ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка системи краудсорсингу соціальної підтримки на базі web-технологій»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Ярослав ПОЛІЩУК
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

Ярослав ПОЛІЩУК

Керівник: Володимир САДОВЕНКО
канд.фіз.-мат.наук, доцент

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

____Поліщуку Ярославу Олексійовичу____

1. Тема кваліфікаційної роботи: «Розробка системи краудсорсингу соціальної підтримки на базі web-технологій»
керівник кваліфікаційної роботи канд. фіз.-мат. наук, доцент Володимир САДОВЕНКО,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: технічна документація фреймворку Django 5.x, бібліотек Leaflet.js, Bootstrap 5, Chart.js та pytest; теоретичні відомості про архітектурні патерни веб-застосунків (MTV); опис методів забезпечення безпеки веб-застосунків (CSRF, XSS, SQL-ін'єкції); наукові публікації з тематики краудсорсингових платформ соціальної підтримки.

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

1. Аналіз предметної галузі мікроволонтерства та постановка вимог до системи краудсорсингу соціальної підтримки.

2. Огляд та обґрунтування засобів реалізації веб-платформи (Python/Django, PostgreSQL, Leaflet.js, Bootstrap 5, Chart.js, Docker).
3. Проєктування системи MicroVolunteer: архітектура, структура бази даних, UML-діаграми.
4. Реалізація, тестування та розгортання веб-платформи.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Загальна архітектура системи.
5. ER-діаграма.
6. Варіанти використання.
7. Діаграма класів.
8. Сценарій взаємодії.
9. Екранні форми.
10. Апробація результатів дослідження.

6. Дата видачі завдання«23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	23.02-30.03.2026	
2	Аналіз та дослідження існуючих аналогів	30.03-05.04.2026	
3	Формулювання вимог та проєктування архітектури системи	06.04-12.04.2026	
4	Проєктування бази даних та розробка UML-діаграм	06.04-12.04.2026	
5	Програмна реалізація модулів системи	30.03-19.04.2026	
6	Тестування та налагодження системи	20.04-24.04.2026	
7	Оформлення роботи: вступ, висновки, анотація	15.04-07.05.2026	

8	Розробка демонстраційних матеріалів	27.04-16.05.2026	
9	Попередній захист роботи	11.05-22.05.2026	

Здобувач вищої освіти _____ Ярослав ПОЛІЩУК
(підпис)

Керівник
кваліфікаційної роботи _____ Володимир САДОВЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 75 стор., 12 таблиці, 22 рис., 43 джерела.

Мета роботи – спрощення організації та координації мікроволонтерської діяльності шляхом розроблення вебзастосунку для взаємодії волонтерів і отримувачів допомоги.

Об'єкт дослідження – процес організації мікроволонтерської діяльності для соціально вразливих груп населення.

Предмет дослідження – засоби та технології розробки вебсистеми краудсорсингу соціальної підтримки на базі фреймворку Django, бібліотеки Leaflet.js та СУБД PostgreSQL.

Короткий зміст роботи: Проаналізовано предметну галузь мікроволонтерства, досліджено наявні програмні рішення (Добробат, Be My Eyes, TaskRabbit) та виявлено їхні обмеження. Спроєктовано архітектуру вебплатформи MicroVolunteer на основі модульної структури Django, розроблено ER-модель бази даних та UML-діаграми. Реалізовано п'ять застосунків: автентифікацію з рольовою моделлю, інтерактивну карту запитів на базі Leaflet.js і Nominatim, систему відгуків та рейтингів, модуль сповіщень і статистичний дашборд. Забезпечено захист від CSRF, XSS, SQL-ін'єкцій і credential stuffing; приватність отримувача реалізовано через зсув геокоординат до 150 м. Проведено тестування з pytest і factory_boy, підготовлено розгортання на базі Docker, Nginx та Gunicorn.

Сферою використання вебзастосунку є автоматизація процесу координації локальної соціальної допомоги між отримувачами підтримки та волонтерами у системах мікроволонтерства.

КЛЮЧОВІ СЛОВА: МІКРОВОЛОНТЕРСТВО, КРАУДСОРСИНГ, ВЕБПЛАТФОРМА, DJANGO, POSTGRESQL, LEAFLET, ГЕОЛОКАЦІЯ, СОЦІАЛЬНА ПІДТРИМКА, ВОЛОНТЕР.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ADR — Architecture Decision Record — запис архітектурного рішення

AJAX — Asynchronous JavaScript and XML — асинхронний JavaScript та XML

API — Application Programming Interface — прикладний програмний інтерфейс

CSRF — Cross-Site Request Forgery — міжсайтова підробка запиту

CRUD — Create, Read, Update, Delete — базові операції з даними

CSS — Cascading Style Sheets — каскадні таблиці стилів

ER — Entity-Relationship — сутність–зв'язок (діаграма бази даних)

HSTS — HTTP Strict Transport Security — суворе транспортне захищення HTTP

HTTP(S) — Hypertext Transfer Protocol (Secure) — протокол передавання гіпертексту

IDOR — Insecure Direct Object Reference — небезпечне пряме посилання на об'єкт

IOM — International Organization for Migration — Міжнародна організація з міграції

JSON — JavaScript Object Notation — текстовий формат обміну даними

MVC / MTV — Model–View–Controller / Model–Template–View — архітектурний патерн

NIST — National Institute of Standards and Technology — Національний інститут стандартів США

OCHA — Office for the Coordination of Humanitarian Affairs — Управління з координації гуманітарних справ ООН

ORM — Object-Relational Mapping — об'єктно-реляційне відображення

OSM — OpenStreetMap — відкрита картографічна платформа

OWASP — Open Web Application Security Project — проєкт безпеки вебзастосунків

SPA — Single Page Application — односторінковий застосунок

SQL — Structured Query Language — мова структурованих запитів

SSE — Server-Sent Events — серверні події (протокол)

SSR — Server-Side Rendering — серверний рендеринг HTML

UAT — User Acceptance Testing — приймальне тестування користувача

UML — Unified Modeling Language — уніфікована мова моделювання

UNHCR — United Nations High Commissioner for Refugees — Верховний комісар ООН у справах біженців

URL — Uniform Resource Locator — уніфікований вказівник ресурсу

WCAG — Web Content Accessibility Guidelines — настанови щодо доступності веб-контенту

WSGI — Web Server Gateway Interface — інтерфейс шлюзу вебсервера

XSS — Cross-Site Scripting — міжсайтовий скриптинг

СУБД — Система управління базами даних

ВПО — Внутрішньо переміщена особа

ВСТУП

Актуальність теми пояснюється кількома чинниками — соціальними та технологічними, які впливають на організацію адресної допомоги в Україні. Після початку повномасштабного вторгнення значно збільшилася кількість людей, які потребують швидкої локальної підтримки: внутрішньо переміщених осіб, літніх людей, осіб з інвалідністю, багатодітних сімей та інших соціально вразливих груп. За даними міжнародних гуманітарних організацій, масштаби внутрішнього переміщення, тривалість гуманітарних потреб і навантаження на місцеві громади залишаються істотними [1, 2, 3]. У таких умовах волонтерська допомога не обмежується великими організаційними кампаніями, оскільки значну частину щоденних потреб становлять короткі локальні дії: доставка продуктів або ліків, супровід, консультація, допомога з побутовими завданнями, передавання інформації чи координація з місцевими службами.

Мета роботи – підвищення ефективності організації мікроволонтерської діяльності шляхом проєктування та впровадження вебплатформи для краудсорсингу соціальної підтримки, що з'єднує людей, які потребують локальної допомоги (літніх людей, осіб з інвалідністю, ВПО, багатодітних сімей) з волонтерами, готовими надати швидку підтримку, на базі вебтехнологій Django, JavaScript, PostgreSQL.

Для досягнення поставленої мети необхідно виконати такі завдання:

1. Провести теоретичний аналіз предметної галузі мікроволонтерства, виявити проблеми координації між волонтерами та отримувачами допомоги, обґрунтувати актуальність задачі.

2. Виконати порівняльний аналіз існуючих програмних рішень, сформулювати функціональні та нефункціональні вимоги до власної системи.

3. Спроєктувати архітектуру вебзастосунку (модульна структура, розподіл відповідальності між компонентами) та структуру бази даних (моделі, зв'язки, обмеження цілісності).

4. Реалізувати серверну частину системи мовою Python із використанням фреймворку Django, зокрема п'ять застосунків: автентифікацію, управління запитами, відгуки, сповіщення та статистику.

5. Реалізувати клієнтську частину з адаптивним інтерфейсом (Bootstrap 5) і доступністю, інтерактивною картою (Leaflet.js + CartoDB), автокомплітом адрес (Nominatim) та діаграмами (Chart.js).

6. Забезпечити захист системи від типових вебзагроз (CSRF, XSS, SQL-ін'єкції, перебір паролів) та механізми збереження приватності отримувача.

7. Розробити комплексний набір автоматизованих тестів (модульних та інтеграційних) із використанням pytest і factory_boy.

8. Підготувати сценарій розгортання системи у контейнеризованому середовищі (Docker + docker-compose + Nginx + Gunicorn) та оформити технічну й користувацьку документацію.

Об'єкт дослідження – процес організації мікроволонтерської діяльності для соціально вразливих груп населення.

Предмет дослідження – засоби та технології розробки вебсистеми краудсорсингу соціальної підтримки на базі фреймворку Django, бібліотеки Leaflet.js та СУБД PostgreSQL.

Наукова новизна роботи полягає у поєднанні моделей мікроволонтерства й цифрового краудсорсингу з геолокаційним та категорійним підбором запитів у межах єдиного вебзастосунку для українського соціального контексту. У межах роботи запропоновано модульну архітектуру вебплатформи MicroVolunteer, яка одночасно реалізує життєвий цикл запиту, інтерактивну карту, репутаційний механізм.

Практичне значення роботи полягає в розробленні вебплатформи MicroVolunteer, яку можна використовувати як прототип для громадських організацій, волонтерських об'єднань і територіальних громад. Система підтримує ролі користувачів, запити, карту, адресний автокомпліт, відгуки, сповіщення, рейтинги, статистику, автоматизовані тести й контейнеризований запуск.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ВИМОГ

1.1 Мікроволонтерство як форма соціальної взаємодопомоги

Мікроволонтерство доцільно розглядати як одну з форм добровільної соціальної участі, у якій допомога надається через короткотривалі, конкретні дії, що не вимагають тривалих зобов'язань. На відміну від традиційного волонтерства, що передбачає регулярну участь у діяльності організації, мікроволонтерська дія може тривати від кількох хвилин до декількох годин і виконуватися у зручний для учасника час — це суттєво знижує поріг входу для людей з обмеженим вільним розкладом.

Мікроволонтерство поєднує цінності добровільної допомоги з цифровими засобами координації. Закон України «Про волонтерську діяльність» визначає волонтерську діяльність як добровільну соціально спрямовану неприбуткову діяльність на благо суспільства [4]. Це означає, що для правового тлумачення вирішальними є не тривалість участі і не масштаб завдання, а добровільність, суспільна користь і відсутність фінансової винагороди. Воно спрямоване на конкретну людину або спільноту, виконується добровільно і не передбачає оплати.

За тривалістю та способом виконання мікроволонтерські дії можна поділити на миттєві (до 30 хвилин), короткострокові (до кількох годин) й епізодичні (разові завдання, що повторюються нерегулярно), а також на фізичні (доставка, супровід, ремонт), цифрові (переклад, транскрипція, модерація) та дистанційні (онлайн-консультація, відеодзвінок). Така класифікація важлива для проєктування системи: MicroVolunteer підтримує категорії, що охоплюють усі три типи, і дозволяє волонтеру фільтрувати запити саме за тим форматом участі, який йому доступний.

Соціальний контекст України суттєво посилює потребу в цифровій координації мікроволонтерства. Після впровадження воєнного стану понад 2 млн людей зареєструвалися як внутрішньо переміщені особи [1]. Це важливо, бо

йдеться не про абстрактну соціальну групу, а про мільйони людей, які в короткий проміжок часу опинилися на новому місці без усталених побутових зв'язків. Після початку повномасштабної війни у 2022 році кількість внутрішньо переміщених осіб перевищила 3,7 млн за даними IOM станом на січень 2026 року, а частка людей, що потребують адресної підтримки у повсякденних справах — доставці ліків, супроводі до лікаря, допомозі з документами, — значно зросла [2].

Звіт United Nations Volunteers «2022 State of the World's Volunteerism Report» підкреслює, що волонтерство є механізмом взаємодії між громадянами, спільнотами та інституціями, а цифрові платформи суттєво знижують транзакційні витрати на організацію цієї взаємодії [5]. Це означає, що платформа тут потрібна не для «діджиталізації» уже наявного процесу, а для того, щоб сама взаємодія не розпадалася при масштабуванні.

Окремою передумовою є цифрова готовність цільової аудиторії. За даними DataReportal «Digital 2024: Ukraine», рівень проникнення інтернету в Україні становить 79,6%, а кількість активних користувачів соцмереж — понад 18 млн [6]. Переважна більшість потенційних отримувачів і надавачів допомоги вже мають доступ до вебплатформ із мобільних пристроїв, тобто технологічний бар'єр для впровадження MicroVolunteer є не головною проблемою; головною проблемою є саме координація, яку така платформа повинна взяти на себе.

Таким чином, мікроволонтерство у межах цієї роботи визначається як цифрово-координована форма добровільної соціальної підтримки, у якій короткотривалі дії узгоджуються через вебплатформу за критеріями геолокації, категорії потреби та вільного часу волонтера. Саме це визначення покладено в основу функціональних вимог до MicroVolunteer і пояснює вибір кожного з його модулів.

1.2 Аналіз існуючих засобів цифрової координації мікроволонтерства

На практиці координація локальної допомоги найчастіше відбувається через Telegram-групи, Facebook-сторінки та Google Sheets. Ці інструменти не проєктувалися для соціальної координації: вони не мають структурованого формату запиту, не підтримують геолокаційний пошук і не відстежують статус виконання — і саме ці три відсутності перетворюють чат у хаотичний потік повідомлень, де реальні запити губляться.

Чат-групи можуть бути ефективними у перший час надзвичайної ситуації, але стають нефункціональними при масштабуванні — коли кількість повідомлень перевищує 50–100 на день, більшість запитів губляться у потоці. Це підтверджується практикою: волонтерські Telegram-спільноти 2022–2024 рр. регулярно стикалися з проблемою «дублювань і пропусків», коли один запит отримував п'ять відгуків, а інший — жодного.

Конкретні технічні обмеження месенджерів для координації волонтерства такі: адреса з тексту не може автоматично відображатися на карті; статус запиту — «знайшли волонтера» чи «ще шукають» — неможливо відстежити без ручної позначки адміністратора; повторні запити не видно, що призводить до дублювання; немає рейтингу волонтерів, тобто отримувач не знає, кому довіряти. Кожне з цих обмежень перетворюється на окрему функціональну вимогу до MicroVolunteer.

Водночас месенджери залишаються незамінним каналом швидкого оповіщення. Саме тому архітектура MicroVolunteer передбачає push-сповіщення через браузер, а не витісняє месенджери повністю — платформа бере на себе структурування, а Telegram залишається для сигналу.

Узагальнений аналіз цифрових каналів виявляє п'ять системних проблем: відсутність геолокаційної фільтрації запитів; відсутність стандартизованого формату опису потреби; неможливість відстеження статусу запиту; відсутність репутаційного механізму; неможливість агрегованої аналітики для адміністраторів. Кожна з цих проблем безпосередньо породжує відповідну функціональну вимогу до MicroVolunteer, описану у підрозділі 1.4.

1.3 Порівняльний аналіз існуючих програмних рішень

Для визначення функціональної ніші MicroVolunteer проведено порівняльний аналіз трьох платформ, які частково перетинаються з цільовою предметною областю: Добробат, Be My Eyes і TaskRabbit. Критерії аналізу обрано відповідно до виявлених проблем: геолокація, категоризація, статус запиту, репутаційний механізм, безкоштовність і соціальна спрямованість.

Добробат — українська ініціатива координації добровольців для відбудови пошкодженого житла. Платформа має чіткий предметний фокус на фізичній реконструкції, проте не підтримує повсякденну взаємодопомогу (доставку ліків, супровід, побутову допомогу). Відсутня двостороння система рейтингів — тобто два з п'яти виявлених обмежень месенджерів залишаються невирішеними [7].

Be My Eyes — міжнародна платформа, яка з'єднує людей з порушеннями зору з волонтерами через короткий відеодзвінок. Вона демонструє, що вузько визначена мікроволонтерська задача (дистанційна зорова допомога) може бути реалізована масштабованим чином [8]. Однак платформа заточена виключно під один тип допомоги й не підтримує геолокаційну модель або широкий спектр потреб — і саме вузькість є її обмеженням для загального випадку.

TaskRabbit — комерційна платформа платних локальних послуг. З точки зору UX вона є найбільш зрілим рішенням: є карта виконавців, детальна категоризація, рейтинги [9]. Проте вона принципово комерційна (оплата обов'язкова) і не підходить для соціальної взаємодопомоги між звичайними людьми — що є принциповою відмінністю від MicroVolunteer.

Порівняння наведено у таблиці 1.1. Як видно з таблиці, жодна з платформ не поєднує одночасно безкоштовність, широку категоризацію, геолокацію і двосторонній репутаційний механізм — саме це і є функціональною нішею MicroVolunteer.

Таблиця 1.1

Порівняльний аналіз існуючих платформ за ключовими критеріями

Критерій	Добробат	Be My Eyes	TaskRabbit	MicroVolunteer
Геолокація та карта	+	−	+	+
Категоризація запитів	+	−	+	+
Система рейтингів	−	+	+	+
Безкоштовність	+	+	−	+
Фокус на Україні	+	−	−	+
Мікрозавдання	−	+	+	+

Дані таблиці 1.1 показують, що кожне з розглянутих рішень покриває лише частину потреб цільової аудиторії. Добробат закривається на одному секторі (відбудова), Be My Eyes — на одному типі взаємодії (відеодзвінок для людей з порушеннями зору), TaskRabbit — на комерційному ринку. MicroVolunteer заповнює перетин: безкоштовна, широкоспекторна, геолокаційна платформа з репутаційним механізмом.

Узагальнену діаграму варіантів використання системи подано на рис. 1.1. Вона показує чотирьох основних акторів — гостя, отримувача, волонтера й адміністратора — та ключові сценарії: перегляд запитів, реєстрація, створення запиту, відгук волонтера, зміна статусу, рейтингування та перегляд статистики.

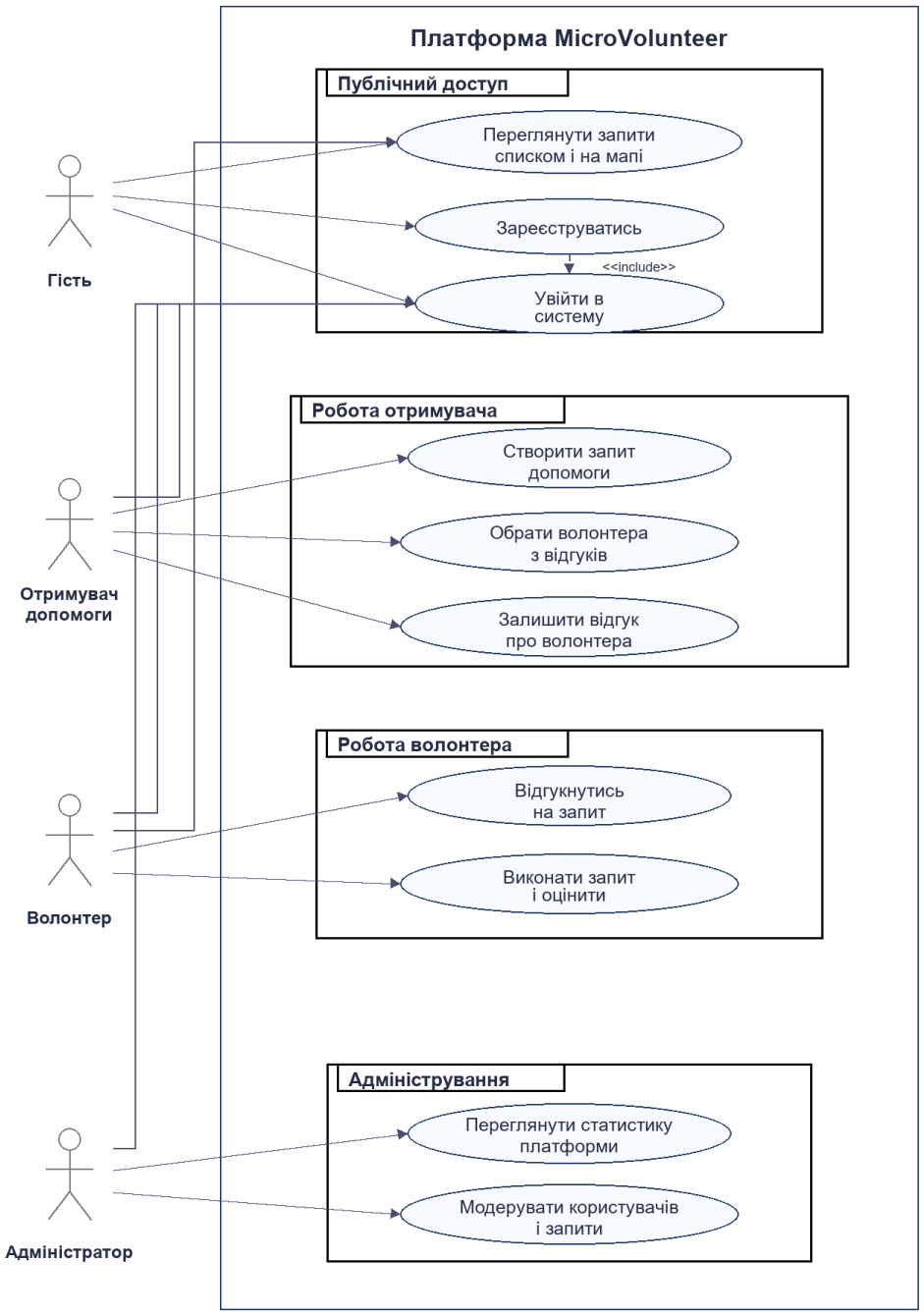


Рис. 1.1 Узагальнена діаграма варіантів використання MicroVolunteer

1.4 Формулювання функціональних і нефункціональних вимог

У системі визначено чотири ролі акторів. Гість може переглядати публічну карту та список запитів без реєстрації — це знижує поріг входу для потенційних волонтерів. Отримувач допомоги після реєстрації може створювати запити,

вказувати категорію, адресу, терміновість і опис, а також відстежувати їхній статус. Волонтер може переглядати запити з фільтрами (за категорією, геолокацією, терміновістю), відгукуватися на них і залишати відгуки після завершення. Адміністратор управляє довідниками категорій, має доступ до статистичного дашборду і може блокувати зловживання.

