НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

«На правах рукопису»
УДК 681.3

До захисту допущено:
В.О. Завідувача кафедри
____Михайло Новотарський
«___»_____2025 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

зі спеціальності 123 «Комп'ютерна інженерія»

на тему: «Веб-платформа для організації волонтерської діяльності в Україні»

Виконала:
студентка ІІ курсу, групи ІО-41мп
Герасименко Марія Ігорівна _____

Керівник:
ст. викл., к.т.н.
Кулаков Олексій Юрійович _____

Консультант з нормоконтролю:
проф., д.т.н.
Кулаков Юрій Олексійович _____

Рецензент:

_____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *Обчислювальної техніки*
(повна назва)

Освітньо-кваліфікаційний ступінь *магістр*
(назва ОКР)

Спеціальність *123. Комп'ютерна інженерія*
(код і назва)

Спеціалізація *123. Комп'ютерні системи та мережі*
(код і назва)

ЗАТВЕРДЖУЮ
В.О. Завідувача кафедри
Михайло Новотарський
(підпис) (ініціали, прізвище)
« » _____ 2025 р.

ЗАВДАННЯ
на магістерську дисертацію студентці
Герасименко Марії Ігорівні
(прізвище, ім'я, по батькові)

1. Тема дисертації «Веб-платформа для організації волонтерської діяльності в Україні»
Науковий керівник дисертації Кулаков Олексій Юрійович, старший викладач, кандидат технічних наук
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від «06» 11 2025 р. № 4841-с

2. Строк подання студентом дисертації 01.12.2025

3. Об'єкт дослідження організація та координація волонтерства за допомогою веб-платформи

4. Предмет дослідження методи та програмні засоби розробки веб-платформи для організації та координації волонтерської діяльності

5. Перелік завдань, які потрібно розробити:

1. Дослідження сучасних методів і цифрових рішень для координації волонтерської діяльності.
2. Вивчення актуальних потреб та викликів волонтерського руху в умовах воєнного стану в Україні.
3. Розроблення інформаційної платформи для волонтерів.

4. Проведення тестування створеної системи.

6. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	проф. каф. ОТ д.т.н., Кулаков Ю.О.		
2	проф. каф. ОТ д.т.н., Кулаков Ю.О.		
3	проф. каф. ОТ д.т.н., Кулаков Ю.О.		
4	проф. каф. ОТ д.т.н., Кулаков Ю.О.		

7. Дата видачі завдання 01 вересня 2025

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
1.	Затвердження теми роботи	01.09.25 – 07.09.25	
2.	Аналіз предметної області та сучасних підходів до організації волонтерської діяльності	08.09.25 – 21.09.25	
3.	Вивчення потреб волонтерського руху та визначення вимог до інформаційної системи	22.09.25 – 02.10.25	
4.	Проєктування та розроблення веб-платформи для організації волонтерської діяльності	03.10.25 – 07.11.25	
5.	Тестування функціональності та оцінювання ефективності роботи системи	08.11.25 – 14.11.25	
6.	Підготовка та оформлення матеріалів магістерської дисертації	15.11.25 – 30.11.25	

Студентка _____ Марія ГЕРАСИМЕНКО
(підпис)

Науковий керівник дисертації _____ Олексій КУЛАКОВ
(підпис)

Реферат

на магістерську дисертацію

виконану на тему: *Веб-платформа для організації волонтерської діяльності в Україні*

студенткою: Герасименко Марією Ігорівною

Робота складається зі вступу та чотирьох розділів. Загальний обсяг роботи аркушів основного тексту 115, ілюстрацій 34, таблиць 25. При підготовці використано літературу з джерел.

Актуальність теми. З огляду на сучасні виклики, зокрема повномасштабну війну в Україні, волонтерський рух став ключовим фактором підтримки суспільства. Волонтери взяли на себе значну частку завдань, пов'язаних з допомогою армії, переміщеним особам, людям похилого віку та іншим групам. Водночас, зростання масштабів та різноманітності волонтерської діяльності вимагає вдосконалення механізмів комунікації для забезпечення оперативного та ефективного задоволення потреб громадян.

Тому створення веб-платформи для організації волонтерської діяльності є актуальним питанням. Такий інструмент об'єднає ініціативи, забезпечить прозорість процесів, оптимізує обробку заявок та сприятиме швидкому реагуванню. Крім того, цифровізація волонтерського руху стане важливим кроком у його подальшому розвитку, забезпечуючи безперервність, довіру та ефективність у довгостроковій перспективі.

Мета і завдання дослідження. Метою магістерської роботи є розробка веб-платформи для організації волонтерської діяльності в Україні. Ця платформа має забезпечити ефективну взаємодію між волонтерами, організаціями та громадянами, сприяти прозорості процесів та збільшити довіру до волонтерського руху.

Завдання:

1. Огляд сучасних підходів та платформ для організації волонтерської діяльності.

2. Аналіз потреб волонтерського руху в умовах війни в Україні.
3. Практична реалізація прототипу платформи.
4. Тестування та перевірка працездатності системи.

Об’єкт дослідження. Організація та координація волонтерства за допомогою веб-платформи.

Предмет дослідження. Методи та програмні засоби розробки веб-платформи для організації та координації волонтерської діяльності.

Наукова новизна. Запропоновано спосіб побудови веб-платформи для комплексної організації та координації волонтерської діяльності в Україні, який поєднує сучасні веб-технології з інструментами для прозорої взаємодії між волонтерами, організаціями та особами, які потребують допомоги. Впроваджено цифрову модель комунікації, що підвищує ефективність обробки запитів та забезпечує вищий рівень довіри до волонтерського руху.

Практичне значення одержаних результатів. Практичне значення результатів полягає в тому, що розроблена веб-платформа може бути використана для підвищення ефективності волонтерства в Україні. Це рішення спрощує взаємодію між волонтерами та організаціями, забезпечує прозорість процесів та пришвидшує реагування на запити громадян. Створений прототип може слугувати основою для стартап-проекту, що сприяє масштабуванню волонтерського руху та залученню нових учасників.

Ключові слова. ВЕБ-ПЛАТФОРМА, АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, КЛІЄНТ-СЕРВЕРНА МОДЕЛЬ, БАЗА ДАНИХ, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС, ФРОНТЕНД, БЕКЕНД, МОДЕЛЮВАННЯ СИСТЕМИ.

ABSTRACT

The thesis consists of an introduction and five chapters. The total volume of the work includes the main text 115, illustrations 34, and tables 25. In the preparation of the thesis, literature from various sources was used.

Relevance of the topic. Taking into account the current challenges, in particular the full-scale war in Ukraine, the volunteer movement has become a key factor in supporting society. Volunteers have taken on a significant share of tasks related to assisting the army, displaced persons, the elderly, and other groups. At the same time, the growing scale and diversity of volunteer activities requires the improvement of communication mechanisms to ensure the prompt and effective satisfaction of citizens' needs.

Therefore, the creation of a web platform for the organization of volunteer activities is a highly relevant issue. Such a tool will unite initiatives, ensure process transparency, optimize request processing, and promote rapid response. In addition, the digitalization of the volunteer movement will become an important step in its further development, ensuring continuity, trust, and efficiency in the long term.

Purpose and objectives of the study. The purpose of this master's thesis is to develop a web platform for organizing volunteer activities in Ukraine. This platform is intended to ensure effective interaction between volunteers, organizations, and citizens, promote process transparency, and increase trust in the volunteer movement.

Objectives:

1. Review of modern approaches and platforms for organizing volunteer activities.
2. Analysis of the needs of the volunteer movement in the conditions of war in Ukraine.
3. Practical implementation of the platform prototype.
4. Testing and verification of the system's functionality.

Object of research. Organization and coordination of volunteering through a web platform.

Subject of research. Methods and software tools for developing a web platform for the organization and coordination of volunteer activities.

Scientific novelty. A method of building a web platform for the comprehensive organization and coordination of volunteer activities in Ukraine is proposed, which combines modern web technologies with tools for transparent interaction between volunteers, organizations and individuals in need of assistance. A digital communication model is implemented, which increases the efficiency of processing requests and ensures a higher level of trust in the volunteer movement.

Practical significance of the obtained results. The practical significance of the results is that the developed web platform can be used to increase the efficiency of volunteering in Ukraine. This solution simplifies interaction between volunteers and organizations, ensures process transparency, and accelerates the response to citizens' requests. The created prototype can serve as a basis for a startup project, contributing to the scaling of the volunteer movement and the involvement of new participants.

Keywords. WEB PLATFORM, SOFTWARE ARCHITECTURE, CLIENT-SERVER MODEL, DATABASE, USER INTERFACE, FRONTEND, BACKEND, SYSTEM MODELING.

ЗМІСТ

ВСТУП 4

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ ВЕБ-ПЛАТФОРМ ДЛЯ ОРГАНІЗАЦІЇ ТА КООРДИНАЦІЇ ВОЛОНТЕРСЬКОЇ ДІЯЛЬНОСТІ 5

1.1. Актуальність теми 5

1.2. Огляд існуючих веб-платформ 7

1.2.1. Національна Волонтерська Платформа 7

1.2.2. Українська Волонтерська Служба (УВС) 9

1.2.3. Платформа для волонтерських заявок від EVO та Міністерства цифрової трансформації України 10

1.2.4. Rubikus.HelpUA 11

1.2.5. Міжнародні та діаспорні організації 12

1.3. Порівняльний аналіз існуючих веб-платформ 13

Висновок до розділу 1 16

РОЗДІЛ 2. МЕТОДИ ТА ПІДХОДИ ДО РОЗРОБКИ ВЕБ-ПЛАТФОРМИ ДЛЯ ВОЛОНТЕРСЬКОЇ ДІЯЛЬНОСТІ 17

2.1. Вимоги до базової функціональності платформи 17

2.2. Архітектурні підходи до створення веб-систем 18

2.2.1. Монолітна архітектура 18

2.2.2. Клієнт-серверна архітектура 20

2.2.3. Багатошарова архітектура 21

2.2.4. Мікросервісна архітектура 22

2.2.5. Подійно-орієнтована архітектура 23

2.2.6. Порівняльний аналіз та обґрунтування вибору архітектурного рішення 24

2.3. Вибір технологій для фронтенду 25

2.3.1 React як інструмент для створення інтерфейсу користувача 25

2.3.2 Vue.js як альтернатива для швидкої розробки веб-інтерфейсів 26

2.3.3 Angular як фреймворк для комплексних веб-застосунків 26

2.3.4 Порівняльний аналіз і обґрунтування вибору фреймворку 27

2.4 Вибір технологій для бекенду 28
2.4.1 Node.js та фреймворк Express 28
2.4.2. Django 29
2.4.3. FastAPI 30
2.4.4. Порівняльний аналіз і обґрунтування вибору 30
2.5. Бази даних та принципи їх побудови у волонтерських сервісах 32
2.5.1. Роль бази даних у веб-платформі 32
2.5.2. Реляційні та нереляційні бази даних 32
2.5.3. Проєктування схеми бази даних 33
2.5.4. Забезпечення безпеки та захисту даних 34
2.5.5. Вибір бази даних для волонтерської платформи 35
Висновок до розділу 2 36
РОЗДІЛ 3. ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБ-ПЛАТФОРМИ 37
3.1. Проєктування архітектури та компонентів системи 37
3.1.1. Загальна архітектура системи 37
3.1.2. Основні компоненти системи 39
3.1.3. Користувацькі ролі та взаємодія 40
3.1.4. Основні об'єкти бази даних 43
3.1.5. Архітектурні принципи 44
3.1.6. Взаємодія компонентів 44
3.2. Реалізація серверної частини (бекенду) 45
3.2.1. Технологічний стек та архітектура 45
3.2.2. Структура проєкту 46
3.2.3. Конфігурація бази даних 47
3.2.4. Система автентифікації та авторизації 48
3.2.5. API маршрути та ендпоінти 48
3.2.6. Валідація даних 49
3.2.7. Модулі та функції бекенду 50

3.3. Реалізація клієнтської частини (фронтенду) 54
3.3.1. Технологічний стек та архітектура 54
3.3.2. Структура проєкту 55
3.3.3. Система автентифікації та управління доступом 57
3.3.4. Основні React-компоненти та їх функціональність 59
3.3.5. Маршрутизація та навігація 61
3.4. Забезпечення безпеки та зберігання даних 62
3.5. Перевірка функціональності роботи веб-платформи 65
3.5.1. Перевірка реєстрації та автентифікації користувачів 65
3.5.2. Перевірка створення та відображення заявок 68
3.5.3. Перевірка роботи сторінки волонтера та фільтрації заявок 70
3.5.4. Перевірка відгуків волонтерів та підтвердження заявником 73
3.5.5. Перевірка виконання завдань і зміни статусів 75
3.5.6. Перевірка адміністративної панелі та журналу аудиту 78
3.5.7. Загальні результати тестування 79
Висновок до розділу 3 81
РОЗДІЛ 4. РОЗРОБКА СТАРТАП ПРОЄКТУ 82
4.1 Загальна характеристика стартап-проекту 82
4.2 Технологічний аудит проекту 85
4.3 Аналіз ринкових можливостей запуску стартап-проекту 87
4.4. Розроблення ринкової стратегії проекту 97
4.5. Розроблення маркетингової програми стартап-проекту 101
Висновок до розділу 4 106
ВИСНОВКИ 107
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 108
ДОДАТОК А 110

ВСТУП

Волонтерство в Україні є ключовим фактором підтримки суспільства під час глибоких соціальних та політичних трансформацій. Особливого значення воно набуло під час повномасштабної війни, коли волонтери стали життєво важливим ресурсом для забезпечення військової підтримки, допомоги внутрішньо переміщеним особам та іншим вразливим групам. За останні роки волонтерський рух еволюціонував від локальних ініціатив до масштабних системних процесів, що вимагають сучасних організаційних та координаційних інструментів.

Водночас, з розвитком волонтерства зростає і складність його завдань. Використання традиційних каналів комунікації, таких як соціальні мережі чи особисті контакти, не завжди забезпечує необхідний рівень ефективності, прозорості та координації. Це призводить до дублювання зусиль, втрати часу та ресурсів, що знижує ефективність допомоги та рівень довіри до волонтерських ініціатив. За таких обставин вкрай важливим є впровадження сучасних цифрових інструментів, здатних оптимізувати процеси комунікації та управління волонтерами.

Розробка веб-платформ для організації та координації волонтерської діяльності створить єдине інформаційне середовище для взаємодії між волонтерами, організаціями та громадянами. Така платформа сприятиме оптимізації процесів, забезпечить прозорість, уникне дублювання ініціатив та забезпечить своєчасне реагування на запити. Крім того, його можна масштабувати та використовувати як основу для стартап-проекту, відкриваючи перспективи для подальшого розвитку та впровадження на національному рівні.

Метою дисертації є розробка веб-платформи для організації та координації волонтерської діяльності в Україні, забезпечення ефективної взаємодії між волонтерами, організаціями та громадянами. Це має сприяти прозорості процесів, підвищувати довіру до волонтерського руху та слугувати основою для подальшої реалізації стартап-проекту.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ ВЕБ-ПЛАТФОРМ ДЛЯ ОРГАНІЗАЦІЇ ТА КООРДИНАЦІЇ ВОЛОНТЕРСЬКОЇ ДІЯЛЬНОСТІ

1.1. Актуальність теми

Волонтерство в сучасному світі – це не просто добра справа, а важлива частина активного громадянського суспільства. Воно дає людям можливість долучитися до вирішення нагальних проблем, допомогти нужденним та зміцнити взаємодопомогу. В умовах глобальних викликів, таких як війна, економічні кризи чи катастрофи, саме волонтери часто першими приходять на допомогу та беруться за завдання, які держава чи великі організації не встигають виконати. Тому розвиток волонтерського руху є показником зрілості суспільства, його згуртованості та готовності до самоорганізації.

Волонтерство має особливе значення для України. Все почалося після Революції Гідності, коли тисячі людей об’єдналися, щоб допомогти армії, переселенцям та постраждалим від зони бойових дій. Нині виникло багато громадських ініціатив, благодійних фондів та спільнот у соціальних мережах, які згодом стали потужними організаціями. Саме тоді волонтерство перестало бути чимось рідкісним і перетворилося на масовий рух, закладаючи основу для нової, більш активної громадянської культури.

З початку повномасштабного вторгнення у 2022 році волонтерський рух набув особливого значення. Сотні тисяч українців щодня стикаються з новими труднощами: необхідністю евакуації, пошуком тимчасового житла, дефіцитом продуктів, ліків та засобів гігієни. Волонтери стали тією силою, яка швидко реагує на ці виклики. Вони організовують гуманітарні вантажі, забезпечують логістику, підтримують соціальні й медичні заклади, допомагають армії, створюють освітні та культурні ініціативи. Фактично, волонтери стали невід’ємною частиною національної системи безпеки й соціальної підтримки.

Українське волонтерство вирізняється своєю багатофункціональністю. На відміну від багатьох інших країн, де волонтерський рух зосереджений переважно на культурі чи освіті, в Україні він охоплює майже всі сфери життя. Це включає допомогу армії, соціальну підтримку, роботу з дітьми та людьми похилого віку, медичну допомогу, а також цифрові ініціативи, пов'язані з інформаційними кампаніями. Таке розмаїття напрямків пояснюється не лише масштабом викликів, а й високим рівнем відповідальності українців.

Високий рівень довіри суспільства до волонтерів є ще одним важливим фактором. Соціологічні дослідження показують, що волонтерські організації стабільно входять до переліку інституцій, яким громадяни найбільше довіряють. Це можна пояснити кількома причинами: волонтерська діяльність є часто прозорішою, ніж у державних установах, оскільки люди бачать, як їхня допомога перетворюється на реальні дії. Крім того, волонтери діють швидко та гнучко, реагуючи на конкретні запити. Громадяни також сприймають волонтерів як людей, які опинялися в подібних ситуаціях, а тому краще розуміють їхні потреби. Водночас зростання руху призвело до проблем, серед яких особливо небезпечним є псевдоволонтерство. Недобросовісна практика з боку окремих осіб чи організацій підриває довіру до руху в цілому, що вимагає створення більш прозорих механізмів моніторингу та перевірки діяльності.

В умовах цифровізації суспільства волонтерська діяльність змінює свій формат. Якщо основними інструментами комунікації раніше були особисті контакти та соціальні мережі, то сьогодні з'являється все більше спеціалізованих онлайн-платформ. Вони дозволяють людям залишати запити на допомогу, координувати дії та відстежувати виконані завдання. Однак більшість цих сервісів орієнтовані на великі організації, а не на пересічних людей. Це створює певний бар'єр для доступності: складні інтерфейси, надмірна кількість полів у формах та відсутність прозорої системи відстеження статусу заявок роблять інструменти незручними для швидкого використання.

Таким чином, актуальність створення нової веб-платформи для волонтерської діяльності визначається низкою важливих факторів. Сучасне

українське суспільство стикається з масштабними та системними викликами, пов'язаними з війною, що вимагають швидкої мобілізації ресурсів та ефективної координації між усіма учасниками волонтерського руху. Високий рівень суспільної довіри до волонтерів потребує підтримки та збереження, що можливо лише завдяки використанню прозорих та перевірених механізмів комунікації, що гарантують чесність та підзвітність. Важливим фактором також є цифровізація, оскільки сучасні технології можуть оптимізувати процес обробки заявок, пришвидшити реагування та допомогти зробити його доступнішим. Згідно з результатами дослідження [1], еволюція українського волонтерського руху характеризується переходом від початкової фази спонтанної самоорганізації до формування системно організованого соціального явища.

Водночас, інші дослідники, зокрема Ю. Маційвський [2], вважають, що збереження неформальних практик ускладнює їх розвиток, і це підтверджує необхідність впровадження сучасних технологічних рішень. Все це свідчить про те, що створення сучасної веб-платформи для координації волонтерської діяльності є надзвичайно актуальним завданням. Такий інструмент може поєднати переваги цифрових технологій з потребами суспільства, зробити процес надання допомоги прозорішим, швидшим та доступнішим.

1.2. Огляд існуючих веб-платформ

1.2.1. Національна Волонтерська Платформа

Національна Волонтерська Платформа – один із ключових державних та громадських проєктів, створених для координації та підтримки волонтерського руху в Україні. Її головна мета – створення єдиного простору, де зустрінуться потреби суспільства та бажання людей допомогти. Завдяки платформі кожен може знайти актуальні волонтерські можливості, а громадські організації можуть залучати волонтерів до своїх програм.

Функціонал платформи характеризується широким спектром можливостей, що робить її універсальним інструментом. На сайті представлені пропозиції в різних сферах: від гуманітарної допомоги та соціальних проектів до культурних та освітніх ініціатив. Це робить ресурс універсальним, оскільки він підходить людям різного віку, професій та з різним досвідом. Крім того, платформа інформує суспільство про історії успіху, мотивує громадян до участі та створює позитивний імідж волонтерства.

Важливо, що Національна Волонтерська Платформа працює за підтримки державних інституцій, зокрема, Міністерства молоді та спорту України та Української волонтерської служби. Це забезпечує додаткову легітимність та довіру. Платформа допомагає волонтерам працювати злагоджено, а також сприяє тому, щоб волонтерський рух визнали на державному рівні.

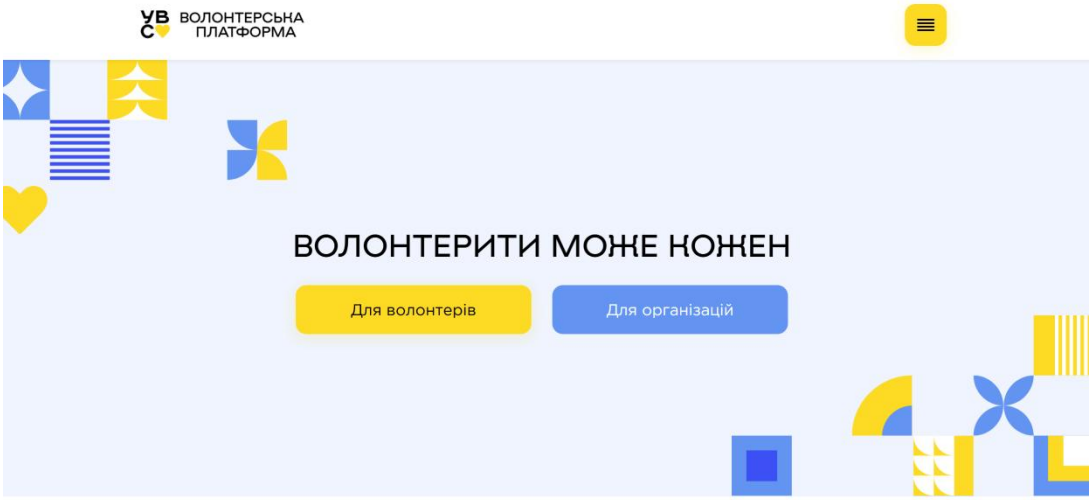

Рис. 1.1. Головна сторінка Національної Волонтерської Платформи

Хоча платформа має значні переваги, вона також має певні обмеження. Вона в першу чергу орієнтована на великі та довгострокові проекти, що іноді ускладнює її використання для нагальних індивідуальних потреб. Для людей, які шукають негайної допомоги на місці, процес реєстрації та подання заявки може здатися складним та надто формальним. Це не применшує цінності платформи, але свідчить про те, що її функціональність більше підходить для стратегічного планування, ніж для швидкої допомоги. Загалом, Національна

Волонтерська Платформа є важливим інструментом розвитку волонтерства в Україні. Вона об'єднує багатьох людей та організацій, допомагає поширювати цінності волонтерства та формувати культуру взаємодопомоги. Однак вона потребує доповнення іншими, більш гнучкими цифровими інструментами, які б краще відповідали потребам окремих користувачів та дозволяли швидше реагувати на запити.

У статті А. Лобуренка [3] наголошується, що такі національні ініціативи відіграють важливу роль у консолідації суспільства, оскільки вони поєднують державну підтримку з активністю громадян, формуючи стабільну основу для розвитку волонтерства.

1.2.2. Українська Волонтерська Служба (УВС)

Українська Волонтерська Служба (УВС) є однією з ключових організацій, що розвивають волонтерський рух в Україні. Їхній підхід є системним та стратегічним: УВС не лише організовує волонтерські ініціативи, а й формує довгострокову культуру взаємодопомоги в суспільстві. На відміну від багатьох інших платформ, які переважно виконують координаційну функцію, УВС поєднує практику з освітою, забезпечуючи сталий розвиток руху.

Організація приділяє особливу увагу роботі з молоддю. За допомогою програм, тренінгів, таборів та форумів УВС прищеплює новим поколінням почуття відповідальності та важливості громадянської активності. Молодь отримує не лише можливість спробувати себе у волонтерстві, а й доступ до знань, що допомагають розвивати лідерські та комунікативні навички. Це створює довгостроковий ефект, адже залучена молодь зможе в майбутньому ініціювати власні проекти та поширювати культуру волонтерства у своєму середовищі.

Ще однією сильною стороною УВС є велика мережа контактів. Організація співпрацює з громадськими об'єднаннями, благодійними фондами, навчальними закладами, бізнесом та міжнародними партнерами. Така співпраця дозволяє реалізовувати масштабні соціальні ініціативи та поширювати

український досвід на міжнародному рівні. Фактично, УВС є сполучною ланкою між локальними ініціативами та глобальними проєктами, що значно підвищує ефективність роботи.

Водночас діяльність організації має певні обмеження. Більшість проектів, що реалізуються УВС, є масштабними та довгостроковими, тому вони менш зручні для людей, які потребують екстреної та місцевої допомоги. Крім того, зосередженість на освітній та культурній сферах означає, що повсякденні або термінові індивідуальні запити часто ігноруються. Це створює простір для доповнення роботи УВС іншими цифровими рішеннями, які можуть забезпечити легше спілкування між волонтерами та тими, хто потребує допомоги.

Українська Волонтерська Служба (УВС) відіграє надзвичайно важливу роль у розвитку волонтерського руху. Вона не лише координує діяльність, а й формує систему суспільних цінностей, сприяє розвитку громадянської активності та забезпечує безперервність руху. Згідно з висновками Панченка А. О., викладеними у статті [4], саме завдяки діяльності подібних організацій волонтерський рух в Україні поступово трансформується зі стихійної активності у системно організоване та структуроване явище.

1.2.3. Платформа для волонтерських заявок від EVO та Міністерства цифрової трансформації України

Одним із сучасних цифрових рішень у сфері волонтерства є платформа для заявок волонтерів, створена EVO у співпраці з Міністерством цифрової трансформації України. Її головна мета – спростити взаємодію між людьми, які її потребують, та волонтерськими організаціями. Цей проект став відповіддю на виклики повномасштабної війни, коли кількість запитів на гуманітарну, соціальну та логістичну підтримку зросла в кілька разів, а традиційні канали комунікації (соціальні мережі, месенджери) виявилися неефективними.

Функціонал платформи дозволяє створити застосунок, у якому користувач може описати свою потребу, додати документи чи фотографії, а

також отримати зворотний зв'язок від волонтерів. Це робить процес більш структурованим та прозорим. Крім того, система може відстежувати статус заявки, дозволяючи заявникам розуміти, на якій стадії знаходиться їхній запит і хто взявся його виконати. Для волонтерських організацій така платформа є зручним інструментом управління, оскільки спрощує облік запитів, розподіл завдань та контроль за їх виконанням.

Сильними сторонами цієї платформи є її сучасний інтерфейс, централізація та державна підтримка. Завдяки цифровим технологіям вдається значно скоротити час між поданням заявки та її обробкою. Платформа також сприяє підвищенню довіри до волонтерського руху, оскільки забезпечує прозорість процесів та зменшує ризик дублювання запитів або шахрайства.

Однак, поряд із перевагами, є й певні обмеження. Платформа в основному орієнтована на співпрацю з великими волонтерськими групами та організаціями. Для звичайних людей, які потребують екстреної та місцевої допомоги, процес реєстрації та створення заявки може бути надмірно складним. На практиці виявляється, що надмірна офіційність системи системи знижує її зручність для простих повсякденних запитів, які потребують мінімального часу на вирішення.

У дослідженні Климчука В. Г. [5] зазначається, що сучасні волонтерські ініціативи у воєнний час стають більш цифровізованими та інституціоналізованими, тобто набувають офіційного статусу та структури. Це підкреслює важливість створення ефективних онлайн-платформ для підвищення прозорості та ефективності. Тому платформу від EVO та Міністерства цифрової трансформації можна вважати важливим кроком до професіоналізації волонтерської діяльності в Україні, хоча вона все ще потребує адаптації до потреб окремих громадян.

1.2.4. Rubikus.HelpUA

Rubikus.HelpUA – це спеціалізована волонтерська ініціатива, що виникла у відповідь на нагальну потребу безпечної евакуації людей з небезпечних

регіонів під час війни. Її діяльність відрізняється від універсальних платформ вузькою, але дуже важливою сферою роботи: організація транспортування та логістичної допомоги цивільним особам, які опинилися в критичних умовах.

Головна мета Rubikus.HelpUA – рятувати життя. Організація допомагає людям похилого віку, сім'ям з дітьми, людям з інвалідністю, а також тим, хто не може евакуюватися самостійно. Координація відбувається швидко та чітко: платформа збирає заявки, координує маршрути та транспорт. Завдяки цій спеціалізації ініціатива здобула довіру та стала важливим інструментом забезпечення безпеки.

Важливою особливістю Rubikus.HelpUA є те, що вона працює на перетині кількох напрямків волонтерства: гуманітарного, соціального та логістичного. Це означає, що волонтери не лише перевозять людей до безпечних місць, але й часто допомагають їм з тимчасовим розміщенням, пошуком житла, їжі чи ліків. Таким чином, ініціатива виконує комплексну функцію підтримки, виходячи далеко за рамки простого «транспортування».

Незважаючи на очевидні переваги, діяльність Rubikus.HelpUA також має деякі обмеження. Оскільки організація зосереджена лише на евакуації, вона не може охопити всі інші напрямки волонтерства – культурні, освітні чи побутові потреби громадян. Це робить її важливою, але все ж таки спеціалізованою частиною загальної екосистеми волонтерських послуг в Україні.

