МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА

Навчально-науковий інститут
«Інститут інформаційних технологій в економіці»

Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА
«Інформаційні управляючі системи і технології»

галузь знань	12 Інформаційні технології
Спеціальність	122 Комп'ютерні науки

Форма навчання: денна (очна)

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему: «Розроблення веб-платформи для підтримки біженців»

здобувача **Безпалько Юліани Сергіївни**

Науковий керівник: д.е.н., професор Мозгаллі О. П.

_____
(підпис)

Робота допущена до захисту перед екзаменаційною комісією з атестації здобувачів вищої освіти (ЕК)

Завідувач кафедри: к.е.н., доцент Тішков Б. О.

_____
(підпис)

Київ 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА
Навчально-науковий інститут «Інститут інформаційних технологій в економіці»
Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА
«Інформаційні управляючі системи і технології»

галузь знань 12 Інформаційні технології
Спеціальність 122 Комп'ютерні науки

ПОГОДЖЕНО:
Керівник проєктної групи (гарант) освітньо-професійної програми

_____Устенко С.В.
«____» _____202_ р.

ЗАТВЕРДЖУЮ:
Завідувач кафедри інформаційних систем в економіці

_____ Тішков Б.О.
«____» _____202_ р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

здобувача вищої освіти Безпалько Юліани Сергіївни

денної форми навчання

на підготовку кваліфікаційної магістерської роботи
на тему: *«Розроблення веб-платформи для підтримки біженців»*

Тему затверджено наказом ректора Університету від «07» березня 2025 р. №464-ст

Кваліфікаційна магістерська робота виконується на матеріалах
відкритих публікацій та інтернет-джерел

План кваліфікаційної магістерської роботи

Розділ І Аналіз сучасних цифрових рішень для підтримки біженців та потреб користувачів у кризових умовах
Розділ ІІ Проєктування та характеристика інформаційної системи веб-платформи для підтримки біженців
Розділ ІІІ Реалізація, тестування та оцінювання ефективності веб-платформи для підтримки біженців
Об'єкт дослідження: процеси цифрової підтримки біженців із використанням веб-платформ у гуманітарному контексті.
Предмет дослідження: функціональні, архітектурні та технологічні аспекти розробки веб-платформи для біженців, що забезпечують комунікацію та надання допомоги.
Мета кваліфікаційної магістерської роботи полягає у розробці веб-платформи, яка забезпечує можливість біженцям і волонтерам взаємодіяти через зручне цифрове середовище,

створювати запити чи пропозиції допомоги, реагувати на них, а також використовувати су
механізми аутентифікації.

Конкретні завдання, які здобувач повинен виконати для досягнення поставленої мети:

У Розділі І проаналізувати існуючі цифрові рішення для підтримки біженців, оцінити
функціональні можливості, сильні та слабкі сторони, а також визначити соціально-тех
потреби цільової аудиторії.

У Розділі ІІ сформувати архітектуру платформи, описати ролі користувачів, обґрунтувати в
технологій, розробити модулі реєстрації, створення запитів і пропозицій, особисті кабіне
механізми взаємодії між користувачами.

У Розділі ІІІ реалізувати та протестувати платформу, оцінити її з точки зору функціональн
стійкості, простоти використання та відповідності гуманітарним цілям, визначити ефектив
запропонованого рішення в умовах реального користування.

Завдання підготував
науковий керівник _____ О.П. Мозгаллі
(підпис)

«10» березня 20

Завдання одержав
здобувач _____ Ю.С. Безпалько
(підпис)

«10» березня 20

Реферат

Кваліфікаційна магістерська робота містить 96 сторінок, 0 таблиць, 56 рисунків, список використаних джерел з 54 найменувань, додатки.

"Розроблення веб-платформи для підтримки біженців"

Об'єктом дослідження є процеси цифрової взаємодії між біженцями та волонтерами за допомогою веб-технологій у контексті гуманітарної підтримки.
Предметом дослідження виступає веб-платформа для підтримки біженців, а також моделі, технології, методи та архітектурні рішення, які забезпечують ефективну комунікацію, доступ до допомоги.
Мета роботи є розроблення веб-платформи, яка дозволяє біженцям і волонтерам ефективно взаємодіяти в онлайн-середовищі шляхом створення запитів чи пропозицій допомоги, аутентифікації за допомогою JWT та Google OAuth, а також використання особистих кабінетів і системи відгуків, та сучасних технологій.

Відповідно до поставленої мети визначені такі завдання:

- Здійснити ґрунтовний аналіз існуючих цифрових рішень і веб-платформ, спрямованих на підтримку біженців та інших вразливих категорій населення.
- Сформувати архітектурну модель веб-платформи з урахуванням особливостей взаємодії між користувачами різних ролей, включаючи модулі аутентифікації, створення запитів і пропозицій допомоги та особисті кабінети.
- Обґрунтувати вибір сучасних веб-технологій і інструментів розробки, які забезпечують стабільність, масштабованість і зручність користування: React, ASP.NET Core Web API, JWT, Google OAuth, SQL Server, Docker.
- Реалізувати технічні рішення, пов’язані з проєктуванням структури бази даних, розробленням клієнтської та серверної логіки, інтерфейсних компонентів платформи.
- Провести тестування розробленої платформи, зокрема перевірку її функціональності.

Теоретична, методична та практична значущість отриманих результатів. Під час дослідження було систематизовано сучасні підходи до створення соціально орієнтованих веб-платформ, що підтримують цифрову взаємодію в умовах гуманітарних викликів. У роботі уточнено моделі рольової поведінки користувачів у середовищах підтримки та розкрито переваги застосування персоналізованого інтерфейсу в кризовому контексті. Практична цінність полягає у розробленні веб-платформи, яка надає біженцям і волонтерам ефективний інструмент для пошуку та надання допомоги, що може бути масштабованим та адаптованим до потреб різних регіонів і ситуацій.

Рік виконання кваліфікаційної магістерської роботи – 2025

Рік захисту роботи – 2025

Ключові слова: веб-платформа, підтримка біженців, рольова модель користувача, інформаційна система, React, ASP.NET Core, JWT, Google OAuth, гуманітарна допомога, CI/CD, Docker.

Відгук
про кваліфікаційну магістерську роботу
здобувача навчально-наукового інституту
«Інститут інформаційних технологій в економіці»
освітньо-професійної програми «Інформаційні управляючі системи і технології»

Безпалько Юліани Сергіївни

на тему ***«Розроблення веб-платформи для підтримки біженців»***

1. Актуальність теми: Дослідження пропонує цифрове рішення для координації гуманітарної допомоги біженцям - це питання, що потребує термінового вирішення в умовах глобальних міграційних криз. Авторка аргументовано показує, як сучасні веб-технології можуть підвищити ефективність взаємодії між волонтерами та особами, які звертаються по допомогу

2. Позитивні риси кваліфікаційної магістерської роботи: Робота відзначається логічною та послідовною структурою дослідження, використанням сучасного технологічного стеку як React, ASP.NET Core Web API, JWT, Google OAuth 2.0, Docker, що забезпечує безпеку й масштабованість

3. Наявність самостійних розробок автора Кваліфікаційна магістерська робота є самостійною розробкою автора, зокрема автор реалізувала REST-сервіси та клієнтські компоненти інтерфейсу та налаштувала CI/CD процеси

4. Цінність теоретичних висновків та практичних рекомендацій: У роботі узагальнено модель «запит – пропозиція допомоги» з гнучкими ролями користувачів, наведено критерії UX-дизайну в кризовому контексті та подано рекомендації щодо інтеграції з державними реєстрами соціальної допомоги. Результати можуть бути застосовані іншими гуманітарними платформами.

5. Наявність недоліків: Робота поки не містить інтеграції з картографічними сервісами й системою сповіщень, що могло б підвищити зручність для користувачів.

6. Загальна оцінка кваліфікаційної магістерської роботи та її допущення до захисту перед ЕК: Кваліфікаційну магістерську роботу вважаю виконаною на високому рівні та рекомендую до захисту перед ЕК

Науковий керівник: *професор кафедри інформаційних систем в економіці, професор, д.е.н.*

_____ *Мозгаллі О. П.*

«__» _____ 20__ р.

Рецензія
на кваліфікаційну магістерську роботу
здобувача вищої освіти
Безпалько Юліани Сергіївни

Тема **«Розроблення веб-платформи для підтримки біженців»**

Актуальність теми кваліфікаційної магістерської роботи і доцільність її розроблення: Робота спрямована на вирішення гострої соціально-гуманітарної проблеми – оперативної координації допомоги людям, вимушеним покинути свої домівки через військові дії чи стихійні лиха. Запропонований цифровий інструмент відповідає пріоритетам міжнародних ініціатив зі сталого розвитку і стратегіям цифрової інклюзії, що зумовлює високу суспільну та практичну значущість дослідження.

Якість проведеного дослідження: Матеріал подано структуровано й логічно, теоретична частина поєднана з практичною реалізацією. Авторка обґрунтовано вибрала сучасний технологічний стек з React 18, ASP.NET Core 8, SQL Server, Docker

Позитивні риси кваліфікаційної магістерської роботи: У роботі простежується вміння самостійно аналізувати проблему, формулювати обґрунтовані технічні рішення й реалізовувати їх на практиці. Авторка успішно впровадила функціонал, що враховує потреби користувачів із різними ролями, подбала про зручність використання інтерфейсу.

Зауваження: Серед зауважень варто зазначити відсутність інтеграції з картографічними сервісами, яка могла б покращити логістику надання допомоги.

Практична значимість висновків і рекомендацій: Робота має високу практичну цінність, оскільки створене рішення може бути використане в реальних умовах волонтерської чи муніципальної діяльності. Рекомендації, викладені в роботі, можуть слугувати основою для впровадження аналогічних інформаційних систем у різних гуманітарних ініціативах

Місце роботи та посада рецензента
Науковий ступінь, учене звання (*за наявності*) Керівник технічної команди (Техлід)
ТОВ «BinarGear»

Підпис засвідчую: Гурський Б.В.

