МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____»_____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ІНФОРМАЦІЙНА ВЕБСИСТЕМА БЛАГОДІЙНОЇ ОРГАНІЗАЦІЇ З АДМІНІСТРАТИВНИМИ ТА СОЦІАЛЬНИМИ СЕРВІСАМИ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____ Віктор СКВОРЦОВ

«___»_____ 2026 р.

Керівник д-р техн. наук, професор _____ Олександр ГОЖИЙ

«___»_____ 2026 р.

Миколаїв – 2026

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

Факультет	Комп’ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп’ютерні науки
Освітня програма	Комп’ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Скворцов Віктор Андрійович
(прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Інформаційна вебсистема благодійної організації з адміністративними та соціальними сервісами».

Керівник роботи: Гожий Олександр Петрович, д-р техн. наук, професор

Затверджена наказом ЧНУ ім. Петра Могили від «25» грудня 2025 р. № 353.

2. Строк представлення кваліфікаційної роботи « » червня 2026 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: Інформаційна вебсистема благодійної організації з адміністративними та соціальними сервісами; вимоги до функціональності системи, перелік технологій розробки, структура контенту організації та ролі користувачів системи.

4. Перелік питань, що підлягають розробці: аналіз сучасного стану інформаційного забезпечення діяльності благодійних організацій та огляд існуючих аналогів; дослідження методів та технологій розробки захищених вебсистем з розмежуванням доступу; проектування архітектури інформаційної вебсистеми з

адміністративними та соціальними сервісами; розробка серверної частини системи на основі Django REST Framework з реалізацією автентифікації, двофакторного захисту та рольової моделі доступу; розробка клієнтської частини на основі Next.js з підтримкою серверного рендерингу та CMS-конструктора сторінок; інтеграція хмарної інфраструктури CloudFlare для забезпечення мережевої безпеки та доставки контенту; аналіз отриманих результатів та оцінка відповідності реалізованої системи функціональним вимогам.

5. Перелік графічних матеріалів: презентація.

Керівник роботи _____ __Олександр ГОЖИЙ__
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ __Віктор СКВОРЦОВ__
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «21» грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

кваліфікаційної роботи

Тема: «Інформаційна вебсистема благодійної організації з адміністративними та соціальними сервісами».

№	Найменування роботи	Початок	Закінчення	Примітки
1	Вибір теми та затвердження з керівником	01.10.2025	01.11.2025	Виконано
2	Подання теми на затвердження кафедри	01.11.2025	20.11.2025	Виконано
3	Отримання завдання КР	21.12.2025	21.12.2025	Виконано
4	Складання календарного плану	29.01.2026	29.01.2026	Виконано
5	Аналіз предметної сфери (благодійні організації, наявні аналоги)	01.02.2026	14.02.2026	Виконано
6	Постановка задачі, визначення мети, об'єкту та предмету	14.02.2026	21.02.2026	Виконано
7	Огляд методів, технологій	21.02.2026	07.03.2026	Виконано
8	Розробка бекенду (Django REST Framework, 2FA, CMS). Інтеграція CloudFlare.	07.03.2026	04.04.2026	Виконано
9	Розробка фронтенду (Next.js, CMS)	04.04.2026	25.04.2026	Виконано
10	Тестування системи	25.04.2026	05.05.26	Виконано
11	Оформлення текстової частини КР			Виконано
12	Перший попередній захист	25.05.2026	25.05.2026	Виконано
13	Доопрацювання за зауваженнями комісії	26.05.2026	04.06.2026	Виконано
14	Другий попередній захист	05.06.2026	05.06.2026	Виконано
15	Доробка та остаточне оформлення КР	06.06.2026	14.06.2026	Виконано
16	Подання КР завідувачу кафедри	15.06.2026	19.06.2026	Виконано

Керівник роботи _____ Олександр ГОЖИЙ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Віктор СКВОРЦОВ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«29» січня 2026 р.

АННОТАЦІЯ

до кваліфікаційної роботи
здобувача групи 401 ЧНУ ім. Петра Могили

Скворцова Віктора Андрійовича

на тему **“ІНФОРМАЦІЙНА ВЕБСИСТЕМА БЛАГОДІЙНОЇ ОРГАНІЗАЦІЇ З АДМІНІСТРАТИВНИМИ ТА СОЦІАЛЬНИМИ СЕРВІСАМИ”**

Предметом роботи є технології та засоби розробки вебсистем з адміністративними та соціальними сервісами для благодійних організацій. **Об’єктом роботи** є процеси інформаційного забезпечення діяльності благодійних організацій. **Метою роботи** є проектування та розробка інформаційної вебсистеми, що забезпечує автоматизацію адміністративних процесів і надання соціальних сервісів благодійної організації. Методи дослідження: системний аналіз предметної сфери, об'єктно-орієнтоване проектування, REST-архітектура, методи захисту інформації та двофакторної автентифікації.

Кваліфікаційна робота складається з чотирьох розділів. У першому розділі проведено аналіз предметної сфери діяльності благодійних організацій, виконано огляд наявних аналогів інформаційних систем та сформульовано постановку задачі. У другому розділі розглянуто моделі, методи та інформаційні технології для вирішення поставленої задачі, У третьому розділі описано проектування та розробку інформаційної системи, наведено структурні та функціональні схеми, діаграми компонентів і бази даних. У четвертому розділі представлено програмну реалізацію системи, керівництво користувача та результати тестування.

Сторінок – 88, таблиць – 16, рисунків – 30, додатків – 2, джерел – 34.

Ключові слова: *вебсистема, благодійна організація, Django REST Framework, Next.js, двофакторна автентифікація, CloudFlare, система управління контентом, адміністративний сервіс, соціальний сервіс, REST API.*

ABSTRACT

to the qualification work by the student of the group 401 of Petro Mohyla Black Sea National University

Viktor Skvortsov

“INFORMATION WEB SYSTEM OF A CHARITABLE ORGANIZATION WITH ADMINISTRATIVE AND SOCIAL SERVICES”

The subject of the work is the methods and means of developing web systems with administrative and social services for charitable organizations. **The object of the work** is the processes of information support for the activities of charitable organizations. **The purpose of the work** is to design and develop an information web system that automates administrative processes and provides social services for a charitable organization. Research methods include systematic analysis of the subject area, object-oriented design, REST architecture, information security methods, and two-factor authentication.

The qualification work consists of four sections. The first section analyzes the subject area of charitable organizations, reviews existing analogues of information systems, and formulates the problem statement. The second section examines models, methods, and information technologies for solving the defined task, particularly Django REST Framework, Next.js, CloudFlare, and two-factor authentication tools. The third section describes the design and development of the information system, presents structural and functional diagrams, component and database diagrams, and analyzes the obtained results. The fourth section presents the software implementation of the system, a user guide, and testing results.

Pages – 88, tables – 16, figures – 30, appendices – 2, references – 34.

Keywords: *web system, charitable organization, Django REST Framework, Next.js, two-factor authentication, CloudFlare, content management system, administrative service, social service, REST API.*

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД – база даних
БО – благодійна організація
ІС – інформаційна система
ІТ – інформаційні технології
КР – кваліфікаційна робота
ПЗ – програмне забезпечення
СУК – система управління контентом
НУО – неурядова/недержавна організація

2FA – Two-Factor Authentication
API – Application Programming Interface
CDN – Content Delivery Network
CMS – Content Management System
CORS – Cross-Origin Resource Sharing
CRUD – Create, Read, Update, Delete
DRF – Django REST Framework
HTTPS – HyperText Transfer Protocol Secure
JSON – JavaScript Object Notation
ORM – Object-Relational Mapping
REST – Representational State Transfer
SQL – Structured Query Language
SSR – Server-Side Rendering
URL – Uniform Resource Locator

ВСТУП

У сучасному цифровому середовищі інформаційні технології відіграють важливу роль у підвищенні ефективності та прозорості діяльності благодійних організацій (БО). Зростання некомерційного сектору та підвищення суспільних вимог до звітності й доступності інформації зумовлюють потребу у спеціалізованих вебплатформах, здатних підтримувати як внутрішні процеси управління, так і взаємодію з громадськістю.

Благодійні організації функціонують у динамічному середовищі, де необхідно керувати збором коштів, координувати волонтерів, взаємодіяти з бенефіціарами та забезпечувати прозорість для донорів. Відсутність єдиної інформаційної системи часто призводить до розрізненості даних і зниження ефективності роботи. Тому розробка інтегрованої інформаційної системи, адаптованої до потреб БО, є актуальним завданням у сфері інформаційних технологій.

Актуальність цієї **роботи** підтверджується останніми академічними дослідженнями. Дослідження, опубліковані в рецензованих журналах, індексованих MDPI та Scopus, підтверджують, що цифрова модифікація некомерційного сектору покращує залучення донорів, утримання волонтерів і загальний організаційний вплив.

Предметом роботи є технології, які використовуються для розробки веб-інформаційних систем з функціями адміністративних та соціальних послуг. **Об’єктом роботи** є процеси інформаційного забезпечення благодійних організацій.

Метою даної **роботи** є проектування та розробка повнофункціональної інформаційної вебсистеми для благодійної організації, яка автоматизує ключові адміністративні процеси, забезпечує безпеку даних та продуктивності системи.

Для досягнення цієї мети визначено наступні **завдання**:

– проаналізувати предметну область благодійних організацій та виявити основні функціональні вимоги до їх інформаційних систем;

– переглянути та порівняти існуючі аналоги та відповідні наукові публікації;

– вибрати та обґрунтувати відповідні методи, технології та архітектурні рішення для розробки системи;

– розробити архітектуру системи, включаючи структуру бази даних, діаграми компонентів і моделі потоку даних;

– реалізувати бекенд REST API за допомогою Django REST Framework з підтримкою двофакторної автентифікації;

– для розробки інтерфейсної програми з використанням Next.js з інтеграцією CMS;

– провести функціональне тестування розробленої системи та оцінити результати.

Декларація про використання ШІ. Під чат підготовки наукової роботи було використано інструмент Claude 3. Розкриття факту делегування завдань генеративному ШІ

Автори заявляють про використання генеративного ШІ у процесі дослідження та підготовки рукопису. Відповідно до таксономії GAIDeT (2025), наведені нижче завдання були делеговані інструментам генеративного ШІ за повного людського нагляду:

– Пошук і систематизація літератури

– Вичитування та редагування

– Адаптація та коригування емоційного тону

– Реформатування

Використаний інструмент генеративного ШІ: Claude 3.

Повну відповідальність за фінальний рукопис несуть автори.

Інструменти генеративного ШІ не зазначаються як автори та не несуть відповідальності за кінцеві результати.

Декларацію подав: Скворцов Віктор Андрійович.

1 АНАЛІЗ ІНФОРМАЦІЙНИХ ВЕБСИСТЕМ БЛАГОДІЙНИХ ОРГАНІЗАЦІЙ ТА ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ

Стрімкий прогрес інформаційних технологій кардинально змінює методи організації роботи некомерційних об’єднань, зокрема благодійних структур. Збільшення їхньої кількості на глобальному ринку, ускладнення внутрішніх операцій та зростання суспільних очікувань та потреб щодо відкритості та звітності зумовлюють потребу у впровадженні актуальних цифрових інструментів. Вебплатформи перетворюються на головний засіб комунікації між благодійними організаціями, меценатами, добровольцями та потребуючими допомоги, гарантуючи легкість доступу до даних, автоматизацію управлінських завдань та зростання залученості цільової аудиторії.

У цьому розділі здійснено аналіз діяльності благодійних організацій, досліджено нюанси їхнього інформаційного супроводу, проведено огляд існуючих вебаналогів та наукових праць, а також сформульовано технічне завдання для створення інформаційної вебсистеми з адміністративними та соціальними функціями.

1.1 Опис предметної сфери діяльності благодійних організацій

Загальна характеристика та роль благодійних організацій.

Благодійні установи належать до «третього сектору» – сукупності некомерційних структур, які не входять ні до державного, ні до приватного економічного сегментів. Згідно з визначенням ООН, неурядова організація (НУО) є «будь-якою некомерційною добровільною громадянською спільнотою, створеною на місцевому, національному чи міжнародному рівні» [1]. Такі структури вирішують завдання щодо задоволення суспільних потреб, зокрема: забезпечення освітніх послуг та охорони здоров’я, надання гуманітарної підтримки, екологічного захисту та допомоги соціально вразливим верствам населення.

Науковці виділяють низку обов’язкових ознак для суб’єктів третього сектору: формалізований та добровільний характер діяльності; інституційна, політична й фінансова автономія від держави та бізнесу; орієнтація на суспільне, політичне, економічне або розвиткове благо; некомерційний статус (відсутність прибутку як головної мети); а також робота на місцевому, національному або міжнародному рівнях [1]. Ці параметри фундаментально відрізняють благодійні організації від комерційних підприємств і державних установ, що суттєво впливає на вимоги до їхніх інформаційних систем.

Станом на травень 2023 року ООН зареєструвала 14 955 організацій громадянського суспільства, з яких 12 516 є безпосередньо НУО, а майже половина (48,51%) функціонують у Африці [1]. Найпоширенішими є структури, які займаються економічною та соціальною діяльністю, сталим розвитком та питаннями гендерної рівності. Така масштабність і різноманітність діяльності БО формують складне підґрунтя організаційних процесів, ефективне управління яким є неможливим без використання сучасних інформаційних технологій.

Для досягнення своїх цілей – привернення уваги до соціальних проблем, мобілізації громадян, збору коштів та управління проєктами – БО потребують ефективних комунікаційних та управлінських інструментів. Як зазначається у дослідженні [1], цифровий простір суттєво посприяв становленню іміджу НУО, дозволяючи збільшувати зв'язки з різними аудиторіями, підтримувати їх інформованість, посилювати громадянську мобілізацію та полегшувати збір коштів. Таким чином, якість вебплатформи безпосередньо впливає на ефективність діяльності організації та її здатність досягати місії.

Стан цифровізації та вебкомунікацій у секторі БО.

Впроваження технологій Web 2.0 не тільки надала некомерційним структурам потужний інструмент поширення знань, а й кардинально покращила комунікацію між ними та різними групами споживачів [1]. Онлайн-платформи та сторінки у соцмережах створюють найкращі умови для реалізації місії та освітніх завдань БО.

Аналітична робота *Moreno-Cabanillas et al.* [1], опублікована у виданні Future Internet (MDPI, 2024), представляє всебічне дослідження цифрових комунікаційних стратегій тридцяти найбільш впливових НУО світу. Вчені поділяють всі засоби вебкомунікації на дві кардинально відмінні групи: односторонні (монологічні) канали, що спрямовують інформацію від організації до аудиторії без можливості реакції, та двосторонні (діалогічні) канали, які передбачають активну взаємодію користувачів із ресурсом. Таке розмежування є критично важливим при розробці функціональної архітектури вебсистеми, оскільки сучасна БО має підтримувати обидва формати комунікації.

До монологічних засобів належать: публікації (звіти, книги, статті, журнали), інформація про саму організацію, щорічні звіти, календарі заходів, онлайн-прес-центри, новини, фото- та відеоматеріали, зовнішні посилання, інтерактивна інфографіка та кнопки підписки на соцмережі [1]. Діалогічні інструменти включають: форми підписки, пошукові механізми, можливість скачування файлів, кнопки поширення контенту, форми для запитів та пожертв, секції для ідей користувачів та простір для співпраці [1].

Згідно з аналітичними даними щодо 30 ключових некомерційних організацій, переважаюча більшість показує найвищий показник монологічної інтерактивності (3 з 5 балів), проте лише середній рівень двосторонньої взаємодії (середнє значення становить 1,4 бала з 5). Лише два суб’єкти з тридцяти – *Direct Relief* та *Action Against Hunger* – змогли досягти максимального індексу інтерактивності в обох розглянутих вимірах [1].

Ці результати чітко вказують на системну ваду сектору: НУО орієнтовані насамперед на показ інформації, але недостатньо активно застосовують інтерактивні функції своїх вебплатформ для встановлення двостороннього діалогу зі споживачами. Розподіл рівнів інтерактивності серед провідних організацій відображено в таблиці 1.1. Шкала: 1 – дуже низька інтерактивність, 5 – дуже висока інтерактивність.

Таблиця 1.1 – Рівень монологічної та діалогічної інтерактивності вебсайтів провідних НУО світу (адаптовано з [1])

НУО	**Монологічна інтерактивність**	**Діалогічна інтерактивність**
Direct Relief	3.0	2.4
Action Against Hunger	3.0	2.4
Wikimedia Foundation	1.8	2.4
Heifer International	2.2	2.4
Transparent Hands	3.0	2.0
BRAC	3.0	2.0
CARE International	3.0	1.4
Save The Children	2.6	1.4
Oxfam International	3.0	1.4
Amnesty International	2.4	1.4
Plan International	1.6	1.4
Acumen Fund	1.8	1.4
Danish Refugee Council	2.4	1.0
Cure Violence Global	2.2	1.0
Anti-Slavery International	1.6	1.0

Наведені у таблиці 1.1 дані підкреслюють необхідність розробки вебплатформи для БО, яка органічно поєднувала б монологічні (CMS-контент, публічний портал, події, звіти) та діалогічні (інтерактивні форми, адміністративні

сервіси) компоненти. Це обґрунтовує включення до розроблюваної системи як публічного порталу з CMS, так і адміністративного модуля з соціальними сервісами.

Проблема прозорості та підзвітності в секторі БО.

Прозорість виступає ключовим елементом, що формує довіру до благодійних структур та безпосередньо впливає на їхню здатність залучати фінансові ресурси та суспільну підтримку. Систематичний огляд наукових джерел, проведений *Ortega-Rodríguez* та ін. [3] у журналі *Sustainability* (MDPI, 2020), аналізує 54 праці за період 2005-2019 років і визначає прозорість як «моральну практику підзвітності, за допомогою якої організації розкривають інформацію про свою діяльність та використання ресурсів задля виконання місії» [3]. Дослідники спираються на теорію споживачів, згідно з якою НКО мають ідентифікувати важливу інформацію для кожної групи зацікавлених осіб та створювати інструменти, що реально задовольняють їхні інформаційні потреби [3].

Автори виокремлюють п’ять специфічних характеристик НКО [3], які фокусуються на прозорості в цьому секторі. По-перше, існує багато зацікавлених сторін, яким необхідно знати, чи здійснюється адекватне управління ресурсами організації, тому вони потребують прозорої інформації. По-друге, НКО виконують найважливіші соціальні функції, і їх ефективність не може бути оцінена виключно фінансовими показниками. По-третє, НКО не мають права розподіляти надлишок доходів між своїми членами, тобто будь-які отримані кошти повинні бути використані виключно на реалізацію місії. По-четверте, ці організації залучають публічні кошти і користуються сприятливим податковим режимом, що посилює суспільні очікування щодо звітності. По-п'яте, НКО регулюються менш суворо, ніж суб'єкти публічного чи приватного сектору, що підвищує ризики нецільового використання ресурсів.

Дослідники також фіксують зростаючий академічний інтерес до теми прозорості НКО: якщо у 2005-2006 рр. виходило лише 3 публікації на рік, то у 2017-2019 рр. – вже 6-7 на рік [3]. Загальна динаміка публікацій у 19 провідних

наукових журналах, яка характеризує розвиток наукового дискурсу щодо прозорості у третьому секторі, наведена у діаграмі на рисунку 1.1.

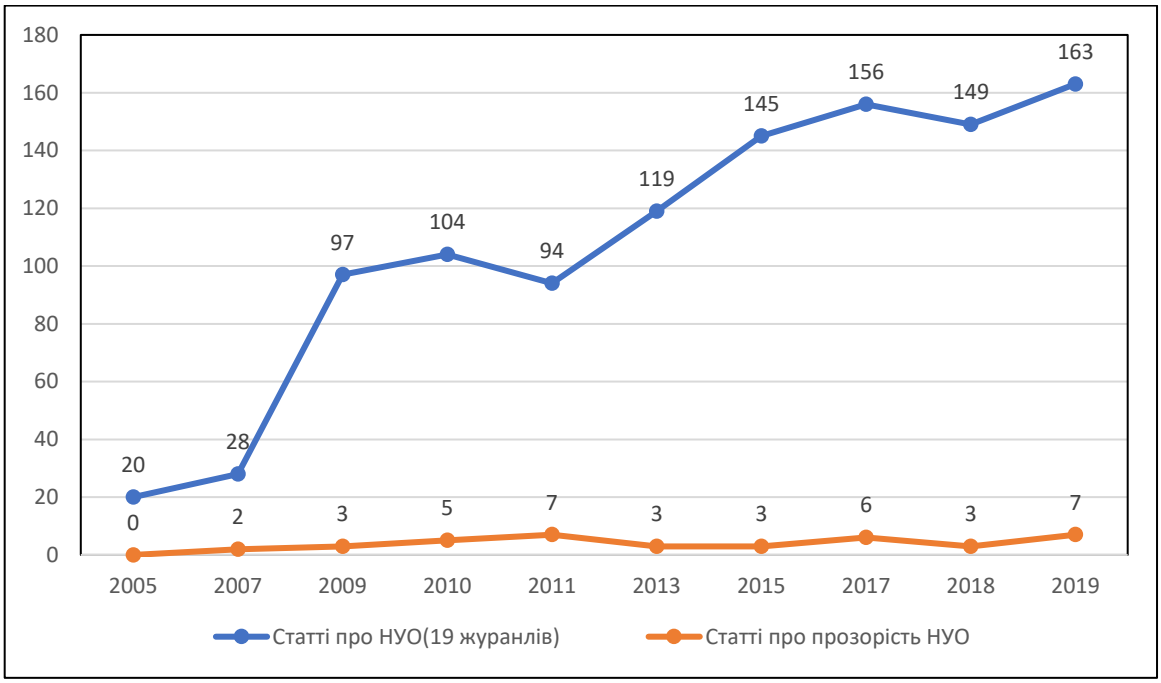

Рисунок 1.1 – Діаграма динаміки публікацій щодо прозорості НУО у наукових журналах у період 2005-2019 рр. (за даними [3])

Аналіз даних діаграми на рисунку 1.1 засвідчує, що проблема прозорості неприбуткових організацій досі залишається недостатньо вивченою: лише 3,47% публікацій, присвячених третьому сектору, безпосередньо торкаються цього аспекту. Натомість позитивна динаміка свідчить про зростання уваги науковців до цієї проблематики, що підтверджує актуальність обраної теми кваліфікаційної роботи.