Як наголошує Маційовський Ю. [6], волонтерський рух в Україні є багатовимірним і включає як універсальні, так і вузькоспеціалізовані сфери діяльності. Досвід Rubikus.HelpUA показує, що спеціалізація дозволяє досягти високої ефективності в конкретній сфері, але вимагає тісної співпраці з іншими організаціями для комплексного задоволення потреб населення.

1.2.5. Міжнародні та діаспорні організації

Окрему та дуже важливу роль у розвитку волонтерського руху в Україні відіграють міжнародні та діаспорні організації. Вони є сполучною ланкою між українським суспільством та світовою спільнотою, мобілізуючи ресурси з-за

кордону для підтримки армії та цивільного населення. Серед найвідоміших ініціатив – Hope for Ukraine, Nova Ukraine, а також численні фонди, створені українською діаспорою в США, Канаді, Німеччині та інших країнах.

Головною перевагою міжнародних та діаспорних організацій є їхня здатність залучати фінансову та гуманітарну допомогу у великих масштабах. Завдяки партнерству з іноземними бізнес-структурами та благодійними фондами вони мають доступ до ресурсів, які було б важко зібрати всередині країни. Це дозволяє їм реалізовувати масштабні програми від закупівлі медичного обладнання та транспортних засобів для армії до організації гуманітарної допомоги та культурних проектів для переміщених осіб.

Міжнародні та діаспорні організації також відіграють важливу роль у наданні інформаційної підтримки Україні. Вони поширюють достовірну інформацію про події, організовують акції солідарності та відстоюють інтереси України на міжнародному рівні. Завдяки цьому світова спільнота приділяє більше уваги українським проблемам, що сприяє залученню нових донорів та партнерів.

Водночас діяльність таких організацій має свої особливості. Вони, як правило, зосереджені на великих стратегічних завданнях та масштабних проектах, тоді як окремі локальні запити залишаються поза їхньою увагою. Це логічно, адже координація повсякденних потреб вимагає іншого рівня організації та ресурсів. Тому міжнародні та діаспорні фонди можуть замінити місцеві волонтерські ініціативи, але ефективно доповнювати їх.

Як зазначає Проценко А. А. [7], волонтерський рух в Україні – це багатошарове явище, яке об'єднує як внутрішні, так і зовнішні ресурси. Діаспора та міжнародні організації є потужним фактором розвитку, забезпечуючи синергію між українським суспільством та світовою спільнотою. Саме завдяки їхній діяльності волонтерство в Україні набуло глобального виміру та стало прикладом міжнародної солідарності.

1.3. Порівняльний аналіз існуючих веб-платформ

Проведений аналіз показує, що волонтерські платформи та організації в Україні суттєво відрізняються за масштабом, спрямованістю, підтримкою та цільовою аудиторією.

Національна Волонтерська Платформа — це стратегічний інструмент із державною підтримкою, призначений для об’єднання великої кількості людей та організацій. Вона ефективна для реалізації масштабних і довгострокових проєктів, але не завжди зручна для швидких індивідуальних запитів.

Українська Волонтерська Служба (УВС) вирізняється акцентом на формуванні цінностей та залученні молоді до соціальної активності. Це сприяє довгостроковому розвитку волонтерства, але водночас обмежує швидкість реагування на нагальні потреби населення.

Rubikus.HelpUA демонструє ефективність вузької спеціалізації. Завдяки чіткій спрямованості на евакуацію людей з небезпечних районів, ця ініціатива досягає високих результатів у своїй галузі. Однак, через обмежену сферу діяльності, вона не може забезпечити комплексний підхід до всіх потреб населення.

Міжнародні та діаспорні організації (Hope for Ukraine, Nova Ukraine та інші) надають масштабну фінансову, гуманітарну та інформаційну підтримку. Їхня головна перевага – доступ до іноземних ресурсів та можливість реалізації великих програм. Водночас вони рідко працюють з місцевими індивідуальними запитами, оскільки їхня діяльність спрямована на стратегічні завдання.

Для більшої чіткості аналізу доцільно узагальнити його у вигляді порівняльної таблиці. Це допоможе продемонструвати основні характеристики, переваги та недоліки найвідоміших волонтерських платформ та організацій в Україні. Такий підхід дозволить чітко побачити відмінності між існуючими рішеннями, виявити їхні сильні та слабкі сторони й обґрунтувати необхідність створення нової веб-платформи, яка поєднає найкращі практики та усуне недоліки.

Таблиця 1.1

Порівняльна характеристика волонтерських платформ

Платформа / Організація	Основний фокус	Сильні сторони	Обмеження	Рівень підтримки
Національна Волонтерська Платформа	Масштабні проєкти, залучення організацій	Державна підтримка, велика база користувачів, популяризація волонтерства	Менш зручна для індивідуальних запитів, неоптимальна структура	Держава, Українська Волонтерська Служба
Українська Волонтерська Служба (УВС)	Освіта, молодіжні програми, культура волонтерства	Формування цінностей, виховання молоді, міжнародні партнерства	Переважно довгострокові ініціативи, низька оперативність	Громадські та міжнародні організації
Платформа EVO + Мінцифра	Подання та обробка заявок	Сучасний інтерфейс, прозорість, централізованість	Орієнтація на організації, складність для простих користувачів	Державна підтримка (Мінцифра)
Rubikus.HelpUA	Евакуація та логістика	Вузька спеціалізація, висока ефективність у конкретній сфері	Обмеженість сфери діяльності, відсутність універсальності	Громадські ініціативи
Міжнародні та діаспорні організації	Масштабні програми, гуманітарна допомога, адвокація	Доступ до фінансів і ресурсів з-за кордону, глобальна підтримка	Не охоплюють локальних індивідуальних потреб	Діаспора, міжнародні донори

Загалом, аналіз показує, що в Україні вже існує ціла екосистема волонтерських служб, але кожна з них має свої сильні сторони та обмеження. Жодна з існуючих платформ не може повністю задовольнити потреби всіх користувачів: від людей, які потребують термінової індивідуальної допомоги, до організацій, які планують довгострокові програми. Це створює передумови для розробки нового веб-рішення, яке поєднує переваги різних підходів: простоту використання, прозорість, швидкість реагування та універсальність.

Висновок до розділу 1

Дослідження предметної області показало, що волонтерство в Україні є ключовим елементом громадянського суспільства. Воно динамічно розвивається у відповідь на суспільні виклики, зокрема під час війни, гуманітарних криз та внутрішнього переміщення. Дослідження існуючих інформаційно-технологічних рішень виявило як їхні сильні сторони, так і суттєві обмеження, що зумовлює необхідність розробки нової, більш зручної та гнучкої веб-платформи для ефективної координації волонтерських ініціатив.

Більшість сервісів орієнтовані на великі організації або мають надмірно складні інтерфейси, що ускладнює ефективну взаємодію окремих осіб. Частина сервісів має технологічні обмеження та недостатню адаптивність. Це свідчить про прогалини у впровадженні доступності, зручності використання та персоналізації.

Аналіз визначив ключові вимоги до майбутньої системи. Вона має бути простою у використанні, забезпечувати мінімальний поріг входу для нових користувачів, підтримувати прозору реєстрацію заявок, забезпечувати швидкі відповіді волонтерів та надавати базові інструменти модерації для адміністраторів. З технічної точки зору, платформа повинна підтримувати доступ через сучасні браузери та мати адаптивний інтерфейс.

У дослідженні особливий акцент робиться на безпеці даних, оскільки волонтерські служби обробляють особисту та контактну інформацію користувачів. Тому майбутні розробки включають автентифікацію, авторизацію, хешування паролів та обмеження доступу на основі ролей користувачів. Це допоможе зміцнити довіру до системи та забезпечити відповідність сучасним стандартам інформаційної безпеки.

Було надано системний аналіз поточного стану цифрової волонтерської інфраструктури України, визначено ключові проблеми існуючих платформ та обґрунтовано необхідність нового рішення, орієнтованого на простоту, прозорість та доступність.

РОЗДІЛ 2
МЕТОДИ ТА ПІДХОДИ ДО РОЗРОБКИ ВЕБ-ПЛАТФОРМИ ДЛЯ ВОЛОНТЕРСЬКОЇ ДІЯЛЬНОСТІ

2.1. Вимоги до базової функціональності платформи

Основне призначення веб-платформи для організації волонтерської допомоги полягає в реалізації механізму взаємодії між громадянами, які потребують підтримки, та волонтерами, які готові її надавати. Основна мета — створення зручного, швидкого та безпечного цифрового середовища для координації запитів, реєстрації волонтерів та обробки інформації щодо наданої допомоги. Платформа орієнтована на використання в межах міста Києва та підтримує три основні ролі: заявник, волонтер та адміністратор, кожна з яких має окремі сценарії роботи та доступ до відповідного функціоналу.

Система користувача забезпечує просту та безпечну процедуру створення облікового запису. При реєстрації користувач вводить ім'я, прізвище, адресу електронної пошти, пароль, номер телефону та вибирає роль заявника або волонтера. Роль адміністратора призначається вручну через панель керування. Після перевірки облікових даних система генерує JWT-токен, дійсний протягом 24 годин, що дозволяє автентифікацію без повторного входу. Кожна роль має свою домашню сторінку, що спрощує навігацію: заявник — сторінку створення та управління заявками; волонтер — список активних заявок; адміністратор — панель керування.

Ключовою функцією платформи є система подання заявок на допомогу. Заявник може створити запит, заповнивши у формі обов’язкові поля: назва запиту (короткий опис потреби), детальний опис, категорія допомоги, місто — Київ, район міста (один з 10 адміністративних районів), контактна інформація — номер телефону або Telegram. Після створення заявка отримує статус «створено» та стає доступною волонтерам. Для зручності передбачено фільтрацію за категорією, районом і рівнем терміновості. Під час перегляду

детальної інформації волонтери бачать лише загальну інформацію заявки — контактні дані заявника відображаються лише після підтвердження відгуку. Заявник може видалити заявку або позначити її як виконану, при цьому всі зміни фіксуються в журналі аудиту.

Волонтер, переглядаючи список активних заявок, може обрати потрібну та натиснути кнопку «Відгукнутись на заявку», після чого заявник у своєму кабінеті бачить перелік відгуків у вигляді імен волонтерів без персональних даних. Заявник може обрати один з відгуків, натиснувши «Прийняти» або «Відхилити». У разі підтвердження система змінює статус заявки на «прийнято», після чого волонтер отримує доступ до контактів заявника для подальшої координації допомоги. У разі відхилення відгуку заявка залишається у статусі «створено» і доступна для інших волонтерів, при цьому жодні контакти заявника не розкриваються. Волонтер може переглядати список своїх активних завдань та позначати їх як виконані, але остаточне завершення заявки виконує лише заявник.

Адміністративна панель забезпечує повний контроль за роботою системи. Адміністратори можуть переглядати список користувачів, керувати ролями, видаляти облікові записи та реагувати на порушення. Вони також мають доступ до всіх заявок, можуть змінювати їх статус та контролювати перебіг виконання кожної заявки. Для забезпечення прозорості має бути впроваджено журнал аудиту, в якому зберігається історія всіх дій: створення, редагування, видалення заявок, зміна ролей користувачів тощо. Це дозволяє підтримувати високий рівень безпеки, відповідальності та довіри до платформи.

2.2. Архітектурні підходи до створення веб-систем

2.2.1. Монолітна архітектура

Монолітна архітектура є одним із найстаріших і найпоширеніших підходів до побудови веб-систем. У цій структурі весь програмний додаток, включаючи інтерфейс користувача, бізнес-логіку та рівень доступу до даних,

розгортається як єдиний програмний блок. Така організація системи приваблива для невеликих або експериментальних проєктів завдяки своїй відносній простоті. Розробники економлять час на інтеграцію окремих компонентів та підтримку складної інфраструктури, оскільки вся функціональність зосереджена в одній програмі. Це дозволяє швидко створювати початкову версію продукту та його розгортати, використовуючи мінімальні технічні ресурси.

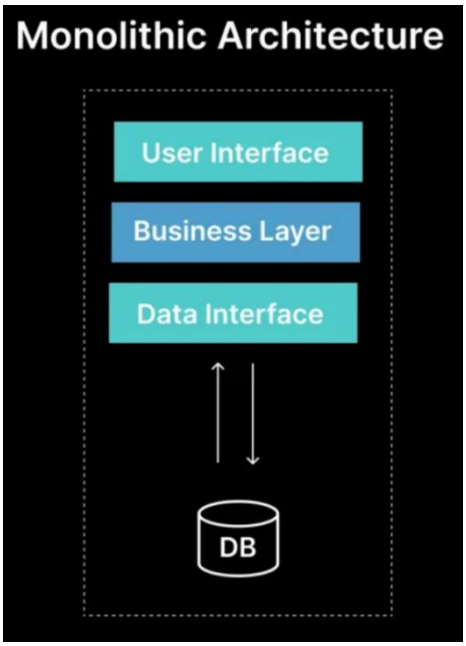

Рис. 2.1. Приклад монолітної архітектури [8]

У роботі Фаулера [9] підкреслюється, що монолітні архітектури є природною відправною точкою для більшості веб-проєктів завдяки простоті їх розгортання та швидкому прототипуванню. Однак, зі зростанням масштабу продукту, ця архітектура стає обмежувальним фактором: будь-які зміни в одному модулі можуть вплинути на стабільність усієї системи. Для волонтерської платформи це означає, що зі збільшенням кількості користувачів і запитів буде важко впроваджувати нові функції без ризику порушення існуючих процесів.

У процесі розвитку функціональності виникають труднощі з обслуговуванням: додавання нових модулів або модифікація існуючих часто впливає на інші частини коду, збільшуючи ризик помилок та ускладнюючи

стабільність системи. Для волонтерської платформи, яка передбачає роботу з різними категоріями користувачів, модуль управління запитами, аналітику та сповіщення, монолітна модель швидко стане неефективною. Тому її слід розглядати лише як проміжне рішення на початковому етапі прототипування.

2.2.2. Клієнт-серверна архітектура

Клієнт-серверна архітектура є класичним та найпоширенішим стандартом у веб-розробці. У цій моделі функції системи чітко розділені: клієнт відповідає за взаємодію з користувачем та формування запитів, тоді як сервер обробляє ці запити, реалізує бізнес-логіку та взаємодіє з базою даних. Зв'язок між компонентами відбувається через мережу за допомогою протоколів HTTP/HTTPS та обміну даними у форматі JSON або XML.

У праці Таненбаум [10] зазначає, що клієнт-серверний підхід забезпечує кращу масштабованість та контроль доступу до ресурсів порівняно з традиційним монолітом. Це досягається шляхом розподілу відповідальності між клієнтським і серверним рівнями, а також централізованим виконанням операцій з даними на сервері. Така централізація спрощує реалізацію механізмів автентифікації та авторизації, забезпечує ефективне кешування та балансування навантаження, а також гарантує надійніший захист конфіденційної інформації.

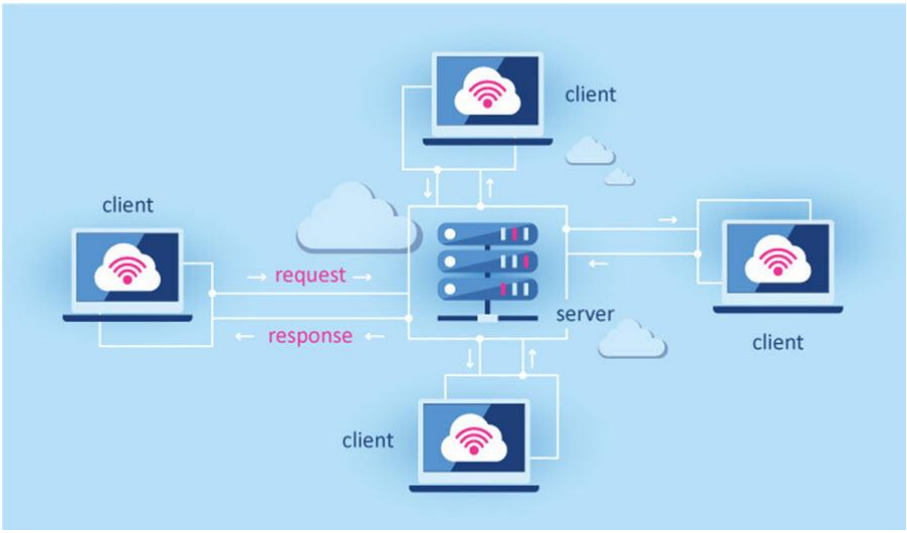

Рис. 2.2. Приклад клієнт-серверної архітектури [11]

Ще однією перевагою клієнт-серверної архітектури є гнучкість: інтерфейс користувача може оновлюватися незалежно від логіки сервера, а вузли сервера можуть масштабуватися відповідно до фактичного навантаження. Цей підхід особливо доречний для волонтерської платформи, де очікується велика кількість подібних запитів, що створює змінне, але передбачуване навантаження.

2.2.3. Багатошарова архітектура

Багатошарова архітектура є розвитком клієнт-серверного підходу, що передбачає логічне сегментування серверного компонента на кілька окремих рівнів. Зазвичай вони включають: рівень представлення (Presentation Layer), що відповідає за обробку клієнтських запитів та генерацію відповідей; рівень бізнес-логіки (Business Logic Layer), який містить алгоритми обробки запитів, механізми перевірки прав доступу та координації взаємодії; а також рівень доступу до даних (Data Access Layer), який забезпечує надійне збереження та отримання інформації з бази даних.

Як зазначає Річардс [12], багатошарова архітектура підвищує модульність та зменшує загальну складність системи завдяки чіткому розмежуванню обов'язків. Це спрощує тестування та подальше обслуговування, а також дозволяє проводити незалежну оптимізацію або модернізацію окремих компонентів. Наприклад, модифікації логіки обробки запитів не впливають на інтерфейс користувача, а зміни в базі даних не вимагають переписування бізнес-логіки.

Для волонтерської платформи багатошарова модель є оптимальним вибором на етапі дисертації. Вона поєднує відносну простоту реалізації з гнучкістю для подальшого розвитку та формує міцну основу для потенційного переходу до складніших архітектурних рішень.

2.2.4. Мікросервісна архітектура

Мікросервісна архітектура фрагментує систему на низку автономних сервісів, кожен з яких реалізує певну бізнес-функцію, таку як керування користувачами, обробка заявок, аналітика або оповіщення. Зв'язок між цими незалежними компонентами відбувається через стандартизовані API та мережеві протоколи.

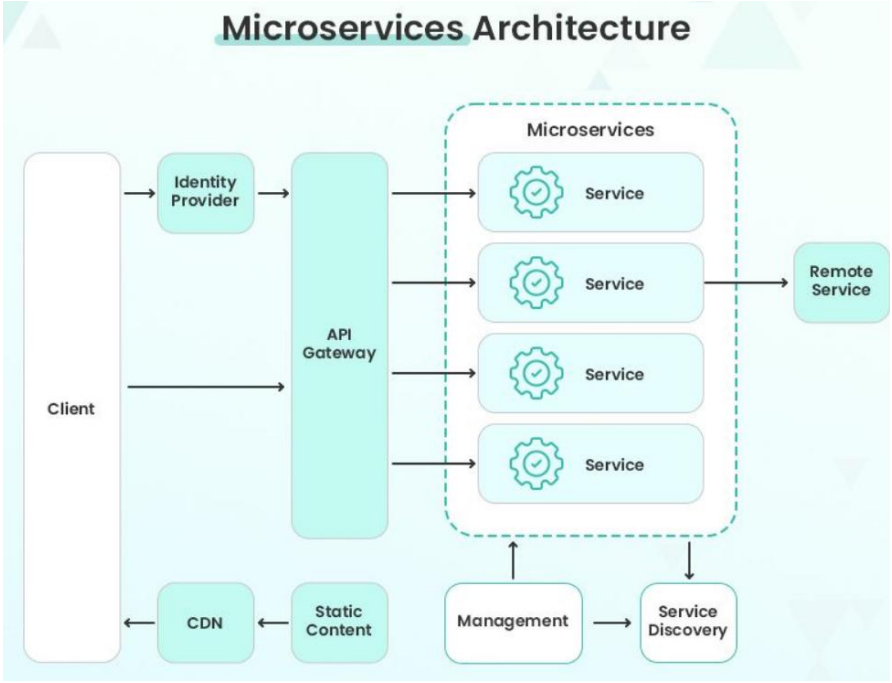

Рис. 2.3. Приклад мікросервісної архітектури [13]

Згідно з дослідженням Ньюмена [14], мікросервісний підхід дозволяє незалежне масштабування кожного компонента, значно підвищуючи відмовостійкість та гнучкість системи. Це робить його надзвичайно привабливим для великих платформ, що характеризуються непередбачуваним або значним навантаженням.

Однак для менших проектів або етапів прототипування ця архітектура є надмірно складною. Вона вимагає підтримки складної допоміжної інфраструктури, включаючи сервіс-дискавері, балансування навантаження та безпечний міжсервісний зв'язок. Таким чином, для волонтерської платформи модель мікросервісів може бути визначена як цільовий напрямок розвитку в майбутньому, але на етапі дисертації більш доцільна простіша архітектура.

2.2.5. Подійно-орієнтована архітектура

Архітектура, керована подіями (EDA), є ключовим сучасним підходом до побудови масштабованих та динамічних веб-систем. Основна ідея моделі полягає в тому, що будь-яка значна зміна стану, така як створення заявки, відкликання волонтера, зміна статусу або завершення обробки, генерує подію. Ця подія записується в центральному журналі або брокері повідомлень. Інші компоненти, підписані на ці події, отримують їх та реагують у режимі реального часу, без необхідності прямого синхронного виклику від відправника.

Цей підхід значно зменшує зв'язок між компонентами: замість прямої синхронної взаємодії через API, модулі обмінюються подіями через спільну інфраструктуру. Це підвищує масштабованість системи, оскільки кожен модуль обробляє дії незалежно. Крім того, EDA забезпечує кращу реактивність, дозволяючи системі миттєво реагувати на зміни та виконувати допоміжні дії, не затримуючи основні процеси.

У статті Крепса [15] зазначається, що підхід до зберігання та передачі подій на основі журналів є вирішальним елементом високонавантажених систем, що працюють у режимі реального часу. Автор зазначає, що обробка подій через центральний журнал або брокер повідомлень (наприклад, Apache Kafka) не лише спрощує масштабування, але й дозволяє відтворювати історії стани системи, що є критично важливим для аналітики та аудиту. Хоча цей підхід широко використовується на великих комерційних платформах, він однаково корисний і для волонтерських систем.

Для веб-платформи для волонтерської діяльності використання EDA дозволяє оптимізувати допоміжні служби, як системи сповіщень, ведення журналу та аналітичний моніторинг. Наприклад, створення запиту автоматично генерує подію, перехоплену службою сповіщень, для надсилання повідомлень волонтерам на основі критеріїв. Аналогічно, зміна статусу може ініціювати подію для оновлення адміністративної панелі або створення звітів.

Водночас, впровадження EDA вимагає додаткової інфраструктури (брокери повідомлень, системи моніторингу та відстеження подій). Це

ускладнює обслуговування та налагодження, особливо на ранніх етапах прототипування. Тому в рамках дисертації доцільно використовувати відповідний сфокусований підхід лише для другорядних, допоміжних завдань, які не є критичними для основної функціональності, таких як аудит та сповіщення.

Таким чином, EDA може служити ефективним доповненням до базової клієнт-серверної моделі. Це створює передумови для подальшого розвитку платформи, коли кількість користувачів та запитів зростає до такої міри, що синхронна взаємодія між компонентами стає вузьким місцем для швидкості реагування та масштабованості системи.

2.2.6. Порівняльний аналіз та обґрунтування вибору архітектурного рішення

Аналіз сучасних архітектурних підходів до розробки веб-систем показує, що вибір архітектури визначає не лише технічні характеристики платформи такі, як продуктивність, масштабованість та відмовостійкість, але й впливає на швидкість розробки, гнучкість та витрати на підтримку. Як зазначається в роботах Фаулера [9] та Річардса [12], архітектурна модель формує «вартість змін» протягом усього життєвого циклу програмного продукту.

Монолітна архітектура забезпечує швидкий старт та простоту на етапі прототипування, але втрачає ефективність у міру масштабування платформи через високий рівень зв'язку компонентів. Клієнт-серверна модель, описана Таненбаумом [10], пропонує більш керовану структуру та централізований контроль доступу, що робить її оптимальним рішенням для початкового етапу впровадження волонтерської платформи. Багатошарова архітектура, як пропонує Річардс [12], підвищує модульність системи та сприяє її модернізації, що особливо важливо для соціально орієнтованих сервісів з динамічно розширюваною функціональністю.

Мікросервісний підхід, виділений Ньюманом [14], є найбільш перспективним для масштабування великих систем; однак на ранніх стадіях

розробки він вимагає надмірно складної інфраструктури. Подійно-орієнтована архітектура, про яку писав Крепс [15], створює додаткові можливості для асинхронної взаємодії та високої продуктивності зі зростанням кількості користувачів та запитів, але її впровадження також слід відкласти, доки платформа не досягне зрілого рівня.

Тому для створення веб-платформи для організації волонтерської діяльності в рамках магістерської дисертації найраціональнішим вибором є клієнт-серверна архітектура з багаторівневим серверним компонентом. Такий підхід забезпечує збалансоване поєднання простоти впровадження, надійності та можливості поступового переходу до складніших моделей.

2.3. Вибір технологій для фронтенду

2.3.1 React як інструмент для створення інтерфейсу користувача

React — це фреймворк з відкритим кодом, створений Meta (раніше Facebook) у 2013 році. Його основним принципом є компонентний підхід, який дозволяє структурувати інтерфейс користувача як набір незалежних та повторно використовуваних блоків. Як зазначає С. Стефанов у своїй книзі [16], компонентна модель React суттєво спрощує керування станом застосунку та забезпечує зручне повторне використання коду у великих веб-системах.

Ключовою технічною перевагою React є використання віртуального DOM. Цей механізм оптимізує оновлення інтерфейсу: система визначає лише ті зміни, що відбулися, та модифікує їх, замість того, щоб повністю перезавантажувати сторінку. Це забезпечує високу продуктивність та плавну реакцію платформи, що є критично важливим під час відображення значних обсягів динамічних даних, таких як оновлення списків заявок або статусів у режимі реального часу.

Екосистема React надзвичайно широка та включає такі інструменти, як React Router для маршрутизації, React Query для оптимізації обробки даних, React Hook Form для ефективної валідації форм та бібліотеки компонентів (Material UI, Bootstrap) для швидкого створення привабливих та зручних

інтерфейсів. Значною перевагою для розробки є велика спільнота та значна кількість доступних навчальних ресурсів, що пришвидшує вирішення поширених проблем.

Явним недоліком React є відсутність суворих, уніфікованих архітектурних правил, що може призвести до різноманітності структур проектів. Однак, для цілей прототипування проектів це не є суттєвим обмеженням, оскільки дозволяє команді визначити пріоритети реалізації ключового функціоналу.

2.3.2 Vue.js як альтернатива для швидкої розробки веб-інтерфейсів

Vue.js є більш гнучким фреймворком, призначеним для швидкої розробки веб-інтерфейсів. Він підтримує синтаксис шаблонів, що робить код інтуїтивно зрозумілим для початківців. Як зазначає Е. Ю. Чен у своїй книзі [17], ключовою перевагою Vue є реактивна система даних, завдяки якій зміни стану автоматично відображаються в інтерфейсі користувача без додаткового коду.

Vue має офіційні бібліотеки для основних завдань: Vue Router для маршрутизації, Pinia для керування станом програми та численні готові компоненти для форм і таблиць. Це дозволяє швидко впроваджувати ключові елементи платформи, такі як реєстрація користувачів, створення та перегляд заявок, а також зміна статусу.

Переваги Vue.js включають низький поріг входження, компактну кодову базу та легкість інтеграції в існуючі проекти. Порівняно з React, він має менше готових промислових рішень та навчальних ресурсів, розроблених для великомасштабних систем. Однак для невеликого проекту Vue.js може бути ефективним вибором.

2.3.3 Angular як фреймворк для комплексних веб-застосунків

Angular — це комплексний фронтенд-фреймворк, який надає розробникам повну архітектуру та набір вбудованих інструментів. Він підтримує двостороннє зв'язування даних, включає механізм впровадження

залежностей (DI) та інтегрує інструменти реактивного програмування (RxJS). Завдяки вбудованим інструментам CLI (інтерфейс командного рядка), Angular спрощує створення та підтримку компонентів, модулів та сервісів, сприяючи оптимізованому та стандартизованому процесу розробки.

Цей фреймворк добре підходить для великомасштабних веб-систем, де підтримка суворої структури та співпраці в команді є важливими. Як зазначає Ш. Шама у своїй книзі [18], його переваги включають інтегровані інструменти для тестування, локалізації інтерфейсу, управління формами та модульності, що робить його придатним для довгострокових та корпоративних проектів.

Однак, Angular має вищий поріг входу та складніший процес налаштування, ніж легші фреймворки, такі як React або Vue.js. Тому для даного проєкту, де головною метою є швидка реалізація робочої моделі платформи, Angular може виявитися надмірно ресурсоємним та вимагати додаткового часу на навчання та налаштування.

2.3.4 Порівняльний аналіз і обґрунтування вибору фреймворку

Вибір фронтенд-фреймворку має вирішальне значення для забезпечення швидкої розробки та простоти подальшого обслуговування веб-платформи. Провідними кандидатами для розгляду є React, Vue.js та Angular.

React вирізняється своєю гнучкістю та високою продуктивністю завдяки механізму віртуального DOM та добре розвиненій екосистемі. Це дозволяє швидко впроваджувати ключові функції платформи, такі як форми реєстрації, динамічні списки запитів та панелі інструментів користувачів, а також спрощує подальше масштабування програми. Хоча його недоліком є відсутність суворої архітектурної структури, це не є критичним обмеженням для невеликого академічного проекту.