Функціональні вимоги наведено в таблиці 1.2. Вісім вимог охоплюють реєстрацію і ролі, створення і відображення запитів, карту, відгуки волонтерів, систему рейтингів, сповіщення, статистику та адміністрування. Формат опису вимог відповідає рекомендаціям ISO/IEC/IEEE 29148:2018 [43]: для кожної вимоги наведено унікальний ідентифікатор, формулювання у вигляді конкретної дії системи, рівень пріоритету за моделлю MoSCoW (Must-have, Should-have, Nice-to-have) і коротке обґрунтування. Це робиться для того, щоб кожна вимога була окремо верифіковною і простежуваною до конкретної функціональної прогалини, виявленої в підрозділах 1.2 і 1.3.

Функціональні вимоги до системи наведено в таблиці 1.2.

Таблиця 1.2

Функціональні вимоги до вебплатформи MicroVolunteer

ID	Вимога	Пріоритет	Обґрунтування
FR-01	Система повинна підтримувати реєстрацію, автентифікацію та розподіл користувачів за ролями «волонтер» і «отримувач допомоги».	Must-have	Рольова модель потрібна для розмежування сценаріїв і прав доступу.
FR-02	Отримувач допомоги повинен мати змогу створювати, переглядати, редагувати та скасовувати запити з категорією, терміновістю, адресою та описом.	Must-have	Це базовий сценарій фіксації потреби у структурованій формі.

Продовження таблиці 1.2

Функціональні вимоги до вебплатформи MicroVolunteer

FR-03	Система повинна відображати активні запити на інтерактивній карті та підтримувати фільтрацію за категорією, терміновістю й місцем.	Must-have	Вимога випливає з проблеми відсутності геолокаційної координації.
FR-04	Волонтер повинен мати змогу відгукнутися на запит, а отримувач — прийняти або відхилити відгук.	Must-have	Потрібно формалізувати перехід від відкритого запиту до виконання.
FR-05	Система повинна підтримувати двосторонні оцінки та текстові відгуки після завершення запиту.	Should-have	Репутаційний механізм підвищує прозорість взаємодії.
FR-06	Користувач повинен мати особистий кабінет з профілем, історією запитів або відгуків і базовою статистикою.	Should-have	Особистий кабінет забезпечує керування власною участю.
FR-07	Система повинна формувати сповіщення про нові відгуки, зміну статусу запиту, завершення, нові оцінки та релевантні запити поблизу.	Should-have	Сповіщення зменшують потребу вручну перевіряти сторінки системи.
FR-08	Адміністратор повинен мати доступ до статистичного дашборду з узагальненими показниками запитів, категорій і активності користувачів.	Nice-to-have	Аналітика потрібна для оцінювання навантаження і планування розвитку.

Нефункціональні вимоги (таблиці 1.3) визначають якість, з якою система має виконувати функціональні вимоги. Вимога безпеки передбачає захист від CSRF, XSS, SQL-ін'єкцій і брутфорсу паролів — це критично, оскільки система зберігає домашні адреси та персональні дані вразливих груп населення. Вимога продуктивності передбачає відповідь сервера до 500 мс при до 100 одночасних користувачів. Вимога доступності передбачає підтримку WCAG 2.1 рівня AA для охоплення людей з порушеннями зору чи моторики.

Таблиця 1.3

Нефункціональні вимоги до вебплатформи MicroVolunteer

ID	Категорія	Вимога	Метрика або критерій перевірки
NFR-01	Безпека	Система повинна зменшувати ризики CSRF, XSS, SQL-ін'єкцій, перебору паролів і несанкціонованого доступу до чужих об'єктів.	Використання CSRF-захисту, auto-escape, ORM, обмеження спроб входу, перевірки прав доступу; орієнтир — OWASP Top 10 [10].
NFR-02	Адаптивність	Інтерфейс повинен коректно працювати на настільних і мобільних екранах.	Перевірка основних сторінок на типових ширинах екрана; використання Bootstrap-підходів.
NFR-03	Розгортання	Система повинна запускатися в контейнеризованому середовищі.	Наявність Docker/Docker Compose конфігурації для вебзастосунку, бази даних і проксі.

Продовження таблиці 1.3

Нефункціональні вимоги до вебплатформи MicroVolunteer

ID	Категорія	Вимога	Метрика або критерій перевірки
NFR-04	Тестованість	Основні модулі повинні бути покриті автоматизованими тестами.	Наявність набору pytest-тестів; фактична кількість — 132 тести за контекстом проєкту.
NFR-05	Продуктивність	Типові сторінки та API-запити повинні відповідати без помітної затримки для користувача.	Орієнтовна ціль — час відповіді API менше 500 мс на типових запитах.
NFR-06	Доступність	Інтерфейс повинен враховувати потреби користувачів із різним рівнем цифрової грамотності та можливими обмеженнями.	Орієнтація на принципи WCAG: сприйнятність, керованість, зрозумілість, надійність [11].

Безпека визначена як пріоритетна нефункціональна вимога не формально, а через характер даних: система зберігає домашні адреси, розклад дня і соціальний статус потенційно вразливих людей. Компрометація цих даних могла б нести реальну загрозу їхній безпеці, тому захист реалізовано як багаторівневу властивість системи, а не як окрему функцію.

Таким чином, сформульовані вимоги безпосередньо відповідають виявленим проблемам і функціональним прогалинам аналогів — кожна з восьми функціональних вимог закриває конкретну проблему.

2 ЗАСОБИ РЕАЛІЗАЦІЇ ВЕБПЛАТФОРМИ

2.1 Середовище розробки та інструментальний ланцюжок

Для розроблення MicroVolunteer сформовано інструментальний ланцюжок, що охоплює три рівні: середовище розробки (IDE, Git, GitHub), мову та фреймворк серверної частини (Python 3.12 + Django 5.2.11), і контейнерний рівень (Docker + docker-compose).

Систему контролю версій побудовано на Git і GitHub. Git відстежує зміни на рівні рядків коду, що уможливлює повернення до будь-якого попереднього стану. GitHub слугує центральним сховищем і точкою для code review.

Контейнерний рівень на Docker і docker-compose необхідний з двох причин. По-перше, він гарантує відтворюваність середовища: будь-який розробник або CI/CD-піпелайн відтворить однакове оточення за однією командою docker-compose up [12, 13]. По-друге, він дозволяє перевіряти взаємодію Django, PostgreSQL і Nginx ще на локальній машині — не чекаючи деплою на сервер для виявлення проблем з nginx.conf чи database migrations.

2.2 Мова Python і вебфреймворк Django

Серверну частину MicroVolunteer реалізовано мовою Python 3.12 [14]. Вибір Python над альтернативами обумовлений конкретними технічними аргументами.

Порівняння Python, Node.js і PHP наведено в таблиці 2.1. Node.js має переваги для high-concurrency real-time застосунків (WebSocket, streaming), але MicroVolunteer не потребує цього: 99% взаємодій — класичний request/response. PHP залишається актуальним для CMS-систем, але його ORM-екосистема (Eloquent в Laravel) поступається Django ORM за декларативністю схеми і вбудованими

міграціями. Python із Django забезпечує найшвидшу реалізацію ORM-схеми, форм, адмін-панелі та автентифікації при найменшому обсязі коду.

Таблиця 2.1

Порівняльний аналіз Python, Node.js і PHP для серверної частини MicroVolunteer

Критерій	Python	Node.js	PHP
Екосистема вебфреймворків	Django, Flask, FastAPI; сильна підтримка повнофункціональних вебзастосунків	Express, NestJS, Fastify; сильна API- та SPA-екосистема	Laravel, Symfony; зріла екосистема класичних вебсайтів
Читабельність коду	Висока завдяки синтаксису й поширеним style guide	Залежить від архітектури та JavaScript/TypeScript-практик	Залежить від фреймворку й стилю проєкту
Робота з предметною логікою	Зручна для моделей, сервісів, тестів і скриптів адміністрування	Зручна для асинхронних API, але потребує окремих рішень	Зручна для CRUD-застосунків, особливо у Laravel/Symfony
Інтеграція з тестуванням	pytest, unittest, factory_boy, pytest-django	Jest, Mocha, Vitest	PHPUnit, Pest
Доцільність для MicroVolunteer	Висока: добре поєднується з Django, ORM і шаблонним SSR	Середня: доцільний для JS-орієнтованої SPA-архітектури	Середня: можливий варіант, але не дає переваг для обраної Django-архітектури

Django 5.2.11 обрано тому, що він є batteries-included фреймворком: містить готові компоненти для автентифікації (django.contrib.auth), адміністрування (django.contrib.admin), форм (django.forms), міграцій (django.db.migrations) і захисту (CSRF, XSS, clickjacking).

Порівняння Django, Flask і FastAPI наведено в таблиці 2.2. Flask є мінімалістичним мікрофреймворком і вимагає явного підключення кожного компонента (ORM, форми, автентифікація) [15]. FastAPI спроєктований для API-first сервісів і не має вбудованого шаблонізатора, що збільшує складність для SSR-застосунку [16]. Django є природним вибором для платформи, де є сторінки реєстрації, профілів, запитів, карти й адмін-панель.

Таблиця 2.2

Порівняльний аналіз Django, Flask і FastAPI

Критерій	Django 5.2.11	Flask	FastAPI
ORM	Вбудований Django ORM, міграції, зв'язки моделей	Немає вбудованого ORM; зазвичай додається SQLAlchemy	Немає власного ORM; інтегрується зі сторонніми рішеннями
Адміністративна панель	Вбудована Django Admin	Потребує сторонніх рішень	Потребує сторонніх рішень
Автентифікація	Вбудована система користувачів, сесій, permissions	Потребує додаткової конфігурації	Переважно реалізується через токени або сторонні пакети
Шаблони і SSR	Вбудована система templates	Jinja2, але архітектуру потрібно збирати самостійно	Основний фокус — API; SSR не є центральним сценарієм

Продовження таблиці 2.2

Порівняльний аналіз Django, Flask і FastAPI

Критерій	Django 5.2.11	Flask	FastAPI
Швидкість прототипування CRUD	Висока для ролей, форм, моделей і адмінки	Середня: більше ручного компонування	Висока для API, нижча для класичного HTML-застосунку
Доцільність для MicroVolunteer	Найвища: відповідає задачам ролей, CRUD, безпеки, тестування	Доцільний для малого сервісу, але потребує більше інтеграцій	Доцільний для API-first, але надлишковий для SSR-архітектури

Таким чином, вибір Python 3.12 + Django 5.2.11 обґрунтовано відповідністю функціональним вимогам: ORM покриває реляційну модель, вбудована автентифікація — вимогу ФВ-1, CSRF-захист — НФВ-1, а адмін-панель — вимогу ФВ-8, не вимагаючи додаткових залежностей.

2.3 СУБД PostgreSQL та ORM Django

PostgreSQL 16 обрано СУБД для MicroVolunteer через структуру предметної галузі: платформа оперує ієрархічно пов'язаними сутностями — User → Profile → HelpRequest → Response → Review → Notification — зі строгими обмеженнями цілісності. Реляційна модель природно описує ці зв'язки через зовнішні ключі, а PostgreSQL забезпечує їхнє виконання на рівні СУБД, а не лише коду [17].

Технічні переваги PostgreSQL 16 для MicroVolunteer такі: підтримка ACID-транзакцій (критично для операцій зміни статусу запиту — коли волонтер приймає запит, одночасно змінюється статус HelpRequest і створюється Response; якщо будь-яка з операцій завершиться помилкою, транзакція відкатується), розширені

типи даних (JSONB для ADR-метаданих), часткові індекси (для фільтрації активних запитів), ролі доступу (для розмежування прав між Django і read-only аналітичним підключенням).

Таблиця 2.3

Порівняльний аналіз PostgreSQL, MySQL і MongoDB

Критерій	PostgreSQL 16	MySQL	MongoDB
Модель даних	Реляційна, з розвиненими типами та обмеженнями	Реляційна, поширена у веброзробці	Документна, JSON-подібні документи
Транзакційність і цілісність	Сильна підтримка ACID, FOREIGN KEY, CHECK, UNIQUE	Підтримує транзакції залежно від рушія	Підтримує транзакції, але модель орієнтована на документи
Відповідність домену MicroVolunteer	Висока: багато зв'язків 1:1, 1:N, M:N	Висока для базових CRUD-сценаріїв	Потрібно додатково контролювати зв'язки
Агрегації та статистика	Розвинені SQL-агрегації та індекси	Добра підтримка SQL-агрегацій	Потужний aggregation pipeline, але інша модель
Інтеграція з Django ORM	Повна й типова для production-середовища	Підтримується	Потребує нетипової інтеграції або відмови від стандартного ORM

Продовження таблиці 2.3

Порівняльний аналіз PostgreSQL, MySQL і MongoDB

Критерій	PostgreSQL 16	MySQL	MongoDB
Доцільність для проєкту	Найвища для реляційної структури MicroVolunteer	Можлива альтернатива	Доцільна для документних сценаріїв, але не для поточної ER-моделі

Як видно з таблиці 2.3, PostgreSQL найкраще відповідає структурі MicroVolunteer. MySQL є поширеним вибором, але поступається PostgreSQL за підтримкою складних запитів і JSON-типів. MongoDB є документоорієнтованою СУБД, що не відповідає реляційній природі системи: зв'язки між User, HelpRequest і Review були б реалізовані через ручні joins у коді, а не через декларативні FK-обмеження. У середовищі тестування використовується SQLite, оскільки pytest-django [18] автоматично створює in-memory базу при кожному запуску — це прискорює цикл тестування до 9,14 с для 132 тестів.

2.4 Засоби клієнтської частини

Клієнтську частину MicroVolunteer реалізовано за стратегією server-side rendering (SSR) із використанням Django Templates. Це означає, що HTML-сторінки формуються повністю на сервері й надсилаються браузеру вже готовими — на відміну від SPA-підходу, де браузер отримує порожній HTML і наповнює його через JavaScript. Вибір SSR обумовлений тим, що MicroVolunteer не має складних UI-станів, які виправдовували б складність React або Vue, і що SSR природно підтримується Django без додаткових залежностей.

Для адаптивної верстки обрано Bootstrap 5. Він забезпечує 12-колонкову сітку, компоненти (картки, форми, модальні вікна, Toast-сповіщення) та утилітарні

класи, що дозволили реалізувати весь UI MicroVolunteer без написання власного CSS для базових елементів, скоротивши час реалізації інтерфейсу.

Таблиця 2.4

Порівняльний аналіз Leaflet, Google Maps і Mapbox для картографічного модуля

Критерій	Leaflet 1.9.4	Google Maps	Mapbox
Ліцензійна й фінансова модель	Відкрита JavaScript-бібліотека; тайли підключаються окремо	Комерційний API з умовами використання та квотами	Комерційна платформа з безкоштовними лімітами та тарифами
Контроль над інтерфейсом	Високий: бібліотека легка й гнучка	Значний, але прив'язаний до екосистеми Google	Високий, особливо для кастомних стилів
Доцільність для дипломного проєкту	Висока: не потребує складної комерційної інтеграції	Середня: потрібні ключі, налаштування білінгу та політик	Середня: потужний інструмент, але надлишковий для базової карти
Інтеграція з Django Templates	Проста через JS і HTML-шаблон	Проста, але з API-ключем	Проста, але з токеном доступу
Відповідність MicroVolunteer	Найвища: потрібні маркери, шар карти й події без важкої GIS-логіки	Можлива альтернатива	Можлива альтернатива для розширеної візуалізації

Картографічний модуль реалізовано на базі Leaflet.js 1.9.4 з тайлами CartoDB Voyager (ADR-003). Порівняння Leaflet, Google Maps і Mapbox наведено в таблиці 2.4. Google Maps і Mapbox потребують реєстрації API-ключа з прив'язкою платіжного методу, тоді як Leaflet є повністю відкритою бібліотекою без ключа.

Для автодоповнення адрес використано Nominatim API (OpenStreetMap) відповідно до Nominatim Usage Policy, що дозволяє до 1 запиту на секунду для некомерційних застосунків [19, 20].

2.5 Контейнеризація і серверна інфраструктура

Для контейнеризації MicroVolunteer використано Docker і docker-compose. Docker-образ сервісу web будується на python:3.12-slim, встановлює залежності з requirements.txt і запускає Gunicorn.

Gunicorn використовується як WSGI-сервер для запуску Django-застосунку в production [21]. Django розробний сервер (runserver) не підходить для production, оскільки є однопоточним і не має захисту від перевантажень. Gunicorn запускає кілька воркерів (типово 2 × CPU + 1), кожен з яких обробляє HTTP-запити незалежно. Nginx стоїть перед Gunicorn як reverse proxy [22]: статичні файли Nginx обслуговує самостійно, не передаючи запити Django; Nginx буферизує повільних клієнтів, захищаючи Gunicorn від slow-client атак; Nginx виконує TLS-завершення, і це означає, що Django-код не потребує обробки шифрування безпосередньо.

Додатковою перевагою docker-compose є розмежування конфігурацій. Локальна розробка використовує settings/development.py з SQLite і DEBUG=True; production запускається з settings/production.py з PostgreSQL, DEBUG=False і Nginx. Переключення між ними відбувається через змінну середовища DJANGO_SETTINGS_MODULE, що задається у docker-compose.override.yml — і випадковий запуск production-налаштувань у локальному середовищі унеможливлений на рівні конфігурації.

3 ПРОЄКТУВАННЯ СИСТЕМИ MICROVOLUNTEER

Проєктування системи MicroVolunteer виконано з урахуванням вимог, сформульованих у попередніх розділах, а також особливостей предметної області локальної соціальної підтримки. Архітектура має підтримувати чіткий поділ відповідальності між модулями, контроль доступу, роботу з геоданими, транзакційну цілісність і можливість подальшого розгортання у серверному середовищі [23, 24].

3.1 Загальна архітектура вебзастосунку

MicroVolunteer спроєктовано як серверний вебзастосунок на базі Django з фізичною схемою: Browser → Nginx → Gunicorn → Django → PostgreSQL. Кожен рівень цього ланцюжка має чітко визначену відповідальність: Nginx приймає HTTPS-запити і або обслуговує статику самостійно, або проксіює до Gunicorn; Gunicorn розподіляє запити між Python-воркерами; Django обробляє бізнес-логіку і звертається до PostgreSQL через ORM.

На рівні прикладної архітектури застосовано парадигму Model-Template-View (MTV), характерну для Django. Model відповідає за опис структури даних і бізнес-правила — обмеження, валідація, зв'язки між сутностями. Template відповідає за генерацію HTML — у MicroVolunteer використовуються Django-шаблони з блоками наслідування ({% extends %} / {% block %}). View отримує HTTP-запит, звертається до моделей і повертає відрендерений шаблон або HTTP-відповідь.

Компонентну діаграму архітектури подано на рис. 3.1. Вона показує розмежування рівнів і точки взаємодії між компонентами: браузер звертається до Nginx по HTTPS, Nginx проксіює до Gunicorn по HTTP (всередині docker-мережі), Django звертається до PostgreSQL через TCP.