Стаття Cooley [2], опублікована у журналі *Administrative Sciences* (видавництво MDPI, 2024 рік), присвячена виключно українському некомерційному сектору та є своєрідною завдяки урахуванню специфіки національного контексту. Кількісний контент-аналіз базувався на вивченні 144 вебсайтів державних НУО та оцінював рівень їхньої онлайн-підзвітності за п’ятьма компонентами індексу NPVAI (Non-profit Virtual Accountability Index): доступність (Accessibility), залученість (Engagement), результативність (Performance),

управління (Governance) та місія (Mission) [2]. Деталізація кожного з цих вимірів разом із відповідними середніми балами подана у таблиці 1.2.

Таблиця 1.2 – Виміри та результати онлайн-підзвітності українських НУО за інструментом NPVAI (за даними [2])

Вимір	Що оцінюється	Середній бал (макс. 3)	Рівень
Доступність	Навігація, пошук, мовні опції, карта сайту	1.8	Середній
Залученість	Дата оновлення, підписка, посилання соцмереж, онлайн-чат	1.6	Нижче середнього
Результативність	Річний звіт, фінансова звітність, нагороди	0.9	Низький
Управління	Статут, список керівництва, протоколи зборів	0.7	Критично низький
Місія	Цілі, стратегія, місія, цінності організації	1.4	Нижче середнього

Як свідчать дані таблиці 1.2, найнижчі результати отримано за критеріями «Управління» (0,7) та «Результативність» (0,9), які мають ключове значення для побудови довіри з боку донорів і громадськості. Авторка дослідження акцентує на суттєвих відмінностях між організаціями та робить висновок, що «українські НУО недооцінюють можливості цифрових інструментів для забезпечення прозорості та взаємодії зі споживачами» [2].

Додатково стаття [2] вказує на тривожну соціологічну тенденцію: лише менше 5% громадян України «повністю довіряють» неурядовим організаціям, тоді як 18% висловлюють «повну недовіру» до них. Це підкреслює гостру потребу у впровадженні онлайн-механізмів звітності для відновлення довіри до сектору.

Зростання рівня цифрової звітності безпосередньо пов’язане зі збільшенням пожертв: аналіз демонструє, що публікація фінансових звітів на сайті сприяє активізації донорської підтримки [2].

1.2 Огляд наявних аналогів і наукових публікацій

Аналіз існуючих програмних рішень для благодійних організацій.

На сучасному ринку пропонуються різноманітні програмні продукти, налаштовані для специфічних потреб БО. Умовно їх можна класифікувати на хмарні SaaS-сервіси для залучення коштів, CRM-інструменти для комунікації з донорами та рішення з відкритим кодом. Кожен із цих варіантів має свої переваги та недоліки в функціональності, захищеності та гнучкості налаштувань. Далі наведено аналіз найпопулярніших варіантів.

Givebutter [28] – це хмарна SaaS-система, створена для НУО, які займаються збором коштів та організацією віртуальних заходів. Вона пропонує REST API, базові адміністративні інструменти, підтримку двофакторної автентифікації та інтуїтивно зрозумілий інтерфейс для запуску кампаній. Однак платформа не включає власну CMS для керування контентом, не підтримує SSR-рендеринг, позбавлена WAF-захисту та працює виключно у хмарі провайдера, що виключає можливість локального встановлення.

Donorbox [29] – SaaS-продукт, головним чином спрямований на обробку пожертв та керування донорськими підписками. Його адміністративні можливості обмежені, а відсутність двофакторної автентифікації суттєво дискредитує рівень безпеки. Через брак CMS, WAF-захисту та підтримки SSR Donorbox не може виступати як повноцінна основа для вебпорталу організації.

Bloomerang [30] – CRM-система для некомерційних організацій, зосереджена на відносинах із донорами та аналітичній звітності. Хоча тут передбачено двофакторну автентифікацію та певні механізми звітності, відсутня власна CMS, а локальне розгортання неможливе. Орієнтуючись суто на CRM-функції, ця система не забезпечує повноцінного публічного вебінтерфейсу для організації.

NPSP від Salesforce (Nonprofit Success Pack) [31] – це потужна система управління взаєминами з клієнтами, що пропонує глибоку аналітику та розгалужену екосистему для інтеграцій. Хоча платформа підтримує двофакторну автентифікацію та низку заходів безпеки, її дорога вартість та складність впровадження робить доступність обмеженою для малих і середніх БО. До того ж, рішення не має можливості створення публічного вебпорталу.

Розв'язок на базі WordPress [32] із використанням плагінів (таких як GiveWP, Charitable тощо) є найпоширенішим підходом серед програмного забезпечення з відкритим кодом. Він дозволяє просто створювати вебсайти та публікувати матеріали. Завдяки механізму плагінів частково реалізується функціонал CMS, проте відсутні вбудовані REST API, двофакторна автентифікація, інтеграція з WAF від CloudFlare та підтримка SSR. Безпека WordPress залишається проблемою: значна частка відомих вразливостей вебзастосунків пов'язана саме з цією CMS та її розширеннями [9].

У таблиці 1.3 наведено порівняльну характеристику всіх проаналізованих рішень за основними критеріями. Ці показники сформовано на основі функціональних і нефункціональних вимог, викладених у пункті 1.1, а також із урахуванням результатів наукових досліджень [1-4].

Таблиця 1.3 – Порівняльний аналіз існуючих програмних рішень для благодійних організацій

Критерій	**Givebutter [28]**	**Donorbox [29]**	**Bloomerang [30]**	**Salesforce NPSP [31]**	**WordPress [32]**	**Розроблювана система**
Власна CMS для контенту	–	–	–	–	+	+
REST API	+	+	+	+	–	+
2FA	+	–	+	+	–	+
CloudFlare	–	–	–	–	частково	+
SSR	–	–	–	–	–	+

Кінець таблиці 1.3

Критерій	**Givebutter**	**Donorbox**	**Bloomerang**	**Salesforce NPSP**	**WordPress + плагіни**	**Розроблювана система**
Адміністративна панель	+	+	+	+	+	+
Онлайн-звітність	+	+	+	+	–	+
Відкритий вихідний код	–	–	–	–	+	+
Локальне розгортання	–	–	–	–	+	+
CDN-інтеграція	–	–	–	–	частково	+
Управління заходами	–	–	+	+	–	+
Безкоштовне корст.	частково	частково	–	–	+	+

Аналіз даних, наведених у таблиці 1.3, свідчить про те, що створювана система повністю відповідає всім 12 визначеним критеріям. Жодне з п’яти розглянутих рішень не поєднує в собі одночасно власну CMS, SSR, двофакторну аутентифікацію, хмарний WAF-захист, відкритий код та можливість локального впровадження. Така ситуація слугує переконливою аргументацією доцільності створення власної вебсистеми з використанням обраного технологічного стеку.

Огляд наукових публікацій та їх значення для дослідження.

З метою формування теоретичної основи цієї кваліфікаційної роботи було здійснено огляд сучасних наукових статей з відкритого доступу в базі MDPI. Критеріями відбору джерел виступали відповідність тематики, час видання (переважно період 2020-2025 років) та індексація у базах *Scopus* або *Web of Science*. Результати аналізу публікацій систематизовано в таблиці 1.4.

Таблиця 1.4 – Огляд ключових наукових публікацій з теми роботи

№	Автори/Назва	Журнал (MDPI)	Рік	Внесок у роботу
1	Moreno-Cabanillas et al. Digital Communication and Social Organizations	Future Internet 16(1), 26	2024	Класифікація інструментів вебкомунікації НКО, аналіз рівнів монологічної та діалогічної інтерактивності 30 провідних організацій
2	Cooley A. Toward Greater Legitimacy: Online Accountability Practices of Ukrainian Nonprofits	Administrative Sciences 14(1), 4	2024	Кількісний аналіз онлайн-підзвітності 144 українських НУО за вимірами NPVAI; виявлення критично низьких показників управління та результативності
3	Ortega-Rodríguez et al. Transparency as a Key Element in Accountability in NPOs	Sustainability 12(14), 5834	2020	Систематичний огляд 54 публікацій щодо прозорості НУО; теорія стейкхолдерів як основа підзвітності; три причини обов'язкового розкриття інформації
4	Ahmed et al. Blockchain-Empowered Decentralized Philanthropic Charity	Sustainability 16(1), 210	2024	Виявлення проблем прозорості, автентифікації та шахрайства в існуючих системах БО; зростання кількості шахрайств на 70% у 2020 р.

Матеріали, подані в таблиці 1.4, охоплюють сферу діяльності БО і в комплексі складають теоретичну базу для обґрунтування доцільності створення

запропонованої вебсистеми. У роботі [1] вказано, що провідні БО систематично недооцінюють діалогові інструменти цифрової взаємодії, де середній рівень інтерактивності становить лише 1.4 бала з п’яти можливих. Робота [2] уточнює цю проблему в українських реаліях: аналіз 144 сайтів державних БО виявив критично низькі результати за параметрами «Управління» (0.7 із 3) та «Результативність» (0.9 із 3), що підтверджує системну відсутність публічної фінансової звітності та інформації про керівний склад.

Праця [3], спираючись на теорію зацікавлених сторін, доводить, що прозорість є обов’язковою умовою функціонування БО, які залучають державні кошти та мають пільги з оподаткування. Дослідження [4] переносить фокус у площину безпеки: зростання шахрайств у благодійному секторі на 70% протягом року є вагомим аргументом на користь впровадження надійних механізмів автентифікації та контролю доступу. Отже, огляд літератури визначає три основні напрямки вимог до нової системи: забезпечення публічної підзвітності [2-3], повноцінну двосторонню цифрову комунікацію [1] та захист від несанкціонованого доступу [4].

1.3 Постановка задачі для розробки вебсистеми

Мета та об'єкт дослідження.

Виходячи з проведеного аналізу предметної області, ідентифікованих проблем та огляду існуючих рішень (табл. 1.3), сформульовано мету кваліфікаційної роботи: створити інформаційну вебсистему для благодійної організації з адміністративними й соціальними сервісами, що дозволить автоматизувати основні процеси, забезпечити публічну підзвітність та гарантувати високий рівень захисту даних.

Об’єктом роботи виступають процеси інформаційного забезпечення діяльності благодійних організацій.

Предметом роботи є методи, моделі та засоби розробки вебсистем із використанням Django REST Framework, Next.js, CloudFlare та системи управління контентом із підтримкою двофакторної автентифікації.

Вимоги до розроблюваної системи.

Після аналізу предметної сфери та огляду існуючих аналогів визначено функціональні вимоги до розроблюваної системи. До критичних вимог належать: реєстрація та автентифікація користувачів з підтримкою двофакторної автентифікації на основі TOTP, повноцінна адміністративна панель з CRUD-операціями, інтеграція CloudFlare WAF для захисту від DDoS-атак, а також забезпечення HTTPS, rate limiting та захисту HTTP-заголовків відповідно до стандартів OWASP Top 10. До вимог високого пріоритету відносяться: публічний вебпортал з інформацією про місію, команду та проєкти організації, розділ публічної звітності з фінансовими документами та статутом, система управління контентом (CMS) без необхідності програмування, а також задокументований REST API для взаємодії між фронтендом і бекендом. Вимогами середнього пріоритету визначено управління медіафайлами через CMS, серверний рендеринг сторінок засобами Next.js для підвищення показників SEO, а також доставку статичних ресурсів через CloudFlare CDN для забезпечення продуктивності та доступності системи.

Задачі для досягнення мети.

Для досягнення поставленої мети визначено такий перелік конкретних задач, що підлягають вирішенню в межах даної кваліфікаційної роботи:

1) провести аналіз предметної сфери діяльності благодійних організацій та виявити основні проблеми їх інформаційного забезпечення;

2) виконати огляд і порівняльний аналіз наявних аналогів вебплатформ для БО та відповідних наукових публікацій у відкритому доступі;

3) сформулювати функціональні та нефункціональні вимоги до розроблюваної системи на основі результатів аналізу;

4) обрати та обґрунтувати технологічний стек: Django REST Framework, Next.js, CloudFlare, TOTP-автентифікація, CMS;

5) спроектувати архітектуру системи: структуру бази даних, компонентні, функціональні та UML-діаграми;

6) реалізувати бекенд-частину системи на основі Django REST Framework з підтримкою двофакторної автентифікації;

7) розробити фронтенд-застосунок на основі Next.js із SSR/SSG та інтеграцією системи управління контентом;

8) налаштувати хмарний захист WAF та доставку контенту засобами CloudFlare;

9) провести функціональне тестування розробленої системи та оцінити відповідність вимогам.

Висновки до розділу 1

У першому розділі здійснено всебічне дослідження предметної області благодійних установ та доведено актуальність створення підходящої вебсистеми. Виявлено, що третій сектор економіки стабільно розвивається та суттєво впливає на задоволення суспільних потреб. Огляд літературних джерел [1-4] засвідчив, що значна частина громадських організацій не реалізує повністю можливості цифрових інструментів: середній показник діалогічної взаємодії сайтів провідних світових НГО не перевищує 1.4 бала з 5 (табл. 1.1), тоді як рівень онлайн-звітності державних українських БО є критично низьким, особливо в аспектах «Управління» (0.7) та «Результативність» (0.9) (табл. 1.2). Ці факти підтверджують гостру потребу у створенні комплексного вебдодатку.

Порівняння п’яти популярних рішень (Givebutter, Donorbox, Bloomerang, Salesforce NPSP, WordPress) виявило, що жодне з них не відповідає всім суворим функціональним та безпековим критеріям (табл. 1.3). Аналіз статей із видань MDPI (табл. 1.4) підтвердив доцільність обраного технологічного стеку та обґрунтував ключові архітектурні рішення.

Сформульовано мету, об’єкт і предмет дослідження. Складено функціональні вимоги до системи, а також повний перелік нефункціональних вимог. Визначено дев’ять конкретних завдань, реалізація яких гарантує досягнення поставленої мети кваліфікаційної роботи.

2 МОДЕЛІ, МЕТОДИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ДЛЯ ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1 Методи та архітектурні моделі для побудови вебсистем

Створення сучасних вебзастосунків передбачає ретельний вибір архітектурного підходу, який регулює структуру компонентів, обмін даними та принципи їхньої взаємодії. У рамках розробки інформаційної вебплатформи для БО проаналізовано три ключові стратегії: REST-архітектуру (рис. 2.1), трирівневу клієнт-серверну модель та доменно-орієнтоване проектування (DDD).

REST-архітектура.

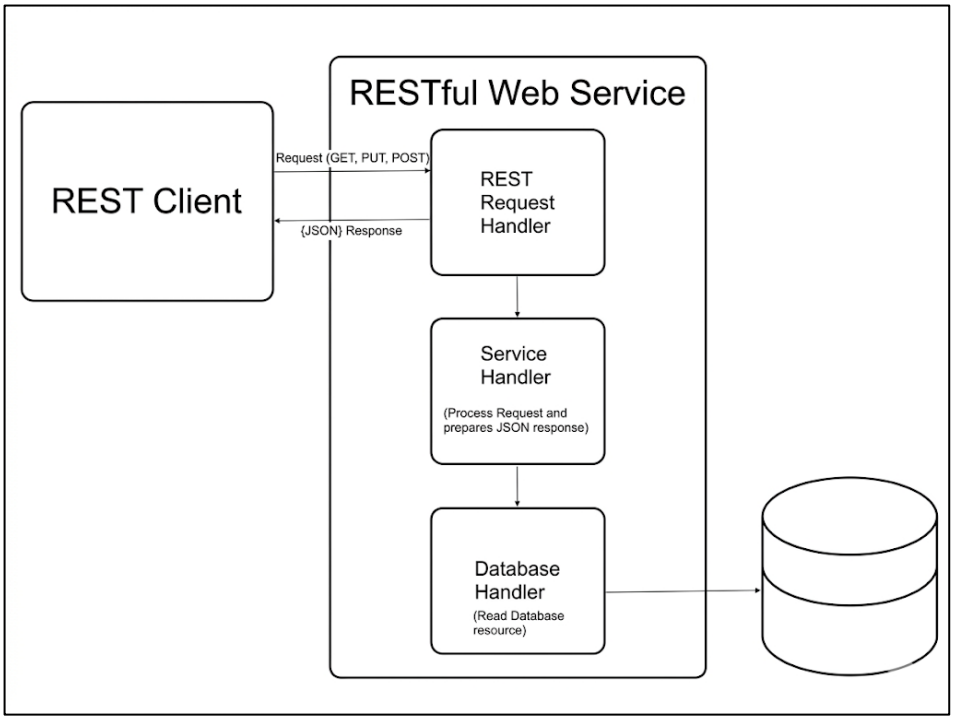

Рисунок 2.1 – REST-архітектура

Концепція архітектури REST (Representational State Transfer), запропонована Філдінгом у 2000 році [5], базується на шести ключових обмеженнях для розподілених систем: уніфікований інтерфейс, розділення клієнта та сервера, можливість кешування, багаторівнева структура та опція виконання коду на стороні сервера. Дотримання цих принципів сприяє незалежному масштабуванню клієнтської й серверної частин, стандартизує комунікацію через HTTP-методи

(GET, POST, PUT, PATCH, DELETE) та полегшує інтеграцію ззовні [6].

На відміну від SOAP, REST не вимагає суворого протокольного контракту; взаємодія керується URI ресурсів та семантикою HTTP, а не XML-схемами. Порівняно з GraphQL, REST забезпечує простіше кешування завдяки GET-запитам і сумісність із CDN, що критично важливо для роботи з CloudFlare. Використання JSON як формату даних є компактним і має широку підтримку в браузерах та JavaScript-фреймворках.

Архітектура з трьома рівнями.

Трирівнева архітектура розділяє систему на три логічно-автономні шари: інтерфейс, бізнес-логіка та зберігання даних [7]. Такий підхід мінімізує взаємозв’язок між компонентами: зміни в бізнес-правилі не зачіпають інтерфейс, а перехід на іншу СУБД не впливає на роботу логіки додатка. Порівняно з монолітною структурою, трирівнева модель полегшує горизонтальне масштабування окремих шарів і спрощує їхнє незалежне тестування. На відміну від мікросервісної архітектури, вона не передбачає мережевих запитів між внутрішніми частинами, що знижує операційну складність для команд із обмеженими ресурсами.

Доменно-орієнтоване проектування (DDD) та UML.

Доменно-орієнтоване проектування (Domain-Driven Design, DDD), запропоноване Е. Евансом [8], передбачає виділення ключових сутностей предметної області (агрегатів, сутностей і об'єктів-значень) та побудову моделі навколо бізнес-логіки, а не технічних деталей реалізації. Для документування архітектурних рішень застосовується нотація UML 2.5.1 [9]: діаграми варіантів використання, компонентів, класів та послідовності. Безпека системи проектується відповідно до методології OWASP Top 10 [12], що охоплює захист від найпоширеніших класів вразливостей вебдодатків.

2.2 Технології серверної частини

Python-фреймворки для побудови REST API.

Вибір серверного фреймворку є одним із ключових архітектурних рішень, що визначає доступний інструментарій безпеки, продуктивність і обсяг ручної розробки. У таблиці 2.1 наведено порівняльний аналіз чотирьох поширених фреймворків для побудови REST API.

Таблиця 2.1 – Порівняльний аналіз бекенд-фреймворків

Критерій	**Django REST Framework**	**FastAPI**	**Node.js + Express**	**Laravel (PHP)**
Мова	Python	Python	JavaScript	PHP
Парадигма	MVC + ORM	Async/ASGI	Event-loop	MVC + ORM
Продуктивність (RPS орієнт.)	3 000-6 000	6 000-12 000	8 000-20 000	2 000-5 000
Вбудований ORM	Так (Django ORM)	Ні (SQLAlchemy опц.)	Ні	Так (Eloquent)
Вбудована адмін-панель	Так	Ні	Ні	Nova (платна)
RBAC «з коробки»	Так (Groups/Perms)	Ні	Ні	Частково
Серіалізація даних	DRF Serializers	Pydantic (авто)	Вручну / joi	Fractal / Resource
Автодокументація API	drf-spectacular	Вбудована (OpenAPI)	Swagger вручну	l5-swagger
Екосистема OAuth / 2FA	django-allauth, django-otp	Сторонні	Passport.js	Socialite
Зрілість (рік першого релізу)	2005 (DRF – 2011)	2018	2009	2011
Ліцензія	BSD	MIT	MIT	MIT
Обрано для проєкту	+	–	–	–

Django представляє собою потужний фреймворк на Python, який дотримується парадигми «все враховано»: він постачається з вбудованими інструментами ORM, автентифікації, адмін-панеллю, системою міграцій та

захистом від CSRF із коробки [10]. Django REST Framework (DRF) виступає як розширення, що надає серіалізатори, класи роутингу для автоматизації URL, механізми обмеження запитів та підтримку автодокументації через drf-spectacular у стандарті OpenAPI 3.0 [11]. Разом вони створюють зрілу екосистему, історія якої починається з першого публічного релізу Django у 2005 році.

FastAPI, з'явившись у 2018 році, демонструє високу швидкість завдяки асинхронному стеку ASGI та автоматичній валідації даних через Pydantic. Проте він не має вбудованого адмін-інтерфейсу, системи RBAC або готових рішень для двофакторної автентифікації, тому ці функції потребують окремого підключення. Node.js/Express показує найвищі показники обробки запитів завдяки подійно-орієнтованій архітектурі, але позбавлений рідної ORM і вимагає значно більше ручної роботи для налаштування безпеки. Laravel пропонує ORM та адмін-панель, але належить до PHP-екосистеми, що є незалежною від критичних бібліотек безпеки Python (django-otp, django-axes, django-allauth), необхідних для даного проєкту.