ФІЗИЧНА ОСОБА-ПІДПРИЄМЕЦЬ
Гурський
Богдан
Вадимович
3748107037
УКРАЇНА, м.ВІННИЦЯ

ЗМІСТ

ВСТУП **4**
РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ НАВЧАЛЬНИХ ПЛАТФОРМ **7**
1.1 Аналіз існуючих інформаційних систем сучасних веб-платформ спрямованих на підтримку біженців 7
1.1.1 Дослідження предметної області 7
1.1.2 Дослідження інформаційних управляючих систем сучасних веб-платформ для підтримки біженців 12
1.2 Обґрунтування вибору підходів і технологій для створення інформаційної управляючої системи 15
РОЗДІЛ 2. ПОСТАНОВКА ЗАВДАННЯ ТА ПРОЄКТУВАННЯ СИСТЕМИ **20**
2.1 Структура і характеристика інформаційної системи у предметній області 20
2.2 Методи та моделі в інформаційних управляючих системах і технологіях 26
РОЗДІЛ 3. РОЗРОБЛЕННЯ ТА РЕАЛІЗАЦІЯ ВЕБ-ПЛАТФОРМИ ДЛЯ ПІДТРИМКИ БІЖЕНЦІВ **30**
3.1 Проєктування структури бази даних веб-платформи 30
3.1.1 Інфологічна (логічна) модель бази даних 30
3.1.2 Обґрунтування вибору СКБД та проєктування фізичної моделі 37
3.1.3 Контрольний приклад та обрахунок прогнозованих обсягів 39
3.2 Реалізація логіки призначення 41
3.3 Розробка програмного забезпечення платформи 43
3.3.1 Проєктування програмного забезпечення 43
3.3.2 Вибір архітектури програмного забезпечення 45
3.3.3 Інструментальні засоби розробки 49
3.3.4 Тестування програмного забезпечення 52
3.3.5 Керівництво користувача 53
3.4 Визначення технічного забезпечення 55
3.4.1 Обґрунтування вибору технічного забезпечення 55
3.4.2 Ресурсні вимоги до технічного забезпечення 57
3.5 Реалізація веб-платформи та перевірка її працездатності 59
ВИСНОВКИ **88**
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ **91**
ДОДАТКИ **97**
Додаток А 97
Код створення бази даних через EF з міграціями 97
Додаток Б 104
Код контроллера для пропозицій на серверній частині 104
Додаток В 109
Код сервісу для пропозицій на серверній частині 109
Додаток Г 116

Код слайсу для пропозицій на клієнтській частині 116
Додаток Д 121
Код апі для пропозицій на клієнтській частині 121
Додаток Е 123
Код для CI/CD (frontend) пайплайну 123
Додаток Є 125
Код Dockerfile (frontend) 125

ВСТУП

Актуальність теми магістерської роботи зумовлена загостренням глобальних гуманітарних криз, спричинених воєнними конфліктами, політичними переслідуваннями та природними катастрофами. На жаль ці умови змушують мільйони людей залишати свої домівки, стикаючись з потребою у терміновій допомозі, інформаційній підтримці та соціальній інтеграції. У цей же час цифрові технології відіграють ключову роль у налагодженні швидкої комунікації між біженцями, волонтерами та відповідними організаціями. Наявні онлайн-рішення часто не відповідають вимогам оперативності, доступності та зручності користування, що знижує їхню ефективність у кризових ситуаціях. Це все зумовлює необхідність розроблення спеціалізованих платформ, здатних забезпечити якісну підтримку вразливим групам населення.

Аналіз останніх досліджень і публікацій. У фокусі сучасної наукової дискусії дедалі частіше опиняється питання цифрової інфраструктури для підтримки біженців. У звітах UNHCR, IOM, а також у дослідженнях Digital Public Goods Alliance наголошується на важливості створення платформ, орієнтованих на гуманітарні потреби, з урахуванням стандартів безпеки, простоти доступу, мовної адаптації та можливості масштабування [1-3]. Разом з тим, огляд наукових публікацій показує, що досі відсутні ґрунтовні розробки, які охоплювали би повний цикл проєктування, реалізації та тестування платформ, спеціально призначених для взаємодії біженців і волонтерів. Вирішення цієї проблеми вимагає об’єднання сучасних підходів до розробки інформаційних систем, технологій веб-доступності та гуманітарного дизайну.

Наукова проблема полягає у відсутності ефективної веб-платформи для швидкої взаємодії між біженцями та волонтерами, яка була б достатньо

масштабованою, доступною, захищеною та зручною для використання в умовах кризового реагування.

Суть проблеми проявляється в тому, що більшість існуючих платформ або не підтримують ролі, специфічні для гуманітарних сценаріїв, або не забезпечують необхідного рівня персоналізації, безпеки та зручності взаємодії для вразливих категорій користувачів.

Об'єктом дослідження є процеси цифрової комунікації та обміну гуманітарною інформацією в середовищі взаємодії між біженцями та волонтерами.

Предметом дослідження виступають архітектура, функціональні модулі, технології аутентифікації та інтерфейсні рішення веб-платформи для підтримки біженців.

Метою дослідження є розроблення веб-платформи, що забезпечує біженцям можливість створювати запити про допомогу, а волонтерам пропозиції допомоги з можливістю взаємного відгуку, з урахуванням безпеки та масштабованості.

Для досягнення поставленої мети були визначені такі *завдання дослідження*:

- проаналізувати сучасні веб-рішення у сфері гуманітарної підтримки, визначити їхні сильні та слабкі сторони;
- дослідити теоретичні підходи до побудови інформаційних систем для вразливих категорій користувачів;
- сформувати архітектурну модель платформи з чітким поділом ролей користувачів та функціоналом особистого кабінету;
- обґрунтувати вибір стеку технологій для реалізації системи;
- реалізувати функціональні модулі, такі як реєстрацію та вхід, створення та перегляд запитів/пропозицій, систему фільтрації та взаємодії.

Методи дослідження, які були використані в роботі, включають

- аналіз нормативно-правових документів та публікацій;

- моделювання архітектури ІУС;
- розробку за методологією об’єктно-орієнтованого програмування;
- системне тестування та експертну оцінку працездатності.

Результати дослідження були отримані в процесі проходження виробничої практики на базі ФОП «Гурський Богдан Вадимович». Там було реалізовано функціональність веб-платформи, що дозволяє взаємодіяти біженцям та волонтерам у режимі реального часу. Система була протестована на предмет відповідності технічному завданню та функціональних вимог.

Практична цінність роботи полягає в можливості впровадження розробленого програмного продукту для підтримки біженців у кризових ситуаціях. Платформа є універсальною для розширення її використання в неурядових організаціях, соціальних ініціативах та муніципальних установах.

Інформаційна база дослідження охоплює офіційні документи UNHCR, IOM, технічну документацію з веб-розробки, наукові дослідження, а також статистичні звіти та рекомендації з гуманітарного проєктування.

Структура магістерської роботи відповідає поставленим цілям і задачам. Вона складається зі вступу, трьох розділів, висновків, списку використаних джерел і додатків. У першому розділі проаналізовано стан вирішення проблеми, виявлено недоліки існуючих рішень. У другому розроблено архітектуру системи та обґрунтовано вибір технологій. У третьому реалізовано функціональність, проведено тестування та сформульовано практичні рекомендації щодо подальшого впровадження платформи.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ НАВЧАЛЬНИХ ПЛАТФОРМ

1.1 Аналіз існуючих інформаційних систем сучасних веб-платформ спрямованих на підтримку біженців

1.1.1 Дослідження предметної області

Дослідження предметної області інформаційних управляючих систем, призначених для підтримки біженців і внутрішньо переміщених осіб неминуче починається з окреслення історико-теоретичного тла, оскільки саме еволюція міжнародних норм і гуманітарних практик зумовила сучасні вимоги до цифрових рішень. Ідея про те, що переміщені люди потребують не лише разової допомоги, а сталих механізмів захисту та інтеграції, сформувалася після катастрофічних наслідків двох світових воєн. Прийняття Женевської конвенції 1951 року та Протоколу 1967 року кодифікувало базові права біженців і поклало на держави-учасниці обов’язок забезпечити доступ до освіти, медичних послуг і правової підтримки. Водночас зусилля соціологів Чиказької школи, зокрема Роберта Парка та Ернеста Берджесса, спрямували увагу на процеси адаптації та асиміляції в приймаючих суспільствах, а географічні моделі Ернста Рейвенштейна акцентували роль дистанції та економічного «тяжіння» у міграційних потоках [4-6]. Сукупність цих теоретичних імпульсів створила концептуальну матрицю, у межах якої цифрові системи мають не просто зареєструвати факт переміщення, а здатні прогнозувати траєкторії подальшої соціальної інтеграції та мінімізувати ризики маргіналізації.

Із розвитком інформаційних технологій наприкінці XX століття гуманітарний сектор почав експериментувати з електронними базами даних і простими системами обліку. Однак по-справжньому переломним стало поширення Інтернету та мобільного зв’язку. Оперативний обмін даними між польовими офісами, центральними штаб-квартирами й донорами відкрив можливість створення розподілених архітектур, що поєднували оперативну

реєстрацію з аналітичною обробкою потоків інформації. У ранніх версіях домінували монолітні програми, написані під конкретні операційні системи й розгорнуті на серверах окремих організацій. З роками, у міру зростання обсягів даних і вимог до гнучкості, гуманітарні ІУС перейшли до багатошарових архітектур і мікросервісної логіки. Окремі компоненти реєстрації користувачів, модуля бюджетних транзакцій, сховища медичних записів та аналітичного ядра було винесено у незалежні контейнери. Така декомпозиція суттєво спростила CI/CD-процеси, дозволила масштабувати лише ті частини застосунку, які справді зазнавали пікового навантаження, і водночас підвищилась відмовостійкість системи.

У центр будь-якої з цих платформ ставиться модуль ідентифікації, покликаний закрити дві протилежні потреби - підтвердження особи та мінімізацію бюрократичного бар’єра.

Паралельна інтеграція з національними реєстрами населення та вже наявними базами Агентства дозволяє уникнути дублювання заявок і підвищити точність статистики. Слід наголосити, що саме на етапі ідентифікації закладаються основні виклики конфіденційності. З одного боку, дані мають бути достатньо деталізовані, аби визначити потреби конкретної особи, з іншого, будь-яке їхнє надмірне збирання створює ризик вторинної травматизації та потенційного витоку чутливої інформації.

Наступною ключовою ланкою є модуль управління ресурсами, який акумулює інформацію про доступне житло, продуктову допомогу, медичні й освітні послуги, психологічну підтримку та правове консультування. У новітніх реалізаціях такий модуль поєднує data lake-підхід з алгоритмами машинного навчання. Зібрані дані зберігаються у сирому вигляді й регулярно збагачуються сторонніми показниками, такими як демографічними, економічними, кліматичними. На основі цих багатовимірних наборів вибудовуються моделі пріоритезації, які оцінюють ступінь нагальності кожного запиту, пропонують оптимальні логістичні маршрути та одночасно прогнозують загальну потребу на місяць чи квартал уперед. Аналітичний блок, своєю чергою, подає результати в

інтерактивному інтерфейсі для координаторів гуманітарних операцій, переводячи «сирі» цифри у зрозумілі візуалізації навантажень, дефіцитів і резервів.

Окремою площиною виступають соціально-етичні вимоги. Поширення GDPR у країнах Європейського Союзу та низки аналогічних актів у світі зумовило тотальний перегляд підходів до зберігання та обробки персональних даних [7]. Для гуманітарних ІУС це, по суті, означало побудову двох паралельних контурів, таких як операційний, де особисті дані шифруються й обробляються в мінімально потрібному обсязі, і аналітичний, де інформація агрегується та анонімізується настільки, аби зберегти цінність для планування, але виключити можливість ідентифікації особи. Запровадження механізму «права на забуття» вимагало автоматизованих процедур видалення або псевдонімізації записів після завершення гуманітарної місії чи закінчення строку зберігання. Парадигма гуманітарної Data Responsibility підсилила ці зобов’язання, підкресливши неформальний, але дуже практичний імператив не завдати шкоди бенефіціару самим фактом збору та зберігання його даних.