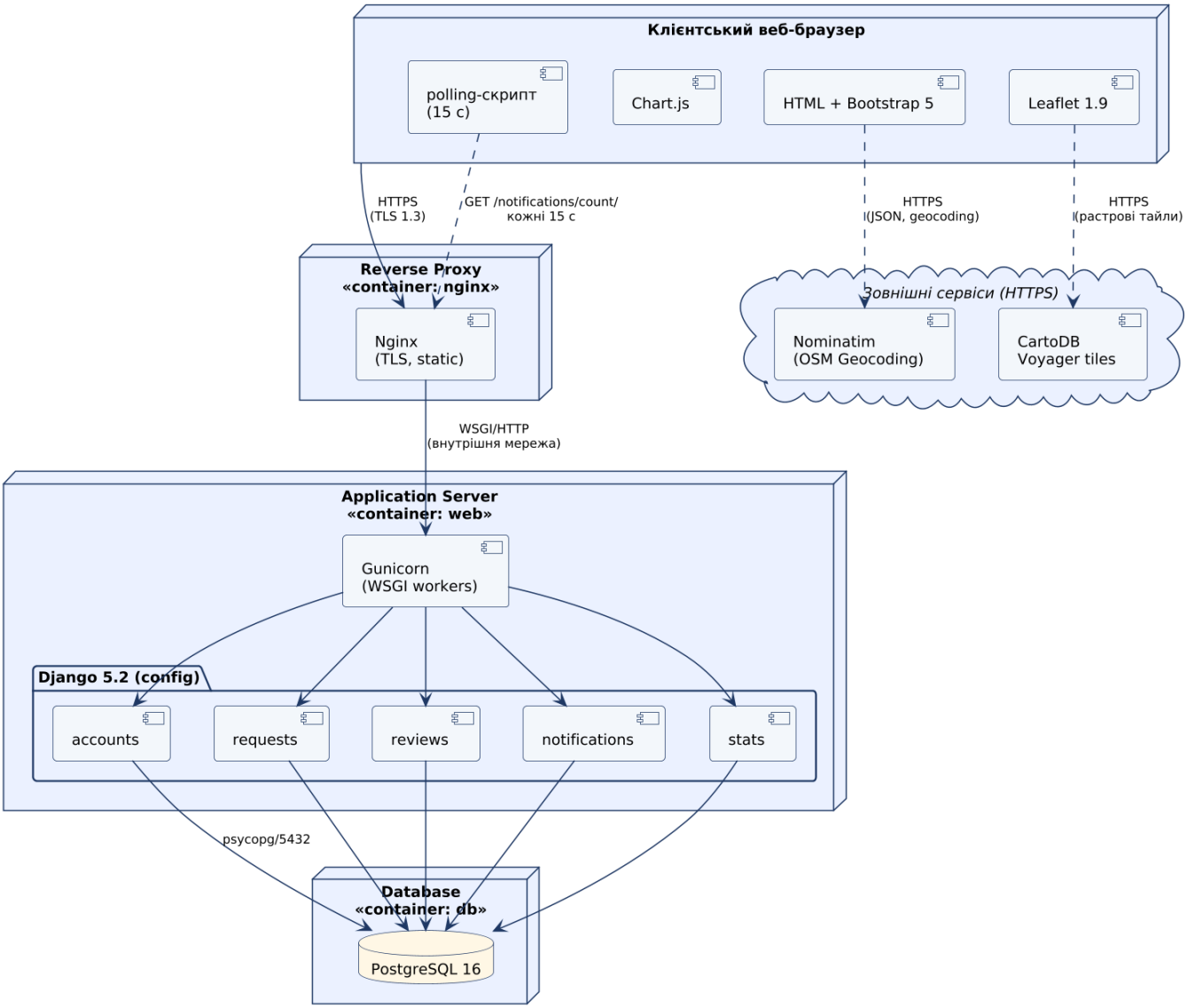


Рис. 3.1 Компонентна діаграма архітектури MicroVolunteer

Для динамічних сценаріїв — оновлення лічильника сповіщень — застосовано lightweight polling: клієнтська частина кожні 10–15 секунд надсилає GET /notifications/count/ і отримує {"count": N}. Це рішення зафіксовано в ADR-010 як компроміс: SSE (Server-Sent Events) забезпечували б справжню реал-тайм поведінку [25], але при 50 одночасних підключеннях могли насичувати Gunicorn worker-pool, оскільки кожне SSE-з'єднання утримує воркер зайнятим протягом усього часу сесії.

3.2 Модульна структура Django-проєкту

Структуру MicroVolunteer організовано як набір п'яти Django-застосунків з чітким розподілом відповідальності: accounts, requests, reviews, notifications, stats. Такий поділ відповідає принципу єдиної відповідальності на рівні модулів: кожен застосунок відповідає за одну предметну підсистему і не вирішує задачі сусідніх.

Застосунок accounts відповідає за весь user lifecycle: реєстрацію, вхід, вихід, редагування профілю. Кастомна модель User розширює AbstractUser, зберігаючи роль у полі user_type (CharField з choices UserType), а is_volunteer і is_recipient реалізовані як @property — не поля бази даних. Роль визначає доступні сценарії взаємодії. Після створення User сигнал post_save автоматично створює відповідний VolunteerProfile або RecipientProfile — це унеможливлює стан, коли User існує без профілю, навіть при переривання транзакції.

Застосунок requests є серцевиною системи: він містить 64 з 132 автоматизованих тестів і реалізує найбільш складну бізнес-логіку — машину станів запиту (ACTIVE → IN_PROGRESS → COMPLETED / CANCELLED / EXPIRED) і геолокаційний пошук через інтеграцію з Nominatim. Кожен стан переходить до наступного лише через дозволені дії конкретної ролі — це зафіксовано в бізнес-правилах і перевіряється у views, повертаючи HTTP 403 з детальним повідомленням при порушенні.

Застосунок reviews реалізує систему рейтингів: отримувач оцінює волонтера після завершення запиту. Відокремлення reviews від requests обумовлено тим, що відгук є наслідком завершеної взаємодії і може існувати незалежно від самого запиту — наприклад, при архівуванні запиту відгуки зберігаються, що дозволяє накопичувати репутаційну історію волонтера.

Застосунок notifications реалізує подієво-орієнтовану модель сповіщень: шість receiver-функцій перехоплюють сигнали Django (post_save, post_delete) і створюють Notification-об'єкти. Це означає, що код views і models не містить жодного виклику «надіслати сповіщення» — цей аспект повністю інкапсульований

у notifications.receivers, що забезпечує decoupled event propagation і спрощує тестування суміжних модулів.

Застосунок stats надає адміністраторам агреговані дані через Chart.js [26]: розподіл запитів за статусами, середній рейтинг по категоріях і таблиця топ-5 користувачів за кількістю отриманих відгуків. Всі запити виконуються ледачою агрегацією на рівні ORM (annotate, aggregate) — без кешування для MVP, але з можливістю додавання Redis-кешу при зростанні навантаження. Конфігураційний пакет config поділено на base.py, development.py і production.py — відповідно до рекомендацій Django Deployment Checklist, що унеможливлює випадковий запуск production-налаштувань у локальному середовищі.

3.3 Проєктування бази даних

Реляційну модель бази даних MicroVolunteer спроєктовано навколо центральної сутності HelpRequest, до якої прив'язані всі інші сутності системи. Це відповідає предметній логіці: запит є одиницею координації, навколо якої обертаються і волонтер (Response), і відгук (Review), і сповіщення (Notification) [27].

Модель User розширює AbstractUser через успадкування, а не через Profile-only підхід. Це дозволяє використовувати стандартні механізми Django (authenticate(), login(), @login_required) без змін, тоді як додаткові атрибути (user_type, phone, city) є частиною тієї ж ORM-моделі. Роль зберігається у полі user_type (CharField з choices UserType), а is_volunteer і is_recipient є @property-властивостями, а не полями БД — це спрощує перевірки в шаблонах і views без join.

Модель HelpRequest містить сім основних полів: recipient (FK → User), category (FK → Category), title, description, address, status (CharField з choices) і urgency. Поле location зберігає текстовий рядок адреси замість PostGIS-координат

— це архітектурне рішення ADR-003: для MVP достатньо текстового пошуку і тайлів Leaflet, а PostGIS вводить залежність від складного розширення PostgreSQL.

Обмеження unique_together = ('help_request', 'volunteer') у моделі Response запобігає подвійному відгуку від одного волонтера на один запит на рівні бази даних, а не лише коду. Це важливо, оскільки паралельні HTTP-запити можуть надійти одночасно, і перевірка лише в коді view не гарантує атомарності — лише constraint на рівні СУБД є надійним захистом від race condition.

ER-діаграму бази даних подано на рис. 3.2. Основні моделі і кардинальності зведено в таблиці 3.1.

Таблиця 3.1

Моделі бази даних MicroVolunteer та основні кардинальності

Модель	Призначення	Основні зв'язки	Кардинальність
User	Базовий користувач системи з роллю волонтера або отримувача	VolunteerProfile, RecipientProfile, HelpRequest, Response, Review, Notification	1:1 з профілем; 1:N із запитами, відгуками, сповіщеннями
VolunteerProfile	Дані волонтера, категорії допомоги, радіус, рейтинг	User, Category	1:1 з User; M:N з Category
RecipientProfile	Дані отримувача допомоги та контактна інформація	User	1:1 з User

Продовження таблиці 3.1

Моделі бази даних MicroVolunteer та основні кардинальності

Модель	Призначення	Основні зв'язки	Кардинальність
Category	Довідник категорій допомоги	HelpRequest, VolunteerProfile	1:N із HelpRequest; M:N з VolunteerProfile
HelpRequest	Запит отримувача на локальну допомогу	User, Category, Response, Review, Notification	N:1 до User і Category; 1:N до відповідей, відгуків і сповіщень
Response	Відгук волонтера на запит	HelpRequest, User	N:1 до HelpRequest; N:1 до User; унікальна пара запит–волонтер
Review	Оцінювання користувача після завершення запиту	HelpRequest, User author, User target	N:1 до запиту та користувачів; унікальна пара запит–автор
Notification	Повідомлення користувача про події системи	User, HelpRequest через related_request	N:1 до User; N:1 до HelpRequest nullable

ER-діаграму бази даних подано на рис. 3.2. Вона відображає первинні ключі, зовнішні ключі та кардинальності зв'язків між вісьма сутностями системи.

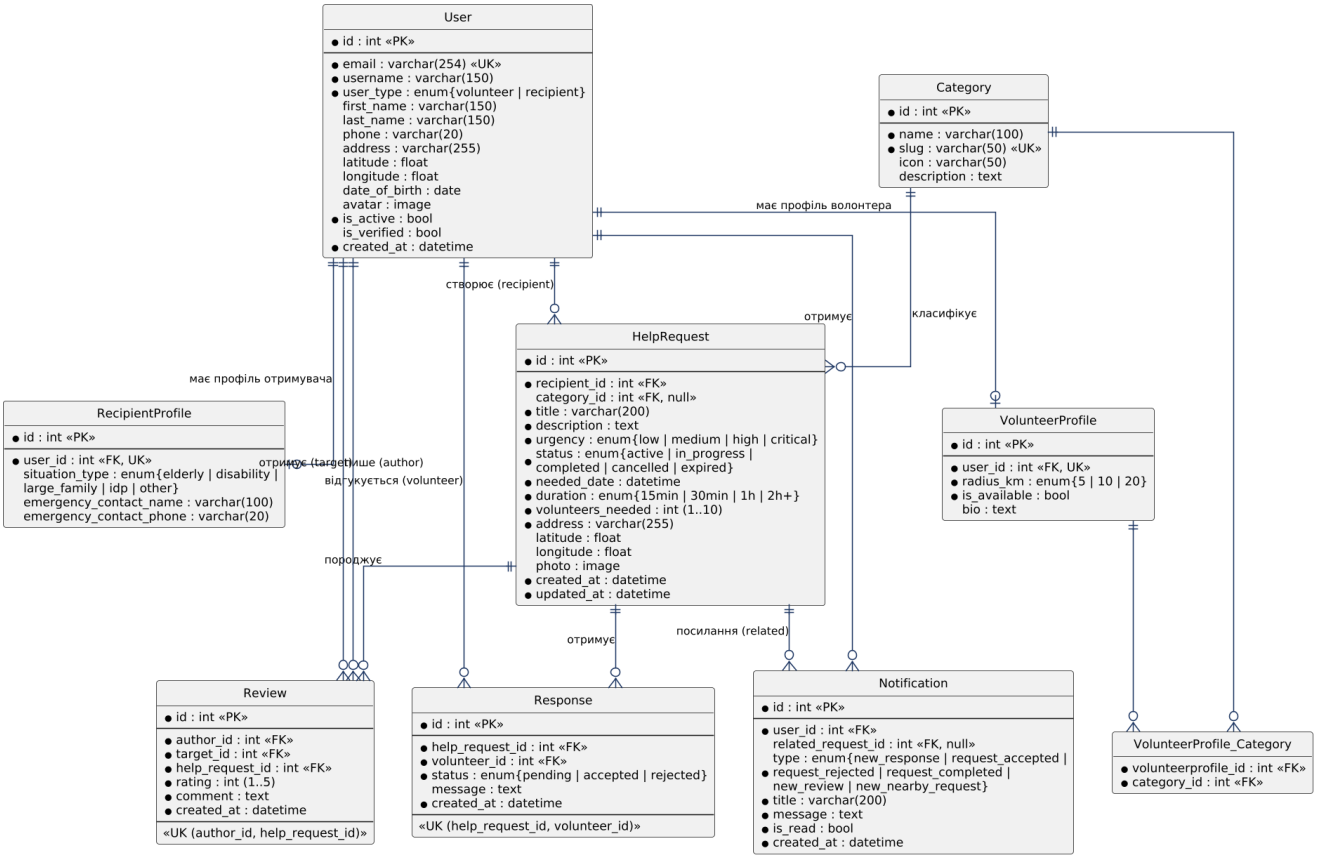


Рис. 3.2 ER-діаграма бази даних MicroVolunteer

Для забезпечення цілісності даних застосовано первинні та зовнішні ключі, унікальні обмеження і обмеження NOT NULL — вони виконуються на рівні СУБД і не можуть бути оминуті помилкою в коді застосунку [17].

3.4 Варіанти використання системи

Варіанти використання MicroVolunteer визначаються чотирма акторами, чиї ролі є взаємовиключними за можливостями, але не за ідентичністю: один і той самий User може бути одночасно отримувачем і волонтером залежно від контексту, але не в межах одного запиту.

Use-case діаграму системи подано на рис. 3.3 та рис 3.4

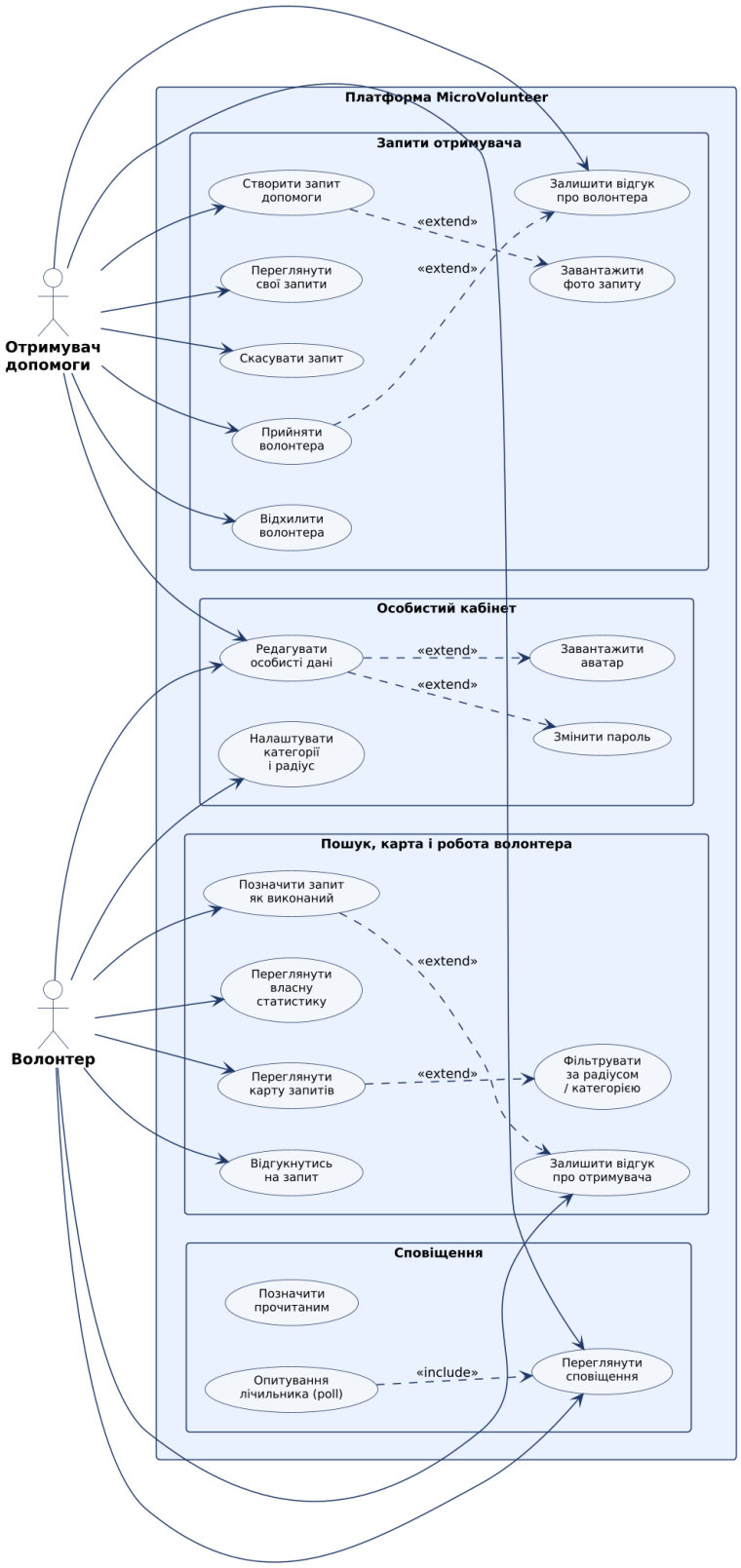


Рис. 3.3 Use-case діаграма системи MicroVolunteer (актори: Отримувач допомоги та Волонтер)

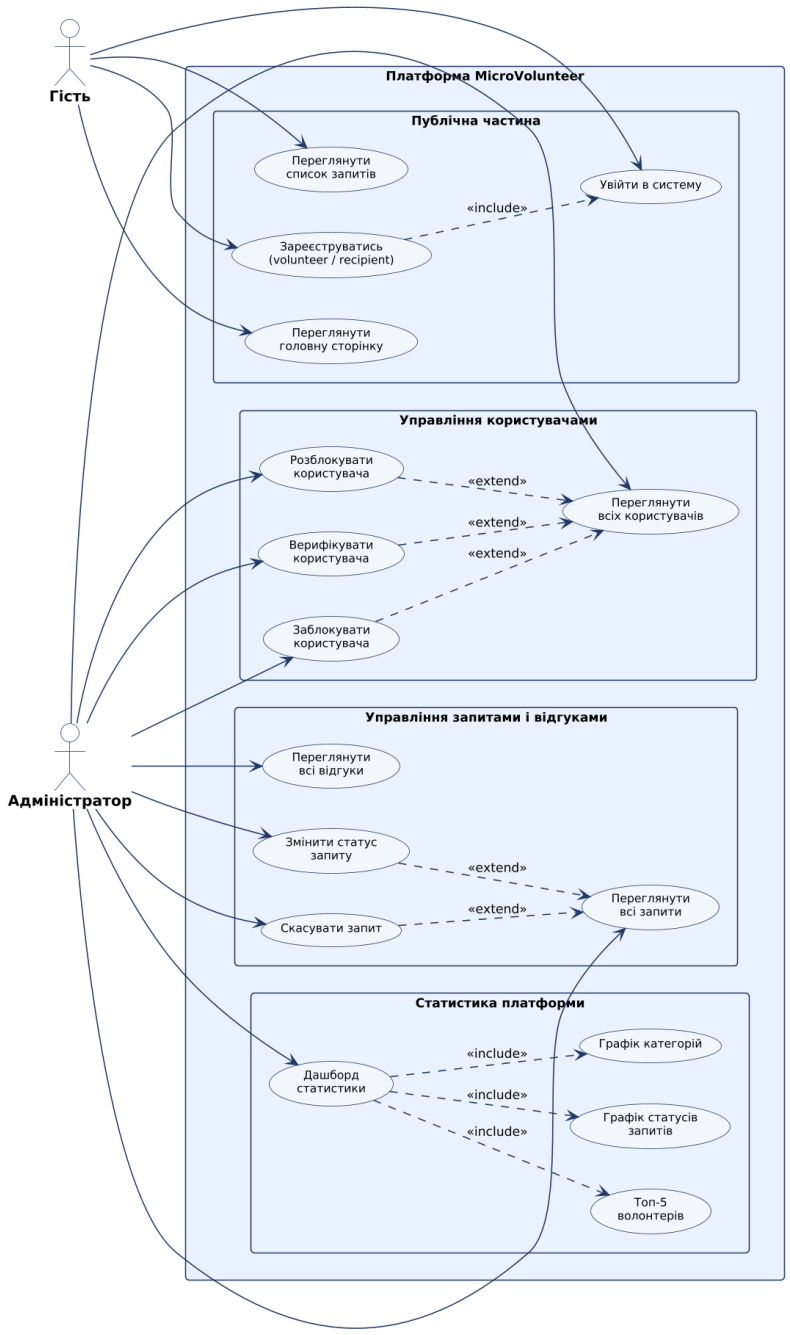


Рис. 3.4 Use-case діаграма системи MicroVolunteer (актори: Отримувач Гість та Адмінітсратор)

Центральним бізнес-сценарієм є життєвий цикл запиту: отримувач створює запит → запит отримує статус OPEN → один або кілька волонтерів відгукуються → отримувач обирає одного → статус змінюється на ACCEPTED → обидва учасники оцінюють одне одного → статус змінюється на COMPLETED. Побічні сценарії включають відхилення відгуку (статус REJECTED для конкретного Response), скасування запиту отримувачем (статус CANCELLED для HelpRequest)

і адміністративне видалення зловживань. Кожен з цих сценаріїв супроводжується відповідним сповіщенням через notifications.receivers.

3.5 Об'єктна модель і ключові сценарії взаємодії

Об'єктна модель MicroVolunteer ґрунтується на тих самих доменних сутностях, що й реляційна модель, але акцент переходить від структури (поля, типи даних) до поведінки (методи, сигнали, бізнес-правила). Діаграму класів предметної галузі подано на рис. 3.5.