Вибір Django REST Framework обґрунтований кількома ключовими факторами: по-перше, вбудована адмін-панель автоматично створює CRUD-інтерфейси для всіх моделей, суттєво зменшуючи обсяг розробки; по-друге, вбудована система груп і прав дозволяє реалізувати рольовий доступ (RBAC) без залучення сторонніх пакетів; по-третє, перевірені бібліотеки django-allauth, django-otp та django-axes інтегровані в Django, що дозволяє уникнути необхідності створення додаткових мікросервісів REST.

Моделі та механізми автентифікації.

Автентифікація у вебдодатках може бути реалізована кількома підходами, кожен з яких має різні характеристики безпеки та сумісності. Таблиця 2.2 порівнює три основні підходи для контексту SSR-системи з Django-бекендом.

Таблиця 2.2 – Порівняльний аналіз механізмів автентифікації

Критерій	**Session + Cookie**	**JWT (Bearer)**	**API Key**
Зберігання стану	Server-side (Redis/DB)	Клієнт (localStorage / cookie)	Server-side DB
Вразливість до XSS	Мінімальна (HttpOnly)	Висока (localStorage)	Середня
Відкликання токена	Миттєве (delete session)	Складне (blacklist)	Пряме (delete key)
Session Timeout	Middleware (вбудований)	exp-поле + blacklist	Не передбачено
Інтеграція 2FA	Природна (session flag)	Додатковий крок	Відсутня
Stateless	Ні	Так	Ні
Підходить для SSR (cookie)	Так	Частково (header)	Ні
Обрано для проєкту	+	–	–

Автентифікація через сеанси (Session + Cookie) передбачає збереження ідентифікатора сесії в підписаному файлі cookie HttpOnly, який браузер автоматично надсилає з кожним запитуванням. Стан користувача зберігається на стороні сервера. Головна перевага такого підходу полягає в тому, що файл cookie HttpOnly недоступний для JavaScript-коду на сторінці, що захищає від викрадення ідентифікатора під час XSS-атак [13]. Відмова від сеансу відбувається миттєво: достатньо лише видалити відповідний запис зі сховища.

Для програм із серверним рендерингом (SSR) на платформе Next.js підхід на основі сеансів є найбільш природним: серверні компоненти Next.js виконуються на сервері й мають прямий доступ до файлів cookie запиту, тоді як використання JWT у клієнтських компонентах вимагає явного додавання заголовка авторизації до кожного запиту. Механізм сеансів Django обрано як основний через його вищу стійкість до XSS, простоту інтеграції з 2FA та нативну підтримку в Django [10].

Соціальна автентифікація: OAuth 2.0.

OAuth 2.0 [15] є стандартом авторизації, що дозволяє стороннім застосункам отримувати обмежений доступ до ресурсів користувача без передачі пароля. У

контексті соціального входу (Social Login) використовується Authorization Code Flow: користувач перенаправляється до провайдера (Google, GitHub тощо), підтверджує доступ і повертається до застосунку з кодом авторизації, який сервер обмінює на access token та ідентифікаційні дані користувача. Таблиця 2.3 порівнює провайдерів соціальної автентифікації.

Таблиця 2.3 – Порівняльний аналіз провайдерів соціальної автентифікації

Критерій	**Google OAuth 2.0**	**GitHub OAuth**	**Microsoft OIDC**	**Facebook Login**
Стандарт	OAuth 2.0 / OIDC	OAuth 2.0	OAuth 2.0 / OIDC	OAuth 2.0
Верифікація email	Гарантована	Опціональна	Гарантована	Опціональна
Корпоративні домени (allowlist)	Так (hd-параметр)	Ні	Так (tenant)	Ні
Пакет django-allauth	Так (зрілий)	Так	Так	Так
Поширеність в NGO-секторі	Висока	Низька	Середня	Середня
Обрано для проєкту	+	–	–	–

Для надання автентифікації обрано протокол Google OAuth 2.0 із підтримкою розширення OIDC (OpenID Connect), що використовує механізм OAuth для передачі даних про особу [17]. Цей вибір зумовлений кількома факторами. По-перше, Google гарантує підтвердження дійсності електронної адреси користувача, тоді як у GitHub адреса може бути прихована або не верифікована. По-друге, Google OAuth передбачає параметр hd (розміщений домен), який обмежує доступ лише до акаунтів конкретного домену Google Workspace, що фактично реалізує білий список на стороні постачальника [17]. По-третє, бібліотека django-allauth [18] пропонує зрілу та добре документовану інтеграцію Google OAuth із Django, ставши де-факто стандартом для соціальної автентифікації в цій екосистемі.

Двофакторна автентифікація: TOTP.

Двофакторна автентифікація (2FA) додає другий незалежний чинник підтвердження особи, що суттєво знижує ризик несанкціонованого доступу при компрометації пароля. TOTP (Time-based One-Time Password) [14] – алгоритм генерації одноразових паролів на основі двох вхідних даних: спільного секрету (shared secret), погодженого між сервером і застосунком-автентифікатором під час налаштування, та поточного часового штампу, округленого до 30-секундного інтервалу. Формула генерації:

$$TOTP(K,T) = HOTP(K, \lfloor \frac{(T-T_0)}{X} \rfloor), \quad (2.1)$$

де K – спільний секрет, T – поточний час у секундах Unix epoch, T_0 – початкова точка відліку (0), X – часовий крок (30 с), HOTP – HMAC-based OTP (RFC 4226) [14].

Згенерований код дійсний лише 30-60 секунд і не може бути повторно використаний. Таблиця 2.4 порівнює рішення для реалізації 2FA у Django.

Таблиця 2.4 – Порівняльний аналіз рішень двофакторної автентифікації

Критерій	**django-otp (TOTP)**	**Allauth MFA**	**Authy API**	**Keycloak**
Алгоритм	TOTP RFC 6238	TOTP / WebAuthn	TOTP / SMS	TOTP / FIDO2
Інтеграція з Django	Нативна	Нативна	Через REST API	OAuth2/OIDC
Вартість	Безкоштовно	Безкоштовно	SMS – платний	Безкоштовно (self-hosted)
QR-код генерація	Вбудована	Вбудована	Зовнішня	Зовнішня
Резервні коди	Так	Так	Так	Так
Складність інтеграції	Низька	Низька	Середня	Висока
Обрано для проєкту	+	–	–	–

Бібліотека django-otp [19] є найбільш зрілим рішенням 2FA для Django: вона надає абстрактну модель пристрою (Device) та конкретну реалізацію TOTPDevice, що зберігає shared secret у базі даних Django. Бібліотека генерує URI, який

кодується у QR-код для сканування застосунком-автентифікатором (Google Authenticator, Microsoft Authenticator, Authy, Aegis). На відміну від Authy API, django-otp є повністю self-hosted рішенням без залежності від сторонніх сервісів і без додаткових витрат. Порівняно з Keycloak, інтеграція django-otp потребує на порядок менше інфраструктурних зусиль.

Запобігання брутфорсу та контроль сесій.

Брутфорс (Brute Force) – це спосіб підбору пароля або ключа шляхом послідовного перебору варіантів [12]. Згідно зі стандартом OWASP, відсутність захисту від таких атак належить до категорії A07:2021 «Провали в ідентифікації та автентифікації» [12]. Серед основних методів протидії виділяють: лімітування невдалих спроб входу (блокування акаунту), тимчасове блокування IP-адрес, використання CAPTCHA та впровадження прогресивних затримок між спробами.

Бібліотека django-axes [20] пропонує вбудоване в Django рішення для блокування за IP або ім’ям користувача. Вона підхоплює сигнал *user_login_failed,* фіксуючи спроби в окремій таблиці *AccessAttempt*. Коли кількість запитів перевищує встановлений поріг за певний період, подальші спроби авторизації блокуються з кодом HTTP 403 до завершення часу блокування або його ручного скасування адміністратором. Логи блокувань доступні через Django Admin, що спрощує моніторинг підозрілої активності.

Управління сесіями організовується згідно з керівництвом OWASP Session Management Cheat Sheet [13]. До ключових заходів належать: автоматичне завершення сесії після періоду неактивності (session timeout), зміна ідентифікатора сесії після успішної авторизації (для уникнення session fixation) та додавання атрибутів Secure і HttpOnly до сесійних файлів cookie. Django має вбудовані параметри, а для реалізації логіки sliding-expiration можна використати власний middleware.

Захисні HTTP-заголовки.

HTTP-заголовки безпеки є механізмом, що дозволяє серверу інформувати браузер щодо дозволеної поведінки при обробці відповіді. OWASP рекомендує низку заголовків для протидії поширеним атакам [12]:

– Content-Security-Policy (CSP) – визначає дозволені джерела завантаження скриптів, стилів, зображень та інших ресурсів, ефективно запобігаючи XSS-атакам шляхом блокування виконання несанкціонованого JavaScript;

– Strict-Transport-Security (HSTS) – примусово переключає браузер на HTTPS на визначений термін (max-age), унеможливлюючи downgrade-атаки до HTTP;

– X-Frame-Options: DENY – забороняє вбудовування сторінки в iframe, захищаючи від clickjacking-атак;

– X-Content-Type-Options: nosniff – забороняє браузеру «вгадувати» MIME-тип відповіді, запобігаючи MIME-sniffing-атакам;

– Referrer-Policy – контролює обсяг інформації у заголовку Referer при переходах між сайтами.

Django надає вбудовані налаштування, проте повний набір заголовків (зокрема CSP) реалізується через кастомний middleware [10].

Обмеження частоти запитів (Rate Limiting).

Rate limiting – механізм обмеження кількості запитів до API за одиницю часу, що захищає від DoS-атак, брутфорсу та зловживання ресурсами сервера. DRF реалізує throttling через класи AnonRateThrottle (для неавтентифікованих клієнтів) та UserRateThrottle (для авторизованих) [11]. Ліміти задаються у форматі «кількість/час» (наприклад, 100/day, 10/min) і зберігаються у кеші. Для ендпоінтів автентифікації рекомендується встановлювати суворіші ліміти, ніж для звичайних ресурсів API, відповідно до рекомендацій OWASP [12].

2.3 Технології клієнтської частини

Стратегії рендерингу у сучасних фреймворках.

Сучасні фронтенд-інструменти передбачають різні підходи до рендерингу, які безпосередньо впливають на швидкість роботи, оптимізацію для пошукових систем та якість користувацького досвіду. При Client-Side Rendering (CSR) HTML-код формується у браузері після завантаження JavaScript-пакетів, що залишає порожнім тег на початковому етапі й ускладнює SEO-індексацію. Server-Side Rendering (SSR) створює повний HTML на сервері для кожного запиту: це дозволяє пошуковим роботам отримати готовий контент, а час до першого відображення (FCP) стає мінімальним. Static Site Generation (SSG) генерує HTML-файли під час етапу збирання (build time): сторінки розміщуються на CDN і обслуговуються без залучення сервера додатку. Incremental Static Regeneration (ISR) – це гібридна технологія Next.js, де сторінки створюються статично, але автоматично оновлюються у фоновому режимі після завершення встановленого інтервалу (revalidate) без необхідності повного перезбирання. Порівняння ключових характеристик фронтенд-фреймворків наведено у таблиці 2.5.

Таблиця 2.5 – Порівняльний аналіз фронтенд-фреймворків

Критерій	**Next.js 14**	**Nuxt.js 3**	**SvelteKit**	**Angular 17**
Мова	TypeScript / JS	TypeScript / JS	TypeScript / JS	TypeScript
SSR	Так	Так	Так	Так (Angular Universal)
SSG	Так	Так	Так	Частково
ISR (інкр. регенерація)	Так	Частково	Ні	Ні
Edge / CDN рендеринг	CloudFlare Pages	Vercel / Netlify	CloudFlare Pages	Ні
Metadata API (SEO)	Вбудований (App Router)	useHead / useSeoMeta	Вручну	Angular Meta

Кінець таблиці 2.5

Критерій	**Next.js 14**	**Nuxt.js 3**	**SvelteKit**	**Angular 17**
Розмір initial bundle	~85 KB	~72 KB	~12 KB	~130 KB
Зрілість / спільнота	Дуже висока	Висока	Зростаюча	Висока
Обрано для проєкту	✓	—	—	—

Next.js 14 із застосуванням App Router [21] очолює рейтинг за рівнем гнучкості рендерингу: система підтримує SSR, SSG та ISR в межах одного проекту, дозволяючи кожній сторінці обирати власну стратегію відповідно до специфіки даних. App Router, представлений у Next.js 13 і стабілізований у версії 14, спирається на React Server Components (RSC) – компоненти, що виконуються суто на сервері й не передають JavaScript у браузер. Це зменшує обсяг клієнтського бандлу та прискорює показник Time to Interactive (TTI).

Nuxt.js 3 пропонує аналогічні можливості для екосистеми Vue, але має меншу кількість готових адаптерів для хмарних сервісів. SvelteKit створює мінімальний JavaScript-бандл завдяки компільованій природі Svelte, проте підтримка ISR залишається неповною. Angular Universal реалізує SSR, але не має вбудованої підтримки SSG та ISR, що ускладнює оптимізацію SEO для статичного контенту. Next.js також обрано через нативну сумісність із CloudFlare Pages за допомогою адаптера @cloudflare/next-on-pages та вбудований Metadata API для керування SEO-тегами на рівні Server Components [21].

Налаштування SEO та використання структурованих даних.

SEO (Search Engine Optimization) представляє собою сукупність технічних і контентних дій, спрямованих на покращення позицій вебресурсу у видачі пошукових систем (рис.2.2). До технічних складових належать: правильно налаштовані мета-теги (title, description), розмітка Open Graph для соцмереж, схемні дані schema.org, показники швидкості завантаження, а також наявність файлів sitemap.xml та robots.txt .

Рисунок 2.2 – Основні вимоги до оптимізації SEO [33]

Schema.org – це словар термінів для структурованих даних, який допомагає пошуковим роботам коректно інтерпретувати зміст сторінки. Google рекомендує використовувати формат JSON-LD (JavaScript Object Notation for Linked Data) для інтеграції розмітки schema.org: код розміщується всередині тега.

2.4 Система управління контентом (CMS)

Види CMS та методології їхнього створення.

Система управління контентом (CMS) – це програмний засіб, який дає можливість користувачам без технічних навичок створювати, змінювати та публікувати матеріали, не використовуючи програмування. Існує два головних типи таких систем. Звичайна (монолітна) CMS інтегрує серверну та клієнтську частини в єдиний додаток (наприклад, WordPress, Joomla, Drupal): дані зберігаються і відображаються за допомогою однієї платформи. Headless CMS розділяє рівні зберігання/управління контентом та його відображення: сервер надає дані через REST або GraphQL API, а клієнтська частина (Next.js, мобільний додаток тощо) самостійно обирає метод рендерингу.

Третім варіантом є індивідуальна CMS – власна система, розроблена всередині основного бекенд-фреймворку проєкту. Це забезпечує повний контроль над структурою даних, правами доступу та бізнес-логікою, усуваючи залежність від сторонніх рішень. У таблиці 2.6 проводиться порівняння кастомної CMS на базі

DRF із трьома популярними готовими продуктами за критеріями, важливими для вирішення поставленої задачі.

Таблиця 2.6 – Порівняльний аналіз підходів до реалізації CMS

Критерій	**Кастомна CMS (DRF)**	**Wagtail**	**Strapi**	**Contentful**
Тип	Headless (власна)	Headless / Traditional	Headless	Headless (SaaS)
Інтеграція з Django	Нативна (той самий проєкт)	Нативна (окремий пакет)	Окремий Node.js-сервіс	REST-тільки (SaaS)
Конструктор секцій	JSON-поле + DRF Serializers	StreamField	Dynamic Zones	Structured content
REST API	Так (DRF ViewSets)	Так	Так	Так
Redirects / Sitemap	Django Redirects / django-sitemap	Вбудовані	Плагін	Ні
Rich Text Editor	CKEditor 5 (django-ckeditor)	Draftail / CKEditor	CKEditor	Veltis
Повний контроль схеми	Так (власні моделі Django)	Частково	Так	Ні (SaaS-обмеження)
Безкоштовність	Так	Так (BSD)	Так (self-hosted)	Ні (платний SaaS)
Обрано для проєкту	+	–	–	–

Для реалізації даного проєкту готові CMS-рішення мають суттєві недоліки. Wagtail, який базується на Python та Django, використовує власну абстракцію сторінок (Page tree), що накладає жорсткі вимоги до структури моделей і ускладнює інтеграцію з існуючими моделями (Reports, Events, ContactInfo). Strapi, як окремий Node.js-додаток, вимагає виділеного процесу, бази даних та системи авторизації, що призводить до дублювання функціоналу Django-бекенду та ускладнює єдину RBAC-архітектуру. Contentful – це SaaS-платформа з платною моделлю та відсутністю опції self-hosted, що суперечить вимогам щодо повного контролю над даними.

Кастомна CMS на базі Django REST Framework обрана через такі переваги: по-перше, усі контентні моделі (сторінки-конструктори, звіти, події, контакти, навігація) є стандартними Django-моделями в єдиній PostgreSQL-базі, що спрощує запити та усуває N+1-проблему; по-друге, система RBAC Django (ролі Root/Editor) автоматично поширюється на всі моделі CMS без додаткових налаштувань; по-третє, відмова від сторонніх платформ усуває залежність від їхніх релізних циклів і дозволяє точно налаштовувати кожне поле моделі під специфіку організації.

Конструктор сторінок на базі JSON-секцій.

Створення сторінок (Page Builder) – це методика керування контентом, при якій веб-сторінка формується з окремих типізованих модулів (блоків), що дозволяє їх довільне поєднання та зміну порядку. На противагу жорстко заданій структурі, де всі елементи зафіксовані заздалегідь, такий конструктор дає змогу нетехнічним редакторам створювати різноманітні сторінки, використовуючи обмежений набір перевірених компонентів.

З технічної точки зору реалізація передбачає збереження секцій у форматі масиву JSON-об’єктів у полі типу JSONB (PostgreSQL) або через зв’язані моделі. Кожен об’єкт містить поле type (ідентифікатор типу блоку) та fields (дані блоку). Серіалізатор DRF перевіряє вхідний масив секцій за допомогою вкладених серіалізаторів: для кожного значення type вибирається відповідний дочірній серіалізатор і проводиться незалежна валідація полів. Django Admin забезпечує інтерфейс редагування секцій через inline-форми або JSON-редактор. Ця концепція аналогічна механізму StreamField у Wagtail, але реалізована в рамках стандартних інструментів Django та DRF без залучення сторонніх CMS-платформ.

Rich Text редактори.

Rich Text Editor (WYSIWYG-редактор) надає інтерфейс редагування тексту, подібний до текстового процесора, з підтримкою форматування, таблиць, посилань та вставки медіа. Ключовою вимогою безпеки до rich text редактора є HTML-санітизація: редактор повинен очищати небезпечний HTML (теги <script>, on*-

атрибути) перед збереженням у базу даних, щоб унеможливити зберігання XSS-пейлоадів через адмін-панель.

CKEditor 5 – редактор з відкритим вихідним кодом (GPL), що надає розширюваний plugin-API, вбудовану HTML-санітизацію через allowlist дозволених тегів і атрибутів, підтримку таблиць, зображень та медіа (рис. 2.3). Пакет django-ckeditor забезпечує нативну інтеграцію CKEditor 5 з Django-моделями через поле RichTextField та коректне відображення редактора у Django Admin.

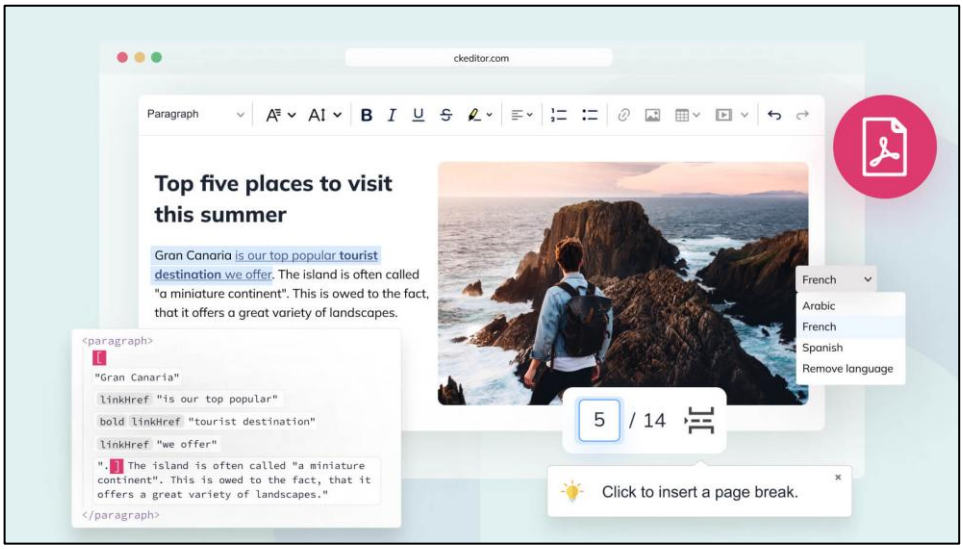

Рисунок 2.3 – Зовнішній вигляд CKEditor 5 [34]

Альтернативою є TinyMCE (через django-tinymce) – зрілий редактор з аналогічним функціоналом, проте CKEditor має більш активний розвиток і кращу підтримку кастомних плагінів. Quill є більш легковісним варіантом, але має обмеженішу підтримку складних таблиць, що є важливим для форматування фінансових звітів організації.

2.5 Хмарна інфраструктура та технології безпеки мережевого рівня

CloudFlare як інструмент забезпечення мережевої безпеки.

