“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
В. о. Завідувач кафедри
_____ Михайло НОВОТАРСЬКИЙ
(підпис)

“__” _____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

на тему: Система моніторингу ресурсів для громадських організацій та волонтерів

Виконав : студент __4__ курсу, групи _ІМ-11_
(шифр групи)

Семиволос Євгеній Артемович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник _доцент, к.т.н. Павлов В. Г._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) асистент Нікольський С.С._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент __професор Корнієнко Б. Я_____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)
Освітньо-професійна програма
“Інженерія програмного забезпечення комп’ютерних систем”
спеціальності 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Михайло НОВОТАРСЬКИЙ
(підпис)
“__” _____2025 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента
Семиволоса Євгенія Артемовича

1. Тема проєкту *Система моніторингу ресурсів для громадських організацій та волонтерів*
керівник проєкту доцент, к.т.н. Павлов В. Г.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 23 травня 2025 року No1705-с

2. Термін здачі студентом закінченого проєкту *02.06.2025.*

3. Вихідні дані до проєкту *див. технічне завдання.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються): огляд існуючих програмних продуктів для моніторингу ресурсів, огляд технологій розробки системи моніторингу ресурсів, Деталі розробки системи, дослідження та аналіз розробленої системи.

5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень) структурна схема системи (структурна схема), ER діаграма бази даних (функціональна схема), Архітектурна структура застосунку (принципова схема).

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Нікольський С. С.		

7. Дата видачі завдання «8 вересня» 2024 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	02.07.2024-31.01.2025	
2.	*Вивчення та аналіз завдання*	01.02.2025-20.04.2025	
3.	*Розробка архітектури та загальної структури системи*	21.04.2025-27.04.2025	
4.	*Розробка структур окремих підсистем*	28.04.2025-04.05.2025	
5.	*Програмна реалізація системи*	05.05.2025-11.05.2025	
6.	*Оформлення пояснювальної записки*	12.05.2025-18.05.2025	
7.	*Захист програмного продукту*	05.06.2025	
8.	*Передзахист*	06.06.2025	
9.	*Захист*	19.06.2025	

Студент-дипломник _____ СЕМИВОЛОС Євгеній
(підпис)

Керівник проєкту _____ ПАВЛОВ Валерій
(підпис)

АНОТАЦІЯ

У даній роботі розглянуто методи розробки веб-системи для моніторингу та координації волонтерської діяльності. Мною було проведено дослідження сучасних підходів до створення інтерактивних веб-додатків із використанням React для клієнтської частини та Django для серверної частини. За результатами дослідження реалізовано веб-застосунок, що забезпечує повний цикл управління волонтерською діяльністю, включаючи реєстрацію користувачів, керування особистими даними, координацію проектів та моніторинг активності. Розроблена система використовує комбінацію технологій React та Django, що забезпечує надійну архітектуру, поєднуючи переваги сучасного JavaScript-фреймворку для інтерфейсу користувача та Python-фреймворку для серверної логіки.

Ключові слова: веб-розробка, React, Django, Python, JavaScript, система моніторингу, волонтерська діяльність, автентифікація, управління користувачами.

ANNOTATION

This Bachelor's project examines methods of developing a web system for monitoring and coordinating volunteer activities. My study includes modern approaches to creating interactive web applications using React for the frontend and Django for the backend. As a result of the research, a web application was developed that provides a complete cycle of volunteer activity management, including user registration, personal data management, project coordination, and activity monitoring. The developed system uses a combination of React and Django technologies, providing a reliable architecture that combines the advantages of a modern JavaScript framework for the user interface and a Python framework for server-side logic.

Key words: web development, React, Django, Python, JavaScript, monitoring system, volunteer activities, authentication, user management.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпляра	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Система моніторингу	3		
			ресурсів для громадських			
			організацій та волонтерів			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Система моніторингу	71		
			ресурсів для громадських			
			організацій та волонтерів			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Система моніторингу	1		
			ресурсів для громадських			
			організацій та волонтерів			
			Структурна схема системи			
	А4	***ІАЛЦ.4672008.005 Д2***	Система моніторингу	1		
			ресурсів для громадських			
			організацій та волонтерів			
			ER діаграма бази даних			
			(функціональна схема)			
	А4	***ІАЛЦ.467200.006 Д3***	Система моніторингу	1		
			ресурсів для громадських			
			організацій та волонтерів			
			Архітектурна структура застосунку			
			(принципова схема)			
	А4	***ІАЛЦ.467200.007 Д4***	Система моніторингу	64		
			ресурсів для громадських			
			організацій та волонтерів			
			Текст програмного коду			

					ІАЛЦ.467200.001 ОА			
Зм.	Лист	№ докум.	Підп	Дата				
Розробив		Семиволос Є. А			*Система моніторингу ресурсів для громадських організацій та волонтерів* Опис альбому	Літ.	Аркуш	Аркушів
Перевірив		Павлов В. Г.					1	1
Реценз.		Корнієнко Б. Я.				*НТУУ "КПІ" ФІОТ, ІМ-11*		
Н. Контр.		Нікольський С. С.						
Затв.		Новотарський М. А.						

ТЕХНІЧНЕ ЗАВДАННЯ

ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Система моніторингу ресурсів для громадських організацій та волонтерів*»

Київ – 2025

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку системи моніторингу ресурсів для громадських організацій (ГО) та волонтерів, а також на подальшу підтримку та вдосконалення системи.

Областю застосування цієї системи є координація гуманітарної допомоги, такої як надання лік, їжі, одягу та обладнання; управління волонтерськими ресурсами; звітність для донорів і партнерів;

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр інженерії програмного забезпечення», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка системи моніторингу ресурсів для ГО та волонтерів, що дозволить ефективно відстежувати надходження та витрати ресурсів та оптимізувати їх розподіл між організаціями та волонтерами.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проекту є офіційні документації, публікації та статті в мережі Інтернет на дану тему, науково-технічна література.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Надати простий і інтуїтивно-зрозумілий інтерфейс системи.

- Надати можливість моніторити ресурси, а саме додавання, редагування ресурсів
- Надати такий функціонал: реєстрація, розподіл завдань, історія активності користувачів-волонтерів
- Надати вичерпну та зрозумілу документацію для розробленого продукту.

5.2. Вимоги до програмного забезпечення

- ОС Windows, Mac або Linux.

5.3. Вимоги до апаратної частини

- ЦП не менше ніж 2 ядра по 2.0 GHz.
- ROM не менше ніж 10 ГБ.
- RAM не менше ніж 4 ГБ.
- Швидкість мережі від 1 Mbps.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	02.07.2024-31.01.2025
Вивчення та аналіз завдання	01.02.2025-20.04.2025
Розробка архітектури та загальної структури системи	21.04.2025-27.04.2025
Розробка структур окремих частин системи	28.04.2025-04.05.2025
Програмна реалізація системи	05.05.2025-11.05.2025
Виправлення помилок	12.05.2025-18.05.2025
Оформлення пояснювальної записки	05.06.2025

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: *Система моніторингу ресурсів для громадських організацій та волонтерів*»

Київ – 2025

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ГО	Громадська організація
НКО	Не комерційна організація
SPA	Single Page Application (односторінковий додаток)
MPA	Multi Page Application (багатосторінковий додаток)
PWA	Progressive Web Application (гібридний додаток)
MTV	Model-Template-View
I/O	Input-Output (ввід-вивід)
CPU	central processing unit (процесор)
API	Application Programming Interface (набір інструментів для розробника)
CRUD	Create, Read, Update, Delete (основні операції управління даними)

ВСТУП

На даний час Україна переживає найважчий період своєї історії — період повномасштабної війни, спричиненої російською агресією. Цей час дуже яскраво продемонстрував, що українці мають велику силу єдності та готові допомагати один одному. Ми всі захищаємо свою країну, але хто як може: хтось з автоматом в руках, хтось донатить, а хтось допомагає постраждалим.

Безперечно, волонтери були і є дуже важливою частиною української єдності. Громадські організації, благодійні фонди та звичайні люди щодня організовують збори, координують доставку гуманітарної допомоги, розподіляють ліки, їжу та інші необхідні речі. І, як і інших сферах життя, у цьому процесі часто виникають труднощі. Буває, що дані про наявні ресурси стають неактуальними через відсутність системи моніторингу.

Метою та призначенням даної роботи є розробка системи моніторингу ресурсів для ГО та волонтерів, що дозволить ефективно відстежувати надходження та витрати ресурсів та оптимізувати їх розподіл між організаціями та волонтерами.

Щоб досягнути поставленої мети, треба проаналізувати потреби ГО та волонтерів у сфері управління ресурсами для формулювання вимог до функціоналу системи. Після цього треба розробити онлайн-базу даних з можливостями фільтрації, пошуку та оновлення інформації в режимі реального часу. Далі треба реалізувати інтеграцію з месенджером (наприклад, Telegram) для автоматизації сповіщень про зміни в наявності ресурсів, створити інтуїтивно зрозумілий інтерфейс з різними ролями користувачів (адміністратор, волонтер) та відповідними правами доступу до даних. Не менш важливим є забезпечення захисту інформації (системи автентифікації, шифрування даних).

Сподіваюся, розробка допоможе оптимізувати роботу волонтерів та ГО, зробивши процес допомоги ще ефективнішим у цей критичний для України час.

РОЗДІЛ 1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1 Визначення предметної області

Громадська організація (ГО) — це добровільне об'єднання громадян, створене для реалізації суспільно корисних цілей у некомерційній сфері, а волонтери — це активні громадяни, які безоплатно долучаються до вирішення соціальних проблем, надаючи свої час, знання та ресурси.

Згідно з офіційним визначенням Державної служби України з питань праці, волонтерська діяльність є "безоплатною, добровільною, неприбутковою діяльністю, спрямованою на досягнення суспільно корисних цілей" [1].

1.1.1 Специфіка роботи ГО та волонтерів

Один із гвинтиків волонтерства пов'язан з необхідністю постійного моніторингу ресурсів. Через обмежені фінансові можливості та залежність від зовнішнього фінансування, ГО змушені ретельно відстежувати кожен напрямок витрат – від закупівлі гуманітарної допомоги до утримання офісу. Водночас динамічність волонтерської діяльності, де кількість активних учасників може різко змінюватись, вимагає оперативного контролю людських ресурсів – їх наявності, кваліфікації та завантаженості.

Ефективний моніторинг стає критично важливим для успішної діяльності. Це дозволяє уникнути дублювання зусиль, раціонально розподіляти обмежені матеріальні запаси, а також забезпечувати прозорість перед донорами. Особливо це актуально в умовах надзвичайних ситуацій, коли ресурси обмежені, а потреби – масштабні.

1.1.2 Типи ресурсів, які потребують моніторингу

ГО та волонтерські ініціативи потребують точної фіксації фінансових надходжень - кожен донат, грант чи спонсорський внесок вимагає ретельного обліку для цільового використання коштів. Організації ризикують зіткнутися з дефіцитом бюджету без чіткого відстеження витрат на оренду приміщень, комунальні послуги чи заробітні плати.

Такі матеріальні активи як гуманітарна допомога чи технічне обладнання, потребують постійного контролю термінів придатності та технічного стану. Наприклад, ліки з певним терміном дії чи комп'ютерна техніка для роботи волонтерських центрів повинні бути точно зафіксовані в системі обліку.

Кількість та доступність волонтерів - ще один критичний параметр, який вимагає постійного моніторингу. Не можливо нехтувати такими речами як кваліфікація, навантаження та емоційний стан кожного активіста. Це допомагає запобігати вигоранню та ефективно розподіляти завдання.

Згідно з дослідженням Центру "Громадський простір" (2023), більшість українських громадських організацій відзначають значні труднощі через відсутність належної системи обліку ресурсів, що безпосередньо впливає на ефективність їхньої роботи [2].

1.2 Різновиди систем

1.2.1 CRM-система

CRM — це програмне забезпечення, яке дозволяє бізнесам спростити взаємодію з клієнтами, автоматизувати продажі, маркетинг та підтримку. Такі системи допомагають компаніям збирати, аналізувати та використовувати дані про клієнтів для покращення ефективності бізнесу [3].

CRM має в собі такі ключові елементи:

- Зберігання інформації про клієнтів, їх сегментація за критеріями;
- Автоматизація продажів;
- Ведення клієнтів від першого контакту до закриття угоди;
- Налаштування нагадувань про наступні кроки (дзвінки, зустрічі);
- Розсилка електронних листів, SMS, повідомлень у соцмережах;
- Аналітика та звіти.

Серед переваг CRM можна виділити персоналізацію взаємодій, через що підвищується лояльність клієнтів; зменшення часу на рутинні завдання; зростання конверсії.

1.2.2 ERP-система

ERP — це комплексні програмні рішення, які інтегрують ключові бізнес-процеси компанії в єдину систему. Вони забезпечують централізоване управління фінансами, логістикою, виробництвом, та іншими операціями через спільну базу даних. Мета ERP — підвищити ефективність, усунути розрізненість інформації та оптимізувати використання ресурсів [4].

ERP-системи справляється з такими задачами:

- Фінанси: Облік доходів/витрат, бюджетування;
- Контроль надходження/відвантаження товарів, прогнозування попиту, управління складами;
- Планування виробничих потужностей, контроль якості, облік витрат;
- Робота з персоналом, рекрутинг, розрахунок зарплат;
- Генерація звітів у реальному часі, аналіз ефективності підрозділів.

Перевагами ERP-систем є скорочення часу на ручне введення даних, прозорість процесів перед всіма працівниками, скорочення витрат та масштабованість.

ERP-системи дуже добре підходять великим корпораціям, але не менш популярними вони є серед малого та середнього бізнесу.

1.2.3 Веб-додаток

Веб-додаток — це програмне забезпечення, доступне через інтернет за допомогою браузера. Така система є дуже популярною в інтернеті, і її використовують чи не кожен.

Складовими веб-додатку є:

- Клієнтська частина: Виконується в браузері користувача
- Серверна частина: Обробляє запити, взаємодіє з базами даних.

Веб-додатки можна розбити на такі класи:

- Single Page Application (SPA) – додаток, який складається з однієї сторінки, контент якої оновлюється динамічно. Доволі висока швидкість роботи, зручно для користувача;
- Multi Page Application (MPA) – додаток, який включає в себе декілька сторінок одночасно, які повністю перезавантажуються після дій користувача. Легше робити SEO-оптимізацію, ідеальний вибір для електронної комерції;
- Progressive Web Application (PWA) – гібрид веб- та мобільного додатку. Такий додаток може частково працювати офлайн. Через такий підхід є доступ через браузер або іконку на робочому столі.

Перевагами веб-додатків є незалежність від додаткових програм, легкість оновлення (зміни вносяться на сервері) та масштабованість, а саме можливість обслуговувати великі обсяги користувачів.

1.2.4 REST API

REST API, або передача стану представлення, є архітектурним стилем для створення веб-сервісів, який базується на принципах простоти, гнучкості та масштабованості. Його концепцію розробив Рой Філдінг у 2000 році як частину розробки протоколу HTTP 1.1. REST став фундаментом сучасного Інтернету, визначаючи правила взаємодії між клієнтами та серверами через стандартизовані HTTP-методи.

Основна ідея полягає в тому, що будь-яка сутність, чи то користувач, чи то товар, представлена як ресурс з унікальним ідентифікатором URI. Доступ до цих ресурсів здійснюється через стандартні HTTP-методи: GET для отримання даних, POST для створення нового ресурсу, PUT/PATCH для оновлення та DELETE для видалення. Важливою особливістю REST є відсутність збереження стану (stateless), коли кожен запит містить повну інформацію для його обробки, а сервер не зберігає контекст між запитами.

Архітектура REST API передбачає кешування відповідей для підвищення продуктивності та розшарування системи на різні рівні, що забезпечує горизонтальну масштабованість. Це дозволяє створювати прості, гнучкі та надійні рішення, які відповідають сучасним вимогам до веб-додатків. Переваги REST API включають універсальність - сумісність з будь-якою мовою програмування або платформою, простоту інтеграції завдяки використанню стандартних HTTP-методів та форматів JSON/XML, а також здатність ефективно масштабуватися для обслуговування мільйонів запитів.

Сьогодні REST API активно використовується провідними технологічними компаніями у різних сферах. Соціальні мережі, такі як Facebook та Instagram, використовують його для роботи з даними користувачів та публікації контенту.

1.3 Аналіз існуючих розробок

Для розробки системи моніторингу ресурсів необхідно мати чітке представлення його кінцевого вигляду та функціоналу. Для виконання цих умов, потрібно оглянути готові рішення, які є дотичними до системи моніторингу ресурсів для ГО та волонтерів, з метою ближчого розуміння кінцевої мети.

Надалі буде розглянуто готові волонтерські системи моніторингу.

1.3.1 CiviCRM

CiviCRM — це безкоштовна CRM-система з відкритим кодом, спеціально розроблена для некомерційних організацій, громадських ініціатив та благодійних фондів. Вона дозволяє керувати волонтерами, організовувати заходи, збирати пожертви та автоматизувати комунікації [5].

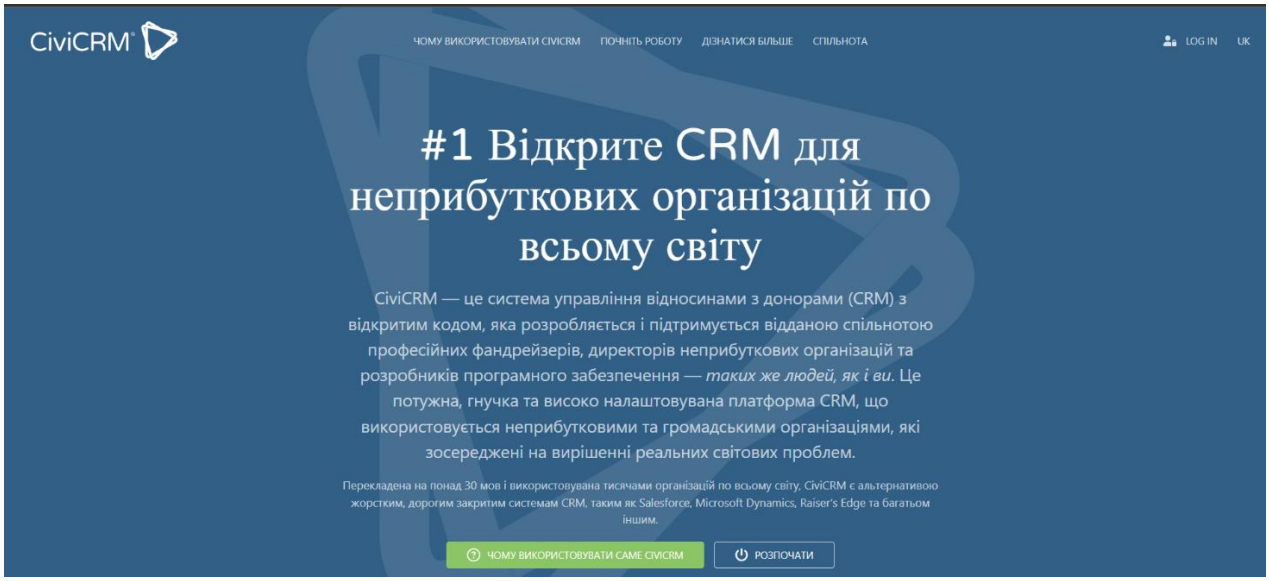


Рисунок 1.1 – Головна сторінка CRM-системи CiviCRM

CiviCRM може служити ефективним інструментом для моніторингу ресурсів завдяки таким речам:

- Модуль CiviContribute дозволяє відстежувати пожертви, генерувати звіти про витрати та доходи, а також інтегруватися з платіжними системами для онлайн-зборів;
- Модуль CiviMember фіксує інформацію про волонтерів, їх активність, навантаження та членство в організації;
- Через кастомні поля можна створювати записи для гуманітарної допомоги, обладнання або інших матеріалів, відстежуючи їх стан і терміни придатності;
- Модуль CiviEvent автоматизує реєстрацію учасників, управління квитками та аналіз ефективності заходів.

Таблиця 1.1 – Аналіз CiviCRM

Критерій	**CiviCRM**
Управління ресурсами	Основним фокусом є управління донорами, подіями, членством. Не підтримує відстеження фізичних ресурсів
Координація волонтерів	Обмежена підтримка через додатки або кастомізацію.
Інтеграція з месенджерами	Відсутня вбудована підтримка. Потрібна інтеграція через API або сторонні сервіси.
Ролі користувачів	Гнучкі ролі (адміністратор, менеджер, донор).
Вартість	Безкоштовна (open-source), але витрати на хостинг та кастомізацію.
Кастомізація	Висока за рахунок модулів та розширень.
Захист даних	Залежить від хостингу. Базове шифрування.

Серед переваг CiviCRM користувачі виділяють безкоштовність та відкритий код, спільнота та підтримка та гнучкість.

Якщо використовувати CiviCRM як системи моніторингу ресурсів, то виникають наступні проблеми:

- Для впровадження потрібні технічні навички;
- Витрати на хостинг, розробників або консультантів;
- Дефіцит автоматизації для складних процесів. CiviCRM може вимагати додаткових модулів для глибокої аналітики.

1.3.2 Better Impact

Better Impact — це хмарна платформа для управління волонтерами, донорами та клієнтами, розроблена для некомерційних організацій, муніципалітетів та благодійних фондів. Вона включає чотири основні модулі: Volunteer Impact (волонтери), Donor Impact (донори), Member Impact (члени організацій) та Client Impact (клієнти). Система дозволяє автоматизувати реєстрацію, планування завдань, комунікацію та звітність, спрощуючи взаємодію з усіма стейкхолдерами [6].

Рисунок 1.2 - Головна сторінка Better Impact

Better Impact знайшов місце в сфері моніторингу ресурсів, а саме:

- Відстеження людських ресурсів: Модуль Volunteer Impact дозволяє фіксувати години роботи волонтерів, їх кваліфікацію, доступність та навантаження;

- Через кастомні поля можна додавати інформацію про гуманітарну допомогу, обладнання або інші матеріали, встановлювати нагадування про терміни придатності;
- Donor Impact забезпечує відстеження пожертв, генерацію квитанцій та аналіз донорських кампаній.

Таблиця 1.2 – Аналіз Better Impact

Критерій	Better Impact
Управління ресурсами	Орієнтоване на волонтерів: відстеження годин, активностей. Не має інструментів для управління фізичними ресурсами.
Координація волонтерів	Спеціалізований інструмент: реєстрація, розклад, автоматичні сповіщення.
Інтеграція з месенджерами	Підтримка email/SMS-сповіщень. API для інтеграції з месенджерами обмежена.
Ролі користувачів	Ролі для адміністраторів, організацій та волонтерів
Вартість	Платна підписка (від $50/міс).
Кастомізація	Обмежена: налаштування форм та звітів.
Захист даних	Хмарне рішення зі стандартним шифруванням.

Для організацій, які працюють у кризових умовах, це дозволяє оперативно корегувати розподіл ресурсів на основі актуальних даних.

Варто зазначити недоліки Better Impact:

- Вартість: Ціна залежить від кількості активних волонтерів і може швидко зростати для великих організацій. Додаткові послуги (навчання, імпорт даних) оплачуються окремо.
- Деякі користувачі відзначають незручність генерації складних звітів і необхідність експорту даних для глибшого аналізу.
- Застарілий дизайн і "громіздкий" UI, що ускладнює навігацію для нетехнічних користувачів.

1.3.3 Odoo

Odoo — це модульна ERP-платформа з відкритим кодом, яка об'єднує інструменти для управління бізнес-процесами: від бухгалтерії та логістики до маркетингу та виробництва. Вона доступна у двох версіях: Community (безкоштовна) та Enterprise (платна з додатковими функціями). Odoo використовують компанії різних розмірів для автоматизації операцій, але її гнучкість також дозволяє адаптувати систему під потреби громадських організацій, благодійних фондів або волонтерських ініціатив. Наприклад, модуль Inventory дозволяє відстежувати гуманітарні запаси, а Project — керувати завданнями волонтерів [7].

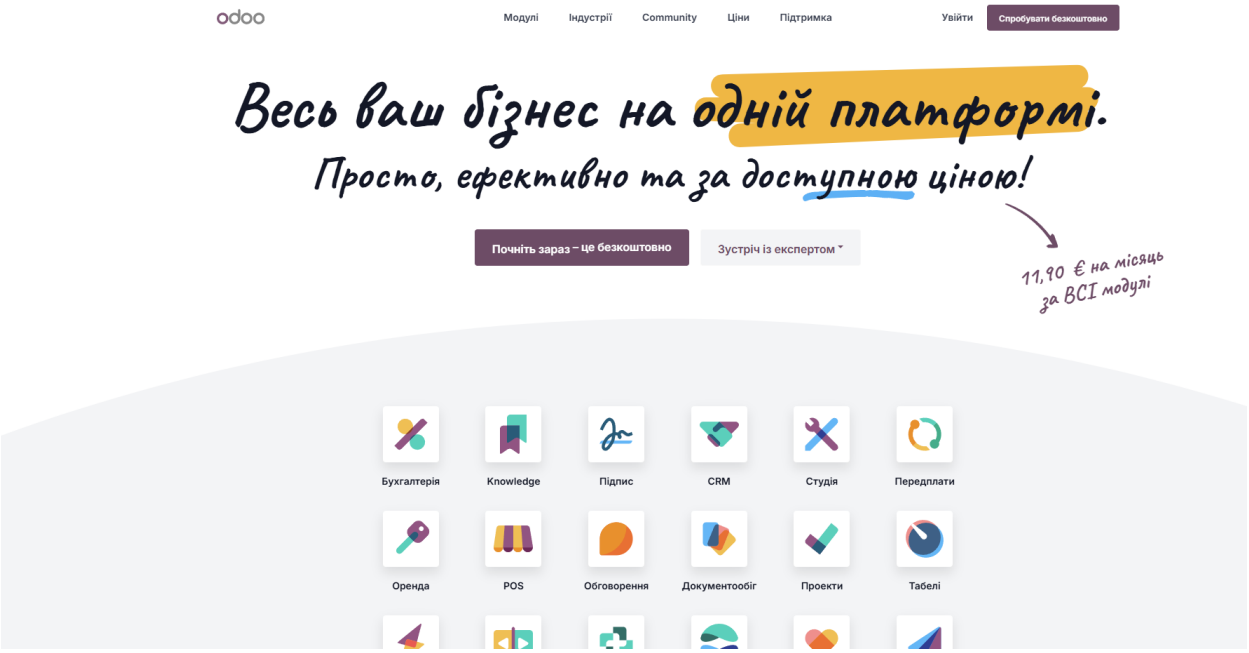


Рисунок 1.3 - Головна сторінка Odoo

Odoo є потужним інструментом моніторингу завдяки інтеграції:

- Модуль Accounting автоматизує облік пожертв, витрат і генерує звіти.
- Модуль HR дозволяє створювати профілі волонтерів, фіксувати їх години роботи, кваліфікацію та навантаження.

- Модуль Inventory відстежує надходження, витрати та терміни придатності товарів (наприклад, ліків або продуктів харчування).
- Модуль Project координує завдання, дедлайни та ресурси, виділені на конкретні ініціативи.

Таблиця 1.3 – Аналіз Odoo

Критерій	**Odoo**
Управління ресурсами	Модуль інвентаризації дозволяє відстежувати фізичні ресурси, але потребує налаштувань для гуманітарних потреб.
Координація волонтерів	Модуль HR дозволяє керувати волонтерами, але не спеціалізований для цього.
Інтеграція з месенджерами	Інтеграція можлива через Odoo Studio або API, але вимагає технічних навичок.
Ролі користувачів	Система ролей налаштовується під потреби (адмін, менеджер, волонтер).
Вартість	Community-версія безкоштовна; Enterprise-версія — від $25/користувач/міс.
Кастомізація	Дуже висока (відкритий код, модульна архітектура).
Захист даних	Рівень захисту залежить від версії (Enterprise має додаткові функції).

Для підвищення ефективності можна використовувати кастомні поля (наприклад, "Тип ресурсу" або "Пріоритетність") та автоматичні алерти (нагадування про критичні запаси).

З плюсів в Oddo можна виділити відкритий код, модульність, гнучкість, масштабованість та активну спільноту та підтримку.

Користувачі Odoo відзначають складність налаштування. Для кастомізації потрібні технічні навички, таких як робота з Python або з XML. Без досвіду адміністрування баз даних важко розгорнути локальну версію.

Також в Oddo присутній такий недолік як витрати на підтримку. Хмарна версія коштує від $24.90/користувач на місяць, що може бути дорого для НКО. Великі обсяги даних сповільнюють роботу системи, особливо на слабких серверах.

Ще деяка частина користувачів зазначають, що деякі модулі мають заплутану навігацію, що ускладнює роботу нетехнічних користувачів.

Таблиця 1.4 – Порівняльна таблиця

Критерій	CiviCRM	Better Impact	Odoo
Управління ресурсами	Без підтримки фізичних ресурсів.	Без інструментів для фізичних ресурсів.	Потребує налаштувань для гуманітарних потреб.
Координація волонтерів	Потрібна кастомізація/ додатки.	Спеціалізований інструмент: реєстрація, розклад.	Модуль HR для керування волонтерами. Не спеціалізований.
Інтеграція з месенджерами	Без вбудованої підтримки.	API для месенджерів обмежена.	Інтеграція через Odoo Studio/API. Треба технічні навички.
Ролі користувачів	Гнучкі ролі: адміністратор, менеджер.	Стандартні: волонтери, адміністратори.	Гнучкі ролі
Вартість	Безкоштовна	Від $50/місяць.	від $25/ міс
Кастомізація	Висока.	Обмежена	Дуже висока
Захист даних	Базове шифрування.	Хмарне рішення зі стандартним шифруванням.	Enterprise має додаткові функції.

ВИСНОВОК ДО РОЗДІЛУ 1

У рамках цього розділу було проаналізовано існуючі рішення для моніторингу ресурсів громадських організацій, такі як CiviCRM, Better Impact та Odoo. Хоча кожна з цих платформ пропонує окремі функції, що частково відповідають концепції дипломної роботи, жодна з них не є повноцінною для вирішення поставлених завдань:

CiviCRM — спеціалізований для неприбуткового сектору, проте обмежений у можливостях автономної роботи та аналізу матеріальних ресурсів.

Better Impact — ефективний для управління волонтерами, але не надає інструментів для моніторингу фінансів або гуманітарних запасів.

Odoo — універсальна ERP-платформа, але її кастомізація під специфічні потреби вимагає технічної експертизи, а безкоштовна версія має обмежений функціонал.

У процесі аналізу можливих підходів до реалізації системи моніторингу ресурсів для громадських організацій та волонтерів було розглянуто кілька варіантів, зокрема використання готових CRM- або ERP-систем, а також розробка власного веб-додатку. У об’єкті розробки цієї роботи – створення системи моніторингу ресурсів, – було прийнято рішення створювати веб-додаток (з архітектурою REST API) з моніторингу ресурсів для ГО та волонтерів.

Таким чином буде досягнуті гнучкість та масштабованість, тому що завдяки розділенню клієнтської та серверної логіки REST-архітектура дозволяє ефективно масштабувати додаток відповідно до зростання числа користувачів або організацій. На відміну від готових CRM або ERP-систем, які зазвичай мають складну структуру, надлишковий функціонал та обмежені можливості кастомізації. Власний веб-додаток дозволяє створити цілеспрямоване рішення з урахуванням особливостей роботи громадських організацій та волонтерів.