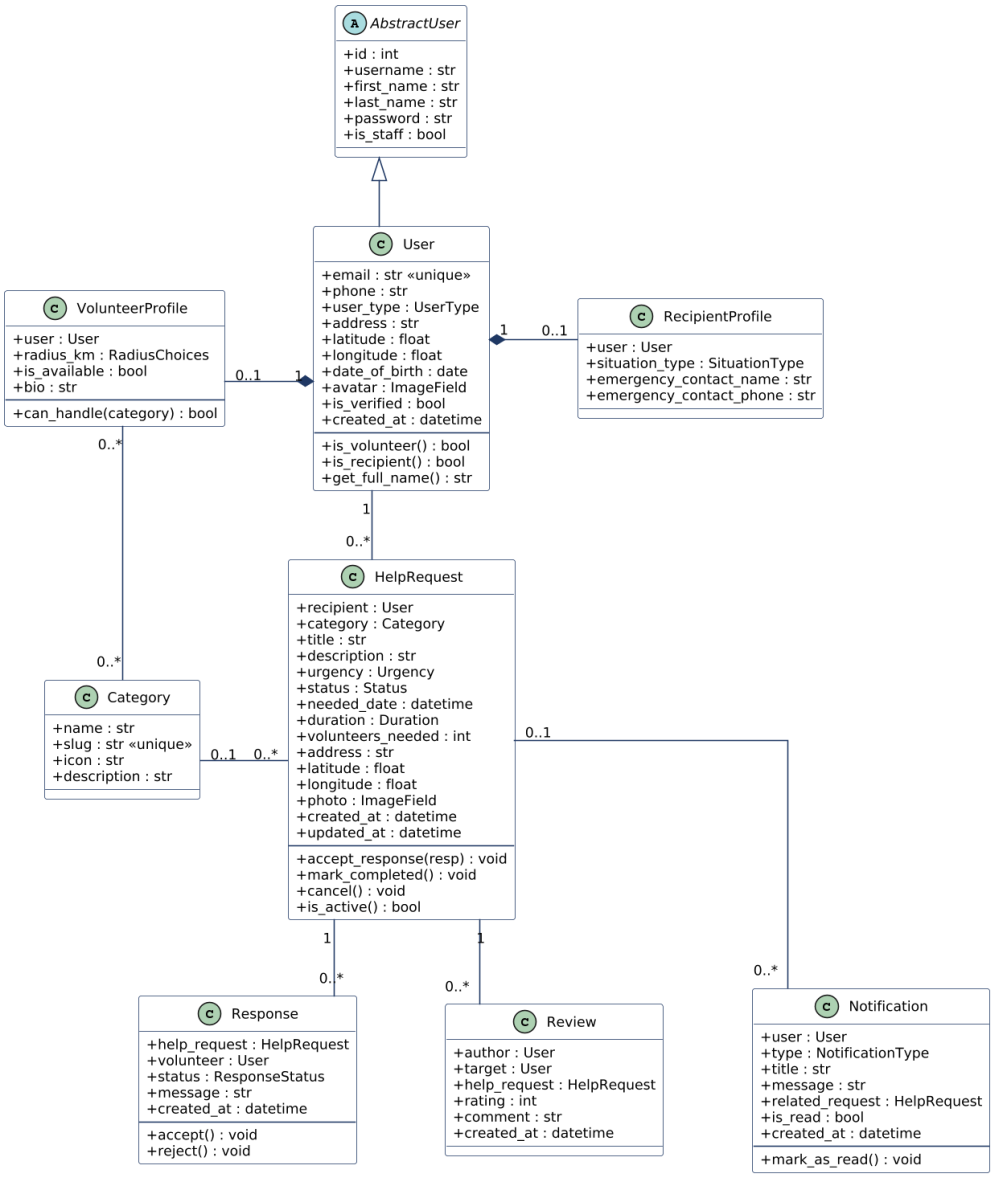


Рис. 3.5 Діаграма класів предметної області MicroVolunteer

Ключовим сценарієм взаємодії є «волонтер відгукується на запит». Sequence-діаграму цього сценарію подано на рис. 3.6. Волонтер відкриває сторінку активного запиту → система перевіряє через Response.objects.filter(help_request=req, volunteer=user).exists() чи він вже відгукувався → якщо ні, створюється новий Response зі статусом PENDING → сигнал post_save запускає receiver, який створює Notification для отримувача → клієнт через polling GET /notifications/count/ отримує оновлений лічильник.

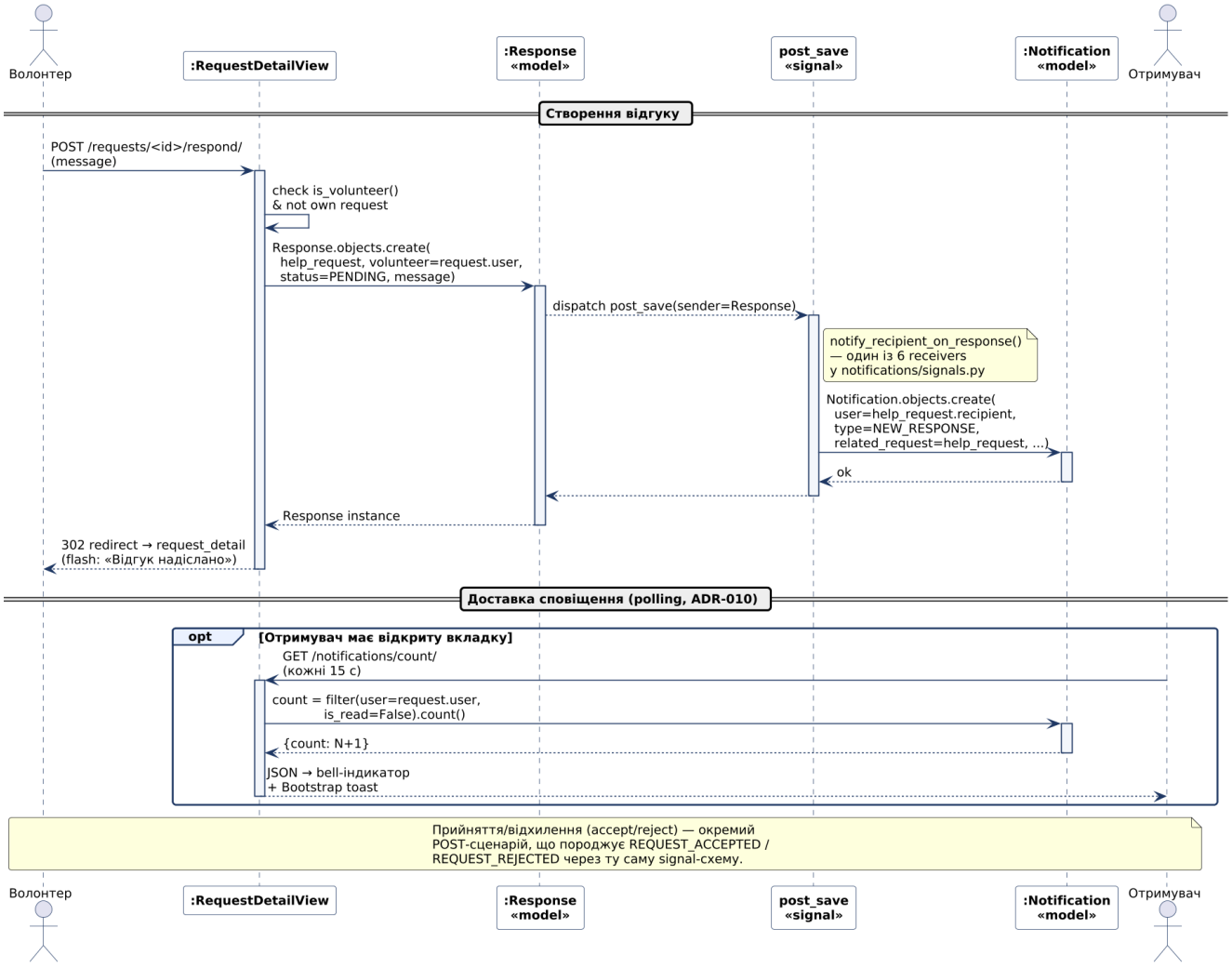


Рис. 3.6 Sequence-діаграма сценарію «волонтер відгукується на запит»

Django-сигнали реалізують decoupled event propagation: відгук на запит (Response) і створення сповіщення (Notification) розмежовані. Це означає, що код view для Response не знає нічого про Notification — він лише зберігає Response, а signals.py автоматично поширює подію. Таке розмежування спрощує тестування: тести accounts і requests не потребують заглушок для notifications.

3.6 Ключові архітектурні рішення (Architecture Decision Records)

Для документування ключових технічних рішень використано формат Architecture Decision Records (ADR). ADR фіксує три аспекти: контекст (чому рішення взагалі потрібно), саме рішення і обґрунтування (чому саме таке), а також наслідки (які переваги і які trade-off). Це важливо для підтримуваності: через три місяці нескладно забути, чому саме Nominatim, а не Google Places API — але ADR відповідає на це питання.

Таблиця 3.2

Зведення актуальних архітектурних рішень MicroVolunteer

ADR	Контекст	Рішення	Наслідки	Альтернативи
ADR-002	Потрібен адресний автокомпліт без платного API	Використано Nominatim OSM з обмеженням countrycodes=ua	Без API-ключа, але потрібно дотримуватися політики використання [20]	Google Places, Mapbox Geocoding
ADR-003	Raw OSM tiles давали проблеми доступу	Використано CartoDB Voyager tiles	Стабільніше відображення карти в локальному середовищі	Raw OSM tiles, Mapbox tiles

Продовження таблиці 3.2

Зведення актуальних архітектурних рішень MicroVolunteer

ADR	Контекст	Рішення	Наслідки	Альтернативи
ADR-004	Потрібно не показувати точну адресу отримувача	Зсув координат приблизно на 150 м для карти	Підвищення приватності, менша точність публічної точки	Приховати карту; показувати точні координати всім
ADR-005	Потрібно обмежити спам-запити	MAX_ACTIVE_REQUESTS = 10 на отримувача	Зменшення зловживань і навантаження	Не обмежувати; ручна модерація
ADR-006	Потрібно редагувати User і профіль разом	Dual-form pattern із prefix для профільної форми	Узгоджене збереження двох пов'язаних форм	Окремі сторінки редагування
ADR-007	Django 5.x не підтримує безпечний GET-logout	POST-only logout	Захист від небажаного logout через GET	GET logout, кастомний endpoint
ADR-008	Потрібна краща робота браузерних менеджерів паролів	Додано autocomplete attributes	Зручніша автентифікація та менше попереджень браузера	Не задавати атрибути

Продовження таблиці 3.2

Зведення актуальних архітектурних рішень MicroVolunteer

ADR	Контекст	Рішення	Наслідки	Альтернативи
ADR-009	Тип NEW_NEARBY_REQUEST існував без реального matching	Matching волонтерів за категоріями, радіусом і адресою	Сповіщення стають предметно корисними	Розсилати всім волонтерам; не розсилати
ADR-010	SSE утримував довгі з'єднання без реальної переваги	Lightweight polling GET /notifications/count/ кожні 15 с	Простіша й стабільніша схема для WSGI	SSE, WebSocket, ручне оновлення
ADR-011	Інтерфейс потребував узгоджених breakpoint'ів	Bootstrap-aligned breakpoints 576/768/992/1200 і utility refactor	Краща адаптивність, touch targets 44 px	Набір неузгоджених кастомних breakpoint'ів
ADR-012	Цільова аудиторія потребує доступнішого інтерфейсу	Accessibility baseline і панель «Зручність»	Клавіатурна навігація, високий контраст, large text, calm mode	Third-party overlay; мінімальні правки CSS

Зведення актуальних архітектурних рішень MicroVolunteer наведено в таблиці 3.2. Ключовим є ADR-010 (lightweight polling замість SSE), прийнятий після експерименту з SSE: при 50 одночасних підключеннях SSE насичувало Gunicorn worker-pool, оскільки кожне з'єднання блокувало воркер протягом усієї сесії. Polling із 10-15-секундним інтервалом збільшує затримку сповіщення, але знімає ризик resource exhaustion при паралельних сесіях.

4 РЕАЛІЗАЦІЯ, ТЕСТУВАННЯ ТА РОЗГОРТАННЯ ВЕБПЛАТФОРМИ

Розділ описує практичну реалізацію п'яти модулів MicroVolunteer, механізми безпеки, автоматизоване і функціональне тестування та контейнеризоване розгортання.

4.1 Опис реалізованих модулів системи

Реалізована платформа MicroVolunteer має модульну структуру, де кожний Django-застосунок є самостійною одиницею з власним набором моделей, views, форм і тестів. Нижче описано кожен з п'яти модулів з акцентом на ключових технічних рішеннях і їхньому обґрунтуванні.

4.1.1 Модуль автентифікації та профілів

Модуль accounts реалізовано у застосунку apps/accounts. Ядром модуля є кастомна модель User, що успадковує AbstractUser і зберігає роль у полі user_type = CharField(choices=UserType). Властивості is_volunteer та is_recipient реалізовані як @property і не є полями бази даних. Такий підхід дозволяє використовувати стандартні механізми Django (authenticate, login, @login_required, @permission_required) без змін [28, 29] — роль перевіряється через user.is_volunteer у шаблонах і views.

Реєстрація розділена на два кроки залежно від ролі: форма реєструє User і одночасно встановлює is_volunteer=True або False, після чого сигнал post_save автоматично створює відповідний профіль (VolunteerProfile або RecipientProfile). Це виключає можливість стану «User без профілю» — навіть якщо реєстрація переривається на проміжному кроці, транзакція відкатиться і обидва об'єкти не збережуться.

Редагування профілю реалізовано через ModelForm із завантаженням аватару. Аватар зберігається через FileField з валідацією розміру та формату на рівні форми — це запобігає завантаженню виконуваних файлів під виглядом зображень. Демонстраційні матеріали модуля (форма реєстрації, сторінка входу, профіль, форма редагування) подано на рисунках 4.1–4.4.

Демонстраційні матеріали до модуля автентифікації наведено на рисунках 4.1–4.4.