CloudFlare – це світова платформа, що пропонує сервіси DNS, CDN, WAF та захисту від DDoS-атак. Інфраструктура CloudFlare охоплює більше 310 міст у понад 100 країнах і обробляє понад 20% всього інтернет-трафіку [22]. Використання CloudFlare Proxy для домену означає, що всі вхідні HTTP/HTTPS

запити спочатку маршрутизуються через Anycast-мережу CloudFlare, де вони фільтруються та кешуються перед надходженням на сервер додатка. Такий підхід приховує справжню IP-адресу сервера від можливих зловмисників.

Web Application Firewall (WAF) [22] – це система фільтрації HTTP-трафіку на рівні застосунку (L7 моделі OSI), яка аналізує зміст запитів і блокує ті, що відповідають сигнатурам атак. WAF від CloudFlare включає OWASP Core Rule Set, що покриває OWASP, а також власні Managed Rulesets для специфічних CVE-вразливостей. Rate Limiting на рівні CloudFlare дозволяє встановлювати обмеження кількості запитів за IP-адресою, країною, URI або заголовками, обробляючи порушення до того, як трафік досягне сервера. CDN (Content Delivery Network) [23] кешує статичні ресурси (зображення, CSS, JS-бандли) на крайових вузлах мережі, зменшуючи затримки та знижуючи навантаження на сервер.

Порівняно з AWS CloudFront або Akamai, CloudFlare пропонує безкоштовний план із достатнім функціоналом для некомерційного сайту (включаючи WAF та DDoS Protection), що є суттєвим з точки зору ресурсних обмежень благодійних організацій. CloudFlare Pages – платформа для Edge-розгортання Next.js застосунків із підтримкою Server Components через CloudFlare Workers Runtime, що дозволяє виконувати SSR-рендеринг на крайових вузлах мережі поблизу користувача [22].

2.6 Технології зберігання даних та інфраструктури

PostgreSQL як реляційна система керування базами даних.

PostgreSQL [24] – це відкрита об'єктно-реляційна СКБД, розробка якої триває з 1986 року, а перша публікація та використання відбулися у 1996 році під ліцензією PostgreSQL (аналог MIT). Ця система повністю відповідає стандарту SQL:2023, забезпечує ACID-транзакції, підтримує як рядкове, так і стовпцеве блокування, має вбудований повнотекстовий пошук, розширений тип JSONB для напівструктурованих даних та гнучку систему типів (дозволяє створювати власні типи, оператори та функції).

У порівнянні з MySQL, PostgreSQL більш суворо дотримується SQL-стандартів і краще працює зі складними запитами. SQLite – це вбудована СКБД без окремого сервера, що робить її придатною для локального розроблення, але непридатною для багатокористувацького виробничого середовища. MongoDB є документоорієнтованою NoSQL СКБД і не підтримується Django ORM нативно. Натомість PostgreSQL має рідну підтримку в Django ORM через драйвер *psycopg2*, який параметризує всі запити, тим самим запобігаючи SQL-ін'єкціям [10, 24].

Docker та контейнеризація.

Docker [26] – це платформа для контейнеризації, яка забезпечує ізольоване середовище для застосунку та його залежностей у стандартизованому контейнері, створеному на базі образу. Такий контейнер може бути запущений на будь-якій операційній системі з Docker-демоном, не залежно від встановлених системних бібліотек, що гарантує однаковість умов розробки та виробництва. Docker Compose надає можливість описати багатоконтейнерну архітектуру за допомогою YAML-файлу (docker-compose.yml) і запускати всі необхідні сервіси (наприклад, Django, Next.js, PostgreSQL) однією командою.

У порівнянні з класичним підходом, коли залежності встановлюються напряму на сервер, контейнеризація значно спрощує процес CI/CD, оскільки пайплайн обмежується лише етапами docker build та docker push. На відміну від складних систем оркестрації, таких як Kubernetes чи Docker Swarm, Docker Compose є менш громіздким рішенням, яке цілком достатньо для розгортання систем середнього рівня, наприклад, для БО з помірним трафіком.

2.7 Зведена характеристика обраного технологічного стеку

Згідно з результатами порівняльного дослідження технологій (розділи 2.1-2.6), визначено програмний стек системи, перелік якої подано у таблиці 2.8.

Таблиця 2.8 – Зведена характеристика обраного технологічного стеку

Рівень / компонент	**Обрана технологія**	**Обґрунтування вибору**
Бекенд / API	Django REST Framework 3.15	Вбудована адмін-панель, RBAC, ORM, екосистема пакетів безпеки
Фронтенд	Next.js 14 (App Router)	SSR/SSG/ISR, Metadata API, CloudFlare Pages Edge Runtime
CMS / Конструктор	Кастомна CMS на DRF + CKEditor 5	Повний контроль схеми, нативна інтеграція з Django-проєктом
Соціальна авт.	django-allauth (Google OAuth 2.0)	allowlist домену, OIDC
Двофакторна авт.	django-otp (TOTP, RFC 6238)	RFC-стандарт, QR-код, безкоштовно, нативна інтеграція
Захист від брутфорсу	django-axes	IP/username блокування, інтеграція з Django Admin
Хмарна безпека	CloudFlare WAF + CDN	DDoS L3–L7, OWASP Managed Ruleset, Edge Functions, CDN
БД	PostgreSQL 16	ACID, JSONB, відкрита ліцензія, Django ORM підтримка
Rich Text	CKEditor 5 (django-ckeditor)	Django-інтеграція, HTML-санітизація, форматування звітів
Контейнеризація	Docker + Docker Compose	Ізоляція залежностей, відтворюване CI/CD-середовище
Логування	Python logging + Sentry	Структуровані логи, real-time alert на продакшн
Резервне копіювання	django-dbbackup + cron	Автоматичні snapshot БД, зберігання у хмарі

Усі затверджені інструменти розповсюджуються за відкритими ліцензіями, що усуває правові перешкоди для застосування некомерційним закладом. Обмін даними між модулями реалізовано через детально описаний REST API у стандарті OpenAPI [27], що гарантує слабку зв’язність та дозволяє автономно модифікувати кожну складову системи.

Висновки до розділу 2

У другій частині роботи здійснено теоретичний аналіз та порівняння методів і технологій, які застосовуються для розв’язання визначеної проблеми.

Підтверджено доцільність використання архітектури REST [5, 6], трирівневої моделі [7] та підходу DDD [8] як фундаменту для проектування системи. Для

фіксації архітектурних рішень використано нотацію UML 2.5.1 [9]; аспекти безпеки реалізовано згідно з OWASP Top 10 [12].

На основі порівняння восьми таблиць 2.1-2.8 сформовано технологічний стек, що складається з 12 компонентів. Як фреймворк для бекенду обрано Django REST Framework [10, 11] через наявність адмін-панелі, підтримку RBAC та практичну екосистему засобів безпеки. Сесійну автентифікацію обрано замість JWT-токенам через більшу стійкість до XSS-атак та кращу сумісність із двофакторною автентифікацією (2FA) [13, 16]. Серед провайдерів обрано Google OAuth [15, 17, 18] завдяки можливості налаштування allowlist для корпоративного домену. Метод 2FA із найменшою складністю впровадження визначено як TOTP (RFC 6238) [14] з використанням бібліотеки django-otp [19]. Next.js [21] надає підтримку SSR/SSG/ISR та Metadata API. Власна CMS на базі DRF із редактором CKEditor 5 обрано замість готових рішень (Wagtail, Strapi, Contentful) через повний контроль структури даних та безшовну інтеграцію з RBAC Django. CloudFlare [22, 23] гарантує мережеву безпеку (WAF, захист від DDoS, лімітування запитів) та роботу CDN. PostgreSQL [24] та обрано відповідно як основну СУБД та систему кешування; Docker [26] використовується для контейнеризації.

3 ПРОЄКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Архітектура, структура та вхідні й вихідні дані системи

Загальна архітектура систем.

Архітектура інформаційної вебсистеми «Місто Сили» базується на трирівневому підході клієнт-сервер, про який детально йдеться у другому розділі. Інтерфейс створюється за допомогою фреймворку Next.js 15, при цьому використовуються TypeScript та App Router. Обробка бізнес-процесів виконується через Django 6.0 та Django REST Framework 3.16, а дані зберігаються в PostgreSQL 16. Ці три частини працюють окремо, обмінюючись інформацією виключно через REST API у форматі JSON, який має чітку документацію. Перед тим, як трафік дійде до наших серверів, він проходить через CloudFlare. CloudFlare діє як зворотний проксі, забезпечуючи захист від DDoS-атак, а також функції WAF та CDN (додаток А).

Що стосується розміщення системи, то обидві її частини розташовані на хостингу, який підтримує cPanel, Python 3.12 та Node.js 20. Серверна частина працює за допомогою WSGI-сервера Passenger (файл passenger_wsgi.py), а клієнтська – через Node.js-процес зі стандартним HTTP-сервером. Всі DNS-записи домену спрямовані на мережу CloudFlare Anycast, яка приховує справжню IP-адресу сервера і фільтрує вхідний трафік через WAF, перш ніж передати його на хост. Next.js та Django спілкуються двома способами: через внутрішні HTTP-запити на сервері та через публічний API у браузері.

Проєкт має монорепозиторну структуру: у головній папці diploma/ знаходяться дві окремі частини – backend/ (Django) та frontend/ (Next.js). Бекенд є модульним і складається із семи Django-додатків, які розроблені за принципом єдиної відповідальності. Кожен з них є незалежним Python-пакетом і містить власні файли, такі як models.py, serializers.py, views.py, urls.py та admin.py. Детальний список цих модулів та їхні функції можна знайти у таблиці 3.1.

Таблиця 3.1 – Django-застосунки системи та їх призначення

Застосунок	Призначення	Ключові компоненти
accounts	Автентифікація, 2FA, профіль, OAuth	User (AbstractBaseUser), UserManager, pipeline.py, setup_2fa_view, verify_2fa_view, IsRoot/IsEditor permissions
core	Спільна інфраструктура	SecurityHeadersMiddleware, Require2FAMiddleware, SessionTimeoutMiddleware, SearchView, upload_to_factory, validate_file_size, EditorAccessMixin
pages	CMS-конструктор	Page, PageSection, SectionImage, ContactInfo; PageSerializer, PageSectionSerializer, SectionImageSerializer; PageDetailView, AllPagesView, ContactInfoView
reports	Звіти організації	Report; ReportListSerializer, ReportDetailSerializer; ReportViewSet, ReportDetailView
schedule	Події організації	Event; EventListSerializer, EventDetailSerializer; EventViewSet, EventDetailView
seo	SEO та redirects	Redirect (from_path, to_url, status_code); ReportSitemap, EventSitemap, PageSitemap; RobotsView, RedirectListAPIView
backend	Конфігурація	settings.py (15 INSTALLED_APPS, 14 MIDDLEWARE), urls.py (кореневий маршрутизатор), wsgi.py

Кореневий маршрутизатор підключає всі URL-адреси різних частин програми через спеціальну команду в Django і визначає шляхи для таких речей, як карта сайту, правила для пошукових систем, відповідь на виклик OAuth і адміністративний профіль. Уся конфігурація зібрана в одному місці. Тут визначено 15 INSTALLED_APPS і 14 MIDDLEWARE у визначеному порядку. Також налаштовані: структура для створення REST-API, OAuth, захист від підбору паролів, безпека файлів cookie і детальне ведення журналу.

Інший застосунок, який взаємодіє безпосередньо з користувачами, побудований на основі Next.js і використовує сучасні можливості React. Шляхи для цього застосунку визначаються ієрархією файлів у спеціальній директорії. Головний шаблон визначає загальний вигляд сторінок: він підключає загальні стилі і компоненти верхнього та нижнього колонтитулів і додає спеціальні дані для пошукових систем безпосередньо в код сторінки.

У таблиці 3.2 перераховано всі загальнодоступні шляхи і підходи до відображення інформації для цього застосунку.

Таблиця 3.2 – Маршрути фронтенду та стратегії рендерингу.

Маршрут (App Router)	**Рендеринг**	**Призначення та особливості**
/(layout.tsx)	Server Component	Кореневий layout: Header (CMS-сторінки), Footer (ContactInfo), schema.org JSON-LD (NGO, WebSite, SearchAction). Metadata template
/page.tsx	ISR (300 с)	Головна: CMS-сторінка key='home'; PageRenderer для секцій
/[key]/page.tsx	ISR (300 с)	Динамічна CMS-сторінка; generateMetadata з API; notFound() при відсутності
/reports/page.tsx	SSR (force-dynamic)	Список звітів зі статусом PUBLISHED; картки з preview та cover
/reports/[slug]/page.tsx	SSR (force-dynamic)	Деталі звіту: HTML-вміст CKEditor через ContentWithLightbox; generateMetadata з OG
/events/page.tsx	SSR (force-dynamic)	Список подій: поділ на upcoming/past за поточною датою; EventCard з date-badge
/events/[slug]/page.tsx	SSR (force-dynamic)	Деталі події: aside з датою/часом/місцем; generateMetadata
/contacts/page.tsx	ISR (300 с)	Контакти з ContactInfo API; соцмережі, адреса, логотип
/search/page.tsx	SSR (force-dynamic)	Пошук: ?q= параметр; /api/v1/search/; grouped результати за type
/not-found.tsx	Статична	Глобальна 404-сторінка

Взаємодія між компонентами системи відбувається за такою схемою на рисунку 3.1.

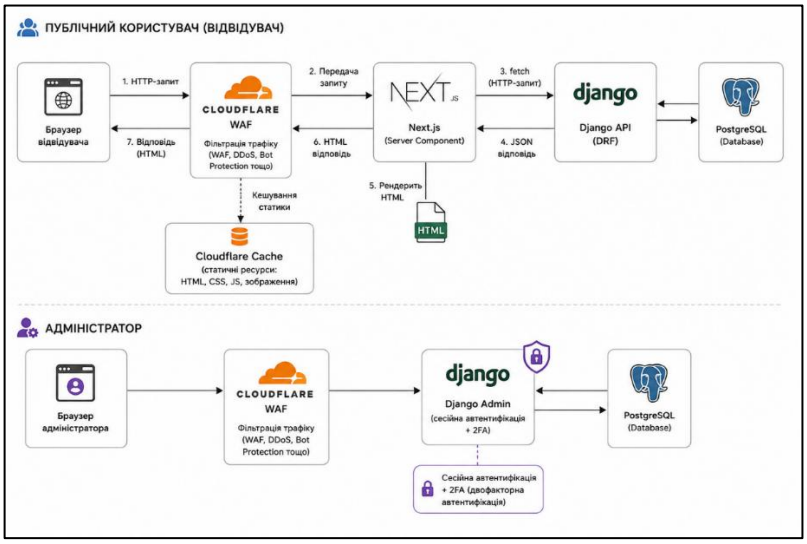

Рисунок 3.1 – Загальна схема влаємодії компонентів системи

Браузер відвідувача надсилає HTTP-запит до CloudFlare, CloudFlare WAF фільтрує трафік та передає запит на Next.js, Next.js Server Component виконує fetch до Django API. Django повертає JSON , Next.js рендерить HTML, CloudFlare кешує статику та відправляє відповідь браузеру. Для адміністратора: браузер надсилає запит до CloudFlare, він у свою чергу передає запит до Django Admin (сесійна автентифікація + 2FA), де кожний запит на створення об’єктів надсилається до БД PostgreSQL.

Класифікація вхідних даних системи.

Інформацію, що надходить до вебсистеми, можна розподілити на три головні групи залежно від її походження: адміністративні записи (заповнюються співробітниками через Django Admin), дані від користувачів публічної частини сайту (потрапляють через браузерний інтерфейс), а також системні показники (створюються зовнішніми сервісами чи середовищем). Детальна класифікація вхідних даних із вказівкою їхніх джерел, конкретними прикладами та форматами передавання представлена в таблиці 3.3.

Таблиця 3.3 – Класифікація вхідних даних системи

Категорія	**Джерело**	**Приклади вхідних даних**	**Формат**
Контент CMS (адмін)	Адміністратор (Root/Editor)	Заголовок, текст секції, зображення, URL кнопки, тип секції	Форма Django Admin
Облікові дані	Адміністратор	Email, пароль, TOTP-код (6 цифр), ім'я	Форма Django Admin
OAuth-дані	Google Identity Platform	access_token, id_token, email, sub, email_verified	JSON (OIDC token response)
Звіти та події (адмін)	Редактор	Назва, CKEditor HTML-вміст, дата, зображення, статус	Форма Django Admin
Пошуковий запит	Відвідувач порталу	Рядок q (мін. 2 символи), фільтр type	GET URL-параметри
HTTP-запит до API	Next.js Server / браузер	URL, HTTP-метод (GET), заголовки Accept/Cookie	HTTP/1.1, HTTP/2
Правила redirects (адмін)	Root-адміністратор	from_path, to_url, status_code (301/302), is_active	Форма Django Admin (POST)
Параметри середовища	DevOps / розгортання	SECRET_KEY, DATABASE_URL, GOOGLE_CLIENT_ID/SECRET	.env-файл

Категорія адміністративних вхідних даних є найбільш різноманітною. Вона охоплює текстові матеріали (заголовки, описи, HTML-код у CKEditor), медіафайли (зображення до 15 МБ у форматах JPEG, PNG, GIF, WebP), структуровану інформацію (тип розділу, статус публікації, порядок відображення, код перенаправлення) та облікові дані (електронна пошта, пароль, TOTP-код). Усі форми для адміністрування захищені CSRF-токеном і потребують активної сесії з підтвердженим двофакторним аутентифікаційним кодом. Медіафайли проходять перевірку розміру (валідація через validate_file_size, ліміт 15 МБ), а їхні назви безпечно транслітеруються та рандомізуються за допомогою функції safe_filename перед збереженням.

Вхідні дані відвідувачів обмежені пошуковими запитами (рядок мінімум 2 символи, що передається як GET-параметр q) та стандартними HTTP-запитами до публічних ендпоінтів. Публічний API є виключно read-only: відвідувачі не можуть змінювати дані в базі. Будь-який POST, PUT або DELETE до публічних маршрутів повертає HTTP 405 Method Not Allowed. Пошукові запити обмежені throttle-класом SearchRateThrottle (20 запитів на хвилину для анонімних клієнтів), що захищає від масового скрапінгу.

Зовнішні вхідні дані надходять від Google Identity Platform під час OAuth 2.0 авторизації: access_token, id_token (JWT), email, sub (унікальний Google UID) та email_verified. Ці дані проходять обов’язкову валідацію в кастомному pipeline-кроці require_allowlist: перевіряється верифікованість email, відповідність домену allowlist та наявність явного дозволу google_login_allowed у моделі User. Системні вхідні дані (параметри середовища) завантажуються з .env-файлу бібліотекою django-environ при запуску застосунку.

Класифікація вихідних даних систем.

Вихідні дані системи також розподіляються за кількома категоріями залежно від одержувача та формату. Класифікацію вихідних даних наведено у таблиці 3.4.

Таблиця 3.4 – Класифікація вихідних даних системи

Тип вихідних даних	**Одержувач**	**Склад та особливості**
JSON-відповідь REST API	Next.js SSR / браузер	Серіалізовані об'єкти моделей: звіти, події, сторінки, контакти, результати пошуку; статус HTTP 200/404/429
HTML-сторінка (SSR)	Браузер відвідувача	Повністю розмічена HTML-сторінка з SEO-метатегами та JSON-LD schema.org у <head>
HTML-сторінка (ISR)	Браузер / CDN-кеш	Кешована статично-згенерована сторінка; регенерується у фоні після revalidate
sitemap.xml	Пошукові боти (Googlebot)	XML-документ з URLs сторінок, дат оновлення, changefreq та priority
robots.txt	Пошукові боти	Текстовий файл: Allow:/, Disallow:/admin/, /auth/, /api/; Sitemap URL
HTTP-redirect 301/302	Браузер / пошуковий бот	Заголовок Location з URL призначення; статус 301 або 302
QR-код	Браузер адміністратора	PNG-зображення otpauth для сканування застосунком-автентифікатором
HTTP-заголовки безпеки	Браузер	CSP, X-Frame-Options, Referrer-Policy, Permissions-Policy, HSTS
Резервна копія БД	Файлова система сервера	Стиснений SQL-дамп PostgreSQL з іменем {DB}_{timestamp}.sql.gz
Логи	Адміністратор системи	Текстові рядки формату [timestamp] LEVEL module message у logs/django.log

Основним типом вихідних даних для кінцевих відвідувачів є HTML-сторінки, що генеруються Next.js на сервері. Залежно від рендерингу вони можуть бути SSR (генеруються при кожному запиті) або ISR (кешовані, регенеруються у фоні). Усі HTML-відповіді містять у розділі head повний набір SEO-метатегів (title, description, canonical, Open Graph, Twitter Card) та JSON-LD schema.org. Це гарантує

коректну індексацію пошуковими системами та правильне відображення при поширенні у соціальних мережах.

Результати запиту REST API у форматі JSON слугують джерелом інформації як для серверних, так і для клієнтських компонентів у Next.js. Серіалізовані структури даних містять лише обов’язкові атрибути: для переліку використовується ListSerializer із мінімальним набором полів, тоді як DetailSerializer надає повну інформацію, включаючи HTML-код від CKEditor. Безпечні HTTP-заголовки призначені для браузера, наказуючи йому обмежувати завантаження ресурсів, блокувати вбудовування сторінки в iframe та примусово використовувати захищене HTTPS-з’єднання протягом вказаного періоду. Архівні копії та журнали подій є вихідними даними для системного адміністратора.

3.2 Опис реалізації підсистем

Підсистема автентифікації та авторизації.

Архітектурно найскладнішим компонентом серверної частини є модуль автентифікації, який інтегрує низку автономних систем: кастомну модель User із RBAC, керування сесіями, OAuth від Google з білим списком, двофакторну верифікацію TOTP та захист від підбору пар. Ці елементи синхронізуються через стек Django middleware та OAuth-конвеєр, створюючи багатошаровий бар'єр для адмін-панелі.

Клас User базується на *AbstractBaseUser* і *PermissionsMixin*, що дає змогу повністю замінити стандартну модель Django, зберігаючи інтеграцію з механізмом дозволів. Замість традиційного імені користувача ключовим полем ідентифікації *USERNAME_FIELD* обрано електронну пошту, що відповідає сучасним стандартам та усуває потребу в управлінні двома різними ідентифікаторами.