РОЗДІЛ 2. ОГЛЯД ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ

В цьому розділі буде розглянуто весь функціонал, що потрібен для реалізації системи моніторингу ресурсів. Також будуть визначені інструменти та фреймворки, завдяки яким буде реалізовано проєкт.

На сьогоднішній день існує велика кількість інструментів для роботи як на фронтенді, так і на бекенді. Надалі будуть розглянуті одні з найкращих інструментів для роботи на кожному з етапів створення сайту.

2.1 Функціональна модель системи. Опис

Функціональна модель системи – це основні компоненти, процеси та взаємодії між ними для ефективного управління ресурсами в громадських організаціях. Веб-застосунок, який є реалізацією системи моніторингу ресурсів, повинен виконувати такі функції:

- Управління ресурсами. Система забезпечує створення, редагування, перегляд та видалення інформації про ресурси. Кожен ресурс містить таку інформацію як назва, категорія, кількість, одиниця виміру, термін придатності, місце зберігання, статус (доступний/використаний);
- Фільтрація та пошук за категоріями;
- Реєстрація та профілі. Зберігаються дані волонтерів (ПІБ, контакти, навички, статус активності);
- Звітність. Фіксується виконана волонтером робота через веб-інтерфейс;

- Ролі користувачів. Гість (перегляд публічних ресурсів), волонтер (Додавання/редагування своїх записів), адміністратор (управління користувачами та системними налаштуваннями).
- Шифрування даних: Захист конфіденційної інформації

Користувач взаємодіє з системою через веб-інтерфейс. Клієнтська частина відправляє запити до API. Сервер обробляє запити, зберігає/отримує дані з бази даних та повертає відповідь. Адміністратор керує системою через певний фреймворк.

З такою моделлю маємо таку перевагу як централізоване управління, а саме той факт, що всі ресурси та волонтери в єдиній системі.

Ця модель забезпечує прозорість, ефективність та контроль у роботі з ресурсами, що є важливим для громадських організацій з обмеженими трудовими чи фінансовими можливостями.

2.2 Клієнтська частина. Огляд технологій

2.2.1 Angular.js

Angular.js — це JavaScript-фреймворк з відкритим вихідним кодом, розроблений Google для створення SPA. Він був представлений у 2010 році та став одним з перших фреймворків, який запровадив концепцію двостороннього зв’язування даних. Спочатку Angular.js створювався як інструмент для спрощення розробки фронтенду, але з часом його замінила нова версія — Angular (2+) – повна переробка з використанням TypeScript [8].

Архітектура Angular.js ґрунтується на шаблонах MVC (Model-View-Controller) та MVVM (Model-View-ViewModel). У MVC модель відповідає за дані, представлення (View) — за відображення інтерфейсу, а контролер

(Controller) керує логікою взаємодії між ними. MVVM у Angular.js реалізується через двостороннє зв’язування даних: зміни в моделі автоматично оновлюють View, і навпаки, що значно спрощує синхронізацію інтерфейсу з даними.

Однією з ключових переваг Angular.js — двостороннє зв’язування даних. Наприклад, якщо користувач змінює значення в полі введення, модель даних оновлюється миттєво без додаткового коду, і ці зміни відображаються у всіх пов’язаних елементах інтерфейсу. Це зменшує кількість шаблонного коду та прискорює розробку простих додатків.

Недоліки Angular.js включають середній поріг входу через складні концепції. низьку продуктивність у великих додатках через надмірне використання двостороннього зв’язування, а також застарілість — з 2016 року фреймворк не підтримується офіційно, а його наступник (Angular 2+) використовує підхід, несумісний з оригінальною версією.

Angular.js став революційним інструментом свого часу, але сьогодні його використання обмежене через застарілу архітектуру та появу сучасних альтернатив.

Таблиця 2.1 – Порівняльний аналіз Angular (2+)

Критерій	***Angular (2+)***
Тип	Повноцінний фреймворк (MVC/MVVM)
Мова	TypeScript
Архітектура	Модель-Вид-Контролер (MVC) або MVVM
Зв’язування даних	Двостороннє
Крива навчання	Висока (TypeScript, ін’єкція залежностей, RxJS)
Продуктивність	Середня (залежить від оптимізації)
Екосистема	Повна (все "з коробки")
Гнучкість	Обмежена (жорстка структура)
Популярність	Корпоративний сектор (Google, Microsoft)
Кращі сценарії	корпоративні додатки зі складними бізнес-правилами

2.2.2 Vue.js

Vue.js — це прогресивний JavaScript-фреймворк з відкритим вихідним кодом, створений Еваном Ю у 2014 році для розробки інтерфейсів веб-додатків. Спочатку він задумувався як легкий інструмент для поетапного вдосконалення існуючих проектів, але з часом перетворився на повноцінний фреймворк, який використовують для складних односторінкових додатків (SPA). Vue.js відомий своєю простотою, гнучкістю та детальною документацією, що зробило його популярним серед розробників різного рівня.

Головна особливість Vue.js — це його прогресивна архітектура. На відміну від Angular чи React, Vue.js можна почати використовувати як легку бібліотеку для окремих елементів сторінки (наприклад, меню, що випадають або форм), а потім масштабувати до повноцінного фреймворку з маршрутизацією, станом додатку (Vuex) та серверним рендерингом (Nuxt.js). Це дозволяє адаптувати його під будь-який рівень складності проекту.

Одна з ключових переваг Vue.js — двостороннє зв’язування даних через директиву v-model. Наприклад, якщо користувач вводить текст у поле форми, зміни миттєво синхронізуються з об’єктом даних компонента, і навпаки — оновлення даних автоматично відображається в інтерфейсі. Це спрощує розробку форм та інших інтерактивних елементів порівняно з тим же React, де потрібно явно керувати станом через useState [9].

Недоліки Vue.js включають меншу популярність у корпоративному сегменті порівняно з React або Angular, що обмежує кількість готових рішень для специфічних задач. Також, незважаючи на простоту, фреймворк може вимагати додаткових інструментів (наприклад, Vuex для глобального стану) для великих проектів, що ускладнює його використання в порівнянні з Angular, де більшість функцій вже інтегровані.

Підбиваючи підсумки, можна сказати, що Vue.js — це ідеальний вибір для розробників, які шукають баланс між простотою, гнучкістю та потужністю, але його застосування в масштабних корпоративних проектах може вимагати додаткових зусиль.

Таблиця 2.2 – Порівняльний аналіз Vue.js

Критерій	***Vue.js***
Тип	Прогресивний фреймворк
Мова	JavaScript/TypeScript
Архітектура	Компонентна з опціональними інструментами (Vuex)
Зв'язування даних	Двостороннє (через v-model)
Крива навчання	Низька (простий синтаксис, документація)
Продуктивність	Висока (легкий фреймворк)
Екосистема	Гнучка (Vuex, Vue Router, Nuxt.js)
Гнучкість	Висока (можна використовувати поступово)
Популярність	Стартапи, невеликі проекти
Кращі сценарії	Швидка розробка SPA, інтеграція з існуючими проектами

2.2.3 React.js

React.js — це JavaScript-бібліотека з відкритим вихідним кодом, створена Facebook (Meta) у 2011 році для розробки динамічних інтерфейсів. Вона була вперше використана у Facebook для оновлення стрічки новин, а в 2013 році стала доступною для всіх. React орієнтований на компонентний підхід, що дозволяє розробникам створювати UI-елементи, які можна .. Його головна мета — ефективне оновлення інтерфейсу за допомогою віртуального DOM, що робить його швидшим за традиційні підходи.

Унікальна риса React.js — це віртуальний DOM. На відміну від інших фреймворків, які працюють з реальним DOM, React створює його легку копію в пам’яті. При змінах даних React порівнює віртуальний DOM з реальним і оновлює лише ті частини інтерфейсу, які змінилися. Це значно покращує продуктивність, особливо у великих додатках. Додатково, React підтримує JSX — синтаксис, що дозволяє поєднувати HTML з JavaScript, роблячи код більш інтуїтивним [10].

React.js використовує однобічний потік даних, який робить код більш передбачуваним. На відміну від Angular або Vue.js, де використовується двостороннє зв’язування, React вимагає явного оновлення стану через функції, таких як useState. Однак за допомогою бібліотеки Formik та інших можна імітувати двостороннє зв’язування для форм, зберігаючи контроль над даними.

Фреймворк React.js має такий недолік як відсутність вбудованих рішень для маршрутизації або HTTP-запитів, однак це компенсується бібліотеками. Також можна зазначити, що надмірна гнучкість може призводити до неузгодженості в архітектурі великих проектів.

Спираючись на наведену інформацію, робимо висновок, що React.js — це потужний інструмент для створення швидких і масштабованих інтерфейсів, але його ефективне використання вимагає глибокого розуміння екосистеми та додаткових бібліотек.

React.js було обрано для цього проєкту через низку ключових переваг, які ідеально відповідають вимогам системи моніторингу ресурсів:

1. Компонентний підхід. Система потребує багато повторюваних UI-елементів (наприклад, списки ресурсів, форми додавання, картки волонтерів). React дозволяє розбити інтерфейс на незалежні компоненти, які можна використовувати повторно;

2. Висока продуктивність завдяки віртуальному DOM. У системі моніторингу ресурсів дані оновлюються часто (наприклад, зміна кількості ліків, додавання нових завдань). React оптимізує рендеринг через віртуальний DOM,

що дозволяє оновлювати лише ті частини інтерфейсу, які змінилися. Це критично важливо для швидкого відображення змін у реальному часі, роботи з великими наборами даних без "підвисань" інтерфейсу.

3. Багата екосистема бібліотек

React.js став оптимальним вибором через швидкість розробки, продуктивність та гнучкість, що дозволило створити інтуїтивний та масштабований інтерфейс для системи моніторингу ресурсів.

Таблиця 2.3 – Порівняльний аналіз React.js

Критерій	***React.js***
Тип	Бібліотека для UI (потрібні додаткові інструменти для повноцінного додатку)
Мова	JavaScript/TypeScript (JSX)
Архітектура	Компонентна з віртуальним DOM
Зв'язування даних	Одностороннє (потрібні бібліотеки типу Redux/MobX)
Крива навчання	Середня (JSX, стан, контекст)
Продуктивність	Висока (віртуальний DOM)
Екосистема	Велика (React Router, Redux, Next.js)
Гнучкість	Дуже висока (вибір інструментів на розсуд розробника)
Популярність	Висока (Facebook, Instagram, Netflix)
Кращі сценарії	Складні UI, масштабовані додатки, мобільні додатки (React Native)

2.3 Серверна частина. Огляд технологій

2.3.1 Node.js (JavaScript)

Node.js — це середовище виконання JavaScript, створене Районом Далем у 2009 році на основі рушія V8 від Google, яке дозволило використовувати JavaScript для серверної розробки. Спочатку воно викликало сумніви через незвичну асинхронну модель, але швидко стало популярним завдяки здатності обробляти тисячі одночасних з'єднань з мінімальними ресурсами. Ця технологія стала ключовою для стартапів та додатків реального часу, таких як чати або потокові сервіси, де швидкість та ефективність критично важливі.

Головна особливість Node.js — це його орієнтована на події архітектура з циклом подій (Event Loop). На відміну серверних мов, які створюють окремий потік для кожного запиту, Node.js використовує один основний потік, асинхронно обробляючи операції введення-виведення (наприклад, запити до бази даних). Це робить його ідеальним для високонавантажених додатків, де важлива швидкість відгуку, але погано підходить для задач із інтенсивними обчисленнями, наприклад, рендеринг відео [11].

Серед переваг платформи Node.js є JavaScript. Це універсальна мова для фронтенду та бекенду, що зменшує розрив між командами розробників. Велика екосистема npm з тисячами бібліотек дозволяє швидко інтегрувати готові рішення, а підтримка реального часу через WebSocket спрощує створення інтерактивних додатків. Однак асинхронний підхід може ускладнювати налагодження через "пекло зворотних викликів", яке частково вирішується синтаксисом async/await.

Недоліки Node.js пов'язані з обмеженням його архітектури: важкі обчислювальні задачі блокують основний потік, погіршуючи продуктивність усього додатку. Також існує ризик залежності від неякісних або застарілих

пакетів npm, що може призводити до вразливостей або несумісностей. Крім того, відсутність строгої типізації в JavaScript ускладнює масштабування великих проектів порівняно з TypeScript або Java.

Таблиця 2.4 – Порівняльний аналіз Node.js

Критерій	***Node.js***
Тип	Runtime (JavaScript)
Мова	JavaScript/TypeScript
Архітектура	Орієнтована на події (Event Loop)
Продуктивність	Висока для I/O-операцій, низька для CPU-інтенсивних задач
Крива навчання	Низька (для простих додатків), але асинхронність може ускладнити
Екосистема	Найбільша (npm), але якість бібліотек може варіюватися
Масштабованість	Добра для мікросервісів, погана для CPU-інтенсивних задач
Кращі сценарії	Real-time додатки, чати, API, потокове відео
Переваги	Асинхронність, велика спільнота, швидкий старт
Недоліки	Callback hell (без async/await), проблеми з CPU-інтенсивними задачами

Node.js безперечно є потужним інструментом для швидких та масштабованих мережевих додатків, але його слід обирати обережно, враховуючи специфіку задач та можливі обмеження.

2.3.2 Spring Boot (Java)

Java Spring Boot — це фреймворк з відкритим кодом, створений у 2014 році як частина екосистеми Spring для спрощення розробки автономних Java-додатків. Він став відповіддю на складність класичного Spring, де розробникам доводилося вручну налаштовувати конфігурації, сервери та XML-файли. Spring Boot автоматизував ці процеси, ставши стандартом для швидкого створення корпоративних рішень, особливо в архітектурі мікросервісів. Його унікальна риса — стартер-пакети, такі як spring-boot-starter-web, які автоматично підключають усі необхідні бібліотеки для певного типу додатків, усуваючи проблеми з сумісністю версій.

Переваги Spring Boot полягають у автоматичній конфігурації компонентів на основі доданих залежностей, що значно зменшує час налаштування. Наприклад, використання файлу application.properties дозволяє швидко підключити базу даних без написання коду. Вбудовані сервери, такі як Tomcat або Jetty, дозволяють запускати додаток як виконуваний JAR-файл без зовнішніх інструментів. Анотації типу @SpringBootApplication приховують рутинні налаштування, зосереджуючи увагу на бізнес-логіці. Також Spring Boot ідеально інтегрується з інструментами для мікросервісів, як-от Spring Cloud або Kubernetes [12].

Недоліки Spring Boot пов’язані з його архітектурою: стартер-пакети часто включають непотрібні залежності, що збільшує розмір виконуваних файлів. Автоматична конфігурація, хоча й зручна, може ускладнити глибоку кастомізацію або інтеграцію з нестандартними системами. Для великих монолітних додатків Spring Boot може бути менш ефективним порівняно зі спеціалізованими рішеннями.

Таблиця 2.5 – Порівняльний аналіз Spring Boot

Критерій	***Spring Boot***
Тип	Framework
Мова	Java
Архітектура	MVC, Dependency Injection
Продуктивність	Дуже висока (JVM оптимізація)
Крива навчання	Висока (синтаксис Java, Spring-специфічні анотації)
Екосистема	Велика (Maven, Spring Ecosystem), стабільна
Масштабованість	Відмінна (підтримка кластерів, розподілених систем)
Кращі сценарії	Корпоративні системи, банківські додатки, великі проекти
Переваги	Стабільність, безпека, підтримка корпоративних стандартів
Недоліки	Велика кількість шаблонного коду, висока складність для дрібних задач

Серед великої кількості фреймоврків Spring Boot займає не останнє місце завдяки швидкої розробки мікросервісів та корпоративних додатків, але його ефективність залежить від правильного використання залежностей та розуміння його архітектурних особливостей.

2.3.3 .NET (C#)

C# — це об’єктно-орієнтована мова програмування, розроблена Microsoft у 2000 році в рамках ініціативи .NET. Її створив Андерс Хейлсберг, автор Turbo Pascal і Delphi, з метою поєднати потужність C++ з простотою Visual Basic. Спочатку C# був задуманий для розробки Windows-додатків, але з часом, завдяки .NET Core, став кросплатформним. Сьогодні він використовується для вебу, IoT та корпоративних рішень.

Унікальна риса C# — це глибока інтеграція з екосистемою Microsoft, зокрема з такими інструментами, як Visual Studio, Azure та SQL Server. Наприклад, мова підтримує LINQ (Language Integrated Query), який дозволяє писати SQL-подібні запити прямо в коді для роботи з колекціями, базами даних або XML. Це робить C# ідеальним для складних бізнес-додатків, де потрібна тісна взаємодія з даними.

Переваги C# включають строгу статичну типізацію, яка зменшує кількість помилок на етапі компіляції, та автоматичне керування пам'яттю через збирач сміття. Кросплатформність .NET Core дозволяє розробляти додатки для будь-яких ОС, а вбудована підтримка асинхронного програмування (async/await) спрощує роботу з мережевими операціями.

Недоліки C# пов'язані з історичною залежністю від Windows: навіть після появи .NET Core деякі бібліотеки та фреймворки такі як WPF залишаються платформно-обмеженими. Для мікроконтролерів або високонавантажених систем C# може поступатися C/C++ через накладні витрати на виконання віртуальної машини .NET.

Таблиця 2.6 – Порівняльний аналіз .NET

Критерій	***.NET***
Тип	Framework
Мова	C#
Архітектура	MVC, Middleware
Продуктивність	Висока (AOT-компіляція в .NET Core)
Крива навчання	Середня (синтаксис C#, LINQ)
Екосистема	Велика (NuGet), інтеграція з Microsoft
Масштабованість	Відмінна (підтримка Docker, Kubernetes)
Кращі сценарії	Корпоративні рішення, ігри (Unity), Windows-додатки
Переваги	Інтеграція з Microsoft, LINQ, async/await
Недоліки	Обмежена кросплатформність для деяких бібліотек

2.3.4 Django (Python)

Django — це високорівневий веб-фреймворк на Python, створений у 2005 році Адріаном Головатим та Саймоном Віллісоном для швидкої розробки складних веб-додатків. Спочатку він використовувався в газеті Lawrence Journal-World для управління контентом, але завдяки своїй ефективності та зручності став одним з найпопулярніших фреймворків у світі. Django дотримується принципу "батарейки в комплекті" — він надає розробникам усі необхідні інструменти "з коробки", від аутентифікації до роботи з базами даних.

Унікальна риса Django — це його ORM (Object-Relational Mapping), який дозволяє працювати з базами даних через Python-код, без необхідності писати SQL-запити вручну. Наприклад, ви можете створити модель даних класом Python, і Django автоматично згенерує SQL-таблиці, а також надасть API для CRUD-операцій. Це робить фреймворк ідеальним для проектів, де важлива швидкість розробки без втрати гнучкості.

Переваги Django включають автоматичну генерацію адмін-панелі для управління даними, вбудований захист від поширених веб-атак, а також можливість масштабування від невеликих блогів до складних систем, таких як Instagram або Spotify. Фреймворк підтримує розділення логіки за шаблоном MTV, що спрощує структуру коду. Крім того, тисячі пакетів як Django REST Framework значно розширюють його функціонал.

Недоліки Django пов’язані з його монолітною архітектурою: для дуже простих завдань (наприклад, статичний сайт) він може бути надмірним через велику кількість вбудованих інструментів. Деякі рішення, такі як ORM, можуть обмежувати гнучкість у складних сценаріях, де потрібен тонкий контроль над SQL.

Таблиця 2.7 – Порівняльний аналіз Django

Критерій	***Django***
Тип	Framework
Мова	Python
Архітектура	MTV (Model-Template-View)
Продуктивність	Середня (інтерпретована мова)
Крива навчання	Низька (простий синтаксис Python)
Екосистема	Велика (PyPI), спеціалізовані пакети для вебу
Масштабованість	Добра для монолітів, обмежена для мікросервісів
Кращі сценарії	Веб-додатки зі швидкою розробкою, CMS, стартапи
Переваги	Швидка розробка, ORM, адмін-панель "з коробки"
Недоліки	Менша продуктивність, обмежена гнучкість ORM для складних запитів

Django на Python є оптимальним для цього проєкту через швидкість розробки (адмін-панель "з коробки", ORM), ідеальну сумісність з React через REST API та баланс між продуктивністю та простотою.

Альтернативи (наприклад, Node.js або Go) могли б використовуватись, але вимагали б більше часу на реалізацію тих же функцій, які Django надає "з коробки".

Як результат, структура системи матиме такий вигляд:

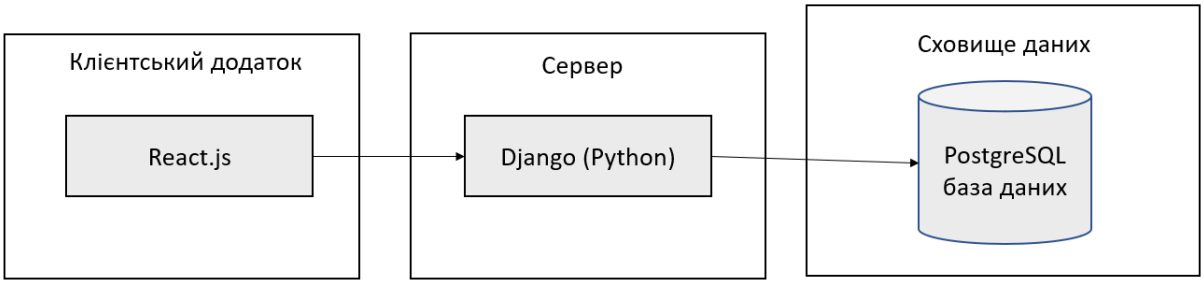


Рисунок 2.1 – Структура системи

ВИСНОВОК ДО РОЗДІЛУ 2

Аналіз сучасних технологій для розробки системи моніторингу ресурсів дозволив визначити оптимальний стек, що поєднує React.js (фронтенд) та Django (бекенд). React.js було обрано через компонентний підхід, який забезпечує швидку розробку динамічних інтерфейсів з можливістю повторного використання UI-елементів, а також віртуальний DOM для ефективного оновлення даних у реальному часі. Для бекенду Django став найкращим рішенням завдяки вбудованим інструментам безпеки, ORM для роботи з PostgreSQL та автоматичній генерації REST API через Django REST.

Порівняння з альтернативами, а саме Vue.js, Angular, Node.js, Spring Boot, показало, що обраний стек є найбільш зрілим для задач проекту:

- Vue.js поступається React у кількості готових компонентів (наприклад, Material-UI), що збільшило б час розробки
- Node.js менш підходить для CRUD-операцій зі складними бізнес-правилами, які ефективніше реалізовувати через Django ORM.
- Spring Boot вимагав би більше часу на налаштування та мав би надмірну складність для середнього проекту.

Важливу роль відіграла екосистема технологій:

- React з бібліотеками (axios, react-router-dom) забезпечує гнучкість у роботі з API та маршрутизацією.
- Django надає готові рішення для автентифікації, адмін-панелі та обробки медіа, що критично для системи з обліком ресурсів.

Підсумовуючи, обраний стек (React + Django) оптимально відповідає вимогам системи до масштабованості, безпеки та швидкості розробки, забезпечуючи ефективну взаємодію між волонтерами та громадськими організаціями.

РОЗДІЛ 3. ДЕТАЛІ РОЗРОБКИ СИСТЕМИ

Відповідно до досліджених матеріалів та обраного способу реалізації ми можемо перейти до фази розробки системи.

3.1 Архітектура системи

Розроблена система має клієнт-серверну архітектуру, що складається з:

- Frontend (клієнтська частина) — реалізована на React, відповідає за взаємодію з користувачем, відображення інтерфейсу, відправку та отримання даних через API.
- Backend (серверна частина) — реалізована на Django (Django REST Framework), забезпечує обробку запитів, зберігання та обробку даних, а також реалізацію бізнес-логіки.
- База даних – була обрана база даних PostgreSQL. Забезпечує надійне зберігання даних [20].

Взаємодія клієнт - сервер здійснюється через HTTP-запити до REST API.

Клієнтська частина має в собі такий функціонал:

- Відображення інтерфейсу користувача (UI);
- Валідацію даних на клієнті;
- Відправку запитів до API та обробку відповідей;
- Зберігання токена автентифікації та даних користувача у localStorage.

Серверна частина буде відповідальне за:

- Обробку HTTP-запитів від клієнта;
- Валідацію та обробку даних на сервері;
- Аутентифікацію та авторизацію користувачів;

- Зберігання даних у базі даних;
- Логування дій користувачів (створення запису в історії змін).

Вся клієнтська частина побудована на основі компонентів. Кожна сторінка або функціональний блок — це окремий компонент. Як приклад, в проекті є такі сторінки як ProfilePage (сторінка профілю волонтера), ResourcesPage (сторінка з ресурсами), або CarouselCard — окрема картка для каруселі на головній сторінці.

Всі запити до серверу винесені в окремий файл-сервіс (services/api.js). Це дозволяє централізовано обробляти всі HTTP-запити, повторно використовувати код і легко змінювати логіку взаємодії з сервером.

```
// services/api.js
const volunteerService = {
  getVolunteerById: async (id) => {
    const response = await api.get(`/volunteers/${id}/`);
    return response.data;
  },
  // ...інші методи
};
```

Рисунок 3.3 - Приклад функції-сервісу

Для зберігання інформації про поточного користувача використовується React Context (UserContext). Це дозволяє отримувати дані користувача у будь-якому компоненті без необхідності прокидати пропси. Токен автентифікації та дані користувача зберігаються у localStorage, що дозволяє зберігати сесію навіть після перезавантаження сторінки. Для кожного компонента створено окремий CSS-файл, що забезпечує ізольованість стилів і зручність підтримки.

Щодо серверної частини, основні сутності системи описані у вигляді моделей Django: Volunteer — волонтер, Resource — ресурс, ActionLog — історія змін. Для кожної моделі створено окремий ViewSet, який відповідає за CRUD-операції. Наприклад, VolunteerViewSet обробляє всі запити, пов’язані з волонтерами.

Таблиця 3.1 – поля сутності Волонтер

id	унікальний ідентифікатор
first_name	Ім’я
last_name	Прізвище
middle_name	По батькові
email, phone, telegram_id	контактна інформація
skills	навички (рядок або список)
organization	організація, до якої належить волонтер
photo	фото профілю
description	додаткова інформація
registration_date, last_login	Дати регістрації та останнього входу в акаунт

Таблиця 3.2 – поля сутності Ресурс

id	унікальний ідентифікатор
name	назва ресурсу
category	категорія (тип) ресурсу
quantity, unit	кількість і одиниця виміру
storage_location	місце зберігання
photo	фото ресурсу
description	опис

Таблиця 3.3 – поля сутності ActionLog (Історія змін)

id	унікальний ідентифікатор
action	тип дії (додано, змінено, видалено)
subject	над чим виконано дію
description	опис дії
performer	хто виконав дію (користувач або система)
timestamp	дата і час

Для перетворення даних між моделями Django та JSON-відповідями використовуються серіалізатори (serializers.py). Вони також відповідають за валідацію даних.

Для захисту приватних ендпоінтів використовується токен-автентифікація (TokenAuthentication) [18].

Важливі дії (наприклад, реєстрація волонтера) фіксуються у моделі ActionLog для подальшого аудиту.

```
Frontend (React):
  ├── components/
  │     ├── profile/
  │     ├── main-page/
  │     └── resources-page/
  ├── services/api.js
  └── contexts/UserContext.js

Backend (Django):
  ├── api/
  │     ├── models.py
  │     ├── views.py
  │     ├── serializers.py
  │     └── urls.py
  └── settings.py
```

Рисунок 3.4 - Схематичне представлення модулів клієнтської частини

Кожен ресурс має свій endpoint:

- /api/volunteers/ — робота з волонтерами
- /api/resources/ — робота з ресурсами
- /api/action-logs/ — історія змін

Формат запитів і відповідей: всі дані передаються у форматі JSON.

В системі є автентифікація, тому для захищених дій (наприклад, редагування профілю, додавання ресурсу) потрібен токен, який передається у заголовку.

3.2 Реалізація серверної частини

Серверна частина проекту організована за допомогою Django та Django REST Framework.

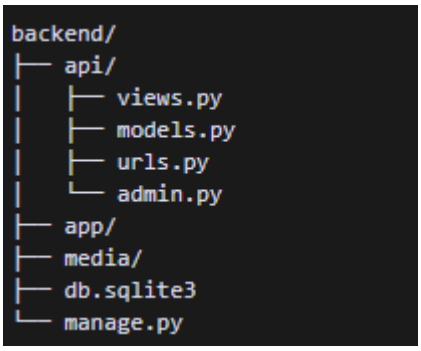


Рисунок 3.5 – Структура проекту серверної частини

3.2.1 Моделі. ViewSet

У системі реалізовано такі основні моделі (класи), які відповідають сутностям предметної області:

- Volunteer (Волонтер): Зберігає інформацію про користувачів-волонтерів: ПІБ, email, телефон, навички, організацію, фото, опис, дати реєстрації та останнього входу.
- Resource (Ресурс): Описує ресурси, якими оперують волонтери: назва, категорія, кількість, одиниця виміру, місце зберігання, фото, опис.
- ActionLog (Історія змін): Фіксує важливі дії в системі (наприклад, реєстрація волонтера, додавання ресурсу), містить тип дії, над чим виконано дію, опис, виконавця та час.

```python
class Volunteer(models.Model):
    STATUS_CHOICES = [
        ('active', 'Активний'),
        ('inactive', 'Неактивний'),
        ('pending', 'Очікує підтвердження'),
        ('blocked', 'Заблокований'),
    ]

    last_name = models.CharField(max_length=100, verbose_name="Прізвище", db_index=True)
    first_name = models.CharField(max_length=100, verbose_name="Ім'я", db_index=True)
    middle_name = models.CharField(max_length=100, blank=True, null=True, verbose_name="По Батькові")
    phone = models.CharField(max_length=20, verbose_name="Телефон")
    email = models.EmailField(max_length=100, verbose_name="Email", db_index=True)
    telegram_id = models.CharField(max_length=100, blank=True, null=True, verbose_name="Telegram ID")
    registration_date = models.DateTimeField(default=timezone.now, verbose_name="Дата реєстрації", db_index=True)
    status = models.CharField(max_length=20, choices=STATUS_CHOICES, default='pending', verbose_name="Статус", db_index=True)
    skills = models.TextField(blank=True, null=True, verbose_name="Навички")
    description = models.TextField(blank=True, null=True, verbose_name="Опис")
    organization = models.CharField(max_length=255, blank=True, null=True, verbose_name="Організація", db_index=True)
    photo = models.ImageField(upload_to='volunteer_photos/', blank=True, null=True, verbose_name="Фото")
    user = models.OneToOneField(User, on_delete=models.CASCADE, blank=True, null=True, verbose_name="Користувач")
    last_login = models.DateTimeField(blank=True, null=True, verbose_name="Останній вхід", db_index=True)
```

Рисунок 3.6 – Приклад моделі волонтера (models.py)

Модель містить всі необхідні поля для зберігання інформації про волонтера, включаючи особисті дані, контактну інформацію та системні поля.

