react-i18next';

export const AdminDashboard = () => {
  const [articles, setArticles] = useState<Article[]>([]);
  const navigate = useNavigate();
  const { t } = useTranslation();

  useEffect(() => {
    const fetchArticles = async () => {
      try {
        const res = await api.get<Article[]>('/Wiki');
        setArticles(res.data);
      } catch (err) {
        console.error("Failed to fetch articles", err);
      }
    };

    fetchArticles();
  }, []);

  const handleDelete = async (id: string) => {
    if (!window.confirm(t('admin.delete_confirm'))) return;
    try {
      await api.delete(`/Wiki/${id}`);
      setArticles(prev => prev.filter(a => a.id !== id));
    } catch (error) {
      console.error(error);
      alert(t('admin.delete_error'));
    }
  }
}
```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				126
Змн.	Арк.	№ докум.	Підпис	Дата		

```

};

return (
  <div className="max-w-6xl mx-auto p-6">
    <div className="flex justify-between items-center mb-8">
      <h1 className="text-3xl font-bold text-slate-100">{t('admin.dashboard_title')}</h1>
      <button
        onClick={() => navigate('/admin/create')}
        className="bg-emerald-600 hover:bg-emerald-500 px-4 py-2 rounded font-bold text-white transition"
      >
        {t('admin.new_article')}
      </button>
    </div>

    <div className="bg-slate-900 rounded-lg border border-slate-800 overflow-hidden">
      <table className="w-full text-left text-slate-300">
        <thead className="bg-slate-950 text-slate-400 uppercase text-xs font-bold">
          <tr>
            <th className="p-4">{t('admin.table_title')}</th>
            <th className="p-4">{t('admin.table_slug')}</th>
            <th className="p-4">{t('admin.table_category')}</th>
            <th className="p-4 text-right">{t('admin.table_actions')}</th>
          </tr>
        </thead>
        <tbody className="divide-y divide-slate-800">
          {articles.map(article => (
            <tr key={article.id} className="hover:bg-slate-800/50 transition">

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		127

```

<td className="p-4 font-bold text-
white">{article.title}</td>
<td className="p-4 text-emerald-
400">{article.slug}</td>
<td className="p-4">
    <span className="bg-slate-800 px-2
py-1 rounded text-xs border border-slate-700">
        {t(`categories.${article.category}`)}
    </span>
</td>
<td className="p-4 text-right space-x-2">
    <button
        onClick={() =>
navigate(`/admin/edit/${article.id}`)}
        className="text-blue-400 hover:text-
blue-300 text-sm font-bold"
    >
        {t('admin.edit')}
    </button>
    <button
        onClick={() => article.id &&
handleDelete(article.id)}
        className="text-red-500 hover:text-
red-400 text-sm font-bold"
    >
        {t('admin.delete')}
    </button>
</td>
</tr>
    )})
</tbody>
</table>
</div>

```

		Магаз А. О.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Єфремов Ю. М.				128
Змн.	Арк.	№ докум.	Підпис	Дата		

```
        </div>
    );
};
```

		Магаз А. О.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		129