Не менш важливою для повсякденної роботи системи є організаційно-управлінська структура. Ієрархія ролей у цифровій платформі зазвичай віддзеркалює реальний поділ праці між польовими офісами, регіональними центрами та глобальними штаб-квартирами. Адміністратори відповідають за технічну цілісність і політику безпеки, тоді як координатори гуманітарних програм концентруються на маршрутизації запитів і забезпеченні ресурсів. Волонтери та профільні фахівці, як юристи, лікарі, психологи отримують обмежений доступ, необхідний для супроводу конкретних кейсів. Взаємодія цих груп регулюється адаптованими протоколами ITIL та COBIT, які задовольняють вимоги до управління інцидентами, запитами на зміни та безперервним удосконаленням сервісів. У такий спосіб інформаційна система стає не тільки інструментом реєстрації, а й каркасом, що підтримує стандартизовані потоки робіт і прозору комунікацію [8].

Подальший розвиток інформаційних управляючих систем у сфері гуманітарної допомоги, зокрема таких, що підтримують біженців неминуче зіштовхується з дедалі складнішими викликами інтеграції з зовнішніми

джерелами даних і сервісами. Ці інтеграції не є факультативними, а навпаки, вони визначають якість, оперативність і масштаб дій системи в реальному середовищі. У сучасних умовах жодна ІУС не функціонує у вакуумі, її цінність прямо пропорційна кількості зовнішніх каналів, з якими вона може безперебійно взаємодіяти.

Одним із ключових напрямів інтеграції є використання відкритих урядових реєстрів, які дозволяють перевіряти актуальність документів, здійснювати ідентифікацію особи, автоматично заповнювати частину форми для пришвидшення процесу отримання допомоги. Це особливо важливо в умовах великої кількості запитів на підтримку, коли ручна перевірка може стати вузьким місцем у ланцюгу обробки. Водночас така інтеграція породжує новий клас юридичних і процедурних обмежень, як от від дотримання норм GDPR до необхідності регламентованого доступу через захищені API з обмеженням частоти запитів.

Інший критично важливий аспект є картографічні сервіси, інтеграція з якими відкриває можливості просторової аналітики. За допомогою таких платформ, як Google Maps, OpenStreetMap або HERE, ІУС отримує можливість візуалізувати місця знаходження волонтерів, складів, медичних пунктів, а також поточні шляхи евакуації. Завдяки цьому гуманітарні штаби можуть у режимі реального часу переорієнтовувати маршрути транспортування допомоги, враховуючи обстріли, перекриття доріг чи зони небезпеки. Це підвищує мобільність системи та її адаптивність до зміни ситуації на місцях.

Не менш вагомим напрямом є інтеграція з фінансовими шлюзами, зокрема з такими міжнародними протоколами, як SEPA та SWIFT [9-10]. Впровадження цих механізмів дозволяє платформі реалізовувати пряму допомогу у вигляді грошових переказів біженцям без необхідності фізичного посередництва. Проте реалізація такої інтеграції передбачає суворе дотримання процедур KYC та AML, навіть у надзвичайних умовах [11]. Це вимагає додаткової ідентифікації отримувача, перевірки транзакцій на підозрілість і ведення докладного аудиту, що додає рівень складності в архітектуру ІУС.

Технічне спрощення таких інтеграцій стало можливим завдяки широкому розповсюдженню REST та GraphQL API, які забезпечують уніфікований інтерфейс обміну даними. Проте це поставило інші виклики, передусім сувору версифікацію зовнішніх API, а також забезпечення зворотної сумісності при оновленнях. У системі, де взаємодія йде з десятками джерел даних, навіть незначна зміна версії одного з них може спричинити збої у всій платформі. Тому окремо підтримується механізм моніторингу версій і автоматичне тестування точок інтеграції.

Водночас із технологічним розвитком формується комплексна стратегія управління змінами, яка враховує як технічні, так і людські чинники. Впровадження нових модулів супроводжується створенням багатомовних текстових інструкцій, відеоуроків і методичних матеріалів, розрахованих на користувачів із різним рівнем цифрової грамотності. У межах системи розгортається внутрішня база знань, яка інтегрується з чат-ботами підтримки, що дозволяє розвантажити служби технічної підтримки й значно скоротити час первинного реагування. Регулярні вебінари для волонтерів і адміністраторів стають частиною політики безперервного навчання, формуючи живе ком’юніті навколо платформи.

Паралельно в системі запроваджується метрична аналітика, яка охоплює як технічні, так і управлінські показники. До ключових належать середній час реакції на інциденти, коефіцієнт використання складів, ефективність фінансових траншів, точність прогнозів аналітичних моделей, а також індекс задоволеності користувачів. Ці дані не лише відображають поточний стан системи, а й дають змогу ухвалювати стратегічні рішення, виявляти вузькі місця й приймати зважені управлінські рішення як на рівні коду, так і в організації процесів.

У цілому, предметна область інформаційних управляючих систем для біженців це не лише програмна платформа чи набір сервісів. Це складна міждисциплінарна структура, що інтегрує нормативно-правові положення міжнародного гуманітарного права, соціологічні теорії міграції та асиміляції, технології машинного навчання для аналітики, стандарти кібербезпеки, а також методики управління життєвим циклом IT-проєктів. Кожен з цих елементів

створює не лише обмеження, а й нові можливості для розширення функціоналу, посилення адаптивності та підвищення довіри з боку користувачів.

1.1.2 Дослідження інформаційних управляючих систем сучасних веб-платформ для підтримки біженців

Інформаційні управляючі системи, які обслуговують гуманітарний сектор сформувалися на перетині двох динамічних тенденцій, таких як стандартизації процесів міжнародних організацій та стрімкого розвитку хмарних технологій. Провідні платформи як UNHCR Refugee App, Flüchtlinge Willkommen та Help Refugees посідають різні ніші, але розкривають спільний вектор еволюції ІУС.

UNHCR Refugee App виступає одним із перших прикладів глобального цифрового рішення у сфері захисту біженців, яке поєднує централізовану реєстрацію користувачів із децентралізованим наданням гуманітарних послуг [1]. Організаційна логіка платформи передбачає дворівневу модель ухвалення рішень. Індивідуальні звернення опрацьовуються польовими офісами, тоді як стратегічне прогнозування потреб здійснюється на базі агрегованих даних у штаб-квартирі UNHCR у Женеві. У системі ідентифікації інтегровано Biometric Identity Management System, що забезпечує уніфіковану історію звернень незалежно від місця реєстрації та сприяє зменшенню дублювання запитів. З урахуванням високої чутливості персональних даних, доцільним є припущення щодо використання сучасних підходів до кібербезпеки, зокрема шифрування за моделлю envelope encryption та маршрутизації запитів через zero-trust-шлюзи у середовищі AWS GovCloud. Такий підхід відповідає сучасним вимогам до захисту даних у міжнародних IT-системах гуманітарного призначення.

Платформа Flüchtlinge Willkommen сформувалася як ініціатива громадянського суспільства, що реалізує принцип соціального посередництва у сфері розміщення біженців [12]. Основна ідея полягає не у масовій реєстрації користувачів, а у створенні механізму персоналізованого підбору приватних

домогосподарств для конкретних осіб або сімей, які шукають притулок. У контексті концептуального моделювання можливо припустити, що її інформаційна система має гібридну архітектуру. Основний функціонал реалізований у межах монолітного застосунку на основі Laravel, який надає REST API для взаємодії з іншими компонентами. При цьому модуль рекомендаційного добору, побудований із використанням алгоритмів колаборативної фільтрації, може бути винесений в окремий Docker-контейнер на Python. Такий підхід забезпечує гнучкість експериментування з алгоритмами без ризику порушення транзакційної цілісності основної бази даних. Водночас обмеження ресурсів, характерні для волонтерських ініціатив, можуть зумовлювати відсутність повноцінного CI/CD-конвеєра та автоматизованого тестування, що робить систему потенційно вразливою до навантаження в умовах різких міграційних сплесків.

Організація Help Refugees (нині Choose Love) з моменту свого створення орієнтувалася на створення мультинаціональної онлайн-платформи для збору пожертв, що визначило пріоритетність прозорості фінансових операцій у її інформаційній системі [14]. У якості гіпотетичної архітектури можна розглядати мікросервісний бекенд, реалізований засобами .NET Core, з інтеграцією платіжного провайдера Stripe через шлюз, сумісний зі стандартом PCI-DSS. Передбачається, що концептуальна модель управління охоплює не лише обробку транзакцій, але й аналітику вищого рівня, зокрема прогнозування нестачі окремих категорій допомоги, автоматичне ініціювання закупівель і контроль виконання заявок у межах погоджених SLA з партнерськими організаціями. Аналітична обробка даних могла б здійснюватися в середовищі Azure Synapse, де в реальному часі оновлюються ключові показники ефективності, як-от середній час виконання запиту, відповідність пожертви реальній потребі та рівень задоволеності одержувачів допомоги на основі мобільного опитування.

Порівняльний аналіз засвідчує, що повнота функціональних можливостей інформаційних систем гуманітарних платформ часто корелює з рівнем централізації фінансових потоків. Чим більший обсяг залучених коштів, то ширшим є набір доступних сервісів і глибшим рівень автоматизації управлінських процесів. Архітектурні та концептуальні підходи до управління в різних системах

суттєво відрізняються. Так, у платформі Help Refugees переважає логіка алгоритмічного зіставлення пари «запит-ресурс», тоді як UNHCR реалізує ієрархічну модель ескалації звернень - від локальної обробки кейсів до централізованого стратегічного прогнозування на глобальному рівні. У всіх моделях ключовим елементом виступають ситуації прийняття рішень, в яких система має здійснити пріоритезацію запитів за критеріями терміновості, вразливості заявника та наявності релевантних ресурсів.

Інноваційні технології штучного інтелекту, машинного навчання та аналітики великих даних впроваджуються у такі системи поступово, переважно у вигляді пілотних модулів. Зокрема, UNHCR розробляє й апробує прототип під назвою «Predictive Impact Model», який, за повідомленнями у науковій літературі, застосовує градієнтний бустинг для ризик-скорингу домогосподарств на основі демографічних характеристик, маршруту переміщення та регіональних показників соціально-економічної вразливості [14]. У рамках концептуального аналізу підходів до автоматизації класифікації гуманітарних потреб розглядається також можливість використання мовних моделей типу BERT для векторизації описів запитів і подальшої кластеризації з метою зменшення фрагментації логістики та полегшення обробки однотипних заявок. Однак масштабне впровадження таких рішень наразі обмежене через відсутність верифікованих датасетів і високу чутливість персональних даних, що потребує дотримання жорстких нормативно-етичних вимог.