Vue.js характеризується низькою кривою навчання, інтуїтивно зрозумілим синтаксисом та базовим набором офіційних інструментів. Він ефективний для швидкого створення простих користувацьких інтерфейсів, але

йому бракує гнучкості та різноманітності готових бібліотек, необхідних для великомасштабних систем, порівняно з React.

Angular пропонує комплексну, готову архітектуру та набір вбудованих інструментів, що робить його ідеальним для великомасштабних корпоративних систем. Однак, його вища крива навчання та складність початкового налаштування роблять цей фреймворк менш придатним для швидкого прототипування в рамках дипломного проекту.

Таблиця 2.1

Порівняння фронтенд-фреймворків для веб-платформи волонтерської діяльності

Критерій	React	Vue.js	Angular
Простота навчання	Середня	Висока	Низька
Гнучкість і масштабованість	Висока	Середня	Висока
Екосистема та спільнота	Дуже розвинена	Добре розвинена	Розвинена
Придатність для швидкої розробки	Висока	Висока	Низька

Отже, для реалізації веб-платформи волонтерської діяльності оптимальним вибором є React, оскільки він забезпечує швидкий старт і подальшу масштабованість. Vue.js можна розглядати як альтернативу для простішого старту, тоді як Angular доцільний для великих і довготривалих проєктів.

2.4 Вибір технологій для бекенду

2.4.1 Node.js та фреймворк Express

Node.js — це серверне середовище виконання JavaScript, побудоване на високопродуктивному рушії V8 від Google. Ключовою технічною особливістю Node.js є його неблокуюча модель вводу/виводу (non-blocking I/O). Це дозволяє

йому ефективно обробляти тисячі одночасних запитів без необхідності додаткових ресурсів для створення нових потоків.

Як зазначає М. Канті у своїй роботі [19], асинхронна архітектура Node.js робить його оптимальним вибором для систем з високою частотою подій, де потрібна швидка обробка запитів та оновлення даних у режимі реального часу. Ця функція є критично важливою для волонтерської веб-платформи, оскільки велика кількість користувачів може одночасно створювати та переглядати запити.

Додатковою перевагою є розгалужена екосистема пакетів NPM та доступність фреймворку Express.js. Express.js дозволяє швидко налаштувати RESTful API та інтегрувати інструменти автентифікації, перевірки даних, ведення журналу та тестування. Легкість інтеграції з фронтенд-фреймворками, такими як React, робить Node.js практичним вибором для систем, які потребують швидкого розгортання.

2.4.2. Django

Django — один із найпопулярніших фреймворків Python, що пропонує повний набір вбудованих інструментів для створення веб-додатків. Він реалізує концепцію «Batteries included», що означає надання всіх необхідних компонентів без додаткового налаштування: систему автентифікації, ORM для взаємодії з базами даних, механізми безпеки, шаблонізатор та інтегровану панель адміністратора.

На основі досліджень А. Меліна [20] підтверджується, що завдяки чітко структурованому підходу Django дозволяє швидко запускати веб-сервіси. Це робить його особливо привабливим для корпоративних та довгострокових проектів, де пріоритетом є безпека та архітектурна передбачуваність. Однак у контексті простої волонтерської платформи його потужні та розширені функції можуть виявитися надмірними, вимагаючи більше часу на налаштування та обслуговування, ніж необхідно.

2.4.3. FastAPI

FastAPI — це сучасний фреймворк Python, який здобув популярність завдяки своїй швидкості, простоті використання та автоматичному генеруванню документації API на основі стандартів OpenAPI/Swagger. Він побудований на асинхронних інструментах Python, зокрема Starlette та Pydantic, що дозволяє розробляти високопродуктивні веб-сервіси з мінімальним кодом.

Т. Аграманес [21] зазначає, що FastAPI — чудовий вибір для малих та середніх веб-проектів, де швидка розробка та легкість інтеграції з іншими сервісами є ключовими. Його лаконічність та зрозумілий синтаксис роблять його доступним навіть для невеликих команд. Однак для масштабних проектів зі складною бізнес-логікою та потребою складного адміністрування FastAPI може вимагати інтеграції додаткових сторонніх рішень.

2.4.4. Порівняльний аналіз і обґрунтування вибору

При виборі технологій для бекенду, важливо враховувати продуктивність, масштабованість, легкість інтеграції з фронтендом, доступність бібліотек та рівень підтримки спільноти. Ці фактори безпосередньо впливають на швидкість розробки та постійну підтримку веб-платформи.

Node.js має асинхронну архітектуру та високу продуктивність з великою кількістю одночасних запитів, що робить його ефективним для динамічних систем. Django надає потужні вбудовані інструменти та високий рівень безпеки, але вимагає більше часу для налаштування та має вищий поріг входження. FastAPI поєднує простоту Python зі швидкістю, але для більших систем часто потрібні додаткові сторонні рішення.

Для платформи волонтерської діяльності, де важливими є швидке створення основного функціоналу, інтеграція з фронтендом і стабільність при зростанні навантаження, Node.js з фреймворком Express.js є оптимальним вибором.

Таблиця 2.2

Порівняння бекенд-технологій для веб-платформи волонтерської діяльності

Критерій	**Node.js (Express.js)**	**Django (Python)**	**FastAPI (Python)**
Парадигма роботи	Асинхронна, неблокуюча I/O	Синхронна (з опцією async у нових версіях)	Асинхронна, оптимізована для REST API
Продуктивність при високих навантаженнях	Висока завдяки подієвій моделі	Середня через синхронний підхід	Висока, близька до Node.js
Інтеграція з фронтендом	Проста завдяки JavaScript-екосистемі	Потребує адаптації та окремих API-шарів	Проста через REST і OpenAPI
Наявність готових інструментів	Багата екосистема NPM	Вбудовані інструменти (ORM, адмін-панель, безпека)	Менше вбудованих рішень, потребує сторонніх
Простота розробки	Легка для JS-розробників	Вища складність, але чітка структура	Легка для тих, хто знайомий із Python
Підходить для	Динамічних систем із великою кількістю запитів	Корпоративних і довгострокових проєктів	Легких і середніх за масштабом веб-сервісів

Порівняльний аналіз трьох розглянутих технологій показав, що Node.js у поєднанні з фреймворком Express.js найкраще відповідає вимогам веб-платформи для волонтерської роботи. Стек забезпечує високу продуктивність, легку фронтенд-інтеграцію та можливість швидкої розробки основного функціоналу. Django більше підходить для проектів зі складною корпоративною структурою та розгалуженою бізнес-логікою, тоді як FastAPI може бути ефективним для менших сервісів, але вимагає додаткових сторонніх рішень для реалізації розширених функцій. Тому вибір Node.js дозволяє досягти оптимального балансу між простотою впровадження, сучасною архітектурою та масштабованістю системи.

2.5. Бази даних та принципи їх побудови у волонтерських сервісах

2.5.1. Роль бази даних у веб-платформі

База даних є фундаментальним компонентом будь-якої веб-системи, оскільки вона забезпечує структуроване зберігання та швидкий доступ до інформації. Для веб-платформи, призначеної для волонтерства, база даних особливо важлива: вона накопичує дані користувачів, їхні запити на допомогу, їхній прогрес, а також історію взаємодії та адміністративні операції.

Стабільність та надійність системи значною мірою залежать від якості її проектування бази даних. Відповідно, вона повинна бути здатною не лише оперативно обслуговувати стандартні запити, але й відповідати високим вимогам безпеки та прозорості, які є ключовими для соціально орієнтованої сфери волонтерства.

2.5.2. Реляційні та нереляційні бази даних

У сучасній розробці веб-платформ використовуються дві основні парадигми для побудови систем зберігання даних: реляційна (SQL) та нереляційна (NoSQL).

Реляційні бази даних, такі як PostgreSQL або MySQL, організовують інформацію за допомогою взаємопов'язаних таблиць. Їхніми основними перевагами є транзакційність, цілісність та узгодженість даних, що є вирішальними в застосунках, де надійність та контроль змін є важливими. Реляційна модель забезпечує дотримання правил ACID (атомірність, узгодженість, ізоляція та довговічність), гарантуючи надійність записів навіть при великих навантаженнях.

Нереляційні бази даних, такі як MongoDB або Firebase, пропонують більшу гнучкість, зберігаючи дані у форматах документів або ключ-значення. Вони підходять для обробки динамічних та напівструктурованих даних, таких як коментарі або стрічки повідомлень у соціальних мережах. Однак у випадку волонтерської платформи структура даних є передбачуваною (користувачі,

запити, категорії, статуси), тому реляційний підхід є більш доцільним. Він сприяє реалізації зв'язків між сутностями та забезпечує простіше управління транзакціями.

Дослідження Кіма та співавторів [22] підтверджує, що у веб-застосунках, де важлива прозорість процесів і необхідно підтримувати історію змін, реляційні бази даних забезпечують кращу підтримку підзвітності та узгодженості даних.

2.5.3. Проєктування схеми бази даних

Правильне проектування структури бази даних має вирішальне значення для майбутньої підтримки та масштабованості системи. У контексті волонтерської платформи базова схема повинна містити сутності, що відображають ключові об'єкти домену:

- Users (Користувачі) – таблиця для зберігання облікових записів, що включає ім’я, електронну адресу, роль (заявник, волонтер, адміністратор), а також хеш пароля.
- Requests (Заявки) – таблиця для фіксації запитів на допомогу з такими атрибутами, як опис проблеми, категорія, місце розташування та поточний статус.
- Assignments (Призначення) – проміжна таблиця, що пов’язує заявки з волонтерами й дозволяє відстежувати виконання завдань.
- Categories (Категорії) – довідник типів допомоги (гуманітарна, медична, транспортна тощо), який дозволяє стандартизувати класифікацію заявок.
- AuditLog (Журнал дій) – таблиця для запису усіх змін, що відбуваються із заявками або користувачами, з метою забезпечення прозорості та можливості проведення перевірок.

Структура бази даних повинна бути нормалізована принаймні до третьої нормальної форми, що запобігає дублюванню даних та зменшує ризик помилок під час оновлень.

2.5.4. Забезпечення безпеки та захисту даних

Волонтерські платформи обробляють персональні дані користувачів і часто містять конфіденційну інформацію про місцезнаходження та потреби заявників. Тому захист даних є ключовою вимогою до архітектури та функціонування системи. Недостатній рівень безпеки може призвести до витоку даних, втрати довіри користувачів і навіть становити загрозу їхньому життю та здоров'ю, особливо у воєнний час.

Одним з ключових аспектів є багаторівнева автентифікація та впровадження моделі доступу на основі ролей. Користувачі різних категорій такі, як заявники, волонтери та адміністратори, повинні мати доступ лише до функціональності, необхідної для виконання їхніх завдань. Такий підхід мінімізує ризик несанкціонованих дій та підвищує загальну безпеку системи.

Важливою складовою безпеки системи є використання middleware-рівня, який забезпечує централізовану перевірку всіх вхідних запитів. На цьому рівні виконується валідація JWT-токена, контроль ролей користувачів, фільтрація та базова валідація даних, а також уніфікована обробка помилок без розкриття внутрішніх механізмів сервера. Такий підхід створює єдиний, безпечний механізм доступу до функціональності платформи та мінімізує ризик несанкціонованих операцій.

Ще одним важливим заходом є хешування паролів за допомогою сильних алгоритмів, таких як bcrypt або Argon2. Це гарантує, що навіть у разі витоку бази даних зловмисники не зможуть відновити паролі користувачів.

Механізми реєстрації та аудиту дій користувачів та адміністраторів не менш важливі. Система повинна зберігати інформацію про всі зміни в запитах, статусах та налаштуваннях, щоб забезпечити прозорість роботи та можливість повторного відтворення у разі виникнення інцидентів безпеки або суперечок.

Додатковим елементом є регулярне резервне копіювання даних та контроль доступу на рівні бази даних. Це захищає інформацію від втрати через технічні збої або кібератак, а також запобігає несанкціонованому редагуванню.

Галфонд та Вієгас у своїй праці [23] наголошують на тому, що аудит та прозорість обробки запитів у веб-системах є ключовими факторами, які не лише підвищують довіру користувачів до платформи, але й забезпечують дотримання сучасних вимог безпеки та нормативних стандартів.

2.5.5. Вибір бази даних для волонтерської платформи

Зважаючи на вимоги магістерської роботи та специфіку домену, оптимальним вибором є реляційна система PostgreSQL. Вона забезпечує транзакційність, підтримує складні зв’язки між таблицями, має потужний механізм індексів і тригерів, а також добре масштабується при збільшенні обсягів даних.

Додатковою перевагою PostgreSQL є активна спільнота та велика кількість бібліотек інтеграції з сучасними веб-фреймворками, що полегшує розробку й подальше обслуговування системи.

Використання PostgreSQL дозволяє реалізувати ключові вимоги платформи: підтримувати коректний запис заявок, швидко виконувати пошук та фільтрацію, а також забезпечувати прозорість усіх змін у системі.

Висновок до розділу 2

У другому розділі було проведено аналіз архітектурних рішень, технологій та інструментів, необхідних для створення веб-платформи волонтерської допомоги. Дослідження дозволило визначити оптимальну структуру системи та її технічні вимоги.

Порівняння клієнт-серверної, багатошарової та мікросервісної архітектур показало, що найраціональнішим вибором для початкової версії є класична клієнт-серверна модель з REST API. Вона забезпечує простоту впровадження, чіткий розподіл логіки та можливість масштабування.

Порівняльний аналіз показав, що React є оптимальним вибором завдяки його компонентній структурі, широкій екосистемі та простоті розробки. Для бекенду обрано Node.js від Express.js, що забезпечує високу продуктивність та легкість побудови REST-сервісів. В якості основної системи управління базами даних передбачається використання PostgreSQL, а для локального тестування – SQLite.

На основі аналізу вимог сформовано базові функції для всіх ролей користувачів. Заявники створюють заявки, волонтери можуть їх переглядати та приймати, адміністратори модерують та контролюють роботу системи. Ця модель забезпечує прозору та логічно структуровану взаємодію між учасниками.

Особлива увага приділяється безпеці: використання JWT-автентифікації, хешування паролів bcrypt, а також впровадження middleware-рівня для перевірки прав доступу. Це забезпечує захист персональних даних та стабільність платформи.

Отже, у другому розділі було закладено технічну основу для подальшого проектування та впровадження платформи. Було обґрунтовано архітектурний підхід, визначено технологічний стек та базу даних, а також сформульовано ключові функціональні вимоги до системи.

РОЗДІЛ 3
ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБ-ПЛАТФОРМИ

3.1. Проєктування архітектури та компонентів системи

3.1.1. Загальна архітектура системи

Волонтерська платформа реалізована за архітектурою клієнт–сервер, що є стандартним підходом для сучасних веб-застосунків. Система складається з трьох основних рівнів:

- Рівень представлення (Presentation Layer) — відповідає за взаємодію користувача із системою через веб-інтерфейс;
- Рівень бізнес-логіки (Business Logic Layer) — виконує обробку запитів, авторизацію, перевірку даних і формування відповідей;
- Рівень даних (Data Layer) — забезпечує зберігання інформації про користувачів, заявки, відгуки волонтерів та журнал дій.

Обмін даними між клієнтом і сервером здійснюється за протоколом HTTP/HTTPS, а формат передачі даних — JSON, що забезпечує простоту інтеграції, узгодженість структур і незалежність від технологій інтерфейсу. Архітектура побудована на принципах REST API, що дозволяє масштабувати систему, підключати додаткові сервіси або мобільні застосунки без зміни базової логіки.

На рисунку 3.1 подано схему архітектури веб-платформи. Вона демонструє потік даних між користувачами, клієнтською частиною, сервером, базою даних та модулем аудиту.

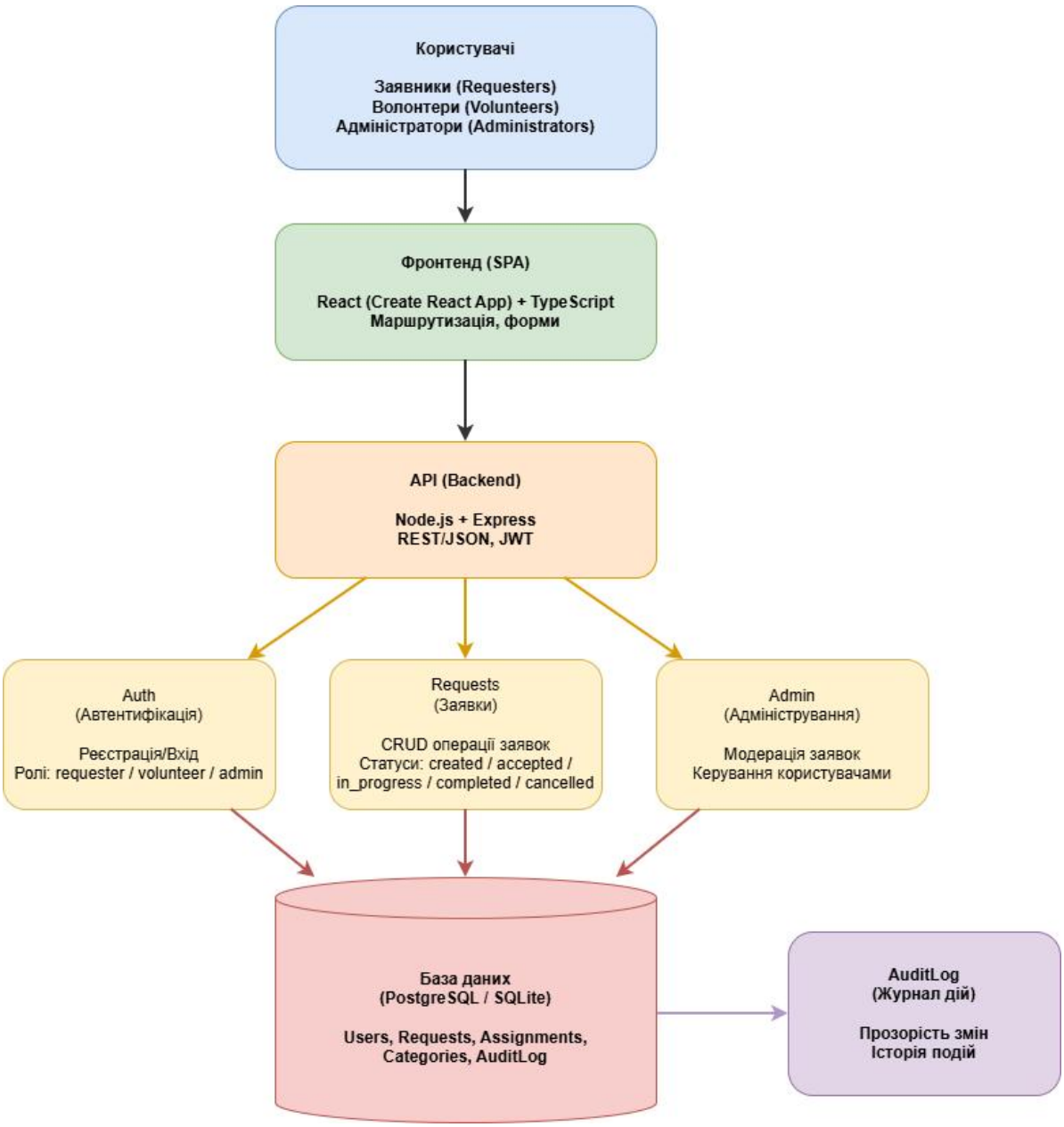

Рис. 3.1. Загальна архітектура веб-платформи для волонтерської діяльності

Згідно з рисунком 3.1, користувачі (заявники, волонтери, адміністратори) взаємодіють із системою через веб-інтерфейс, який надсилає запити до серверної частини. Сервер обробляє дані, виконує авторизацію та звертається до бази даних для збереження або отримання інформації. Усі зміни фіксуються в журналі дій (AuditLog), що забезпечує прозорість і контроль процесів.

3.1.2. Основні компоненти системи

Архітектура веб-платформи складається з кількох взаємозалежних компонентів, кожен з яких виконує чітко визначену функцію в межах загальної структури. Усі модулі взаємодіють через єдиний прикладний програмний інтерфейс (API), що забезпечує узгоджений обмін даними та сприяє спрощенню підтримки коду. Ключовими структурними елементами системи є модулі автентифікації користувачів, обробки заявок, адміністрування та журналювання дій.

Модуль автентифікації (Auth) відповідає за реєстрацію нових користувачів, безпечний вхід у систему та верифікацію прав доступу. Він реалізує функції перевірки облікових даних, видачі токенів авторизації, а також контролює доступність маршрутів для кожної ролі. Це гарантує захищений вхід і коректне розмежування прав між різними категоріями користувачів.

Модуль обробки заявок (Requests) реалізує повний цикл операцій із заявками, включаючи створення, перегляд та зміну статусів. Кожна заявка містить обов'язкову інформацію про потребу, категорію допомоги, місце розташування та контактні дані заявника. Цей модуль є носієм основної бізнес-логіки системи, оскільки через нього здійснюється більшість ключових користувацьких взаємодій.

Модуль адміністрування (Admin) призначений для централізованого керування даними та налаштуваннями системи. Адміністратор має можливість переглядати всі заявки, змінювати їхні статуси, видаляти заявки, а також управляти обліковими записами користувачів. Його функціонал спрямований на підтримку порядку, забезпечення актуальності інформації та запобігання дублюванню чи зловживанням.

Окрему, але критично важливу роль виконує модуль журналювання дій (AuditLog). Він фіксує всі значущі події в системі: авторизації, створення, редагування чи видалення заявок, а також зміни ролей і статусів. Цей компонент забезпечує прозорість функціонування платформи, дозволяючи здійснювати аудит і повне відтворення історії змін.

3.1.3. Користувацькі ролі та взаємодія

У межах розробленої системи передбачено три основні ролі користувачів: заявник, волонтер та адміністратор. Кожна з цих ролей виконує чітко визначені функції та наділена відповідними правами доступу.

Заявник (Requester) — це користувач, який створює запити про необхідність допомоги. Він заповнює форму заявки, обирає категорію допомоги, район Києва, рівень терміновості та спосіб зв’язку. Після створення заявка автоматично фіксується у базі даних зі статусом «створено».

Волонтер (Volunteer) має можливість переглядати всі доступні заявки, фільтрувати їх за статусом, терміновістю та районами, а також відгукуватися на відповідні. Після натискання кнопки «Відгукнутись» у системі створюється запис призначення зі статусом «Очікує підтвердження». Контактні дані заявника волонтер отримує лише після підтвердження відгуку заявником, що забезпечує приватність та безпеку.

Адміністратор (Admin) здійснює нагляд за роботою платформи, має доступ до всіх заявок, користувачів і записів аудиту, може редагувати та видаляти дані. Це дозволяє підтримувати стабільність та порядок у системі.

Для наочності роботи системи нижче наведено блок-схеми алгоритмів взаємодії основних ролей — заявника та волонтера.

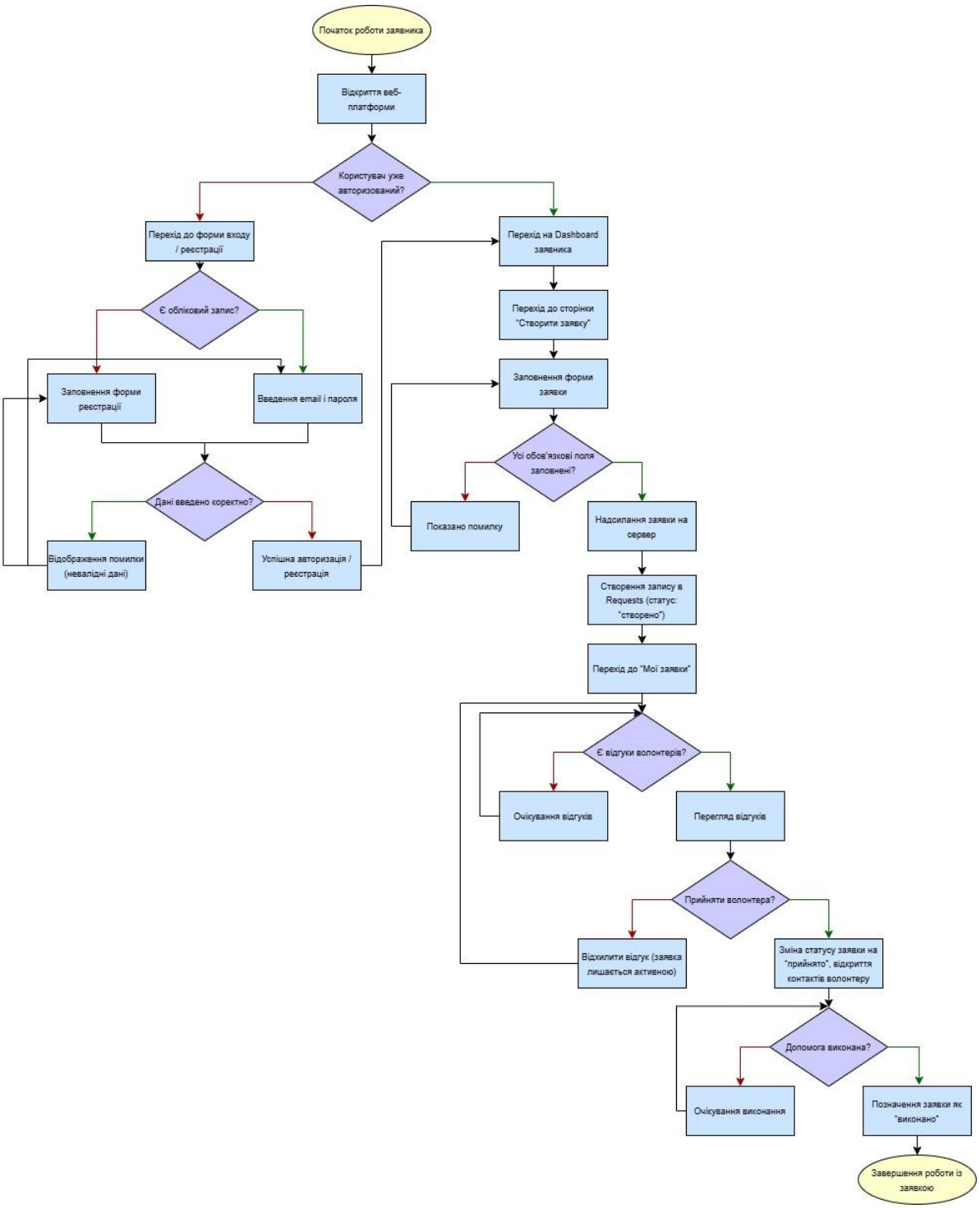

Рис. 3.2. Блок-схема алгоритму роботи заявника у веб-платформі

Представлена блок-схема на рисунку 3.2 демонструє повний алгоритм роботи заявника від моменту входу в систему до створення заявки та взаємодії з відгуками волонтерів. Схема відображає всі ключові етапи: авторизацію, валідацію введених даних, формування заявки, обробку відгуків та фінальне завершення виконання. Така послідовність забезпечує контрольованість

процесу, прозорість дій заявника та безпечне надання доступу до персональних контактних даних лише після підтвердження участі волонтера.

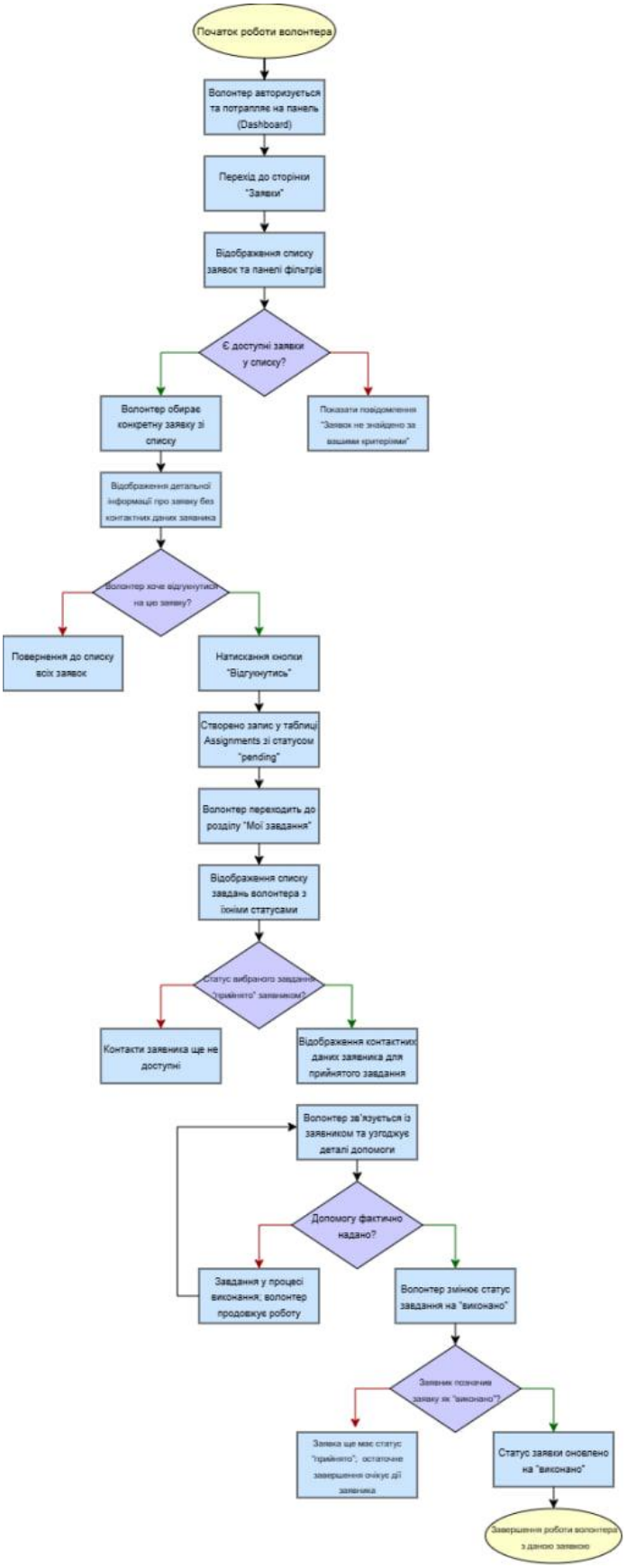

Рис. 3.3. Блок-схема алгоритму роботи волонтера у веб-платформі

Блок-схема алгоритму волонтера відображає логіку його роботи у системі: перегляд доступних заявок, використання фільтрів, подання відгуку, отримання статусу підтвердження та завершення завдання. Особливістю цього процесу є поетапний доступ до інформації — волонтер бачить лише публічні дані, а контакти відкриваються тільки після підтвердження заявником. Це забезпечує конфіденційність, формує безпечну модель взаємодії та гарантує роботу волонтера лише з актуальними й підтвердженими завданнями.