У файлі backend/api/urls.py налаштовано маршрутизацію API.

urls.py

```python
from django.urls import path, include
from .views import ResourceViewSet, VolunteerViewSet, action_log_list
from rest_framework.routers import DefaultRouter

router = DefaultRouter()
router.register(r'resources', ResourceViewSet)
router.register(r'volunteers', VolunteerViewSet)

urlpatterns = [
    path('history/', action_log_list, name='action_log_list'),
    path('', include(router.urls)),
]
```

Рисунок 3.7 – Налаштування основних маршрутів API

Тут використовується DefaultRouter з Django REST framework для автоматичного створення URL-шляхів для API ендпоінтів [14].

Один рядок коду router.register(r'volunteers', VolunteerViewSet) автоматично створює наступні URL-маршрути:

- GET /volunteers/ - список всіх волонтерів;
- POST /volunteers/ - створення нового волонтера;
- GET /volunteers/{id}/ - отримання конкретного волонтера;
- PUT /volunteers/{id}/ - повне оновлення волонтера;
- PATCH /volunteers/{id}/ - часткове оновлення волонтера;

- DELETE /volunteers/{id}/ - видалення волонтера.

ViewSet - це важливий компонент Django REST Framework. ViewSet об'єднує логіку для набору пов'язаних представлень в єдиний клас. Для кожної сутності створено ViewSet, який забезпечує CRUD-операції. У проекті це демонструється в backend/api/views.py.

Цей ViewSet забезпечує фільтрацію за статусом та організацією, пошук по різних полях, сортування результатів, автоматичне логування дій та керування правами доступу. ViewSet дозволяє легко додавати нові методи та дії.

```python
class VolunteerViewSet(ActionLoggingMixin, viewsets.ModelViewSet):
    queryset = Volunteer.objects.all().select_related('user')
    serializer_class = VolunteerSerializer
    filter_backends = [DjangoFilterBackend, filters.SearchFilter, filters.OrderingFilter]
    filterset_fields = ['status', 'organization']
    search_fields = ['last_name', 'first_name', 'middle_name', 'skills', 'description', 'email', 'phone']
    ordering_fields = ['last_name', 'first_name', 'registration_date', 'last_login']
    ordering = ['last_name', 'first_name']
    permission_classes = [IsAuthenticated]

    def get_permissions(self):
        if self.action in ['create', 'login', 'list', 'retrieve']:
            return []
        return [IsAuthenticated()]
```

Рисунок 3.8 - Реалізація VolunteerViewSet

3.2.2 Реєстрація та автентифікація

```python
def create(self, request, *args, **kwargs):
    required_fields = ['first_name', 'last_name', 'email', 'phone', 'password']
    missing_fields = [field for field in required_fields if field not in request.data]

    if missing_fields:
        return Response(
            {"message": f"Відсутні обов'язкові поля: {', '.join(missing_fields)}"},
            status=status.HTTP_400_BAD_REQUEST
        )

    serializer = self.get_serializer(data=request.data)
    if not serializer.is_valid():
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    email = serializer.validated_data.get('email')

    # Перевірка унікальності email
    if User.objects.filter(email=email).exists() or Volunteer.objects.filter(email=email).exists():
        return Response(
            {"message": "Користувач з такою електронною поштою вже існує"},
            status=status.HTTP_400_BAD_REQUEST
        )
```

Рисунок 3.9 - Реалізація методу create (реєстрація)

Процес реєстрації починається з отримання POST-запиту на ендпоінт /volunteers/ з даними нового волонтера. Спочатку система перевіряє наявність усіх обов'язкових полів: ім'я, прізвище, email, телефон та пароль. Якщо якесь з цих полів відсутнє, повертається помилка 400 Bad Request з переліком відсутніх полів. Далі відбувається валідація даних через серіалізатор, який перевіряє формат та коректність введених даних. Особлива увага приділяється перевірці унікальності email - система перевіряє його відсутність як в таблиці користувачів Django (User), так і в таблиці волонтерів (Volunteer). Якщо email вже існує, повертається відповідна помилка. При успішній валідації система створює нового користувача Django з наданим email та паролем, а потім створює пов'язаний запис волонтера з усіма додатковими даними. Після створення запису автоматично генерується запис в історії дій через ActionLoggingMixin, де фіксується створення нового волонтера. Нарешті, система створює токен автентифікації для нового користувача та повертає успішну відповідь з токеном та даними створеного волонтера.

Автентифікація відбувається через POST-запит на ендпоінт /volunteers/login/ з наданням email та пароля [16]. Спершу система перевіряє наявність обох полів у запиті, і якщо якесь відсутнє, повертає помилку 400 Bad Request. Далі система шукає користувача за наданим email в базі даних Django. Якщо користувача не знайдено, повертається помилка 404 Not Found. Якщо користувач існує, система перевіряє наданий пароль за допомогою вбудованого механізму Django для перевірки хешованих паролів. При невірному паролі повертається помилка 401 Unauthorized. Після успішної перевірки пароля система шукає пов'язаний запис волонтера. Якщо запис волонтера не знайдено, повертається помилка 404 Not Found. Далі перевіряється статус волонтера - якщо він не 'active', система повертає помилку 403 Forbidden з повідомленням про необхідність підтвердження адміністратором. При успішній перевірці статусу система оновлює час останнього входу волонтера, створює або отримує

існуючий токен автентифікації та повертає успішну відповідь, яка містить токен та дані волонтера.

Система має історію змін, яка реалізована в класі ActionLoggingMixin.

```python
class ActionLoggingMixin:
    def get_performer_name(self):
        """Get the name of the action performer"""
        if self.request.user.is_authenticated:
            volunteer = getattr(self.request.user, 'volunteer', None)
            if volunteer:
                return f"{volunteer.first_name} {volunteer.last_name}"
            return self.request.user.get_full_name() or self.request.user.username
        return None

    def log_action(self, action, subject, description):
        """Create an action log entry"""
        ActionLog.objects.create(
            action=action,
            subject=subject,
            description=description,
            performer=self.get_performer_name()
        )
```

Рисунок 3.10- Реалізація системи логування

3.2.3 Адмін панель

В проекті було використано адміністративна панель Django.

Адмін-панель налаштована через декоратор @admin.register(Volunteer) у файлі backend/api/admin.py, де визначено клас VolunteerAdmin, який розширює функціональність стандартного інтерфейсу адміністратора.

Через параметр list_display ми налаштували відображення основних полів волонтера у списку (прізвище, ім'я, email, телефон, статус, організація, дата реєстрації та останній вхід), list_filter дозволяє фільтрувати волонтерів за статусом, датою реєстрації та організацією, а search_fields забезпечує пошук по різних полях. Особливу увагу приділено організації полів через fieldsets, де всі поля згруповані в логічні секції: "Особиста інформація" (прізвище, ім'я, по батькові, фото), "Контактна інформація" (телефон, email, telegram_id), "Додаткова інформація" (статус, навички, опис, організація) та "Системна

інформація" (користувач, дата реєстрації, останній вхід), причому остання секція за замовчуванням згорнута для кращої організації інтерфейсу.

Така реалізація адмін-панелі дозволяє ефективно управляти даними волонтерів: переглядати, редагувати, видаляти записи, змінювати статуси, здійснювати пошук та фільтрацію, що значно спрощує процес адміністрування системи без необхідності написання додаткового коду.

```python
@admin.register(Volunteer)
class VolunteerAdmin(admin.ModelAdmin):
    list_display = ('last_name', 'first_name', 'email', 'phone', 'status', 'organization', 'registration_date', 'last_login')
    list_filter = ('status', 'registration_date', 'organization')
    search_fields = ('last_name', 'first_name', 'email', 'phone', 'telegram_id', 'skills', 'organization', 'description')
    date_hierarchy = 'registration_date'
    fieldsets = (
        ('Особиста інформація', {
            'fields': ('last_name', 'first_name', 'middle_name', 'photo')
        }),
        ('Контактна інформація', {
            'fields': ('phone', 'email', 'telegram_id')
        }),
        ('Додаткова інформація', {
            'fields': ('status', 'skills', 'description', 'organization')
        }),
        ('Системна інформація', {
            'fields': ('user', 'registration_date', 'last_login'),
            'classes': ('collapse',)
        }),
    )
```

Рисунок 3.11 - Реалізація адмін панелі

3.3 Реалізація клієнтської частини

```
frontend/
├── src/
│   ├── components/
│   │   ├── auth/
│   │   ├── common/
│   │   ├── main-page/
│   │   ├── navigation/
│   │   ├── profile/
│   │   ├── resources-page/
│   │   ├── volunteers-page/
│   │   └── mission-page/
│   │
│   ├── contexts/
│   │   └── UserContext.js
│   │
│   ├── services/
│   │   └── api.js
│   │
│   ├── utils/
│   │   └── helpers.js
│   │
│   ├── App.js
│   ├── App.css
│   └── index.js
│
├── public/
│   ├── index.html
│   └── assets/
```

Рисунок 3.12 - Структура клієнтської частини

3.3.1 Основні компоненти

Компонент auth/

Містить компоненти для автентифікації та реєстрації користувачів:

- LoginModal.js - модальне вікно для входу в систему. Обробляє введення email та пароля, відправляє запит на сервер та обробляє помилки автентифікації. Використовує контекст користувача для збереження стану автентифікації.
- RegisterPage.js - сторінка реєстрації нового волонтера. Містить форму з валідацією полів, завантаженням фотографії та обробкою помилок.

Компонент common/

Містить загальні компоненти, які використовуються в різних частинах додатку:

- Footer.js - підвал сайту з контактною інформацією та посиланнями
- Інші допоміжні компоненти для відображення сповіщень, завантажувачів тощо

Компонент main-page/

Реалізує головну сторінку сайту:

- MainPage.js - основний компонент, який об'єднує всі секції головної сторінки
- join-us-section/JoinUsSection.js - секція із закликом приєднатися до волонтерської спільноти
- action-button/ActionButton.js - кастомізована кнопка з анімацією для головних дій

Компонент profile/

Відповідає за функціонал профілю користувача:

- ProfilePage.js - сторінка профілю волонтера, де відображається особиста інформація, статистика активності та можливість редагування даних. Реалізує функціонал оновлення даних профілю та зміни статусу активності.

Компонент resources-page/

Керує відображенням та управлінням ресурсами:

- ResourcesPage.js - головна сторінка ресурсів з фільтрацією та сортуванням
- resource-details/ResourceDetailsPage.js - детальна інформація про конкретний ресурс
- HistoryPage.js - сторінка історії дій з ресурсами

Компонент volunteers-page/

Відображає інформацію про волонтерів:

- VolunteersPage.js - список всіх волонтерів з можливістю фільтрації та пошуку
- Включає компоненти для відображення карток волонтерів та їх детальної інформації

Компонент navigation/

Відповідає за навігацію по сайту:

- Navigation.js - головне меню навігації сайту.

Компонент mission-page/

Містить інформацію про місію проекту:

- MissionPage.js - сторінка з описом цілей та завдань волонтерської платформи

Кожен компонент розроблений з урахуванням принципів повторного використання коду, розділення відповідальності, оптимізації продуктивності через React.memo та useCallback, адаптивного дизайну, доступності для

користувачів з обмеженими можливостями та інтуїтивно зрозумілого інтерфейсу [15].

Всі компоненти використовують Material-UI[13] для забезпечення єдиного стилю інтерфейсу та власні CSS-стилі для кастомізації під потреби проекту.

3.3.2 API

Файл services/api.js є ключовим компонентом frontend частини, який відповідає за всю взаємодію з серверним API. Цей файл реалізує сервіс, що інкапсулює всю логіку HTTP-запитів до бекенду.

На початку файлу створюється екземпляр axios з базовою конфігурацією [17]. Це дозволяє встановити базову URL-адресу для всіх запитів та налаштувати заголовки за замовчуванням. У нашому проекті це реалізовано через створення константи api, яка використовується для всіх подальших запитів.

Основним компонентом файлу є об'єкт volunteerService, який містить методи для роботи з волонтерами.

Метод реєстрації нового волонтера приймає об'єкт з даними користувача, створює FormData для відправки файлів (наприклад, фотографії профілю), та виконує POST-запит до серверу. Особливістю реалізації є автоматичний вхід після успішної реєстрації, що покращує користувацький досвід.

Сервіс також містить метод для входу користувача в систему. Цей метод відправляє дані автентифікації на сервер та обробляє отриману відповідь. У разі успішної автентифікації, токен зберігається в локальному сховищі браузера для подальшого використання.

Важливою частиною сервісу є обробка помилок. Кожен метод має блок try-catch, який перехоплює можливі помилки та форматує їх у зрозумілий для

користувача вигляд. Це дозволяє frontend-компонентам правильно відображати повідомлення про помилки.

Для роботи з захищеними ендпоінтами сервіс автоматично додає токен автентифікації до заголовків запитів. Це реалізовано через інтерцептор axios, який перехоплює кожен вихідний запит та додає необхідні заголовки.

Сервіс також містить методи для оновлення профілю користувача, отримання списку волонтерів, роботи з ресурсами та історією дій. Кожен метод відповідає за конкретний ендпоінт API та інкапсулює логіку взаємодії з ним.

Особливу увагу приділено обробці різних типів даних. Наприклад, при відправці файлів використовується FormData, а для звичайних даних - JSON. Це забезпечує коректну передачу даних на сервер.

Для оптимізації продуктивності в сервісі реалізовано кешування відповідей сервера для певних запитів. Це дозволяє зменшити кількість звернень до сервера та прискорити роботу додатку.

Важливим аспектом є також підтримка скасування запитів. Це реалізовано за допомогою CancelToken з axios, що дозволяє правильно обробляти ситуації, коли користувач переходить на іншу сторінку до завершення запиту.

Таким чином, services/api.js є центральним компонентом, який забезпечує всю взаємодію між frontend та backend частинами додатку, реалізуючи необхідні механізми безпеки, обробки помилок та оптимізації продуктивності.

3.3.3. UserContext

Файл contexts/UserContext.js є ключовим компонентом для управління глобальним станом користувача в додатку. Він використовує React Context API для забезпечення доступу до даних користувача та функцій автентифікації з будь-якого компонента додатку [19].

Цей контекст використовується в компонентах додатку. Наприклад, в компоненті LoginModal.js.

UserContext забезпечує наступну функціональність:

1. Зберігання стану користувача (дані профілю, статус автентифікації)
2. Управління процесом входу та виходу з системи
3. Автоматична перевірка автентифікації при завантаженні додатку
4. Оновлення даних користувача
5. Обробка помилок автентифікації
6. Збереження токену автентифікації в локальному сховищі

Важливою особливістю реалізації є використання useCallback для мемоізації функцій, що покращує продуктивність при перерендерингу компонентів. Також реалізовано механізм обробки помилок та стану завантаження, що дозволяє компонентам відповідно реагувати на різні ситуації.

Контекст інтегрується в додаток через UserProvider в кореневому компоненті, що робить функціонал доступним для всіх дочірніх компонентів. Хук useUser надає зручний спосіб доступу до контексту в будь-якому компоненті.

Така архітектура забезпечує централізоване управління станом користувача, що спрощує розробку та підтримку додатку, а також забезпечує узгодженість даних користувача в усьому додатку.

3.4 Робота з даними

На серверній частині дані зберігаються в базі даних через Django ORM.

```python
class Volunteer(models.Model):
    STATUS_CHOICES = [
        ('active', 'Активний'),
        ('inactive', 'Неактивний'),
        ('pending', 'Очікує підтвердження'),
        ('blocked', 'Заблокований'),
    ]

    last_name = models.CharField(max_length=100, verbose_name="Прізвище", db_index=True)
    first_name = models.CharField(max_length=100, verbose_name="Ім'я", db_index=True)
    middle_name = models.CharField(max_length=100, blank=True, null=True, verbose_name="По Батькові")
    phone = models.CharField(max_length=20, verbose_name="Телефон")
    email = models.EmailField(max_length=100, verbose_name="Email", db_index=True)
    telegram_id = models.CharField(max_length=100, blank=True, null=True, verbose_name="Telegram ID")
    registration_date = models.DateTimeField(default=timezone.now, verbose_name="Дата реєстрації", db_index=True)
    status = models.CharField(max_length=20, choices=STATUS_CHOICES, default='pending', verbose_name="Статус", db_index=True)
    skills = models.TextField(blank=True, null=True, verbose_name="Навички")
    description = models.TextField(blank=True, null=True, verbose_name="Опис")
    organization = models.CharField(max_length=255, blank=True, null=True, verbose_name="Організація", db_index=True)
    photo = models.ImageField(upload_to='volunteer_photos/', blank=True, null=True, verbose_name="Фото")
    user = models.OneToOneField(User, on_delete=models.CASCADE, blank=True, null=True, verbose_name="Користувач")
    last_login = models.DateTimeField(blank=True, null=True, verbose_name="Останній вхід", db_index=True)
```

Рисунок 3.13 - Основна модель для зберігання даних про волонтерів

На клієнтській частині для тимчасового зберігання даних використовується localStorage.

```javascript
// Збереження токену
localStorage.setItem('authToken', response.token);

// Отримання токену
const token = localStorage.getItem('authToken');

// Видалення при виході
localStorage.removeItem('authToken');
```

Рисунок 3.14 - Приклад зберігання токену автентифікації

На серверній частині фільтрація, пошук, сортування реалізована у ViewSet.

```python
class VolunteerViewSet(ActionLoggingMixin, viewsets.ModelViewSet):
    queryset = Volunteer.objects.all().select_related('user')
    serializer_class = VolunteerSerializer
    filter_backends = [DjangoFilterBackend, filters.SearchFilter, filters.OrderingFilter]
    filterset_fields = ['status', 'organization']
    search_fields = ['last_name', 'first_name', 'middle_name', 'skills', 'description', 'email', 'phone']
    ordering_fields = ['last_name', 'first_name', 'registration_date', 'last_login']
    ordering = ['last_name', 'first_name']
```

Рисунок 3.15 - ViewSet реалізація

На серверній частині валідація реалізована в методі create ViewSet.

На клієнтській частині валідація реалізована в компонентах форм.

```
const validateForm = useCallback(() => {
    if (formData.password.length < VALIDATION_RULES.password.minLength) {
        setError(VALIDATION_RULES.password.message);
        return false;
    }
    if (formData.password !== formData.confirmPassword) {
        setError(VALIDATION_RULES.confirmPassword.message);
        return false;
    }
    return true;
}, [formData.password, formData.confirmPassword]);
```

Рисунок 3.16 - Валідація в компоненті реєстрації

Для обробки помилок використовується єдиний формат відповіді з сервера.

```
const handleError = useCallback((error) => {
    if (error.response?.status) {
        setError(ERROR_MESSAGES[error.response.status] ||
            `Не вдалося увійти: ${error.response?.data?.detail || error.response?.data?.message || 'Невідома помилка'}`);
    } else if (error.message === 'Network Error') {
        setError(ERROR_MESSAGES.networkError);
    } else if (typeof error === 'string') {
        setError(error);
    } else if (error.message) {
        setError(error.message);
    } else {
        setError(ERROR_MESSAGES.default);
    }
}, []);
```

Рисунок 3.17 - Обробка помилок єдиного формату

Така архітектура забезпечує надійне зберігання даних в базі даних, ефективний пошук та фільтрацію даних, валідацію даних на обох рівнях (клієнт та сервер), зручну обробку помилок, оптимізацію запитів до бази даних, безпечне зберігання чутливих даних, гнучкість у розширенні функціоналу [21].

3.5 Безпека системи

В системі реалізовано багаторівневий захист персональних даних користувачів. На рівні бази даних це забезпечується через модель Django, де паролі автоматично хешуються перед збереженням.

На рівні API реалізовано систему дозволів, яка контролює доступ до персональних даних.

```python
class VolunteerViewSet(ActionLoggingMixin, viewsets.ModelViewSet):
    permission_classes = [IsAuthenticated]

    def get_permissions(self):
        if self.action in ['create', 'login', 'list', 'retrieve']:
            return []
        return [IsAuthenticated()]
```

Рисунок 3.18 - Приклад контролю доступу до персональних даних

Система використовує токен-базовану автентифікацію. При вході користувача генерується унікальний токен.

На клієнтській стороні токен зберігається в localStorage та використовується для всіх захищених запитів.

```javascript
// Додавання токену до заголовків запитів
api.interceptors.request.use(config => {
    const token = localStorage.getItem('authToken');
    if (token) {
        config.headers.Authorization = `Token ${token}`;
    }
    return config;
});
```

Рисунок 3.19 - Додавання токену до заголовків запитів

При виході користувача відбувається очищення локального сховища.

```javascript
const logout = useCallback(() => {
    // Видалення токену
    localStorage.removeItem('authToken');
    // Очищення даних користувача в контексті
    setUser(null);
}, []);
```

Рисунок 3.20 - Очищення локального сховища від токена

При видаленні акаунту система виконує повне очищення даних.

На серверній стороні при видаленні акаунту відбувається каскадне видалення всіх пов'язаних даних завдяки налаштуванням моделі.

```python
user = models.OneToOneField(
    User,
    on_delete=models.CASCADE,  # Каскадне видалення
    blank=True,
    null=True,
    verbose_name="Користувач"
)
```

Рисунок 3.21 - Каскадне видалення даних

Система також веде журнал всіх дій з безпеки через ActionLoggingMixin.

Така комплексна система безпеки забезпечує захист персональних даних користувачів, безпечну автентифікацію та авторизацію, контроль доступу до ресурсів, логування дій безпеки, безпечне видалення даних, захист від несанкціонованого доступу, відповідність вимогам захисту персональних даних.

ВИСНОВОК ДО РОЗДІЛУ 3

Упродовж цього розділу було описано процес розробки системи моніторингу ресурсів для волонтерів. Розробка була структурована на п'ять основних підрозділів, кожен з яких супроводжувався детальним описом та релевантними фрагментами коду з реального проекту.

В результаті роботи над програмною частиною було успішно реалізовано повноцінний веб-застосунок з використанням сучасного стеку технологій Django та React. На серверній частині було створено надійну систему управління даними з використанням Django REST Framework, що забезпечує ефективну роботу з базою даних, автентифікацію користувачів та захист персональних даних. Було впроваджено систему логування дій користувачів, що дозволяє відстежувати всі важливі операції в системі.

На клієнтській частині було розроблено інтуїтивно зрозумілий інтерфейс з використанням компонентного підходу React та бібліотеки Material-UI. Реалізовано функціонал реєстрації та авторизації користувачів, управління профілем волонтера, роботу з ресурсами та систему фільтрації і пошуку. Особлива увага була приділена безпеці даних та зручності користування системою.

Важливим досягненням стала реалізація ефективної взаємодії між клієнтською та серверною частинами через REST API, що забезпечує швидку та надійну передачу даних. Використання контекстів React для управління станом значно покращило продуктивність системи.

Реалізований функціонал повністю відповідає поставленим вимогам та забезпечує всі необхідні можливості для ефективної роботи волонтерів з ресурсами.

РОЗДІЛ 4. ТЕСТУВАННННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

Реалізацією системи моніторингу ресурсів є веб-додаток, який зустрічає користувача банером «Ласкаво просимо до системи моніторингу ресурсів!». Це головна сторінка, на якій можна дізнатися, про що саме веб-додаток.

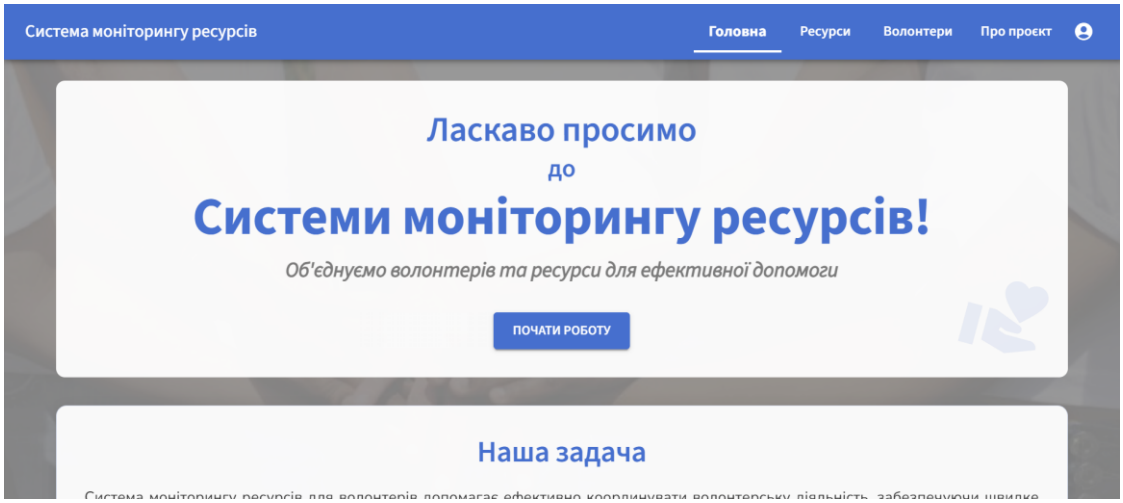


Рисунок 4.1 – Головна сторінка

Якщо прогорнути нижче, то можна дізнатися функціонал додатка, а саме те, що цей додаток про моніторинг ресурсів, базу волонтерів та категорії ресурсів.

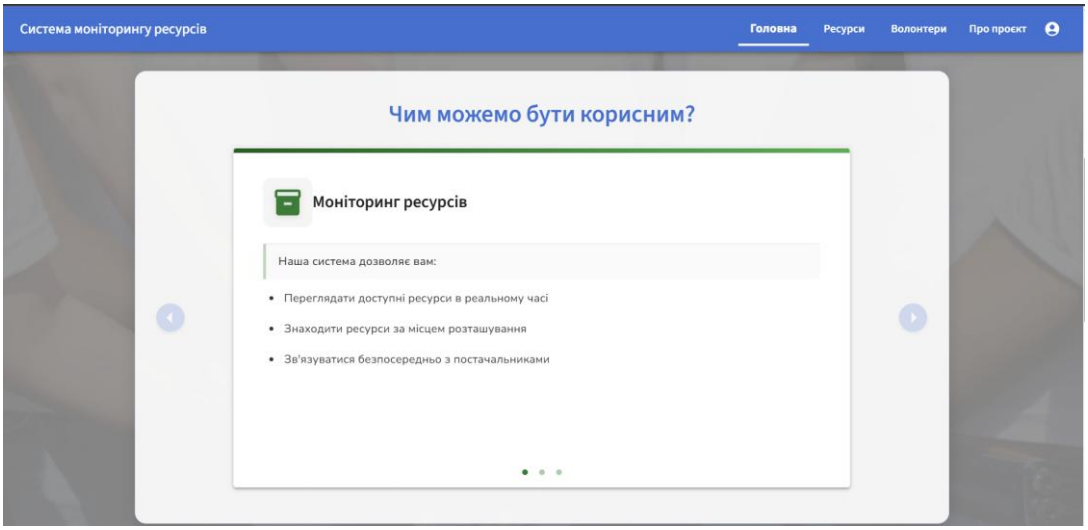


Рисунок 4.2 – Розділ «Чим можемо бути корисним?»

В самому кінці маємо розділ «Приєднуйтесь до нас», де пропонують зареєструватися волонтером та зробити свій вклад в розвиток суспільства.

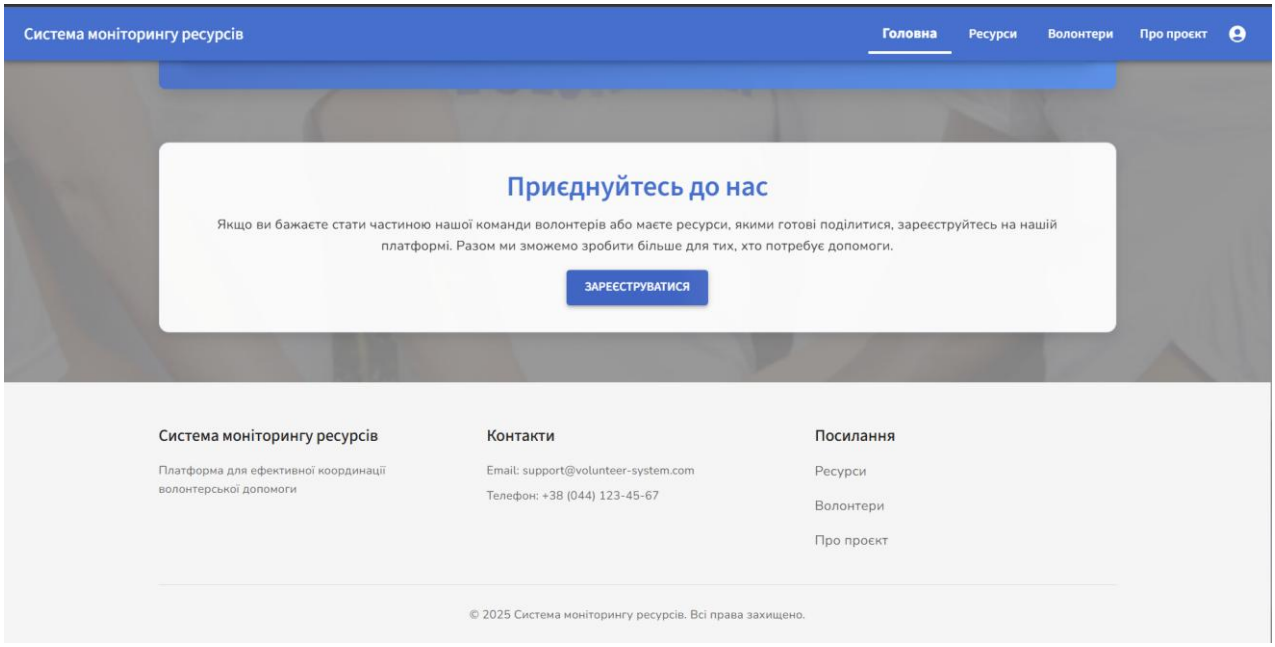


Рисунок 4.3 – Розділ «Приєднуйтесь до нас»

Перейдемо на вкладку «Ресурси». Тут маємо пошукову стрічку з різними фільтрами. Без фільтрів підвантажується з бази даних 9 видів ресурсів. Картки ресурсів мають коротку інформацію про себе, а саме назва, категорія, локація, опис та доступна кількість.