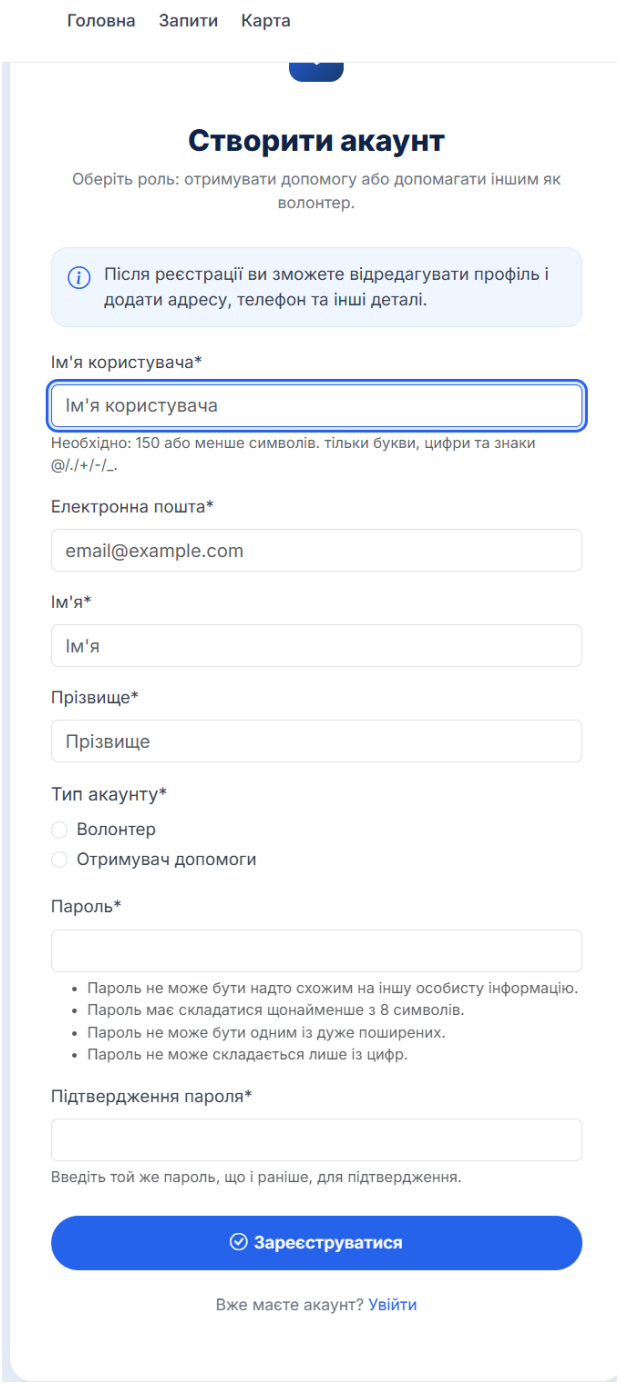


Рис. 4.1 Форма реєстрації користувача

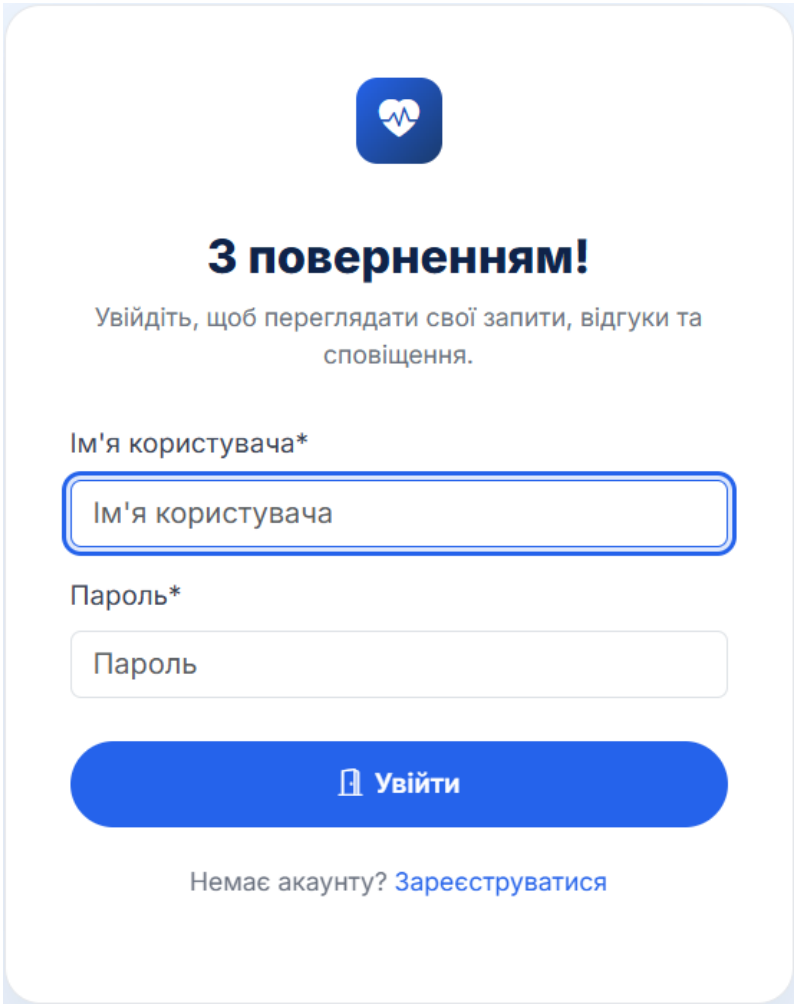


Рис. 4.2 Сторінка входу до системи

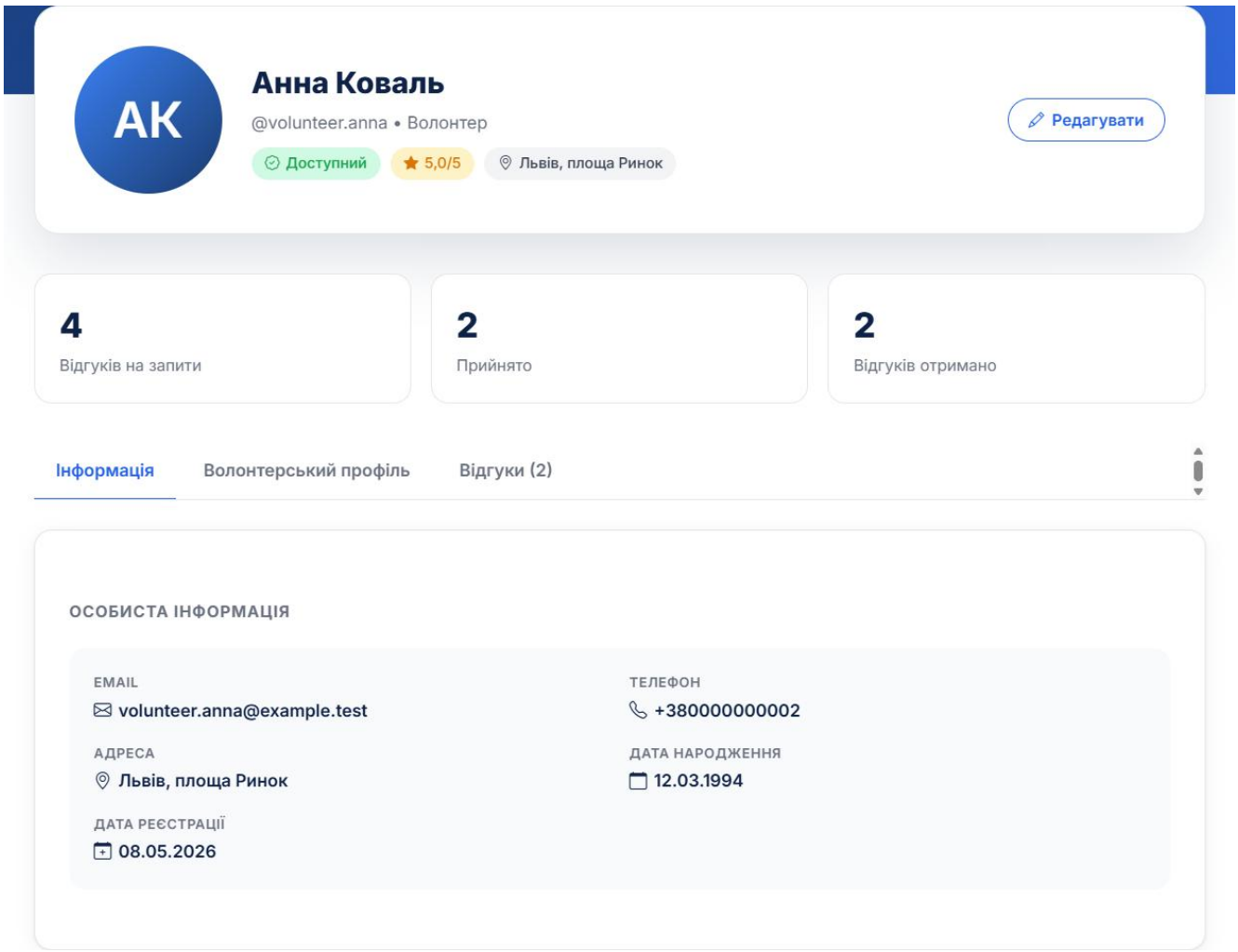


Рис. 4.3 Сторінка профілю користувача

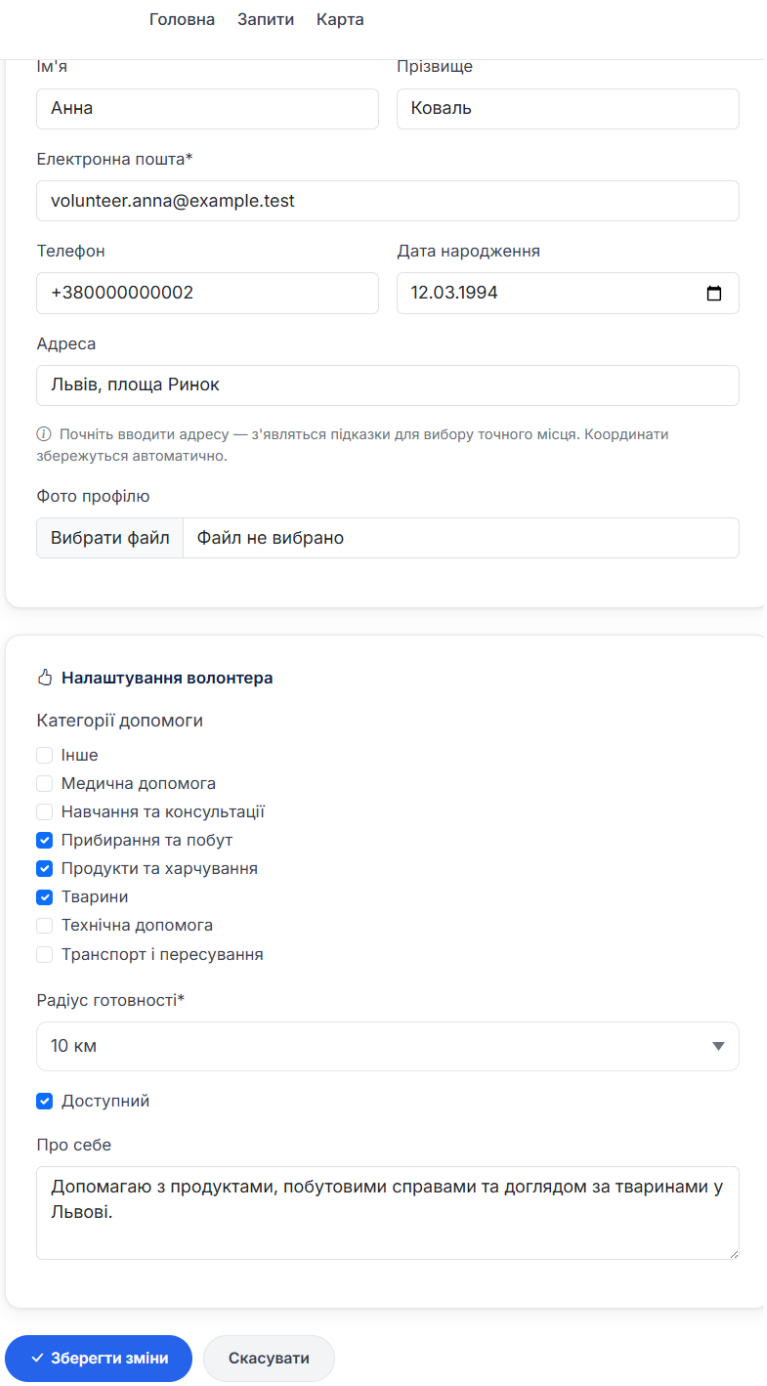


Рис. 4.4 Форма редагування профілю

4.1.2 Модуль запитів та інтерактивної карти

Модуль requests реалізовано у застосунку apps/requests. Це центральний модуль системи: він містить 64 з 132 автоматизованих тестів і реалізує найбільш складну бізнес-логіку — машину станів запиту і геолокаційний пошук.

Машина станів HelpRequest реалізована через CharField зі choices і набір view-функцій, що перевіряють допустимість переходу: ACTIVE → IN_PROGRESS дозволяється лише власнику запиту після вибору конкретного волонтера; IN_PROGRESS → COMPLETED — після підтвердження завершення; будь-який стан → CANCELLED — лише власником або адміністратором. Стани ACCEPTED, REJECTED, PENDING належать моделі Response, а не HelpRequest. Перевірки здійснюються у views, а не в моделі — це дозволяє повертати HTTP 403 з детальним повідомленням замість виключення Django на рівні ORM.

Інтерактивну карту реалізовано на Leaflet.js 1.9.4 з тайлами CartoDB Voyager. При завантаженні сторінки JavaScript отримує GeoJSON-дані активних запитів через GET /requests/map/data/ і відображає маркери з popup-картками. Для автодоповнення адрес під час створення запиту і редагування профілю Nominatim викликається напряму з браузера — відповідний код знаходиться в templates/requests/request_form.html і templates/accounts/profile_edit.html. Демонстраційні матеріали (список, форма, деталі, карта, автодоповнення) подано на рисунках 4.5–4.9.

Демонстраційні матеріали до модуля запитів наведено на рисунках 4.5–4.9.

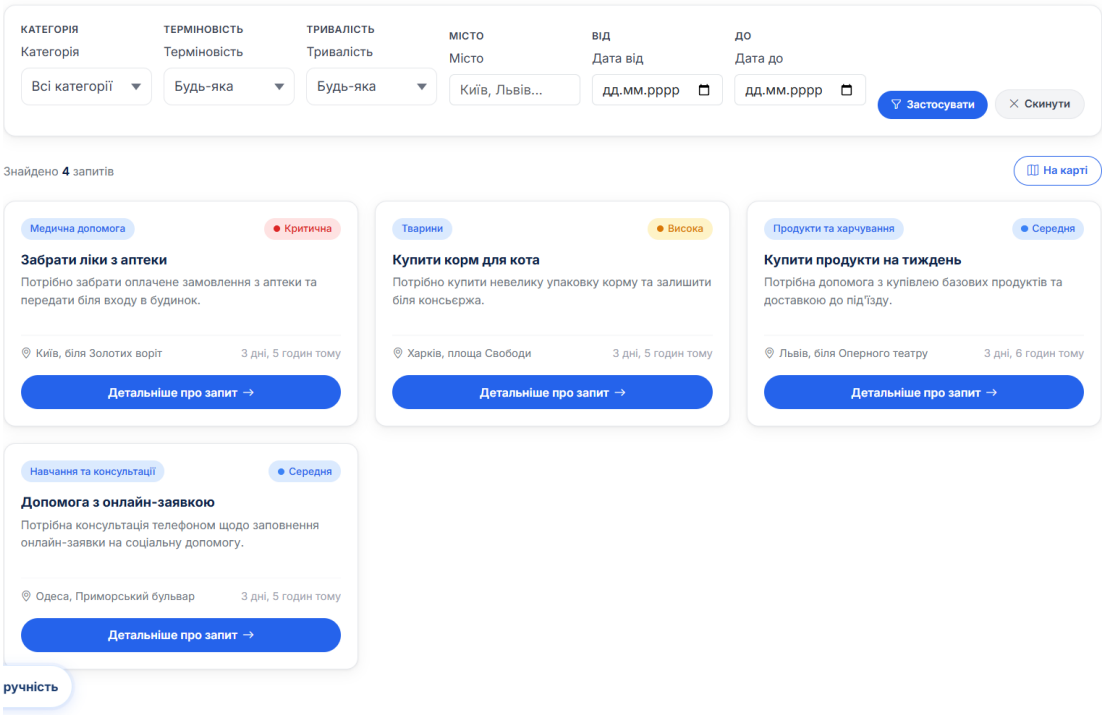


Рис. 4.5 Список запитів із фільтрами

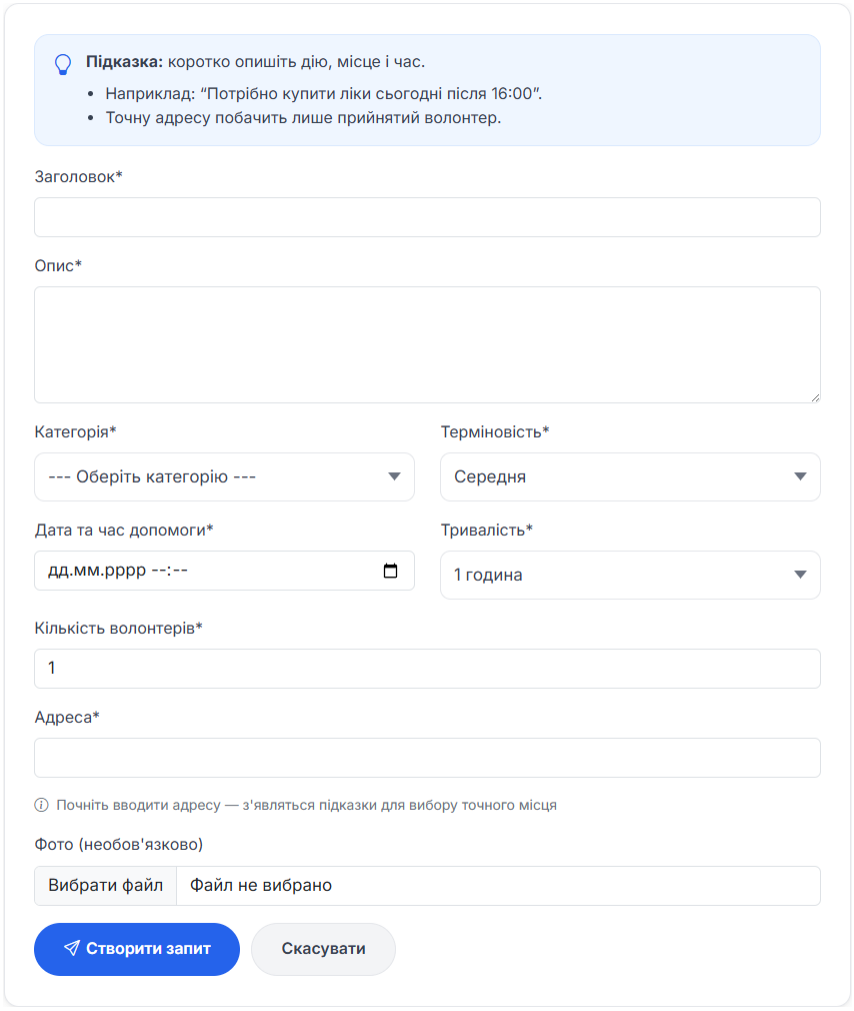


Рис. 4.6 Форма створення запиту на допомогу

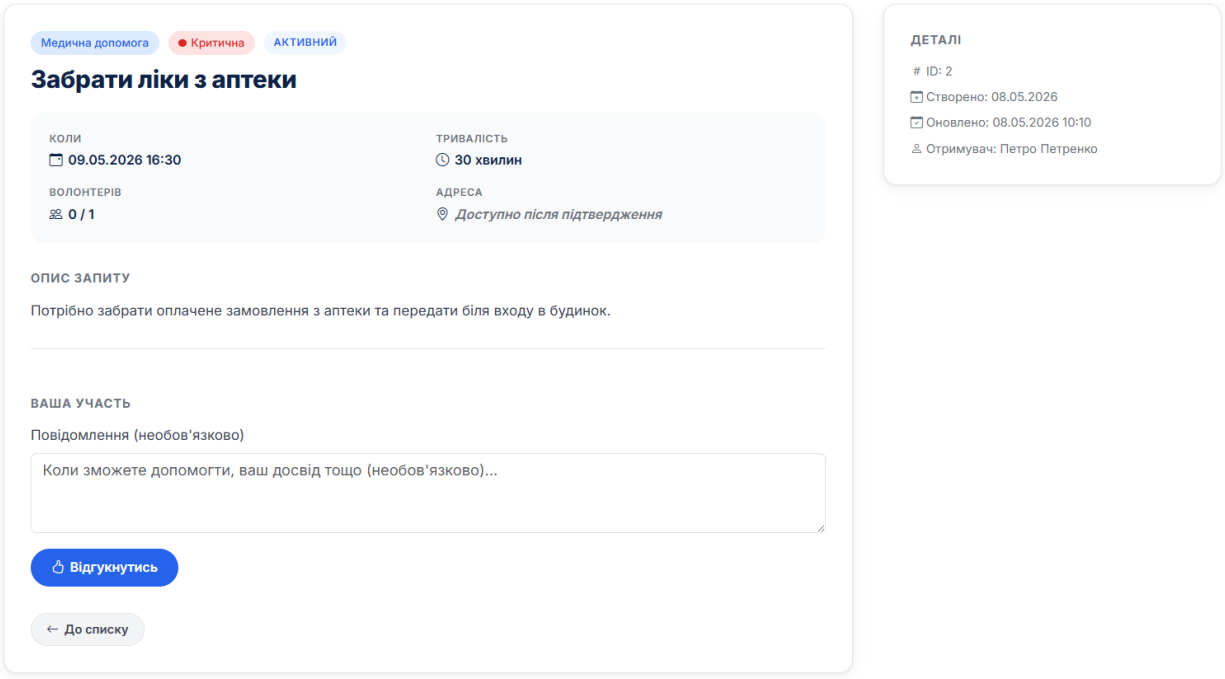


Рис. 4.7 Детальна сторінка запиту

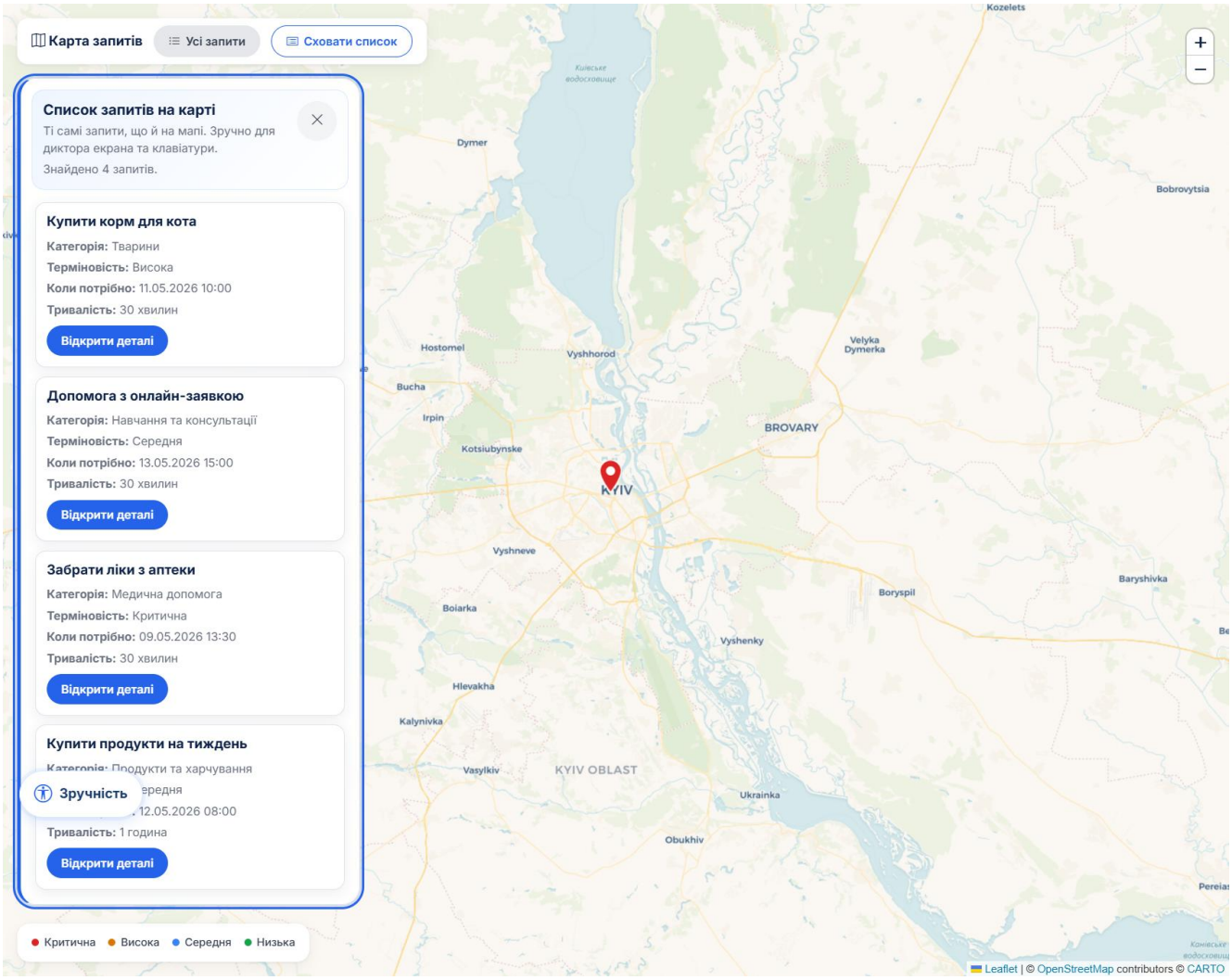


Рис. 4.8 Інтерактивна карта активних запитів

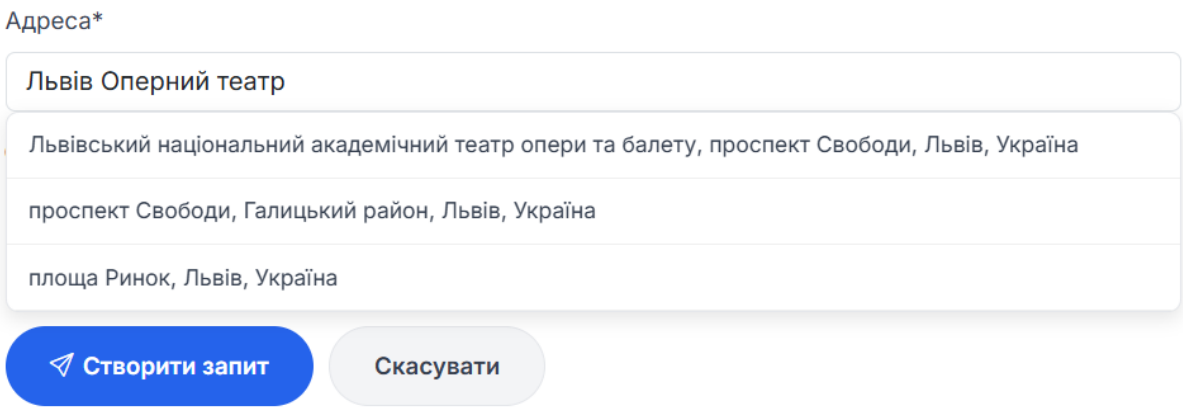


Рис. 4.9 Автодоповнення адрес через Nominatim

4.1.3 Модуль відгуків та рейтингів

Модуль reviews реалізовано у застосунку apps/reviews. Основна сутність — модель Review з полями: help_request (FK → HelpRequest), author (FK → User),

target (FK → User), rating (IntegerField, 1–5), comment (TextField, обов'язкове на рівні моделі) і created_at.

Система рейтингів реалізована як оцінювання волонтера отримувачем після завершення запиту (статус COMPLETED). Середній рейтинг відображається в профілі як агрегація: User.reviews_received.aggregate(Avg('rating')). Демонстраційні матеріали (форма відгуку, профіль із рейтингом) подано на рисунках 4.10–4.11.

Демонстраційні матеріали до модуля відгуків наведено на риссунках 4.10–4.11.