3.1.4. Основні об'єкти бази даних

Архітектура бази даних платформи побудована за реляційною моделлю та включає кілька ключових об'єктів.

Основною таблицею є Users, яка містить дані про користувачів, зокрема ім’я, електронну адресу, пароль (у зашифрованому вигляді), роль і статус активності. Це дозволяє чітко визначати права доступу кожного користувача та організовувати безпечну автентифікацію.

Таблиця Requests призначена для зберігання інформації про заявки. У ній фіксується опис потреби, категорія допомоги, район Києва, рівень терміновості, контактні дані та статус виконання. Ця таблиця є центральною для всієї системи, оскільки пов’язана з іншими об'єктами.

Таблиця Assignments забезпечує зв’язок між заявкою та волонтером. У ній фіксується, який волонтер відгукнувся на заявку, дата відгуку, і який поточний статус взаємодії (очікує підтвердження, прийнято, відхилено). Такий підхід дозволяє детально відстежувати процес виконання кожної заявки.

Таблиця Categories містить перелік типів допомоги — медичну, гуманітарну, транспортну, інформаційну тощо. Це дає змогу стандартизувати класифікацію запитів і полегшує пошук потрібної інформації.

Окремою частиною структури є таблиця AuditLog, у якій зберігаються всі дії користувачів: створення та видалення записів, зміна статусів заявок, зміна ролей користувачів та реєстрація в системі. Наявність цього журналу дозволяє відновити історію подій та забезпечує контроль достовірності операцій.

3.1.5. Архітектурні принципи

Розроблена архітектура системи ґрунтується на принципах розділення відповідальностей, модульності, безпеки та прозорості. Кожен рівень виконує власну функцію: інтерфейс відповідає за взаємодію користувача із системою, серверна логіка — за обробку запитів і виконання правил бізнес-процесів, а база даних — за збереження та цілісність інформації. Такий поділ забезпечує стабільність і полегшує підтримку коду.

Модульність реалізується через ізольовані компоненти з чітко визначеними інтерфейсами. Це дозволяє вносити зміни в один із модулів без впливу на інші частини системи, що значно спрощує налагодження та тестування.

Безпека є невід'ємною частиною архітектури. Усі дії користувачів проходять перевірку авторизації, а дані підлягають валідації перед збереженням. Паролі зберігаються у зашифрованому вигляді, що унеможливлює їхню компрометацію.

Прозорість забезпечується за рахунок ведення журналу дій, який фіксує всі зміни в системі. Це підвищує довіру до платформи та дозволяє виявляти можливі помилки або зловживання.

3.1.6. Взаємодія компонентів

Усі компоненти системи взаємодіють відповідно до чітко визначеного сценарію, який описує життєвий цикл заявки від моменту її створення до завершення. Спершу заявник формує опис потреби, обирає категорію допомоги та публікує заявку. Після цього вона стає доступною для перегляду волонтерами.

Коли волонтер знаходить заявку, що відповідає його можливостям, він ініціює дію «Відгукнутись на заявку». У цей момент система створює запис у таблиці Assignments зі статусом «Очікує підтвердження». Заявник може переглянути відгук у розділі «Мої заявки» і або підтвердити, або відхилити

пропозицію волонтера. Після підтвердження волонтер отримує доступ до контактних даних заявника.

Після надання допомоги волонтер позначає своє завдання як виконане, а заявник змінює статус заявки на «Виконано». Паралельно адміністратор може контролювати кількість активних заявок, відстежувати статистику виконання та переглядати історію змін. Потоки даних, що передаються між компонентами, охоплюють створення нових заявок, оновлення статусів, формування звітів і запис усіх подій у журнал аудиту.

Завдяки такій логіці взаємодії система забезпечує послідовність процесів, контроль виконання, безпеку користувачів та прозорість усіх операцій.

3.2. Реалізація серверної частини (бекенду)

3.2.1. Технологічний стек та архітектура

Серверна частина волонтерської платформи реалізована на основі середовища Node.js, яке забезпечує неблокуючу подієву модель, що є оптимальною для створення високопродуктивних веб-сервісів з великою кількістю одночасних з’єднань. У якості основного фреймворку використано Express.js, який надає зручні інструменти для побудови REST API та гнучке керування маршрутизацією запитів.

Архітектура веб-платформи реалізована за принципом клієнт-серверної взаємодії, що забезпечує чітке розділення між інтерфейсом користувача та серверною логікою. Клієнтська частина (frontend) виконує роль рівня представлення, відповідає за відображення інформації, взаємодію з користувачем і надсилання запитів до сервера через REST API. Серверна частина (backend) обробляє ці запити, виконує бізнес-логіку, перевіряє права доступу, звертається до бази даних і формує відповіді у форматі JSON. Такий підхід спрощує підтримку та масштабування системи, дозволяє розвивати клієнт і сервер незалежно один від одного та забезпечує можливість інтеграції з мобільними застосунками або сторонніми сервісами в майбутньому.

До складу основних програмних залежностей входять такі бібліотеки та пакети:

- Express.js (v4.19.0) — веб-фреймворк для організації HTTP-запитів і створення REST API;
- SQLite3 (v5.1.6) — реляційна база даних для середовища розробки;
- PostgreSQL (v8.11.3) — СУБД для промислового розгортання;
- JWT (v9.0.2) — реалізація токенів автентифікації;
- bcryptjs (v2.4.3) — алгоритм хешування паролів користувачів;
- express-validator (v7.0.1) — інструмент для перевірки коректності введених даних;
- CORS (v2.8.5) — пакет для контролю крос-доменного доступу.

Використання цього стеку дозволяє реалізувати сучасний, безпечний і легкий у підтримці сервер, здатний обробляти значну кількість одночасних запитів без втрати стабільності.

3.2.2. Структура проєкту

Серверна частина платформи побудована за модульним принципом, що забезпечує логічне розділення коду за функціональністю. Кожен каталог виконує чітко визначену роль:

```
backend/
├── config/          # Конфігурація бази даних
├── middleware/      # Проміжні обробники для авторизації
├── routes/          # REST API маршрути
├── utils/           # Допоміжні утиліти
├── tests/           # Тестові сценарії
├── schema.sql       # Схема PostgreSQL
├── schema-sqlite.sql # Схема SQLite
└── server.js        # Точка входу застосунку
```

Рис. 3.4. Структура директорій бекенд-частини проєкту

Файл server.js є центральною точкою запуску системи: він ініціалізує сервер, підключає middleware, маршрути, обробку помилок та інші глобальні налаштування.

У каталозі config/ зберігаються налаштування з'єднання з базою даних, тоді як middleware/ містить проміжні обробники, що виконують перевірку прав користувачів і токенів. Каталог routes/ включає всі REST API маршрути, згруповані за логічними модулями (автентифікація, заявки, завдання, категорії, адміністративні дії).

Завдяки такій структурі проєкт залишається легко масштабованим і придатним до розширення, а додавання нових компонентів не впливає на стабільність існуючих модулів.

3.2.3. Конфігурація бази даних

У системі реалізовано адаптивну підтримку двох баз даних — SQLite для розробки та PostgreSQL для production-середовища. Це дозволяє одночасно забезпечити простоту локального тестування і стабільність промислового розгортання.

SQLite використовується на етапі розробки через відсутність необхідності в окремому сервері та можливість швидкого налаштування схеми бази. Вона функціонує як файлова база даних, що зберігається безпосередньо у проєктному каталозі.

PostgreSQL, своєю чергою, є високопродуктивною СУБД, що підтримує транзакції, ACID-принципи, індекси та розширені типи даних.

Конфігурація бази даних реалізована так, що система автоматично визначає потрібне середовище роботи, спираючись на змінні середовища (process.env.NODE_ENV). Це забезпечує однакову логіку роботи застосунку як у режимі розробки, так і після розгортання.

3.2.4. Система автентифікації та авторизації

Автентифікація користувачів у системі реалізована за допомогою JSON Web Tokens (JWT). Після успішного входу користувач отримує токен, який зберігає інформацію про його роль і термін дії. Кожен запит до захищених ресурсів перевіряється middleware-функцією authenticateToken, яка розшифровує токен і верифікує його за допомогою секретного ключа, що зберігається у змінній середовища JWT_SECRET.

У рамках платформи реалізовано рольову модель доступу (RBAC), яка передбачає три рівні привілеїв:

- Requester — користувач, який створює заявки на допомогу;
- Volunteer — користувач, який відгукується на створені заявки;
- Admin — адміністратор, що має повний контроль над системою.

Розмежування прав доступу забезпечує безпеку даних і виключає можливість несанкціонованих дій. Для реалізації перевірок доступу використовуються middleware-функції requireAdmin, requireVolunteer, requireRequester, що дозволяє контролювати права на рівні окремих маршрутів.

3.2.5. API маршрути та ендпоінти

Сервер надає набір REST API, згрупованих за функціональними модулями:

- /api/auth — реєстрація користувачів, вхід до системи, отримання профілю;
- /api/requests — створення, перегляд та видалення заявок;
- /api/assignments — управління завданнями волонтерів і статусами їх виконання;
- /api/categories — отримання списку категорій допомоги;
- /api/admin — адміністративна панель: керування користувачами, моніторинг системи та журнал аудиту.

Кожен ендпоінт має чітко визначений метод (GET, POST, PUT, PATCH, DELETE) та відповідну перевірку доступу. Усі відповіді від сервера

надсилаються у форматі JSON, що спрощує інтеграцію з клієнтськими додатками.

3.2.6. Надійність, безпека та архітектурні принципи серверної частини

Серверний компонент платформи реалізує комплексний підхід до забезпечення надійності та безпеки системи, що охоплює перевірку даних, обробку помилок, механізми захисту, оптимізацію продуктивності та застосування архітектурних патернів. Достовірність введених користувачем даних перевіряється за допомогою бібліотеки express-validator, яка дозволяє визначати правила перевірки для кожного маршруту. Під час створення облікового запису перевіряється унікальність електронної пошти та забезпечується перевірка мінімальної довжини пароля. Під час створення заявок перевіряються ім'я, опис, категорія, рівень терміновості та контактна інформація. У разі розбіжностей система повертає структуроване повідомлення про помилку, що дає змогу швидко її виправити.

Для підтримки стабільності реалізовано централізовану обробку помилок. Усі винятки перехоплюються сервером та повертаються у стандартизованому форматі, що запобігає розкриттю внутрішніх технічних деталей. Крім того, всі суттєві дії користувача (створення та редагування заявок, зміна статусу та адміністративні операції) записуються в AuditLog. Це підвищує прозорість роботи системи та дозволяє проводити детальний аналіз у разі виникнення інцидентів безпеки.

Проєкт використовує багаторівневі механізми безпеки. Паролі користувачів зберігаються хешованими за допомогою алгоритму bcryptjs, що забезпечує їхню стійкість до компрометації. Усі приватні запити API дозволені лише з дійсним токеном JWT, який перевіряє автентичність користувача. Налаштування CORS регулюють дозволені джерела запитів до сервера, а фільтрація та перевірка введених даних захищає від атак SQL-ін'єкцій та запобігає атакам міжсайтового скриптингу. Сукупність цих заходів гарантує

збереження персональних даних та стійкість системи до поширених типів кіберзагроз.

Оптимізація продуктивності включає використання індексів бази даних для пришвидшення фільтрації та пошуку заявок за статусом, роллю та іншими параметрами. Для роботи з великими наборами даних реалізовано пагінацію, що зменшує навантаження на сервер та пришвидшує обробку запитів. Статичні довідники, зокрема категорії допомоги, отримуються безпосередньо з бази даних, що гарантує актуальність інформації.

Під час розробки було використано низку архітектурних принципів для забезпечення гнучкості та розширюваності системи. Модульна структура спрощує обслуговування та тестування окремих компонентів. Middleware-підхід дозволяє послідовно обробляти HTTP-запити на рівнях автентифікації, авторизації, обробки помилок та валідації, забезпечуючи чіткий розподіл відповідальності. Використання патерну Adapter у модулі бази даних створює єдиний інтерфейс для взаємодії як із PostgreSQL, так і зі SQLite, що спрощує налаштування середовищ розробки та production.

3.2.7. Модулі та функції бекенду

Серверна частина складається з низки модулів, кожен із яких виконує певну функцію.

Модуль config/database.js відповідає за підключення до бази даних і вибір драйвера залежно від середовища.

Модуль middleware/auth.js здійснює перевірку JWT-токенів і контроль доступу за ролями.

Модуль utils/validation.js містить набір функцій для перевірки даних користувачів і заявок.

У каталозі routes/ розташовані модулі з визначенням усіх маршрутів API: автентифікація, заявки, завдання волонтерів, категорії допомоги та адміністративний інтерфейс.

Файл server.js виконує роль точки входу — він ініціалізує сервер Express, підключає middleware, маршрути, обробники помилок і запускає веб-сервер на вказаному порту.

Модель даних включає сутності User, Request, Assignment, Category, AuditLog, між якими встановлені відповідні зв’язки: користувач створює заявки, волонтер відгукується на них, адміністратор відстежує дії через журнал аудиту.

Діаграма класів, наведена на рисунку 3.5, відображає логічну структуру серверної частини системи, зв’язки між модулями, моделями та користувацькими ролями.

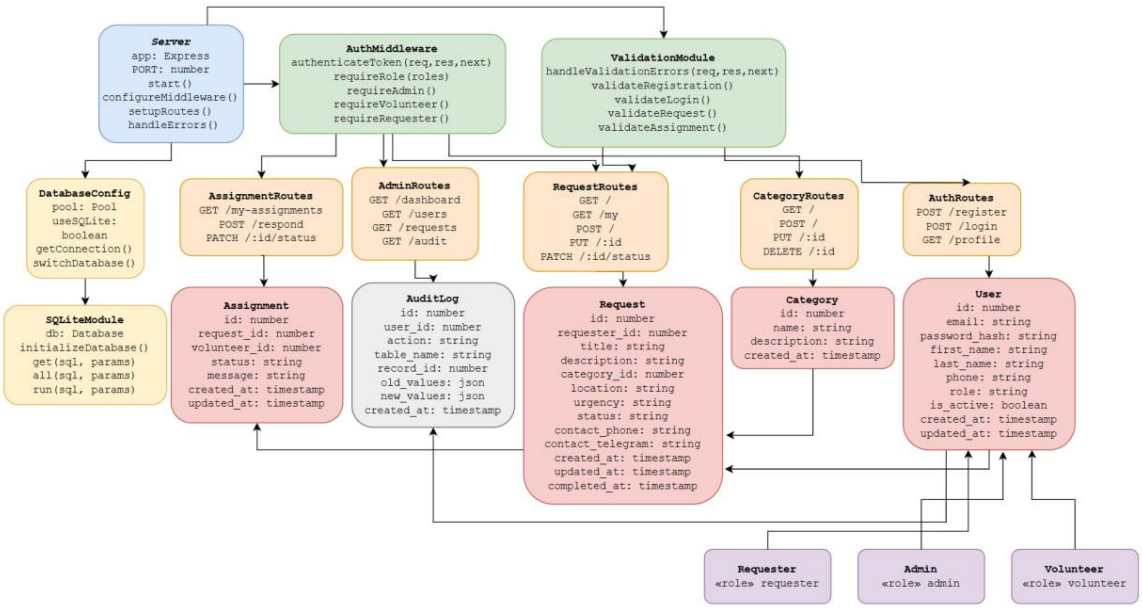

Рис. 3.5. UML-діаграма класів серверної частини системи

Діаграма демонструє модульну побудову системи, розділення рівнів відповідальності та зв’язки між основними компонентами: сервером, middleware-модулями, маршрутами API та моделями бази даних.

У центрі архітектури розташований клас Server, який відповідає за ініціалізацію програми на базі Express.js, налаштування проміжних обробників (middleware), підключення маршрутів, обробку помилок та запуск HTTP-сервера.

Він напряму пов’язаний зі:

- DatabaseConfig — модуль управління базою даних, що забезпечує прозоре перемикання між SQLite (розробка) та PostgreSQL (production) відповідно до налаштувань середовища.
- AuthMiddleware та ValidationModule представляють собою основні проміжні компоненти, що відповідають за авторизацію користувачів та перевірку валідності вхідних параметрів.

Модуль DatabaseConfig забезпечує підключення до відповідної бази даних через перевірку змінних середовища. У режимі розробки використовується SQLite, тоді як у продакшн-середовищі — PostgreSQL. Зв'язок DatabaseConfig → SQLiteModule показує, що цей конфігураційний клас викликає функції ініціалізації та роботи з файлом бази даних, реалізовані в модулі SQLiteModule.

Middleware представлений двома основними класами:

- AuthMiddleware реалізує механізми автентифікації та авторизації, зокрема перевірку токенів JWT і рольову модель доступу через Higher-Order Functions (requireAdmin, requireVolunteer, requireRequester).
- ValidationModule відповідає за попередню перевірку даних користувачів, заявок і завдань перед тим, як вони потрапляють до маршрутизаторів API.

Ці два модулі пов'язані стрілками зі всіма маршрутами (Routes), оскільки вони інтегруються у кожен запит.

Рівень маршрутизації складається з п'яти основних блоків, кожен із яких відповідає за певну частину бізнес-логіки:

- AuthRoutes — маршрути автентифікації: реєстрація, вхід, перегляд профілю. Працює з моделлю User.
- RequestRoutes — CRUD-операції над заявками користувачів, зв'язаний із моделлю Request.
- AssignmentRoutes — маршрути, що дозволяють волонтерам відгукуватися на заявки; працюють із таблицею Assignment.

- CategoryRoutes — маршрути для управління довідником категорій допомоги (Category), доступні лише адміністраторам.
- AdminRoutes — панель адміністратора з можливістю перегляду користувачів, заявок та журналу дій (AuditLog).

Стрілки між AuthMiddleware → Routes і ValidationModule → Routes позначають, що перед виконанням кожного запиту спершу проходить авторизація та валідація.

Моделі (класи, що відповідають таблицям бази даних) розташовані в нижній частині діаграми:

- User — зберігає дані користувачів, включно з email, паролем (у хешованому вигляді), ім’ям, роллю.
- Request — описує заявки на допомогу з посиланням на автора (заявника), категорію, локацію, терміновість і статус виконання.
- Assignment — реалізує зв’язок між волонтером і заявкою; містить статус («Очікує підтвердження», «Прийнято», «Відхилено»).
- Category — довідник категорій допомоги, який використовується для класифікації заявок.
- AuditLog — журнал аудиту, що фіксує всі важливі зміни в системі: створення заявок, оновлення статусів, адміністративні дії.

У нижній частині діаграми подано три похідні ролі, які наслідують властивості класу User:

- Requester — користувач, що створює заявки;
- Volunteer — користувач, який переглядає заявки та може відгукуватися на них;
- Admin — адміністратор, який модерує систему, користувачів і заявки.

Таким чином, діаграма класів наочно ілюструє структурну організацію системи, яка базується на чіткому розмежуванні функціональних рівнів: ядра серверної логіки, проміжних обробників, маршрутизації API та моделей даних. Внаслідок інтеграції принципів модульності, розділення відповідальностей і багаторівневої безпеки, розроблена архітектура відповідає вимогам гнучкості,

розширюваності та створює міцну основу для перспективного масштабування платформи.

3.3. Реалізація клієнтської частини (фронтенду)

3.3.1. Технологічний стек та архітектура

Клієнтська частина волонтерської платформи побудована за допомогою React 19 у поєднанні з TypeScript. Цей вибір забезпечує сучасний, типізований підхід до розробки веб-інтерфейсу. TypeScript значно зменшує кількість помилок компіляції, полегшуючи налагодження та підвищуючи загальну надійність коду. Додаток побудовано за принципом односторінкового додатку (SPA), де контент оновлюється динамічно без перезавантаження сторінки, що робить взаємодію з платформою швидкою та зручною.

Архітектура базується на компонентному підході: інтерфейс складається з незалежних блоків, кожен з яких виконує свою певну функцію (наприклад, форма входу, перелік/список заявок, панель адміністратора). Це забезпечує повторне використання коду, ізоляцію логіки та спрощує тестування та оновлення системи. Компоненти взаємодіють через властивості (props) та глобальний контекст (Context API), який використовується для централізованого зберігання інформації про користувача.

React-хуки (useState, useEffect, useContext) використовуються для керування станом компонентів, що дозволяє реалізувати логіку без використання компонентів класу. Хуки роблять код більш компактним та зрозумілим, а керування станом більш ефективним. Глобальний стан автентифікації реалізовано через Context API, який забезпечує доступ до даних користувача в будь-якому компоненті та підтримує автоматичне відновлення сеансу після оновлення сторінки.

Маршрутизація підтримується бібліотекою React Router DOM (v7.9.3), що дозволяє переходи між сторінками без повного оновлення сторінки. Для взаємодії з сервером використовується Axios (v1.12.2). Цей HTTP-клієнт

забезпечує централізоване керування запитами, автоматичне додавання токенів авторизації до заголовків та обробку помилок. Це централізоване керування запитами спрощує розширення програми та підвищує безпеку.

Для обробки дати використовується Moment.js (v2.30.1), що забезпечує локалізацію часу українською мовою, зручне форматування дати та відображення позначок часу в запитах. Тестування компонентів виконується за допомогою бібліотеки React Testing Library, яка дозволяє перевіряти правильність рендерингу, поведінку елементів інтерфейсу та обробку дій користувача.

З погляду принципів проєктування, клієнтська частина системи дотримується моделі розділення відповідальностей та реалізує архітектурний шаблон, що містить ключові елементи Model-View-Controller (MVC). У цій структурі:

- Model представлена даними, що надходять із REST API;
- рівень View (інтерфейсу) формується React-компонентами;
- Controller реалізований через глобальні контексти та механізми маршрутизації, які забезпечують координацію логіки взаємодії між користувацьким інтерфейсом та сервером.

Такий підхід забезпечує модульність, легку масштабованість та високу продуктивність клієнтської частини, що важливо для систем, призначених для інтерактивної роботи зі значною кількістю користувачів.

3.3.2. Структура проєкту

Структура клієнтської частини платформи побудована таким чином, щоб забезпечити логічне розділення функціональності, зручність навігації у коді та можливість масштабування. Усі файли згруповано за їхнім призначенням: інтерфейсні компоненти, глобальний контекст, сервіси для роботи з API та початкові файли застосунку. Такий підхід відповідає принципам Separation of Concerns (розділення відповідальностей) і покращує підтримуваність коду на подальших етапах розвитку системи.

```
frontend/src/
├── components/              # Компоненти інтерфейсу користувача
│   ├── Dashboard.tsx        # Головна панель з вибором дій за роллю
│   ├── Login.tsx            # Форма авторизації
│   ├── Register.tsx         # Форма реєстрації
│   ├── RequestsList.tsx     # Перегляд та фільтрація активних заявок
│   ├── CreateRequest.tsx    # Створення нової заявки заявником
│   ├── MyRequests.tsx       # Перегляд та керування власними заявками
│   ├── MyAssignments.tsx    # Список завдань волонтера
│   └── AdminPanel.tsx       # Адміністративна панель керування платформою
│
├── contexts/
│   └── AuthContext.tsx      # Глобальний стан автентифікації та ролей користувача
│
├── services/
│   └── api.ts               # Централізований Axios-клієнт для роботи з бекендом
│
├── App.tsx                  # Конфігурація компонентів та маршрутизації
└── index.tsx                # Точка входу в застосунок
```

Рис. 3.6. Структура директорій фронтенд-частини проєкту

Папка components/ містить окремі сторінки та інтерфейсні блоки, кожен з яких виконує чітко визначену функцію. Наприклад, RequestsList.tsx відповідає за отримання та відображення списку заявок, CreateRequest.tsx — за створення нової заявки, а AdminPanel.tsx — за управління користувачами та даними платформи. Така організація спрощує повторне використання UI-елементів, а також робить компоненти незалежними один від одного.

Глобальна логіка автентифікації зосереджена в AuthContext.tsx, що містить інформацію про поточного користувача, токен авторизації та методи управління сесією (login, logout, register). Використання Context API усуває потребу передавати дані через декілька рівнів компонентів.

Папка services/ містить модуль api.ts, який реалізує централізований клієнт для запитів до серверної частини. Він відповідає за додавання JWT-

токена до заголовків запитів, обробку помилок та визначення базової адреси API. Такий підхід полегшує зміну адреси сервера або розширення API в майбутньому.

Файл App.tsx виконує роль основного контейнера інтерфейсу та містить конфігурацію React Router, яка визначає доступні маршрути та їхню доступність залежно від ролі користувача.

Файл index.tsx виконує ініціалізацію React-додатку та підключає кореневий компонент.

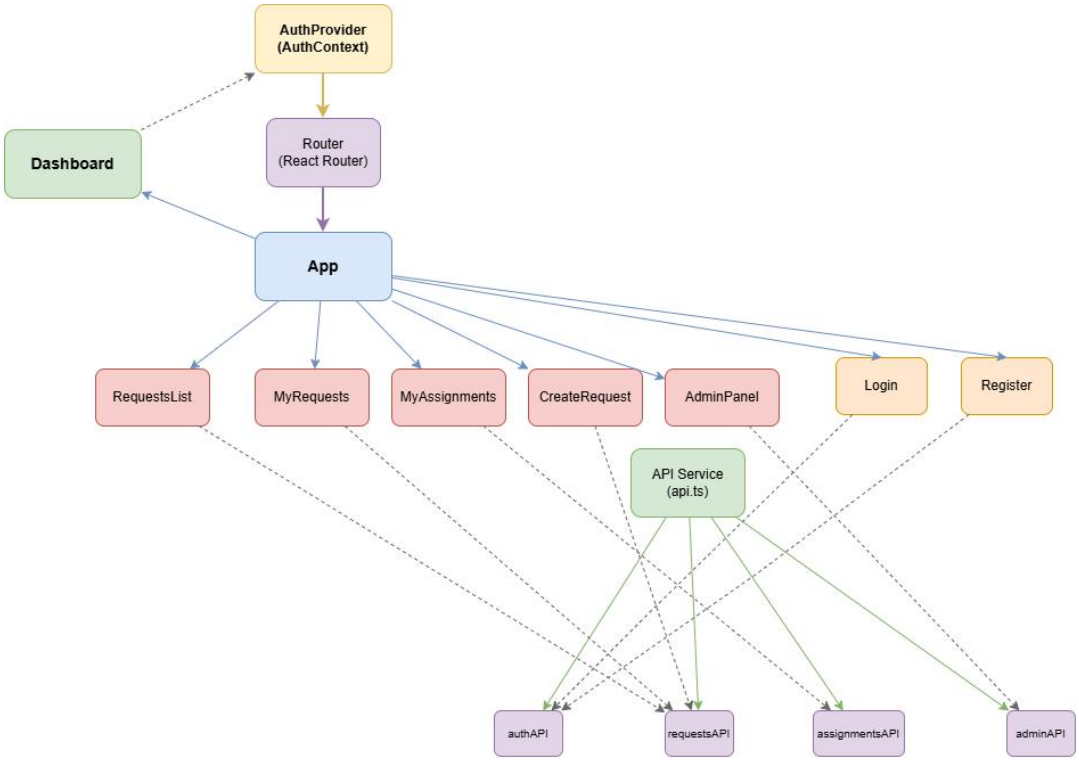

Рис. 3.7. Ієрархія компонентів клієнтської частини

Загалом, обрана структура дозволяє розширювати функціональність без порушення цілісності системи, підтримує зручність командної розробки та забезпечує високу читабельність коду.

3.3.3. Система автентифікації та управління доступом

Система автентифікації та авторизації у клієнтській частині платформи побудована на використанні механізму JSON Web Token (JWT) у поєднанні з глобальним контекстом стану (React Context API). Такий підхід забезпечує

збереження сесії користувача між оновленнями сторінки, а також гарантує контроль доступу до функцій та інтерфейсних елементів відповідно до ролі користувача. Після успішного входу на платформу сервер генерує токен, який передається клієнту і зберігається у локальному сховищі браузера (localStorage). Це дозволяє відновити активну сесію навіть після закриття вкладки або перезавантаження сторінки.

Ключова логіка автентифікації реалізована у модулі AuthContext, який інкапсулює інформацію про поточного користувача, отриманий токен, а також методи для входу, реєстрації та виходу із системи. AuthContext автоматично перевіряє наявність збереженого токена під час завантаження сторінки і, у разі його існування, виконує запит на отримання актуального профілю користувача. Це дозволяє уникнути повторної авторизації та підвищує зручність взаємодії з системою. Додатково в контексті відстежується стан завантаження, що забезпечує коректну обробку сценаріїв, коли дані користувача ще не отримані з сервера.

Для захисту маршрутів у застосунку використовується спеціалізований компонент ProtectedRoute, який верифікує наявність автентифікованого користувача перед рендерингом відповідної сторінки. Якщо користувач не авторизований або токен є недійсним, здійснюється перенаправлення на сторінку входу. Завдяки цьому механізму обмежується доступ до функціоналу, який не повинен бути доступний анонімним відвідувачам платформи. Компонент також враховує стан завантаження автентифікаційних даних, що запобігає передчасному відображенню інтерфейсу до завершення перевірки.