Загалом комплексний огляд підтверджує, що на сучасному етапі розвитку цифрових гуманітарних сервісів не існує платформи, яка б одночасно забезпечувала високу масштабованість, гнучкість сценаріїв прийняття рішень, повну відповідність нормам Загального регламенту захисту даних і принципам етики Data Responsibility. Це засвідчує актуальність подальших досліджень та розробок у напрямку інтеграції інтелектуальних рішень, збереження етичних стандартів і підвищення інклюзивності в інформаційних системах підтримки вразливих категорій населення.

1.2 Обґрунтування вибору підходів і технологій для створення інформаційної управляючої системи

Обґрунтовуючи вибір конкретних підходів і технологій для створення інформаційної управляючої системи веб-платформи підтримки біженців, необхідно виходити не лише з модних трендів індустрії, а передусім із вимог, заданих самою предметною областю. Для гуманітарного середовища ключовими залишаються гарантована доступність сервісу у стресових умовах, безкомпромісна безпека персональних даних, здатність швидко масштабуватися у разі різкого сплеску запитів і, водночас, зручність користування для людей з дуже різним рівнем цифрової грамотності. У такій парадигмі кожен вибраний інструмент має доводити свою доречність не ринковою популярністю, а здатністю закривати конкретні функціональні та нефункціональні вимоги.

Клієнтську частину платформи вирішено будувати на основі зв’язки React і TypeScript [15-16]. Вибір React пояснюється насамперед архітектурними перевагами його віртуального DOM, який мінімізує зайві перерендерення, що особливо важливо, коли в одній сесії відкрито десятки інтерактивних форм. Під час гуманітарних криз користувачі часто заходять із малопотужних мобільних пристроїв або з нестабільним мобільним інтернетом. Оптимізація рендерингу дозволяє знизити навантаження на процесор і, відповідно, продовжити час роботи акумулятора. Компонентна модель React забезпечує чітку декомпозицію інтерфейсу, тобто форму подання заявки, компоненти можна розробляти ізольовано, підтримуючи єдиний дизайн-систем. Використання TypeScript додає ще один, критичний для великих застосунків, рівень безпеки. Сувора статична типізація дозволяє відловити несумісність контрактів між компонентами ще на етапі компіляції, знижуючи ризик помилок, які в надлишкових умовах могли б паралізувати роботу цілих модулів. Крім того, TypeScript суттєво спрощує інтеграцію з інструментами статичного аналізу, автогенерацію документації у проєкті завдяки зрозумілій сигнатурі об’єктів.

Для керування глобальним станом обрано Redux Toolkit, оскільки він усуває більшість рутинних операцій класичного Redux і постачає строго детермінований підхід до мутацій стану. Маршрутизацію між численними підрозділами застосунку виконує React Router, що підтримує lazy loading [17]. Завдяки цьому користувач не завантажує зайвих скриптів до моменту, поки вони справді знадобляться. Для HTTP-комунікацій використано Axios [18]. Його перехоплювачі дозволяють тонко керувати вставкою JWT-токенів і обробкою повторної автентифікації без дублювання коду у кожному запиті.

Серверну частину веб-платформи реалізовано з використанням .NET 8 Web API, що є сучасним інструментом для створення масштабованих, продуктивних і безпечних веб-сервісів [19]. Вибір цієї технології зумовлений не лише її відповідністю архітектурним потребам, а й вражаючими показниками продуктивності, стабільності та підтримки з боку спільноти розробників. Порівняльні бенчмарки, проведені на основі простих сценаріїв серіалізації/десеріалізації JSON, показують, що .NET 8 демонструє затримку обробки запиту менше ніж 1 мілісекунда у більшості стандартних операцій. Завдяки впровадженню вдосконаленого Thread-Pool та можливості використання AOT-компіляції система здатна зберігати стабільну пропускну здатність навіть у разі високих пікових навантажень.

Ці технічні характеристики є критично важливими для платформи підтримки біженців, яка повинна бути доступною та надійною цілодобово і справлятися з десятками тисяч конкурентних з’єднань під час кризових ситуацій або масового подання заявок. Завдяки продуктивності .NET 8, платформа забезпечує високу щільність обробки запитів на одиницю апаратного ресурсу, що дозволяє мінімізувати витрати на масштабування інфраструктури в хмарному середовищі або при розміщенні на власних серверах. У випадках, коли потрібна швидка реакція на події, така як реєстрація нових запитів, оновлення статусів або масова розсилка повідомлень, .NET 8 здатен забезпечити низький час відповіді без втрати стабільності системи.

Однією з переваг використання ASP.NET Core є потужна модель проміжного програмного забезпечення, яка дозволяє будувати виразну та

контрольовану обробку HTTP-запитів. Кожен запит, який надходить до сервера, проходить через послідовність проміжних обробників, серед яких реєстрація логів, перевірка частоти запитів, перевірка валідності токенів автентифікації, перевірка ролей і прав доступу та валідація вхідних даних на основі анотацій моделей. Така послідовна структура забезпечує чітку трасованість усіх етапів обробки запиту, що особливо важливо в контексті відповідності вимогам OWASP і проведення формального аудиту безпеки.

Перевага цієї архітектури полягає в її прозорості та тестованості. Кожен компонент middleware можна протестувати окремо, перевірити на відповідність специфікаціям безпеки, і за потреби легко замінити чи розширити. Наприклад, додавання нових механізмів автентифікації або обмеження доступу для певних ролей не вимагає переписування бізнес-логіки або контролерів, а лише модифікації відповідного шару обробки запиту.

Ще однією перевагою є гнучкість налаштувань обробки помилок та логування. ASP.NET Core дозволяє інтегрувати сучасні системи журналювання подій, зокрема Serilog, Seq або Application Insights. Це дає змогу отримувати повну аналітику по роботі API як час відповіді, частота помилок, IP-активність користувачів тощо. Завдяки цьому адміністратори можуть оперативно виявляти відхилення від очікуваної поведінки системи, локалізувати вузькі місця та оптимізувати продуктивність або безпеку у реальному часі.

Доступ до даних у системі реалізовано за допомогою Entity Framework Core, що дозволяє працювати з базою даних у формі об’єктів C# без необхідності писати SQL-запити вручну [20]. Використовується підхід Code First, який передбачає спочатку створення моделей у коді, а вже потім автоматичне формування схеми бази даних за допомогою міграцій. Це значно спрощує підтримку проекту, особливо в умовах, коли нормативні вимоги до структури даних часто змінюються достатньо оновити модель і створити нову міграцію.

Для ізоляції бізнес-логіки від деталей роботи з БД застосовано патерни Repository та Unit of Work [21]. Вони не лише роблять код чистішим і зручнішим для тестування, а й забезпечують цілісність транзакцій, що критично важливо,

наприклад, при паралельному оновленні записів складу та донорських переказів обидві операції мають бути виконані або разом, або жодна з них.

Аутентифікація користувачів у системі реалізована з використанням стандартизованих JWT-токенів [22]. Це сучасний і широко визнаний підхід до забезпечення безпечного доступу, який дозволяє уникнути зберігання сесійного стану на сервері, тим самим знижуючи навантаження на інфраструктуру. JWT-токени створюються на сервері після успішного входу користувача, підписуються за допомогою криптографічного алгоритму і потім передаються клієнту для зберігання, переважно у LocalStorage або HTTP-only cookie. У сам токен вбудовуються ключові атрибути такі як роль користувача, строк дії токена, а також, за потреби, додаткові claims, які дозволяють приймати контекстні рішення на фронтенді або у мікросервісах.

Важливою перевагою є впровадження OAuth 2.0-автентифікації через обліковий запис Google, яка істотно спрощує процедуру входу для більшості користувачів, особливо тих, хто має обмежений технічний досвід або перебуває в умовах терміновості [23]. Крім зручності, такий механізм також забезпечує високий рівень безпеки, оскільки криптографічні операції, включно з обміном ключами, валідацією сертифікатів і збереженням даних автентифікації, делегуються надійному провайдеру Google. Це дозволяє зменшити поверхню атаки в самій системі та зняти частину відповідальності за захист паролів, їх хешування та зберігання з розробника платформи.

Канал зв’язку між клієнтом і сервером захищено протоколом TLS, щонайменше версії 1.2, з актуальним шифруванням, що забезпечує конфіденційність і цілісність переданих даних. Це гарантує, що токени автентифікації, персональні дані користувача або будь-яка інша чутлива інформація не можуть бути перехоплені або змінені під час транспортування мережею.

Додатковий рівень захисту забезпечено через впровадження захисних HTTP-заголовків і директив Content Security Policy. Завдяки цим налаштуванням браузер суворо обмежує джерела скриптів, стилів, медіа та інших елементів, що значно знижує ризик XSS-атак.

У сфері CI/CD платформа впровадила формалізовану DevOps-практику, яка реалізована за допомогою GitHub Actions [24]. Автоматизований пайплайн охоплює всі основні етапи життєвого циклу програмного забезпечення. На першому етапі запускаються перевірки коду, включаючи типовий статичний аналіз. Далі проходить юніт-тестування за допомогою xUnit, що забезпечує автоматичну перевірку логіки на рівні сервісів, контролерів і обробників подій.

Після успішного тестування відбувається збірка Docker-образів, які містять як серверну, так і фронтенд-частину, з урахуванням параметрів production-оточення [25]. Після цього виконується автоматичне сканування образів на наявність вразливостей, із використанням вбудованих або зовнішніх інструментів, що дозволяє на ранньому етапі виявити потенційно небезпечні бібліотеки чи залежності. У разі успішного проходження всіх перевірок образ автоматично пушиться до Docker Registry, звідки може бути розгорнутий у staging або production-середовище.

Усі вхідні параметри проходять централізовану валідацію, базові принципи мінімізації прав доповнено політикою CORS, налаштованою в режимі робочого білого списку [26]. Вразливість SQL-ін’єкцій купується параметризованими запитами Entity Framework Core та додатковим рівнем Web Application Firewall на інфраструктурному периметрі.

Сукупність наведених рішень формує узгоджену, технологічно цілісну екосистему. React із TypeScript забезпечує легкий, швидкий та розширюваний фронтенд, здатний до прогресивної деградації у малопотужних пристроях. .NET 8 із Entity Framework Core й SQL Server гарантує високу пропускну здатність, транзакційну цілісність і багатий інструментарій аналітики. Docker закладає фундамент для гнучкого масштабу, а GitHub Actions перетворюють релізний цикл на безперервну, підконтрольну конвеєрну лінію. Усі ці компоненти разом задовольняють вимоги безпеки, продуктивності, підтримуваності і дозволяють оперативно адаптуватися до законодавчих змін чи нових гуманітарних викликів без кардинальної перебудови архітектури.