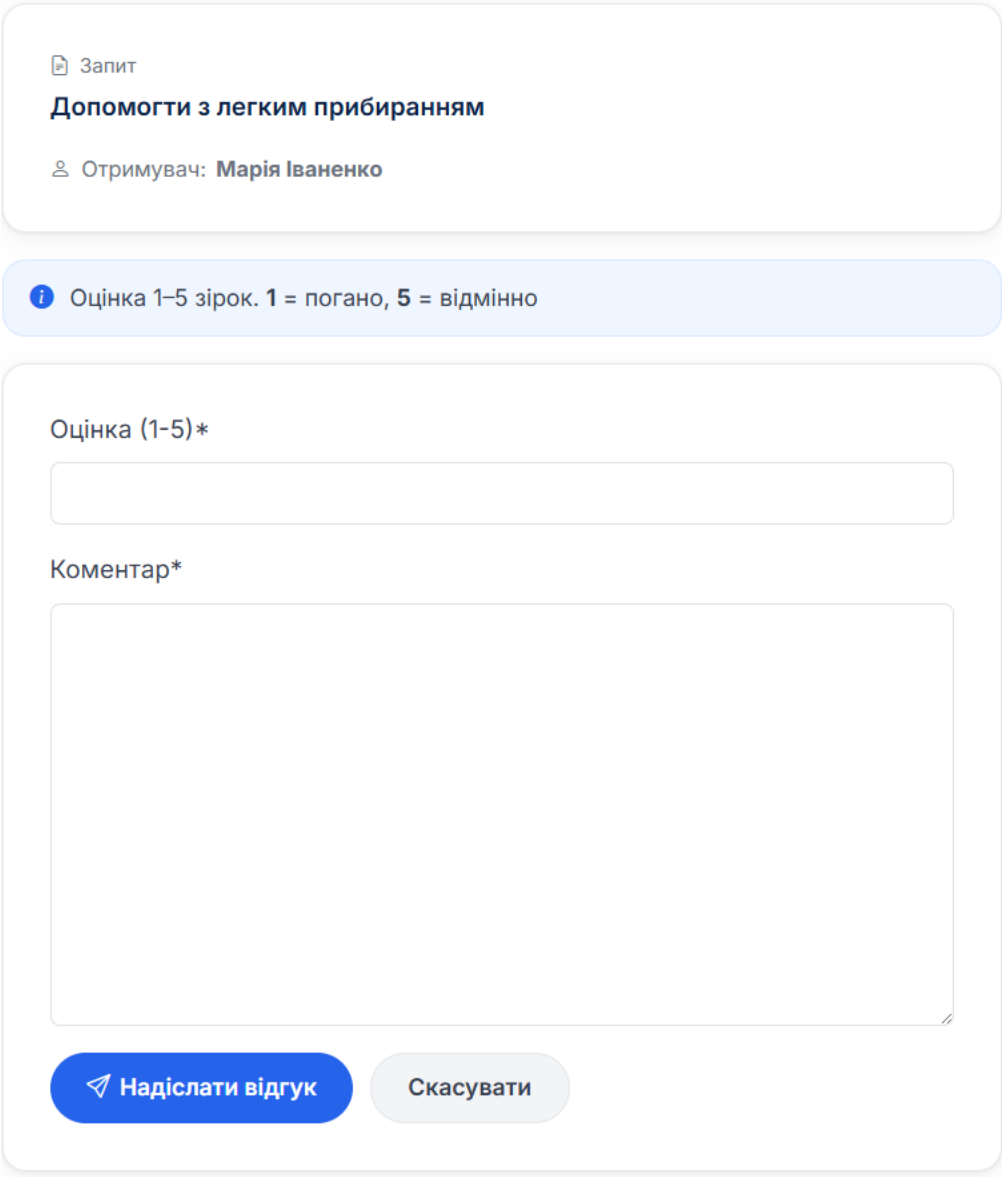


Рис. 4.10 Форма створення відгуку та оцінки

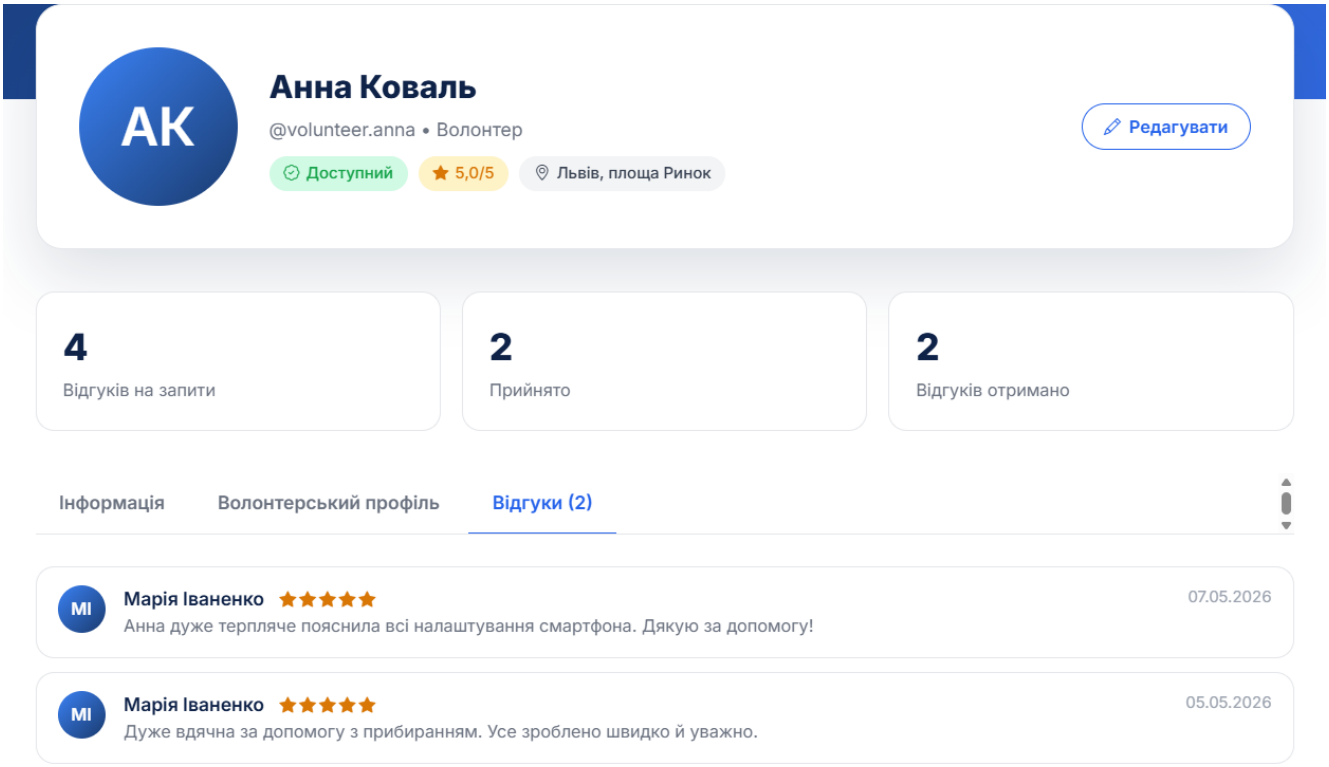


Рис. 4.11 Відгуки та рейтинг у профілі користувача

4.1.4 Модуль сповіщень

Модуль notifications реалізовано у застосунку apps/notifications як event-driven підсистему. Шість receiver-функцій перехоплюють Django-сигнали і створюють Notification-об'єкти: on_review_created, on_response_status_change, on_response_received, on_new_request_created, on_request_completed і on_request_completed_recipient.

Актуальний механізм доставки сповіщень — lightweight polling (ADR-010): клієнтська частина кожні 15 секунд (POLL_INTERVAL_MS = 15000) надсилає GET /notifications/count/ і отримує JSON {"count": N}. При N > 0 у навігаційній панелі відображається badge. Якщо N збільшився порівняно з попереднім запитом, з'являється Toast-повідомлення. Такий підхід вносить затримку до 15 секунд, але не ризикує насичити Gunicorn worker-pool, як це робили SSE-з'єднання.

Демонстраційні матеріали до модуля сповіщень наведено на рисунках 4.12–4.14.

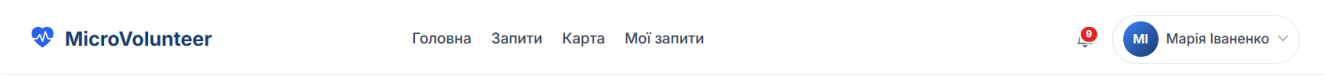


Рис. 4.12 Індикатор непрочитаних сповіщень у навігаційній панелі

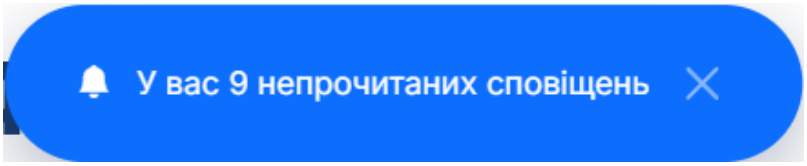


Рис. 4.13 Toast-повідомлення про нову подію

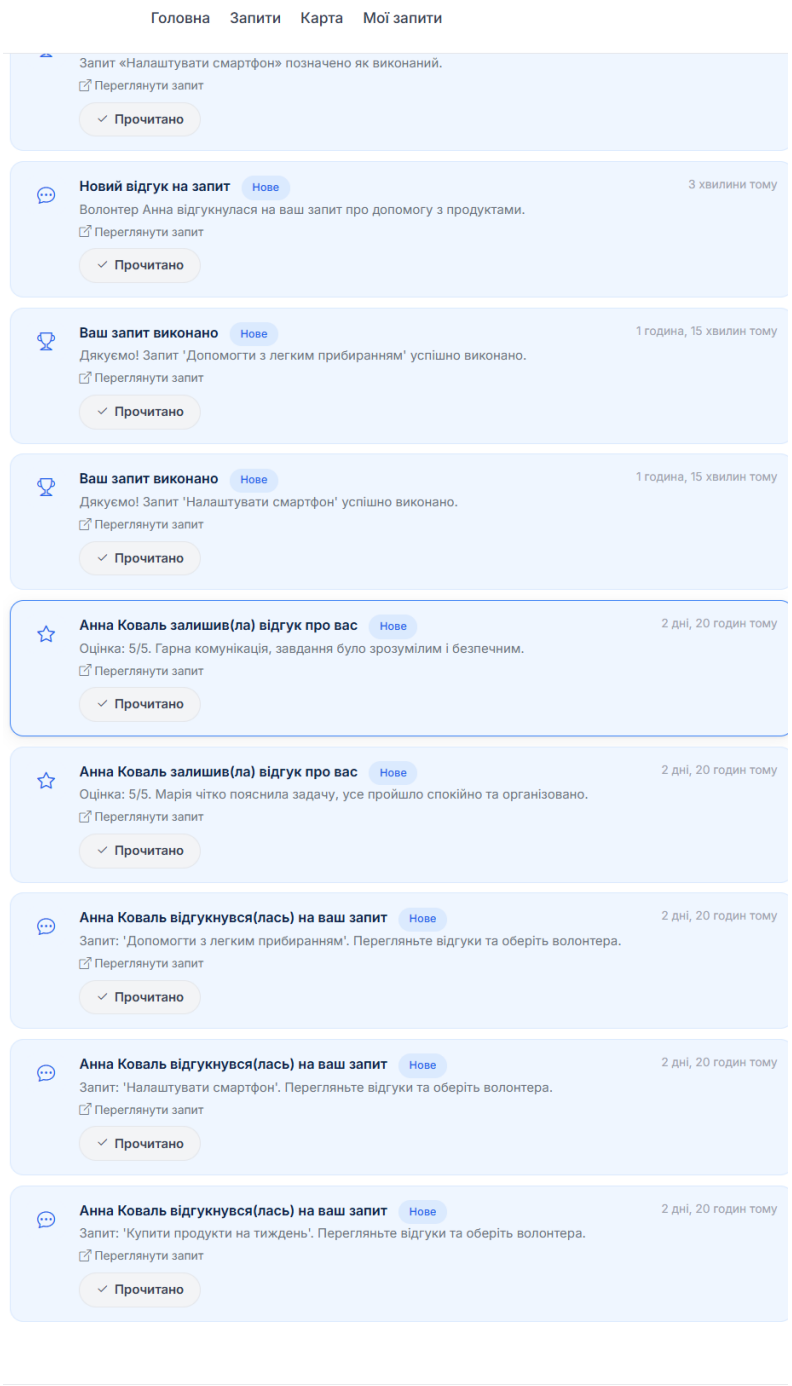


Рис. 4.14 Сторінка списку сповіщень

4.1.5 Модуль статистичного дашборду

Модуль stats реалізовано у застосунку apps/stats. Дашборд доступний лише адміністраторам (@staff_member_required) і відображає: кругову діаграму розподілу запитів за статусами, середній рейтинг по категоріях і таблицю топ-5 користувачів за кількістю отриманих відгуків. Тижневої динаміки активності дашборд не містить.

Всі агрегації виконуються через Django ORM: HelpRequest.objects.values('status').annotate(count=Count('id')) для кругової діаграми розподілу за статусами, аналогічні annotate-запити — для рейтингу і топ-користувачів. Дані передаються у шаблон як JSON і обробляються Chart.js на клієнті. Демонстраційні матеріали дашборду подано на рисунках 4.15–4.16.

Демонстраційні матеріали до модуля статистики наведено на рисунках 4.15–4.16.

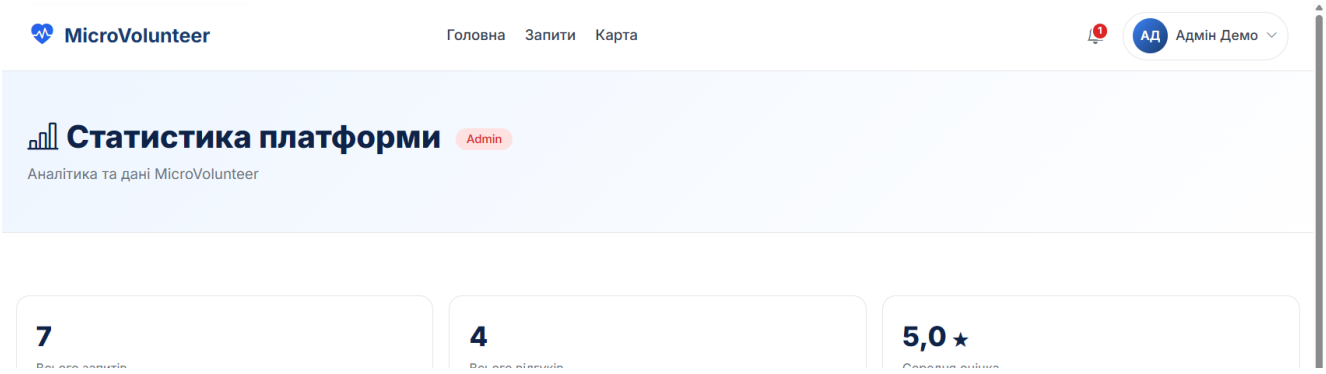


Рис. 4.15 Статистичний дашборд адміністратора: зведені показники

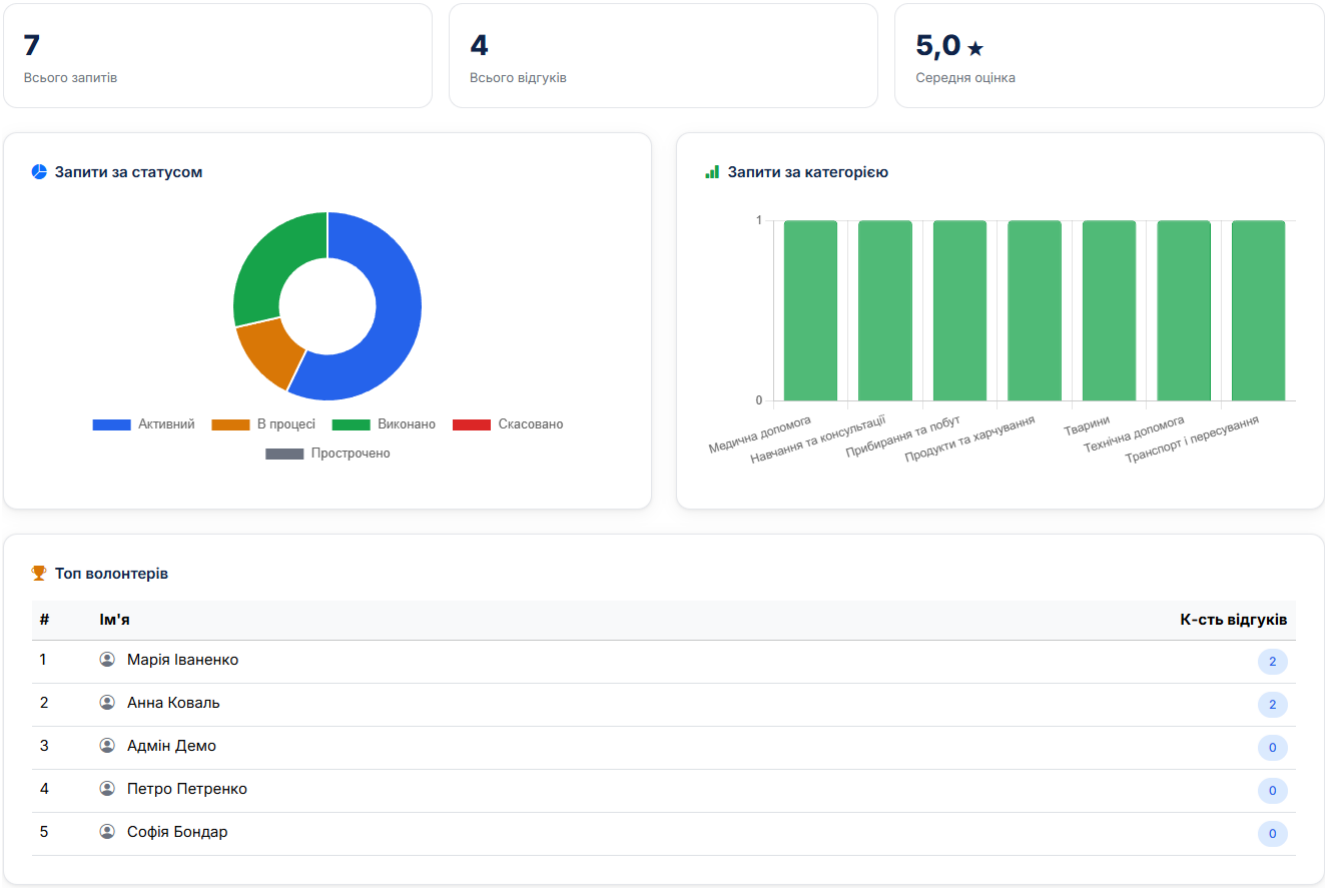


Рис. 4.16 Діаграми Chart.js та топ волонтерів на дашборді

4.2 Безпека вебзастосунку та приватність даних

Безпека MicroVolunteer побудована за моделлю defense-in-depth це п'ять незалежних рівнів: транспортний, HTTP-сесій і заголовків, автентифікація та авторизація, прикладний і рівень даних [30, 31, 32]. Саме тому збій одного механізму не відкриває автоматичний доступ до критичних даних.

На транспортному рівні задано HSTS (SECURE_HSTS_SECONDS=31536000, SECURE_HSTS_INCLUDE_SUBDOMAINS=True, SECURE_HSTS_PRELOAD=True) [33] — браузер отримує вказівку звертатися до сервісу виключно через HTTPS протягом року, включно з усіма піддоменами.

Сесійні та CSRF cookies передаються тільки через захищене з'єднання (SESSION_COOKIE_SECURE=True, CSRF_COOKIE_SECURE=True). Nginx працює як reverse proxy перед Gunicorn і самостійно обслуговує static/media-файли, що відділяє периметр обробки HTTP-запитів від прикладної логіки. У settings/production.py задано заголовки SECURE_CONTENT_TYPE_NOSNIFF=True, X_FRAME_OPTIONS='DENY' і SECURE_BROWSER_XSS_FILTER=True, які закривають вектори content-sniffing, clickjacking і reflected XSS без змін у бізнес-логіці [30].

Паролі хешуються через pbkdf2_sha256 із сіллю, це ускладнює масовий brute-force навіть після компрометації бази даних [29, 34, 35]. Додатково задіяно чотири password validators: схожість із атрибутами акаунта, мінімальна довжина 8 символів, заборона поширених паролів і заборона суто числових комбінацій. Захист від перебору реалізовано через django-axes [36]: після 5 невдалих спроб вхід блокується на 1 годину. Авторизація побудована на ролях Volunteer і Recipient через @login_required, LoginRequiredMixin і декоратори @volunteer_required / @recipient_required [28]; горизонтальна ескалація обмежується через get_object_or_404 з фільтрами за власником або статусом заявки, а точна адреса й телефон отримувача відкриваються волонтеру лише після підтвердження відгуку.

Зіставлення загроз OWASP Top 10:2021 із заходами безпеки MicroVolunteer наведено в таблиці 4.1 [10].