У системі реалізовано розмежування доступу за ролями (RBAC — Role-Based Access Control). Передбачено три основні ролі: заявник, волонтер та адміністратор. Кожна роль має власний набір доступних можливостей. Заявник створює та переглядає власні заявки, а також взаємодіє з відгуками волонтерів. Волонтер переглядає загальний список активних заявок та має можливість відгукнутися на будь-яку з них. Адміністратор отримує розширені права, що включають модерацію контенту, управління користувачами та доступ до

журналу аудиту. Такий підхід гарантує, що дані та функціональність системи використовуються відповідно до наданих повноважень.

Таким чином, поєднання JWT, Context API та механізму захищених маршрутів дозволяє клієнтській частині забезпечити безперервність сеансу, безпечний обмін даними із сервером і коректне застосування рольової моделі доступу. Ця архітектура є гнучкою, легко масштабованою та придатною для подальшого розширення, включаючи інтеграцію багатофакторної автентифікації або зовнішніх сервісів авторизації (наприклад, Google OAuth) у майбутній перспективі.

3.3.4. Основні React-компоненти та їх функціональність

Клієнтська частина веб-платформи побудована з використанням компонентного підходу, що дає змогу розділити інтерфейс на незалежні функціональні блоки. Кожен компонент відповідає за конкретний елемент взаємодії користувача із системою. Завдяки цьому досягається модульність, можливість повторного використання елементів, а також зменшується загальна складність супроводу коду. Ключові компоненти інтерфейсу охоплюють механізми автентифікації, перегляду та створення заявок, управління відгуками волонтерів та виконання адміністративних функцій.

Головним елементом взаємодії є компонент Dashboard, який слугує початковою сторінкою для автентифікованого користувача. Інтерфейс цієї панелі є адаптивним і залежить від ролі користувача. Для заявника відображаються функції створення та керування власними заявками; для волонтера — перегляд доступних запитів та управління поточними завданнями; а для адміністратора — доступ до адмінпанелі. Таким чином, Dashboard забезпечує персоналізований доступ до функціоналу платформи.

Для перегляду всіх публічних заявок використовується компонент RequestsList. Він формує список доступних запитів на допомогу, отриманих із серверної частини за допомогою HTTP-запитів. Додатково передбачені інструменти фільтрації за статусом, рівнем терміновості та районом Києва.

Ключовою особливістю є механізм відгуку волонтера: користувач із відповідною роллю може ініціювати відгук, проте контактні дані заявника залишаються прихованими до моменту підтвердження заявником. Така поведінка є важливою для забезпечення приватності та безпеки.

Створення нових заявок реалізується у компоненті CreateRequest. Він являє собою форму з обов'язковими полями, що описують потребу, її категорію, район міста та контактні дані. Для полегшення заповнення використовуються випадаючі списки стандартизованих категорій та районів. Система також дозволяє вибрати рівень терміновості, що сприяє швидшому реагуванню на пріоритетні заяви. Після заповнення форми дані надсилаються на сервер для фіксації у базі даних.

Компонент MyRequests призначений для заявників та дозволяє відстежувати статус створених ними заявок. Якщо на заявку надійшов відгук волонтера, заявник може ухвалити рішення: підтвердити або відхилити пропозицію. Лише після підтвердження контакти заявника стають доступними для волонтера. Такий підхід запобігає небажаним контактам і забезпечує контроль над процесом передачі інформації.

Управління завданнями волонтера здійснюється через компонент MyAssignments. Він відображає список заявок, на які волонтер відгукнувся та отримав підтвердження. Компонент дозволяє переглядати детальну інформацію та зв'язуватися із заявником напряму. Також передбачена можливість зміни статусу завдання, наприклад, на «виконано» після надання допомоги. Таким чином формується історія участі волонтера у волонтерській діяльності.

Для адміністратора реалізований компонент AdminPanel, який забезпечує централізоване управління користувачами та заявками. Він надає можливість переглядати список усіх зареєстрованих користувачів, змінювати їхні ролі або видаляти облікові записи. Аналогічно, адміністратор може переглядати всі заявки, контролювати їхні статуси та видаляти дублікати чи некоректні записи. Додатковим функціональним елементом є журнал аудиту, де фіксуються всі зміни у системі, що дозволяє забезпечити її прозорість і контрольованість.

Загалом вся система побудована на принципах, що забезпечують зручність використання для всіх типів користувачів, підтримують послідовність інтерфейсу та гарантують достовірність і безпечність обміну даними.

3.3.5. Маршрутизація та навігація

Маршрутизація у клієнтській частині платформи реалізована за допомогою бібліотеки React Router DOM, яка дозволяє організувати навігацію між сторінками без перезавантаження веб-сторінки. Це відповідає принципам SPA (Single Page Application), коли вся логіка маршрутизації виконується на боці клієнта, а зміна вмісту інтерфейсу здійснюється динамічно. Такий підхід забезпечує високу швидкість роботи інтерфейсу та покращує користувацький досвід, оскільки взаємодія з платформою відбувається плавно та без видимих затримок.

Основним контейнером маршрутизації є компонент App, який містить визначення основних маршрутів і дозволяє керувати тим, які сторінки доступні користувачу залежно від його статусу автентифікації. Для цього використовується спеціальний компонент ProtectedRoute, що перевіряє наявність активної сесії користувача перед тим, як дозволити доступ до тієї чи іншої сторінки. Якщо користувач не авторизований, система автоматично перенаправляє його на сторінку входу. Це дозволяє запобігти несанкціонованому доступу до функцій платформи, що потребують ідентифікації особи.

Навігація між сторінками платформи організована з урахуванням ролей користувачів. Заявник після авторизації потрапляє на панель управління, звідки може перейти до створення заявок та перегляду власних запитів. Волонтер з панелі управління може перейти до списку усіх доступних заявок і керувати завданнями, які були йому призначені після підтвердження з боку заявника. Адміністратор, у свою чергу, з панелі управління може перейти до окремої адміністративної панелі, що дозволяє здійснювати контроль за всією системою та її користувачами. Таким чином, маршрутизація платформи виконує не лише

навігаційну, але й рольову функцію, забезпечуючи персоналізований доступ до функціоналу.

Окрему роль відіграє маршрутизація за замовчуванням. Якщо користувач відкриває кореневу сторінку платформи, система автоматично перенаправляє його на сторінку з переліком активних заявок. Це дозволяє забезпечити інтуїтивну логіку використання системи, при якій найважливіший функціонал — перегляд потреб у допомозі — знаходиться у центрі взаємодії. Такий підхід також полегшує процес залучення волонтерів, адже вони одразу бачать актуальні запити, що потребують реагування.

Отже, реалізована система маршрутизації забезпечує зручну навігацію між розділами платформи, узгоджену з ролями користувачів та їхніми повноваженнями. Плавність переходів між веб-сторінками та відсутність перезавантаження сприяють швидкій взаємодії користувача з інтерфейсом і створюють сучасний, зрозумілий і доступний додаток.

3.4. Забезпечення безпеки та зберігання даних

Забезпечення безпеки користувацьких даних є одним із ключових аспектів функціонування волонтерської платформи, оскільки система оперує персональною та контактною інформацією заявників і волонтерів. Для мінімізації ризиків несанкціонованого доступу та гарантування конфіденційності даних у проєкті застосовано комплексний підхід, що поєднує методи автентифікації, авторизації, шифрування та контроль дій користувачів.

Основним механізмом автентифікації є JWT-токени. Після успішного входу чи реєстрації користувач отримує токен, який зберігається у браузері та додається у заголовки кожного наступного запиту. Це дозволяє уникнути збереження сесій на сервері та спрощує масштабування. Термін дії токена обмежено 24 годинами, що вимагає повторного підтвердження після його закінчення. Токен підписується секретним ключем із серверних змінних середовища для додаткового захисту.

Система також реалізує рольову модель доступу (RBAC). Користувачеві під час реєстрації призначається роль: заявник, волонтер або адміністратор. Кожен маршрут API має відповідний рівень доступу, який перевіряється через проміжний обробник (middleware) на сервері. Це дозволяє чітко розмежувати можливості різних груп користувачів (наприклад, волонтер не може видаляти заявки) і запобігти небажаним діям.

Особливу увагу приділено захисту паролів. Вони зберігаються виключно в зашифрованому вигляді за допомогою алгоритму bcryptjs. Цей механізм забезпечує надійний захист паролів через використання випадкових значень та багаторазового хешування, що робить неможливим відновлення оригінальних паролів навіть при компрометації бази даних. Перевірка пароля здійснюється шляхом порівняння хешу введеного значення з збереженим у базі даних.

У подальшому розвитку системи передбачається впровадження верифікації користувачів через державний сервіс «Дія». Така інтеграція забезпечить додатковий рівень безпеки, дозволить підтверджувати особу користувачів за допомогою офіційних електронних документів і зменшить ризики створення фіктивних облікових записів. Це сприятиме підвищенню довіри між волонтерами, заявниками та адміністраторами платформи, а також дозволить розширити функціональні можливості системи в напрямі цифрової ідентифікації.

Важливим елементом безпеки є контроль доступу до контактних даних заявника. Контактний номер або обліковий запис Telegram не відображаються волонтеру автоматично. Спершу волонтер подає відгук, після чого заявник має підтвердити його. Лише після підтвердження система розкриває контактні дані у відповідному завданні. Цей механізм захищає заявників від небажаних контактів і зміцнює довіру до платформи.

Дані зберігаються у реляційній базі даних: PostgreSQL у продакшені та SQLite для локальної розробки/тестування. Додатково реалізовано журнал аудиту, де фіксуються ключові зміни в системі: створення та редагування заявок, зміни ролей, прийняття або відхилення відгуків. Це дає змогу

контролювати діяльність у системі та відтворювати ланцюжок дій у спірних ситуаціях.

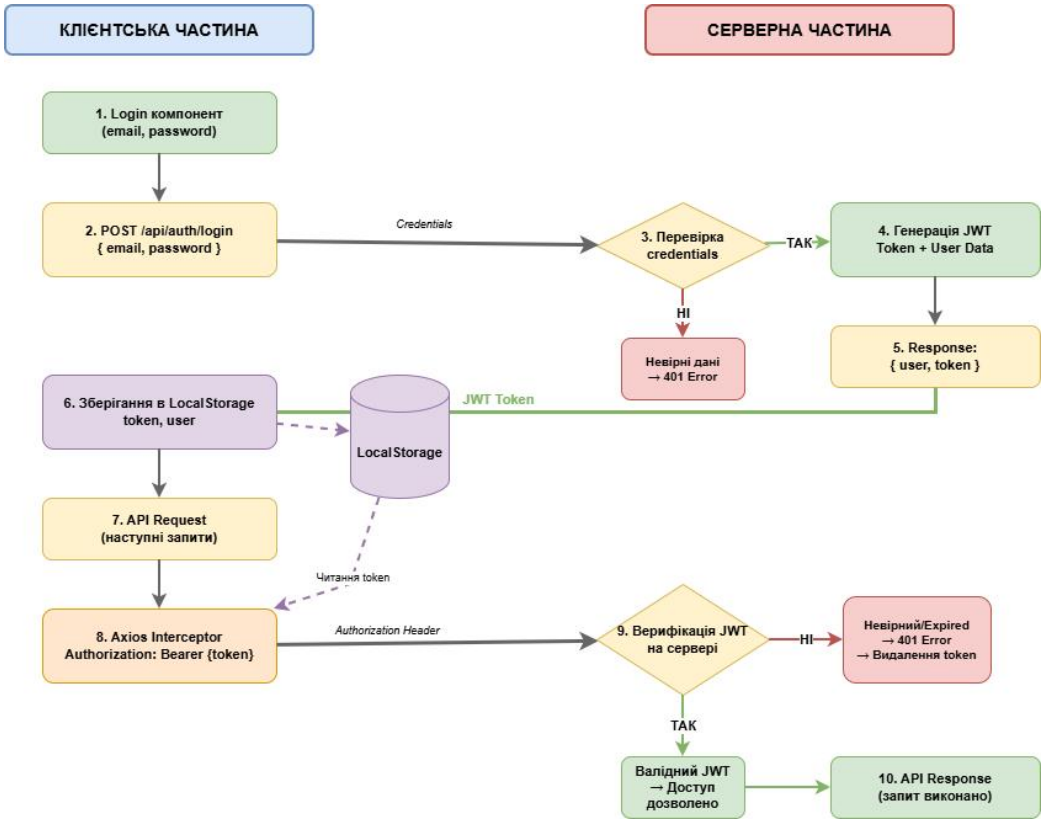

Рис. 3.8. Схема автентифікації користувача за допомогою JWT

На рисунку 3.8 наведено схему автентифікації, реалізовану за допомогою механізму JSON Web Token (JWT). Процес починається із введення користувачем облікових даних у формі Login. Клієнтський інтерфейс ініціює HTTP-запит типу POST до кінцевої точки /api/auth/login, передаючи credentials. Серверна частина системи виконує верифікацію наданих даних у базі даних. У разі успішної автентифікації генерується JWT-токен, і сервер повертає об'єкт, що містить інформацію про користувача (user) та згенерований токен (token). Клієнт зберігає отриманий токен у LocalStorage браузера. При наступних API-запитах спеціалізований перехоплювач Axios Interceptor автоматично додає токен до заголовка запиту у форматі Authorization: Bearer {token}. Сервер виконує верифікацію JWT при кожному захищеному запиті. У випадку невалідності токена або його відсутності повертається статус помилки 401

(Unauthorized), після чого токен видаляється з LocalStorage і здійснюється перенаправлення (редирект) користувача на сторінку входу (/login).

Таким чином, система забезпечує комплексний захист даних, що ґрунтується на поєднанні криптографічних методів, структурованої авторизації, обмеженого доступу до персональних даних та механізмів журналювання дій. Ці заходи дозволяють створити безпечне середовище взаємодії та підтримують довіру користувачів до платформи.

Особливістю розробленої системи є поетапний доступ до конфіденційної інформації — контактні дані заявника відкриваються лише після підтвердження його згоди. Крім того, журнал аудиту автоматично фіксує всі зміни, забезпечуючи прозорість і контроль дій користувачів. У майбутньому планується впровадження верифікації через сервіс «Дія», що підвищить рівень безпеки та довіри до системи.

3.5. Перевірка функціональності роботи веб-платформи

3.5.1. Перевірка реєстрації та автентифікації користувачів

Першим етапом перевірки було тестування механізмів створення облікових записів і входу до системи. На сторінці «Реєстрація» користувач заповнює форму, зазначаючи ім'я, прізвище, електронну адресу, пароль, телефон і роль — «заявник» або «волонтер». У процесі перевірки було підтверджено, що система коректно відображає повідомлення про обов'язкові поля та не дозволяє реєстрацію з уже існуючою електронною адресою.

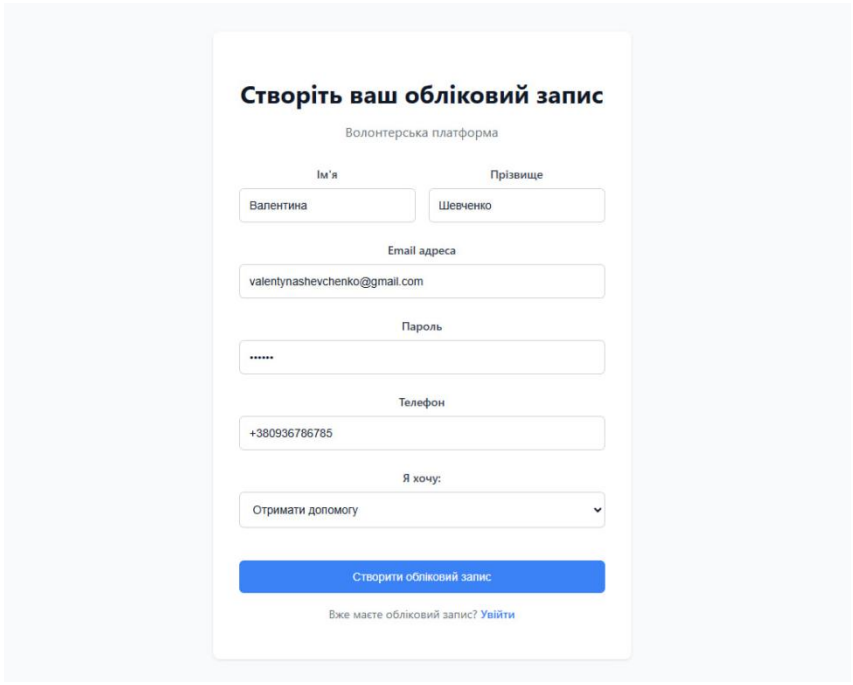

Рис. 3.9. Форма реєстрації нового користувача у веб-платформі

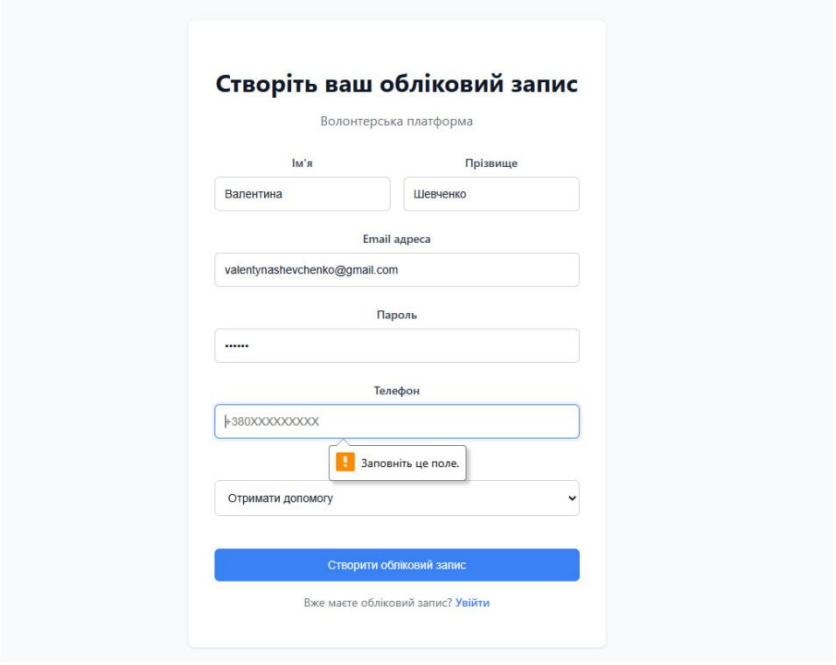

Рис. 3.10. Відображення повідомлення про обов’язкові поля під час реєстрації

Після успішного створення облікового запису сервер генерує JWT-токен, який автоматично зберігається у localStorage разом з даними користувача. Це дозволяє користувачу залишатися авторизованим навіть після оновлення сторінки.

Було також перевірено коректність перенаправлення після входу та реєстрації: всі користувачі незалежно від ролі автоматично потрапляють на сторінку «Панель управління» (/dashboard), де відображаються відповідні дії згідно з їхньою роллю — заявники бачать можливість створити заявку та переглянути свої заявки, волонтери можуть переглянути заявки та свої завдання, а адміністратори отримують доступ до панелі керування.

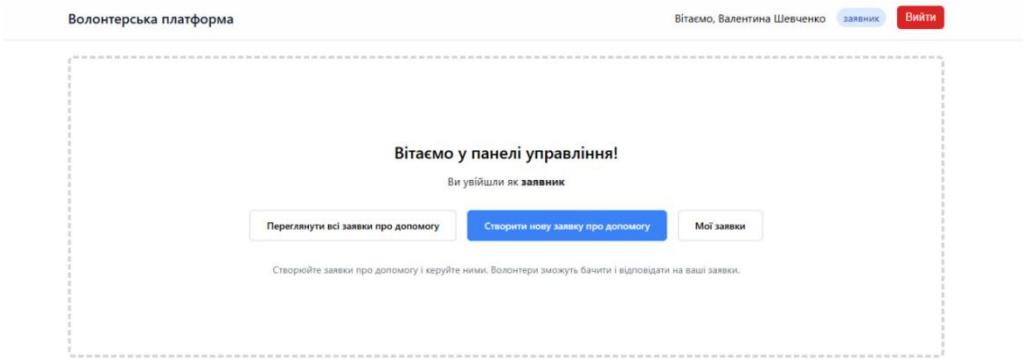

Рис. 3.11. Сторінка входу та результат успішної автентифікації користувача

Додатково перевірено поведінку при неавторизованому доступі: спроба перейти до захищеного маршруту без наявності користувача у контексті авторизації призводить до перенаправлення на сторінку входу. Це підтвердило правильну роботу механізму ProtectedRoute і реалізацію клієнтської авторизації через контекст AuthContext.

Проведена перевірка підтвердила стабільну роботу всіх компонентів, пов’язаних із системою реєстрації, входу та захисту маршрутів. Усі користувачі успішно проходять автентифікацію. Неавторизовані запити коректно блокуються на клієнтському рівні через механізм ProtectedRoute (перенаправлення на сторінку входу) та на серверному рівні через перевірку JWT-токенів. Реалізація механізмів ProtectedRoute та AuthContext забезпечує зручну роботу з інтерфейсом і коректне розмежування прав користувачів.

3.5.2. Перевірка створення та відображення заявок

Другим етапом стало тестування створення заявок про потребу у допомозі. На сторінці «Створити заявку» заявник заповнює форму, зазначаючи назву, опис, категорію допомоги, район Києва, рівень терміновості й контактну інформацію — номер телефону або Telegram.

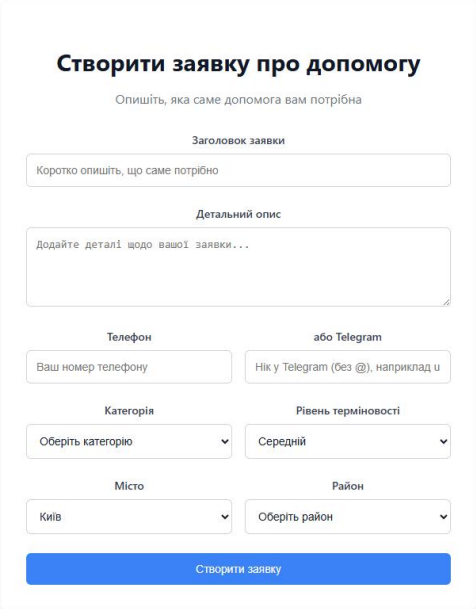

Рис. 3.12. Інтерфейс сторінки «Створити заявку»

Було перевірено, що система не дозволяє відправити форму з порожніми обов'язковими полями або без способу зв'язку (телефон або Telegram).

Рис. 3.13. Повідомлення про незаповнені поля під час створення заявки

Після успішного надсилання запит передається до API, де сервер виконує перевірку даних і створює новий запис у таблиці Requests зі статусом «створено». На екрані користувача з'являється повідомлення про успішне створення, форма очищається, а у списку заявок система відображає нову позицію.

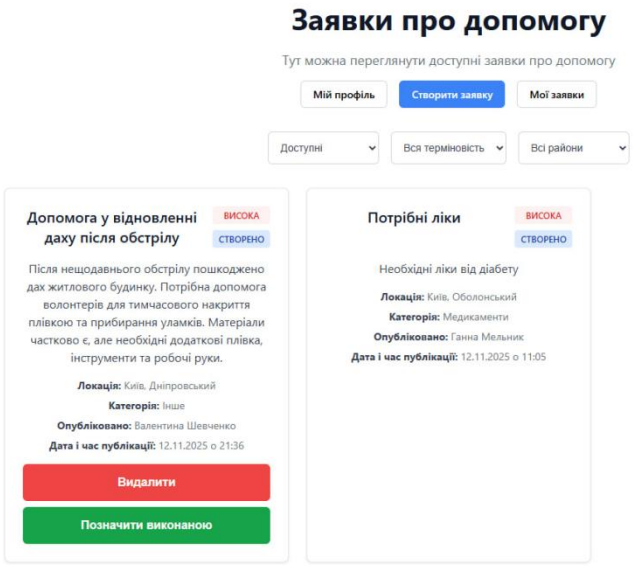

Рис. 3.14. Нова заявка у списку доступних заявок

Під час тестування також перевірено можливості управління заявками: заявники можуть видаляти свої заявки або змінювати їх статус на «виконано».

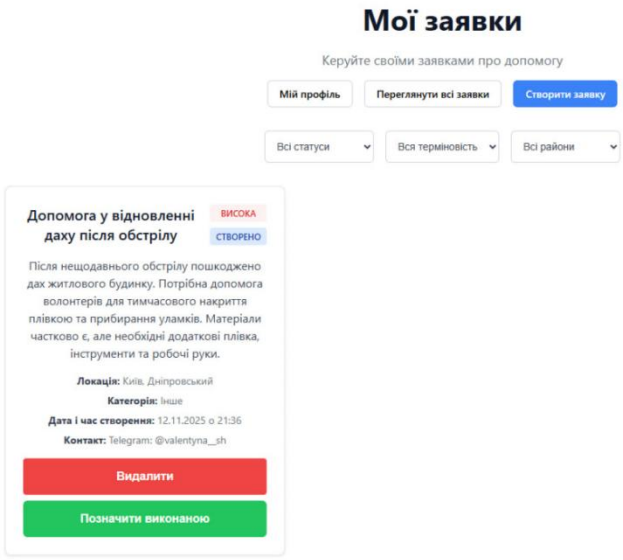

Рис. 3.15. Управління статусом створеної заявки користувачем

Отже, функціональність створення та управління заявками працює коректно. Система забезпечує перевірку обов’язкових полів, відображає повідомлення про помилки, підтримує зв’язок із серверною частиною через API та синхронно оновлює інтерфейс після успішного створення запису. Проведені тести підтвердили стабільність клієнтської логіки та узгодженість обміну даними між фронтендом і бекендом.

3.5.3. Перевірка роботи сторінки волонтера та фільтрації заявок

Наступним етапом є тестування ролі волонтера. Після входу до системи волонтер переходить на сторінку «Панель управління», звідки може перейти на сторінку «Заявки про допомогу», де система отримує актуальні дані через API-запит. На основі отриманої інформації формується список активних заявок із коротким описом і категорією.

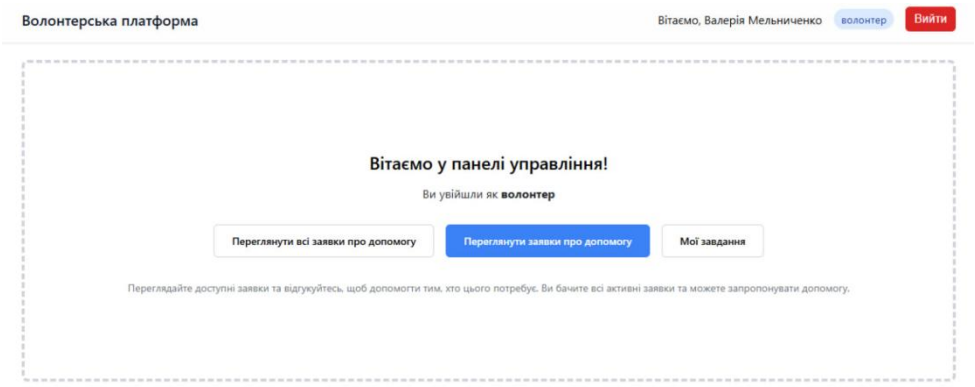

Рис. 3.16. Головна панель волонтера після входу до системи

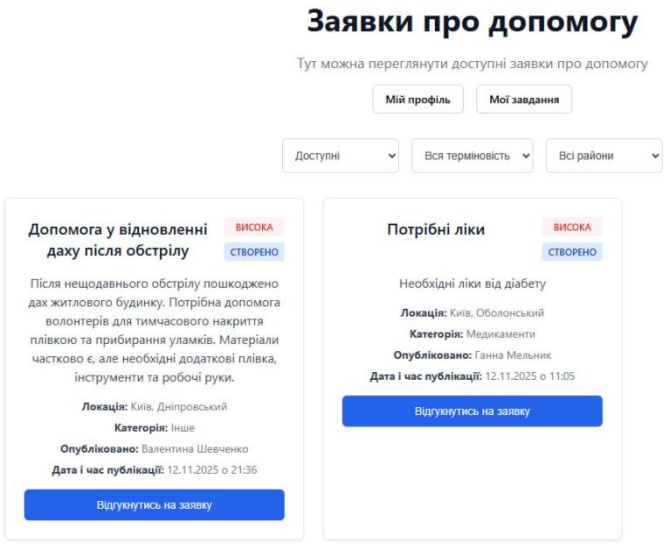

Рис. 3.17. Перегляд списку активних заявок волонтером

Було перевірено коректність роботи фільтрів: волонтер може обирати статус заявки, район Києва або рівень терміновості. Фільтри можуть застосовуватися одночасно, а оновлення списку відбувається без перезавантаження сторінки, що підтверджує правильну реалізацію React Hooks.

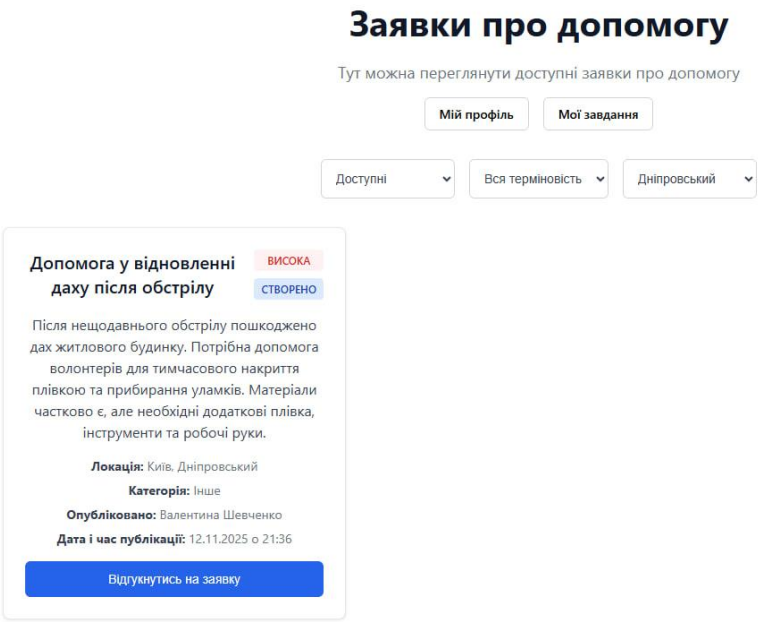

Рис. 3.18. Використання фільтрів під час перегляду заявок

Крім того, тестувалася логіка обмеження доступу до персональних даних. У переліку заявок волонтер бачить лише ім'я заявника — контактні номери чи посилання на Telegram взагалі не відображаються в списку заявок. Контактні дані стають доступними лише після прийняття заявки заявником у розділі «Мої заявки». Такий механізм забезпечує конфіденційність і відповідає принципам захисту персональної інформації.