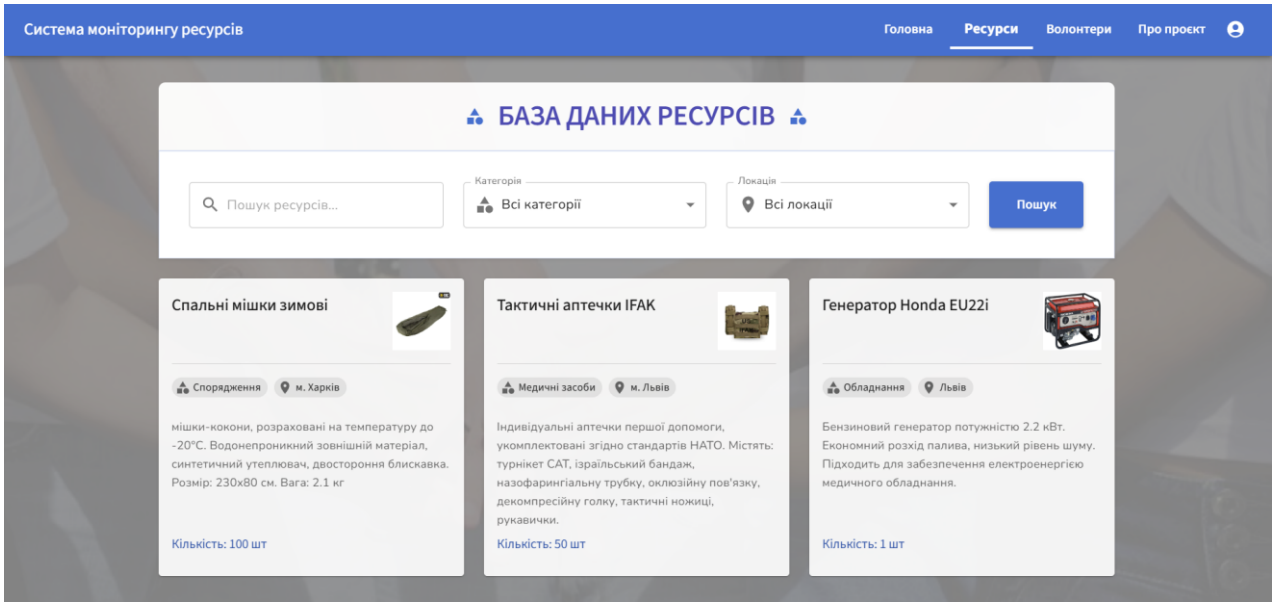


Рисунок 4.4 – Вкладка «Ресурси» при неавторизованому користувачеві

Можна обрати будь-який фільтр і натиснути «Пошук». Як приклад, оберемо категорії «продукти» та локацію м. Дніпро.

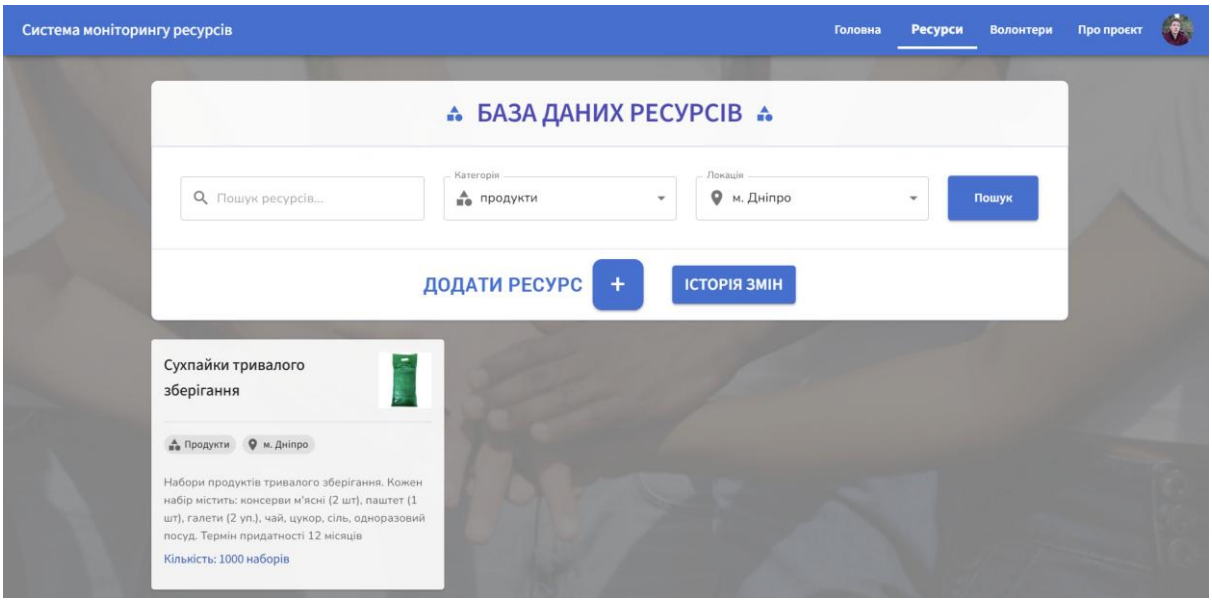


Рисунок 4.5 – Вкладка «Ресурси» з фільтрами

Якщо треба дізнатися більш детальну інформацію про ресурс, то для цього треба натиснути на відповідну картку ресурса. Тоді користувача перенаправляють на сторінку з детальною інформацією про ресурс. Тепер можна дізнатися, чи доступний ресурс зараз, хто створив інформацію про цей ресурс та коли це було зроблено

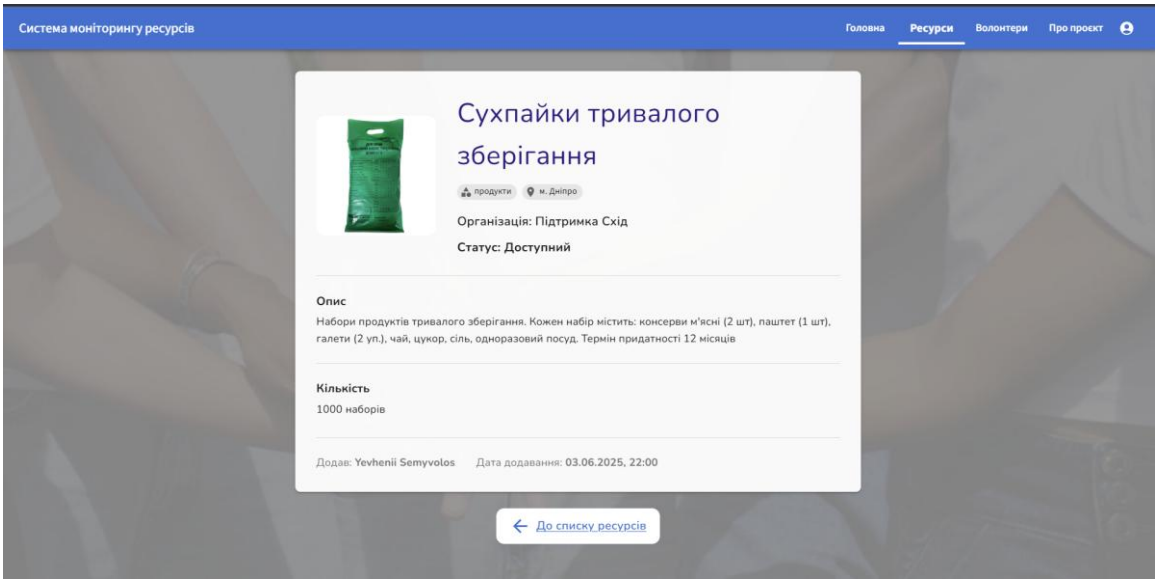


Рисунок 4.6 – Детальна інформація про ресурс

Тепер розглянемо вкладку «Волонтери». Тут будь-який користувач має доступ до публічних даних волонтерів. На головній сторінці цієї вкладки є пошукова стріка та перші 6 підвантажених волонтерів. На картках волонтера є така інформація як ПІБ, Організація, Навички та Опис волонтера.

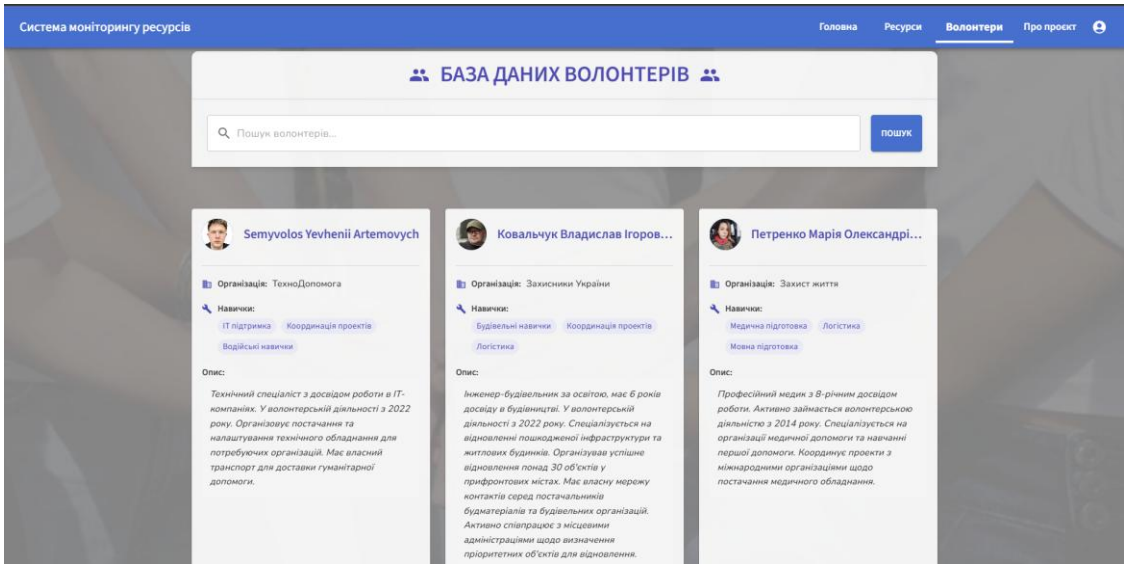


Рисунок 4.7 – Вкладка «Волонтери»

Можна використовувати пошук. Як приклад, давайте знайдемо волонтера з іменем Владислав.

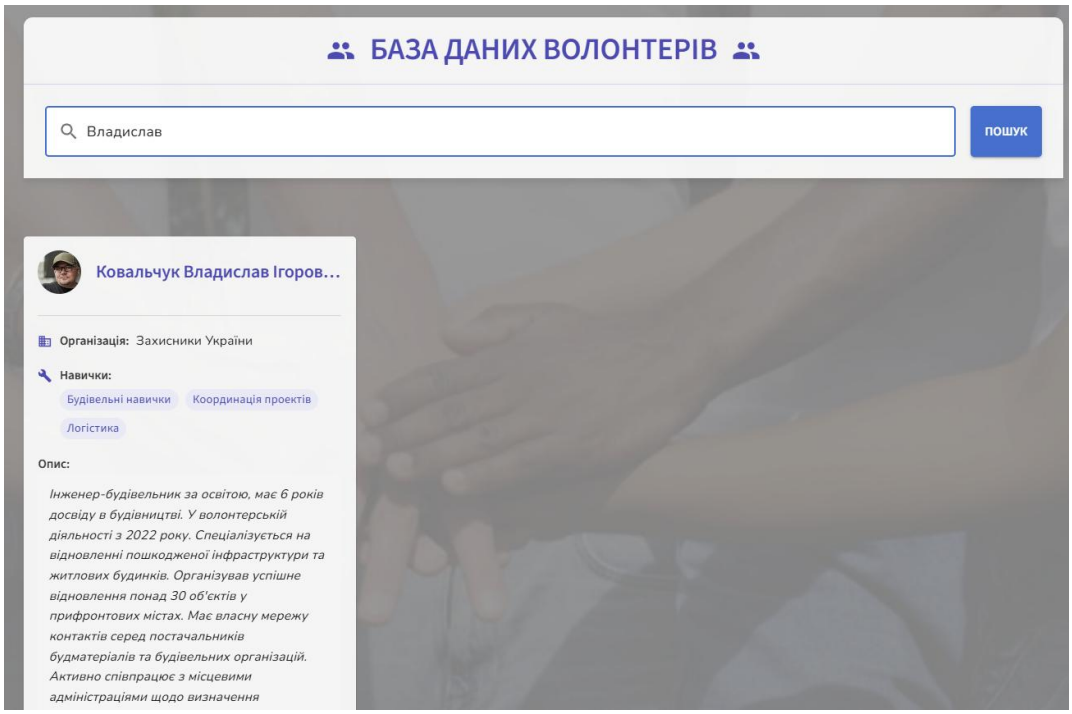


Рисунок 4.8 – Вкладка «Волонтери» з фільтром «Владислав»

Можна знайти волонтера за його навичками. Для цього достатньо вписати необхідні навички в поле «Пошук».

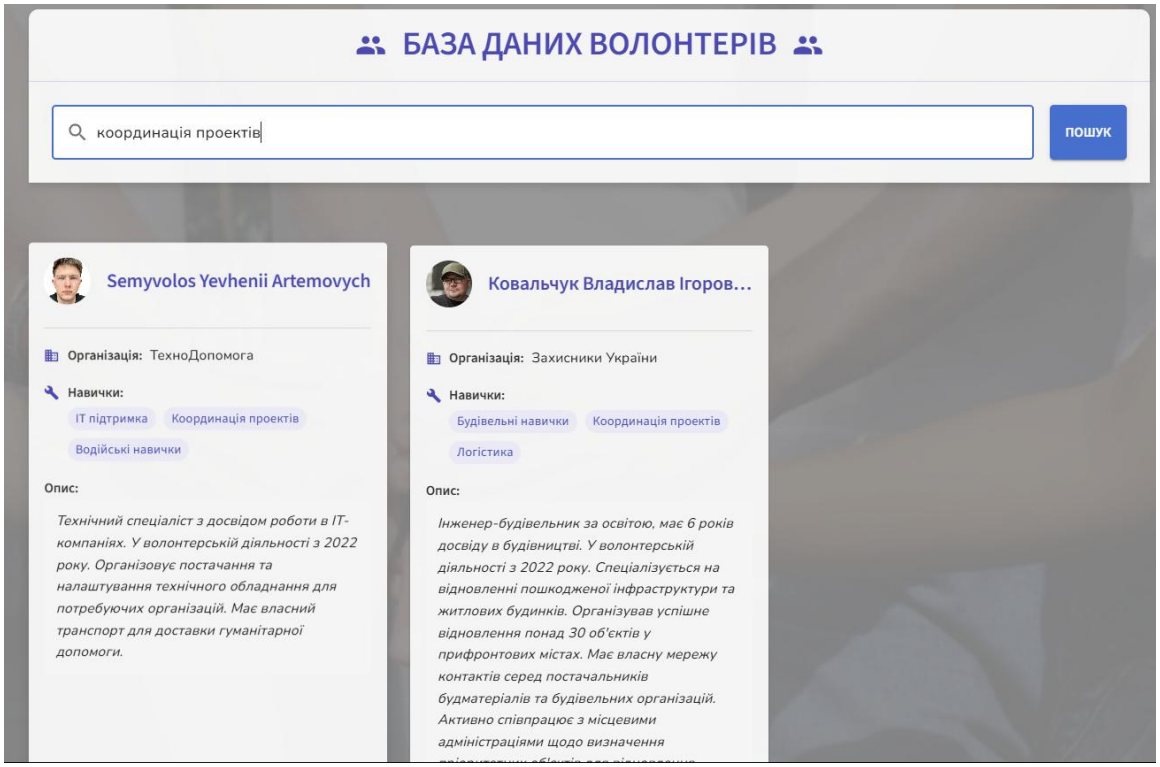


Рисунок 4.9 – Вкладка «Волонтери» з фільтром «координація проектів»

Якщо треба дізнатися більш детальну інформацію про волонтера, необхідно натиснути на відповідну картку волонтера.

На вкладці «Про проект» можна дізнатися про місію та цілі цієї системи.

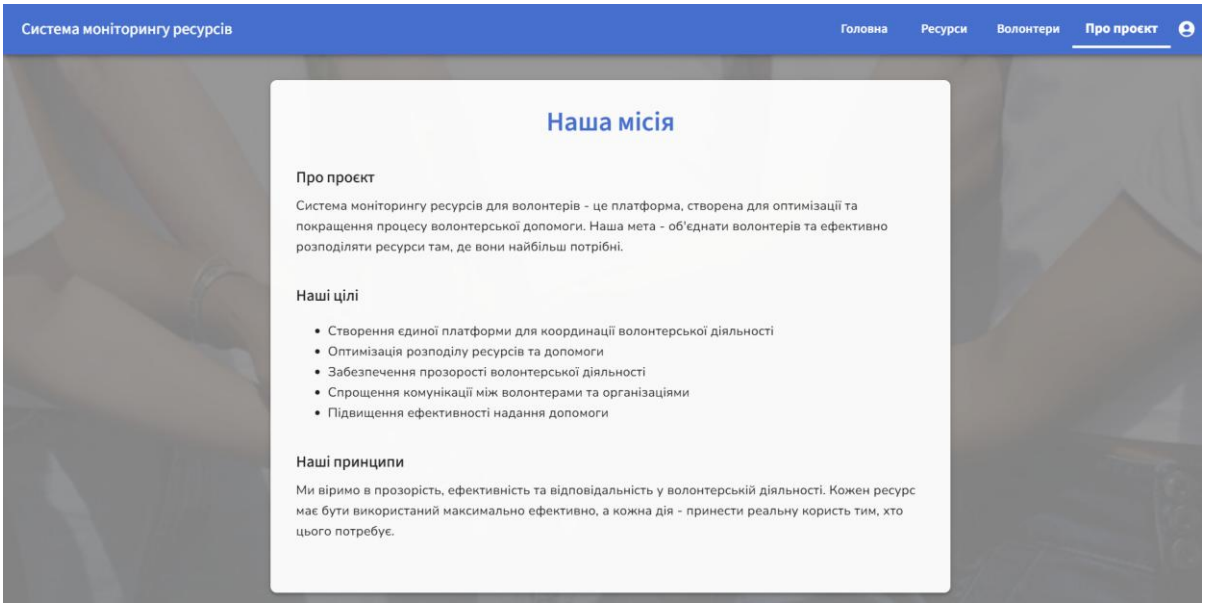


Рисунок 4.10 – Вкладка «Про проект»

Коли користувач вперше відвідує сайт, то він робіть всі дії як незареєстрований волонтер, тобто як гість. Якщо користувач буде авторизованим, то в нього з’являться додаткові можливості, а саме редагування власного профілю та вносити зміни в інформацію про ресурси.

Для реєстрації нового волонтера необхідно натиснути на іконку аватара в правому верхньому кутку меню навігації. Після цього на екрані користувача з’явиться модальне вікно, яке запропонує авторизуватись.

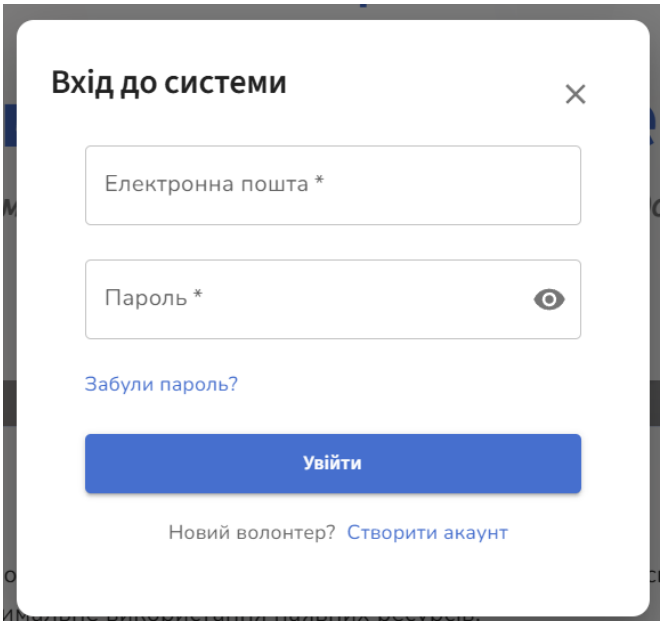


Рисунок 4.11 – Модальне вікно «Вхід до системи»

Так як користувач буде реєструватись, необхідно натиснути на посилання «Створити акаунт», після чого відкриється сторінка реєстрації.

При реєстрації необхідно вказати такі параметри як ПІБ, номер телефону, електрону пошту, Телеграм ID(опціонально), навички, опис себе (опціонально), організація, пароль та повтроний пароль. Також є можливість завантажити власний аватар, але ця опція не є обов'язковою.

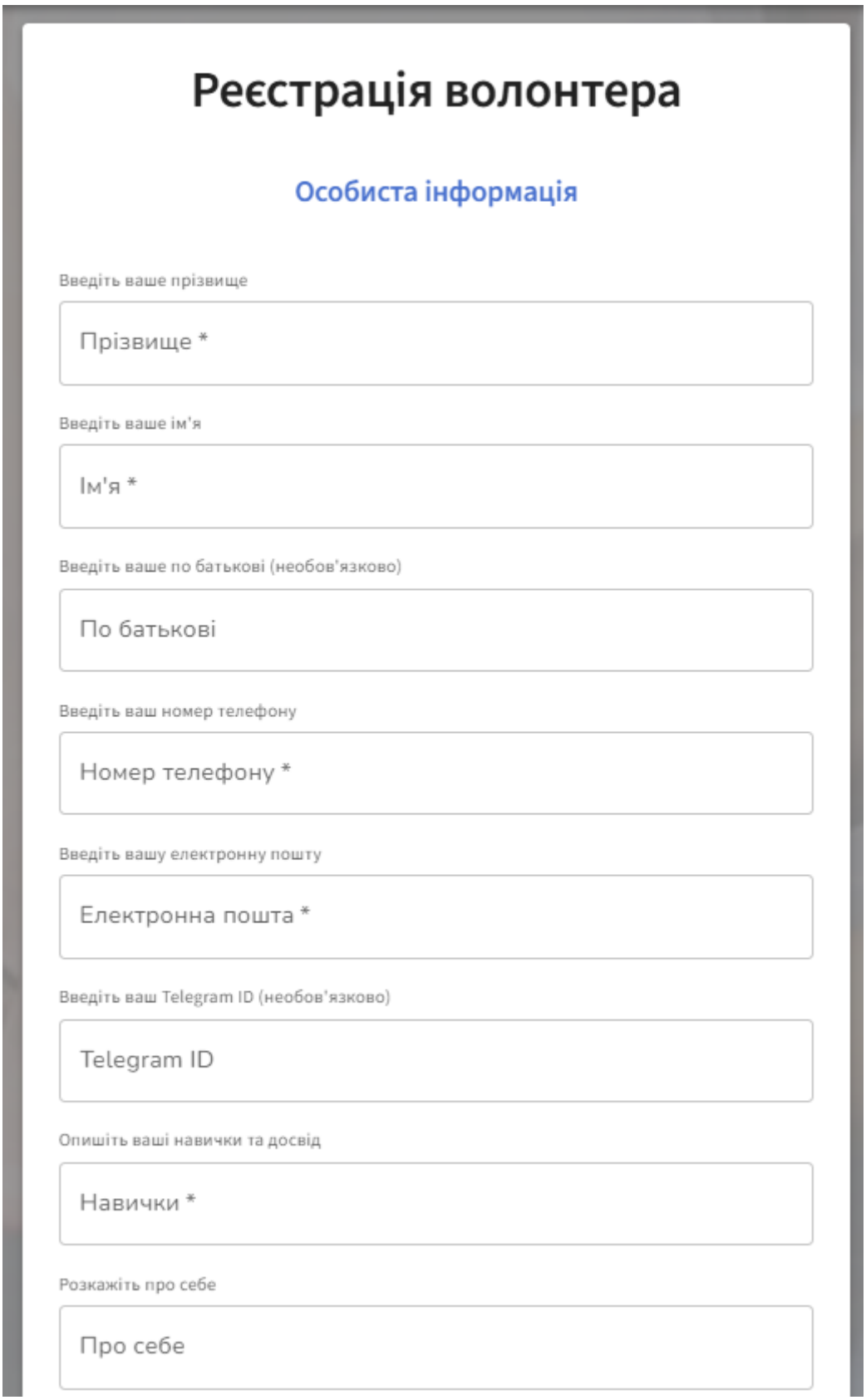


Рисунок 4.12 – Сторінка реєстрації

В реєстрації є валідація даних. Якщо буде вказана неправильна пошта або пошта вже є в використанні, то користувач не зможе зареєструватись.

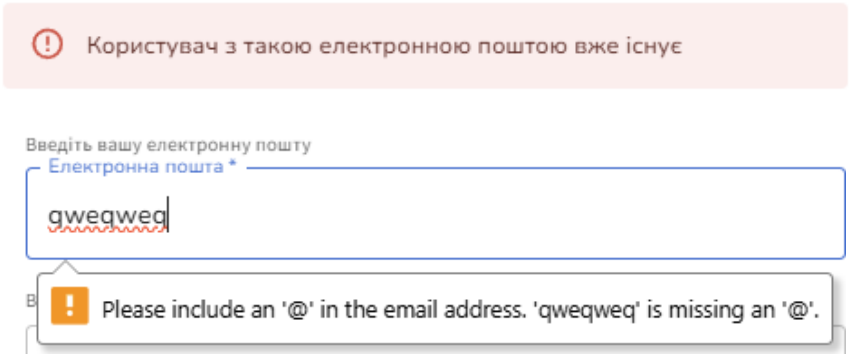


Рисунок 4.13 – Приклади помилок при введенні електронної пошти

Так само реєстрація неможлива, якщо введено неправильний повтор паролю, або пароль не відповідає вимогам.

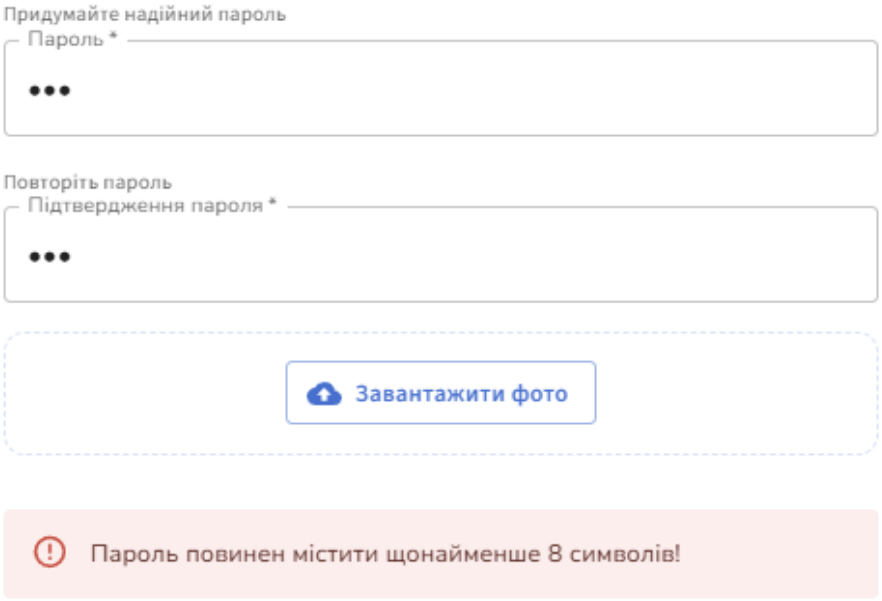


Рисунок 4.14 – Приклад вводу слабкого паролю

Після реєстрації користувача перенаправляють на головну сторінку, на якій є повідомлення, що реєстрація пройшла успішно, і треба чекати підтвердження від адміністратора.

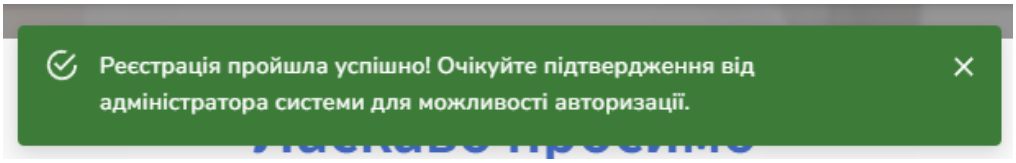


Рисунок 4.15 – Повідомлення про успішну реєстрацію

Якщо спробувати авторизуватись, коли адміністратор не підтвердив користувача, то буде повідомлення «Адміністратор ще не підтвердив вас»

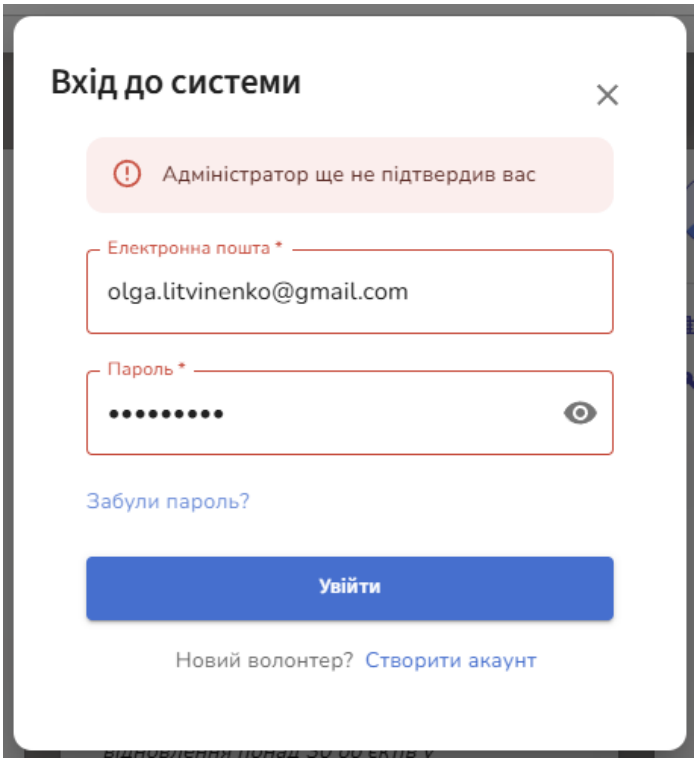


Рисунок 4.16 – Невдала спроба авторизації через не підтвердження адміном

Якщо адміністратор підтвердить нового волонтера, то буде можливість авторизуватись. Після авторизації повинен змінитись аватар в правому верхньому куту, якщо при реєстрації волонтер прикріпив фото.

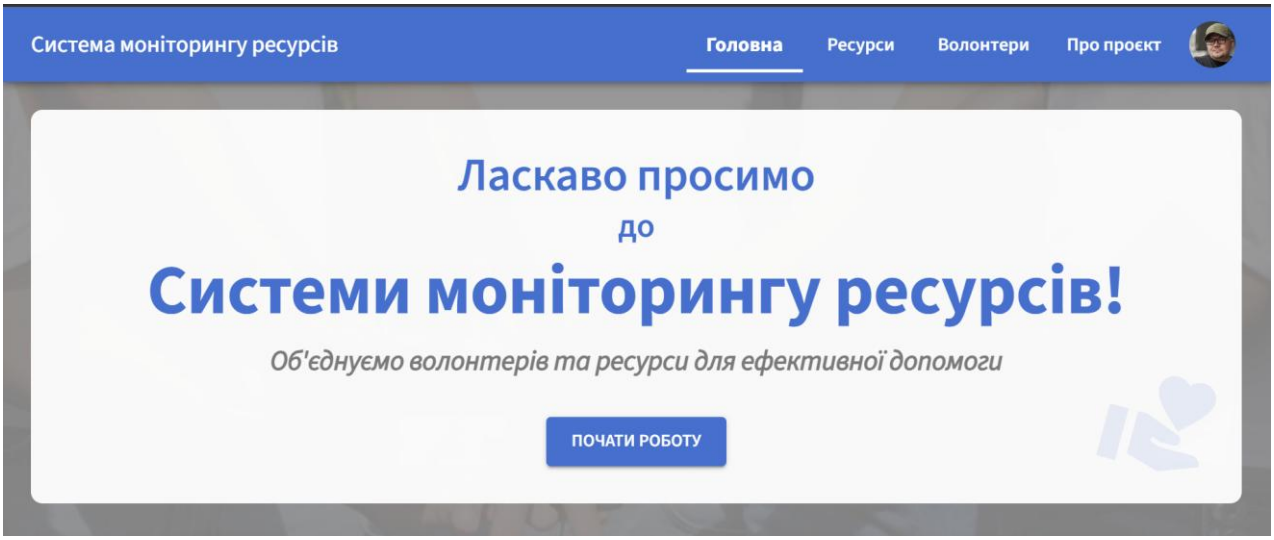


Рисунок 4.17 – Успішне автентифікація

Можна переглянути профіль користувача.