Таблиця 4.1

Зіставлення загроз OWASP Top 10:2021 із заходами безпеки MicroVolunteer

Категорія OWASP Top 10:2021	Реалізовані або запроєктовані заходи	Обмеження
A01 Broken Access Control	LoginRequiredMixin, рольові декоратори, staff-only дашборд, object-level перевірки	Потрібен окремий аудит усіх URL перед релізом

Продовження таблиці 4.1

Зіставлення загроз OWASP Top 10:2021 із заходами безпеки MicroVolunteer

Категорія OWASP Top 10:2021	Реалізовані або запроєктовані заходи	Обмеження
A02 Cryptographic Failures	HTTPS, secure cookies, HSTS, захист SECRET_KEY через .env, стандартний PBKDF2-SHA-256 у production	Потрібен контроль реального .env і TLS-налаштувань
A03 Injection	Django ORM, форми Django, auto-escape у шаблонах	Raw SQL не використовується; перевіряти при майбутніх змінах
A04 Insecure Design	ADR-підхід, обмеження активних запитів, приватність координат	Не замінює повний threat modeling
A05 Security Misconfiguration	split settings, production checklist, DEBUG=False для production [37]	Потребує перевірки реального production .env
A06 Vulnerable and Outdated Components	Використання актуальних документацій Django, Bootstrap, Leaflet, Docker [12], [38], [39], [28]	Потрібен регулярний dependency audit
A07 Identification and Authentication Failures	Валідатори паролів, django-axes, autocomplete для менеджерів паролів	2FA не реалізовано
A08 Software and Data Integrity Failures	CSRF-токени, POST-only дії, контроль залежностей	Немає підпису артефактів деплою

Продовження таблиці 4.1

Зіставлення загроз OWASP Top 10:2021 із заходами безпеки MicroVolunteer

Категорія OWASP Top 10:2021	Реалізовані або запроєктовані заходи	Обмеження
A09 Security Logging and Monitoring Failures	Логи django-axes, Django messages, адміністративні дії	Немає централізованого SIEM/alerting
A10 Server-Side Request Forgery	Система не виконує довільні server-side URL-запити від користувача	Nominatim викликається з клієнта; майбутні server-side інтеграції потребуватимуть перевірки

4.3 Автоматизоване тестування

Автоматизоване тестування MicroVolunteer охоплює 132 тести у п'яти застосунках: accounts — 32, requests — 64, reviews — 15, notifications — 16, stats — 5 [40].

Тестові дані генеруються через factory_boy [41]: у conftest.py визначено вісім фабрик (UserFactory, VolunteerFactory, RecipientFactory, CategoryFactory, HelpRequestFactory, ResponseFactory, ReviewFactory, NotificationFactory). Використання фабрик замість fixtures прискорює написання тестів і дозволяє точно контролювати стан об'єктів (наприклад, request зі статусом ACCEPTED і двома Response). Тести перевіряють: коректність моделей (поля, обмеження, методи), форм (валідація, збереження), views (статус-коди, редиректи, рольові обмеження), сигналів (автоматичне створення Profile, генерація Notification) і JSON endpoint-ів.

Розподіл тестів за модулями наведено в таблиці 4.2.

Таблиця 4.2

Розподіл автоматизованих тестів за модулями системи

Застосунок	К-сть тестів	Основні перевірки
accounts	32	моделі, сигнали, форми, views, logout, autocomplete, декоратори, автостворення профілю
requests	64	моделі, форми, views, management-команди, карта, offset_coordinates, haversine_distance, прийняття волонтера
reviews	15	модель відгуку, форма, views, унікальність і валідація оцінки
notifications	16	модель, views, notification_count, mark-read, сигнали, on_new_request_created
stats	5	staff/non-staff/anonymous доступ і форма JSON-відповіді
Разом	132	модульні й інтеграційні перевірки основних сценаріїв

Контрольний запуск python -m pytest --cov=apps [42] підтвердив: 132 тести пройшли, загальне покриття коду застосунків — 95%. Час виконання — 9,14 с на in-memory SQLite, що відповідає вимозі швидкого CI-циклу.

4.4 Функціональне тестування і демонстрація результатів

Функціональне тестування виконано ручною валідацією сім ключових сценаріїв, що охоплюють повний користувацький шлях від реєстрації до завершення запиту. Сценарії наведено в таблиці 4.3.

Таблиця 4.3

Сценарії функціонального тестування MicroVolunteer

ID	Сценарій	Очікуваний результат	Фактичний результат	Статус
UAT-01	Реєстрація отримувача та автоматичний вхід	Створено користувача і RecipientProfile, перенаправлення у профіль	Підтверджено автоматизованими тестами реєстрації, входу та сигналу профілю	Пройдено
UAT-02	Реєстрація волонтера і редагування профілю	Створено VolunteerProfile, збережено адресу, радіус і категорії	Збереження профільних даних волонтера перевірено тестами views і форм	Пройдено
UAT-03	Створення запиту з адресою через Nominatim	Запит активний, координати заповнені, маркер з'явився на карті	Підтверджено тестами; наявний скриншот карти (рис. 4.8)	Пройдено
UAT-04	Відгук волонтера на активний запит	Створено Response, отримувач отримав NEW_RESPONSE	Створення response і сповіщення отримувача перевірено тестами	Пройдено

Продовження таблиці 4.3

Сценарії функціонального тестування MicroVolunteer

ID	Сценарій	Очікуваний результат	Фактичний результат	Статус
UAT-05	Прийняття волонтера отримувачем	Обраний відгук має ACCEPTED, інші — REJECTED, волонтери отримали сповіщення	Сценарій прийняття, auto-reject і сповіщення перевірено тестами	Пройдено
UAT-06	Завершення запиту та створення відгуку	Статус запиту COMPLETED, доступна форма оцінювання	Завершення запиту, доступ до відгуків і валідація Review перевірені тестами	Пройдено
UAT-07	Перегляд статистичного дашборду staff-користувачем	Відображено картки, діаграми статусів і категорій	Staff-only доступ і JSON-дані дашборду перевірено тестами; наявні скриншоти дашборду (рис. 4.15–4.16)	Пройдено

За результатами перевірки всі сценарії виконані без критичних відхилень. Виявлені незначні недоліки UX (зайвий scroll при відкритті popup на мобільному пристрої, некоректне відображення довгих адрес у маркері) зафіксовані як backlog-задачі для наступного ітерування.

4.5 Розгортання у контейнеризованому середовищі

Для розгортання MicroVolunteer підготовлено docker-compose.yml із трьома сервісами [13]: web (Django, запускається через python manage.py runserver 0.0.0.0:8000 у development-режимі), db (PostgreSQL 16) і nginx (reverse proxy + static files). Gunicorn прописано в Dockerfile, але docker-compose.yml перевизначає команду запуску на runserver. Назовні відкрито порт 8000 сервісу web і порт 80 Nginx.

Процедура розгортання складається з шести кроків: клонування репозиторію, копіювання .env.production (SECRET_KEY, DATABASE_URL, ALLOWED_HOSTS), docker-compose up --build -d, виконання міграцій (docker-compose exec web python manage.py migrate), збирання статики (collectstatic) і створення superuser. Всі команди задокументовані у README.md репозиторію.

Для відповідності Django Deployment Checklist ключові налаштування production: DEBUG=False, ALLOWED_HOSTS встановлено явно, SESSION_COOKIE_SECURE=True, CSRF_COOKIE_SECURE=True, SECRET_KEY зчитується зі змінної середовища.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи спроєктовано і реалізовано вебплатформу MicroVolunteer для цифрової координації мікроволонтерської діяльності, тобто систему краудсорсингу соціальної підтримки, що з'єднує людей, які потребують допомоги у повсякденних справах, з волонтерами у їхній локальній громаді.

У процесі роботи було виконано такі задачі:

1. Проведено теоретичний аналіз предметної галузі мікроволонтерства. Визначено контекст: в умовах повномасштабної війни і понад 3,7 млн внутрішньо переміщених осіб попит на локальну адресну допомогу суттєво перевищує можливості існуючих неструктурованих каналів координації (месенджери, соцмережі, таблиці). Виявлено п'ять системних проблем: відсутність геолокаційної фільтрації, неструктурований формат, неможливість відстеження статусу, відсутність репутаційного механізму, неможливість аналітики.

2. Виконано порівняльний аналіз чотирьох платформ (Добробат, Be My Eyes, TaskRabbit, MicroVolunteer) за шістьма критеріями. Показано, що жодна з наявних платформ не поєднує безкоштовність, геолокацію, широкий спектр категорій і двосторонній рейтинг Сформульовано вісім функціональних і шість нефункціональних вимог.

3. Обґрунтовано технологічний стек: Python 3.12 / Django 5.2.11, PostgreSQL 16, Leaflet.js + Nominatim, Bootstrap 5, Docker + docker-compose. Спроєктовано модульну архітектуру з п'яти Django-застосунків, реляційну модель із восьми сутностей і одинадцять ADR. Ключові рішення: SSR замість SPA, CartoDB Voyager замість Google Maps, lightweight polling замість SSE, SQLite для тестів замість PostgreSQL.

4. Реалізовано серверну і клієнтську частини: 5 застосунків, машина станів HelpRequest, геолокаційний пошук, подієво-орієнтовані сповіщення на основі polling-моделі, статистичний дашборд.

5. Забезпечено п'ятирівневий захист від загроз OWASP Top 10:2021: HTTPS/HSTS, HTTP security headers, pbkdf2_sha256 × 1 000 000 ітерацій + django-axes, рольова авторизація через декоратори, ORM-параметризація проти SQL-ін'єкцій.

6. Розроблено комплексний набір тестів: 132 автоматизованих тести (95% code coverage, 9,14 с), 7 функціональних сценаріїв без критичних відхилень.

7. Підготовлено розгортання через docker-compose (web + db + nginx) відповідно до Django Deployment Checklist.

Перспективи розвитку: мобільний PWA-клієнт, push-сповіщення, верифікація волонтерів, розширена геоаналітика та інтеграція з адміністративними базами даних соціальних служб.

Робота пройшла апробацію. За її результатами було опубліковано такі тези доповідей:

1. Поліщук Я. О., Садовенко В. С. Розробка системи краудсорсингу соціальної підтримки на базі веб-технологій. Матеріали ІІ Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії». 26 листопада 2025 року, м. Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С.91-93.

2. Поліщук Я. О., Садовенко В. С. Забезпечення безпеки інформації у web-застосунку краудсорсингу соціальної підтримки MicroVolunteer. Збірник матеріалів Всеукраїнської конференції молодих учених «Інформаційні технології». 20 травня 2026 року, м.Київ, Київський столичний університет імені Бориса Грінченка. Збірник тез. 2026, Інформаційні технології – 2026. С. 253-258.

ПЕРЕЛІК ПОСИЛАНЬ

1. Більше 2 мільйонів людей зареєструвались як внутрішньо переміщені особи після впровадження воєнного стану // Міністерство соціальної політики України. 25.04.2022. URL: https://www.msp.gov.ua/press-center/news/bilshe-2-milyoniv-lyudey-zareyestruvalys-yak-vnutrishno-peremishcheni-osoby-pislya-vprovadzhennya-voyennoho-stanu (дата звернення: 29.04.2026).

2. Ukraine Internal Displacement Report: General Population Survey Round 22, January 2026 / International Organization for Migration. Geneva : IOM, 2026. URL: https://dtm.iom.int/ (дата звернення: 29.04.2026).

3. Ukraine Humanitarian Needs and Response Plan 2024 / United Nations Office for the Coordination of Humanitarian Affairs. New York : OCHA, 2024. URL: https://www.unocha.org/ (дата звернення: 28.03.2026).

4. Про волонтерську діяльність : Закон України від 19.04.2011 № 3236-VI // База даних «Законодавство України» / Верховна Рада України. URL: https://zakon.rada.gov.ua/go/3236-17 (дата звернення: 25.04.2026).

5. 2022 State of the World's Volunteerism Report: Building Equal and Inclusive Societies / United Nations Volunteers. Bonn : UNV, 2022. URL: https://swvr2022.unv.org/ (дата звернення: 28.03.2026).

6. Kemp S. Digital 2024: Ukraine / DataReportal. 2024. URL: https://datareportal.com/reports/digital-2024-ukraine (дата звернення: 28.03.2026).

7. Добробат : офіційний сайт. URL: https://www.dobrobat.in.ua/ (дата звернення: 04.04.2026).

8. Be My Eyes : офіційний сайт. URL: https://www.bemyeyes.com/ (дата звернення: 04.04.2026).

9. TaskRabbit : офіційний сайт. URL: https://www.taskrabbit.com/ (дата звернення: 04.04.2026).

10. OWASP Top Ten Web Application Security Risks / OWASP Foundation. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 11.04.2026).

11. Web Content Accessibility Guidelines (WCAG) 2.2 : W3C Recommendation / World Wide Web Consortium. URL: https://www.w3.org/TR/WCAG22/ (дата звернення: 11.04.2026).

12. Docker documentation / Docker Inc. URL: https://docs.docker.com/ (дата звернення: 17.04.2026).

13. Docker Compose documentation / Docker Inc. URL: https://docs.docker.com/compose/ (дата звернення: 17.04.2026).

14. Python 3.12 documentation / Python Software Foundation. URL: https://docs.python.org/3.12/ (дата звернення: 17.04.2026).

15. Flask Documentation / Pallets Projects. URL: https://flask.palletsprojects.com/en/latest/ (дата звернення: 17.04.2026).

16. FastAPI documentation / S. Ramírez. URL: https://fastapi.tiangolo.com/ (дата звернення: 17.04.2026).

17. PostgreSQL 16 Documentation / The PostgreSQL Global Development Group. URL: https://www.postgresql.org/docs/16/ (дата звернення: 17.04.2026).

18. pytest-django Documentation / pytest-django contributors. URL: https://pytest-django.readthedocs.io/en/latest/ (дата звернення: 23.04.2026).

19. Nominatim documentation / OpenStreetMap Foundation. URL: https://nominatim.org/release-docs/latest/ (дата звернення: 14.05.2026).

20. Nominatim Usage Policy / OpenStreetMap Foundation. URL: https://operations.osmfoundation.org/policies/nominatim/ (дата звернення: 14.05.2026).

21. Gunicorn documentation / Gunicorn contributors. URL: https://gunicorn.org/ (дата звернення: 14.05.2026).

22. nginx documentation / NGINX. URL: https://nginx.org/en/docs/ (дата звернення: 14.05.2026).

23. Guide to the Software Engineering Body of Knowledge (SWEBOK Guide). Version 4.0 / IEEE Computer Society. URL: https://www.computer.org/education/bodies-of-knowledge/software-engineering (дата звернення: 11.04.2026).

24. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Boston : Addison-Wesley Professional, 2021. 464 p.

25. HTML Living Standard: Server-sent events / WHATWG. URL: https://html.spec.whatwg.org/multipage/server-sent-events.html (дата звернення: 05.05.2026).

26. Chart.js documentation / Chart.js contributors. URL: https://www.chartjs.org/docs/latest/ (дата звернення: 05.05.2026).

27. Elmasri R., Navathe S. B. Fundamentals of Database Systems. 7th ed. Boston : Pearson, 2016. 1272 p.

28. Django documentation. Version 5.2 / Django Software Foundation. URL: https://docs.djangoproject.com/en/5.2/ (дата звернення: 17.04.2026).

29. NIST Special Publication 800-63B. Digital Identity Guidelines: Authentication and Lifecycle Management / National Institute of Standards and Technology. URL: https://pages.nist.gov/800-63-3/sp800-63b.html (дата звернення: 18.04.2026).

30. Security in Django // Django documentation / Django Software Foundation. URL: https://docs.djangoproject.com/en/5.2/topics/security/ (дата звернення: 18.04.2026).

31. OWASP Application Security Verification Standard 4.0.3 / OWASP Foundation. URL: https://owasp.org/www-project-application-security-verification-standard/ (дата звернення: 18.04.2026).

32. NIST Special Publication 800-160 Vol. 2. Developing Cyber-Resilient Systems: A Systems Security Engineering Approach / National Institute of Standards and Technology. URL: https://csrc.nist.gov/pubs/sp/800/160/v2/r1/final (дата звернення: 18.04.2026).

33. RFC 6797. HTTP Strict Transport Security (HSTS) / IETF. URL: https://www.rfc-editor.org/rfc/rfc6797 (дата звернення: 18.04.2026).

34. Password Storage Cheat Sheet / OWASP Foundation. URL: https://cheatsheetseries.owasp.org/cheatsheets/PasswordStorageCheatSheet.html (дата звернення: 18.04.2026).

35. NIST Special Publication 800-132. Recommendation for Password-Based Key Derivation / National Institute of Standards and Technology. URL: https://csrc.nist.gov/publications/detail/sp/800-132/final (дата звернення: 18.04.2026).

36. django-axes documentation / django-axes contributors. URL: https://django-axes.readthedocs.io/en/latest/ (дата звернення: 05.05.2026).

37. Deployment checklist // Django documentation / Django Software Foundation. URL: https://docs.djangoproject.com/en/5.2/howto/deployment/checklist/ (дата звернення: 05.05.2026).

38. Bootstrap v5.3 documentation / Bootstrap Team. URL: https://getbootstrap.com/docs/5.3/ (дата звернення: 05.05.2026).

39. Leaflet API reference. Version 1.9.4 / Leaflet. URL: https://leafletjs.com/reference.html (дата звернення: 05.05.2026).

40. Ammann P., Offutt J. Introduction to Software Testing. 2nd ed. Cambridge : Cambridge University Press, 2016. 364 p.

41. factoryboy documentation / Factory Boy contributors. URL: https://factoryboy.readthedocs.io/en/stable/ (дата звернення: 23.04.2026).

42. pytest documentation / pytest-dev. URL: https://docs.pytest.org/en/stable/ (дата звернення: 23.04.2026).

43. ISO/IEC/IEEE 29148:2018. Systems and software engineering — Life cycle processes — Requirements engineering. 2nd ed. Geneva : ISO, 2018. 92 p. URL: https://www.iso.org/standard/72089.html (дата звернення: 11.04.2026).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка системи краудсорсингу соціальної підтримки на базі web-технологій

Виконав студент 4 курсу
групи ПД-41
Поліщук Ярослав Олексійович
Керівник роботи
канд.фіз.-мат.наук, доцент Садовенко Володимир Сергійович

Київ – 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – спрощення організації та координації мікроволонтерської діяльності шляхом розроблення вебзастосунку для взаємодії волонтерів і отримувачів допомоги.

Об’єкт дослідження – процес організації мікроволонтерської діяльності для соціально вразливих груп населення.

Предмет дослідження – засоби та технології розробки вебсистеми краудсорсингу соціальної підтримки на базі фреймворку Django, бібліотеки Leaflet.js та СУБД PostgreSQL.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести теоретичний аналіз предметної галузі мікроволонтерства, виявити проблеми координації між волонтерами та отримувачами допомоги, обґрунтувати актуальність задачі.
2. Виконати порівняльний аналіз існуючих програмних рішень, сформулювати функціональні та нефункціональні вимоги до власної системи.
3. Спроєктувати архітектуру вебзастосунку та структуру бази даних.
4. Реалізувати серверну та клієнську частину.
5. Провести тестування програмного забезпечення.

3

АНАЛІЗ АНАЛОГІВ

Критерій	Добробат	Be My Eyes	TaskRabbit	MicroVolunteer
Геолокація та карта запитів	+	–	+	+
Категоризація запитів	+	–	+	+
Двостороння система рейтингів	–	+	+	+
Безкоштовне користування	+	+	–	+
Фокус на Україні	+	–	–	+
Мікрозавдання (короткотривалі)	–	+	+	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Реєстрація та автентифікація з розмежуванням ролей (волонтер / отримувач допомоги);
2. Створення, редагування та керування запитами допомоги з категоризацією, рівнями терміновості та приблизним виділенням часу на виконання;
3. Перегляд запитів на інтерактивній карті з фільтрацією за місцем, категорією, пріоритетом, назвою та часом;
4. Захист приватності отримувача допомоги через приховування точної адреси та зсув геокоординат.
5. Механізм відгуку волонтерів на запит з підтвердженням отримувачем;
6. Двостороння система рейтингів і відгуків після виконання запиту;
7. Особистий кабінет з профілем, статистикою та історією активності.