РОЗДІЛ 2. ПОСТАНОВКА ЗАВДАННЯ ТА ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Структура і характеристика інформаційної системи у предметній області

Процес створення інформаційної управляючої системи, спрямований на підтримку біженців, розпочався з глибокого та всебічного вивчення предметної області. Для цього було проведено аналіз наявних соціальних ініціатив та державних програм, спрямованих на допомогу переміщеним особам, а також оцінено досвід міжнародних організацій у впровадженні схожих цифрових рішень. Паралельно здійснювався детальний порівняльний аналіз існуючих інформаційних систем, щоби виокремити їхні сильні й слабкі сторони, визначити типові технічні підходи та бізнес-логіку. За результатами цих досліджень сформовано чіткий перелік основних вимог до майбутньої платформи, таких як масштабованість бази даних, зручний інтерфейс для різних категорій користувачів. Окрім цього, на ранніх етапах було визначено ключові ролі користувачів, описано інформаційні потоки між ними та деталізовано базові сценарії взаємодії.

Концептуальна модель системи створювалася із застосуванням перевірених сучасних методологій моделювання складних програмних систем. У якості головного інструменту обрали мову уніфікованого моделювання UML, що надало змогу формалізувати всі ключові компоненти архітектури та їхні зв’язки. За допомогою діаграм класів було окреслено структуру баз даних та сутності, які зберігатимуть інформацію про користувачів, запити й відповіді, на діаграмах послідовностей. Вони відобразили порядок виклику сервісів і обміну повідомленнями між модулями, на діаграмах розгортання вони показали фізичне розташування компонентів і залежності між ними. Такий підхід дозволив не тільки чітко зафіксувати функціональні блоки платформи, але й забезпечити їхню подальшу масштабованість та гнучкість. У майбутньому можна буде легко

додавати нові сервіси чи змінювати бізнес-логіку без кардинальних перебудов системи.

Одним із центральних завдань на етапі проєктування стало визначення й деталізація користувацьких ролей та пов’язаних із ними функціональних можливостей. У системі виокремлено дві основні рольові категорії такі як «біженець» та «волонтер», кожна з яких наділена власним набором дій і прав. Наприклад, роль «біженець» передбачає можливість створення персональних звернень за допомогою зручної форми, відстеження статусу запитів. Роль «волонтер» має ті ж самі можливості, але з пропозиціями. Обом ролям забезпечено єдину систему аутентифікації та захищеного обміну даними, а також можливість взаємодіяти через підтвердження допомоги. Такий розподіл ролей гарантує, що кожен користувач отримує саме ті інструменти, які потрібні для максимально ефективної роботи в межах платформи.

Рисунок 2.1 – Діаграма прецедентів

Джерело: розроблено автором на основі власних досліджень

Наступним етапом стало проектування діаграми прецедентів, яка чітко візуалізує всі можливі сценарії взаємодії користувачів із системою. Завдяки цьому

вдалося чітко визначити межі функціональності системи та окреслити ролі й повноваження кожної категорії користувачів.

Деталізацію кожного прецеденту було виконано за допомогою діаграм послідовності, що дозволило чітко представити взаємодію компонентів системи в часовому вимірі. Ці діаграми наочно відображають повідомлення, які обмінюються між клієнтським інтерфейсом на базі React, серверною частиною, що базується на ASP.NET Core Web API, та рівнем доступу до даних. Завдяки таким діаграмам стало зрозуміло, які саме методи API викликаються, як відбувається обробка даних, які відповіді отримує користувач у результаті своїх дій та яким чином забезпечується синхронізація статусів заявок у системі.

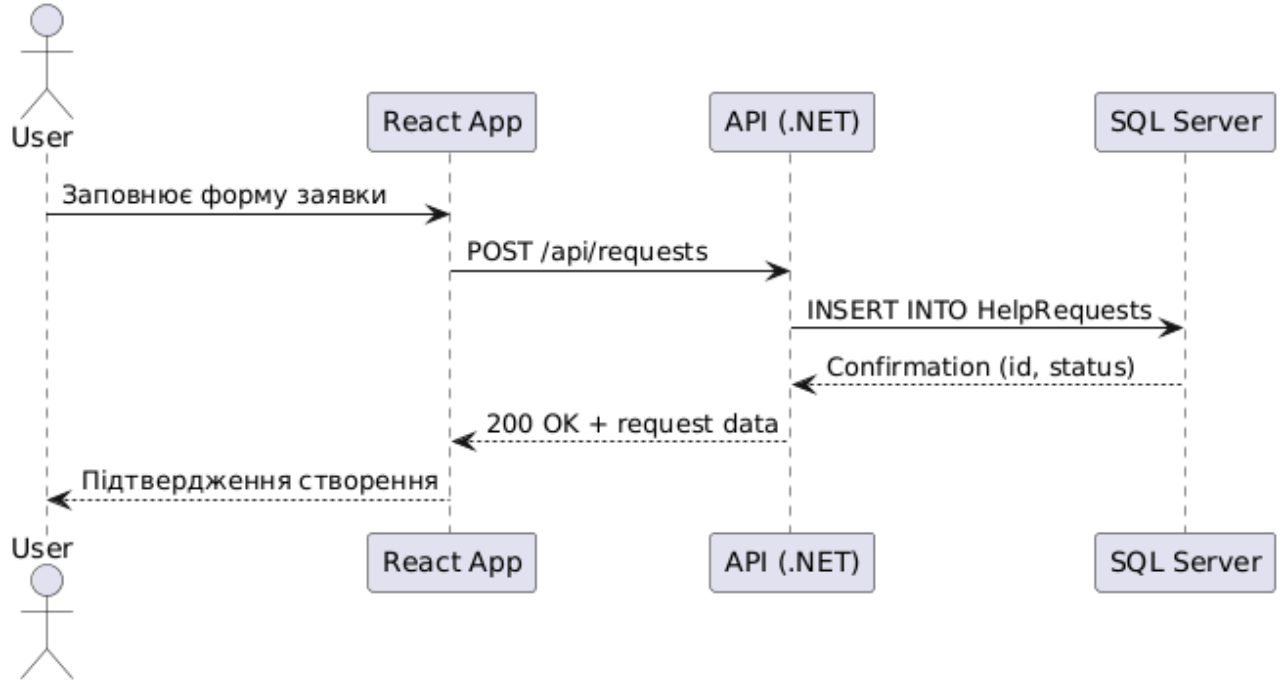


Рисунок 2.2 – Діаграма послідовності - Створення заявки

Джерело: розроблено автором на основі власних досліджень

Особлива увага була приділена розробці діаграми діяльності. Вона дозволила чітко представити паралельні і послідовні процеси в рамках функціонування системи. Ця діаграма охоплює такі складні процеси, як одночасне створення та обробка великої кількості заявок, паралельні комунікації користувачів, синхронізацію статусів. Завдяки діаграмі діяльності вдалося ідентифікувати «вузькі місця» у взаємодії компонентів, оптимізувати інформаційні потоки та забезпечити ефективність роботи платформи навіть за умов значного навантаження.

Рисунок 2.3 – Діаграма діяльності - Обробка заявки

Джерело: розроблено автором на основі власних досліджень

Для представлення статичної структури системи було створено діаграми класів, які чітко відображають логічну модель платформи. Ці діаграми включають сутності. Кожна з цих сутностей має чітко визначені атрибути та методи, що дозволяє організувати ефективну роботу з даними у системі. Також на цих діаграмах представлено класи керування, які відповідають за реалізацію бізнес-логіки, обробку запитів, керування статусами, та іншими важливими функціями. Крім того, граничні класи на цих діаграмах відповідають за інтерфейс користувача та взаємодію з клієнтом.

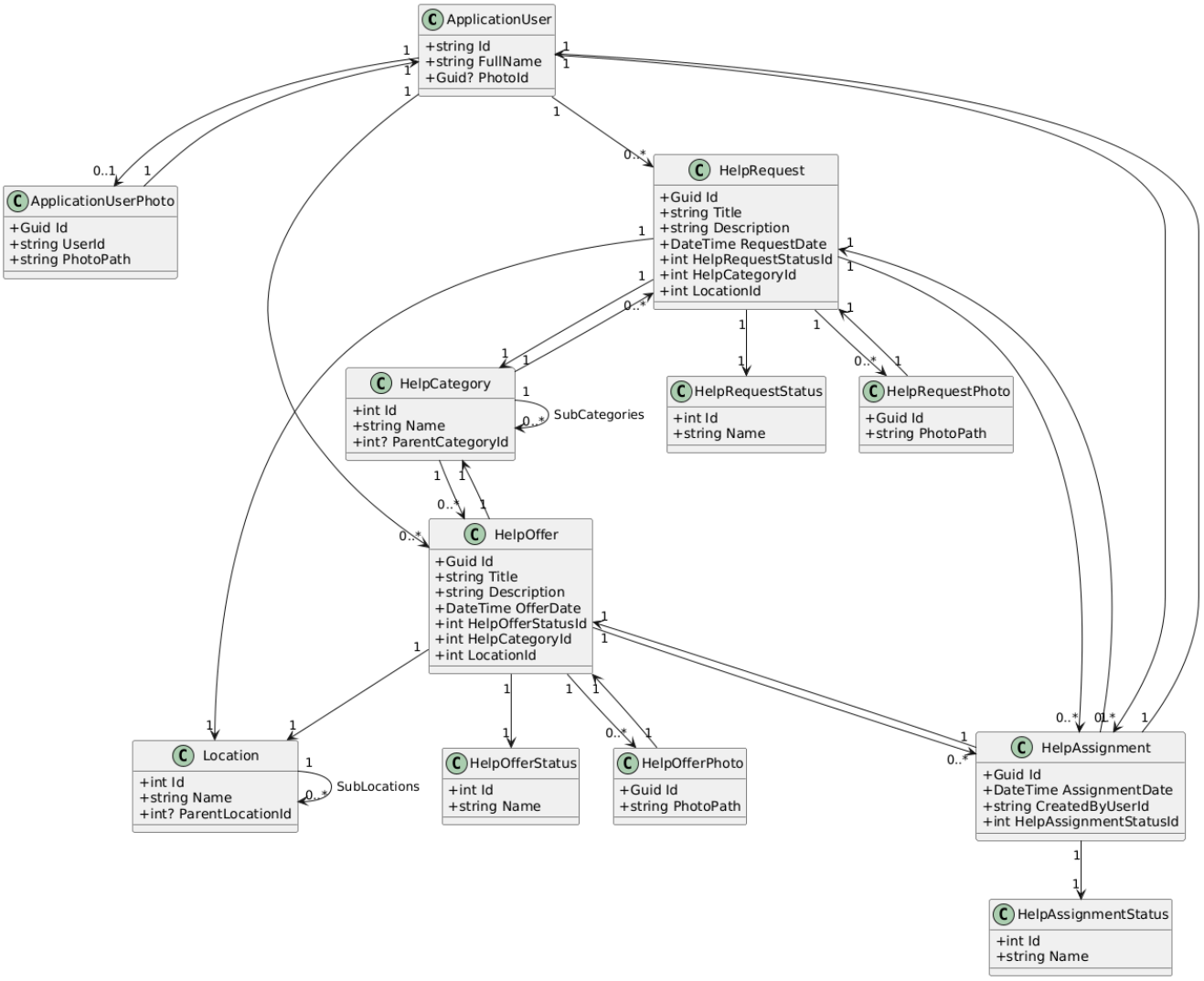


Рисунок 2.4 – Діаграма класів

Джерело: розроблено автором на основі власних досліджень

Ключовим аспектом реалізації системи є її архітектурна структура. Для забезпечення високої масштабованості, продуктивності та відмовостійкості було обрано монолітну архітектуру з використанням контейнеризації Docker. Кожен сервіс, такий як автентифікація, управління заявками, комунікація між користувачами, має свій власний контейнер та чітко визначені інтерфейси взаємодії.