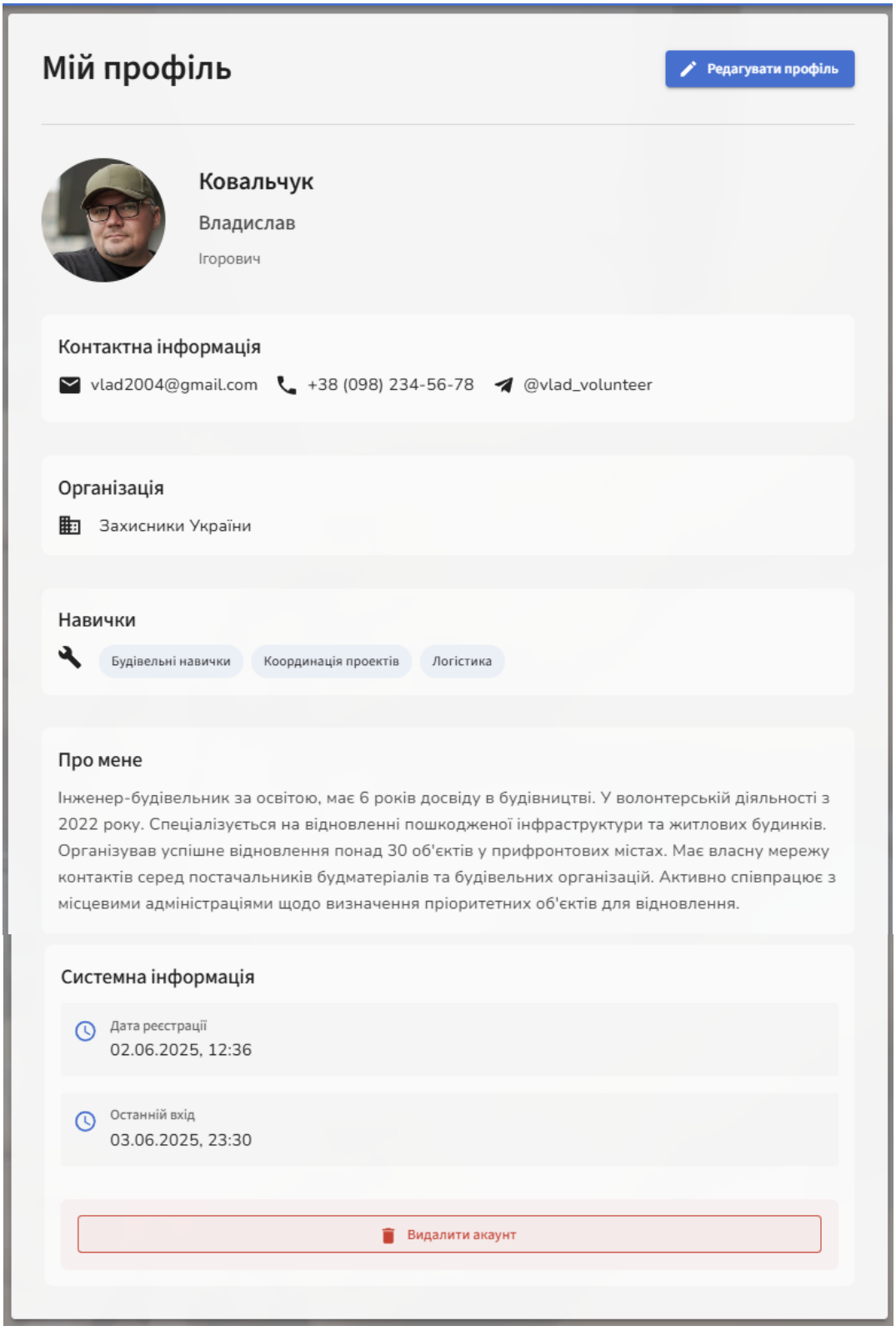


Рисунок 4.18 – Профіль користувача

Якщо в процесі реєстрації була вказана якась хибна інформація, її можна відредагувати, натиснувши кнопку «Редагувати профіль»

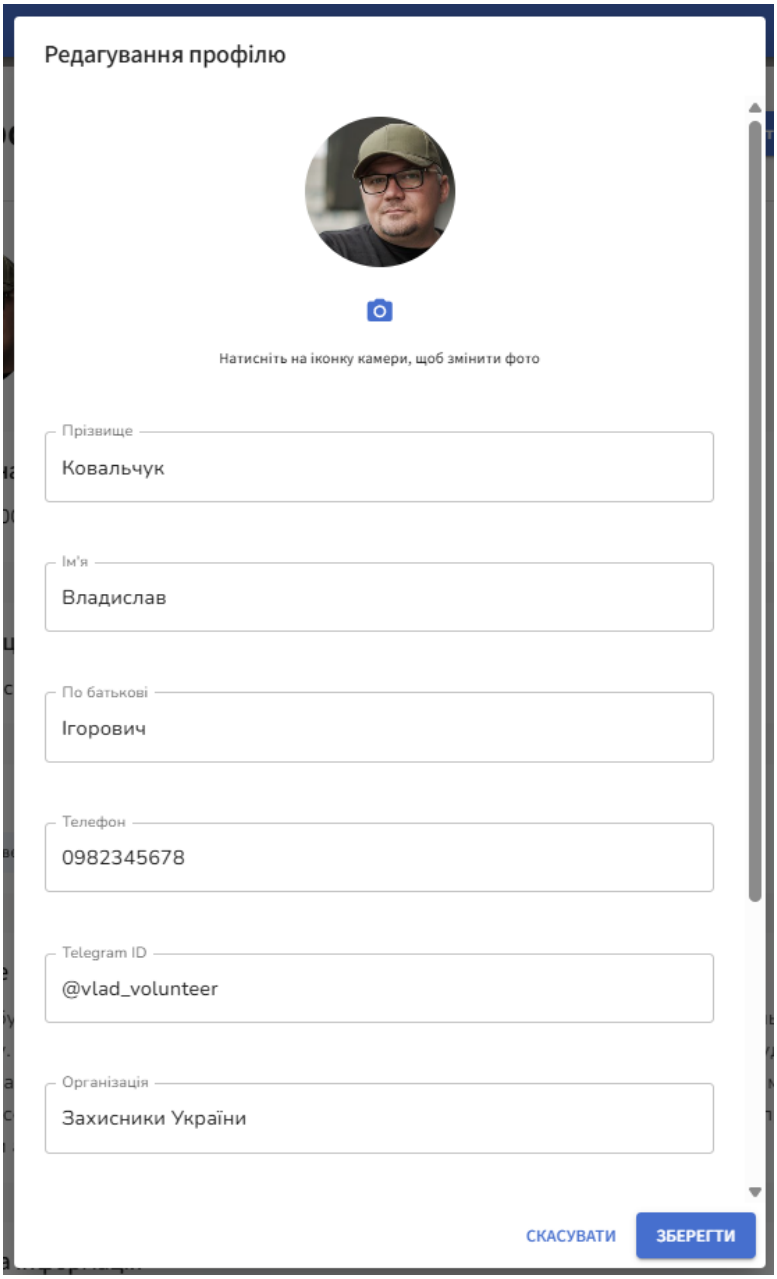


Рисунок 4.19 – Редагування профілю користувача

Для авторизованого користувача на вкладці «Ресурси» доступні нові кнопки, а саме «Додати ресурс» та «Історія змін».

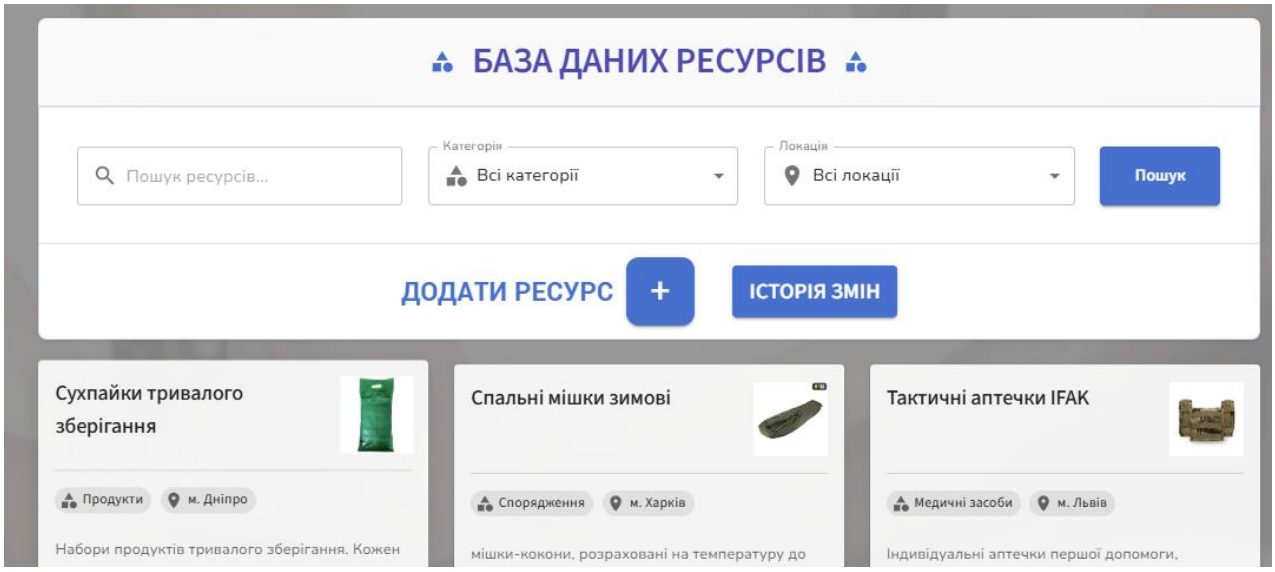


Рисунок 4.20 – Нові кнопки «Додати ресурс» та «Історія змін»

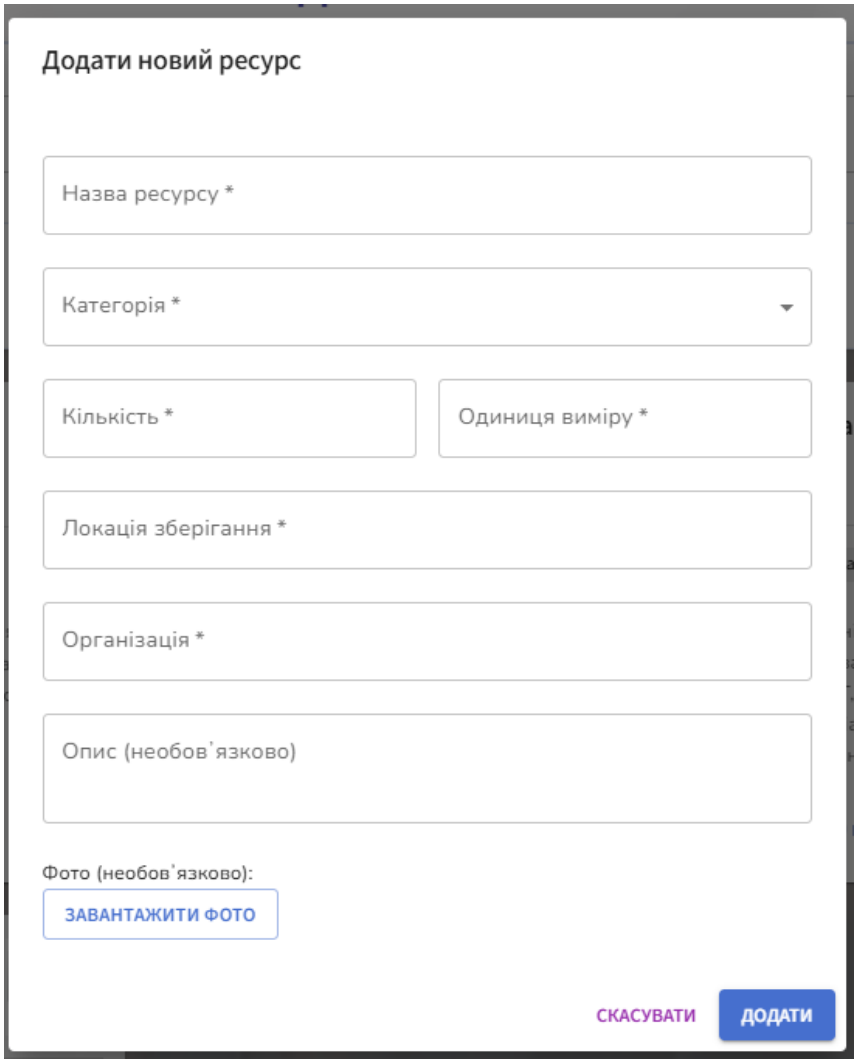


Рисунок 4.21 – Форма для заповнення інформації про новий ресурс

Також для авторизованого волонтера доступна функція змінити кількість будь-якого ресурса.

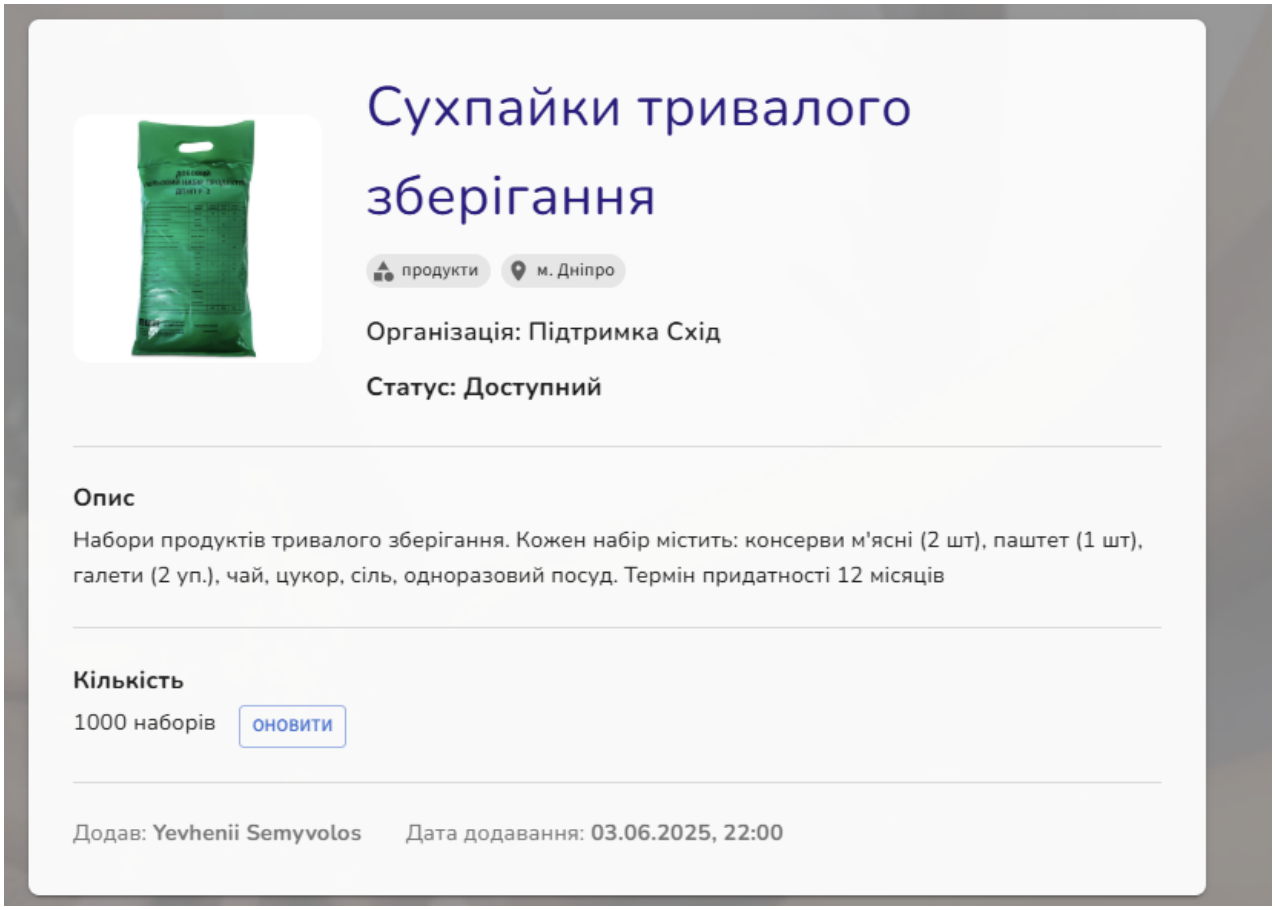


Рисунок 4.22 – Кнопка «оновити» для кількості ресурсу

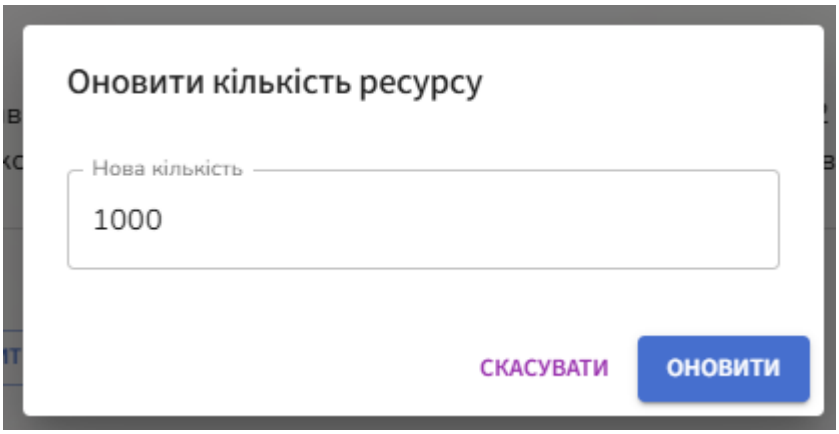


Рисунок 4.23 – Модальне вікно для оновлення кількості ресурсу

Якщо кількість оновити на значення 0, ресурс видаляється з бази даних.

При натисканні на кнопку «Історія змін» відкриється таблиця подій, які відбувалися у системі. Тут є можливість фільтрувати дії у системі за типом об’єкту (Волонтер чи Ресурс), за типом дії та за датою.

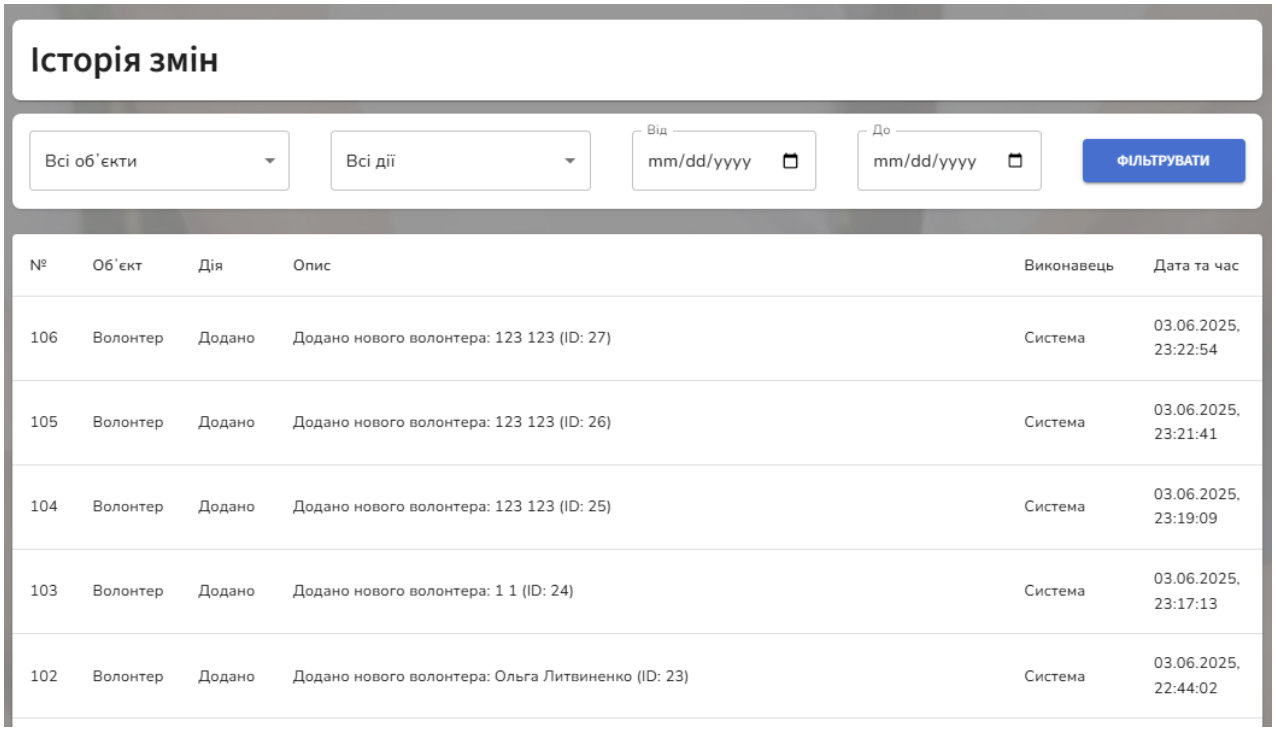


Рисунок 4.24 – Вкладка «Історія змін»

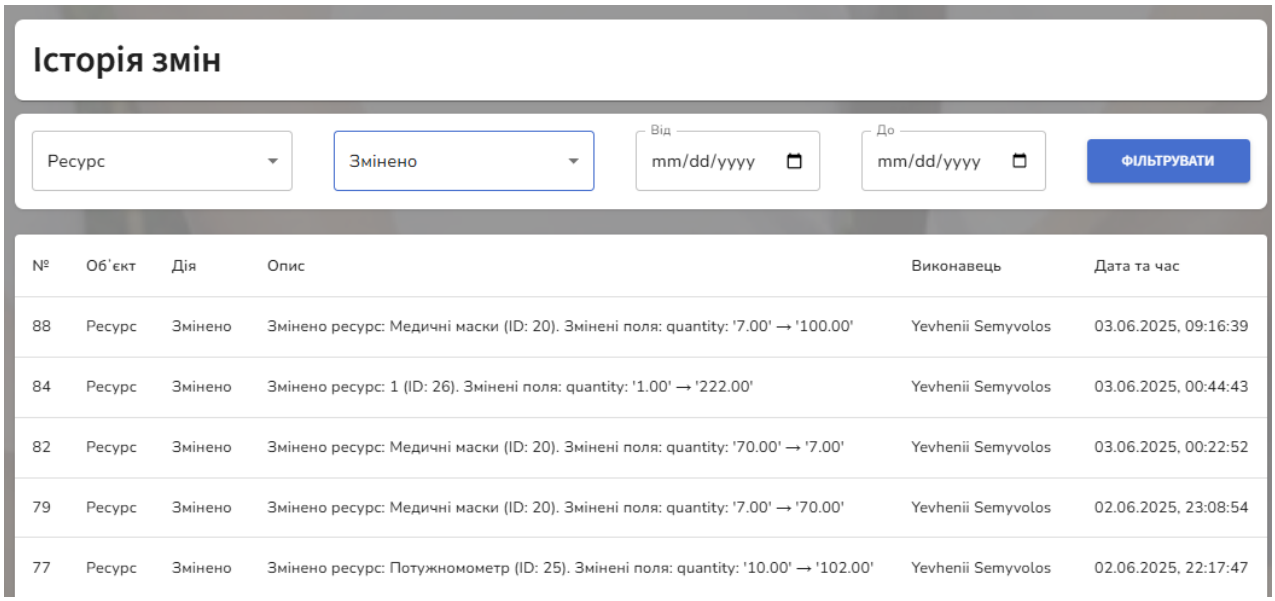


Рисунок 4.24 – Вкладка «Історія змін» з фільтрами

ВИСНОВОК ДО РОЗДІЛУ 4

Цей розділ був присвячений огляду та тестуванню розробленої системи моніторингу ресурсів для волонтерів. Для детального ознайомлення було представлено опис основних сторінок системи: головна сторінка, сторінка ресурсів, профіль волонтера, сторінка реєстрації та авторизації, а також сторінка історії дій. Було продемонстровано роботу ключових функцій системи та перевірено їх коректне виконання.

Серед основних можливостей системи можна виділити:

- реєстрація та авторизація волонтерів з підтвердженням адміністратором;
- пошук ресурсів за категоріями та локацією;
- сортування та фільтрація ресурсів;
- управління профілем волонтера
- додавання та редагування інформації про ресурси;
- перегляд історії дій з ресурсами;

Особлива увага була приділена безпеці системи та захисту персональних даних волонтерів. Реалізовано багаторівневу систему автентифікації та авторизації, а також механізми валідації даних на клієнтській та серверній частинах.

Інтерфейс системи розроблено з урахуванням сучасних принципів UI/UX дизайну, що забезпечує інтуїтивно зрозумілу навігацію та комфортну роботу користувачів. Використання React та Material-UI дозволило створити адаптивний інтерфейс, який коректно відображається на різних пристроях.

Таким чином, було повністю протестовано систему та продемонстровано її функціональність. Програмний комплекс працює стабільно, забезпечує надійне зберігання даних та виконує всі заплановані функції, необхідні для ефективної координації волонтерської діяльності та управління ресурсами.

ВИСНОВКИ

Результатом виконання даного бакалаврського проєкту є система моніторингу ресурсів для волонтерів, яка значно полегшує процес координації та управління волонтерською діяльністю.

У першому розділі було проведено детальний аналіз існуючих систем управління волонтерськими ресурсами та їх обмежень. Було розглянуто такі системи як CiviCRM, Better Impact та Odoo, визначено їх переваги та недоліки. На основі цього аналізу було прийнято рішення про розробку власної системи з використанням REST API архітектури, що дозволяє досягти необхідної гнучкості та масштабованості.

У другому розділі було проведено ґрунтовний аналіз сучасних технологій для розробки веб-застосунків. Для клієнтської частини було обрано React.js через його компонентний підхід та багату екосистему, включаючи Material-UI для створення сучасного користувацького інтерфейсу. Для серверної частини було обрано Django через його потужні вбудовані можливості безпеки, ORM та зручність створення REST API. Використання TypeScript забезпечило додатковий рівень типізації та надійності коду.

У третьому розділі було описано процес розробки системи. Було реалізовано:

- Систему реєстрації та авторизації з підтвердженням адміністратором
- Управління профілями волонтерів
- Функціонал додавання та редагування ресурсів
- Систему пошуку та фільтрації ресурсів
- Історію дій користувачів

Особлива увага була приділена безпеці даних та валідації користувацького вводу, включаючи перевірку форматів посилань та банківських реквізитів.

У четвертому розділі було проведено всебічне тестування системи. Було перевірено:

- Коректність роботи всіх функціональних модулів
- Безпеку та захист даних
- Валідацію користувацького вводу
- Адаптивність інтерфейсу
- Стабільність роботи системи під навантаженням

Розроблена система успішно вирішує поставлені задачі та надає зручний інструмент для координації волонтерської діяльності. Система є особливо актуальною в сучасних умовах, коли ефективна організація волонтерської допомоги має критичне значення. Завдяки використанню сучасних технологій та методів розробки, система є надійною, безпечною та зручною у використанні.

Важливо відзначити, що система має потенціал для подальшого розвитку та масштабування, зокрема можливе додавання нових функцій для аналітики, розширення категорій ресурсів та інтеграції з іншими платформами. Таким чином, розроблений проєкт не тільки вирішує поточні потреби волонтерів, але й створює основу для подальшого розвитку та вдосконалення системи координації волонтерської діяльності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Про волонтерську діяльність [Електронний ресурс] – Режим доступу до ресурсу: https://www.dsp.gov.ua/pro-volontersku-diyalnist/
2. Розділ "Ефективність управління ресурсами в НУО" [Електронний ресурс]. Режим доступу до ресурсу – https://www.prostir.ua/research
3. Офіційний гайд Salesforce про CRM [Електронний ресурс] – Режим доступу до ресурсу: https://www.salesforce.com/crm/what-is-crm/
4. Офіційний опис SAP ERP [Електронний ресурс] – Режим доступу до ресурсу: https://www.sap.com/ukraine/products/erp/what-is-erp.html
5. What is CiviCRM [Електронний ресурс] – Режим доступу до ресурсу: https://docs.civicrm.org/user/en/latest/introduction/what-is-civicrm/
6. Офіційний сайт BETTER IMPACT [Електронний ресурс] – Режим доступу до ресурсу: https://www.betterimpact.com/
7. Офіційний сайт ODOO [Електронний ресурс] – Режим доступу до ресурсу: https://www.odoo.com/ru_RU
8. AngularJS Official Documentation (v1.8) [Електронний ресурс] – Режим доступу до ресурсу: https://angularjs.org/
9. Що таке двостороннє прив’язування даних в Angular? [Електронний ресурс] – Режим доступу до ресурсу: https://aleia.mercy.cx.ua/maysternist/shho-take-dvostoronnie-priv-yazuvannya-danikh-v-angular.html
10. Virtual DOM and Internals [Електронний ресурс] – Режим доступу до ресурсу: https://legacy.reactjs.org/docs/faq-internals.html
11. The Node.js Event Loop [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/en/learn/asynchronous-work/event-loop-timers-and-nexttick

12. Що таке Java Spring Boot? [Електронний ресурс] – Режим доступу до ресурсу: https://azure.microsoft.com/ru-ru/resources/cloud-computing-dictionary/what-is-java-spring-boot
13. Material UI – Overview [Електронний ресурс] – Режим доступу до ресурсу: https://mui.com/material-ui/getting-started/
14. Django REST framework Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.django-rest-framework.org/
15. React – A JavaScript library for building user interfaces [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/
16. Authentication with Django REST Framework [Електронний ресурс] – Режим доступу до ресурсу: https://django-rest-framework-simplejwt.readthedocs.io/
17. Axios HTTP Client [Електронний ресурс] – Режим доступу до ресурсу: https://axios-http.com/docs/intro
18. Web Security Best Practices [Електронний ресурс] – Режим доступу до ресурсу: https://owasp.org/www-project-web-security-testing-guide/
19. React Context API [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/reference/react/useContext
20. PostgreSQL 17.5 Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/docs/current/index.html
21. Database Design Best Practices [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/docs/current/tutorial.html

ДОДАТОК А

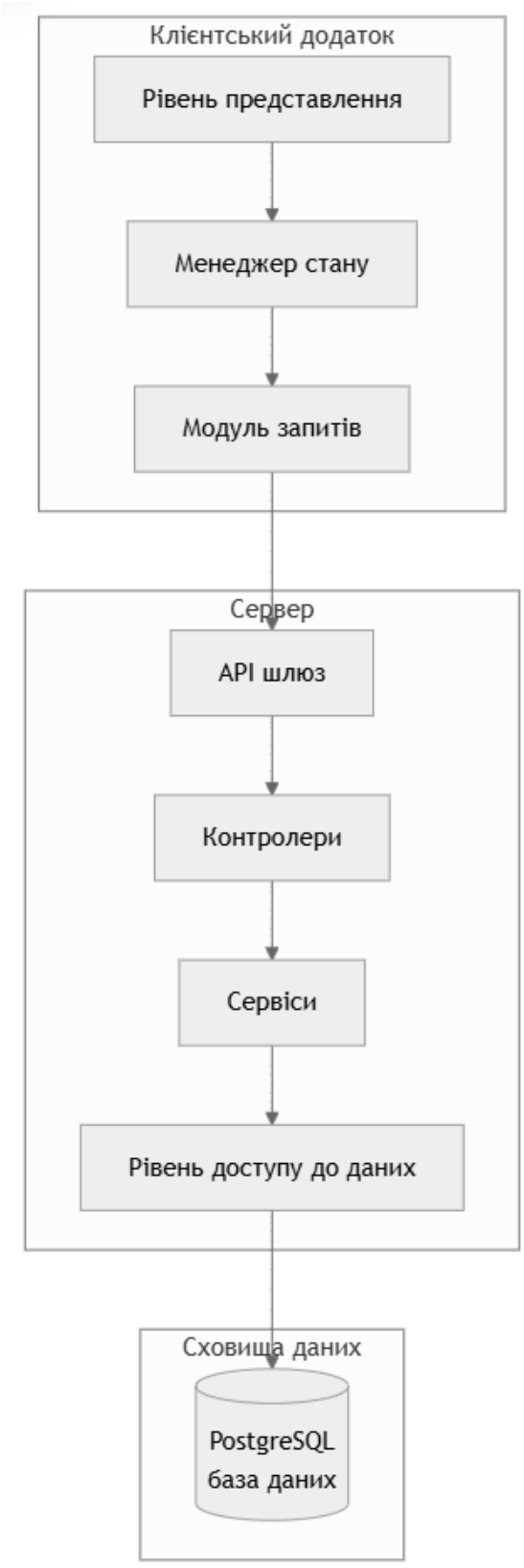
Система моніторингу ресурсів для громадських організацій та волонтерів

Структурна схема системи (структурна схема)

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.004 Д1		
		№ докум.	Підпис	Дата			
Розробив		Семиволос Є. А.			**Система моніторингу ресурсів для громадських організацій та волонтерів** Структурна схема системи	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Павлов В. Г.					
Реценз.		Корнієнко Б. Я.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-11	
Н. Контр.		Нікольський С. С.					
Затвердив		Новотарський М. А.					