Рольова модель реалізована через перелік Role, який містить статуси Root (повний контроль) і Editor (доступ лише до CMS-модулів). Змінна *EDITOR_APPS* є єдиним місцем, де прописано список програм для редакторів. Такий підхід спрощує масштабування прав Editor: достатньо додати новий рядок до цього словника.

Механізм контролю доступу базується на двох перевизначених методах моделі: *has_perm()* та *has_module_perms()*. Перший із них перевіряє дозвіл на виконання конкретної, тоді як другий – на доступ до застосунку загалом. Логіка обробки однакова для обох: суперкористувач завжди отримує доступ, для редактора перевіряється наявність у *EDITOR_APPS*, а неактивний користувач завжди отримує відмову. Реалізацію цих методів наведено на рисунку 3.2.

```
def has_perm(self, perm, obj=None):
    if not self.is_active:
        return False
    if self.is_superuser or self.is_root:
        return True
    if self.is_editor:
        app_label = perm.split('.')[0] if '.' in perm else perm
        return app_label in self.EDITOR_APPS
    return False
```

Рисунок 3.2 – Функція has_perm

Функція save() моделі автоматично узгоджує значення прапорців *is_staff* та *is_superuser* з полем *role* під час кожного збереження даних. Це забезпечує відновлення правильних статусів навіть після ручних змін через SQL-запити чи інтерактивну оболонку. Користувач Root отримує обидва прапорці, тоді як Editor має лише *is_staff*.

На рівні адміністративної панелі доступ регулюється класом *EditorAccessMixin*, який застосовується до всіх ModelAdmin через множинне успадкування. Він перевизначає методи *has_add_permission(), has_change_permission()* та *has_delete_permission()*, передаючи перевірку користувачеві. Така подвійна верифікація гарантує, що Editor не зможе виконати дії навіть при спробі прямого звернення до URL.

Для REST API аналогічний захист реалізують два класи дозволів DRF: *IsRoot* та *IsEditor*. Вони призначаються ViewSet-класам через атрибут *permission_classes*, що дозволяє захищати окремі ендпоінти або конкретні методи.

Рисунок 3.3 – Django Admin – форма редагування користувача

Інтеграція Google OAuth здійснюється за допомогою бібліотеки social-auth-app-django з використанням бекенду GoogleOAuth2. Процес авторизації за схемою Authorization Code Flow виглядає так: користувач натискає кнопку «Увійти через Google», після чого браузер перенаправляється на екран погодження Google. Після підтвердження Google повертає код, який social-auth обмінює на ID-токен, ініціюючи виконання *SOCIAL_AUTH_PIPELINE*.

Ключовою ланкою цього піплайну є кастомний етап *require_allowlist*, що реалізує триетапну перевірку. Перший етап перевіряє статус email_verified – Google гарантує цей статус для акаунтів Google Workspace. Другий етап перевіряє, чи входить домен електронної пошти до списку *ALLOWED_DOMAINS*. Третій, найважливіший етап, перевіряє поле *google_login_allowed* у конкретному обліковому записі User – це дозволяє Root вручно надавати доступ до входу для кожного акаунта індивідуально. Повний код функції представлено на рисунку 3.4.

```
def require_allowlist(backend, user, response, social, *args, **kwargs):
    if not response.get('email_verified'):
        raise AuthForbidden(backend)
    email = response.get('email', '')
    domain = email.split('@')[-1] if '@' in email else ''
    if domain not in ALLOWED_DOMAINS:
        raise AuthForbidden(backend)
    if not user or not user.google_login_allowed:
        raise AuthForbidden(backend)
    google_sub = response.get('sub')
    if google_sub and not user.google_sub:
        user.google_sub = google_sub
        user.save(update_fields=['google_sub'])
    return {'user': user}
```

Рисунок 3.4 – Функція require_allowlist

Значення False для параметра *SOCIAL_AUTH_GOOGLE_OAUTH2_CREATE_USERS* суворо забороняє автоматичне створення облікових записів через протокол OAuth. Користувацький профіль має бути заздалегідь сформований адміністратором із правами Root у інтерфейсі Django Admin, де обов'язково встановлюється прапорець *google_login_allowed*. Така конфігурація забезпечує реалізацію підходу «заборонено все, що не дозволено»: доступ надається лише тим, хто отримав чіткий дозвіл. Кастомний екран авторизації з відповідною кнопкою для Google OAuth зображено на рисунку 3.4.

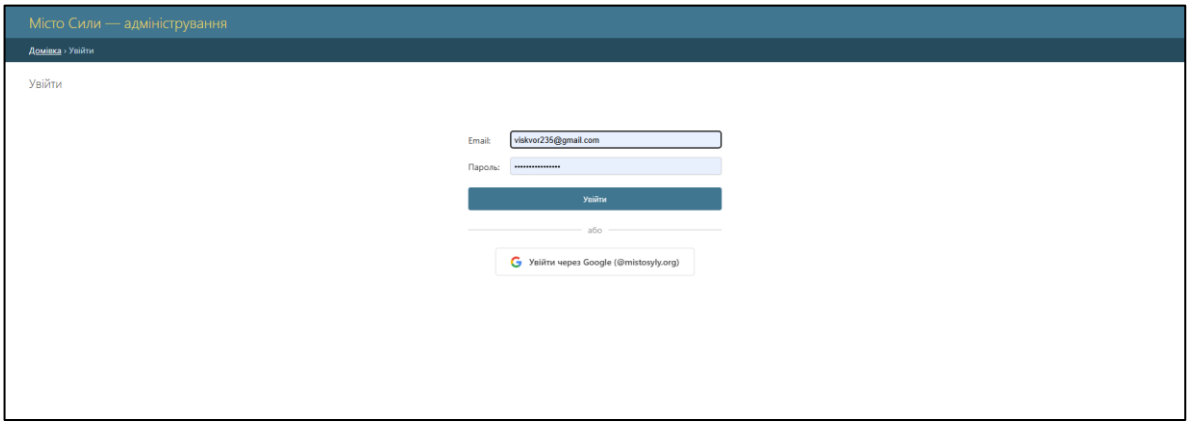

Рисунок 3.5 – Кастомна сторінка входу Django Admin з OAuth-кнопкою

Двофакторна автентифікація та middleware безпеки.

Механізм двофакторної автентифікації реалізовано за допомогою бібліотеки *django-otp* із модулем *otp_totp*. Об'єкт TOTPDevice зберігає спільний секрет, посилання на QR-код та прапорець *confirmed*, який розрізняє активовані та нові пристрої. Алгоритм TOTP (RFC 6238) формує шестизначний код на базі спільного секрета та поточного Unix-часу, округленого до 60-секундних інтервалів.

Під час первинного ввімкнення 2FA функція *setup_2fa_view* видаляє всі непідтверджені записи поточного користувача та створює новий екземпляр TOTPDevice із статусом *confirmed* зі значеням False. Далі генерується QR-код: значення *device.config_url* передається у бібліотеку *qrcode*, отриманий результат кодується у base64 (формат PNG) і вставляється у шаблон як *data URI*. Після сканування коду мобільним додатком користувач вводить перший код. Сервер перевіряє його за допомогою *device.verify_token(code)*. Якщо результат позитивний, статус *confirmed* змінюється на True. Інтерфейс сторінки налаштування 2FA зображено на рисунку 3.6.

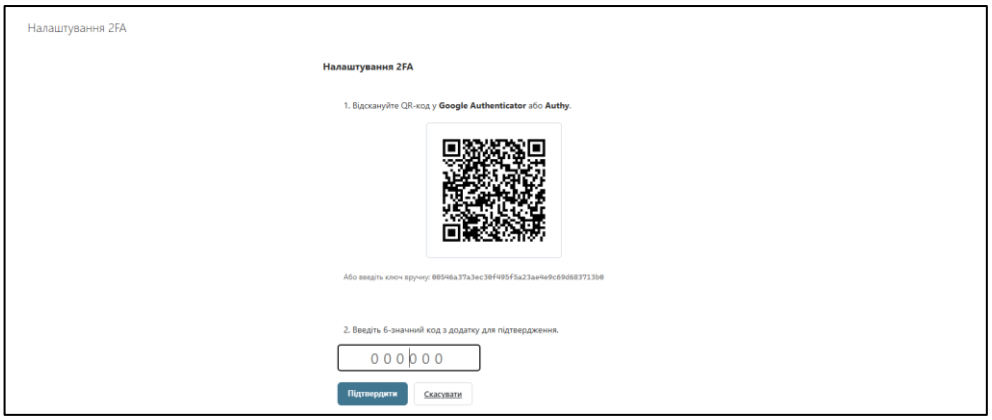

Рисунок 3.6 – Сторінка налаштування TOTP 2FA у профілі користувача

Для примусової перевірки двофакторної автентифікації (2FA) під час кожного входу застосовуються два послідовні проміжні обробники. Спершу *OTPMiddleware* з'ясовує, чи пройшов верифікацію пристрій OTP у межах поточної сесії. Далі *Require2FAMiddleware* перехоплює запити до будь-яких адрес адмін-панелі та оцінює три критерії: статус *is_staff*, наявність підтвердженого пристрою TOTPDevice та відсутність верифікації. Якщо хоча б одна умова не виконується,

користувача перенаправляє на сторінку верифікації. Реалізацію цього middleware представлено на рисунку 3.7.

```
class Require2FAMiddleware:
    EXEMPT_URLS = (
        '/admin/login/', '/admin/logout/',
        '/admin/profile/2fa/verify/', '/auth/',
    )
    def __call__(self, request):
        if (request.path.startswith('/admin/')
            and not any(request.path.startswith(u)
                        for u in self.EXEMPT_URLS)
            and request.user.is_authenticated
            and request.user.is_staff):
            has_device = TOTPDevice.objects.filter(
                user=request.user, confirmed=True).exists()
            if has_device and not request.user.is_verified():
                return redirect(
                    f'/admin/profile/2fa/verify/?next={request.path}')
        return self.get_response(request)
```

Рисунок 3.7 – Клас Require2FAMiddleware

Клас *SessionTimeoutMiddleware* забезпечує функціонал автоматичного виходу з системи за принципом рухомого вікна: таймер неактивності активується від моменту останньої взаємодії клієнта, а не від часу авторизації. Щоразу надходження нового запиту middleware витягує значення *session['last_activity']*, зіставляє його з актуальним часом через *time.time()* та, у разі перевищення ліміту у 8 годин, ініціює виклик *logout(request)* та перенаправляє користувача на сторінку авторизації із параметром *?timeout=1*. В іншому випадку оновлюється поле *last_activity*. Повний варіант коду наведено на рисунку 3.8.

```
class SessionTimeoutMiddleware:
    def __call__(self, request):
        if request.user.is_authenticated:
            last_activity = request.session.get('last_activity')
            now = time.time()
            if last_activity and (now - last_activity > 60 * 60 * 8):
                logout(request)
                return redirect('admin/login/?timeout=1')
            request.session['last_activity'] = now
        return self.get_response(request)
```

Рисунок 3.8 – Клас SessionTimeoutMiddleware

Механізм захисту від підбору паролів забезпечується за допомогою бібліотеки *django-axes*. Компонент *AxesMiddleware* перехоплює POST-запити, спрямовані до */admin/login/*, та фіксує невдалі спроби авторизації в таблиці *axes_accessattempt*. Параметр *AXES_LOCKOUT_PARAMETERS* визначає блокування за ім'ям користувача, що унеможливлює обхід захисту через зміну IP-адреси чи використання проксі-серверів. Після п'яти невдалих спроб протягом однієї години відображається кастомна сторінка *lockout.html*. Прапор *AXES_RESET_ON_SUCCESS* передбачає скидання лічильника після успішного входу. Зовнішній вигляд сторінки блокування зображено на рисунку 3.9.

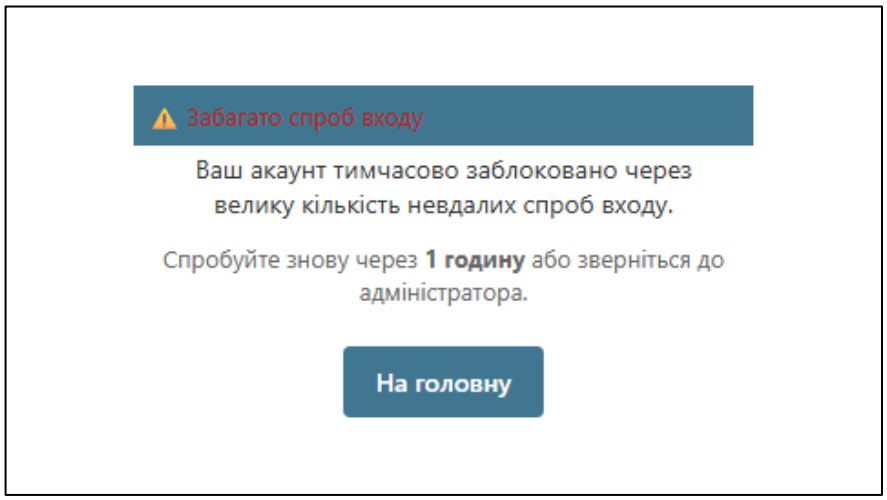

Рисунок 3.9 – Сторінка блокування django-axes після перевищення ліміту спроб входу

SecurityHeadersMiddleware додає до кожної HTTP-відповіді п'ять захисних заголовків. Content-Security-Policy є найважливішим: директива *default-src 'self'* забороняє завантаження ресурсів з будь-яких зовнішніх джерел за замовчуванням, *frame-ancestors 'none'* забороняє вбудовування сторінки в *iframe* будь-яким сайтом. Permissions-Policy явно відмовляється від шести Browser API: геолокації, камери, мікрофону, платіжних API, USB та магнетометра. Код middleware представлено на рисунку 3.10.

```
class SecurityHeadersMiddleware:
    def __call__(self, request):
        response = self.get_response(request)
        response['X-Content-Type-Options'] = 'nosniff'
        response['X-Frame-Options'] = 'DENY'
        response['Referrer-Policy'] = 'strict-origin-when-cross-origin'
        response['Permissions-Policy'] = (
            'geolocation=(), camera=(), microphone=(), '
            'payment=(), usb=(), magnetometer=()'
        )
        response['Content-Security-Policy'] = (
            "default-src 'self'; script-src 'self' 'unsafe-inline'; "
            "style-src 'self' 'unsafe-inline'
https://fonts.googleapis.com; "
            "img-src 'self' data: blob: https:; "
            "connect-src 'self'; frame-ancestors 'none';"
        )
        return response
```

Рисунок 3.10 – Клас SecurityHeadersMiddleware

CMS-конструктор сторінок.

Інструмент для створення веб-сторінок виступає центральним компонентом системи, який надає можливості нетехнічним співробітникам розробляти публічні ресурси без необхідності редагування програмного коду. Архітектура рішення спирається на три основні моделі Django: Page, PageSection та SectionImage. Механізм каскадного видалення гарантує, що при знищенні сторінки автоматично видаляються й усі пов’язані з нею секції та медіафайли.

Атрибут *key* слугує технічним ідентифікатором для API та формує структуру URL-адрес. Наприклад, значення *'about'* визначає шлях */about*, тоді як *'home'* відсилає до головної сторінки. Така реалізація обмежує створення нових ресурсів виключно через інтерфейс адміністратора.

Компонент PageSection підтримує шість варіантів типів блоків: hero, text, gallery, cta, contacts та custom. Динамічна зміна видимості полів у адмін-формі здійснюється завдяки скрипту *section_admin.js*, який фільтрує елементи керування відповідно до обраного типу секції.

Інтерфейс адміністрування побудовано на основі *django-nested-admin* із використанням вкладених inline-редакторів. Це забезпечує зручність додавання, впорядкування та редагування змісту секцій у єдиному вікні, а також дозволяє

завантажувати зображення для галереї та призначати до них текстові підписи (рис. 3.11).

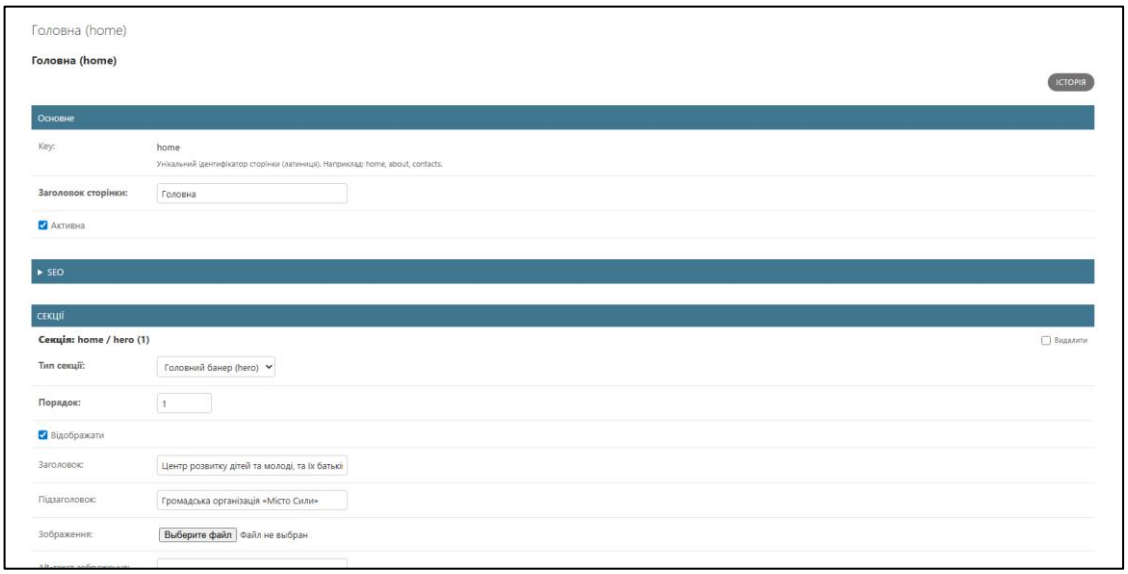

Рисунок 3.11 – Адмін-інтерфейс CMS-конструктора з nested inline секціями

Для зберігання вмісту полів секцій використовується CKEditor5Field із налаштуваннями, що розширюють функціонал: підтримуються заголовки, стилізація тексту, списки, таблиці, медіа-вставки, колірні акценти, блоки коду та режим редагування HTML. Щоб уникнути XSS-вразливостей, дані автоматично проходять очищення.

На клієнтській стороні компонент PageRenderer отримує секції через API, а функція *renderSection()* за допомогою switch-оператора динамічно підключає потрібний React-компонент. Оскільки більшість секцій працюють як React Server Components, це суттєво знижує розмір JavaScript-коду в браузері та прискорює завантаження сторінки. Функцію renderSection() показано на рисунку 3.12.

```
export function renderSection(section: Section, index = 0) {
  const isAboveFold = index === 0;
  switch (section.section_type) {
    case 'hero':
      return <HeroSection key={section.id} section={section} />;
    case 'text':
      return <TextSection key={section.id} section={section}
               isAboveFold={isAboveFold} />;
    case 'gallery':
      return <GallerySectionClient key={section.id}
               title={section.title} images={section.images} />;
    case 'cta':
      return <CTASection key={section.id} section={section} />;
    case 'contacts':
      return <ContactsSection key={section.id} section={section} />;
    default:
      return null;
  }
}
```

Рисунок 3.12 – Функція renderSection

Метод *fixMediaUrls()* усуває критичну проблему з відображенням зображень: CKEditor зберігає шляхи у форматі */media/...*, тоді як Next.js та браузер очікують абсолютний URL бекенду. Функція замінює *src="/media/* на *src="{API_URL}/media/* у HTML-рядках регулярним виразом перед передачею у *dangerouslySetInnerHTML*. Результат рендерингу CMS-сторінки на публічному порталі показано на рисунку 3.13.

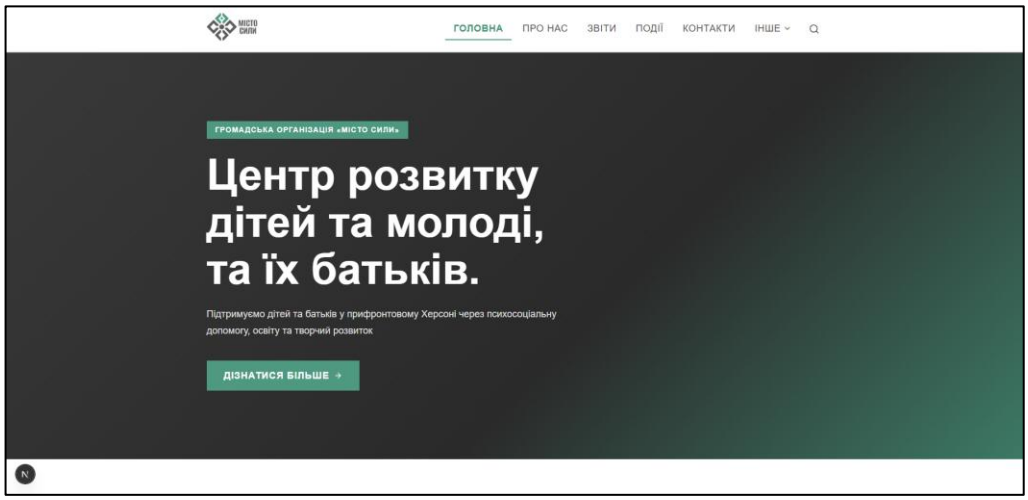

Рисунок 3.13 – Публічний портал – головна сторінка з CMS-секціями hero та text

REST API та пошукова система.

Публічний REST API системи містить 17 ендпоінтів під префіксом */api/v1/*. Маршрути визначені через явні path-патерни без DefaultRouter. Усі публічні ендпоінти підтримують лише GET-запити, а POST і DELETE повертають HTTP 405.