Нефункціональні вимоги:

1. Безпека: захист від CSRF, XSS, SQL-ін'єкцій; хешування паролів; захист від брутфорсу;
2. Адаптивність інтерфейсу для коректної роботи на настільних, планшетних і мобільних пристроях.
3. Сумісність із сучасними веббраузерами.
4. Українська локалізація інтерфейсу.
5. Контейнеризація додатку для простого розгортання;

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Python 3.12
мова бекенду

Django 5.2
веб-фреймворк, ORM, MTV

PostgreSQL 16
СУБД (продакшн)

Pytest
Тести

Bootstrap 5
адаптивний UI

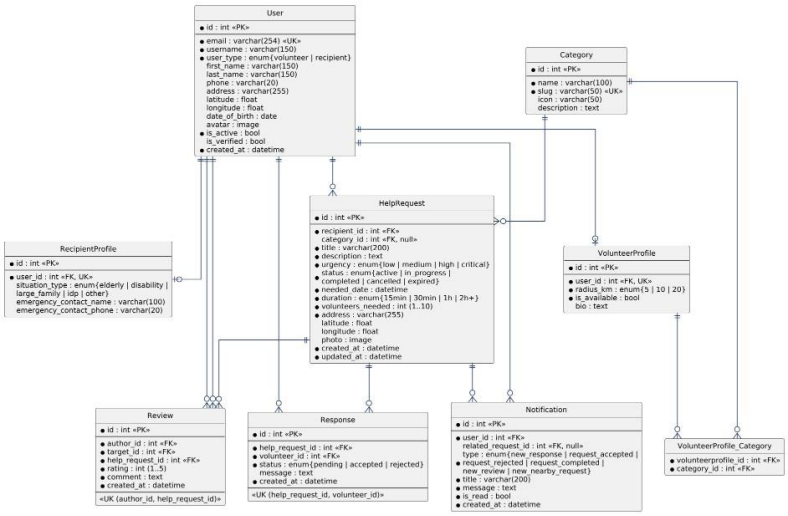
Leaflet.js + CartoDB
інтерактивна карта

Chart.js
статистичні графіки

Docker + Nginx
контейнеризація та проксі

6

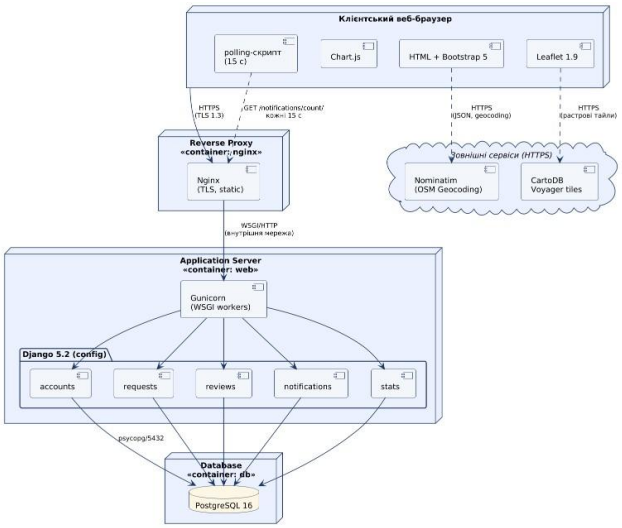
ER-ДІАГРАМА БАЗИ ДАННИХ



ER-діаграма БД MicroVolunteer (9 сутностей)

7

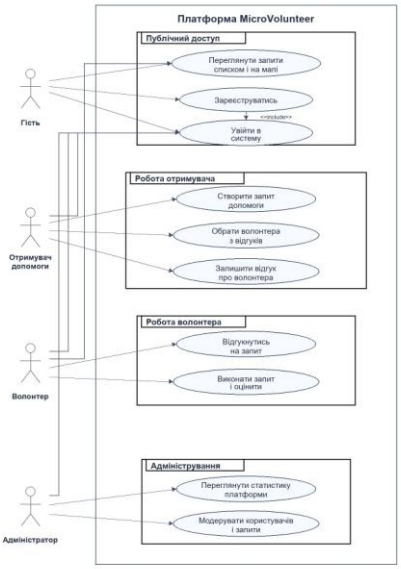
ЗАГАЛЬНА АРХІТЕКТУРА СИСТЕМИ



Компонентна діаграма архітектури MicroVolunteer

8

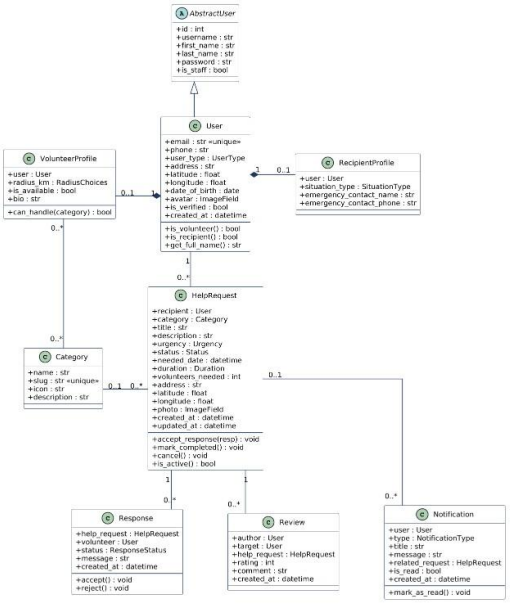
ВАРІАНТИ ВИКОРИСТАННЯ



Use-case діаграма (4 актори)

9

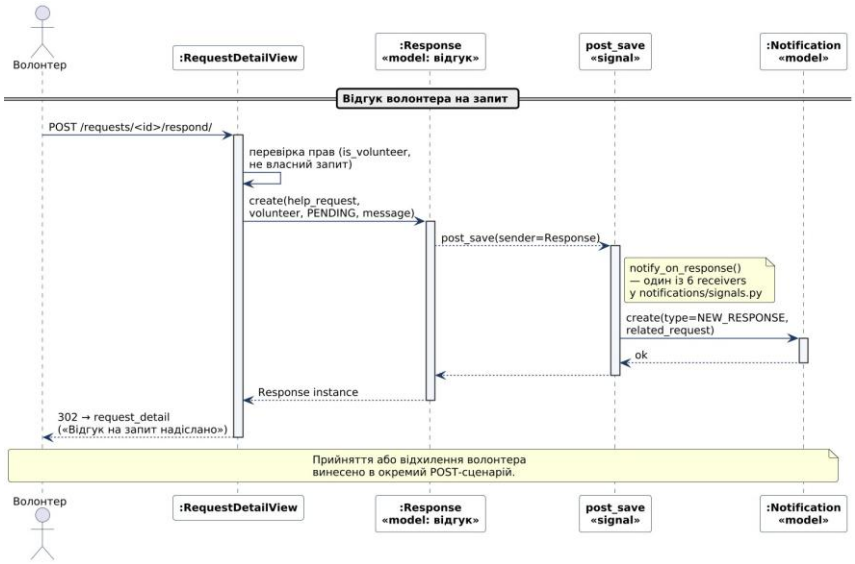
ДІАГРАМА КЛАСІВ



Діаграма класів предметної області

10

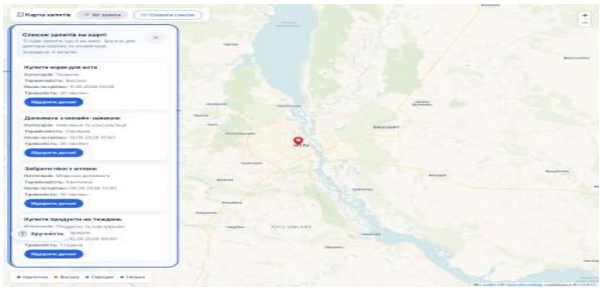
СЦЕНАРІЙ ВЗАЄМОДІЇ



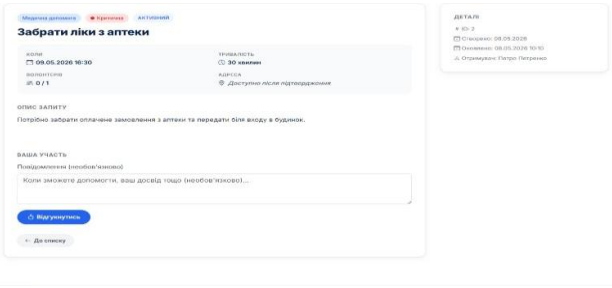
Sequence-діаграма: волонтер відгукується на запит

11

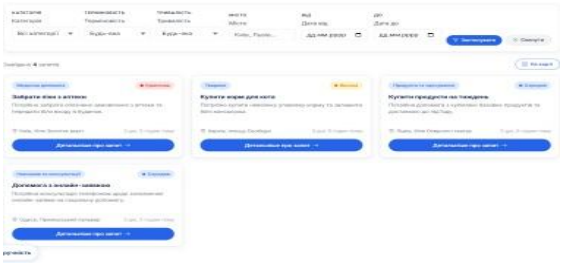
ЕКРАННІ ФОРМИ ВЕБПЛАТФОРМИ MICROVOLUNTEER



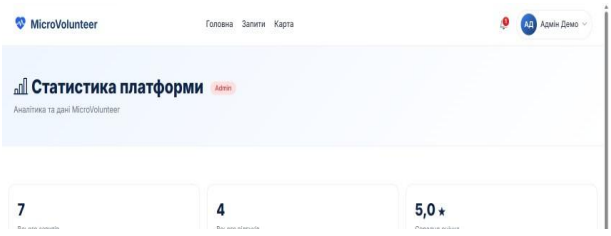
Інтерактивна карта активних запитів



Детальна сторінка запиту



Список запитів із фільтрами



Статистичний дашборд адміністратора

12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Поліщук Я. О., Садовенко В. С. Розробка системи краудсорсингу соціальної підтримки на базі веб-технологій. Матеріали II Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії». 18 листопада 2025 року, м. Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С.91-93.
2. Поліщук Я. О., Садовенко В. С. Забезпечення безпеки інформації у web-застосунку краудсорсингу соціальної підтримки MicroVolunteer. Матеріали XIII Всеукраїнської конференції «IT-2026». 14 травня 2026 року, м.Київ, Київський столичний університет імені Бориса Грінченка. Збірник тез. 2026, С. 253-258

13

ВИСНОВКИ

1. Проведено аналіз предметної галузі мікроволонтерства; виявлено 5 системних проблем неструктурованої координації та визначено функціональну нішу MicroVolunteer на основі порівняння з Добробат, Be My Eyes, TaskRabbit.
2. Сформульовано функціональні та 4 нефункціональні вимоги, а також обґрунтовано вибір технологій розробки.
3. Спроєктовано архітектуру вебзастосунку та реляційну структуру бази даних.
4. Реалізовано серверну та клієнтську частини системи: геолокаційний пошук, керування запитами, рейтинги, сповіщення та статистичний дашборд.
5. Проведене тестування підтвердило працездатність системи та відповідність основним вимогам.
6. Підготовлено сценарій контейнеризованого розгортання вебзастосунку

14

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

```
# apps/accounts/models.py
from django.contrib.auth.models import AbstractUser
from django.core.validators import
FileExtensionValidator
from django.db import models


class User(AbstractUser):
    """Custom user model with volunteer/recipient role
distinction."""

    class UserType(models.TextChoices):
        VOLUNTEER = 'volunteer', 'Волонтер'
        RECIPIENT = 'recipient', 'Отримувач допомоги'

    email = models.EmailField('Електронна пошта',
unique=True)
    phone = models.CharField('Телефон',
max_length=20, blank=True)
    user_type = models.CharField('Тип акаунту',
max_length=10, choices=UserType.choices)
    address = models.CharField('Адреса',
max_length=255, blank=True)
    latitude = models.FloatField('Широта', null=True,
blank=True)
    longitude = models.FloatField('Довгота', null=True,
blank=True)
    avatar = models.ImageField(
        'Фото профілю',
        upload_to='avatars/',
        blank=True,

validators=[FileExtensionValidator(allowed_extensions=
['jpg', 'jpeg', 'png', 'webp'])],
    )
    is_verified = models.BooleanField('Верифікований',
default=False)
    created_at = models.DateTimeField('Дата реєстрації',
auto_now_add=True)

    class Meta:
        verbose_name = 'Користувач'
        verbose_name_plural = 'Користувачі'
        ordering = ['-created_at', '-id']

    @property
    def is_volunteer(self):
        return self.user_type ==
self.UserType.VOLUNTEER

    @property
    def is_recipient(self):
        return self.user_type == self.UserType.RECIPIENT


class VolunteerProfile(models.Model):
    """Extended profile for volunteer users."""

    class RadiusChoices(models.IntegerChoices):
        SMALL = 5, '5 км'
        MEDIUM = 10, '10 км'
        LARGE = 20, '20 км'

    user = models.OneToOneField(
        User, on_delete=models.CASCADE,
        related_name='volunteer_profile',
verbose_name='Користувач'
    )
    categories = models.ManyToManyField(
        'requests.Category', blank=True,
        related_name='volunteers',
verbose_name='Категорії допомоги'
    )
    radius_km = models.IntegerField(
        'Радіус готовності',
choices=RadiusChoices.choices,
default=RadiusChoices.MEDIUM
    )
    is_available = models.BooleanField('Доступний',
default=True)
    bio = models.TextField('Про себе', blank=True)


class RecipientProfile(models.Model):
    """Extended profile for help-recipient users."""

    class SituationType(models.TextChoices):
        ELDERLY = 'elderly', 'Літня людина'
        DISABILITY = 'disability', 'Особа з інвалідністю'
        LARGE_FAMILY = 'large_family', 'Багатодітна
сім\'я'
        IDP = 'idp', 'ВПО (внутрішньо переміщена
особа)'
        OTHER = 'other', 'Інше'

    user = models.OneToOneField(
        User, on_delete=models.CASCADE,
        related_name='recipient_profile',
verbose_name='Користувач'
    )
    situation_type = models.CharField(
        'Тип ситуації', max_length=20,
choices=SituationType.choices, blank=True
    )
    emergency_contact_name =
models.CharField('Контактна особа (ім\'я)',
max_length=100, blank=True)
    emergency_contact_phone =
models.CharField('Контактна особа (телефон)',
max_length=20, blank=True)


# apps/requests/models.py
```

```
class Category(models.Model):
    name = models.CharField('Назва', max_length=100)
    slug = models.SlugField('URL-ідентифікатор',
unique=True)
    icon = models.CharField('CSS-клас іконки',
max_length=50, blank=True)
    description = models.TextField('Опис', blank=True)

    class Meta:
        verbose_name = 'Категорія'
        verbose_name_plural = 'Категорії'
        ordering = ['name']


class HelpRequest(models.Model):
    class Urgency(models.TextChoices):
        LOW = 'low', 'Низька'
        MEDIUM = 'medium', 'Середня'
        HIGH = 'high', 'Висока'
        CRITICAL = 'critical', 'Критична'

    class Status(models.TextChoices):
        ACTIVE = 'active', 'Активний'
        IN_PROGRESS = 'in_progress', 'В процесі'
        COMPLETED = 'completed', 'Виконано'
        CANCELLED = 'cancelled', 'Скасовано'
        EXPIRED = 'expired', 'Прострочено'

    recipient = models.ForeignKey(
        'accounts.User', on_delete=models.CASCADE,
        related_name='help_requests',
verbose_name='Отримувач'
    )
    title = models.CharField('Заголовок',
max_length=200)
    description = models.TextField('Опис')
    category = models.ForeignKey(
        Category, on_delete=models.SET_NULL,
null=True,
        related_name='help_requests',
verbose_name='Категорія'
    )
    urgency = models.CharField('Терміновість',
max_length=10, choices=Urgency.choices,
default=Urgency.MEDIUM)
    status = models.CharField('Статус', max_length=15,
choices=Status.choices, default=Status.ACTIVE)
    needed_date = models.DateTimeField('Дата та час
допомоги')
    address = models.CharField('Адреса',
max_length=255)
    latitude = models.FloatField('Широта', null=True,
blank=True)
    longitude = models.FloatField('Довгота', null=True,
blank=True)
    created_at = models.DateTimeField('Дата створення',
auto_now_add=True)
    updated_at = models.DateTimeField('Дата
оновлення', auto_now=True)

    class Meta:
        verbose_name = 'Запит допомоги'
        verbose_name_plural = 'Запити допомоги'
        ordering = ['-created_at']


class Response(models.Model):
    class Status(models.TextChoices):
        PENDING = 'pending', 'Очікує'
        ACCEPTED = 'accepted', 'Прийнято'
        REJECTED = 'rejected', 'Відхилено'

    help_request = models.ForeignKey(
        HelpRequest, on_delete=models.CASCADE,
        related_name='responses', verbose_name='Запит'
    )
    volunteer = models.ForeignKey(
        'accounts.User', on_delete=models.CASCADE,
        related_name='volunteer_responses',
verbose_name='Волонтер'
    )
    status = models.CharField('Статус', max_length=10,
choices=Status.choices, default=Status.PENDING)
    message = models.TextField('Повідомлення',
blank=True)
    created_at = models.DateTimeField('Дата відгуку',
auto_now_add=True)

    class Meta:
        verbose_name = 'Відгук волонтера'
        verbose_name_plural = 'Відгуки волонтерів'
        ordering = ['-created_at']
        unique_together = ['help_request', 'volunteer']


# apps/requests/utils.py
import math


def haversine_distance(lat1: float, lon1: float, lat2: float,
lon2: float) -> float:
    """
    Calculate the great-circle distance between two points
    on Earth using the Haversine formula.
    """
    R = 6371
    lat1_rad = math.radians(lat1)
    lat2_rad = math.radians(lat2)
    delta_lat = math.radians(lat2 - lat1)
    delta_lon = math.radians(lon2 - lon1)

    a = (
        math.sin(delta_lat / 2) ** 2
        + math.cos(lat1_rad) * math.cos(lat2_rad) *
math.sin(delta_lon / 2) ** 2
    )
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    return R * c


def offset_coordinates(lat: float, lon: float, offset_meters:
float = 100) -> tuple[float, float]:
    """
    Offset coordinates by approximately the given
distance in a random direction.
```

```
    Used for privacy — hides exact location until
volunteer is confirmed.
    """
    import random

    lat_offset = offset_meters / 111_320
    lon_offset = offset_meters / (111_320 *
math.cos(math.radians(lat)))

    new_lat = lat + random.uniform(-lat_offset, lat_offset)
    new_lon = lon + random.uniform(-lon_offset,
lon_offset)
    return new_lat, new_lon


# apps/notifications/signals.py
def _volunteer_matches_request(volunteer,
help_request):
    """
    Decide whether a volunteer should be notified about a
new request.
    """
    profile = getattr(volunteer, "volunteer_profile", None)
    if profile is None or not profile.is_available:
        return False

    if help_request.category_id and
profile.categories.exists():
        if not
profile.categories.filter(pk=help_request.category_id).exi
sts():
            return False

    if (
        volunteer.latitude is not None
        and volunteer.longitude is not None
        and help_request.latitude is not None
        and help_request.longitude is not None
    ):
        distance_km = haversine_distance(
            volunteer.latitude,
            volunteer.longitude,
            help_request.latitude,
            help_request.longitude,
        )
        return distance_km <= profile.radius_km

    volunteer_address = (volunteer.address or
"").casefold()
    request_address = (help_request.address or
"").casefold()
    if not volunteer_address or not request_address:
        return False

    return any(
        len(part) >= 3 and part in request_address
        for part in volunteer_address.replace(",", " ").split()
    )


@receiver(post_save, sender=HelpRequest)
def on_new_request_created(sender, instance, created,
**kwargs):
    """Notify matching verified volunteers when a new
active request is created."""
    if not created:
        return
    if instance.status != HelpRequest.Status.ACTIVE:
        return

    volunteers = (
        User.objects.filter(
            user_type=User.UserType.VOLUNTEER,
            is_active=True,
            is_verified=True,
            volunteer_profile__is_available=True,
        )
        .select_related("volunteer_profile")
        .prefetch_related("volunteer_profile__categories")
    )

    for volunteer in volunteers:
        if not _volunteer_matches_request(volunteer,
instance):
            continue
        Notification.objects.get_or_create(
            user=volunteer,

type=Notification.Type.NEW_NEARBY_REQUEST,
            related_request=instance,
            defaults={
                "title": "Новий запит поблизу",
                "message": f"Поруч з вами з'явився запит:
'{instance.title}'.",
            },
        )


@receiver(post_save, sender=Response)
def on_response_received(sender, instance, created,
**kwargs):
    """Notify the recipient when a volunteer submits a
new response to their request."""
    if not created:
        return
    Notification.objects.get_or_create(
        user=instance.help_request.recipient,
        type=Notification.Type.NEW_RESPONSE,
        related_request=instance.help_request,
        defaults={
            "title": f"{_display_name(instance.volunteer)}
відгукнувся(лась) на ваш запит",
            "message": (
                f"Запит: '{instance.help_request.title}'. "
                "Перегляньте відгуки та оберіть волонтера."
            ),
        },
    )


# apps/notifications/views.py
class NotificationListView(LoginRequiredMixin,
ListView):
    """Пагінований список сповіщень поточного
користувача."""
```

```
    model = Notification
    template_name = "notifications/notification_list.html"
    context_object_name = "notifications"
    paginate_by = 20

    def get_queryset(self):
        return
Notification.objects.filter(user=self.request.user)


@login_required
def mark_read(request, pk):
    """Позначає одне сповіщення як прочитане."""
    if request.method != "POST":
        return HttpResponseNotAllowed(["POST"])
    notification = get_object_or_404(Notification, pk=pk,
user=request.user)
    notification.is_read = True
    notification.save(update_fields=["is_read"])
    messages.success(request, "Сповіщення позначено
як прочитане")
    return redirect("notifications:notification-list")


@login_required
def mark_all_read(request):
    """Позначає всі непрочитані сповіщення як
прочитані."""
    if request.method != "POST":
        return HttpResponseNotAllowed(["POST"])
    Notification.objects.filter(user=request.user,
is_read=False).update(is_read=True)
    messages.success(request, "Всі сповіщення
позначено як прочитані")
    return redirect("notifications:notification-list")


@login_required
@require_GET
@never_cache
def notification_count(request):
    """Повертає кількість непрочитаних сповіщень у
форматі JSON для polling."""
    count = Notification.objects.filter(user=request.user,
is_read=False).count()
    return JsonResponse({"count": count})


# apps/reviews/models.py
class Review(models.Model):
    """Відгук після виконання запиту допомоги."""

    author = models.ForeignKey(
        'accounts.User', on_delete=models.CASCADE,
        related_name='reviews_written',
verbose_name='Автор'
    )
    target = models.ForeignKey(
        'accounts.User', on_delete=models.CASCADE,
        related_name='reviews_received',
verbose_name='Про кого'
    )
    help_request = models.ForeignKey(
        'requests.HelpRequest',
on_delete=models.CASCADE,
        related_name='reviews', verbose_name='Запит
допомоги'
    )
    rating = models.IntegerField(
        'Оцінка', validators=[MinValueValidator(1),
MaxValueValidator(5)]
    )
    comment = models.TextField('Коментар')
    created_at = models.DateTimeField('Дата створення',
auto_now_add=True)

    class Meta:
        verbose_name = 'Відгук'
        verbose_name_plural = 'Відгуки'
        ordering = ['-created_at']
        unique_together = ['author', 'help_request']

    def __str__(self):
        return f"{self.author} -> {self.target}:
{self.rating}/5"
```