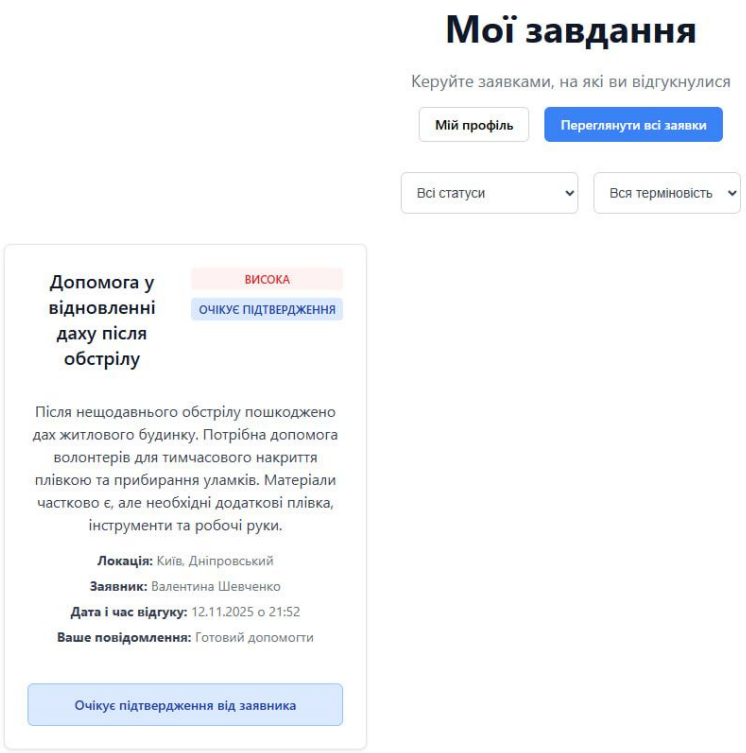

Рис. 3.19. Відображення персональних даних заявника до підтвердження заявки заявником

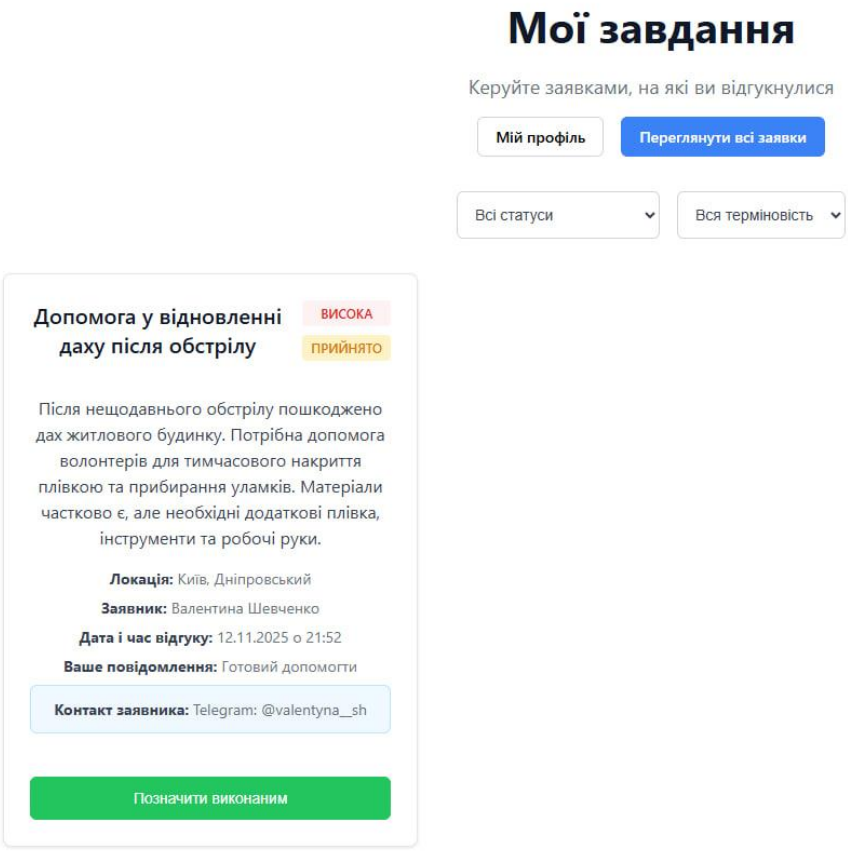

Рис. 3.20. Контактні дані заявника після підтвердження заявки

У результаті тестування підтверджено коректну роботу механізму відображення заявок для волонтерів, фільтрації за кількома параметрами та обмеження доступу до персональних даних. Реалізований підхід до поступового відкриття контактної інформації лише після підтвердження заявником гарантує конфіденційність користувачів та відповідає вимогам безпеки платформи.

3.5.4. Перевірка відгуків волонтерів та підтвердження заявником

У цьому етапі перевірки досліджувався процес взаємодії між заявником і волонтером. Після перегляду заявки волонтер натискає кнопку «Відгукнутись», що створює запис у таблиці Assignments зі статусом «pending» (очікує підтвердження).

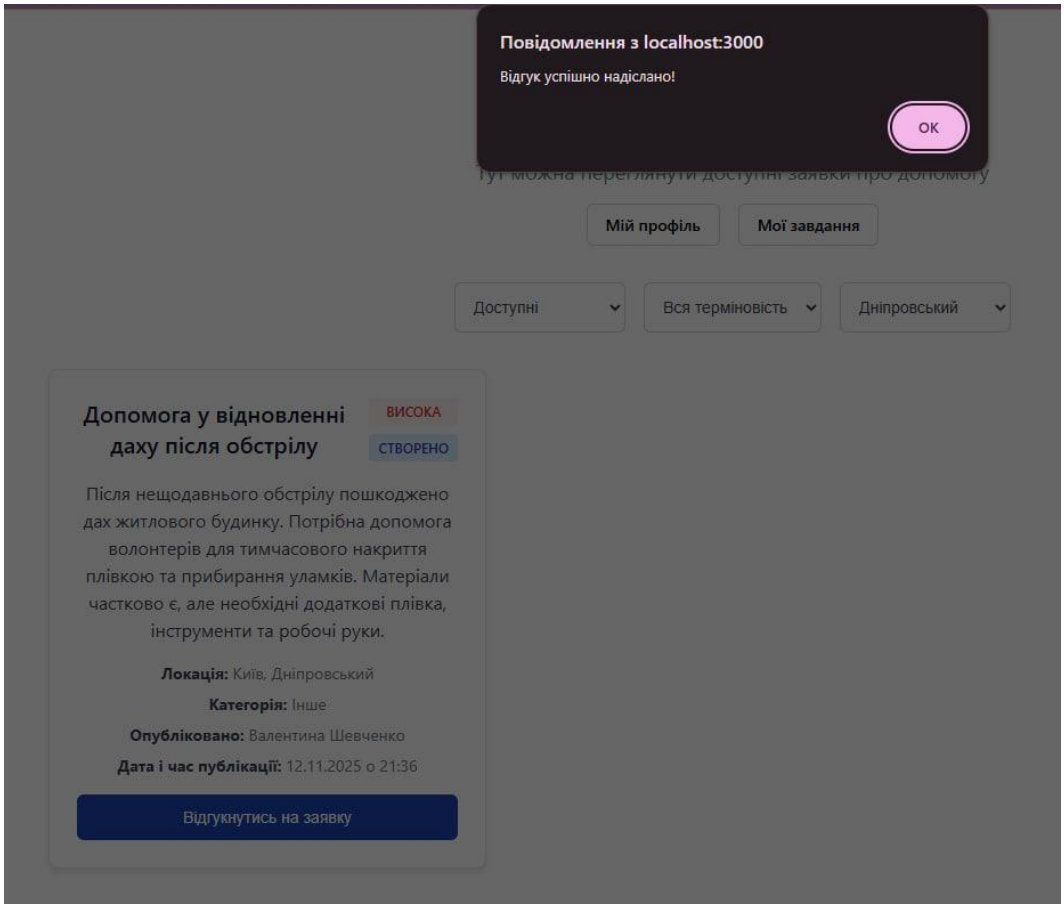

Рис. 3.21. Відправлення відгуку волонтером на заявку

На стороні заявника у кабінеті «Мої заявки» з'являється повідомлення про новий відгук. Заявник може прийняти одного волонтера або відхилити його

участь. Після прийняття система автоматично змінює статус заявки на «прийнято», роблячи її недоступною для інших волонтерів, а волонтеру стають доступними контактні дані заявника у розділі «Мої завдання». Якщо відгук відхилено, контактна інформація залишається прихованою.

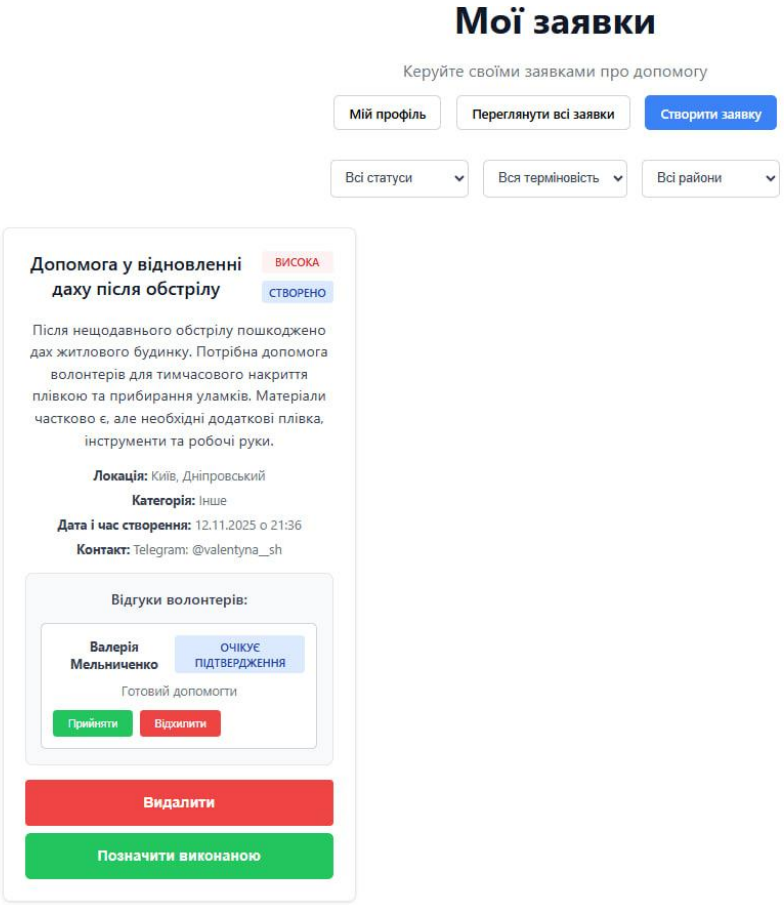

Рис. 3.22. Повідомлення про новий відгук волонтера у кабінеті заявника

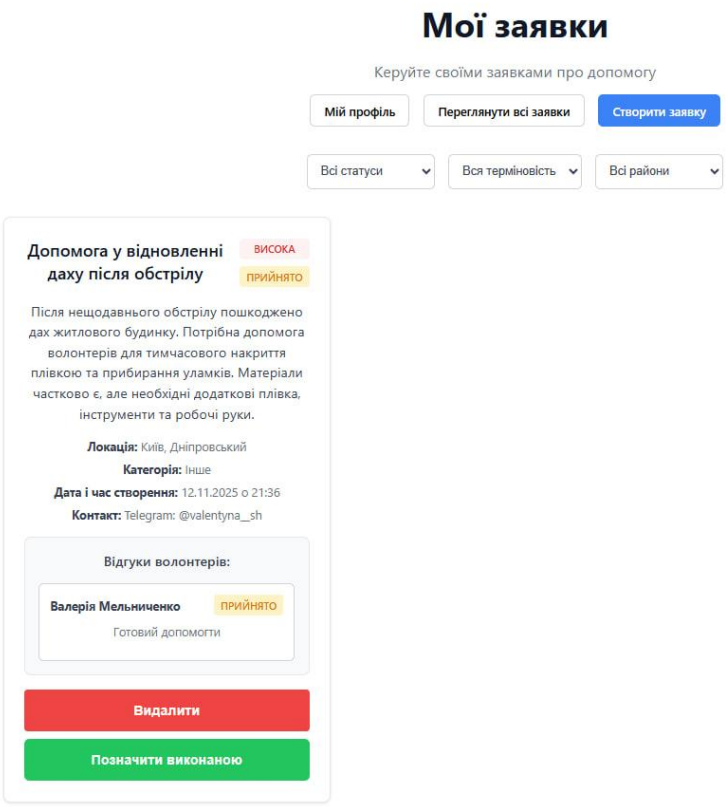

Рис. 3.23. Підтвердження відгуку волонтера заявником

Проведене тестування підтвердило коректну взаємодію між клієнтською та серверною частинами під час створення і прийняття волонтерських відгуків. Система успішно виконує логіку призначення, автоматично оновлює статуси та забезпечує доступ лише до необхідних персональних даних. Така реалізація гарантує безпечну та контрольовану співпрацю між користувачами різних ролей.

3.5.5. Перевірка виконання завдань і зміни статусів

Після підтвердження заявки волонтер переходить у розділ «Мої завдання». Тут він бачить перелік усіх активних відгуків зі статусами. Волонтер може позначити своє завдання як «виконано», але це не змінює автоматично статус заявки у заявника.

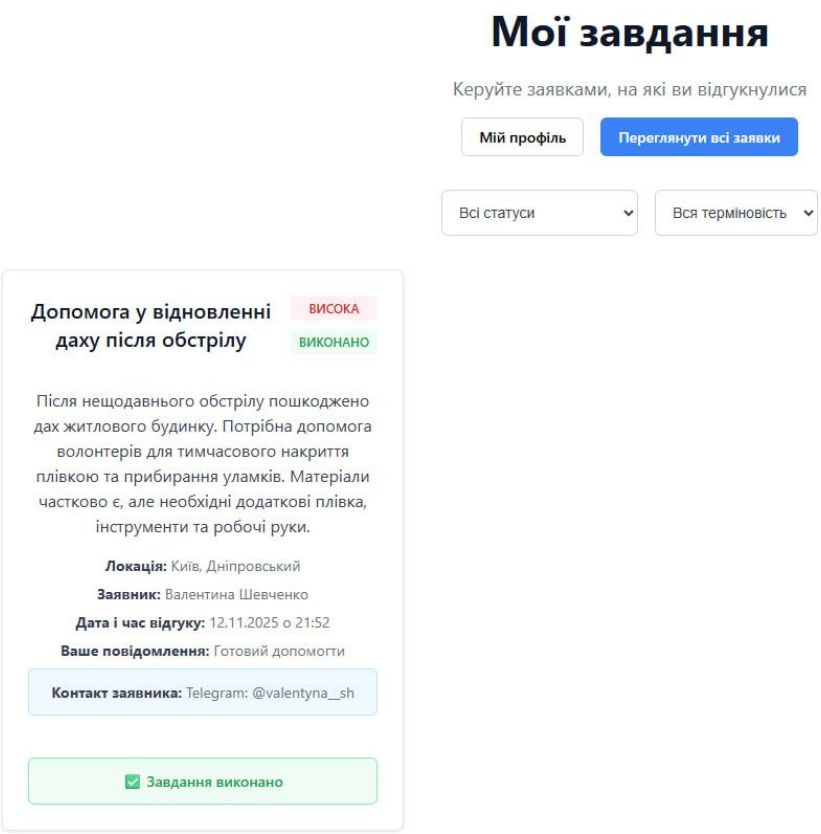

Рис. 3.24. Завершення завдання волонтером

Було перевірено, що після оновлення сторінки статуси зберігаються, а на стороні заявника заявник може самостійно змінити статус заявки на «виконано» через кнопку «Позначити виконаною». У таблиці AuditLog автоматично фіксується запис із вказівкою користувача, дії, таблиці, запису та опису змін.

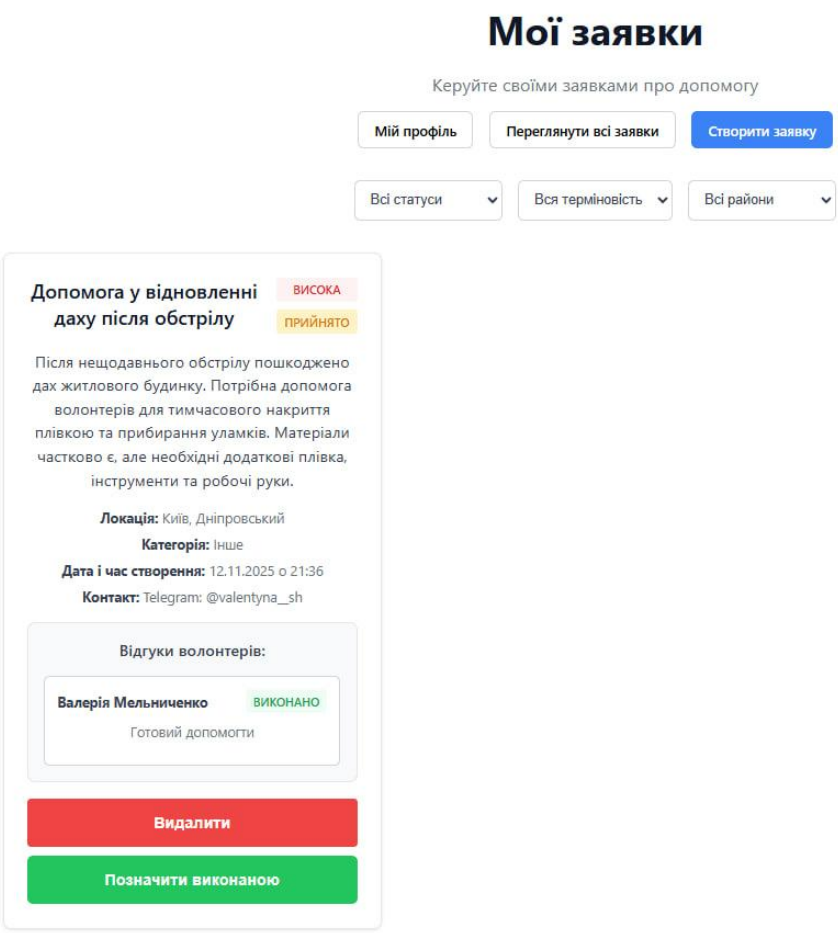

Рис. 3.25. Перегляд статусу заявки заявником після завершення завдання волонтером

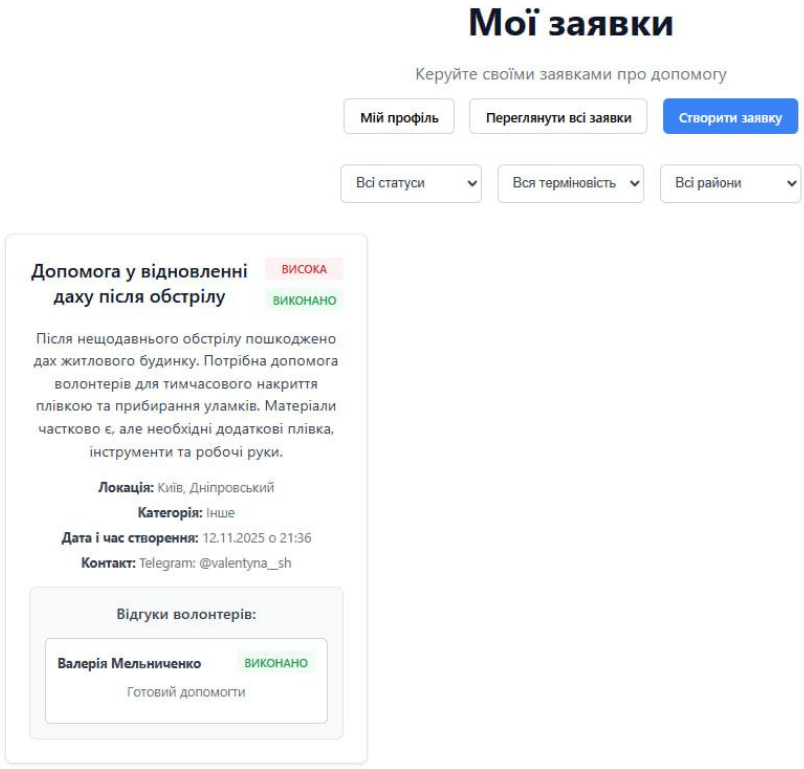

Рис. 3.26. Підтвердження виконання заявки заявником

Таким чином підтверджено, що система коректно обробляє взаємодію між волонтером і заявником, а обидві сторони бачать актуальний стан у реальному часі, хоча процес завершення заявки контролюється заявником, а не волонтером.

3.5.6. Перевірка адміністративної панелі та журналу аудиту

Окремий етап перевірки стосувався функцій адміністратора. У розділі «Адмінпанель» адміністратор може переглядати користувачів, заявки та журнал дій. Було протестовано, що при спробі доступу без відповідної ролі сервер повертає помилку, а клієнтська частина відображає повідомлення про відсутність прав доступу.

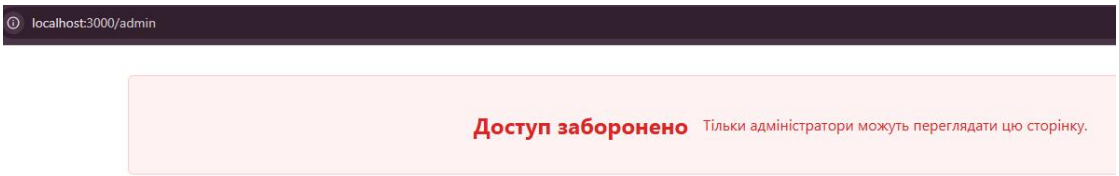

Рис. 3.27. Відображення помилки доступу при спробі відкрити адмінпанель без відповідних прав

Під час тестування адміністратор успішно змінював статуси заявок, змінював ролі користувачів та видаляв користувачів і заявки, а також переглядав записи журналу аудиту. Всі зміни відображалися миттєво, що свідчить про узгоджену роботу бекенду та фронтенду.

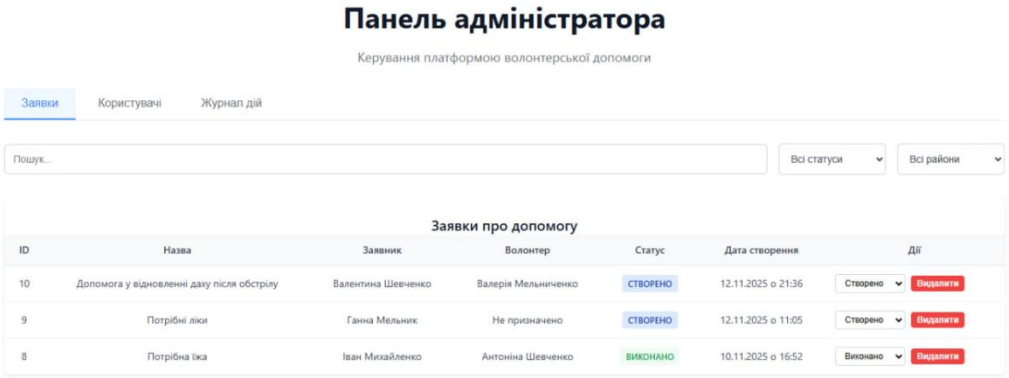

Рис. 3.28. Інтерфейс адміністративної панелі з основними розділами

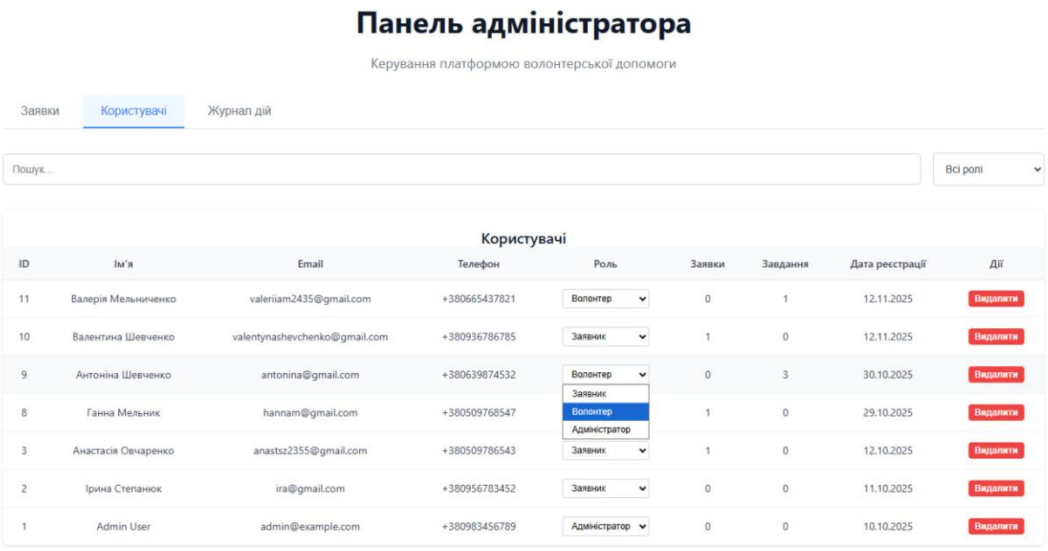

Рис. 3.29. Керування користувачами в адміністративній панелі

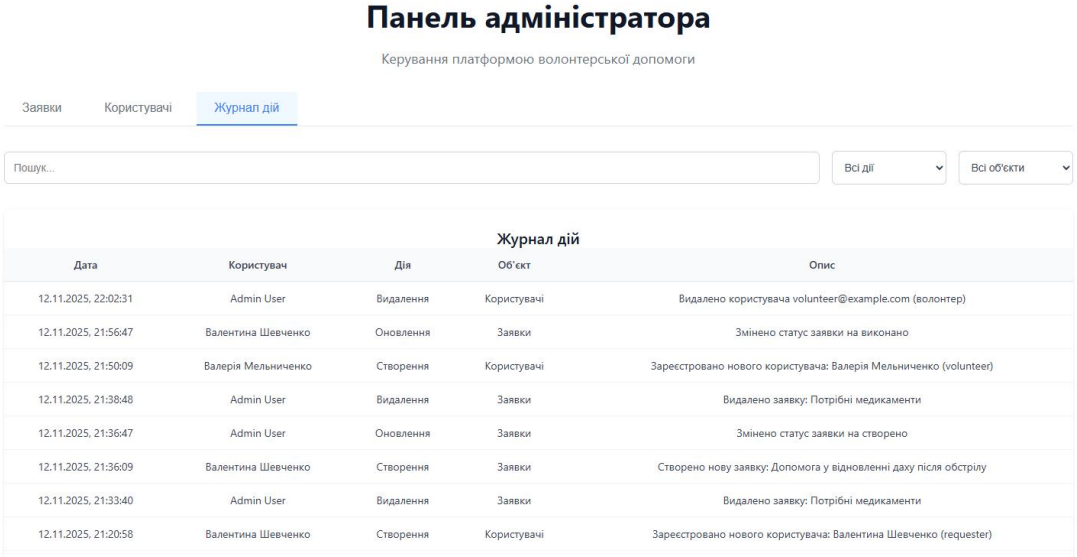

Рис. 3.30. Журнал дій з фіксацією дій користувачів

Таким чином підтверджено, що рольова модель системи реалізована правильно, а механізми контролю забезпечують повну прозорість дій усіх користувачів через систему аудиту.

3.5.7. Загальні результати тестування

За результатами комплексної перевірки підтверджено стабільну роботу всіх ключових компонентів веб-платформи, які відповідають функціональним вимогам. Механізми реєстрації та авторизації працюють коректно, створення й

обробка заявок відбуваються без збоїв, рольова модель забезпечує належний розподіл доступів, а адміністративні інструменти дають змогу ефективно керувати даними.

Особливу увагу приділено безпеці персональних даних і контрольованому доступу: система відображає лише релевантну інформацію відповідно до ролі користувача, а приватні контакти доступні виключно в межах погодженої взаємодії. Усі тести завершилися успішно, що підтверджує готовність платформи до розгортання та подальшого використання.

Висновок до розділу 3

У третьому розділі представлено процес проєктування та реалізації веб-платформи для координації волонтерської діяльності. Описано загальну архітектуру системи, логіку взаємодії користувачів та структуру даних, що стало основою для створення цілісної програмної моделі.

Серверна частина реалізована на Node.js з використанням фреймворку Express та REST API. Забезпечено обробку запитів, автентифікацію й авторизацію користувачів, валідацію даних та взаємодію з базою даних (SQLite у розробці та PostgreSQL у production). Використання JWT, моделі доступу на основі ролей (RBAC) та журналу аудиту забезпечує цілісність даних та контроль над ключовими діями в системі.

Клієнтська частина створена на React 19 з використанням TypeScript. Реалізовано компонентну структуру інтерфейсу, контекст автентифікації, захищені маршрути і централізовану роботу з HTTP-запитами через Axios. Система підтримує динамічне оновлення даних без перезавантаження сторінок (SPA).

Тестування підтвердило коректність усіх ключових сценаріїв: авторизації, створення та обробки заявок, взаємодії між волонтерами та заявниками, а також роботи адміністративної панелі. Було перевірено поведінку системи під час недійсних або неавторизованих запитів, а також узгодженість логіки клієнта та сервера.