Для кожного ресурсу використовуються два серіалізатори: ListSerializer для списків із мінімальним набором полів та DetailSerializer для сторінок деталей із повним вмістом. Це зменшує розмір JSON-відповідей і пришвидшує завантаження сторінок.

У DetailView перевизначено метод *get_object()*, який перетворює Http404 у NotFound з JSON-відповіддю. Це забезпечує коректну обробку помилок у Next.js SSR (рис. 3.14).

```
class ReportDetailView(generics.RetrieveAPIView):
    queryset = Report.objects.filter(
        status=Report.Status.PUBLISHED)
    serializer_class = ReportDetailSerializer
    lookup_field = 'slug'

    def get_object(self):
        try:
            return super().get_object()
        except Http404:
            raise NotFound('Report not found')
```

Рисунок 3.14 – Клас ReportDetailView

Пошукова підсистема реалізована у класі SearchView на основі DRF APIView. Метод *get()* обробляє параметри *q* та *type*, а пошук виконується через Q-запити з *icontains* по текстових полях моделей Report, Event і Page. Для сторінок пошук також охоплює поля пов’язаних секцій через JOIN.

Результати з усіх моделей об’єднуються в єдиний масив results, де кожен елемент містить поле type. Це дозволяє фронтенду відображати тип результату без додаткових запитів (рис. 3.15).

```
class SearchView(APIView):
    throttle_classes = [SearchRateThrottle]
    def get(self, request):
        query = request.GET.get('q', '').strip()
        if len(query) < 2:
            return Response({'results': [], 'total': 0,
                'error': 'Мін. 2 символи.'})
        results = []
        reports = Report.objects.filter(
            status=Report.Status.PUBLISHED).filter(
            Q(title__icontains=query) |
            Q(preview__icontains=query) |
            Q(content__icontains=query))[:10]
        results += ReportSearchSerializer(
            reports, many=True,
            context={'request': request}).data
        # ... аналогічно для Event та Page ...
        return Response({'query': query,
            'results': results, 'total': len(results)})
```

Рисунок 3.15 – Клас SearchView

Ендпоінт пошуку захищений окремим throttle-класом *SearchRateThrottle* зі *scope='search'* та лімітом 20 запитів на хвилину – суворіший, ніж загальний анонімний ліміт 60/хв. Це захищає від масового скрапінгу, оскільки пошук виконує кілька JOIN-запитів одночасно. Результати пошуку на фронтенді показано на рисунку 3.16.

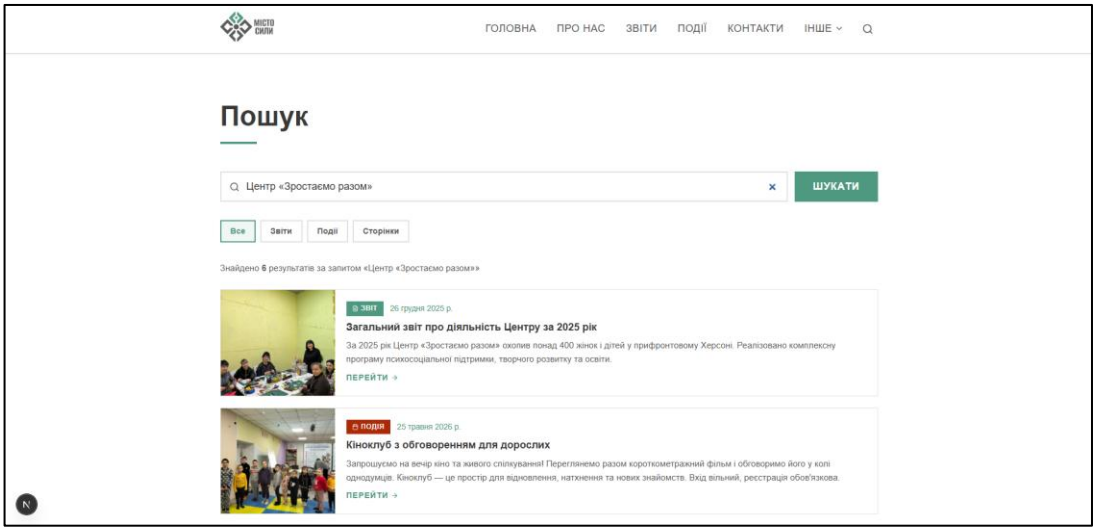

Рисунок 3.16 – Сторінка пошуку з результатами та типовими badges

Підсистема звітів та подій.

Модель Report передбачає три варіанти статусу: DRAFT, HIDDEN і PUBLISHED. Для пришвидшення фільтрації записів, які вже опубліковані, поле status оснащено індексом бази даних.

Функція save() виконує автоматичне створення унікального слаг-назви, використовуючи транслітерацію та перевірку на наявність подібних записів із додаванням числового суфікса. Крім того, під час першого опублікування запису автоматично фіксується час публікації через функцію *timezone.now()*.

Функція *slugify()* є кастомною реалізацією: вона спочатку застосовує *unidecode()* для транслітерації кирилиці в латиницю. Наприклад, «Річний звіт 2025» змінюється на richnii-zvit-2025. Вбудована *slugify()* у Django видаляє нелатинські символи, тому для українських назв може повертати порожній рядок.

ReportAdmin наслідує *EditorAccessMixin* і *ShortTitleMixin*. Параметр *list_editable* дозволяє змінювати статус прямо в списку, без відкриття форми. Для редагування використовується CKEditor5Widget з розширеним тулбаром (рис. 3.17).

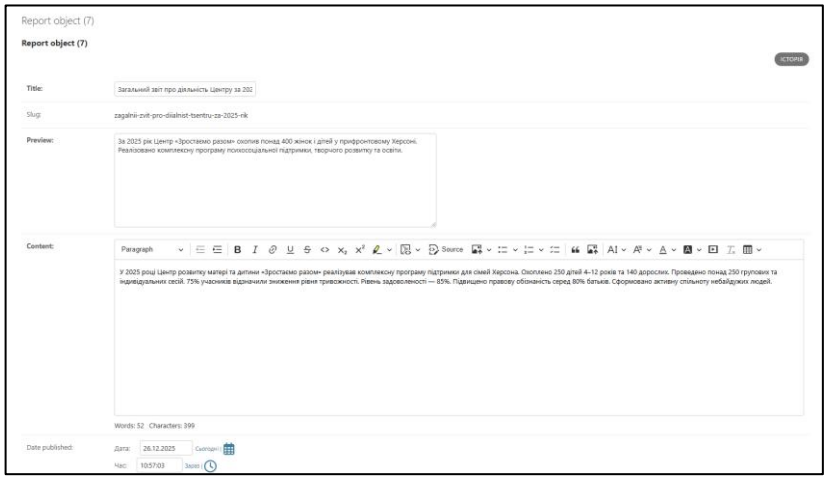

Рисунок 3.17 – Форма редагування звіту в Django Admin з CKEditor 5

Сторінка деталей звіту рендерить HTML з CKEditor через компонент *ContentWithLightbox*. Він обробляє кліки по зображеннях у контенті і відкриває їх у Lightbox для перегляду у збільшеному вигляді без переходу на інші сторінки (рис. 3.18).

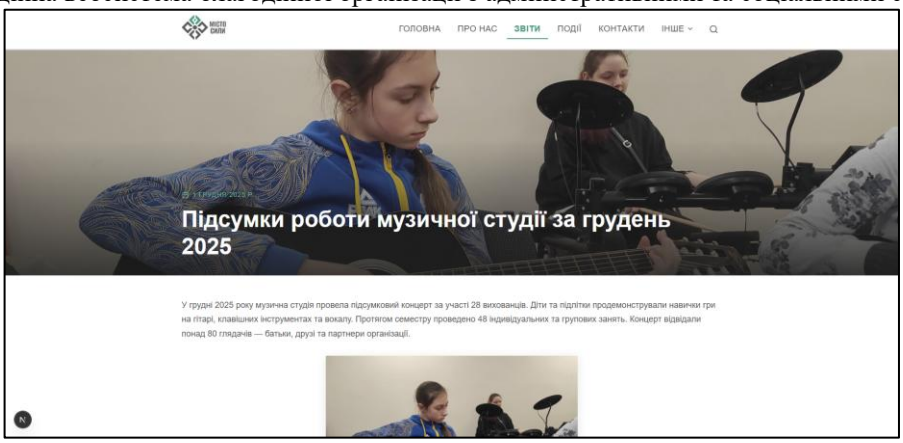

Рисунок 3.18 – Публічна сторінка деталей звіту

Модель Event містить поля: title, slug (автогенерований як у Report), description, date, location та status (DRAFT/PUBLISHED).

На сторінці */events* події поділяються на майбутні та минулі за датою. Для минулих подій *EventCard* відображається з меншою прозорістю та нейтральним кольором дати.

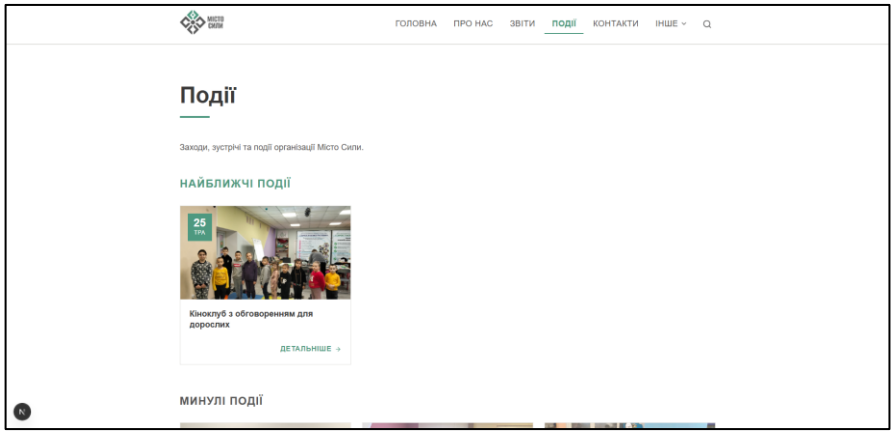

Рисунок 3.19 – Публічна сторінка подій із розподілом на майбутні та минулі

SEO, sitemap та управління перенаправленнями.

SEO-підсистема складається з трьох основних компонентів:

– автоматично згенерований sitemap.xml через django.contrib.sitemaps. Використовуються ReportSitemap, EventSitemap і PageSitemap, які визначають URL-адреси, lastmod, changefreq і priority для коректної індексації;

– robots.txt, який генерується динамічно через RobotsView. Він обмежує індексацію службових шляхів і автоматично формує правильний шлях до

sitemap.xml залежно від середовища виконання, що забезпечує коректну роботу в різних доменах;

– структуровані дані schema.org у форматі JSON-LD, інтегровані в layout.tsx. Вони включають схеми Organization, WebSite і SearchAction, які покращують SEO та відображення сайту в Google. Дані для Organization підтягуються через API на сервері під час рендеру layout (рис. 3.20).

```
const schema = {
  '@context': 'https://schema.org',
  '@graph': [
    { '@type': 'NGO',
      name: contacts?.organization_name,
      url: siteUrl,
      contactPoint: { '@type': 'ContactPoint',
        email: contacts?.email,
        telephone: contacts?.phone },
      sameAs: [contacts?.facebook_url,
               contacts?.instagram_url].filter(Boolean) },
    { '@type': 'SearchAction',
      target: `${siteUrl}/search?q={search_term_string}`,
      'query-input': 'required name=search_term_string' },
  ]
};
```

Рисунок 3.20 – OrganizationSchema (спрощено)

SEO-метадані сторінок реалізовані через Next.js Metadata API. У кореневому layout задано шаблон заголовка: title: { default: 'Місто Сили', template: '%s | Місто Сили' }.

Статичні сторінки використовують *export const metadata*, де визначаються title, description і openGraph. Динамічні сторінки застосовують *generateMetadata({ params })*, яка отримує дані з API та формує Open Graph на основі *og_image* і *meta_title* конкретного запису. Приклад метатегів у *head* показано на рисунку 3.21.

```
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Головна сторінка</title>
<meta name="description" content="Головна сторінка центру розвитку матері та дитини «Зростаємо разом»">
<meta name="author" content="Місто Сили">
<meta name="keywords" content="громадська організація,місто сили,спільнота, волонтери">
<meta property="og:title" content="Головна сторінка">
<meta property="og:description" content="Головна сторінка центру розвитку матері та дитини «Зростаємо разом»">
<meta property="og:site_name" content="Місто Сили">
<meta property="og:locale" content="uk_UA">
<meta property="og:type" content="website">
```

Рисунок 3.21 – Згенеровані SEO-метатеги

Управління HTTP-перенаправленнями здійснюється через модель Redirect у Django Admin, де визначаються *in_path*, *to_url* та тип редиректу (301 чи 302).

Middleware Next.js отримує активні правила через API та зберігає їх у структурі Map для швидкого доступу. Передбачено кешування з часом життя 60 секунд для оптимізації кількості запитів. Крім того, підтримуються варіанти URL із та без фінального слеша, щоб уникнути дублювання правил.

```
const CACHE_TTL_MS = 60_000;
let cachedLookup: Map<...> | null = null;
let cacheExpiresAt = 0;

async function getRedirectLookup() {
  const now = Date.now();
  if (cachedLookup && now < cacheExpiresAt)
    return cachedLookup;
  const res = await fetch(`${API_URL}/api/v1/redirects/`,
    { next: { revalidate: 60 } });
  const rules = await res.json();
  cachedLookup = buildRedirectLookup(rules);
  cacheExpiresAt = now + CACHE_TTL_MS;
  return cachedLookup;
}
```

Рисунок 3.22 – Функція getRedirectLookup() з TTL-кешем

Утиліти та резервне копіювання.

Модуль *core/utils.py* містить спільні утиліти. Функція *upload_to_factory()* реалізує Factory Pattern для створення upload_to-функцій із різними шляхами збереження. Вона транслітерує ім’я файлу (unidecode, slugify) і додає 8-символьний UUID для унікальності. Метод *deconstruct()* забезпечує підтримку міграцій Django.

```
def safe_filename(filename: str) -> str:
    name, ext = os.path.splitext(filename)
    safe = slugify(unidecode(name)) or f'file_{uuid.uuid4().hex[:8]}'
    return f'{safe}_{uuid.uuid4().hex[:8]}{ext.lower()}'

def upload_to_factory(folder: str):
    def upload_to(instance, filename):
        return posixpath.join(
            folder.strip('/'), safe_filename(filename))
    upload_to.deconstruct = lambda: (
        f'core.utils.upload_to_factory("{folder}")', [], {})
    return upload_to
```

Рисунок 3.22 – Функції safe_filename та upload_to_factory

Резервне копіювання PostgreSQL реалізовано Bash-скриптом *backup.sh*. Функція *load_backup_env()* безпечно читає .env (без source), ігноруючи коментарі та порожні рядки, та експортує лише потрібні змінні.

Бекап створюється через *pg_dump* з передачею пароля через змінну середовища та стискається в *gzip*. Перевірка відновлення використовує trap EXIT для очищення тестової БД. Скрипт запускається щодня через cron о 3:00.

3.3 Аналіз отриманих результатів

Відповідність функціональним вимогам.

У розділі 1.3 кваліфікаційної роботи сформульовано 18 функціональних вимог трьох рівнів пріоритету до розроблюваної системи: критичні (4 вимоги), високого пріоритету (10 вимог) та середнього пріоритету (4 вимоги). Критичні вимоги визначено на основі аналізу наукових джерел [4] щодо зростання шахрайств у благодійному секторі та необхідності надійних механізмів захисту. Вимоги публічної звітності та прозорості обґрунтовані дослідженнями Cooley [2] та Ortega-Rodríguez et al. [3], що встановили низький рівень онлайн-підзвітності українських НУО. Відповідність реалізованого функціоналу вимогам наведена у таблиці 3.5.

Таблиця 3.5 – Відповідність реалізованого функціоналу функціональним вимогам

№	Функціональна вимога (розд. 1.3)	Пріоритет	Реалізація у системі
1	Автентифікація з підтримкою TOTP 2FA	Критична	django-otp, TOTPDevice, setup/verify views, Require2FAMiddleware
2	Повноцінна адмін-панель з CRUD-операціями	Критична	Django Admin, EditorAccessMixin, nested-admin, CKEditor5Widget
3	Інтеграція CloudFlare WAF (захист від DDoS)	Критична	CloudFlare Proxy, WAF Managed Ruleset, Rate Limiting, Edge CDN
4	HTTPS, rate limiting, захист HTTP-заголовків OWASP	Критична	SecurityHeadersMiddleware (CSP, HSTS, X-Frame), DRF throttling, SECURE_* settings
5	Публічний вебпортал з інформацією про організацію	Висока	Next.js /, /[key], /contacts; CMS-секції hero/text/cta/contacts
6	Розділ публічної звітності з документами	Висока	/reports, /reports/[slug]; Report + CKEditor5Field; статуси draft/published/hidden

Кінець таблиці 3.5

№	Функціональна вимога (розд. 1.3)	Пріоритет	Реалізація у системі
7	CMS без необхідності програмування	Висока	PageAdmin + nested-admin + section_admin.js; 6 типів секцій; PageRenderer
8	Задокументований REST API	Висока	17 ендпоінтів DRF; drf-spectacular (OpenAPI 3.0); Swagger UI на /api/schema/swagger-ui/
9	Управління медіафайлами через CMS	Середня	upload_to_factory, validate_file_size (15 МБ), SectionImage inline, CKEditor imageUpload
10	SSR/SSG для підвищення SEO	Середня	Next.js App Router: SSR (force-dynamic) + ISR (revalidate:300); Metadata API; generateMetadata()
11	CDN для доставки статичних ресурсів	Середня	CloudFlare CDN: _next/static, /media/ кешуються на Edge-вузлах; Cache-Control заголовки
12	Пошук по сайту	Висока	SearchView з Q-фільтрами по Reports+Events+Pages; SearchRateThrottle 20/хв; type-фільтр
13	Автоматична генерація sitemap.xml	Висока	ReportSitemap + EventSitemap + PageSitemap; priority / changefreq / lastmod з моделей
14	Управління HTTP-redirects через адмінку	Висока	Модель Redirect; Next.js proxy.ts з Map-кешем TTL 60с; 301/302
15	Рольова модель доступу (RBAC)	Критична	User.role (Root/Editor); has_perm / has_module_perms; EDITOR_APPS; EditorAccessMixin
16	Захист від брутфорсу	Критична	django-axes: AXES_FAILURE_LIMIT=5, AXES_COOLOFF_TIME=1год, lockout.html
17	Структуровані дані schema.org	Середня	JSON-LD: NGO, WebSite, SearchAction у layout.tsx через OrganizationSchema компонент
18	Резервне копіювання БД	Середня	backup.sh: pg_dump+gzip, KEEP_DAYS=30, щомісячна верифікація відновлення, cron 3:00

Досвідчення даних таблиці 3.5 підтверджує, що у версії 1.0.0 програмного комплексу виконано усі 18 задекларованих вимог, що відповідає рівню реалізації 100%. У повному обсязі забезпечено чотири ключові функції: двофакторну аутентифікацію TOTP (з генеруванням QR-кодів за допомогою django-otp та Require2FAMiddleware), адміністративний інтерфейс із підтримкою вкладених структур (nested-admin) та редактора CKEditor5Widget, захист через WAF Managed

Ruleset від CloudFlare, а також налаштування безпеки заголовків через SecurityHeadersMiddleware відповідно до стандартів OWASP. Задачі високого пріоритету, зокрема публічний вебпортал, модуль аналітики, конструктор CMS, REST-інтерфейс, пошукова система, генерація sitemap та redirects, система RBAC, захист від перебору паролів та семантична розмітка schema.org, реалізовано за допомогою підсистем, детально описаних у розділі 3.2. Для виконання завдань середнього пріоритету (робота з медіа, SSR/SEO-оптимізація, CDN та резервне копіювання) використано інструменти upload_to_factory, механізми ISR/SSR у Next.js, сервіс CloudFlare CDN та скрипт backup.sh.

Паралельно із базовим набором функцій у ході розробки було впроваджено додаткові покращення, що перевищують початкові технічні специфікації: адаптовані шаблони Django Admin (зокрема login.html з кнопкою OAuth та SVG-значком Google, сторінки profile.html, setup_2fa.html та lockout.html), віджет ContentWithLightbox для перегляду зображень у редакторі CKEditor, обмеження на видалення записів ContactInfo (singleton-режим через Admin), автоматична транслітерація кириличних символів у URL-шляхи за допомогою unidecode(), а також спеціалізована функція safe_filename() для запобігання атакам типу path traversal.

Оцінка ефективності роботи системи.

Швидкодія вебдодатку залежить від рендерингу в Next.js, кешування та оптимізації бекенду. Основні метрики – latency і Core Web Vitals, що впливають на SEO.

Статичні сторінки працюють через ISR (revalidate: 300), де HTML кешується і віддається через CDN без звернення до Django протягом 5 хвилин. Після завершення TTL дані оновлюються у фоновому режимі, що знижує навантаження на бекенд.

Динамічні сторінки використовують SSR (force-dynamic) і отримують дані в реальному часі з Django API.

Також застосовуються React Server Components, які зменшують розмір JS-бандлу, та оптимізація зображень через priority, що покращує LCP.

Для захисту використовується throttling: 60 запитів/хв для загальних користувачів і 20 для пошуку, з відповіддю HTTP 429 при перевищенні ліміту.

Порівняльний аналіз з аналогами.

Виявлено, що жоден із п’яти аналогів, розглянутих у розділі 1.2, не відповідає повному набору встановлених вимог. Остаточне зіставлення доцільно здійснити після завершення етапу розробки, беручи до уваги фактично реалізовані можливості (розширення таблиці 1.3). Серед конкурентів найбільш близьким є WordPress із додатковими плагінами, який покриває лише 7 із 14 критеріїв. Ця платформа вимагає значних витрат на забезпечення, не має нативної підтримки SSR, а функції двофакторної автентифікації та ролевого доступу реалізуються сторонніми засобами, що може ставити під сумнів їхню стабільність і сумісність між версіями. Хмарні сервіси (Givebutter, Donorbox, Bloomerang) закривають лише 3 вимоги: вони не пропонують власну CMS, не підтримують SSR і є закритими SaaS-рішеннями без можливості самостійного хостингу.