ДОДАТОК Б

Система моніторингу ресурсів для громадських організацій та волонтерів

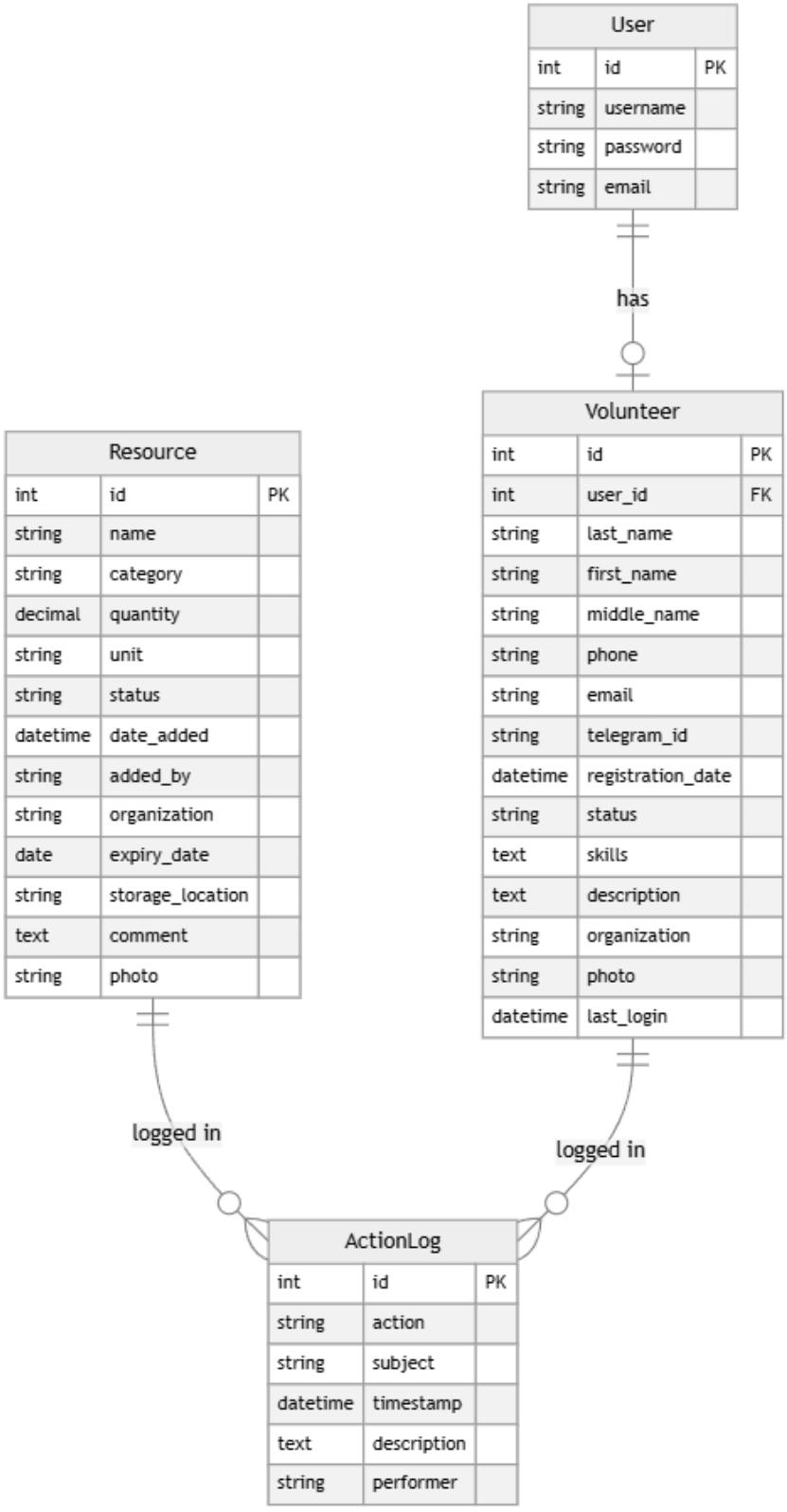
ER діаграма бази даних

(функціональна схема)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.005 Д2
		№ докум.	Підпис	Дата	
Розробив		Семиволос Є. А.			**Система моніторингу ресурсів для громадських організацій та волонтерів**
Перевірив		Павлов В. Г.			ER діаграма бази даних
Реценз.		Корнієнко Б. Я.			
Н. Контр.		Нікольський С. С.			
Затвердив		Новотарський М. А.			

Літ.	Аркуш	Аркушів
	1	1

НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-11

ДОДАТОК В

Система моніторингу ресурсів для громадських організацій та волонтерів

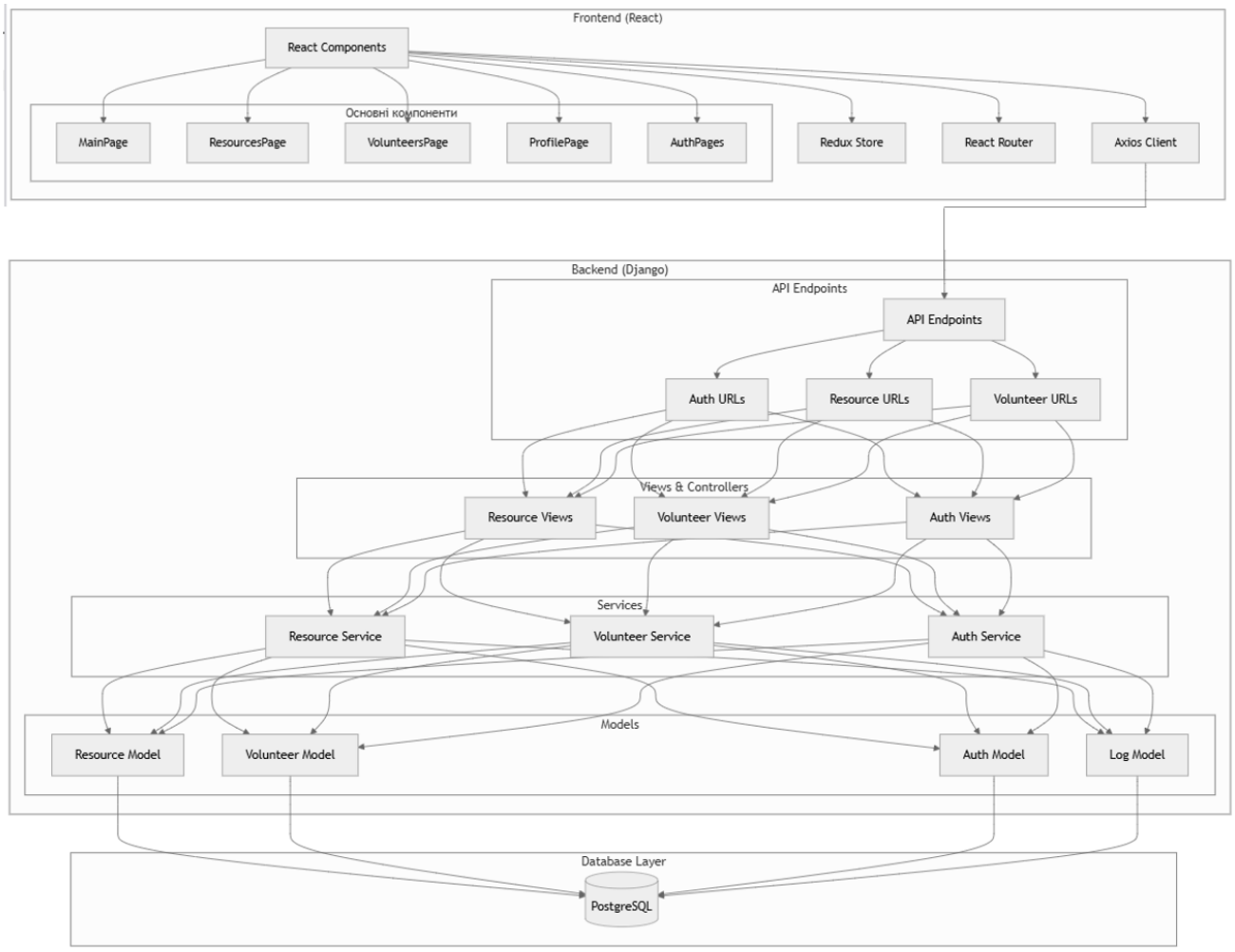
Архітектурна схема додатку

(принципова схема)

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.006 Д3		
		№ докум.	Підпис	Дата			
Розробив		Семиволос Є. А.			**Система моніторингу ресурсів для громадських організацій та волонтерів** Архітектурна схема додатку	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Павлов В. Г.					
Реценз.		Корнієнко Б. Я.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-11	
Н. Контр.		Нікольський С. С.					
Затвердив		Новотарський М. А.					

ДОДАТОК Г

Система моніторингу ресурсів для громадських організацій та волонтерів

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 64

Київ 2025 р

GitHub репозиторій:

https://github.com/EugeneSemivolos/resource-monitoring-system-for-volunteers

Серверна частина

manage.py

```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys
def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'app.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)
if __name__ == '__main__':
    main()
```

app/settings.py

```
from pathlib import Path
# Побудова шляхів всередині проекту таким чином: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-vr40--!u5o_1f7^_35cmwiq8r7(!17m*%n27jwv8_9_s0z%p1u'
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
ALLOWED_HOSTS = ['*']
# Application definition
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
```

```
    'django.contrib.staticfiles',
    'api',
    'rest_framework',
    'rest_framework.authtoken',
    'corsheaders',
    'django_filters',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'app.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'app.wsgi.application'


# База даних
# https://docs.djangoproject.com/en/5.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'volunteer_resources',
        'USER': 'postgres',
        'PASSWORD': '5454',
        'HOST': 'localhost',
        'PORT': '5432',
```

```
    }
}


# Валідація паролів
# https://docs.djangoproject.com/en/5.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Інтернаціоналізація
# https://docs.djangoproject.com/en/5.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True


# Статичні файли (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.2/howto/static-files/

STATIC_URL = 'static/'
STATIC_ROOT = BASE_DIR / 'static'

# Media файли
MEDIA_URL = '/media/'
MEDIA_ROOT = BASE_DIR / 'media'

# Тип поля первинного ключа за замовчуванням
# https://docs.djangoproject.com/en/5.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

```
# Налаштування CORS
CORS_ALLOW_ALL_ORIGINS = True

# Налаштування Django REST Framework
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 10,
    'DEFAULT_FILTER_BACKENDS': ['django_filters.rest_framework.DjangoFilterBackend'],
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.TokenAuthentication',
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework.authentication.BasicAuthentication',
    ],
}

# Налаштування кешування
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
        'LOCATION': 'unique-snowflake',
        'TIMEOUT': 300,  # 5 хвилин
    }
}
```

app/urls.py

```
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('api.urls')),
]

# Додаємо обслуговування медіа-файлів у режимі розробки
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

api/admin.py

```
from django.contrib import admin
from .models import Resource, Volunteer, ActionLog

# Register your models here.
@admin.register(Resource)
class ResourceAdmin(admin.ModelAdmin):
```

```
    list_display = ('name', 'category', 'quantity', 'unit', 'status', 'date_added',
'added_by', 'storage_location')
    list_filter = ('status', 'category', 'organization')
    search_fields = ('name', 'category', 'added_by', 'organization',
'storage_location')
    date_hierarchy = 'date_added'
    fieldsets = (
        ('Основна інформація', {
            'fields': ('name', 'category', 'quantity', 'unit', 'status', 'photo')
        }),
        ('Деталі зберігання', {
            'fields': ('storage_location', 'organization', 'expiry_date')
        }),
        ('Додаткова інформація', {
            'fields': ('comment',)
        }),
        ('Службова інформація', {
            'fields': ('added_by', 'date_added'),
        }),
    )

@admin.register(Volunteer)
class VolunteerAdmin(admin.ModelAdmin):
    list_display = ('last_name', 'first_name', 'email', 'phone', 'status',
'organization', 'registration_date', 'last_login')
    list_filter = ('status', 'registration_date', 'organization')
    search_fields = ('last_name', 'first_name', 'email', 'phone', 'telegram_id',
'skills', 'organization', 'description')
    date_hierarchy = 'registration_date'
    fieldsets = (
        ('Особиста інформація', {
            'fields': ('last_name', 'first_name', 'middle_name', 'photo')
        }),
        ('Контактна інформація', {
            'fields': ('phone', 'email', 'telegram_id')
        }),
        ('Додаткова інформація', {
            'fields': ('status', 'skills', 'description', 'organization')
        }),
        ('Системна інформація', {
            'fields': ('user', 'registration_date', 'last_login'),
            'classes': ('collapse',)
        }),
    )

@admin.register(ActionLog)
class ActionLogAdmin(admin.ModelAdmin):
    list_display = ('id', 'action', 'subject', 'performer', 'timestamp')
    list_filter = ('action', 'subject')
    search_fields = ('action', 'subject', 'performer', 'description')
```

```
    date_hierarchy = 'timestamp'
```

api/apps.py

```
from django.apps import AppConfig


class ApiConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'api'
```

api/models.py

```
from django.db import models
from django.utils import timezone
from django.contrib.auth.models import User

class Resource(models.Model):
    STATUS_CHOICES = [
        ('available', 'Доступний'),
        ('reserved', 'Зарезервований'),
        ('depleted', 'Вичерпаний'),
    ]

    CATEGORY_CHOICES = [
        ('медичні засоби', 'Медичні засоби'),
        ('спорядження', 'Спорядження'),
        ('продукти', 'Продукти'),
        ('обладнання', 'Обладнання'),
        ('одяг', 'Одяг'),
        ('інше', 'Інше'),
    ]

    name = models.CharField(max_length=255, verbose_name="Назва", db_index=True)
    category = models.CharField(max_length=100, choices=CATEGORY_CHOICES,
default='інше', verbose_name="Категорія", db_index=True)
    quantity = models.DecimalField(max_digits=10, decimal_places=2,
verbose_name="Кількість")
    unit = models.CharField(max_length=50, verbose_name="Одиниця виміру")
    status = models.CharField(max_length=20, choices=STATUS_CHOICES,
default='available', verbose_name="Статус", db_index=True)
    date_added = models.DateTimeField(default=timezone.now, verbose_name="Дата
додавання", db_index=True)
    added_by = models.CharField(max_length=100, verbose_name="Хто додав")
    organization = models.CharField(max_length=255, verbose_name="Організація",
db_index=True)
    expiry_date = models.DateField(null=True, blank=True, verbose_name="Термін
придатності", db_index=True)
    storage_location = models.CharField(max_length=255, verbose_name="Місце
зберігання", db_index=True)
```

```
    comment = models.TextField(blank=True, null=True, verbose_name="Коментар")
    photo = models.ImageField(upload_to='resource_photos/', blank=True, null=True,
verbose_name="Фото")

    def __str__(self):
        return self.name

    class Meta:
        verbose_name = "Ресурс"
        verbose_name_plural = "Ресурси"
        indexes = [
            models.Index(fields=['name']),
            models.Index(fields=['category']),
            models.Index(fields=['status']),
            models.Index(fields=['date_added']),
            models.Index(fields=['organization']),
            models.Index(fields=['storage_location']),
        ]

class Volunteer(models.Model):
    STATUS_CHOICES = [
        ('active', 'Активний'),
        ('inactive', 'Неактивний'),
        ('pending', 'Очікує підтвердження'),
        ('blocked', 'Заблокований'),
    ]

    last_name = models.CharField(max_length=100, verbose_name="Прізвище",
db_index=True)
    first_name = models.CharField(max_length=100, verbose_name="Ім'я", db_index=True)
    middle_name = models.CharField(max_length=100, blank=True, null=True,
verbose_name="По Батькові")
    phone = models.CharField(max_length=20, verbose_name="Телефон")
    email = models.EmailField(max_length=100, verbose_name="Email", db_index=True)
    telegram_id = models.CharField(max_length=100, blank=True, null=True,
verbose_name="Telegram ID")
    registration_date = models.DateTimeField(default=timezone.now, verbose_name="Дата
реєстрації", db_index=True)
    status = models.CharField(max_length=20, choices=STATUS_CHOICES, default='pending',
verbose_name="Статус", db_index=True)
    skills = models.TextField(blank=True, null=True, verbose_name="Навички")
    description = models.TextField(blank=True, null=True, verbose_name="Опис")
    organization = models.CharField(max_length=255, blank=True, null=True,
verbose_name="Організація", db_index=True)
    photo = models.ImageField(upload_to='volunteer_photos/', blank=True, null=True,
verbose_name="Фото")
    user = models.OneToOneField(User, on_delete=models.CASCADE, blank=True, null=True,
verbose_name="Користувач")
    last_login = models.DateTimeField(blank=True, null=True, verbose_name="Останній
вхід", db_index=True)
```

```
    def __str__(self):
        return f"{self.last_name} {self.first_name}"

    class Meta:
        verbose_name = "Волонтер"
        verbose_name_plural = "Волонтери"
        indexes = [
            models.Index(fields=['last_name', 'first_name']),
            models.Index(fields=['email']),
            models.Index(fields=['status']),
            models.Index(fields=['registration_date']),
            models.Index(fields=['organization']),
        ]

class ActionLog(models.Model):
    ACTION_CHOICES = [
        ('added', 'Додано'),
        ('updated', 'Змінено'),
        ('deleted', 'Видалено'),
    ]
    SUBJECT_CHOICES = [
        ('resource', 'Ресурс'),
        ('volunteer', 'Волонтер'),
    ]
    action = models.CharField(max_length=10, choices=ACTION_CHOICES,
verbose_name='Дія')
    subject = models.CharField(max_length=20, choices=SUBJECT_CHOICES,
verbose_name='Предмет')
    timestamp = models.DateTimeField(auto_now_add=True, verbose_name='Час дії')
    description = models.TextField(blank=True, null=True, verbose_name='Опис дії')
    performer = models.CharField(max_length=200, blank=True, null=True,
verbose_name='Виконавець')

    def __str__(self):
        return f"{self.get_action_display()} {self.get_subject_display()} (#{self.id})"

    class Meta:
        verbose_name = 'Журнал дій'
        verbose_name_plural = 'Журнал дій'
```

api/urls.py

```
from django.urls import path, include
from .views import ResourceViewSet, VolunteerViewSet, action_log_list
from rest_framework.routers import DefaultRouter

router = DefaultRouter()
router.register(r'resources', ResourceViewSet)
```

```
router.register(r'volunteers', VolunteerViewSet)

urlpatterns = [
    path('history/', action_log_list, name='action_log_list'),
    path('', include(router.urls)),]
```

api/views.py

```
from rest_framework.decorators import api_view, action
from rest_framework.response import Response
from rest_framework import viewsets, filters, status
from .models import Resource, Volunteer, ActionLog
from rest_framework import serializers
from django_filters.rest_framework import DjangoFilterBackend
from django.contrib.auth.models import User
from django.utils import timezone
from django.db import transaction
from rest_framework.permissions import IsAuthenticated
from rest_framework.authtoken.models import Token

class ActionLoggingMixin:
    def get_performer_name(self):
        """Get the name of the action performer"""
        if self.request.user.is_authenticated:
            volunteer = getattr(self.request.user, 'volunteer', None)
            if volunteer:
                return f"{volunteer.first_name} {volunteer.last_name}"
            return self.request.user.get_full_name() or self.request.user.username
        return None

    def log_action(self, action, subject, description):
        """Create an action log entry"""
        ActionLog.objects.create(
            action=action,
            subject=subject,
            description=description,
            performer=self.get_performer_name()
        )

# Створюємо серіалізатор для моделі Resource
class ResourceSerializer(serializers.ModelSerializer):
    class Meta:
        model = Resource
        fields = '__all__'

# Створюємо API представлення
class ResourceViewSet(ActionLoggingMixin, viewsets.ModelViewSet):
    queryset = Resource.objects.all().select_related()
    serializer_class = ResourceSerializer
    filter_backends = [DjangoFilterBackend, filters.SearchFilter,
filters.OrderingFilter]
```

```
    filterset_fields = ['category', 'status', 'organization', 'storage_location']
    search_fields = ['name', 'comment', 'added_by', 'organization']
    ordering_fields = ['name', 'date_added', 'quantity', 'expiry_date']
    ordering = ['-date_added']
    permission_classes = [IsAuthenticated]

    def get_permissions(self):
        if self.action in ['list', 'retrieve']:
            return []
        return [IsAuthenticated()]

    def perform_create(self, serializer):
        instance = serializer.save()
        self.log_action(
            'added',
            'resource',
            f"Додано новий ресурс: {instance.name} (ID: {instance.id})"
        )

    def perform_update(self, serializer):
        old_instance = self.get_object()
        old_data = {field.name: getattr(old_instance, field.name) for field in
old_instance._meta.fields}
        instance = serializer.save()
        new_data = {field.name: getattr(instance, field.name) for field in
instance._meta.fields}

        changed_fields = [
            f"{key}: '{old_data[key]}' → '{new_data[key]}'"
            for key in new_data
            if old_data[key] != new_data[key]
        ]

        description = f"Змінено ресурс: {instance.name} (ID: {instance.id}). "
        if changed_fields:
            description += "Змінені поля: " + "; ".join(changed_fields)
        else:
            description += "Без змін у полях."

        self.log_action('updated', 'resource', description)

    def perform_destroy(self, instance):
        self.log_action(
            'deleted',
            'resource',
            f"Видалено ресурс: {instance.name} (ID: {instance.id})"
        )
        instance.delete()

# Створюємо серіалізатор для моделі Volunteer
```

```
class VolunteerSerializer(serializers.ModelSerializer):
    password = serializers.CharField(write_only=True, required=True)

    class Meta:
        model = Volunteer
        fields = '__all__'
        extra_kwargs = {'user': {'read_only': True}}

    def create(self, validated_data):
        password = validated_data.pop('password', None)
        volunteer = Volunteer.objects.create(**validated_data)
        return volunteer

# Створюємо API представлення для моделі Volunteer
class VolunteerViewSet(ActionLoggingMixin, viewsets.ModelViewSet):
    queryset = Volunteer.objects.all().select_related('user')
    serializer_class = VolunteerSerializer
    filter_backends = [DjangoFilterBackend, filters.SearchFilter,
filters.OrderingFilter]
    filterset_fields = ['status', 'organization']
    search_fields = ['last_name', 'first_name', 'middle_name', 'skills', 'description',
'email', 'phone']
    ordering_fields = ['last_name', 'first_name', 'registration_date', 'last_login']
    ordering = ['last_name', 'first_name']
    permission_classes = [IsAuthenticated]

    def get_permissions(self):
        if self.action in ['create', 'login', 'list', 'retrieve']:
            return []
        return [IsAuthenticated()]

    def perform_create(self, serializer):
        instance = serializer.save()
        self.log_action(
            'added',
            'volunteer',
            f"Додано нового волонтера: {instance.first_name} {instance.last_name} (ID:
{instance.id})"
        )

    def perform_update(self, serializer):
        old_instance = self.get_object()
        old_data = {field.name: getattr(old_instance, field.name) for field in
old_instance._meta.fields}
        instance = serializer.save()
        new_data = {field.name: getattr(instance, field.name) for field in
instance._meta.fields}

        changed_fields = [
            f"{key}: '{old_data[key]}' → '{new_data[key]}'"
```

```
            for key in new_data
            if old_data[key] != new_data[key]
        ]

        description = f"Змінено волонтера: {instance.first_name} {instance.last_name}
(ID: {instance.id}). "
        if changed_fields:
            description += "Змінені поля: " + "; ".join(changed_fields)
        else:
            description += "Без змін у полях."

        self.log_action('updated', 'volunteer', description)

    def perform_destroy(self, instance):
        self.log_action(
            'deleted',
            'volunteer',
            f"Видалено волонтера: {instance.first_name} {instance.last_name}"
        )
        instance.delete()

    def create(self, request, *args, **kwargs):
        required_fields = ['first_name', 'last_name', 'email', 'phone', 'password']
        missing_fields = [field for field in required_fields if field not in
request.data]

        if missing_fields:
            return Response(
                {"message": f"Відсутні обов'язкові поля: {', '.join(missing_fields)}"},
                status=status.HTTP_400_BAD_REQUEST
            )

        serializer = self.get_serializer(data=request.data)
        if not serializer.is_valid():
            return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

        email = serializer.validated_data.get('email')

        # Перевірка унікальності email
        if User.objects.filter(email=email).exists() or
Volunteer.objects.filter(email=email).exists():
            return Response(
                {"message": "Користувач з такою електронною поштою вже існує"},
                status=status.HTTP_400_BAD_REQUEST
            )

        try:
            with transaction.atomic():
                # Створюємо користувача Django
                user = User.objects.create_user(
```

```
                    username=email,
                    email=email,
                    password=serializer.validated_data['password'],
                    first_name=serializer.validated_data['first_name'],
                    last_name=serializer.validated_data['last_name']
                )

                # Створюємо волонтера
                volunteer = serializer.save(user=user)

                # Створюємо запис в історії змін з системою як виконавцем
                ActionLog.objects.create(
                    action='added',
                    subject='volunteer',
                    description=f"Додано нового волонтера: {volunteer.first_name}
{volunteer.last_name} (ID: {volunteer.id})",
                    performer='Система'
                )

                # Створюємо токен для користувача
                token = Token.objects.create(user=user)

                return Response({
                    "message": "Реєстрація успішна",
                    "token": token.key,
                    "volunteer": self.get_serializer(volunteer).data
                }, status=status.HTTP_201_CREATED)

        except Exception as e:
            return Response(
                {"message": f"Помилка при реєстрації: {str(e)}"},
                status=status.HTTP_400_BAD_REQUEST
            )

    @action(detail=False, methods=['post'], url_path='login')
    def login(self, request):
        email = request.data.get('email')
        password = request.data.get('password')

        if not email or not password:
            return Response(
                {"message": "Необхідно вказати email та пароль"},
                status=status.HTTP_400_BAD_REQUEST
            )

        try:
            user = User.objects.get(email=email)
        except User.DoesNotExist:
            return Response(
                {"message": "Користувача з такою електронною поштою не знайдено"},
```

```
                status=status.HTTP_404_NOT_FOUND
            )

        if not user.check_password(password):
            return Response(
                {"message": "Введено неправильний логін або пароль"},
                status=status.HTTP_401_UNAUTHORIZED
            )

        try:
            volunteer = Volunteer.objects.get(user=user)
        except Volunteer.DoesNotExist:
            return Response(
                {"message": "Обліковий запис волонтера не знайдено"},
                status=status.HTTP_404_NOT_FOUND
            )

        if volunteer.status != 'active':
            return Response(
                {"message": "Адміністратор ще не підтвердив вас"},
                status=status.HTTP_403_FORBIDDEN
            )

        # Оновлюємо час останнього входу
        volunteer.last_login = timezone.now()
        volunteer.save()

        # Отримуємо або створюємо токен
        token, _ = Token.objects.get_or_create(user=user)

        return Response({
            "message": "Вхід успішний",
            "token": token.key,
            "volunteer": self.get_serializer(volunteer).data
        })

@api_view(['GET'])
def action_log_list(request):
    if not request.user.is_authenticated:
        return Response({"message": "Необхідна автентифікація"},
status=status.HTTP_401_UNAUTHORIZED)

    # Get query parameters
    offset = int(request.GET.get('offset', 0))
    page_size = int(request.GET.get('page_size', 10))
    subject = request.GET.get('subject', '')
    action = request.GET.get('action', '')
    date_from = request.GET.get('date_from', '')
    date_to = request.GET.get('date_to', '')
```

```
    # Start with all logs
    logs = ActionLog.objects.all()

    # Apply filters
    if subject:
        logs = logs.filter(subject=subject)
    if action:
        logs = logs.filter(action=action)
    if date_from:
        logs = logs.filter(timestamp__date__gte=date_from)
    if date_to:
        logs = logs.filter(timestamp__date__lte=date_to)

    # Order by timestamp descending
    logs = logs.order_by('-timestamp')

    # Get total count
    total_count = logs.count()

    # Apply pagination
    logs = logs[offset:offset + page_size]

    # Format response
    data = {
        'total': total_count,
        'results': [{
            'id': log.id,
            'timestamp': log.timestamp,
            'action': log.action,
            'subject': log.subject,
            'description': log.description,
            'performer': log.performer
        } for log in logs]
        }

    return Response(data)
```

Клієнтська частина

index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import reportWebVitals from './reportWebVitals';
import CssBaseline from '@mui/material/CssBaseline';


const root = ReactDOM.createRoot(document.getElementById('root'));
```

```
root.render(
  <React.StrictMode>
      <CssBaseline />
      <App />
    </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