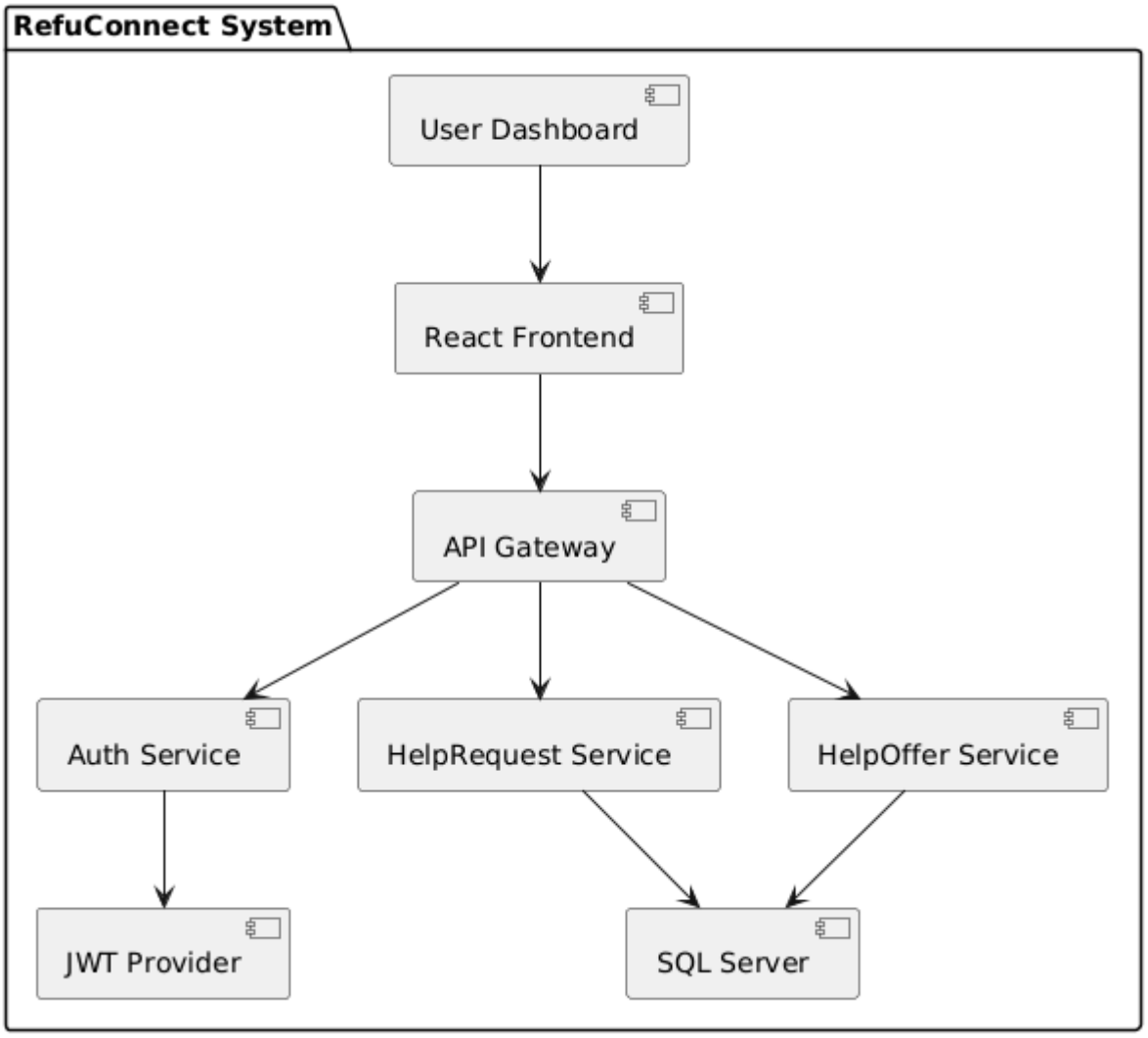


Рисунок 2.5 – Діаграма компонентів - Архітектура системи

Джерело: розроблено автором на основі власних досліджень

Для гарантування безпеки та конфіденційності даних в системі впроваджено багаторівневу модель захисту. На рівні взаємодії з клієнтською частиною застосовується автентифікація на базі JSON Web Token. Після успішного ведення облікових даних користувач отримує токен доступу. Це дозволяє мінімізувати ризик злому. Крім того, інтеграція з Google OAuth 2.0 забезпечує можливість «єдиного входу» та додатковий рівень довіри завдяки перевіреній сторонній платформі, а також підтримує сувору багато етапну перевірку особистості користувачів через Google Identity Platform.

Усі дані, що передаються між клієнтом та сервером, захищені протоколом HTTPS із використанням сучасних версій TLS [27]. Сертифікати отримуються та автоматично поновлюються, що гарантує безперервність служби без простоїв.

Важливим аспектом проектування стала не лише безпека, а й висока зручність інтерфейсу для кінцевих користувачів. Фронтенд створено на базі React з TypeScript, що дозволило застосувати строгі типи для всіх компонентів та зменшити кількість помилок ще на етапі компіляції. Використано підхід модульного дизайну з code splitting, lazy loading, що забезпечує миттєве завантаження ключових екранів та покращений SEO. Адаптивність інтерфейсу гарантує комфортну роботу на пристроях різного розміру. Для керування глобальним станом застосовано Redux Toolkit.

2.2 Методи та моделі в інформаційних управляючих системах і технологіях

У розробці інформаційних управляючих систем протягом останніх років дедалі більшої популярності набуває підхід багатошарової монолітної архітектури, що поєднує в собі простоту розгортання та високий рівень модульності внутрішньої реалізації. Саме такий шлях обрала і я в RefuConnect, реалізувавши систему як моноліт із чітким розмежуванням відповідальностей на чотири основні шари як Presentation, Application, Domain та Infrastructure. Така структура узгоджується з принципами Domain-Driven Design та сучасними практиками DevOps, забезпечуючи прозорість трасованості викликів, формалізацію контрактів між компонентами й одночасно зберігаючи єдиний артефакт для розгортання.

Шар Presentation у системі є тим місцем, де кожен користувацький рух і кожна точка контакту з додатком набувають конкретної форми, і саме тому особливу увагу було приділено вибору технологій та підходів для його побудови. Фронтенд реалізовано за допомогою React 18, який забезпечує декларативний опис інтерфейсу та розвинені можливості оптимізації через сучасні механізми, такі як concurrent rendering і Suspense. Проте головним інструментом стилізації та уніфікації візуальної мови став MUI Design [28]. Тому що ця бібліотека дозволяє

миттєво створювати доступні й адаптивні компоненти згідно з Material Design-стандартами, використовуючи ThemeProvider для централізованого визначення палітри кольорів, типографіки та просторових відступів. Було використано styled API, щоб максимально зберегти консистентність теми та при цьому мати можливість тонко підлаштовувати вигляд окремих компонентів під потреби проєкту. Усе це поєднується з потужною організацією стану через Redux Toolkit, де createSlice генерує зрозумілі редюсери та екшени, а createAsyncThunk відповідає за асинхронні запити до сервера. Кожен компонент ізольовано інкапсулює свою логіку через кастомні хуки. Цей підхід дозволяє розділяти відповідальність, тримати UI передбачуваним і полегшує тестування як юнітами, так і e2e-сценаріями. Також було використано React.lazy і Suspense для динамічного розбиття бандлів на дрібніші чанки, що пришвидшує початкове завантаження, а memo-представлення складних таблиць і графіків гарантують плавний рендеринг навіть за великої кількості даних.

Шар Application, реалізований на базі ASP.NET Core 8 Web API і служить надійним мостом між Presentation і бізнес-логікою. Його внутрішня будова ретельно продумана відповідно до принципів чистої архітектури. Кожен контролер виконує лише дві функції. Перша - система приймає HTTP-запит, декорований атрибутами [HttpGet], [HttpPost], [Route] і за потреби [Authorize], друге - це валідує вхідні DTO, використовуючи DataAnnotations [29]. Всі невідповідності припиняються на рівні middleware, яке перехоплює ModelStateErrors і формує об’єкт ProblemDetails за стандартом RFC 7807 [30]. Тому клієнт отримує чітко структуровану інформацію про помилки ще до звернення до бізнес-сервісів. Далі контролер делегує обробку кожного запиту відповідному сервісу через інтерфейси, ін’єковані через вбудований DI-контейнер, що полегшує написання модульних тестів із замінними моками та fake-реалізаціями. Особливу увагу було приділено pipeline-фільтрам, які на ранніх стадіях обробки запиту реєструють кореляційний ідентифікатор, перевіряють політики CORS, авторизацію через JWT-токени. Для потреб документації та тестування сторонніх споживачів API підключено Swagger UI та автоматичну

генерацію OpenAPI-специфікацій, які оновлюються при кожному ребілді, даючи змогу отримати актуальний контракт без додаткових зусиль [31].

У надрах додатка розташований доменний шар, сконструйований за найкращими практиками Domain-Driven Design, де кожен агрегат виконує роль чітко визначеного сегмента бізнес-логіки [32]. Агрегати HelpRequest, HelpOffer та HelpAssignment побудовані таким чином, щоб об’єднати основні сценарії. HelpRequest має методи для зміни статусу, додавання коментарів і перевірки валідності даних. HelpOffer відповідає за реєстрацію та верифікацію профілів волонтерів. HelpAssignment відповідає за призначення допомоги, зачіпаючи питання розкладення завдань та обмежень за географією та часом. Ці сутності спілкуються між собою через добре проєктовані контракти, а всі ключові зміни фіксуються як Domain Events, що дозволяє в перспективі підключити будь-який брокер повідомлень або перейти на моделей event sourcing без потреби переписувати ядро. Незмінні дані, такі як контактні дані чи часові слоти, оформлено як Value Objects із власними внутрішніми валідаторами, що унеможливлює створення некоректних або неповних станів. Бізнес-правила інкапсульовані всередині агрегатів.

Шар Infrastructure відповідає за фізичне збереження та доступ до даних. Він побудований на базі Entity Framework Core 8 та SQL Server 2022 [20, 33]. Для маппінгу між доменними моделями та таблицями було використовувано автомапер, що дозволяє точно налаштовувати типи стовпців, зв’язки «один-до-багатьох», «багато-до-багатьох» та політики каскадного видалення [34]. Політика ON DELETE RESTRICT забезпечує недоторканість довідникових таблиць, що критично для відтворюваності історії взаємодій у сфері гуманітарної допомоги. Весь транзакційний цикл завершується викликом SaveChangesAsync, що об’єднує зміни в єдиний атомарний блок.