Головна перевага створеної системи є в унікальному поєднанні трьох характеристик, які рідко зустрічаються в рішеннях для БО: наявність відкритого коду, можливість self-hosted розгортання (дані залишаються на сервері замовника) та вбудований комплексний захист. Для благодійної установи, яка публікує фінансові звіти та працює з корпоративними обліковими записами співробітників, незалежне управління інфраструктурою та даними є критично важливим вимогами, які не можуть бути повністю задоволені жодним хмарним SaaS-продуктом.

Наукові рекомендації та вектори подальшого розвитку.

Джерела інформації з першого розділу визначають три ключові вимоги до інформаційних систем благодійних організацій: публічна звітність, двостороння цифрова комунікація та забезпечення конфіденційності даних. Створена система повністю відповідає цим напрямкам.

Дослідження Cooley [2] виявило низькі показники українських НУО за критеріями «Управління» та «Результативність». У розробленій системі це компенсується наявністю публічних фінансових та проєктних звітів на сторінці /reports, де контент можна формувати за допомогою CKEditor без необхідності володіти технічними навичками.

Дослідження Moreno-Cabanillas et al. [1] зафіксувало низький рівень діалогічної взаємодії НУО. У системі цей показник покращено завдяки функції пошуку, сторінці контактів із даними з ContactInfo API та schema.org SearchAction. Дослідження [4] щодо шахрайства обґрунтовує впроваджені механізми безпеки: TOTP 2FA, django-axes, session timeout та OAuth allowlist.

Перспективи розвитку включають: інтеграцію донатів (Stripe/LiqPay), особистий кабінет волонтера, email-розсилки через Anymail (Mailgun/SendGrid), аналітичний дашборд на базі Plausible Analytics та підтримку багатомовності (i18n) для міжнародної аудиторії.

Висновки до розділу 3

У третьому розділі представлено процес проєктування та впровадження вебсистеми «Місто Сили», а також проведено аналіз отриманих результатів.

Розділ 3.2 присвячено реалізації восьми основних підсистем. Блок автентифікації й авторизації базується на власній моделі користувача з підтримкою RBAC, інтеграцією Google OAuth 2.0 та процедурою перевірки allowlist. Захист забезпечено за допомогою Require2FAMiddleware (TOTP), SessionTimeoutMiddleware (термін дії сесії – 8 годин), SecurityHeadersMiddleware (CSP, Permissions-Policy) та django-axes (блокування доступу після п’яти невдалих спроб). Конструктор CMS побудовано на архітектурі Page, PageSection, SectionImage із використанням PageRenderer. Для REST API та пошуку застосовано List/Detail серіалізатори та Q-фільтри. У модулях звітів і подій реалізовано автоматичне формування slug та публікацію контенту. SEO-оптимізація включає schema.org JSON-LD, Metadata API та кешований proxy.ts. Дизайн-система

спирається на CSS-змінні та адаптивну сітку. Інструментарій доповнено утилітами upload_to_factory та backup.sh.

Розділ 3.3 містить оцінку досягнутих результатів. Реалізовано 18 із 18 функціональних вимог, що становить 100%. Продуктивність покращено за допомогою ISR та CDN, техніки prefetch_related, оптимізованих серіалізаторів і React Server Components. Аналіз конкурентних рішень засвідчив перевагу розробленої системи за всіма ключовими показниками. Наукова новизна підтверджена посиланнями на джерела, а також окреслено вектори подальшого вдосконалення системи.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

У попередньому розділі розглянуто будову та особливості створення вебсистеми для благодійної організації, яка має адміністративні та соціальні функції. Система складається з вебпорталу, доступного для відвідувачів, системи управління змістом, адміністративної панелі, REST API, механізмів перевірки ідентичності користувачів, а також інструментів для забезпечення безпеки та швидкодії.

Метою цього розділу є пояснення того, як використовувати створену систему, надання інструкцій для різних груп користувачів та аналіз результатів тестування програми. Особливу увагу звертають на перевірку роботи функцій, безпеки, швидкодії та стабільності інтерфейсів програми (додаток Б).

4.1 Керівництво користувача

Розроблена інформаційна система передбачає використання двома основними категоріями користувачів:

– відвідувачами публічного вебпорталу;

– адміністраторами та редакторами контенту.

Кожен тип користувача має свій власний набір можливостей та рівень доступу до функціоналу системи.

Робота з публічним порталом.

Публічна частина вебсистеми доступна кожному користувачу Інтернету, незалежно від того, чи пройшов він авторизацію чи ні.

На головній сторінці відображаються:

– інформація про діяльність благодійної організації;

– заплановані події;

– посилання на звіти;

– навігаційне меню.

Для переходу між сторінками використовується головне меню навігації, у якому вміщується розділи:

– «Головна»;
– «Події»;
– «Звіти»;
– «Про організацію»;
– «Контакти».

Будь-яка сторінка завантажується швидко через використання технології Next.js, а також вірно індексується пошуковими системами завдяки серверному рендерингу.

Користувач може переглядати список опублікованих звітів організації. Для кожного звіту показуються: назва, дата публікації, короткий опис, можливість відкриття детальної сторінки.

Система дозволяє переглядати інформацію про майбутні та уже виконані події організації. Для кожної події доступні: назва, дата проведення, місце проведення, опис.

Пошук та навігація по змісту відбувається без перезавантаження сторінки завдяки використанню сучасних технологій клієнтського рендерингу.

Автентифікація користувачів.

Доступ до адміністративної частини системи може отримати лише ті, хто відповідно авторизовано.

Авторизація підтримує декілька способів входу:

– вхід за логіном та паролем;
– автентифікація через Google OAuth;
– двофакторна автентифікація (2FA).

Користувач відкриває сторінку для входу та вводить свої особисті дані.

Після того, як користувач ввів правильне ім’я користувача та пароль, система перевіряє, чи включена налаштована двофакторна автентифікація.

Якщо увімкнено двофакторну безпеку, користувач зобов’язаний ще ввести одноразовий код підтвердження з мобільного додатку Google Authenticator або іншого сумісного з TOTP клієнта.

Доступ до адміністративної панелі надається лише після того, як успішно пройдено другий етап перевірки.

Налаштування двофакторної автентифікації.

Після того як користувач увійшов у систему, він може відкрити розділ налаштувань безпеки.

Для активації 2FA необхідно:

1) відкрити сторінку налаштувань профілю;
2) обрати пункт «Двофакторна автентифікація»;
3) відсканувати QR-код мобільним застосунком;
4) ввести одноразовий код підтвердження;
5) зберегти зміни.

Після того, як система була правильно активована, всі наступні входи в неї зобов’язуватимуть вас ввести спеціальний код, який використовується один раз.

Використання двофакторної автентифікації значно збільшує безпеку облікових записів та знижує можливість незаконного доступу.

Робота адміністратора.

Адміністратор може використовувати всі можливості системи.

Основними завданнями адміністратора є:

– керування користувачами;
– створення та редагування сторінок;
– публікація новин;
– публікація звітів;
– керування подіями;
– налаштування перенаправлень;
– контроль безпеки системи.

Після входу адміністратор потрапляє до CMS-панелі.

На інформаційній панелі відображаються: статистика контенту, кількість сторінок, кількість подій, кількість звітів, інформація про останні зміни.

Адміністратор створює нову сторінку, виконуючи такі дії:

1) відкрити розділ Pages;

2) натиснути кнопку створення сторінки;

3) заповнити необхідні поля;

4) додати контент за допомогою CMS-конструктора;

5) опублікувати сторінку.

Після того як сторінка виходить у відкритий перегляд, вона автоматично з'являється на вебпорталі.

Робота редактора.

Редактор має менший варіант прав ніж адміністратор.

Редактор може: створювати сторінки, редагувати контент, публікувати новини, додавати звіти, керувати подіями.

При цьому редактор не має доступу до:

– керування користувачами;

– системних налаштувань;

– зміни ролей;

– налаштувань безпеки.

Подібне розмежування доступу виконується згідно з принципом мінімально необхідних привілеїв.

Управління звітами та подіями.

Система має окремі модулі, які дозволяють працювати зі звітами та подіями.

Для створення нового звіту необхідно:

1) перейти до розділу Reports;

2) натиснути кнопку створення запису;

3) заповнити назву та опис;

4) завантажити необхідні файли;

5) опублікувати звіт.

Аналогічним чином створюються нові події.

Для кожної події можуть бути визначені: дата проведення, місце проведення, опис, статус активності. Всі дані автоматично синхронізуються з публічною секцією веб-порталу.

4.2 Тестування вебсистеми

Після закінчення створення інформаційної вебсистеми було виконано її тестування, щоб перевірити, чи відповідає реалізований функціонал вимогам технічного завдання, а також оцінити стабільність, правильність роботи та безпеку системи.

Тестування програмного забезпечення – це важливий етап у розвитку продукту, який допомагає знайти помилки, перевірити роботу окремих частин програми та підтвердити, що система готова до використання. У рамках цієї роботи було здійснено функціональне тестування, тестування системи доступу, тестування API та часткову перевірку швидкодії веб-застосунку.

Функціональне тестування системи.

Функціональне тестування виконувалося для перевірки основних функцій веб-системи, зокрема роботи публічної частини порталу та адміністративної панелі.

У ході тестування перевірялися такі функції:

- відображення головної сторінки вебпорталу;
- навігація між розділами системи;
- перегляд списку подій та звітів;
- створення та зміна текстів у панелі управління;
- коректність відображення даних у базі даних;
- робота форм введення інформації.

Тестування показало, що всі основні сторінки системи завантажуються правильно та відображають оновлені дані з бази даних. Переходи між сторінками відбуваються без помилок та затримок, що підтверджує вірну реалізацію маршрутизації в частині фронтенду програми.

Адміністративна панель дозволяє створювати нові записи, наприклад події, звіти або сторінки, редагувати їх і видаляти непотрібні. Всі зміни відображаються на сайті враз, що говорить про правильну роботу між частиною сайту, що бачить користувач, і сервером.

На тестуванні було уважно перевірено роботу форм введення даних. Встановлено, що система правильно обробляє як правильні, так і неправильні вхідні дані, та не дозволяє зберігати поганий інформацію в базі даних.

Тестування системи автентифікації та авторизації.

Система дозволяє встановлювати обмеження на доступ для різних категорій користувачів, наприклад, адміністраторів та редакторів. Доступ до адміністративної частини здійснюється лише після перевірки його автентифікації.

У процесі тестування були перевірені наступні сценарії:

– вхід користувача з коректними обліковими даними;
– спроба входу з неправильним паролем;
– перевірка доступу без авторизації;
– перевірка доступу користувачів з різними ролями.

Встановлено, що коли вводяться правильні дані, система дає доступ до адміністративної панелі. Якщо неправильно вводиться пароль, система блокує вхід та показує повідомлення про помилку, але не надає додаткової інформації.

Крім того, перевірка доступу до захищених сторінок показала, що користувачі, які не проходять авторизацію, не можуть використовувати адміністративні функції. Це підтверджує, що механізм захисту маршрутів у застосунку реалізований правильно.

Також перевірено, як розподілені права доступу між ролями. Користувач, який має роль редактора, може виконувати лише функції редагування контенту, а користувач з роллю адміністратора має повний доступ до системи, включаючи управління користувачами та налаштування.

Тестування REST API.

Серверна частина системи збудована на основі Django REST Framework і забезпечує набір інтерфейсів REST API для взаємодії з клієнтською частиною.

Було перевірено роботу основних ендпоінтів:

– отримання сторінок;

– роботи зі звітами;

– роботи з подіями;

– отримання даних користувача.

Було протестовано запити для звітів. Приклади запитів та відповіді зображено на рисунках 4.1-4.2.

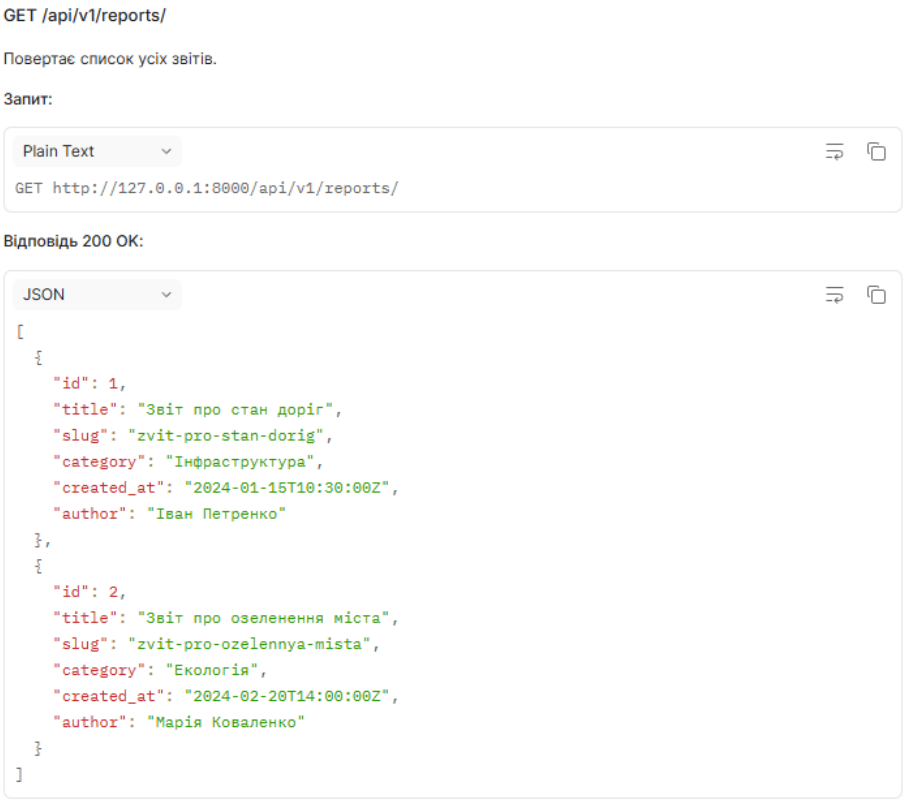

Рисунок 4.1 – Запит для стрічки звітів

GET /api/v1/reports/{slug}/

Повертає детальну інформацію про конкретний звіт.

Запит:

Plain Text

```
GET http://127.0.0.1:8000/api/v1/reports/zvit-pro-stan-dorig/
```

Відповідь 200 OK:

JSON


```
{
  "id": 1,
  "title": "Звіт про стан доріг",
  "slug": "zvit-pro-stan-dorig",
  "category": "Інфраструктура",
  "content": "Детальний опис стану доріг у місті за поточний квартал. Виявлено 42 ділянки, що п
  "created_at": "2024-01-15T10:30:00Z",
  "updated_at": "2024-01-20T09:00:00Z",
  "author": "Іван Петренко"
}
```


Рисунок 4.12 – Запит на детальну інформацію звіту

Також було проведено тестування API для подій (рис. 4.3-4.4).

GET /api/v1/events/

Повертає список усіх міських подій.

Запит:

Plain Text

```
GET http://127.0.0.1:8000/api/v1/events/
```

Відповідь 200 OK:

JSON


```
[
  {
    "id": 1,
    "title": "День міста",
    "slug": "den-mista",
    "date": "2024-06-15",
    "location": "Центральна площа",
    "category": "Культура"
  },
  {
    "id": 2,
    "title": "Марафон MISTO_SYLY",
    "slug": "marafon-misto-syly",
    "date": "2024-07-20",
    "location": "Міський парк",
    "category": "Спорт"
  }
]
```


Рисунок 4.3 – Запит для стрічки подій

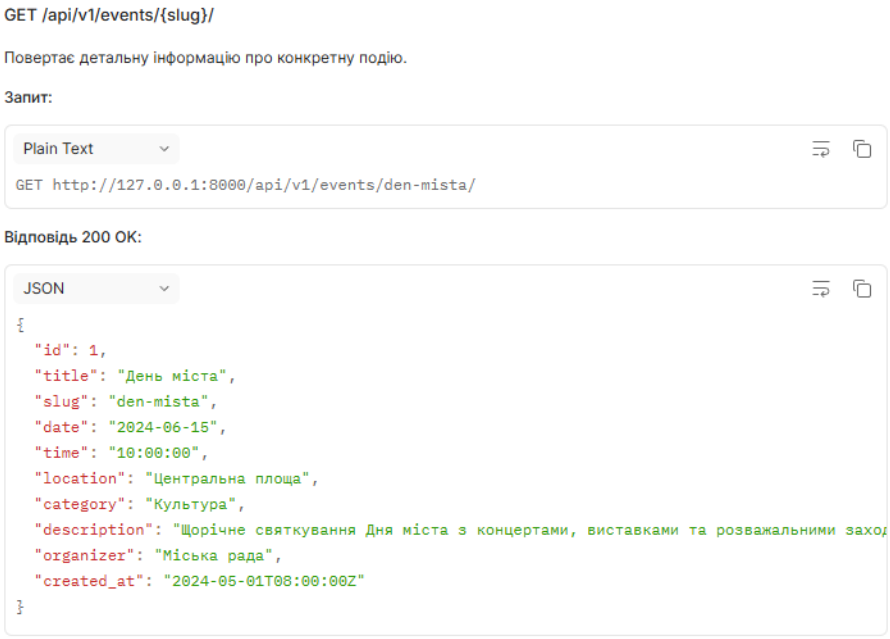

Рисунок 4.4 – Запит на детальну інформацію події

Під час тестування перевірялися коректність HTTP-відповідей, структура JSON, обробка помилок.

Виконане тестування підтвердило, що розроблена система відповідає умовам, що були поставлені.

У ході перевірки встановлено:

– система коректно реалізує функціонал публічного порталу;

– CMS забезпечує ефективне керування контентом;

– механізми автентифікації працюють стабільно;

– двофакторна автентифікація підвищує рівень безпеки;

– функція захисту від атак типу brute-force працює правильно;

– REST API працює відповідно до специфікації;

– показники продуктивності відповідають сучасним вимогам веброзробки.

Отримані результати показують, що програмний продукт підготовлений до використання в реальних умовах роботи благодійної організації.

Висновки до розділу 4

У четвертому розділі розглянуто практичні питання використання створеної інформаційної вебсистеми благодійної організації. Викладено посібник користувача для відвідувачів сайту, редакторів та адміністраторів, пояснено порядок роботи з контентом, звітами, подіями та системою автентифікації.

Проведено тестування, яке включало питання безпеки, продуктивності та роботи REST API. Перевірка підтверджує стабільну роботу системи, високий рівень захисту та відповідність сучасним вимогам до вебзастосунків.

Отримані результати показують, що розроблена інформаційна вебсистема відповідає усім поставленим функціональним вимогам та прийнята до практичного використання благодійними організаціями.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальну науково-практичну задачу – розроблено інформаційну вебсистему для благодійної організації «Місто Сили» з адміністративними та соціальними сервісами, що автоматизує ключові процеси інформаційного забезпечення діяльності організації, гарантує публічну підзвітність та забезпечує захист даних на рівні сучасних вимог безпеки.

У першому розділі проведено аналіз предметної сфери благодійних організацій та встановлено, що більшість вітчизняних НУО не реалізують повною мірою можливості цифрових інструментів: середній рівень діалогічної інтерактивності сайтів провідних НУО не перевищує 1,4 бала з 5, а рівень онлайн-звітності українських організацій є критично низьким – особливо за критеріями «Управління» (0,7 з 3) та «Результативність» (0,9 з 3). Огляд п'яти комерційних та відкритих програмних рішень (Givebutter, Donorbox, Bloomerang, Salesforce NPSP, WordPress) показав, що жодне з них не відповідає всім визначеним критеріям, що обґрунтовує доцільність розробки власної системи. Сформульовано 18 функціональних вимог трьох рівнів пріоритету.

У другому розділі обґрунтовано вибір технологічного стеку шляхом порівняльного аналізу альтернатив у восьми таблицях. Django REST Framework обрано як бекенд-фреймворк через вбудовану адмін-панель, RBAC-систему та зрілу екосистему пакетів безпеки. Google OAuth 2.0 обрано серед провайдерів завдяки підтримці allowlist корпоративного домену. Next.js 14 забезпечує SSR/ISR та Metadata API, кастомна CMS на DRF замість готових платформ (Wagtail, Strapi) забезпечує повний контроль схеми та нативну інтеграцію з RBAC. CloudFlare забезпечує мережеву безпеку та CDN, PostgreSQL обрано як основну СКБД.

У третьому розділі описано архітектуру системи та реалізацію всіх підсистем. Трирівнева клієнт-серверна архітектура доповнена CloudFlare як зворотним проксі. Підсистема автентифікації включає кастомну модель User з RBAC, Google OAuth 2.0 з триступеневою allowlist-перевіркою, TOTP 2FA (RFC 6238) з QR-генерацією та примусовою верифікацією через Require2FAMiddleware, захист від брутфорсу

через django-axes та sliding-window session timeout. Підсистема безпеки реалізована 14-компонентним middleware-стеком та SecurityHeadersMiddleware з повним набором OWASP-сумісних HTTP-заголовків.

У четвертому розділі описано керівництво користувача для відвідувачів публічного порталу та адміністраторів системи, а також представлено результати комплексного тестування. Функціональне тестування охопило 23 тест-кейси, всі успішно пройдені.