App.js

```
import React, { useEffect, useState } from 'react';
import { BrowserRouter, Routes, Route, useLocation } from 'react-router-dom';
import MainPage from './components/main-page/MainPage';
import Navigation from './components/navigation/Navigation';
import ResourcesPage from './components/resources-page/ResourcesPage';
import VolunteersPage from './components/volunteers-page/VolunteersPage';
import RegisterPage from './components/auth/RegisterPage';
import ResourceDetailsPage from './components/resources-page/resource-
details/ResourceDetailsPage';
import { UserProvider } from './contexts/UserContext';
import HistoryPage from './components/resources-page/HistoryPage';
import MissionPage from './components/mission-page/MissionPage';
import ProfilePage from './components/profile/ProfilePage';
import Footer from './components/common/Footer';
import './App.css';

// Компонент для скролу на початок при зміні маршруту
const ScrollToTop = () => {
  const { pathname } = useLocation();
  useEffect(() => {
    window.scrollTo(0, 0);
  }, [pathname]);
  return null;
};

// Компонент для обробки паралакс-ефекту
const ParallaxEffect = () => {
  useEffect(() => {
    let ticking = false;
    let lastScrollY = 0;

    const handleScroll = () => {
      lastScrollY = window.scrollY;
      if (!ticking) {
        window.requestAnimationFrame(() => {
          const parallaxBg = document.querySelector('.parallax-background');
```

```
          if (parallaxBg) {
            parallaxBg.style.transform = `translateY(${lastScrollY * 0.1}px)`;
          }
          ticking = false;
        });
        ticking = true;
      }
    };

    handleScroll();
    window.addEventListener('scroll', handleScroll, { passive: true });
    return () => window.removeEventListener('scroll', handleScroll);
  }, []);

  return null;
};

const AppRoutes = ({ navValue, setNavValue, loginModalOpen, setLoginModalOpen }) => (
  <Routes>
    <Route
      path="/"
      element={
        <MainPage
          setNavValue={setNavValue}
          setLoginModalOpen={setLoginModalOpen}
        />
      }
    />
    <Route path="/resources" element={<ResourcesPage />} />
    <Route path="/volunteers" element={<VolunteersPage />} />
    <Route path="/mission" element={<MissionPage />} />
    <Route path="/register" element={<RegisterPage />} />
    <Route path="/profile" element={<ProfilePage />} />
    <Route path="/resources/:id" element={<ResourceDetailsPage />} />
    <Route
      path="/history"
      element={
        <HistoryPage
          navValue={navValue}
          setNavValue={setNavValue}
          loginModalOpen={loginModalOpen}
          setLoginModalOpen={setLoginModalOpen}
        />
      }
    />
  </Routes>
);

function App() {
  const [navValue, setNavValue] = useState(0);
```

```
  const [loginModalOpen, setLoginModalOpen] = useState(false);

  return (
    <UserProvider>
      <BrowserRouter>
        <ScrollToTop />
        <ParallaxEffect />
        <div className="app-wrapper parallax-container">
          <div className="parallax-background" />
          <div className="parallax-overlay" />
          <Navigation
            navValue={navValue}
            setNavValue={setNavValue}
            loginModalOpen={loginModalOpen}
            setLoginModalOpen={setLoginModalOpen}
          />
          <div className="content-wrapper">
            <AppRoutes
              navValue={navValue}
              setNavValue={setNavValue}
              loginModalOpen={loginModalOpen}
              setLoginModalOpen={setLoginModalOpen}
            />
          </div>
          <Footer />
        </div>
      </BrowserRouter>
    </UserProvider>
  );
}

export default App;
```

services/api.js

```
import axios from 'axios';

// Base URL for all API calls - приклад URL для локального сервера
const API_BASE_URL = 'http://localhost:8000/api/';

// Create an axios instance with default config
const api = axios.create({
  baseURL: API_BASE_URL,
  headers: {
    'Content-Type': 'application/json'
  },
  withCredentials: false
});

// Add a request interceptor to include the token in requests if available
```

```
api.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem('token');
    if (token) {
      config.headers['Authorization'] = `Token ${token}`;
    }
    return config;
  },
  (error) => {
    return Promise.reject(error);
  }
);

// API service для операцій з волонтерами
const volunteerService = {
  // Реєстрація нового волонтера
  registerVolunteer: async (volunteerData) => {
    try {
      const formData = new FormData();

      // Маппінг полів з frontend до backend моделі
      const fieldMapping = {
        lastName: 'last_name',
        firstName: 'first_name',
        middleName: 'middle_name',
        phone: 'phone',
        email: 'email',
        telegramId: 'telegram_id',
        skills: 'skills',
        description: 'description',
        organization: 'organization',
        password: 'password'
      };

      // Логування даних, які надсилаються (без пароля)
      const logData = {...volunteerData};
      delete logData.password;
      delete logData.confirmPassword;
      console.log('Дані для реєстрації:', logData);

      // Додаємо всі текстові поля до formData з правильними іменами полів
      Object.keys(volunteerData).forEach(key => {
        if (key !== 'photoUrl' && key !== 'confirmPassword' && volunteerData[key] !==
null) {
          const backendField = fieldMapping[key] || key;
          formData.append(backendField, volunteerData[key]);
          console.log(`Додано поле ${key} → ${backendField}: ${key === 'password' ?
'******' : volunteerData[key]}`);
        }
      });
```

```
// Додаємо фото, якщо воно є, з правильним іменем поля
if (volunteerData.photoUrl) {
  formData.append('photo', volunteerData.photoUrl);
  console.log('Додано фото:', volunteerData.photoUrl.name);
}

// Для діагностики - виведемо всі поля FormData
console.log('FormData містить наступні поля:');
for (let pair of formData.entries()) {
  console.log(pair[0] + ': ' + (pair[0] === 'password' ? '******' : pair[1]));
}

console.log('Відправка даних на сервер для реєстрації...');
try {
  const response = await api.post('/volunteers/', formData, {
    headers: {
      'Content-Type': 'multipart/form-data'
    }
  });

  console.log('Дані успішно надіслані на сервер:', response.data);

  // Якщо реєстрація успішна, відразу виконуємо вхід
  if (response.data && response.data.success) {
    // Оскільки бекенд не повертає токен при реєстрації, виконуємо вхід
    const loginData = {
      email: volunteerData.email,
      password: volunteerData.password
    };

    try {
      const loginResponse = await volunteerService.loginVolunteer(loginData);
      console.log('Автоматичний вхід після реєстрації:', loginResponse);
      return {
        ...response.data,
        autoLogin: true
      };
    } catch (loginError) {
      console.warn('Помилка автоматичного входу після реєстрації:', loginError);
      // Повертаємо дані реєстрації навіть якщо автовхід не вдався
      return response.data;
    }
  }

  return response.data;
} catch (error) {
  console.error('Помилка запиту до сервера:', error);
  console.error('Статус:', error.response?.status);
  console.error('Відповідь сервера:', error.response?.data);
```

```
      // Повертаємо зрозуміле повідомлення про помилку
      if (error.response && error.response.data) {
        console.log('Деталі помилки від сервера:', error.response.data);

        if (typeof error.response.data === 'object') {
          // Детальний вивід всіх помилок валідації
          Object.keys(error.response.data).forEach(key => {
            console.error(`Помилка поля ${key}:`, error.response.data[key]);
          });
        }

        if (error.response.data.message) {
          throw new Error(error.response.data.message);
        } else if (error.response.data.email) {
          throw new Error(`Помилка з email: ${error.response.data.email}`);
        } else if (error.response.data.password) {
          throw new Error(`Помилка з паролем: ${error.response.data.password}`);
        } else if (typeof error.response.data === 'string') {
          throw new Error(error.response.data);
        }
      }

      throw error;
    }
  } catch (error) {
    console.error('Помилка при реєстрації волонтера:', error);
    throw error;
  }
},

// Вхід волонтера
loginVolunteer: async (credentials) => {
  try {
    // Підготовка даних для запиту
    const loginData = {
      email: credentials.email,
      password: credentials.password
    };

    const response = await api.post('/volunteers/login/', loginData);

    if (response.data && response.data.token) {
      // Зберігаємо токен авторизації та дані користувача
      localStorage.setItem('token', response.data.token);
      localStorage.setItem('currentUser', JSON.stringify(response.data.volunteer));

      return {
        success: true,
        message: 'Успішний вхід',
```

```
        token: response.data.token,
        volunteer: response.data.volunteer
      };
    }

    throw new Error('Неправильні дані автентифікації');
  } catch (error) {
    // Повертаємо зрозуміле повідомлення про помилку
    if (error.response && error.response.data && error.response.data.message) {
      throw new Error(error.response.data.message);
    }

    throw error;
  }
},

// Вихід користувача
logoutVolunteer: () => {
  localStorage.removeItem('token');
  localStorage.removeItem('currentUser');
},

// Перевірка чи авторизований користувач
isAuthenticated: () => {
  return !!localStorage.getItem('token');
},

// Отримання даних поточного користувача з локального сховища
getCurrentUser: () => {
  const userData = localStorage.getItem('currentUser');
  return userData ? JSON.parse(userData) : null;
},

// Отримання актуальних даних користувача з сервера
fetchCurrentUser: async () => {
  try {
    const token = localStorage.getItem('token');
    const currentUser = JSON.parse(localStorage.getItem('currentUser'));

    if (!token || !currentUser || !currentUser.id) {
      return null;
    }

    const response = await api.get(`/volunteers/${currentUser.id}/`);
    if (response.data) {
      // Оновлюємо дані в локальному сховищі
      localStorage.setItem('currentUser', JSON.stringify(response.data));
      return response.data;
    }
    return null;
```

```
  } catch (error) {
    console.error('Помилка отримання даних користувача:', error);
    if (error.response && error.response.status === 401) {
      // Якщо токен недійсний, очищаємо локальне сховище
      localStorage.removeItem('token');
      localStorage.removeItem('currentUser');
    }
    throw error;
  }
},

// Отримання списку волонтерів
getVolunteers: async () => {
  try {
    const response = await api.get('/volunteers/');
    return response.data;
  } catch (error) {
    console.error('Помилка отримання списку волонтерів:', error);
    if (error.response) {
      // Якщо сервер відповів з помилкою
      if (error.response.status === 401) {
        // Для неавторизованих запитів просто повертаємо дані
        return { results: [], count: 0 };
      }
      throw new Error(error.response.data.message || 'Помилка отримання даних');
    } else if (error.request) {
      // Якщо запит не дійшов до сервера
      throw new Error('Не вдалося з\'єднатися з сервером');
    } else {
      throw new Error('Помилка при формуванні запиту');
    }
  }
},

// Отримання даних волонтера за ID
getVolunteerById: async (id) => {
  try {
    const response = await api.get(`/volunteers/${id}/`);
    return response.data;
  } catch (error) {
    console.error('Помилка отримання даних волонтера:', error);
    throw error;
  }
},

// Оновлення даних волонтера
updateVolunteer: async (id, formData) => {
  try {
    const response = await api.patch(`/volunteers/${id}/`, formData, {
      headers: {
```

```
          'Content-Type': 'multipart/form-data'
        }
      });

      // Оновлюємо дані в локальному сховищі
      if (response.data) {
        const currentUser = JSON.parse(localStorage.getItem('currentUser'));
        const updatedUser = { ...currentUser, ...response.data };
        localStorage.setItem('currentUser', JSON.stringify(updatedUser));
      }

      return response.data;
    } catch (error) {
      console.error('Помилка при оновленні профілю:', error);

      if (error.response && error.response.data) {
        if (typeof error.response.data === 'object') {
          const errorMessage = Object.entries(error.response.data)
            .map(([key, value]) => `${key}: ${value}`)
            .join('; ');
          throw new Error(errorMessage);
        } else {
          throw new Error(error.response.data);
        }
      }

      throw new Error('Помилка при оновленні профілю');
    }
  },

  // Видалення акаунту волонтера
  deleteVolunteer: async (id) => {
    try {
      await api.delete(`/volunteers/${id}/`);
      // Очищаємо дані користувача з локального сховища
      localStorage.removeItem('token');
      localStorage.removeItem('currentUser');
    } catch (error) {
      console.error('Помилка при видаленні акаунту:', error);
      if (error.response && error.response.data) {
        throw new Error(error.response.data.message || 'Помилка при видаленні
акаунту');
      }
      throw new Error('Помилка при видаленні акаунту');
    }
  }
};

// API service для операцій з ресурсами
const resourceService = {
```

```
  // Додати новий ресурс
  addResource: async (resourceData) => {
    try {
      const formData = new FormData();
      // Маппінг полів з frontend до backend моделі
      const fieldMapping = {
        name: 'name',
        category: 'category',
        quantity: 'quantity',
        unit: 'unit',
        storage_location: 'storage_location',
        comment: 'comment',
        organization: 'organization',
        added_by: 'added_by'
      };

      Object.keys(resourceData).forEach(key => {
        if (key !== 'photo' && resourceData[key] !== null && resourceData[key] !==
undefined) {
          const backendField = fieldMapping[key] || key;
          formData.append(backendField, resourceData[key]);
        }
      });

      if (resourceData.photo) {
        formData.append('photo', resourceData.photo);
      }

      const response = await api.post('/resources/', formData, {
        headers: {
          'Content-Type': 'multipart/form-data'
        }
      });
      return response.data;
    } catch (error) {
      if (error.response && error.response.data && error.response.data.message) {
        throw new Error(error.response.data.message);
      }
      throw error;
    }
  },

  // Оновити ресурс
  updateResource: async (id, updateData) => {
    try {
      const response = await api.patch(`/resources/${id}/`, updateData);
      return response.data;
    } catch (error) {
      if (error.response && error.response.data && error.response.data.message) {
        throw new Error(error.response.data.message);
```

```
    }
    throw error;
  }
},

// Пошук ресурсів з фільтрами
searchResources: async (searchParams) => {
  try {
    const { searchTerm, category, location } = searchParams;
    let url = '/resources/?';

    if (searchTerm) {
      url += `search=${encodeURIComponent(searchTerm)}&`;
    }
    if (category && category !== 'all') {
      url += `category=${encodeURIComponent(category)}&`;
    }
    if (location && location !== 'all') {
      url += `storage_location=${encodeURIComponent(location)}&`;
    }

    const response = await api.get(url);
    return response.data;
  } catch (error) {
    if (error.response && error.response.data && error.response.data.message) {
      throw new Error(error.response.data.message);
    }
    throw error;
  }
},

// Видалити ресурс
deleteResource: async (id) => {
  try {
    await api.delete(`/resources/${id}/`);
  } catch (error) {
    if (error.response && error.response.data && error.response.data.message) {
      throw new Error(error.response.data.message);
    }
    throw error;
  }
},

// Отримати список ресурсів
getResources: async () => {
  try {
    const response = await api.get('/resources/');
    return response.data;
  } catch (error) {
    if (error.response && error.response.data && error.response.data.message) {
```

```
        throw new Error(error.response.data.message);
      }
      throw error;
    }
  },

  // Отримати ресурс за ID
  getResourceById: async (id) => {
    try {
      const response = await api.get(`/resources/${id}/`);
      return response.data;
    } catch (error) {
      if (error.response && error.response.data && error.response.data.message) {
        throw new Error(error.response.data.message);
      }
      throw error;
    }
  }
};

export { volunteerService, resourceService };
export default api;
```

contexts/UserContext.js

```
import React, { createContext, useState, useEffect, useContext, useCallback } from
'react';
import { volunteerService } from '../services/api';

// Створення контексту
export const UserContext = createContext();

// Кастомний хук для використання контексту користувача
export const useUser = () => useContext(UserContext);

// Компонент UserProvider для обгортання застосунку
export const UserProvider = ({ children }) => {
  const [state, setState] = useState({
    user: null,
    loading: true,
    shouldRefreshVolunteers: false
  });

  const updateState = (updates) => {
    setState(prev => ({ ...prev, ...updates }));
  };

  // Перевірка авторизації при завантаженні
  useEffect(() => {
    const checkLoggedInUser = async () => {
```

```
      try {
        const currentUser = volunteerService.getCurrentUser();
        if (currentUser?.id) {
          const userData = await volunteerService.getVolunteerById(currentUser.id);
          updateState({ user: userData });
        }
      } catch (error) {
        console.error('Помилка при отриманні даних користувача:', error);
      } finally {
        updateState({ loading: false });
      }
    };

    checkLoggedInUser();
  }, []);

  // Функції для роботи з користувачем
  const login = useCallback(async (credentials) => {
    try {
      const response = await volunteerService.loginVolunteer(credentials);
      if (response.volunteer) {
        const userData = await
volunteerService.getVolunteerById(response.volunteer.id);
        updateState({ user: userData });
      }
      return response;
    } catch (error) {
      throw error;
    }
  }, []);

  const logout = useCallback(() => {
    volunteerService.logoutVolunteer();
    updateState({ user: null });
  }, []);

  const updateUser = useCallback((userData) => {
    updateState({ user: userData });
    localStorage.setItem('currentUser', JSON.stringify(userData));
    updateState({ shouldRefreshVolunteers: true });
  }, []);

  const setShouldRefreshVolunteers = useCallback((value) => {
    updateState({ shouldRefreshVolunteers: value });
  }, []);

  const contextValue = {
    user: state.user,
    loading: state.loading,
    login,
```

```
    logout,
    updateUser,
    shouldRefreshVolunteers: state.shouldRefreshVolunteers,
    setShouldRefreshVolunteers,
    isAuthenticated: !!state.user
  };

  return (
    <UserContext.Provider value={contextValue}>
      {children}
    </UserContext.Provider>
  );
};
```

components/auth/LoginModal.js

```
import React, { useState, useCallback } from 'react';
import {
  Dialog,
  DialogContent,
  DialogActions,
  TextField,
  Button,
  Typography,
  IconButton,
  InputAdornment,
  Alert,
  Box
} from '@mui/material';
import { useNavigate } from 'react-router-dom';
import CloseIcon from '@mui/icons-material/Close';
import VisibilityIcon from '@mui/icons-material/Visibility';
import VisibilityOffIcon from '@mui/icons-material/VisibilityOff';
import { useUser } from '../../contexts/UserContext';
import './LoginModal.css';

const INITIAL_FORM_STATE = {
  email: '',
  password: ''
};

const ERROR_MESSAGES = {
  403: 'Адміністратор ще не підтвердив вас',
  401: 'Введено неправильний логін або пароль',
  404: 'Користувача з такою електронною поштою не знайдено',
  500: 'Помилка сервера. Спробуйте пізніше',
  networkError: 'Не вдалося з\'єднатися з сервером. Перевірте підключення до
інтернету',
  default: 'Виникла неочікувана помилка. Спробуйте пізніше'
};
```

```
const LoginModal = ({ open, onClose }) => {
  const navigate = useNavigate();
  const { login } = useUser();

  const [formData, setFormData] = useState(INITIAL_FORM_STATE);
  const [showPassword, setShowPassword] = useState(false);
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState(null);

  const handleChange = useCallback((e) => {
    const { name, value } = e.target;
    setFormData(prev => ({ ...prev, [name]: value }));
    setError(null);
  }, []);

  const handleError = useCallback((error) => {
    if (error.response?.status) {
      setError(ERROR_MESSAGES[error.response.status] ||
        `Не вдалося увійти: ${error.response?.data?.detail ||
error.response?.data?.message || 'Невідома помилка'}`);
    } else if (error.message === 'Network Error') {
      setError(ERROR_MESSAGES.networkError);
    } else if (typeof error === 'string') {
      setError(error);
    } else if (error.message) {
      setError(error.message);
    } else {
      setError(ERROR_MESSAGES.default);
    }
  }, []);

  const handleSubmit = async (e) => {
    e.preventDefault();
    setIsLoading(true);
    setError(null);

    try {
      await login(formData);
      onClose();
    } catch (error) {
      handleError(error);
    } finally {
      setIsLoading(false);
    }
  };

  const handleTogglePasswordVisibility = useCallback(() => {
    setShowPassword(prev => !prev);
  }, []);
```

```
const handleCreateAccount = useCallback(() => {
  onClose();
  navigate('/register');
}, [onClose, navigate]);

const renderPasswordField = () => (
  <TextField
    margin="dense"
    label="Пароль"
    type={showPassword ? "text" : "password"}
    name="password"
    value={formData.password}
    onChange={handleChange}
    fullWidth
    required
    variant="outlined"
    className="login-input"
    disabled={isLoading}
    error={!!error}
    InputProps={{
      endAdornment: (
        <InputAdornment position="end">
          <IconButton
            aria-label="toggle password visibility"
            onClick={handleTogglePasswordVisibility}
            edge="end"
            disabled={isLoading}
          >
            {showPassword ? <VisibilityOffIcon /> : <VisibilityIcon />}
          </IconButton>
        </InputAdornment>
      ),
    }}
  />
);

return (
    <Dialog
      open={open}
      onClose={onClose}
      maxWidth="xs"
      fullWidth
      className="login-dialog"
    >
      <Box className="custom-dialog-title">
        <Typography variant="h5" component="div">
          Вхід до системи
        </Typography>
        <IconButton
```

```
      aria-label="close"
      onClick={onClose}
      className="close-button"
    >
      <CloseIcon />
    </IconButton>
  </Box>

  <form onSubmit={handleSubmit}>
    <DialogContent className="login-content">
    {error && (
      <Alert severity="error" sx={{ mb: 2 }}>
        {error}
      </Alert>
    )}

      <TextField
        margin="dense"
        label="Електронна пошта"
        type="email"
        name="email"
        value={formData.email}
        onChange={handleChange}
        fullWidth
        required
        variant="outlined"
        className="login-input"
        disabled={isLoading}
      error={!!error}
    />

    {renderPasswordField()}

      <Typography
        variant="body2"
        color="primary"
        className="forgot-password-link"
      onClick={() => {}} // TODO: Implement forgot password functionality
      >
        Забули пароль?
      </Typography>
    </DialogContent>

    <DialogActions className="login-actions">
      <Button
        type="submit"
        variant="contained"
        color="primary"
        fullWidth
        className="login-button"
```

```
              disabled={isLoading}
            >
              {isLoading ? 'Перевірка...' : 'Увійти'}
            </Button>

            <Typography variant="body2" className="create-account-text">
              Новий волонтер?
              <span
                className="create-account-link"
                onClick={handleCreateAccount}
              >
                Створити акаунт
              </span>
            </Typography>
          </DialogActions>
        </form>
      </Dialog>
  );
};

export default LoginModal;
```

components/auth/RegisterPage.js

```
import React, { useState, useEffect, useCallback } from 'react';
import {
  Container,
  Paper,
  TextField,
  Button,
  Typography,
  Box,
  Avatar,
  Alert
} from '@mui/material';
import CloudUploadIcon from '@mui/icons-material/CloudUpload';
import { useNavigate } from 'react-router-dom';
import { volunteerService } from '../../services/api';
import './RegisterPage.css';

const INITIAL_FORM_STATE = {
    lastName: '',
    firstName: '',
    middleName: '',
    phone: '',
    email: '',
```

```
    telegramId: '',
    skills: '',
    description: '',
    organization: '',
    password: '',
    confirmPassword: '',
    photoUrl: null
};

const VALIDATION_RULES = {
  password: {
    minLength: 8,
    message: 'Пароль повинен містити щонайменше 8 символів!'
  },
  confirmPassword: {
    match: 'password',
    message: 'Паролі не співпадають!'
  }
};

const RegisterPage = () => {
  const navigate = useNavigate();
  const [formData, setFormData] = useState(INITIAL_FORM_STATE);
  const [photoPreview, setPhotoPreview] = useState(null);
  const [isSubmitting, setIsSubmitting] = useState(false);
  const [error, setError] = useState(null);

  const handleParallax = useCallback(() => {
    let ticking = false;
    let lastScrollY = 0;

    const updateParallax = () => {
      lastScrollY = window.scrollY;
      if (!ticking) {
        window.requestAnimationFrame(() => {
          const parallaxBg = document.querySelector('.parallax-background');
          if (parallaxBg) {
            parallaxBg.style.transform = `translateY(${lastScrollY * 0.1}px)`;
          }
          ticking = false;
        });
        ticking = true;
      }
    };

    updateParallax();
    window.addEventListener('scroll', updateParallax, { passive: true });
    return () => window.removeEventListener('scroll', updateParallax);
  }, []);
```

```
useEffect(() => {
  const cleanup = handleParallax();
  return () => cleanup();
}, [handleParallax]);

const handleChange = useCallback((e) => {
  const { name, value } = e.target;
  setFormData(prev => ({ ...prev, [name]: value }));
  setError(null);
}, []);

const handlePhotoChange = useCallback((e) => {
  const file = e.target.files[0];
  if (file) {
    const reader = new FileReader();
    reader.onloadend = () => {
      setPhotoPreview(reader.result);
      setFormData(prev => ({ ...prev, photoUrl: file }));
    };
    reader.readAsDataURL(file);
  }
}, []);

const validateForm = useCallback(() => {
  if (formData.password.length < VALIDATION_RULES.password.minLength) {
    setError(VALIDATION_RULES.password.message);
    return false;
  }
  if (formData.password !== formData.confirmPassword) {
    setError(VALIDATION_RULES.confirmPassword.message);
    return false;
  }
  return true;
}, [formData.password, formData.confirmPassword]);

const handleSubmit = async (e) => {
  e.preventDefault();
  if (!validateForm()) return;

  setIsSubmitting(true);
  setError(null);

  try {
    await volunteerService.registerVolunteer(formData);

    // Показуємо успішне повідомлення
    navigate('/', {
      state: {
        showLoginModal: false,
        registrationSuccess: true,
```

```
          message: 'Реєстрація пройшла успішно! Очікуйте підтвердження від
адміністратора системи для можливості авторизації.'
        }
      });
    } catch (error) {
      setError(error.message || 'Помилка при реєстрації. Спробуйте ще раз.');
      setIsSubmitting(false);
    }
  };

  const renderField = useCallback(({ name, label, description, type = 'text', required
= false }) => (
    <Box className="field-container">
      <Typography variant="caption" className="field-description">
        {description}
      </Typography>
      <TextField
        label={label}
        name={name}
        type={type}
        value={formData[name]}
        onChange={handleChange}
        fullWidth
        required={required}
        variant="outlined"
        className="form-field"
        disabled={isSubmitting}
      />
    </Box>
  ), [formData, handleChange, isSubmitting]);

  const renderPhotoUpload = () => (
    <Box className="photo-upload-container">
      <input
        type="file"
        accept="image/*"
        id="photo-upload"
        onChange={handlePhotoChange}
        style={{ display: 'none' }}
      />
      <label htmlFor="photo-upload">
        <Button
          variant="outlined"
          component="span"
          startIcon={<CloudUploadIcon />}
          className="upload-button"
          disabled={isSubmitting}
        >
          Завантажити фото
        </Button>
```

```
      </label>
      {photoPreview && (
        <Avatar
          src={photoPreview}
          alt="Preview"
          className="photo-preview"
        />
      )}
    </Box>
  );

  return (
    <div className="app-wrapper">
      <Container maxWidth="sm" className="register-page-container">
        <Paper elevation={3} className="register-page-paper">
          <Typography variant="h4" component="h1" align="center" className="register-
page-title">
            Реєстрація волонтера
          </Typography>

          <form onSubmit={handleSubmit}>
            <Box className="register-page-content">
              <Box className="form-section">
                <Typography variant="h6" className="section-title">
                  Особиста інформація
                </Typography>

                {renderField({
                  name: 'lastName',
                  label: 'Прізвище',
                  description: 'Введіть ваше прізвище',
                  required: true
                })}

                {renderField({
                  name: 'firstName',
                  label: "Ім'я",
                  description: "Введіть ваше ім'я",
                  required: true
                })}

                {renderField({
                  name: 'middleName',
                  label: 'По батькові',
                  description: "Введіть ваше по батькові (необов'язково)"
                })}

                {renderField({
                  name: 'phone',
                  label: 'Номер телефону',
```

```
  description: 'Введіть ваш номер телефону',
  required: true
})}

{renderField({
  name: 'email',
  label: 'Електронна пошта',
  description: 'Введіть вашу електронну пошту',
  type: 'email',
  required: true
})}

{renderField({
  name: 'telegramId',
  label: 'Telegram ID',
  description: "Введіть ваш Telegram ID (необов'язково)"
})}

{renderField({
  name: 'skills',
  label: 'Навички',
  description: 'Опишіть ваші навички та досвід',
  required: true
})}

{renderField({
  name: 'description',
  label: 'Про себе',
  description: 'Розкажіть про себе'
})}

{renderField({
  name: 'organization',
  label: 'Організація',
  description: 'Вкажіть вашу організацію'
})}

{renderField({
  name: 'password',
  label: 'Пароль',
  description: 'Придумайте надійний пароль',
  type: 'password',
  required: true
})}

{renderField({
  name: 'confirmPassword',
  label: 'Підтвердження пароля',
  description: 'Повторіть пароль',
  type: 'password',
```

```
                required: true
              })}

              {renderPhotoUpload()}
            </Box>

            {error && (
              <Alert severity="error" className="error-alert">
                {error}
              </Alert>
            )}

            <Box className="form-actions">
            <Button
                type="button"
                variant="outlined"
                onClick={() => navigate('/')}
              className="cancel-button"
                disabled={isSubmitting}
            >
              Скасувати
            </Button>
            <Button
              type="submit"
              variant="contained"
              color="primary"
              className="submit-button"
              disabled={isSubmitting}
            >
              {isSubmitting ? 'Реєстрація...' : 'Зареєструватися'}
            </Button>
            </Box>
          </Box>
        </form>
      </Paper>
    </Container>
  </div>
 );
};

export default RegisterPage;
```

components/common/Footer.js

```
import React from 'react';
import { Box, Container, Typography, Link } from '@mui/material';
import './Footer.css';

const Footer = () => {
  const currentYear = new Date().getFullYear();
```

```
  return (
    <Box component="footer" className="footer">
      <Container maxWidth="lg">
        <Box className="footer-content">
          <Box className="footer-section">
            <Typography variant="h6" gutterBottom>
              Система моніторингу ресурсів
            </Typography>
            <Typography variant="body2" color="textSecondary">
              Платформа для ефективної координації волонтерської допомоги
            </Typography>
          </Box>

          <Box className="footer-section">
            <Typography variant="h6" gutterBottom>
              Контакти
            </Typography>
            <Typography variant="body2" color="textSecondary">
              Email: support@volunteer-system.com
            </Typography>
            <Typography variant="body2" color="textSecondary">
              Телефон: +38 (044) 123-45-67
            </Typography>
          </Box>

          <Box className="footer-section">
            <Typography variant="h6" gutterBottom>
              Посилання
            </Typography>
            <Link href="/resources" color="inherit">
              Ресурси
            </Link>
            <Link href="/volunteers" color="inherit">
              Волонтери
            </Link>
            <Link href="/mission" color="inherit">
              Про проєкт
            </Link>
          </Box>
        </Box>

        <Box className="footer-bottom">
          <Typography variant="body2" color="textSecondary" align="center">
            © {currentYear} Система моніторингу ресурсів. Всі права захищено.
          </Typography>
        </Box>
      </Container>
    </Box>
  );
};
```

```
export default Footer;
```