Окрему увагу приділено організації процесів безперервної інтеграції та доставки. За допомогою GitHub Actions автоматично запускаються юніт та інтеграційні тести на кожному пул-реквесті, виконується статичний аналіз коду через SonarCloud та перевірка якості фронтенд-пакетів через ESLint і Prettier

[35-36]. Після успішного проходження всіх перевірок артефакти пакуються в Docker-образ.

Процес створення HelpAssignment проходить так, що сервіс спочатку перевіряє актуальність обраної заявки і стан волонтера, після чого через dbContext.Database.BeginTransactionAsync() відкриває транзакцію, виконує оновлення статусів та створення нового запису, слідом викликає SaveChangesAsync(), а в разі успіху CommitAsync(). Лише після успішного фіксування даних у базі повертатиметься клієнту відповідь зі статусом Pending. Така атомарність операцій виключає можливість виникнення неконсистентного стану між різними сутностями й забезпечує надійність роботи системи в умовах високої навантаженості.

Складні запити до бази реалізовано за допомогою поєднання Specification Pattern і Query Object. Кожна специфікація інкапсулює окрему умову фільтрації. Ці специфікації динамічно комбінуються для формування кінцевого LINQ-виразу всередині Query Object, що дозволяє повторно використовувати бізнес-умови, спрощувати їх тестування й уникати дублювання коду [37].

Поперечні аспекти, такі як управління крос-доменною взаємодією для кожного HTTP-запиту, винесені до проміжного програмного забезпечення. Це створює єдину точку контролю для всіх запитів, спрощує централізоване налаштування заголовків відповіді та уніфіковане логування траєкторій викликів у розподіленому середовищі.

Делегований вхід через Google OAuth 2.0 мінімізує потребу в підтримці власної інфраструктури для аутентифікації. Усі етапи розробки та розгортання автоматизовані в CI/CD-конвеєрі GitHub Actions

Найважливішим науково-методичним результатом описаного підходу є доведення того, що моноліт із багатошаровою архітектурою може зберігати баланс між експлуатаційною надійністю та готовністю до розвитку. Завдяки чіткому розділенню відповідальностей, інкапсуляції бізнес-правил у домені та автоматизації DevSecOps-процесів, створено платформу, яка вже зараз може бути розширена модулями машинного навчання для прогнозування черги запитів або геопросторовими алгоритмами оптимізації маршрутів доставки допомоги.

РОЗДІЛ 3. РОЗРОБЛЕННЯ ТА РЕАЛІЗАЦІЯ ВЕБ-ПЛАТФОРМИ ДЛЯ ПІДТРИМКИ БІЖЕНЦІВ

3.1 Проєктування структури бази даних веб-платформи

3.1.1 Інфологічна (логічна) модель бази даних

Проєктування бази даних інформаційної системи розпочалося з побудови інфологічної моделі, що охоплює основні сутності предметної області. Запити на допомогу, пропозиції, користувачів, категорії допомоги, статуси виконання та локації. Метою цього етапу було створити гнучку, масштабовану і логічно узгоджену структуру, яка б забезпечувала стабільне функціонування системи в умовах реального навантаження.

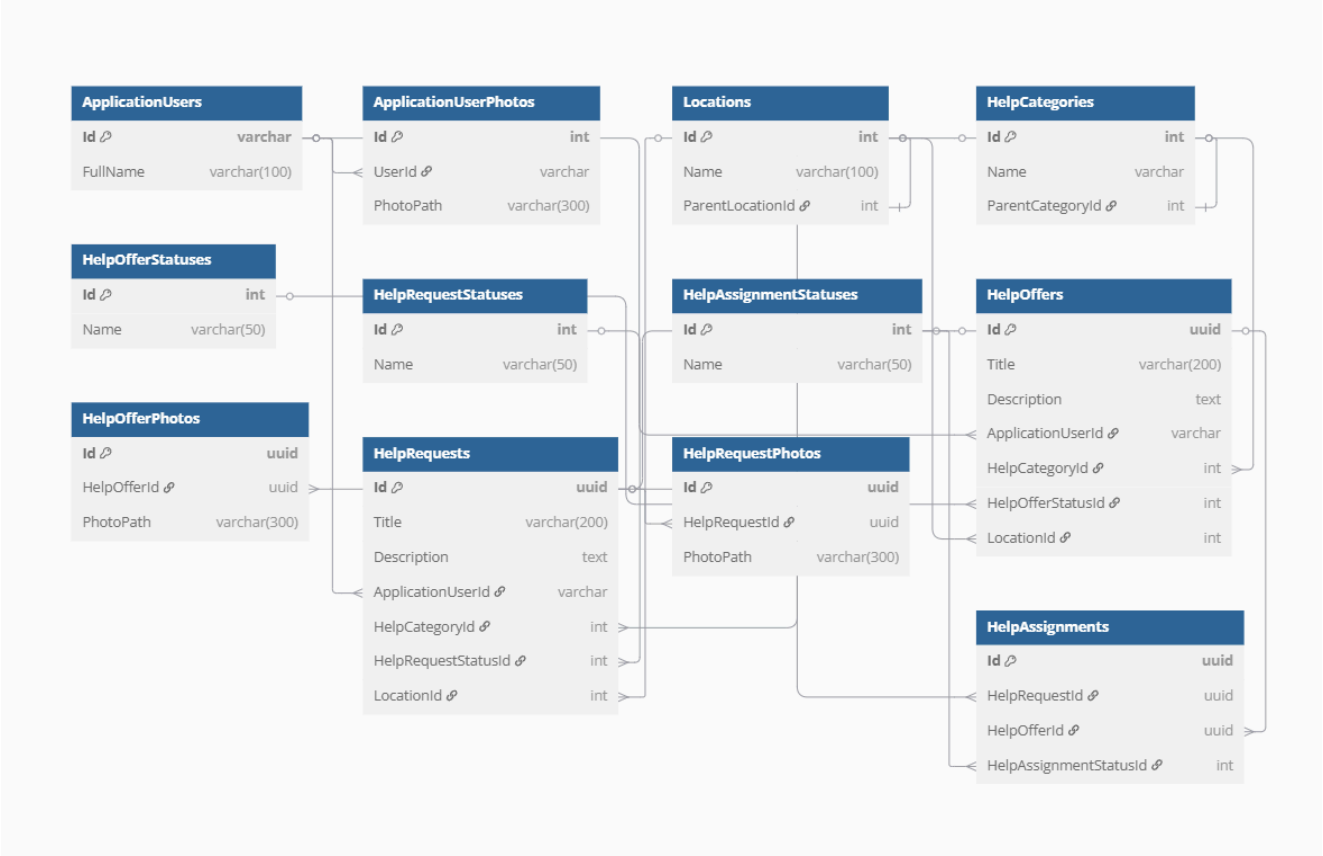


Рисунок 3.1 – Загальна структура бази даних платформи підтримки для біженців

Джерело: розроблено автором самостійно

Центральною сутністю в моделі бази даних виступає ApplicationUser - це узагальнене уявлення про користувача платформи, яке реалізується шляхом розширення стандартного класу IdentityUser, що входить до складу ASP.NET Core Identity. Такий підхід дозволяє зберігати не лише базові атрибути на кшталт логіна, електронної пошти, номера телефону чи гешованого пароля, але й вводити додаткові властивості, необхідні для функціонування платформи в гуманітарному контексті. Зокрема, до таких властивостей можна віднести ім’я, прізвище, стать, дату народження, країну походження, мову спілкування тощо. Це дозволяє гнучко моделювати профіль користувача відповідно до його ролі, як біженець, волонтер, адміністратор.

Важливими є зв’язки ApplicationUser з іншими сутностями. Через зовнішні ключі реалізовано взаємозв’язки типу "один-до-багатьох" з таблицями HelpRequest та HelpOffer. Це означає, що один користувач може створювати кілька запитів або пропозицій допомоги, і в кожному з них зберігається посилання на ініціатора.

Окремо варто виділити зв’язок із сутністю ApplicationUserPhoto, яка має зв’язок типу "один-до-одного" з користувачем. Вона відповідає за зберігання метаінформації про профільне зображення користувача, зокрема шляху до файлу на сервері чи в хмарному сховищі. Така декомпозиція дає змогу краще управляти файлами, реалізовувати політики кешування, а також забезпечити гнучке масштабування у разі зростання кількості користувачів.

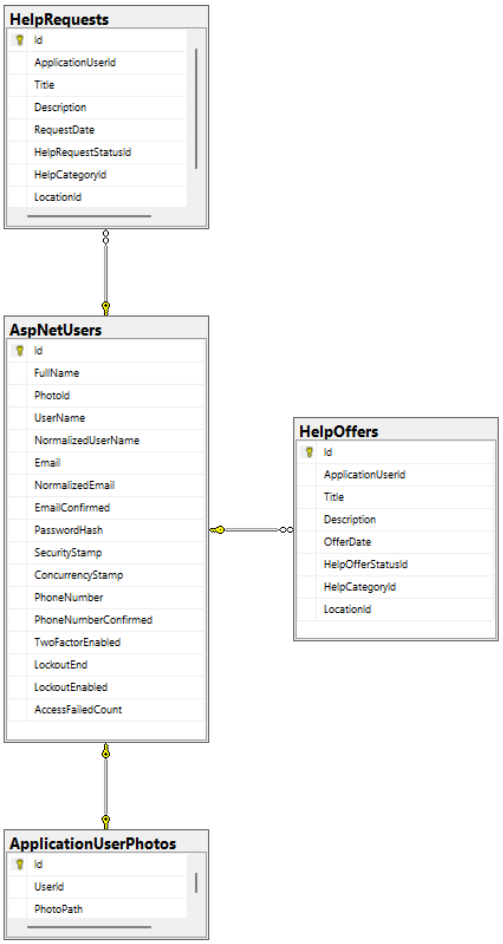


Рисунок 3.2 – Фрагмент діаграми з фокусом на користувача

Джерело: розроблено автором самостійно

Сутності HelpRequest та HelpOffer - це запити й пропозиції допомоги, які створюють користувачі залежно від своєї ролі. Вони мають подібну структуру, поля назви, опису, дати створення, статусу, географічної локації, а також категорії допомоги. Зв’язки реалізовано з такими таблицями як Location, багаторівнева структура місць, де підтримуються як основні регіони, так і підлокації, що дозволяє деталізувати запит, HelpCategory - це категоризація за типом допомоги , підтримує ієрархію через зв’язки батьківської і дочірньої категорій.