Впроваджена система відповідає визначеним вимогам, забезпечує стабільну роботу, безпечну обробку даних та підтримує ефективну взаємодію між усіма учасниками волонтерського процесу. Отримані результати створюють міцну основу для подальшого розвитку платформи та розширення її функціональності.

РОЗДІЛ 4
РОЗРОБКА СТАРТАП ПРОЄКТУ

4.1 Загальна характеристика стартап-проекту

Основна ідея стартап-проєкту полягає у створенні веб-платформи, яка об’єднує заявників, що потребують допомоги, та волонтерів, готових її надати. Система дозволяє швидко створювати заявки, фільтрувати їх за районом, статусом і терміновістю, а також контролювати процес виконання у зручному інтерфейсі. Платформа може застосовуватись для координації гуманітарної, транспортної, медичної чи соціальної допомоги в межах міста Києва з перспективою подальшого масштабування на інші регіони. Основними перевагами є зручність пошуку та надання допомоги, прозорість комунікації, автоматизований облік заявок і підвищена безпека персональних даних. Порівняно з наявними аналогами, розроблена система вирізняється простим інтерфейсом, розширеними можливостями фільтрації та конфіденційністю: контакти заявників стають доступними волонтерам лише після підтвердження відгуку. Більш детальні характеристики наведено у таблицях 4.1–4.2.

Таблиця 4.1

Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Розробка веб-платформи, що поєднує заявників, які потребують допомоги, та волонтерів, готових її надати. Система дозволяє створювати,	1. Створення ефективного каналу комунікації між реципієнтами допомоги та її надавачами.	Скорочення часу очікування допомоги. Прямий доступ до необхідних ресурсів.
	2. Впровадження механізмів цифрового контролю та аудиту всіх операцій.	Підвищення довіри за рахунок прозорості. Верифікація використання ресурсів

Таблиця 4.1 (продовження)

Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
переглядати й фільтрувати заявки за статусом, терміновістю, категорією чи районом.	3. Забезпечення високого рівня захисту персональних даних та конфіденційності.	Гарантована безпека особистої інформації. Запобігання небажаним або шахрайським контактам.
	4. Можливість швидкого масштабування рішення та його адаптації до змінних локальних потреб.	Стабільність роботи платформи навіть при пікових навантаженнях. Постійний доступ до актуального функціоналу.

Таблиця 4.2

Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Конкурент 1	Конкурент 2			
1	Простота інтерфейсу та навігації	Інтуїтивно зрозумілий інтерфейс із розділенням ролей	Базовий дизайн, фокус на списках можливостей	Мінімалістичний, проте неадаптивний	Недостатньо оптимізовано під мобільні пристрої	Звичний UX для користувачів подібних платформ	Збалансований інтерфейс із адаптацією під усі ролі

Таблиця 4.2 (продовження)

Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Конкурент 1	Конкурент 2			
2	Можливість фільтрації заявок	Фільтри за статусом, терміновістю та районами	Часткова фільтрація за типом діяльності	Відсутня фільтрація	Не реалізовано розширений пошук	Стандартна фільтрація за напрямками	Гнучка багаторівнева система фільтрів
3	Захист персональних даних	Контакти приховані до підтвердження заявки	Публічні контакти волонтерів	Контакти видимі одразу після заявки	Відсутність двоетапної перевірки	Базова система автентифікації	Контрольований доступ і конфіденційність
4	Механізм підтвердження відгуків	Заявник підтверджує волонтера вручну	Автоматичне закріплення	Без модерації	Може потребувати затвердження адміністратором	Частково автоматизований процес	Прозорий і безпечний механізм взаємодії
5	Наявність журналу дій	Автоматичне логування в AuditLog	Відсутній	Відсутній	Не реалізовано аудит на клієнтському рівні	Часткова звітність через адміністративні інтерфейси	Підвищення прозорості та контролю
6	Рольова модель користувачів	Три ролі: заявник, волонтер, адміністратор	Дві ролі (користувач, організатор)	Дві ролі	Відсутність спеціалізації під групи волонтерів	Базовий рівень керування доступами	Гнучке управління ролями користувачів

Таблиця 4.2 (продовження)

Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Конкурент 1	Конкурент 2			
7	Аналітика та статистика	Панель адміністратора з метриками	Обмежена статистика	Відсутня	Відсутність візуалізації даних	Частковий збір аналітики	Повноцінна система статистики в адмінпанелі
8	Прозорість взаємодії	Історія змін заявок і дій користувачів	Часткова звітність	Без звітності	Відсутня можливість перевірки історії	Типове відображення статусів	Повна історія дій у реальному часі

4.2 Технологічний аудит проекту

Таблиця 4.3

Технологічна здійсненність ідеї проекту

№ п/п	Ідея проєкту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Розробка веб-платформи для організації волонтерської	React, TypeScript, Node.js, Express, PostgreSQL	Технології є відкритими та не потребують додаткової	Повністю доступні авторам проєкту, мають відкриту документацію

Таблиця 4.3 (продовження)

Технологічна здійсненність ідеї проекту

№ п/п	Ідея проєкту	Технології її реалізації	Наявність технологій	Доступність технологій
	діяльності		розробки	
2	Захист персональних даних і автентифікація користувачів	JWT, HTTPS, валідація запитів	Реалізуються за допомогою стандартних бібліотек	Доступні без обмежень у середовищі Node.js
3	Робота з базою даних та обмін даними через API	PostgreSQL, REST API, Axios	Технології стабільні та не потребують доопрацювання	Повністю доступні авторам і підтримуються сучасними інструментами розробки
Обрані технології реалізації ідеї проекту: React, Node.js, PostgreSQL				

За результатами аналізу таблиці встановлено, що запропонований проєкт є технологічно здійсненним, оскільки всі використані технології: React, TypeScript, Node.js, Express, PostgreSQL та JWT — є відкритими, стабільними й широко застосовуються у сучасній веб-розробці. Вони не потребують додаткових розробок або ліцензування, а також повністю доступні авторам проєкту.Технологічний шлях реалізації доцільно здійснювати на основі клієнт-серверної архітектури з використанням REST API для обміну даними між фронтендом і бекендом, що забезпечує масштабованість, безпеку та гнучкість системи.

4.3 Аналіз ринкових можливостей запуску стартап-проекту

Для оцінки перспектив впровадження веб-платформи волонтерської взаємодії було проведено попередній аналіз ринкових можливостей та умов розвитку цифрових сервісів соціального спрямування. Дослідження охоплює наявних гравців ринку, динаміку попиту, технічні обмеження та рівень відкритості до інновацій. Особливу увагу приділено оцінці бар'єрів входу на ринок, потенційній аудиторії користувачів і загальній тенденції зростання волонтерських ініціатив.

Таблиця 4.4

Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців	2–3 основні платформи
2	Загальний обсяг користувачів	Близько 200 тисяч
3	Динаміка ринку	Зростає
4	Наявність обмежень для входу	Низькі технічні бар'єри, висока конкуренція.
5	Специфічні вимоги до стандартизації	Дотримання вимог безпеки та захисту персональних даних.
6	Середня норма рентабельності	Умовна, соціальний проєкт із грантовим фінансуванням.

Таблиця 4.5

Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів
1	Оперативне отримання допомоги у кризових ситуаціях	Особи, що потребують гуманітарної, медичної чи психологічної підтримки	Шукають швидке реагування та простоту користування, уникають бюрократичних процедур	Зручний інтерфейс, стабільність роботи, гарантія конфіденційності
2	Швидкий пошук та координація запитів про допомогу	Волонтери, благодійні організації, ініціативні групи	Активно користуються мобільними інструментами, орієнтовані на ефективність і швидкість виконання	Прозорість процесів, ефективна фільтрація, доступність платформи
3	Централізо-ване управління волонтерсь-кою діяльністю	Координатори та адміністратори волонтерських ініціатив	Орієнтовані на моніторинг, керування командами та ресурсами	Аналітика, звітність, контроль виконання, технічна підтримка

Таблиця 4.5 (продовження)

Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів
4	Підвищення рівня довіри та безпеки взаємодії	Усі категорії користувачів платформи (волонтери, заявники, адміністратори)	Уникають ризиків зловживань, віддають перевагу перевіреним платформам	Верифікація користувачів, захист даних, прозорість дій у системі

Таблиця 4.6

Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Висока конкуренція на ринку волонтерських платформ	Існують популярні платформи з подібним функціоналом, що ускладнює залучення користувачів	Розробка унікального функціоналу, орієнтація на локальні спільноти та підвищення зручності користування
2	Недовіра користувачів до нових цифрових сервісів	Користувачі можуть сумніватися у безпеці персональних даних і достовірності заявок	Запровадження системи верифікації користувачів і відкритої політики безпеки

Таблиця 4.6 (продовження)

Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
3	Обмежене фінансування для просування проєкту	Відсутність стабільного джерела фінансування може сповільнити розвиток і технічну підтримку	Пошук грантів, партнерських програм, залучення спонсорів
4	Технічні ризики і кіберзагрози	Можливість несанкціонованого доступу або збоїв у роботі платформи	Використання захищених серверів, регулярне оновлення системи, резервне копіювання
5	Зниження волонтерської активності	Соціальні чи економічні зміни можуть призвести до зменшення участі користувачів	Проведення інформаційних кампаній, мотиваційних програм і співпраця з організаціями

Таблиця 4.7

Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Зростання суспільної активності та культури волонтерства	Підвищення готовності громадян брати участь у волонтерських ініціативах і цифрових платформах допомоги	Активне залучення нових користувачів через соціальні мережі, проведення інформаційних кампаній
2	Підтримка цифровізації державними	Держава стимулює створення онлайн-сервісів, що підвищують	Співпраця з державними структурами, участь у грантових та інноваційних

Таблиця 4.7 (продовження)

Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
	програмами	ефективність комунікації між громадянами	програмах
3	Швидкий розвиток веб-технологій і хмарних сервісів	Нові інструменти забезпечують просту масштабованість і стабільну роботу платформи	Використання сучасних технологій, впровадження регулярних оновлень
4	Підвищення інтересу донорів і бізнесу до соціальних проєктів	Бізнес готовий підтримувати соціальні ініціативи в рамках корпоративної відповідальності	Пошук партнерств, створення спільних акцій і залучення фінансування
5	Зростання довіри до перевірених цифрових інструментів	Користувачі охочіше взаємодіють із платформами, що мають прозорі процеси й репутацію	Формування позитивного іміджу бренду через відкритість, відгуки користувачів та стабільність роботи

Таблиця 4.8

Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	У чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Тип конкуренції — монополістична	На ринку діє кілька платформ, що пропонують схожий	Формування унікальної пропозиції — створення інтуїтивно зрозумілого інтерфейсу,

Таблиця 4.8 (продовження)

Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	У чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
	функціонал, але з різним рівнем зручності та локалізацією	покращення UX/UI, розширення аналітики заявок
2. За рівнем конкурентної боротьби — національний	Основні конкуренти функціонують в межах України, переважно у великих містах	Розширення діяльності на регіональному рівні, фокус на локальних ініціативах та волонтерських осередках
3. За галузевою ознакою — внутрішньогалузева	Усі гравці працюють у сфері волонтерських та соціальних онлайн-платформ	Підвищення ефективності внутрішніх процесів, інтеграція з державними порталами та месенджерами
4. Конкуренція за видами товарів — товарно-видова	Платформи відрізняються набором сервісів	Розширення можливостей через додаткові модулі — чат між волонтером і заявником, сповіщення, мобільна версія
5. За характером конкурентних переваг — нецінова	Конкуренція ведеться через зручність використання, швидкість реакції та рівень довіри	Зосередження на надійності, прозорості процесів і безпеці персональних даних
6. За інтенсивністю — немарочна	Брендова прив'язаність користувачів відсутня, вибір залежить від досвіду використання	Акцент на формуванні позитивної репутації та публічній відкритості результатів роботи

Таблиця 4.9

Аналіз конкуренції в галузі за М. Портером

	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
Складові аналізу	Національна Волонтерська Платформа, Українська Волонтерська Служба, Rubikus.Help	Нові IT-ініціативи, стартапи, громадські організації, які можуть створити аналогічні сервіси для волонтерів	Хмарні сервіси (AWS, Firebase), платформи авторизації (Google Auth, Auth0), поштові та комунікаційні сервіси (Gmail API, Telegram API)	Волонтери, координа-тори проєктів, благодійні фонди, заявники, державні структури	Соціальні мережі та месенджери (Facebook, Telegram, Viber), де частково координує-ться допомога
Висновки	Конкурентна боротьба помірна: кілька гравців охоплюють схожі функції, але мають різну глибину реалізації й масштаби діяльності	Вхід на ринок можливий для нових платформ — бар'єри низькі, але потрібні ресурси для маркетингу та здобуття довіри користувачів	Постачальники не домінують — існує широкий вибір технологій і постачальникі в послуг із гнучкими умовами	Користува-чі диктують умови через високі вимоги до зручності, надійності та безпеки платформи	Соцмережі та чати можуть слугувати альтернати-вою, але офіційна платформа забезпечує структуро-ваність і довіру до даних

Аналіз конкурентного середовища показав, що ринок волонтерських онлайн-платформ в Україні перебуває на стадії розвитку та характеризується помірним рівнем конкуренції. Основними гравцями є Національна Волонтерська Платформа, Українська Волонтерська Служба та Rubikus.Help. Запропонований проєкт має потенціал бути конкурентоспроможним завдяки зручності, прозорості процесів і надійному захисту даних.

Таблиця 4.10

Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Зручність та простота користування	Від інтуїтивного інтерфейсу залежить залучення користувачів і швидкість виконання волонтерських завдань.
2	Прозорість процесів і контроль дій	Система аудиту забезпечує довіру користувачів, дозволяє відстежувати історію заявок та мінімізує зловживання.
3	Надійність та безпека даних	Використання авторизації через JWT і захищених протоколів зберігає персональні дані та запобігає несанкціонованому доступу.
4	Гнучкість і масштабованість системи	Платформа побудована на сучасному технологічному стеку (Node.js, React, PostgreSQL), що забезпечує легку інтеграцію нових модулів та адаптацію під різні регіони.
5	Швидкість обробки заявок	Завдяки оптимізованому API користувачі оперативно отримують інформацію, що підвищує ефективність волонтерських дій і рівень задоволеності користувачів.

Таблиця 4.11

Порівняльний аналіз сильних та слабких сторін стартап проекту

№ п/п	Фактор конкурентоспроможності	Бали 1–20	Рейтинг товарів-конкурентів у порівнянні з веб-платформою						
			–3	–2	–1	0	1	2	3
1	Зручність інтерфейсу та простота навігації	18						+	
2	Прозорість процесів і контроль дій користувачів	19							+
3	Безпека персональних даних і стабільність роботи	18						+	
4	Гнучкість і масштабованість системи	17					+		
5	Соціальна значущість і довіра користувачів	20							+

Враховуючи результати попередніх таблиць, зроблено висновок ринкового аналізу та узагальнено його у вигляді SWOT-аналізу (табл. 4.12)

Таблиця 4.12

SWOT-аналіз стартап-проекту

Сильні сторони	Слабкі сторони
1. Прозора система взаємодії між заявниками та волонтерами. 2. Високий рівень довіри користувачів завдяки відкритості процесів. 3. Адаптивний інтерфейс і простота використання.	1. Обмежений маркетинговий бюджет на просування. 2. Залежність від стабільності серверів і підключення до інтернету. 3. Недостатня впізнаваність серед широкої аудиторії.

Таблиця 4.12 (продовження)

SWOT-аналіз стартап-проекту

Можливості	Загрози
4. Зростання волонтерського руху в Україні. 5. Можливість інтеграції з державними сервісами (наприклад, «Дія»). 6. Підтримка з боку благодійних організацій та міжнародних партнерів. 7. Розширення функціоналу для координації гуманітарної допомоги.	4. Активізація конкурентів із подібним функціоналом. 5. Кібератаки та ризики витоку даних. 6. Можливе зниження активності користувачів у післявоєнний період. 7. Обмежене фінансування на підтримку та розвиток проєкту.

Виходячи з проведеного SWOT-аналізу (табл. 4.12), розроблено кілька альтернативних варіантів ринкової поведінки стартап-проєкту. Кожна альтернатива передбачає різний набір заходів для виходу на ринок, враховуючи ресурси, можливості підтримки та часові рамки реалізації. Результати наведено у таблиці 4.13.

Таблиця 4.13

Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Базова: запуск MVP-версії платформи з основними функціями (реєстрація волонтерів, обробка заявок, панель координатора). Просування через соціальні мережі та співпрацю з локальними організаціями.	Висока	3 місяці

Таблиця 4.13 (продовження)

Альтернативи ринкового впровадження стартап-проекту

2	Оптимізована: розширення функціоналу системи, інтеграція з державними сервісами («Дія»), запуск мобільної версії. Пошук спонсорів серед міжнародних фондів.	Середня	6 місяців
3	Інноваційна: створення масштабованої екосистеми з AI-підбором волонтерів, геолокаційними модулями та аналітикою активності. Вимагає значних інвестицій та технічної підтримки.	Низька	9 місяців

Серед розглянутих альтернатив найбільш доцільною є базова альтернатива, оскільки вона потребує мінімальних ресурсів, має найвищу ймовірність реалізації та дозволяє швидко протестувати ідею на ринку, з подальшим масштабуванням проєкту.

4.4. Розроблення ринкової стратегії проекту

Визначення цільових груп потенційних споживачів є важливим етапом розроблення ринкової стратегії стартап-проєкту. Це дозволяє зрозуміти, для яких категорій користувачів продукт матиме найбільшу цінність і як найефективніше просувати платформу серед них. Результати аналізу наведено в таблиці 4.14.

Таблиця 4.14

Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит у межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Активні волонтери, що вже співпрацюють із благодійними фондами чи локальними ініціативами.	Висока	Високий попит на зручний інструмент координації	Середня	Висока
2	Представники НУО, волонтерських штабів та громадських організацій.	Висока	Високий	Середня	Середня
3	Нові користувачі, які вперше шукають можливість долучитися до волонтерства.	Середня	Потенційно високий, за умови активного просування	Низька	Висока
4	Партнерські компанії або бізнеси, які прагнуть підтримати соціальні ініціативи.	Середня	Середній	Середня	Середня
Обрано цільові групи: активні волонтери, громадські організації та нові користувачі, які прагнуть долучитися до волонтерського руху					

За результатами аналізу потенційних груп споживачів для проєкту обрано стратегію диференційованого маркетингу, оскільки платформа орієнтована на кілька сегментів — активних волонтерів, громадські організації та нових користувачів. Для кожної групи передбачено окремі інструменти комунікації та взаємодії, що забезпечує ефективніше охоплення ринку та підвищує конкурентоспроможність проєкту.

Таблиця 4.15

Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проєкту	Стратегія охоплення ринку	Ключові конкурентоспромо-жні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Стратегія зростання	Залучення нових користувачів і партнерів	Інноваційність, зручність, довіра користувачів	Стратегія диференційованого маркетингу

Таблиця 4.16

Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проєкт «першопрохід-цем» на ринку?	Чи буде компанія шукати нових споживачів або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
1	Так	Стратегія поєднання нових та існуючих користувачів	Відсутня потреба у копіюванні — акцент на власній	Стратегія наслідування лідера

Таблиця 4.16 (продовження)

Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проєкт «першопрохід-цем» на ринку?	Чи буде компанія шукати нових споживачів або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
		волонтерських сервісів	зручності, прозорості та інтеграції з організаціями	

Таблиця 4.17

Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспро-можні позиції власного стартап-проєкту	Вибір асоціацій, які мають сформувати комплексну позицію власного проєкту (три ключових)
1	Простота використання, прозорість взаємодії між волонтерами й заявниками	Стратегія диференційо-ваного маркетингу	Зручний інтерфейс, швидкий доступ до заявок, безпечне зберігання даних	Довіра, зручність, ефективність

Таблиця 4.17 (продовження)

Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспро-можні позиції власного стартап-проєкту	Вибір асоціацій, які мають сформувати комплексну позицію власного проєкту (три ключових)
2	Доступність інформації та швидке реагування на потреби користувачів	Стратегія концентрова-ного зростання	Оперативна обробка запитів, автоматичні сповіщення, можливість фільтрації заявок	Надійність, швидкість, прозорість

4.5. Розроблення маркетингової програми стартап-проекту

На основі проведеного аналізу конкурентоспроможності та визначених потреб цільової аудиторії сформовано ключові переваги концепції веб-платформи для організації волонтерської діяльності. У таблиці 4.18 наведено основні потреби користувачів, вигоди, що пропонує платформа, та її ключові переваги порівняно з існуючими рішеннями.

Таблиця 4.18

Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Необхідність швидкого зв’язку між волонтерами та тими, хто потребує допомоги	Миттєва комунікація через зручний веб-інтерфейс	Централізована система пошуку волонтерів, автоматичні сповіщення про нові заявки
2	Потреба у прозорості та безпеці волонтерської діяльності	Перевірені користувачі, контроль достовірності заявок	Механізми верифікації користувачів, журнал дій (AuditLog)
3	Необхідність ефективного розподілу ресурсів	Можливість фільтрації заявок за статусом, районом, категорією	Оптимізація процесу пошуку допомоги та зниження навантаження на організаторів

Таблиця 4.19

Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
I. Товар за задумом	Веб-платформа для організації та координації волонтерської діяльності		
II. Товар у реальному виконанні	Властивості/ характеристики	М/Нм	Вр/Тх/Тл/Е/Ор
	Функціональність Зручність Безпека	Нм Нм Нм	Тл/Ор Тх Ор
	Якість: стабільна робота веб-платформи та перевірені модулі		
	Пакування: власний веб-сайт		

Таблиця 4.19 (продовження)

Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові
	Марка: Volunteer Help UA
III. Товар із підкріпленням	До продажу: демоверсія веб-платформи
	Після продажу: техпідтримка, оновлення, навчальні матеріали
За рахунок чого потенційний товар буде захищено від копіювання: використання авторського коду, ліцензійних бібліотек і захист бази даних	

Таблиця 4.20

Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	Безкоштовні сервіси з базовими можливостя-ми	Платформи з розширеним функціоналом (приблизно 3000–6000 грн/міс для організацій)	Середній доход – від 15 000 до 25 000 грн/міс (волонтери та координатори)	Нижня межа – 0 грн (базовий доступ), верхня – 2000 грн/міс (для організацій з розширеними можливостями)

Таблиця 4.21

Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Використання соціальних мереж	Онлайн-просування, підтримка користувачів, технічне обслуговування	1 рівень (прямий збут)	Власна система збуту через офіційний сайт та

Таблиця 4.21 (продовження)

Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
				партнерів
2	Орієнтація на швидкий доступ до сервісів без посередників	Реєстрація користувачів, надання консультацій, обробка заявок	1 рівень	Власний онлайн-канал без залучення посередників

Таблиця 4.22

Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціювання	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Активне використання соціальних мереж та онлайн-платформ для волонтерства	Instagram, Facebook, Telegram, сайт платформи	Надійність, прозорість, зручність у користуванні	Сформувати довіру до сервісу та стимулювати реєстрацію	«Єднаймося, щоб допомагати разом!»

Таблиця 4.22 (продовження)

Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціювання	Завдання рекламного повідомлення	Концепція рекламного звернення
2	Орієнтація на співпрацю та обмін досвідом між волонтерами	Онлайн-форуми, спільноти, e-mail-розсилки	Соціальна цінність і підтримка спільноти	Підвищити впізнаваність бренду та залучити нових партнерів	«Volunteer Help UA — твоя сила у спільних добрих справах!»

Висновок до розділу 4

У результаті проведеного аналізу встановлено, що проєкт має реальні передумови для успішної ринкової комерціалізації. Аналіз попиту свідчить про зростаючу потребу в цифрових інструментах для координації волонтерських ініціатив, пошуку проєктів та обміну ресурсами між організаціями. Динаміка розвитку ринку IT-рішень соціального спрямування демонструє стабільне зростання, що робить даний сегмент привабливим для інноваційних стартапів. Водночас проєкт має соціальну складову, що підвищує його привабливість серед користувачів і партнерських організацій.

З огляду на результати дослідження визначено, що веб-платформа має сприятливі перспективи впровадження. Цільові групи: активні волонтери, громадські організації та користувачі, які лише починають волонтерську діяльність — характеризуються високим рівнем залучення, готовністю до цифрової взаємодії та потребою в зручних засобах комунікації. Бар'єри входження на ринок є незначними, а рівень конкуренції помірний, оскільки більшість наявних платформ мають обмежену функціональність або фокусуються лише на окремих напрямах діяльності.

Для ринкової реалізації проєкту доцільно обрати альтернативу зростання із застосуванням стратегії диференційованого маркетингу, що передбачає поетапне розширення цільових сегментів і впровадження нових функцій — таких як система рейтингів волонтерів, інтеграція з месенджерами та автоматизоване формування звітності для організацій. Обрана стратегія дає змогу зміцнити конкурентні позиції проєкту, забезпечити його гнучкість і масштабованість.

Подальша імплементація проєкту є доцільною та перспективною. Веб-платформа має потенціал стати важливим елементом національної волонтерської екосистеми, сприяти розвитку громадянського суспільства, посиленню комунікації між учасниками волонтерського руху та ефективнішому використанню людських і інформаційних ресурсів.

ВИСНОВКИ

У ході виконання магістерської роботи було успішно створено повноцінну веб-платформу для організації волонтерської діяльності, яка забезпечує ефективну взаємодію між громадянами, які цього потребують, та волонтерами. Система розроблена відповідно до сучасних стандартів безпеки, зручності використання, масштабованості та надійності, що підтверджує її високий рівень практичної придатності.

Створена платформа демонструє комплексне цифрове рішення для вирішення актуальної соціальної проблеми – потреби в ефективних інструментах координації волонтерських рухів. Система оптимізує процеси подання заявок, пошуку волонтерів, моніторингу статусів завдань та моніторингу дій усіх користувачів. Реалізований функціонал гарантує прозорість, ефективність та зручність використання, що є значною перевагою порівняно з існуючими аналогами.

В результаті розробки було створено гнучке, стабільне та багатофункціональне рішення, яке поєднує технічну ефективність із соціальною корисністю. Система характеризується оптимально спроєктованою архітектурою та ергономічним інтерфейсом, що гарантує злагоджену і безвідмовну комунікацію між її основними модулями. Платформа демонструє стабільну роботу, швидкий обмін даними та можливість подальшого масштабування без необхідності радикальних змін коду чи інфраструктури.

Практичне значення розробленої системи полягає в тому, що вона може бути застосована як базова модель для впровадження у волонтерських організаціях, благодійних фондах або місцевих ініціативах. Вона сприяє ефективній комунікації, забезпечує облік активності користувачів та підтримує ключові принципи прозорості та довіри у процесі надання допомоги.

У межах магістерської роботи досягнуто поставлену мету — створено сучасну, стабільну та соціально важливу систему, яка поєднує технічну досконалість з відчутним впливом на розвиток волонтерського руху в Україні.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *Трофименко А. В.* Волонтерський рух в Україні в умовах повномасштабної війни: основні тенденції розвитку // Вісник Міжнародного університету. — 2024.
2. *Маційовський Ю.* Волонтерство та псевдоволонтерство в Україні: новий погляд крізь призму неформальних інститутів // Інститут політичних і етнонаціональних досліджень НАН України. — 2024.
3. *Лобуренко А.* Волонтерство в Україні: виклики та перспективи розвитку // Volunteer Country. — 2024.
4. *Панченко А. О.* Волонтерська діяльність як одна із форм благодійної діяльності в Україні // Вісник юридичних наук УжНУ. — 2024.
5. *Климчук В. Р.* Особливості волонтерської діяльності в умовах війни // Уманський державний педагогічний університет. — 2024.
6. *Маційовський Ю.* Волонтерство та псевдоволонтерство в Україні: новий погляд крізь призму неформальних інститутів // ІПіЕНД НАН України. — 2024.
7. *Проценко О. О.* Волонтерство в сучасній Україні: фактори актуалізації // Science and Education a New Dimension. — 2021.
8. OpenXcell. Web Application Architecture // OpenXcell Blog. — 2023. — URL: https://www.openxcell.com/blog/web-application-architecture/
9. *Fowler M.* Patterns of Enterprise Application Architecture. — Addison-Wesley, 2002.
10. *Tanenbaum A.* Distributed Systems: Principles and Paradigms. — Prentice Hall, 2007.
11. Volyn.com.ua. Khostynh saitu: vzaiemodiia klient–server // Volyn News. — 2023. — URL: https://www.volyn.com.ua/news/256294-khostynh-saitu-vzaiemodiia-kliient-server
12. *Richards M.* Software Architecture Patterns. — O'Reilly Media, 2015.
13. Radixweb. Web Application Architecture Guide // Radixweb Blog. — 2024. — URL: https://radixweb.com/blog/web-application-architecture-guide
14. *Newman S.* Building Microservices. — O'Reilly Media, 2021.
15. *Kreps J.* The Log: What Every Software Engineer Should Know About Real-Time Data's Unifying Abstraction // LinkedIn Engineering Blog. — 2013.
16. *Stefanov S.* React Up & Running: Building Web Applications. — O'Reilly Media, 2016.
17. *You Y.* The Majesty of Vue.js. — Leanpub, 2018.
18. *Shama S.* Angular: Up and Running. — O'Reilly Media, 2018.
19. *Kanti M.* Node.js Web Development. — Packt Publishing, 2020.
20. *Melin A.* Django for Professionals: Production Websites with Python & Django. — Independently published, 2021.

21. *Agrawanes T.* Building APIs with FastAPI and Python. — Leanpub, 2022.

22. *Kim H.*, *Lee J.*, *Park S.* Relational vs. NoSQL Databases in Web Applications: A Comparative Study // Journal of Information Systems Research. — 2022.

23. *Halfond W.*, *Viegas J.* Securing Web Applications: Database Security Principles and Practices // ACM Press. — 2021.