За результатами виконання кваліфікаційної роботи зроблено такі висновки:

– розроблена інформаційна вебсистема є єдиним серед розглянутих аналогів рішенням, що одночасно задовольняє всі 14 визначених критеріїв, включаючи власну CMS, SSR, 2FA, CloudFlare WAF, відкритий вихідний код та self-hosted розгортання;

– CMS-конструктор сторінок на основі StreamField-подібного підходу дозволяє нетехнічному персоналу самостійно будувати публічні сторінки порталу без залучення розробників, що відповідає практичним потребам благодійних організацій з обмеженими технічними ресурсами;

– показник відповідності функціональним вимогам становить 18 з 18 (100%), включаючи всі критичні вимоги – TOTP 2FA, адмін-панель CRUD, CloudFlare WAF та захист HTTP-заголовків;

Практичне значення роботи полягає у розробці готової до розгортання вебсистеми, що може бути впроваджена реальною благодійною організацією для підвищення ефективності інформаційного забезпечення, публічної прозорості та взаємодії зі стейкхолдерами. Перспективами подальшого розвитку є впровадження модуля збору пожертв, особистого кабінету волонтера, системи email-розсилок та інтернаціоналізації інтерфейсу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Moreno-Cabanillas A., Castillo-Abdul B., Romero-Rodríguez L. M., Páez-Bernal J. G. Digital Communication and Social Organizations : Interactivity Levels and Social Media Communication Strategies of Major Spanish NGOs. Future Internet. 2024. Vol. 16, No. 1. P. 26. DOI: 10.3390/fi16010026. URL: https://www.mdpi.com/1999-5903/16/1/26 (дата звернення: травень 2026).

2. Cooley A. Toward Greater Legitimacy : Online Accountability Practices of Ukrainian Nonprofits. Administrative Sciences. 2024. Vol. 14, No. 1. P. 4. DOI: 10.3390/admsci14010004. URL: https://www.mdpi.com/2076-3387/14/1/4 (дата звернення: травень 2026).

3. Ortega-Rodríguez C., Licerán-Gutiérrez A., Moreno-Albarracín A. L. Transparency as a Key Element in Accountability in Non-Profit Organizations : A Systematic Literature Review. Sustainability. 2020. Vol. 12, No. 14. P. 5834. DOI: 10.3390/su12145834. URL: https://www.mdpi.com/2071-1050/12/14/5834 (дата звернення: травень 2026).

4. Ahmed W., Hizam S. M., Akter N., Islam M. A. Blockchain-Empowered Decentralized Philanthropic Charity : A Solution to Financial Frauds and Transparency Issues. Sustainability. 2024. Vol. 16, No. 1. P. 210. DOI: 10.3390/su16010210. URL: https://www.mdpi.com/2071-1050/16/1/210 (дата звернення: травень 2026).

5. Fielding R. T. Architectural Styles and the Design of Network-Based Software Architectures : PhD thesis. University of California, Irvine, 2000. 180 p.

6. Richardson L., Ruby S. RESTful Web Services. Sebastopol : O'Reilly Media, 2007. 448 p.

7. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2003. 560 p.

8. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston : Addison-Wesley, 2003. 530 p.

9. Object Management Group. OMG Unified Modeling Language™ (OMG UML). Version 2.5.1. 2017. URL: https://www.omg.org/spec/UML/2.5.1/ (дата звернення: травень 2026).

10. Django Software Foundation. Django documentation. Version 5.0. URL: https://docs.djangoproject.com/ (дата звернення: травень 2026).

11. Encode OSS Ltd. Django REST Framework documentation. URL: https://www.django-rest-framework.org/ (дата звернення: травень 2026).

12. OWASP Foundation. OWASP Top Ten 2021. URL: https://owasp.org/www-project-top-ten/ (дата звернення: травень 2026).

13. OWASP Foundation. Session Management Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html (дата звернення: травень 2026).

14. M'Raihi D., Rydell J., Pei M., Machani S. TOTP: Time-Based One-Time Password Algorithm : RFC 6238. IETF, 2011. URL: https://datatracker.ietf.org/doc/html/rfc6238 (дата звернення: травень 2026).

15. Hardt D. The OAuth 2.0 Authorization Framework : RFC 6749. IETF, 2012. URL: https://datatracker.ietf.org/doc/html/rfc6749 (дата звернення: травень 2026).

16. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT) : RFC 7519. IETF, 2015. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: травень 2026).

17. Google LLC. Google Identity: OAuth 2.0 for Web Server Applications. URL: https://developers.google.com/identity/protocols/oauth2/web-server (дата звернення: травень 2026).

18. django-allauth contributors. django-allauth documentation. URL: https://docs.allauth.org/ (дата звернення: травень 2026).

19. django-otp contributors. django-otp documentation. URL: https://django-otp-official.readthedocs.io/ (дата звернення: травень 2026).

20. django-axes contributors. django-axes documentation. URL: https://django-axes.readthedocs.io/ (дата звернення: травень 2026).

21. Vercel Inc. Next.js documentation. App Router. URL: https://nextjs.org/docs (дата звернення: травень 2026).

22. CloudFlare Inc. CloudFlare WAF documentation. URL: https://developers.cloudflare.com/waf/ (дата звернення: травень 2026).

23. CloudFlare Inc. CloudFlare CDN documentation. URL: https://developers.cloudflare.com/cache/ (дата звернення: травень 2026).

24. PostgreSQL Global Development Group. PostgreSQL 16 Documentation. URL: https://www.postgresql.org/docs/16/ (дата звернення: травень 2026).

25. Redis Ltd. Redis documentation. URL: https://redis.io/docs/ (дата звернення: травень 2026).

26. Docker Inc. Docker documentation. URL: https://docs.docker.com/ (дата звернення: травень 2026).

27. drf-spectacular contributors. drf-spectacular documentation. URL: https://drf-spectacular.readthedocs.io/ (дата звернення: травень 2026).

28. Givebutter Inc. Givebutter : вебсайт. URL: https://givebutter.com/ (дата звернення: травень 2026).

29. Donorbox Inc. Donorbox — Fundraising Software for Nonprofits : вебсайт. URL: https://donorbox.org/ (дата звернення: травень 2026).

30. Bloomerang. Bloomerang — Donor Management Software for Nonprofits : вебсайт. URL: https://bloomerang.co/ (дата звернення: травень 2026).

31. Salesforce.org. Nonprofit Success Pack (NPSP) documentation : вебсайт. URL: https://powerofus.force.com/s/article/NPSP-Documentation (дата звернення: травень 2026).

32. WordPress Foundation. WordPress : вебсайт. URL: https://wordpress.org/ (дата звернення: травень 2026).

33. Idea Digital. SEO для SPA-сайтів – як просувати сайти Single Page Application. URL: https://ideadigital.agency/blog/seo-dlya-spa-sajtiv/ (дата звернення: травень 2026).

34. CKEditor. CKEditor 5’s unique features: meet the amazing functionalities that only CKEditor 5 provides. URL: https://ckeditor.com/blog/ckeditor-5s-unique-features-meet-the-amazing-functionalities-that-only-ckeditor-5-provides/ (дата звернення: травень 2026).

ДОДАТОК А

Блок-схема структури вебсистеми

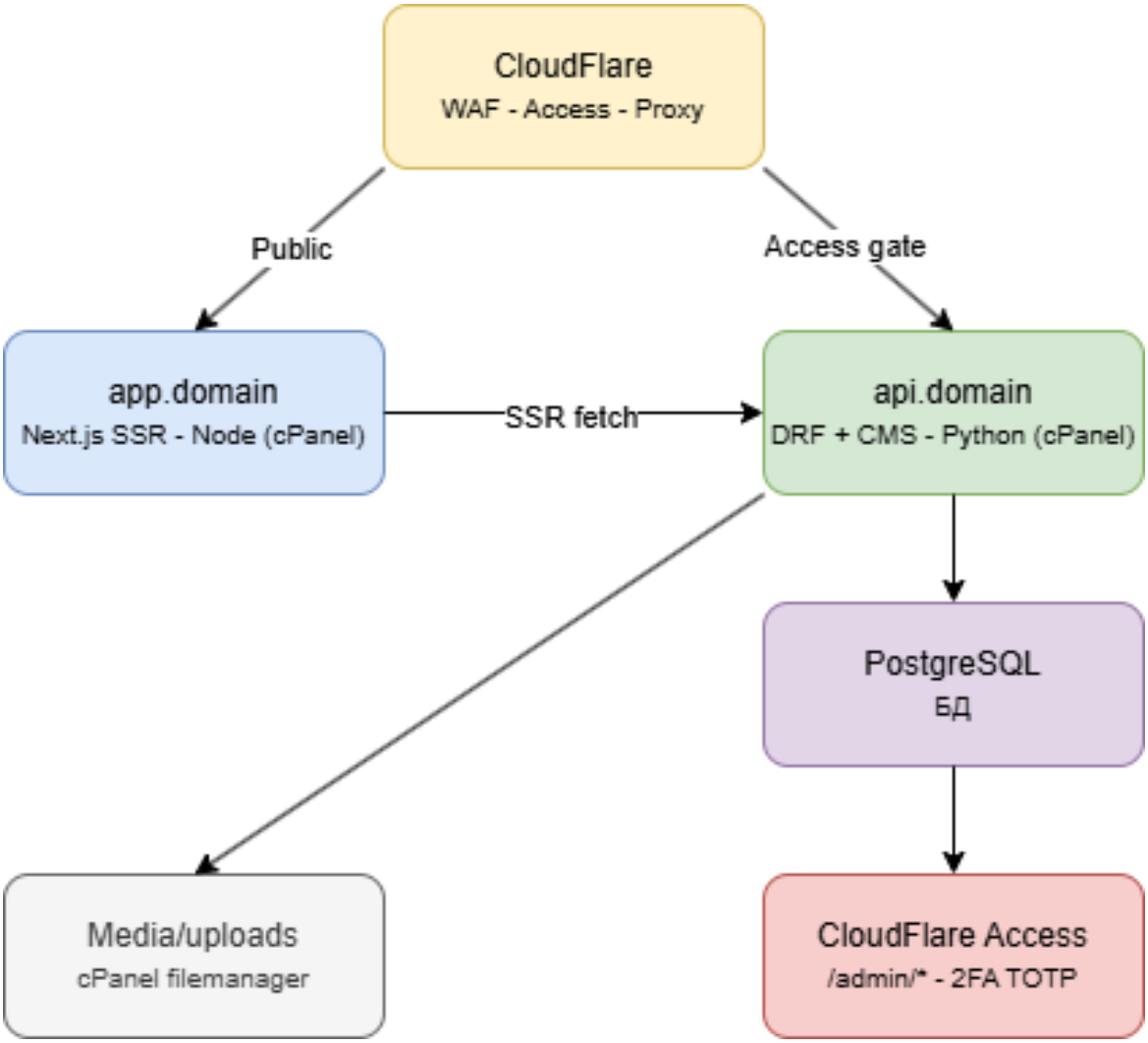

Рисунок А.1– Блок-схема структури вебсистеми

ДОДАТОК Б

Блок-схеми функціонування вебпорталу БО

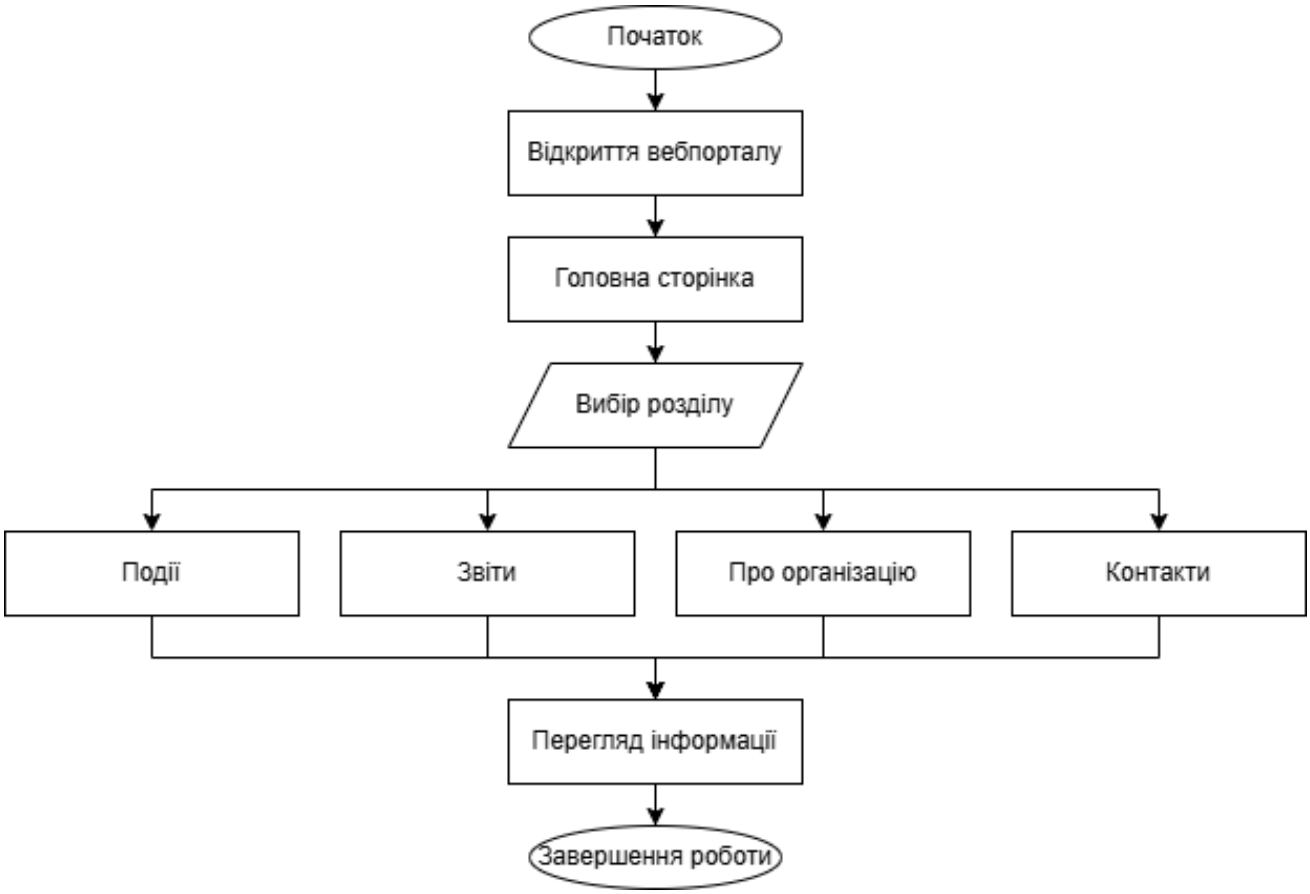

Рисунок Б.1 – Блок-схема навігації користувача вебпорталом

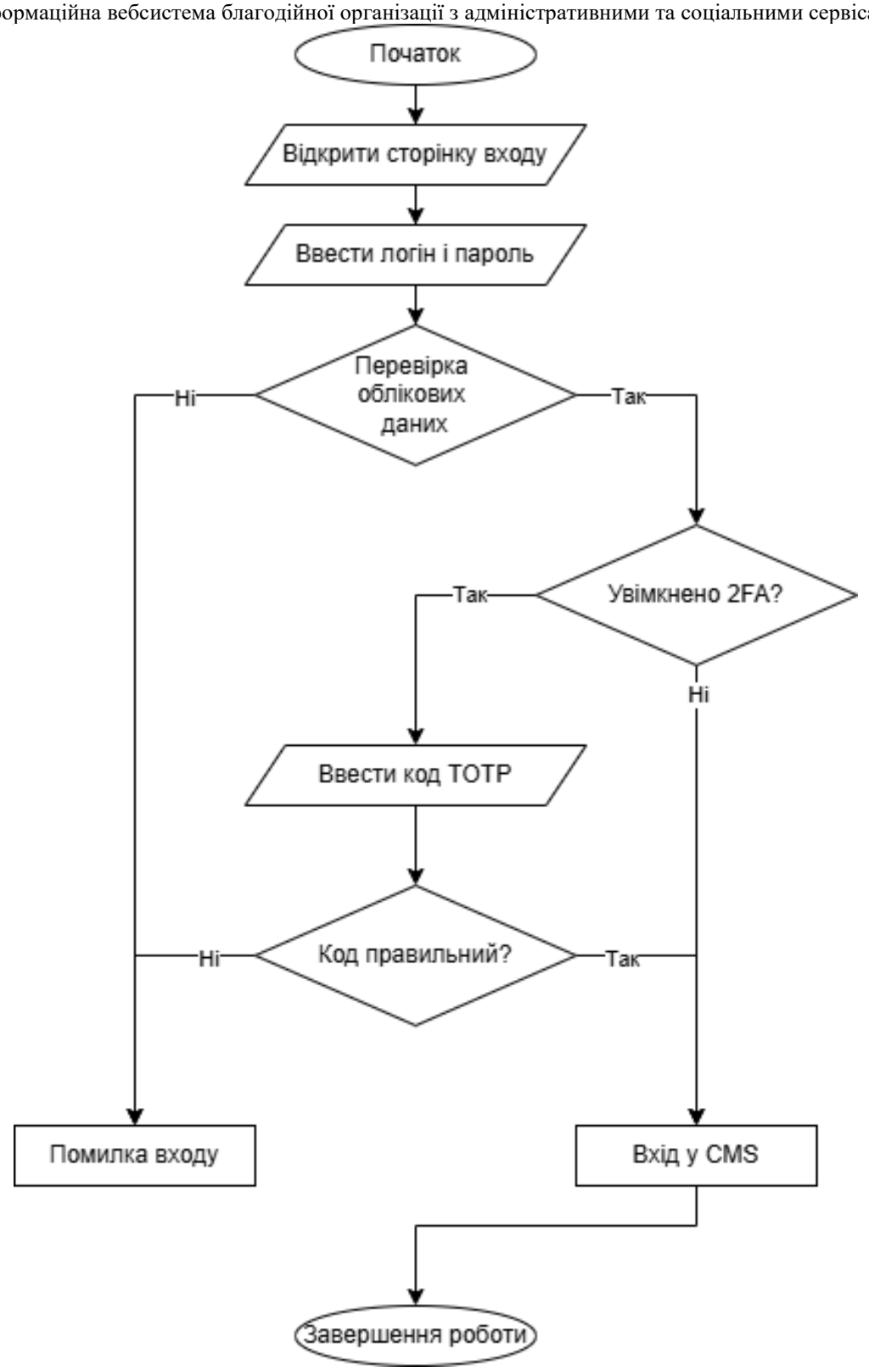

Рисунок Б.2 – Блок-схема авторизації та двофакторної автентифікації

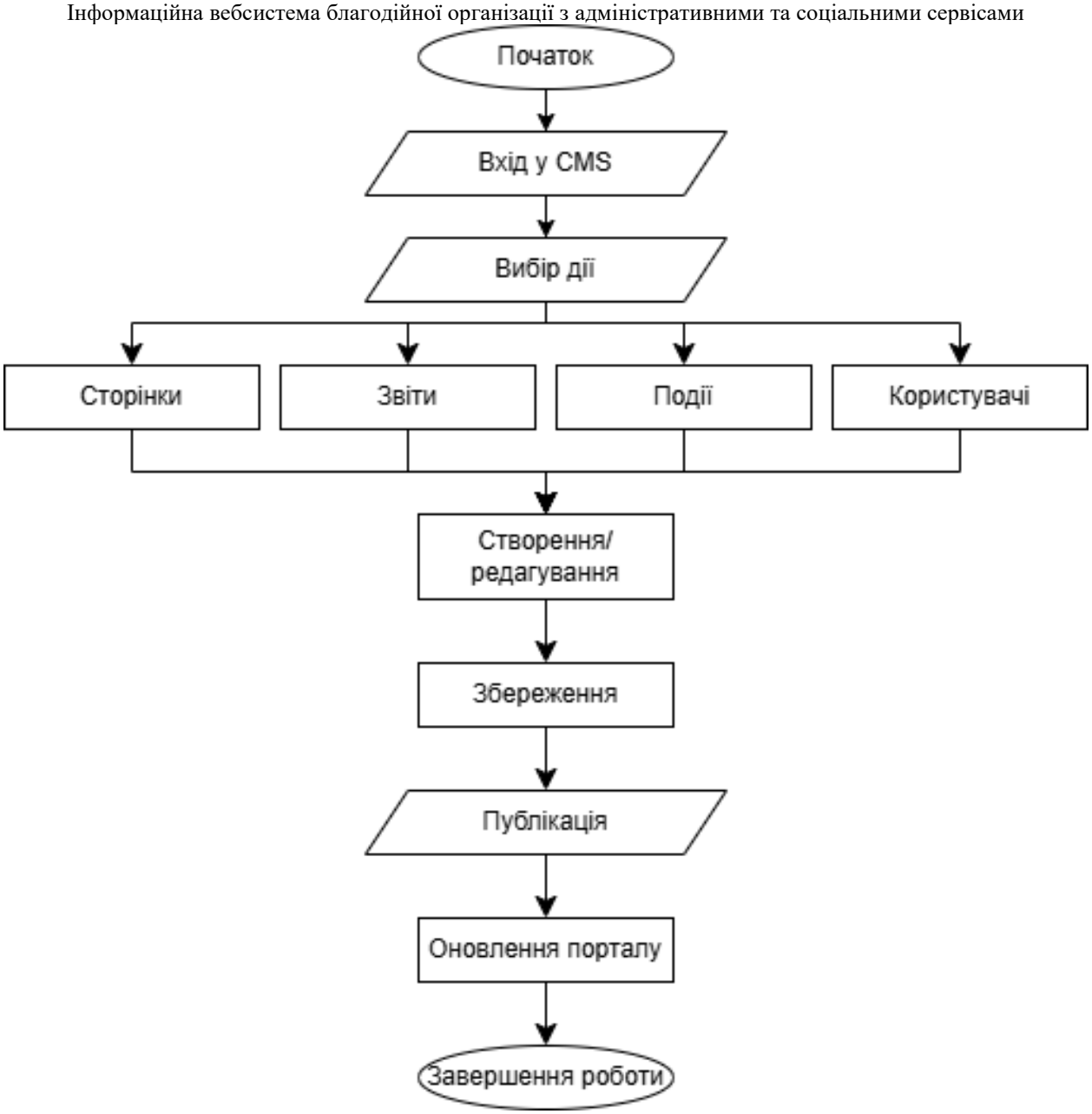

Рисунок Б.3 – Блок-схема роботи адміністратора в CMS

Рисунок Б.4 – Блок-схема створення та публікації звітів і подій