components/main-page/MainPage.js

```
import React, { useEffect, useState } from 'react';
import { Typography, Container, Box, Alert, Snackbar } from '@mui/material';
import InventoryIcon from '@mui/icons-material/Inventory';
import PeopleIcon from '@mui/icons-material/People';
import CategoryIcon from '@mui/icons-material/Category';
import { Carousel } from 'react-responsive-carousel';
import "react-responsive-carousel/lib/styles/carousel.min.css";
import { useLocation, useNavigate } from 'react-router-dom';
import Mission from './mission/Mission';
import CarouselCard from './carousel-card/CarouselCard';
import WelcomeSection from './welcome-section/WelcomeSection';
import JoinUsSection from './join-us-section/JoinUsSection';
import './MainPage.css';

// Дані для карток у каруселі
const CARDS_DATA = [
  {
    id: 'resources',
    icon: <InventoryIcon className="card-icon resource-icon" />,
    name: "Моніторинг ресурсів",
    description: {
      title: "Наша система дозволяє вам:",
      items: [
        "Переглядати доступні ресурси в реальному часі",
        "Знаходити ресурси за місцем розташування",
        "Зв'язуватися безпосередньо з постачальниками"
      ]
    }
  },
  {
    id: 'volunteers',
    icon: <PeopleIcon className="card-icon volunteer-icon" />,
    name: "База волонтерів",
    description: {
      title: "Створіть свій профіль волонтера та:",
      items: [
        "Опишіть свій досвід та навички",
        "Вкажіть напрямки волонтерської діяльності",
        "Додайте контактну інформацію для зв'язку"
      ]
    }
  },
  {
    id: 'categories',
    icon: <CategoryIcon className="card-icon curator-icon" />,
    name: "Категорії ресурсів",
```

```
    description: {
      title: "Зручна система категорій:",
      items: [
        "Медичні засоби та ліки",
        "Продукти харчування",
        "Одяг та спорядження",
        "Технічне обладнання"
      ]
    }
  }
];

// Налаштування каруселі
const CAROUSEL_SETTINGS = {
  showThumbs: false,
  infiniteLoop: true,
  autoPlay: true,
  interval: 6000,
  showStatus: false,
  swipeable: true,
  useKeyboardArrows: true,
  emulateTouch: true,
  stopOnHover: true
};

const MainPage = ({ setNavValue }) => {
  const location = useLocation();
  const navigate = useNavigate();
  const [openSnackbar, setOpenSnackbar] = useState(false);
  const [snackbarMessage, setSnackbarMessage] = useState('');
  const registrationSuccess = location.state?.registrationSuccess;
  const message = location.state?.message;

  useEffect(() => {
    if (registrationSuccess && message) {
      setSnackbarMessage(message);
      setOpenSnackbar(true);
      // Очищаємо стан location після показу повідомлення
      navigate('/', { replace: true });
    }
  }, [registrationSuccess, message, navigate]);

  const handleCloseSnackbar = (event, reason) => {
    if (reason === 'clickaway') {
      return;
    }
    setOpenSnackbar(false);
  };

  const renderMissionSection = () => (
```

```
      <Box className="content-box">
        <Typography variant="h4" gutterBottom className="section-title">
          Наша задача
        </Typography>

        <p className="section-text">
        Система моніторингу ресурсів для волонтерів допомагає ефективно координувати
        волонтерську діяльність, забезпечуючи швидке реагування на потреби громади
        та оптимальне використання наявних ресурсів.
        </p>

        <p className="section-text">
        Волонтерство відіграє критичну роль у сучасному суспільстві, особливо в часи
кризи.
        Громадянське суспільство активізується там, де державні механізми не можуть
забезпечити
        всі потреби громадян, особливо найбільш вразливих груп населення.
        </p>
      </Box>
  );

  const renderCarouselSection = () => (
      <div className="carousel-container">
        <Typography variant="h4" component="h2" className="section-title">
          Чим можемо бути корисним?
        </Typography>

        <Carousel className="card-carousel" {...CAROUSEL_SETTINGS}>
        {CARDS_DATA.map(({ id, icon, name, description }) => (
            <CarouselCard
            key={id}
            icon={icon}
            name={name}
            description={description}
            />
          ))}
        </Carousel>
      </div>
  );

  return (
    <Container maxWidth={false} className="main-page-container">
      <Snackbar
        open={openSnackbar}
        autoHideDuration={5000}
        onClose={handleCloseSnackbar}
        anchorOrigin={{ vertical: 'top', horizontal: 'center' }}
        sx={{
          top: '80px !important',
          zIndex: 1000
```

```
        }}
      >
        <Alert
          onClose={handleCloseSnackbar}
          severity="success"
          variant="filled"
          sx={{
            width: '100%',
            maxWidth: '600px',
            boxShadow: '0 4px 12px rgba(0,0,0,0.15)'
          }}
        >
          {snackbarMessage}
        </Alert>
      </Snackbar>
      <div className="welcome-section">
        <WelcomeSection setNavValue={setNavValue} />
      </div>
      {renderMissionSection()}
      {renderCarouselSection()}
      <div className="mission-section">
        <Mission />
      </div>
      <div className="join-us-section">
        <JoinUsSection />
      </div>
    </Container>
  );
};

export default MainPage;
```

components/mission-page/MissionPage.js

```
import React, { memo } from 'react';
import { Container, Typography, Paper, Box } from '@mui/material';

const MISSION_CONTENT = {
  title: 'Наша місія',
  sections: {
    about: {
      title: 'Про проєкт',
      content: `Система моніторингу ресурсів для волонтерів - це платформа, створена
для оптимізації та
        покращення процесу волонтерської допомоги. Наша мета - об'єднати волонтерів та
ефективно
        розподіляти ресурси там, де вони найбільш потрібні.`
    },
```

```
    goals: {
      title: 'Наші цілі',
      items: [
        'Створення єдиної платформи для координації волонтерської діяльності',
        'Оптимізація розподілу ресурсів та допомоги',
        'Забезпечення прозорості волонтерської діяльності',
        'Спрощення комунікації між волонтерами та організаціями',
        'Підвищення ефективності надання допомоги'
      ]
    },
    principles: {
      title: 'Наші принципи',
      content: `Ми віримо в прозорість, ефективність та відповідальність у
волонтерській діяльності.
        Кожен ресурс має бути використаний максимально ефективно, а кожна дія -
принести
        реальну користь тим, хто цього потребує.`
    }
  }
};

const MissionPage = () => {
  const renderSection = (section, type = 'text') => (
    <Box sx={{ mb: 4 }}>
      <Typography variant="h6" gutterBottom>
        {section.title}
      </Typography>
      {type === 'list' ? (
        <Typography component="div">
          <ul>
            {section.items.map((item, index) => (
              <li key={index}>{item}</li>
            ))}
          </ul>
        </Typography>
      ) : (
        <Typography paragraph>
          {section.content}
        </Typography>
      )}
    </Box>
  );

  return (
    <Container maxWidth="md" sx={{ py: 4 }}>
      <Paper
        elevation={3}
        sx={{
          p: 4,
          backgroundColor: 'rgba(255, 255, 255, 0.95)',
```

```
          borderRadius: 2
        }}
      >
        <Box sx={{ textAlign: 'center', mb: 4 }}>
          <Typography
            variant="h4"
            component="h1"
            gutterBottom
            color="primary"
          >
            {MISSION_CONTENT.title}
          </Typography>
        </Box>

        {renderSection(MISSION_CONTENT.sections.about)}
        {renderSection(MISSION_CONTENT.sections.goals, 'list')}
        {renderSection(MISSION_CONTENT.sections.principles)}
      </Paper>
    </Container>
  );
};

export default memo(MissionPage);
```

components/navigation/Navigation.js

```
import React from 'react';
import { AppBar, Toolbar, Typography, IconButton, Tabs, Tab, Avatar, Menu, MenuItem }
from '@mui/material';
import AccountCircleIcon from '@mui/icons-material/AccountCircle';
import { useNavigate, useLocation } from 'react-router-dom';
import LoginModal from '../auth/LoginModal';
import { useUser } from '../../contexts/UserContext';
import './Navigation.css';

const Navigation = ({ navValue, setNavValue, loginModalOpen, setLoginModalOpen }) => {
  const navigate = useNavigate();
  const location = useLocation();
  const { user, logout, isAuthenticated } = useUser();
  const [anchorEl, setAnchorEl] = React.useState(null);

  // Встановлюємо значення навігації в false, якщо ми на сторінці профілю
  React.useEffect(() => {
    if (location.pathname === '/profile' && setNavValue) {
      setNavValue(false);
    }
  }, [location.pathname, setNavValue]);

  const getPhotoUrl = () => {
```

```
  if (!user || !user.photo) {
    return null;
  }
  if (user.photo.startsWith('http')) {
    return user.photo;
  }
  return `http://localhost:8000${user.photo}`;
};

const handleLogoClick = () => {
  setNavValue && setNavValue(0);
  navigate('/');
};

const handleAvatarClick = (event) => {
  if (isAuthenticated) {
    setAnchorEl(event.currentTarget);
  } else {
    setLoginModalOpen(true);
  }
};

const handleMenuClose = () => {
  setAnchorEl(null);
};

const handleLoginClose = () => {
  setLoginModalOpen(false);
};

const handleLogout = () => {
  logout();
  handleMenuClose();
  navigate('/');
  setNavValue && setNavValue(0);
};

const handleProfile = () => {
  navigate('/profile');
  handleMenuClose();
};

const handleTabChange = (event, newValue) => {
  setNavValue(newValue);
  switch (newValue) {
    case 0:
      navigate('/');
      break;
    case 1:
      navigate('/resources');
```

```
      break;
    case 2:
      navigate('/volunteers');
      break;
    case 3:
      navigate('/mission');
      break;
    default:
      break;
  }
};
return (
  <>
    <AppBar position="fixed" className="app-bar">
      <Toolbar>
        <Typography
          variant="h6"
          onClick={handleLogoClick}
          className="site-title"
        >
          Система моніторингу ресурсів
        </Typography>
        <Tabs
          value={navValue}
          onChange={handleTabChange}
          aria-label="navigation tabs"
          className="navigation-tabs"
        >
          <Tab label="Головна" />
          <Tab label="Ресурси" />
          <Tab label="Волонтери" />
          <Tab label="Про проєкт" />
        </Tabs>
        <IconButton
          color="inherit"
          className="user-icon"
          onClick={handleAvatarClick}
        >
          {isAuthenticated && user ? (
            <Avatar
              alt={`${user.first_name} ${user.last_name}`}
              src={getPhotoUrl()}
              className="user-avatar"
            >
              <AccountCircleIcon />
            </Avatar>
          ) : (
            <AccountCircleIcon />
          )}
        </IconButton>
```

```
        </Toolbar>
      </AppBar>

      {/* меню користувача */}
      <Menu
        anchorEl={anchorEl}
        open={Boolean(anchorEl)}
        onClose={handleMenuClose}
        anchorOrigin={{
          vertical: 'bottom',
          horizontal: 'right',
        }}
        transformOrigin={{
          vertical: 'top',
          horizontal: 'right',
        }}
        sx={{
          '& .MuiPaper-root': {
            marginTop: '8px',
            minWidth: '200px',
            boxShadow: '0 2px 10px rgba(0,0,0,0.1)',
          }
        }}
        disableScrollLock={true}
      >
        <MenuItem onClick={handleProfile}>Мій профіль</MenuItem>
        <MenuItem onClick={handleLogout}>Вийти</MenuItem>
      </Menu>

      <LoginModal
        open={loginModalOpen}
        onClose={handleLoginClose}
      />
    </>
  );
};

export default Navigation;
```

components/profile/ProfilePage.js

```
import React, { useState, useEffect, useCallback } from 'react';
import {…} from '@mui/material';
import {…} from '@mui/icons-material';
import { useUser } from '../../contexts/UserContext';
import { volunteerService } from '../../services/api';
import EditProfileForm from './EditProfileForm';
import { useNavigate } from 'react-router-dom';
import './ProfilePage.css';

const formatDate = (dateString) => {
```

```
  if (!dateString) return 'не вказано';
  return new Date(dateString).toLocaleString('uk-UA', {
    year: 'numeric',
    month: '2-digit',
    day: '2-digit',
    hour: '2-digit',
    minute: '2-digit'
  });
};

const formatPhone = (phone) => {
  if (!phone) return 'не вказано';
  const cleaned = phone.replace(/\D/g, '');
  if (cleaned.length === 10) {
    return `+38 (${cleaned.slice(0,3)}) ${cleaned.slice(3,6)}-
${cleaned.slice(6,8)}-${cleaned.slice(8)}`;
  }
  return phone;
};

const parseSkills = (skills) => {
  if (!skills) return [];
  if (typeof skills === 'string') {
    return skills.split(',').map(skill => skill.trim()).filter(skill => skill);
  }
  return Array.isArray(skills) ? skills : [];
};

const ProfilePage = () => {
  const navigate = useNavigate();
  const { user, updateUser, logout } = useUser();
  const [volunteerData, setVolunteerData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [isEditModalOpen, setIsEditModalOpen] = useState(false);
  const [isDeleteDialogOpen, setIsDeleteDialogOpen] = useState(false);
  const [isDeleting, setIsDeleting] = useState(false);

  const fetchVolunteerData = useCallback(async () => {
    if (!user?.id) {
      setError('Не вдалося визначити ID користувача');
      setLoading(false);
      return;
    }

    try {
      setLoading(true);
      setError(null);
      const response = await volunteerService.getVolunteerById(user.id);
      setVolunteerData(response);
```

```
    } catch (err) {
      console.error('Помилка при завантаженні даних профілю:', err);
      if (err.response?.status === 404) {
        setError('Профіль не знайдено. Можливо, ви були видалені з системи.');
      } else if (err.response?.status === 401) {
        setError('Необхідно повторно авторизуватися в системі.');
      } else {
        setError('Не вдалося завантажити дані профілю. Спробуйте оновити
сторінку.');
      }
    } finally {
      setLoading(false);
    }
  }, [user]);

  useEffect(() => {
    fetchVolunteerData();
  }, [fetchVolunteerData]);

  const handleProfileUpdate = useCallback((updatedData) => {
    setVolunteerData(updatedData);
    updateUser(updatedData);
  }, [updateUser]);

  const getPhotoUrl = useCallback((photo) => {
    if (!photo) return null;
    return photo.startsWith('http') ? photo : `http://localhost:8000${photo}`;
  }, []);

  const handleDeleteAccount = async () => {
    try {
      setIsDeleting(true);
      await volunteerService.deleteVolunteer(user.id);
      logout();
      navigate('/');
    } catch (error) {
      setError(error.message);
      setIsDeleting(false);
    }
  };

  const renderContactInfo = () => (
    <Box className="profile-section">
      <Typography variant="h6" gutterBottom>
        Контактна інформація
      </Typography>
      <Box className="contact-info">
        <Box className="contact-item">
          <EmailIcon />
          <Typography>{volunteerData.email}</Typography>
```

```
      </Box>
      <Box className="contact-item">
        <PhoneIcon />
        <Typography>{formatPhone(volunteerData.phone)}</Typography>
      </Box>
      {volunteerData.telegram_id && (
        <Box className="contact-item">
          <TelegramIcon />
          <Typography>{volunteerData.telegram_id}</Typography>
        </Box>
      )}
    </Box>
  </Box>
);

const renderOrganizationInfo = () => (
  volunteerData.organization && (
    <Box className="profile-section">
      <Typography variant="h6" gutterBottom>
        Організація
      </Typography>
      <Box className="organization-info">
        <BusinessIcon />
        <Typography>{volunteerData.organization}</Typography>
      </Box>
    </Box>
  )
);

const renderSkills = () => (
  volunteerData.skills && (
    <Box className="profile-section">
      <Typography variant="h6" gutterBottom>
        Навички
      </Typography>
      <Box className="skills-container">
        <BuildIcon />
        <Box className="skills-chips">
          {parseSkills(volunteerData.skills).map((skill, index) => (
            <Chip
              key={index}
              label={skill}
              className="skill-chip"
            />
          ))}
        </Box>
      </Box>
    </Box>
  )
);
```

```
const renderDescription = () => (
  volunteerData.description && (
    <Box className="profile-section">
      <Typography variant="h6" gutterBottom>
        Про мене
      </Typography>
      <Typography className="profile-description">
        {volunteerData.description}
      </Typography>
    </Box>
  )
);

const renderSystemInfo = () => (
  <Box className="profile-section">
    <Typography variant="h6" gutterBottom>
      Системна інформація
    </Typography>
    <Box className="system-info">
      <Box className="info-item">
        <AccessTimeIcon />
        <Box>
          <Typography variant="subtitle2" color="textSecondary">
            Дата реєстрації
          </Typography>
          <Typography>
            {formatDate(volunteerData.registration_date)}
          </Typography>
        </Box>
      </Box>
      <Box className="info-item">
        <AccessTimeIcon />
        <Box>
          <Typography variant="subtitle2" color="textSecondary">
            Останній вхід
          </Typography>
          <Typography>
            {formatDate(volunteerData.last_login)}
          </Typography>
        </Box>
      </Box>
      <Box className="info-item delete-account">
        <Button
          variant="outlined"
          color="error"
          startIcon={<DeleteIcon />}
          onClick={() => setIsDeleteDialogOpen(true)}
          fullWidth
        >
```

```
            Видалити акаунт
          </Button>
        </Box>
      </Box>
    </Box>
  );

  if (!user) {
    return (
      <Container maxWidth="md" className="profile-container">
        <Typography variant="h5" align="center">
          Для перегляду профілю необхідно авторизуватися
        </Typography>
      </Container>
    );
  }

  if (loading && !volunteerData) {
    return (
      <Container maxWidth="md" className="profile-container">
        <Box display="flex" justifyContent="center" alignItems="center"
minHeight="200px">
          <CircularProgress />
        </Box>
      </Container>
    );
  }

  if (error) {
    return (
      <Container maxWidth="md" className="profile-container">
        <Alert
          severity="error"
          action={
            <Button
              color="inherit"
              size="small"
              startIcon={<RefreshIcon />}
              onClick={fetchVolunteerData}
            >
              Спробувати ще раз
            </Button>
          }
        >
          {error}
        </Alert>
      </Container>
    );
  }
```

```
if (!volunteerData) {
  return (
    <Container maxWidth="md" className="profile-container">
      <Alert severity="warning">
        Дані профілю не знайдено
      </Alert>
    </Container>
  );
}

return (
  <Container maxWidth="md" className="profile-container">
    <Paper elevation={3} className="profile-paper">
      <Box className="profile-header">
        <Typography variant="h4" className="profile-title">
          Мій профіль
        </Typography>
        <Button
          variant="contained"
          startIcon={<EditIcon />}
          className="edit-profile-button"
          onClick={() => setIsEditModalOpen(true)}
        >
          Редагувати профіль
        </Button>
      </Box>

      <Divider className="section-divider" />

      <Box className="profile-main-info">
        <Box className="profile-avatar-container">
          <Avatar
            src={getPhotoUrl(volunteerData.photo)}
            onError={(e) => { e.target.src = null }}
            className="profile-avatar"
          >
            <PersonIcon />
          </Avatar>
        </Box>

        <Box className="profile-name-container">
          <Typography variant="h5" className="profile-last-name">
            {volunteerData.last_name || 'Прізвище не вказано'}
          </Typography>
          <Typography variant="h6" className="profile-first-name">
            {volunteerData.first_name || 'Ім\'я не вказано'}
          </Typography>
          {volunteerData.middle_name && (
            <Typography variant="subtitle1" className="profile-middle-name">
              {volunteerData.middle_name}
```

```
              </Typography>
            )}
          </Box>
        </Box>

        <Box className="profile-details">
          {renderContactInfo()}
          {renderOrganizationInfo()}
          {renderSkills()}
          {renderDescription()}
          {renderSystemInfo()}
        </Box>
      </Paper>

      <Dialog
        open={isDeleteDialogOpen}
        onClose={() => !isDeleting && setIsDeleteDialogOpen(false)}
      >
        <DialogTitle>Підтвердження видалення акаунту</DialogTitle>
        <DialogContent>
          <DialogContentText>
            Ви впевнені, що хочете видалити свій акаунт? Ця дія є незворотною і
призведе до втрати всіх ваших даних.
          </DialogContentText>
        </DialogContent>
        <DialogActions>
          <Button
            onClick={() => setIsDeleteDialogOpen(false)}
            disabled={isDeleting}
          >
            Скасувати
          </Button>
          <Button
            onClick={handleDeleteAccount}
            color="error"
            disabled={isDeleting}
            startIcon={isDeleting ? <CircularProgress size={20} /> : <DeleteIcon
/>}
          >
            {isDeleting ? 'Видалення...' : 'Видалити'}
          </Button>
        </DialogActions>
      </Dialog>

      <EditProfileForm
        open={isEditModalOpen}
        onClose={() => setIsEditModalOpen(false)}
        volunteerData={volunteerData}
        onUpdate={handleProfileUpdate}
      />
```

```
    </Container>
  );
};

export default ProfilePage;
```

components/resources-page/ResourcesPage.js

```
import React, { useState, useEffect, useCallback } from 'react';
import {
  Container,
  Typography,
  CircularProgress,
  Button,
} from '@mui/material';
import CategoryIcon from '@mui/icons-material/Category';
import SearchComponent from './search-component/SearchComponent';
import ResourceCard from './resource-card/ResourceCard';
import './ResourcesPage.css';
import AddResourceModal from './AddResourceModal';
import { useUser } from '../../contexts/UserContext';
import { useLocation } from 'react-router-dom';
import { resourceService } from '../../services/api';

// Константи
const DEFAULT_LOCATION = 'all';

// Варіанти категорій відповідно до значень на бекенді
const CATEGORY_OPTIONS = {
  ALL: 'all',
  MEDICAL: 'медичні засоби',
  EQUIPMENT: 'спорядження',
  FOOD: 'продукти',
  TECH: 'обладнання',
  CLOTHES: 'одяг',
  OTHER: 'інше'
};

// Масив категорій для фільтра
const CATEGORIES_ARRAY = Object.values(CATEGORY_OPTIONS);

const ResourcesPage = () => {
  const [searchState, setSearchState] = useState({
    term: '',
    category: CATEGORY_OPTIONS.ALL,
    location: DEFAULT_LOCATION
  });
  const [resources, setResources] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const { isAuthenticated } = useUser();
```

```
const [addModalOpen, setAddModalOpen] = useState(false);
const location = useLocation();

const formatResourceData = useCallback((item) => ({
  id: item.id,
  name: item.name,
  category: item.category.charAt(0).toUpperCase() + item.category.slice(1),
  location: item.storage_location,
  quantity: parseFloat(item.quantity),
  unit: item.unit,
  description: item.comment || 'Опис відсутній',
  photo: item.photo || null,
  status: item.status
}), []);

const processApiResponse = useCallback((data) => {
  const resultsArray = data.results ? data.results : data;

  if (!Array.isArray(resultsArray)) {
    console.error('API response is not an array or paginated object:', data);
    throw new Error('Unexpected API response format');
  }

  return resultsArray.map(formatResourceData);
}, [formatResourceData]);

const fetchResources = useCallback(async () => {
  try {
    setLoading(true);
    setError(null);
    const data = await resourceService.getResources();
    if (data) {
      setResources(processApiResponse(data));
    }
  } catch (error) {
    console.error('Error fetching resources:', error);
    setError('Не вдалося завантажити дані про ресурси. Спробуйте оновити сторінку.');
  } finally {
    setLoading(false);
  }
}, [processApiResponse]);

useEffect(() => {
  fetchResources();
}, [fetchResources, location.key]);

useEffect(() => {
  if (location.state?.updatedResourceId) {
    fetchResources();
    window.history.replaceState({}, document.title);
```

```
    }
  }, [location.state, fetchResources]);

  useEffect(() => {
    if (location.state?.deletedResourceId) {
      setResources(prev => prev.filter(r => r.id !==
location.state.deletedResourceId));
      window.history.replaceState({}, document.title);
    }
  }, [location.state]);

  const handleSearchResults = useCallback((searchResults) => {
    if (searchResults) {
      setResources(processApiResponse(searchResults));
    }
  }, [processApiResponse]);

  const handleNewResource = useCallback((newResource) => {
    setResources(prev => [formatResourceData(newResource), ...prev]);
  }, [formatResourceData]);

  const getFilteredResources = useCallback(() => {
    return resources.filter(resource => {
      const categoryMatch = searchState.category === CATEGORY_OPTIONS.ALL ||
                            resource.category.toLowerCase() ===
searchState.category.toLowerCase();

      const locationMatch = searchState.location === DEFAULT_LOCATION ||
                            resource.location === searchState.location;

      const searchMatch = !searchState.term ||
                          resource.name.toLowerCase().includes(searchState.term.toLowerCa
se()) ||
                          (resource.description &&
resource.description.toLowerCase().includes(searchState.term.toLowerCase()));

    return categoryMatch && locationMatch && searchMatch;
  });
  }, [resources, searchState]);

  const uniqueLocations = [
    DEFAULT_LOCATION,
    ...new Set(resources.map(resource => resource.location).filter(Boolean))
  ];

  // Функції відображення
  const renderHeader = () => (
    <div className="header-container">
      <div className="header-content">
        <CategoryIcon className="header-icon" />
```

```
      <Typography variant="h4" className="header-title">
        База даних ресурсів
      </Typography>
      <CategoryIcon className="header-icon" />
    </div>
  </div>
);

const renderAddResourceSection = () => (
  isAuthenticated && (
      <div className="add-resource-container">
        <div className="add-resource-content">
          <span className="add-resource-title">ДОДАТИ РЕСУРС</span>
          <Button
            variant="contained"
            color="primary"
            className="add-resource-button"
            onClick={() => setAddModalOpen(true)}
            aria-label="Додати ресурс"
          >
            +
          </Button>
          <Button
            variant="contained"
            color="primary"
            className="history-button"
            onClick={() => window.location.href = '/history'}
            aria-label="Історія змін"
          >
            Історія змін
          </Button>
        </div>
      </div>
  )
);

const renderContent = () => {
  if (loading) {
    return (
      <div className="resources-loading">
        <CircularProgress />
      </div>
    );
  }

  if (error) {
    return (
      <div className="resources-error">
        <Typography color="error">{error}</Typography>
      </div>
```

```
      );
    }

    const filteredResources = getFilteredResources();

    if (filteredResources.length === 0) {
      return (
        <div className="resources-empty">
          <Typography variant="h6">Ресурси не знайдено</Typography>
        </div>
      );
    }

    return (
      <div className="resources-grid">
        {filteredResources.map((resource) => (
          <div className="resources-grid-item" key={resource.id}>
            <ResourceCard resource={resource} />
          </div>
        ))}
      </div>
    );
  };

  // Основний рендер
  return (
    <Container maxWidth="lg" className="resources-container">
      {renderHeader()}

      <SearchComponent
        searchTerm={searchState.term}
        setSearchTerm={(term) => setSearchState(prev => ({ ...prev, term }))}
        categoryFilter={searchState.category}
        setCategoryFilter={(category) => setSearchState(prev => ({ ...prev, category
}))}
        locationFilter={searchState.location}
        setLocationFilter={(location) => setSearchState(prev => ({ ...prev, location
}))}
        categories={CATEGORIES_ARRAY}
        locations={uniqueLocations}
        onSearch={handleSearchResults}
      />

      {renderAddResourceSection()}

      <AddResourceModal
        open={addModalOpen}
        onClose={() => setAddModalOpen(false)}
        onResourceAdded={handleNewResource}
      />
```

```
      {renderContent()}
    </Container>
  );
};

export default ResourcesPage;
```

components/ volunteers-page/VolunteersPage.js

```
import React, { useState, useEffect, useCallback } from 'react';
import {
  Container,
  Typography,
  CircularProgress,
  Button,
} from '@mui/material';
import {
  People as PeopleIcon,
  Refresh as RefreshIcon
} from '@mui/icons-material';
import VolunteerSearchComponent from './volunteer-search/VolunteerSearchComponent';
import VolunteerCard from './volunteer-card/VolunteerCard';
import { volunteerService } from '../../services/api';
import { useLocation } from 'react-router-dom';
import { useUser } from '../../contexts/UserContext';
import './VolunteersPage.css';

const VolunteersPage = () => {
  const [searchState, setSearchState] = useState({
    term: '',
    volunteers: [],
    loading: true,
    error: null
  });

  const location = useLocation();
  const { shouldRefreshVolunteers, setShouldRefreshVolunteers } = useUser();

  const processApiResponse = useCallback((data) => {
    const resultsArray = data.results ? data.results : data;

    if (!Array.isArray(resultsArray)) {
      console.error('API response is not an array or paginated object:', data);
      throw new Error('Unexpected API response format');
    }

    return resultsArray;
  }, []);
```

```
  const fetchVolunteers = useCallback(async () => {
    try {
      setSearchState(prev => ({ ...prev, loading: true, error: null }));
      const data = await volunteerService.getVolunteers();
      const resultsArray = processApiResponse(data);
      setSearchState(prev => ({
        ...prev,
        volunteers: resultsArray,
        loading: false
      }));
    } catch (error) {
      console.error('Помилка при завантаженні даних:', error);
      setSearchState(prev => ({
        ...prev,
        error: error.message,
        loading: false
      }));
    }
  }, [processApiResponse]);

  useEffect(() => {
    fetchVolunteers();
  }, [fetchVolunteers]);

  useEffect(() => {
    if (location.state?.refresh || shouldRefreshVolunteers) {
      fetchVolunteers();
      window.history.replaceState({}, document.title);
      setShouldRefreshVolunteers(false);
    }
  }, [location.state, shouldRefreshVolunteers, setShouldRefreshVolunteers,
fetchVolunteers]);

  const getFilteredVolunteers = useCallback(() => {
    return searchState.volunteers.filter(volunteer => {
      if (!searchState.term) return true;

      const searchTermLower = searchState.term.toLowerCase();
      const searchableFields = [
        `${volunteer.last_name || ''} ${volunteer.first_name || ''}
${volunteer.middle_name || ''}`,
        volunteer.skills || '',
        volunteer.organization || '',
        volunteer.description || ''
      ];

      return searchableFields.some(field =>
        field.toLowerCase().includes(searchTermLower)
      );
```

```
  });
}, [searchState.volunteers, searchState.term]);

const handleSearchTermChange = useCallback((term) => {
  setSearchState(prev => ({ ...prev, term }));
}, []);

const renderHeader = () => (
  <div className="volunteers-header-container">
    <div className="volunteers-header-content">
      <PeopleIcon className="volunteers-header-icon" />
      <Typography variant="h4" className="header-title">
        База даних волонтерів
      </Typography>
      <PeopleIcon className="volunteers-header-icon" />
    </div>
  </div>
);

const renderLoading = () => (
  <div className="volunteers-loading">
    <CircularProgress />
  </div>
);

const renderError = () => (
  <div className="volunteers-error">
    <Typography color="error">
      {searchState.error}
      <Button
        startIcon={<RefreshIcon />}
        onClick={fetchVolunteers}
        color="primary"
        style={{ marginLeft: '1rem' }}
      >
        Оновити
      </Button>
    </Typography>
  </div>
);

const renderNoResults = () => (
  <div className="volunteers-no-results">
    <Typography variant="body1">Волонтерів не знайдено</Typography>
    <Button
      startIcon={<RefreshIcon />}
      onClick={fetchVolunteers}
      color="primary"
      style={{ marginTop: '1rem' }}
    >
```

```
        Оновити список
      </Button>
    </div>
  );

  const renderVolunteersList = () => {
    const filteredVolunteers = getFilteredVolunteers();

    if (filteredVolunteers.length === 0) {
      return renderNoResults();
    }

    return (
      <div className="volunteers-grid">
        {filteredVolunteers.map((volunteer) => (
          <div className="volunteers-grid-item" key={volunteer.id}>
            <VolunteerCard volunteer={volunteer} />
          </div>
        ))}
      </div>
    );
  };

  const renderContent = () => {
    if (searchState.loading) {
      return renderLoading();
    }

    if (searchState.error) {
      return renderError();
    }

    return renderVolunteersList();
  };

  return (
    <Container maxWidth="lg" className="volunteers-container">
      {renderHeader()}

      <div className="volunteers-controls">
        <VolunteerSearchComponent
          searchTerm={searchState.term}
          setSearchTerm={handleSearchTermChange}
        />
      </div>

      {renderContent()}
    </Container>
  );
};
```

```
export default VolunteersPage;
```