ДОДАТОК А

Веб-платформа для організації волонтерської діяльності в Україні

Лістинг програми

Аркушів

Київ – 2025

```
-- Users table
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    email TEXT UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,
    first_name TEXT NOT NULL,
    last_name TEXT NOT NULL,
    phone TEXT,
    role TEXT NOT NULL DEFAULT 'requester' CHECK (role IN ('requester', 'volunteer',
'admin')),
    is_active BOOLEAN DEFAULT 1,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

-- Categories table
CREATE TABLE IF NOT EXISTS categories (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL UNIQUE,
    description TEXT,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

-- Requests table
CREATE TABLE IF NOT EXISTS requests (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    requester_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    title TEXT NOT NULL,
    description TEXT NOT NULL,
    category_id INTEGER REFERENCES categories(id),
    location TEXT,
    urgency TEXT DEFAULT 'medium' CHECK (urgency IN ('low', 'medium', 'high')),
    status TEXT DEFAULT 'created' CHECK (status IN ('created', 'accepted',
'in_progress', 'completed', 'cancelled')),
    contact_phone TEXT,
    contact_telegram TEXT,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    completed_at DATETIME
);

-- Assignments table (volunteer responses to requests)
CREATE TABLE IF NOT EXISTS assignments (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    request_id INTEGER NOT NULL REFERENCES requests(id) ON DELETE CASCADE,
    volunteer_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    status TEXT DEFAULT 'pending' CHECK (status IN ('pending', 'accepted', 'rejected',
'completed')),
    message TEXT,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(request_id, volunteer_id)
);

-- Audit log table
CREATE TABLE IF NOT EXISTS audit_log (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER REFERENCES users(id) ON DELETE SET NULL,
    action TEXT NOT NULL,
    table_name TEXT,
```

```
    record_id INTEGER,
    old_values TEXT,
    new_values TEXT,
    ip_address TEXT,
    user_agent TEXT,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

-- Insert default categories
INSERT OR IGNORE INTO categories (name, description) VALUES
('Medicine', 'Medical supplies and prescription medications'),
('Food', 'Food assistance and meal delivery'),
('Transport', 'Transportation and mobility assistance'),
('Shelter', 'Housing and shelter assistance'),
('Other', 'Other types of assistance');

-- Create indexes for better performance
CREATE INDEX IF NOT EXISTS idx_requests_status ON requests(status);
CREATE INDEX IF NOT EXISTS idx_requests_requester ON requests(requester_id);
CREATE INDEX IF NOT EXISTS idx_assignments_volunteer ON assignments(volunteer_id);
CREATE INDEX IF NOT EXISTS idx_assignments_request ON assignments(request_id);
CREATE INDEX IF NOT EXISTS idx_users_role ON users(role);
CREATE INDEX IF NOT EXISTS idx_audit_log_user ON audit_log(user_id);

const jwt = require('jsonwebtoken');
const pool = require('../config/database');

const authenticateToken = async (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];

  if (!token) {
    return res.status(401).json({ error: 'Необхідний токен доступу' });
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET ||
'fallback_secret_key_12345');

    const result = await pool.query(
      'SELECT id, email, role, is_active FROM users WHERE id = ?',
      [decoded.userId]
    );

    const user = result.rows[0] || result[0];
    if (!user || !user.is_active) {
      return res.status(401).json({ error: 'Невірний токен або користувач не
знайдений' });
    }

    req.user = user;
    next();
  } catch (error) {
    return res.status(403).json({ error: 'Невірний токен' });
  }
};

const requireRole = (roles) => {
  return (req, res, next) => {
    if (!req.user) {
      return res.status(401).json({ error: 'Authentication required' });
```

```
    }

    if (!roles.includes(req.user.role)) {
      return res.status(403).json({ error: 'Insufficient permissions' });
    }

    next();
  };
};

const requireAdmin = requireRole(['admin']);
const requireVolunteer = requireRole(['volunteer', 'admin']);
const requireRequester = requireRole(['requester', 'admin']);

module.exports = {
  authenticateToken,
  requireRole,
  requireAdmin,
  requireVolunteer,
  requireRequester
};

const express = require('express');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const pool = require('../config/database');
const { validateRegistration, validateLogin } = require('../utils/validation');
const { authenticateToken } = require('../middleware/auth');

const router = express.Router();

router.post('/register', validateRegistration, async (req, res) => {
  try {
    const { email, password, firstName, lastName, phone, role = 'requester' } =
req.body;
    console.log('Registration data received:', { email, firstName, lastName, role });

    const existingUser = await pool.query('SELECT id FROM users WHERE email = ?',
[email]);
    if (existingUser.rows.length > 0) {
      return res.status(400).json({ error: 'Користувач з таким email вже існує' });
    }

    const saltRounds = 10;
    const passwordHash = await bcrypt.hash(password, saltRounds);

    const result = await pool.query(
      'INSERT INTO users (email, password_hash, first_name, last_name, phone, role)
VALUES (?, ?, ?, ?, ?, ?)',
      [email, passwordHash, firstName, lastName, phone, role]
    );

    console.log('Insert result:', result);

    const userResult = await pool.query(
      'SELECT id, email, first_name, last_name, role, created_at FROM users WHERE email
= ?',
      [email]
    );

    const user = userResult.rows[0] || userResult[0];
```

```
    const token = jwt.sign(
      { userId: user.id, email: user.email, role: user.role },
      process.env.JWT_SECRET || 'fallback_secret_key_12345',
      { expiresIn: '24h' }
    );

    console.log('Registered user:', {
      id: user.id,
      email: user.email,
      firstName: user.first_name,
      lastName: user.last_name,
      role: user.role
    });

    try {
      console.log('Logging user registration to audit log:', {
        user_id: user.id,
        action: 'create',
        table_name: 'users',
        record_id: user.id,
        description: `Зареєстровано нового користувача: ${firstName} ${lastName}
(${role})`
      });

      await pool.query(`
        INSERT INTO audit_log (user_id, action, table_name, record_id, new_values)
        VALUES (?, ?, ?, ?, ?)
      `, [user.id, 'create', 'users', user.id, `Зареєстровано нового користувача:
${firstName} ${lastName} (${role})`]);

      console.log('Audit log entry created successfully');
    } catch (auditError) {
      console.error('Error creating audit log entry:', auditError);
    }

    res.status(201).json({
      message: 'Користувач успішно зареєстрований',
      user: {
        id: user.id,
        email: user.email,
        firstName: user.first_name,
        lastName: user.last_name,
        role: user.role,
        createdAt: user.created_at
      },
      token
    });
  } catch (error) {
    console.error('Помилка реєстрації:', error);
    res.status(500).json({ error: 'Внутрішня помилка сервера' });
  }
});

router.post('/login', validateLogin, async (req, res) => {
  try {
    const { email, password } = req.body;

    const result = await pool.query(
      'SELECT id, email, password_hash, first_name, last_name, role, is_active FROM
users WHERE email = ?',
```

```
      [email]
    );

    if (result.rows.length === 0) {
      return res.status(401).json({ error: 'Невірний email або пароль' });
    }

    const user = result.rows[0];

    if (!user.is_active) {
      return res.status(401).json({ error: 'Обліковий запис деактивовано' });
    }

    const isValidPassword = await bcrypt.compare(password, user.password_hash);
    if (!isValidPassword) {
      return res.status(401).json({ error: 'Невірний email або пароль' });
    }

    const token = jwt.sign(
      { userId: user.id, email: user.email, role: user.role },
      process.env.JWT_SECRET || 'fallback_secret_key_12345',
      { expiresIn: '24h' }
    );

    res.json({
      message: 'Вхід успішний',
      user: {
        id: user.id,
        email: user.email,
        firstName: user.first_name,
        lastName: user.last_name,
        role: user.role
      },
      token
    });
  } catch (error) {
    console.error('Помилка входу:', error);
    res.status(500).json({ error: 'Внутрішня помилка сервера' });
  }
});

router.get('/profile', authenticateToken, async (req, res) => {
  try {
    const result = await pool.query(
      'SELECT id, email, first_name, last_name, phone, role, created_at FROM users
WHERE id = ?',
      [req.user.id]
    );

    if (result.rows.length === 0) {
      return res.status(404).json({ error: 'User not found' });
    }

    const user = result.rows[0];
    res.json({
      id: user.id,
      email: user.email,
      firstName: user.first_name,
      lastName: user.last_name,
      phone: user.phone,
      role: user.role,
```

```
      createdAt: user.created_at
    });
  } catch (error) {
    console.error('Помилка отримання профілю:', error);
    res.status(500).json({ error: 'Внутрішня помилка сервера' });
  }
});

module.exports = router;

import React, { createContext, useContext, useState, useEffect, ReactNode } from
'react';
import { authAPI } from '../services/api';

interface User {
  id: number;
  email: string;
  firstName: string;
  lastName: string;
  role: 'requester' | 'volunteer' | 'admin';
}

interface AuthContextType {
  user: User | null;
  token: string | null;
  login: (email: string, password: string) => Promise<void>;
  register: (userData: any) => Promise<void>;
  logout: () => void;
  loading: boolean;
}

const AuthContext = createContext<AuthContextType | undefined>(undefined);

export const useAuth = () => {
  const context = useContext(AuthContext);
  if (context === undefined) {
    throw new Error('useAuth must be used within an AuthProvider');
  }
  return context;
};

interface AuthProviderProps {
  children: ReactNode;
}

export const AuthProvider: React.FC<AuthProviderProps> = ({ children }) => {
  const [user, setUser] = useState<User | null>(null);
  const [token, setToken] = useState<string | null>(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const storedToken = localStorage.getItem('token');
    const storedUser = localStorage.getItem('user');

    if (storedToken && storedUser) {
      setToken(storedToken);
      const userData = JSON.parse(storedUser);
      setUser(userData);

          if (!userData.role || !userData.firstName) {
        fetchUserProfile();
```

```
    }
  }
  setLoading(false);
}, []);

const fetchUserProfile = async () => {
  try {
    const response = await authAPI.getProfile();
    const userData = response.data;
    setUser(userData);
    localStorage.setItem('user', JSON.stringify(userData));
  } catch (error) {
    console.error('Failed to fetch user profile:', error);
  }
};

const login = async (email: string, password: string) => {
  try {
    const response = await authAPI.login({ email, password });
    const { user: userData, token: userToken } = response.data;

    setUser(userData);
    setToken(userToken);
    localStorage.setItem('token', userToken);
    localStorage.setItem('user', JSON.stringify(userData));
  } catch (error) {
    throw error;
  }
};

const register = async (userData: any) => {
  try {
    const response = await authAPI.register(userData);
    const { user: newUser, token: userToken } = response.data;

    setUser(newUser);
    setToken(userToken);
    localStorage.setItem('token', userToken);
    localStorage.setItem('user', JSON.stringify(newUser));
  } catch (error) {
    console.error('Registration error:', error);
    throw error;
  }
};

const logout = () => {
  setUser(null);
  setToken(null);
  localStorage.removeItem('token');
  localStorage.removeItem('user');
};

const value = {
  user,
  token,
  login,
  register,
  logout,
  loading,
};
```

```
  return <AuthContext.Provider value={value}>{children}</AuthContext.Provider>;

const express = require('express');
const pool = require('../config/database');
const { validateRequest } = require('../utils/validation');
const { authenticateToken, requireRequester, requireAdmin } =
require('../middleware/auth');

const router = express.Router();

router.get('/', async (req, res) => {
  try {
    const { status, category, urgency, page = 1, limit = 10 } = req.query;
    const offset = (page - 1) * limit;

    let query = `
      SELECT r.id, r.title, r.description, r.category_id, r.location, r.urgency,
r.status, r.created_at, r.updated_at,
             r.requester_id, u.first_name, u.last_name, u.email as requester_email,
c.name as category_name,
             (u.first_name || ' ' || u.last_name) as requester_name
      FROM requests r
      JOIN users u ON r.requester_id = u.id
      LEFT JOIN categories c ON r.category_id = c.id
      WHERE 1=1
    `;
    const queryParams = [];
    let paramCount = 0;

    if (status) {
      query += ` AND r.status = ?`;
      queryParams.push(status);
    }
    if (category) {
      query += ` AND r.category_id = ?`;
      queryParams.push(category);
    }
    if (urgency) {
      query += ` AND r.urgency = ?`;
      queryParams.push(urgency);
    }

    query += ` AND r.status = 'created'`;

    query += ` ORDER BY r.created_at DESC LIMIT $${++paramCount} OFFSET
$${++paramCount}`;
    queryParams.push(limit, offset);

    console.log('Public requests query:', query);
    console.log('Public requests params:', queryParams);
    const result = await pool.query(query, queryParams);
    console.log('Public requests result:', result.rows.map(r => ({ id: r.id, title:
r.title, status: r.status })));

    let countQuery = 'SELECT COUNT(*) FROM requests r WHERE 1=1';
    const countParams = [];
    let countParamCount = 0;

    if (status) {
      countQuery += ` AND r.status = ?`;
      countParams.push(status);
```

```
    }
    if (category) {
      countQuery += ` AND r.category_id = ?`;
      countParams.push(category);
    }
    if (urgency) {
      countQuery += ` AND r.urgency = ?`;
      countParams.push(urgency);
    }

    countQuery += ` AND r.status = 'created'`;

    const countResult = await pool.query(countQuery, countParams);
    const total = parseInt(countResult.rows[0].count);

    res.json({
      requests: result.rows,
      pagination: {
        page: parseInt(page),
        limit: parseInt(limit),
        total,
        pages: Math.ceil(total / limit)
      }
    });
  } catch (error) {
    console.error('Помилка отримання заявок:', error);
    res.status(500).json({ error: 'Internal server error' });
  }
});

router.get('/my', authenticateToken, async (req, res) => {
  try {
    const { status, category, urgency, page = 1, limit = 10 } = req.query;
    const offset = (page - 1) * limit;

    console.log('Getting my requests for user:', req.user.id, 'role:', req.user.role,
'filters:', { status, category, urgency });
    console.log('Full query params:', { status, category, urgency, page, limit });

    let query = `
      SELECT r.*, u.first_name, u.last_name, u.email as requester_email, u.phone as
requester_phone, c.name as category_name,
             (u.first_name || ' ' || u.last_name) as requester_name
      FROM requests r
      JOIN users u ON r.requester_id = u.id
      LEFT JOIN categories c ON r.category_id = c.id
      WHERE 1=1
    `;
    const queryParams = [];

    if (status) {
      query += ` AND r.status = ?`;
      queryParams.push(status);
    }
    if (category) {
      query += ` AND r.category_id = ?`;
      queryParams.push(category);
    }
    if (urgency) {
      query += ` AND r.urgency = ?`;
      queryParams.push(urgency);
```

```
    }

    if (req.user.role === 'volunteer') {
      query += ` AND r.status = 'created'`;
    } else if (req.user.role === 'requester') {
      query += ` AND r.requester_id = ?`;
      queryParams.push(req.user.id);
    }

    query += ` ORDER BY r.created_at DESC LIMIT ? OFFSET ?`;
    queryParams.push(limit, offset);

    console.log('Final SQL query:', query);
    console.log('Query parameters:', queryParams);

    const result = await pool.query(query, queryParams);

    console.log('Query result:', result.rows.length, 'requests found');
    console.log('Requests:', result.rows.map(r => ({ id: r.id, title: r.title, status:
r.status })));

    let countQuery = 'SELECT COUNT(*) FROM requests r WHERE 1=1';
    const countParams = [];

    if (status) {
      countQuery += ` AND r.status = ?`;
      countParams.push(status);
    }
    if (category) {
      countQuery += ` AND r.category_id = ?`;
      countParams.push(category);
    }
    if (urgency) {
      countQuery += ` AND r.urgency = ?`;
      countParams.push(urgency);
    }

    if (req.user.role === 'volunteer') {
      countQuery += ` AND r.status = 'created'`;
    } else if (req.user.role === 'requester') {
      countQuery += ` AND r.requester_id = ?`;
      countParams.push(req.user.id);
    }

    const countResult = await pool.query(countQuery, countParams);
    const total = parseInt(countResult.rows[0].count);

    res.json({
      requests: result.rows,
      pagination: {
        page: parseInt(page),
        limit: parseInt(limit),
        total,
        pages: Math.ceil(total / limit)
      }
    });
  } catch (error) {
    console.error('Помилка отримання моїх заявок:', error);
    res.status(500).json({ error: 'Internal server error' });
  }
});
```

```
router.get('/:id', authenticateToken, async (req, res) => {
  try {
    const { id } = req.params;

    const result = await pool.query(`
      SELECT r.*, u.first_name, u.last_name, u.email as requester_email, c.name as
category_name
      FROM requests r
      JOIN users u ON r.requester_id = u.id
      LEFT JOIN categories c ON r.category_id = c.id
      WHERE r.id = ?
    `, [id]);

    if (result.rows.length === 0) {
      return res.status(404).json({ error: 'Request not found' });
    }

    const request = result.rows[0];

    const isRequester = String(request.requester_id) === String(req.user.id);
    const isAdmin = req.user.role === 'admin';
    let isAcceptedVolunteer = false;
    if (req.user.role === 'volunteer') {
      const a = await pool.query('SELECT 1 FROM assignments WHERE request_id = ? AND
volunteer_id = ? AND status = ?', [id, req.user.id, 'accepted']);
      isAcceptedVolunteer = (a.rows && a.rows.length > 0) || (a[0] ? true : false);
    }

    if (!isRequester && !isAdmin && !isAcceptedVolunteer) {
      delete request.contact_phone;
      delete request.contact_telegram;
    }

    res.json(request);
  } catch (error) {
    console.error('Помилка отримання заявки:', error);
    res.status(500).json({ error: 'Internal server error' });
  }
});

router.post('/', authenticateToken, requireRequester, validateRequest, async (req, res)
=> {
  try {
    const { title, description, categoryId, location, urgency = 'medium', contactPhone,
contactTelegram } = req.body;

    const result = await pool.query(`
      INSERT INTO requests (requester_id, title, description, category_id, location,
urgency, contact_phone, contact_telegram)
      VALUES (?, ?, ?, ?, ?, ?, ?, ?)
    `, [req.user.id, title, description, categoryId, location, urgency, contactPhone,
contactTelegram]);

    await pool.query(`
      INSERT INTO audit_log (user_id, action, table_name, record_id, new_values)
      VALUES (?, ?, ?, ?, ?)
    `, [req.user.id, 'create', 'requests', result.id, `Створено нову заявку:
${title}`]);

    const createdRequest = await pool.query(
```

```
      'SELECT * FROM requests WHERE id = ?',
      [result.id]
    );

    res.status(201).json(createdRequest.rows[0]);
  } catch (error) {
    console.error('Помилка створення заявки:', error);
    res.status(500).json({ error: 'Internal server error' });
  }
});

router.put('/:id', authenticateToken, validateRequest, async (req, res) => {
  try {
    const { id } = req.params;
    const { title, description, categoryId, location, urgency } = req.body;

    const existingRequest = await pool.query(
      'SELECT requester_id, status FROM requests WHERE id = ?',
      [id]
    );

    if (existingRequest.rows.length === 0) {
      return res.status(404).json({ error: 'Request not found' });
    }

    const request = existingRequest.rows[0];

    if (req.user.role === 'requester' && request.requester_id !== req.user.id) {
      return res.status(403).json({ error: 'Access denied' });
    }

    if (request.status !== 'created' && req.user.role === 'requester') {
      return res.status(400).json({ error: 'Cannot edit request that has been
accepted' });
    }

    await pool.query(`
      UPDATE requests
      SET title = ?, description = ?, category_id = ?, location = ?, urgency = ?,
updated_at = datetime('now')
      WHERE id = ?
    `, [title, description, categoryId, location, urgency, id]);

    const updatedRequest = await pool.query(
      'SELECT * FROM requests WHERE id = ?',
      [id]
    );

    res.json(updatedRequest.rows[0]);
  } catch (error) {
    console.error('Помилка оновлення заявки:', error);
    res.status(500).json({ error: 'Internal server error' });
  }
});

router.delete('/:id', authenticateToken, async (req, res) => {
  try {
    const { id } = req.params;

    const existingRequest = await pool.query(
      'SELECT requester_id, status FROM requests WHERE id = ?',
```

```
      [id]
    );

    if (existingRequest.rows.length === 0) {
      return res.status(404).json({ error: 'Request not found' });
    }

    const request = existingRequest.rows[0];

    if (req.user.role === 'requester' && request.requester_id !== req.user.id) {
      return res.status(403).json({ error: 'Access denied' });
    }

    if (req.user.role === 'requester') {
      console.log('Cancelling request:', id, 'for user:', req.user.id);

      await pool.query(
        `UPDATE requests SET status = 'cancelled', updated_at = datetime('now') WHERE
id = ? AND requester_id = ?`,
        [id, req.user.id]
      );

      await pool.query(`
        INSERT INTO audit_log (user_id, action, table_name, record_id, new_values)
        VALUES (?, ?, ?, ?, ?)
      `, [req.user.id, 'update', 'requests', id, 'Скасовано заявку']);

      const updatedRequest = await pool.query(
        'SELECT * FROM requests WHERE id = ?',
        [id]
      );

      console.log('Updated request:', updatedRequest.rows[0]);

      if (updatedRequest.rows.length === 0) {
        return res.status(403).json({ error: 'Insufficient permissions' });
      }
      return res.json({ message: 'Request cancelled', request:
updatedRequest.rows[0] });
    }

    await pool.query('DELETE FROM requests WHERE id = ?', [id]);

    await pool.query(`
      INSERT INTO audit_log (user_id, action, table_name, record_id, new_values)
      VALUES (?, ?, ?, ?, ?)
    `, [req.user.id, 'delete', 'requests', id, 'Видалено заявку']);

    res.json({ message: 'Request deleted' });
  } catch (error) {
    console.error('Помилка видалення заявки:', error);
    res.status(500).json({ error: 'Internal server error' });
  }
});

router.patch('/:id/status', authenticateToken, async (req, res) => {
  try {
    const { id } = req.params;
    const { status } = req.body;
```

```
    if (!['created', 'accepted', 'in_progress', 'completed',
'cancelled'].includes(status)) {
      return res.status(400).json({ error: 'Invalid status' });
    }

    if (req.user.role === 'admin') {
      await pool.query(
        'UPDATE requests SET status = ?, updated_at = datetime(\'now\'), completed_at =
CASE WHEN ? = ? THEN datetime(\'now\') ELSE completed_at END WHERE id = ?',
        [status, status, 'completed', id]
      );
    } else {
      await pool.query(
        'UPDATE requests SET status = ?, updated_at = datetime(\'now\'), completed_at =
CASE WHEN ? = ? THEN datetime(\'now\') ELSE completed_at END WHERE id = ? AND
requester_id = ?',
        [status, status, 'completed', id, req.user.id]
      );
    }

    if (status === 'completed') {
      await pool.query(
        'UPDATE assignments SET status = ?, updated_at = datetime(\'now\') WHERE
request_id = ? AND (status = \'accepted\' OR status = \'pending\')',
        ['completed', id]
      );
    }

    await pool.query(`
      INSERT INTO audit_log (user_id, action, table_name, record_id, new_values)
      VALUES (?, ?, ?, ?, ?)
    `, [req.user.id, 'update', 'requests', id, `Змінено статус заявки на ${status ===
'created' ? 'створено' : status === 'accepted' ? 'прийнято' : status === 'in_progress' ?
'в процесі' : status === 'completed' ? 'виконано' : 'скасовано'}`]);

    const updatedRequest = await pool.query(
      'SELECT * FROM requests WHERE id = ?',
      [id]
    );

    const updatedRow = updatedRequest.rows[0];
    if (!updatedRow) {
      return res.status(404).json({ error: 'Request not found' });
    }

    res.json(updatedRow);
  } catch (error) {
    console.error('Помилка оновлення статусу заявки:', error);
    res.status(500).json({ error: 'Internal server error' });
  }
});

module.exports = router;

const express = require('express');
const pool = require('../config/database');
const { validateAssignment } = require('../utils/validation');
const { authenticateToken, requireVolunteer } = require('../middleware/auth');

const router = express.Router();
```

```
router.get('/my-assignments', authenticateToken, requireVolunteer, async (req, res) =>
{
  try {
    console.log(`Getting assignments for volunteer ${req.user.id}`);
    console.log('User object:', req.user);

    const result = await pool.query(`
      SELECT a.*, r.title, r.description, r.status as request_status, r.urgency,
r.location,
             r.contact_phone, r.contact_telegram,
             u.first_name, u.last_name, u.email as requester_email
      FROM assignments a
      JOIN requests r ON a.request_id = r.id
      JOIN users u ON r.requester_id = u.id
      WHERE a.volunteer_id = ?
      ORDER BY a.created_at DESC
    `, [req.user.id]);

    console.log(`Found ${result.rows.length} assignments for volunteer ${req.user.id}:`,
result.rows);
    res.json(result.rows);
  } catch (error) {
    console.error('Помилка отримання завдань:', error);
    console.error('Деталі помилки:', error.message);
    console.error('Стек помилки:', error.stack);
    res.status(500).json({ error: 'Внутрішня помилка сервера' });
  }
});

router.post('/respond', authenticateToken, requireVolunteer, validateAssignment, async
(req, res) => {
  try {
    const { requestId, message } = req.body;

    const requestResult = await pool.query(
      'SELECT id, status, requester_id FROM requests WHERE id = ?',
      [requestId]
    );

    if (requestResult.rows.length === 0) {
      return res.status(404).json({ error: 'Request not found' });
    }

    const request = requestResult.rows[0];

    if (request.status !== 'created') {
      return res.status(400).json({ error: 'Request is no longer available' });
    }

    const existingAssignment = await pool.query(
      'SELECT id FROM assignments WHERE request_id = ? AND volunteer_id = ?',
      [requestId, req.user.id]
    );

    if (existingAssignment.rows.length > 0) {
      return res.status(400).json({ error: 'You have already responded to this
request' });
    }

    const result = await pool.query(`
      INSERT INTO assignments (request_id, volunteer_id, message, status)
```

```
      VALUES (?, ?, ?, 'pending')
    `, [requestId, req.user.id, message]);

    const createdAssignment = await pool.query(
      'SELECT * FROM assignments WHERE id = ?',
      [result.id]
    );

    res.status(201).json(createdAssignment.rows[0]);
  } catch (error) {
    console.error('Помилка відгуку на заявку:', error);
    res.status(500).json({ error: 'Внутрішня помилка сервера' });
  }
});

router.patch('/:id/status', authenticateToken, async (req, res) => {
  try {
    const { id } = req.params;
    const { status } = req.body;

    if (!['pending', 'accepted', 'rejected', 'completed'].includes(status)) {
      return res.status(400).json({ error: 'Invalid status' });
    }

    const assignmentResult = await pool.query(`
      SELECT a.*, r.requester_id, r.status as request_status
      FROM assignments a
      JOIN requests r ON a.request_id = r.id
      WHERE a.id = ?
    `, [id]);

    if (assignmentResult.rows.length === 0) {
      return res.status(404).json({ error: 'Assignment not found' });
    }

    const assignment = assignmentResult.rows[0];

    const canUpdate = req.user.role === 'admin' ||
      (req.user.role === 'volunteer' && assignment.volunteer_id === req.user.id) ||
      (req.user.role === 'requester' && assignment.requester_id === req.user.id);

    if (!canUpdate) {
      return res.status(403).json({ error: 'Access denied' });
    }

    const result = await pool.query(`
      UPDATE assignments
      SET status = ?, updated_at = datetime('now')
      WHERE id = ?
    `, [status, id]);

    const updatedAssignment = await pool.query(
      'SELECT * FROM assignments WHERE id = ?',
      [id]
    );

    if (status === 'accepted') {
      await pool.query(
        'UPDATE requests SET status = ?, updated_at = datetime(\'now\') WHERE id = ?',
        ['accepted', assignment.request_id]
      );
```

```
    }

    if (status === 'rejected') {
      console.log(`Assignment ${id} rejected, updating request ${assignment.request_id}
status to created`);
      const updateResult = await pool.query(
        'UPDATE requests SET status = ?, updated_at = datetime(\'now\') WHERE id = ?',
        ['created', assignment.request_id]
      );
      console.log(`Request ${assignment.request_id} status update result:`,
updateResult);

      const verifyResult = await pool.query(
        'SELECT id, status FROM requests WHERE id = ?',
        [assignment.request_id]
      );
      console.log(`Request ${assignment.request_id} current status:`,
verifyResult.rows[0]);
    }

    res.json(updatedAssignment.rows[0]);
  } catch (error) {
    console.error('Помилка оновлення статусу завдання:', error);
    res.status(500).json({ error: 'Внутрішня помилка сервера' });
  }
});

router.get('/request/:requestId', authenticateToken, async (req, res) => {
  try {
    const { requestId } = req.params;
    console.log(`Getting assignments for request ${requestId} by user ${req.user.id}
(role: ${req.user.role})`);

    const requestResult = await pool.query(
      'SELECT requester_id FROM requests WHERE id = ?',
      [requestId]
    );

    if (requestResult.rows.length === 0) {
      console.log(`Request ${requestId} not found`);
      return res.status(404).json({ error: 'Request not found' });
    }

    const request = requestResult.rows[0];
    console.log(`Request ${requestId} belongs to requester ${request.requester_id}`);

    if (req.user.role === 'requester' && parseInt(request.requester_id) !==
parseInt(req.user.id)) {
      console.log(`Access denied: user ${req.user.id} is not the requester of request
${requestId}`);
      return res.status(403).json({ error: 'Access denied' });
    }

    const result = await pool.query(`
      SELECT a.*, u.first_name, u.last_name, u.email as volunteer_email
      FROM assignments a
      JOIN users u ON a.volunteer_id = u.id
      WHERE a.request_id = ?
      ORDER BY a.created_at DESC
    `, [requestId]);
```

```
    console.log(`Found ${result.rows.length} assignments for request ${requestId}:`,
result.rows);
    res.json(result.rows);
  } catch (error) {
    console.error('Помилка отримання завдань заявки:', error);
    res.status(500).json({ error: 'Внутрішня помилка сервера' });
  }
});

module.exports = router;
```