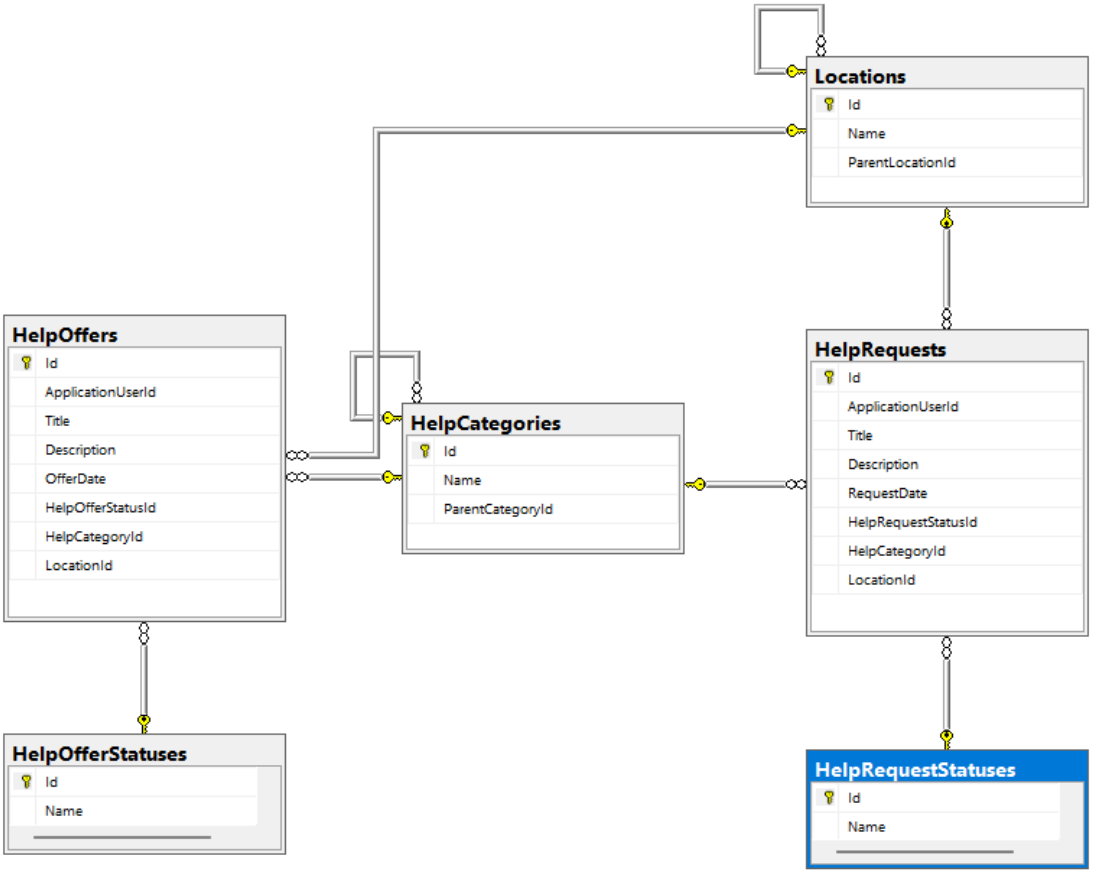


Рисунок 3.3 – Візуалізація HelpRequest та HelpOffer у взаємозв’язку з категоріями, локаціями, статусами

Джерело: розроблено автором самостійно

HelpRequestStatus і HelpOfferStatus - окремі довідники статусів із заздалегідь визначеними станами.

Фотографії, які прикріплюються до запитів і пропозицій, зберігаються у відповідних таблицях HelpRequestPhoto і HelpOfferPhoto, які містять шлях до файлу та зовнішній ключ на основну сутність.

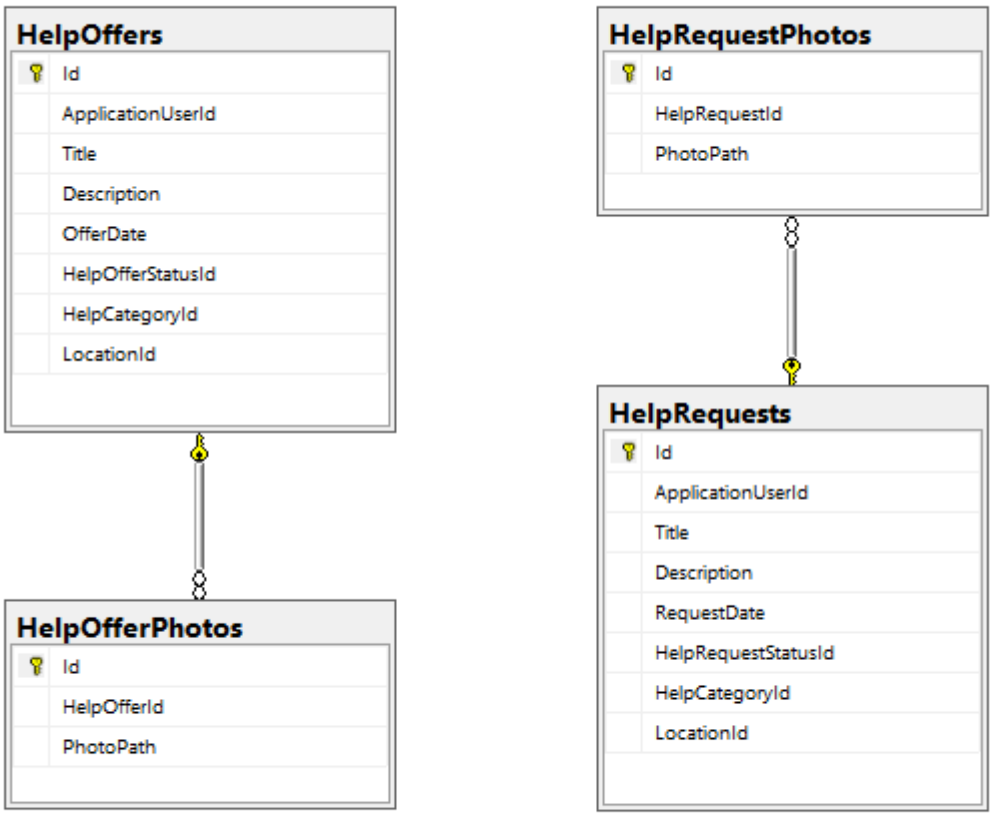


Рисунок 3.4 – Візуалізація фото-зв’язків

Джерело: розроблено автором самостійно

Особливу роль у логіці взаємодії відіграє таблиця HelpAssignment. Вона забезпечує зв’язок між запитом і пропозицією. Наприклад, коли волонтер вирішує допомогти певному біженцю. Тут також є посилання на таблицю HelpAssignmentStatus, яка фіксує статус виконання.

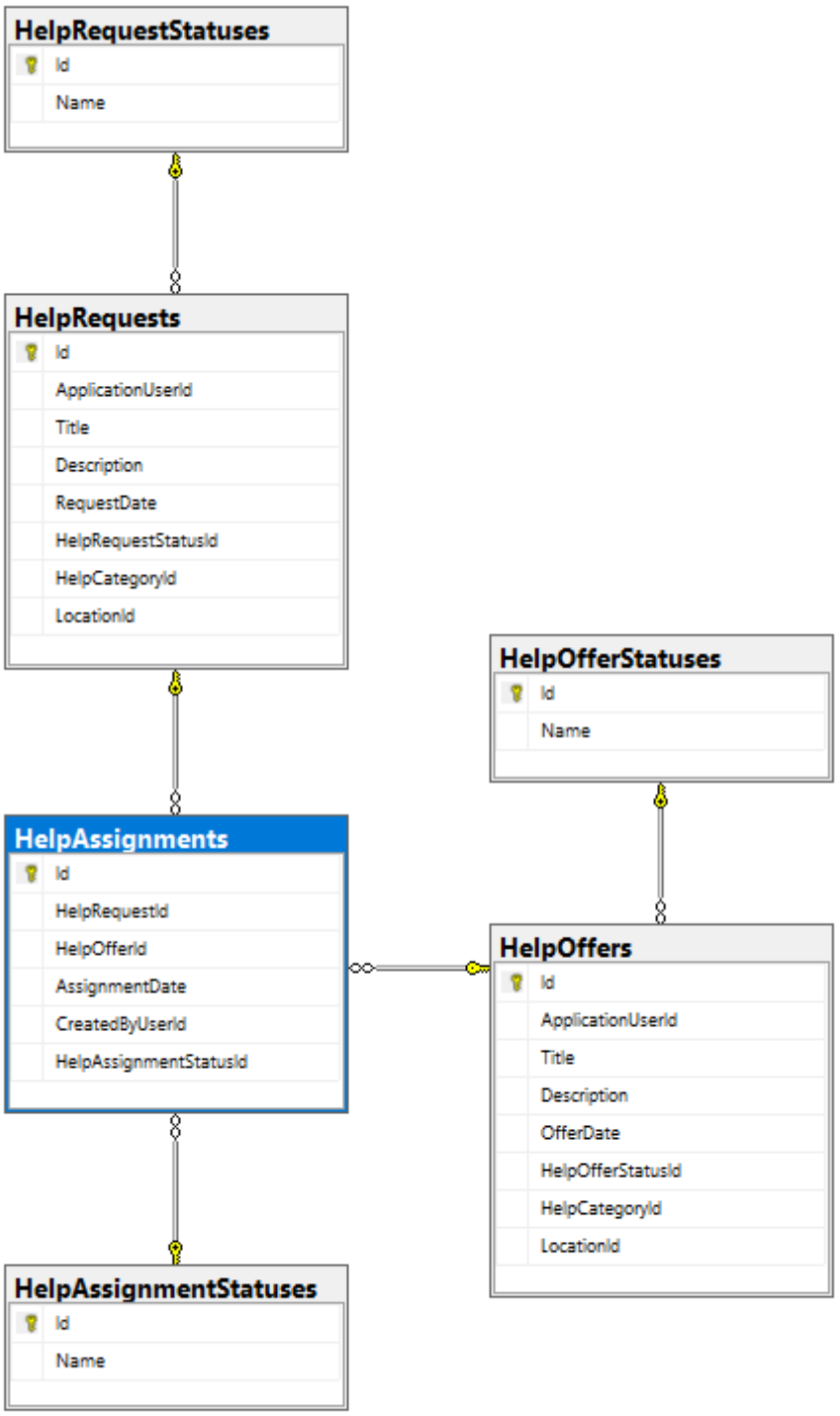


Рисунок 3.5 – Таблиця HelpAssignment та її зв’язки

Джерело: розроблено автором самостійно

Для забезпечення консистентності даних у розробленій системі особливу увагу приділено налаштуванню зовнішніх ключів між таблицями. Кожен зовнішній ключ має чітко визначену поведінку при операціях видалення, яка

відповідає логіці функціонування системи та вимогам збереження історичних даних. Зокрема, для менш критичних зв’язаних об’єктів, таких як фотографії користувача, реалізовано каскадне видалення. Це дозволяє автоматично очищати залежні ресурси при видаленні основного запису, уникнувши залишкових сирітських записів у базі даних. Водночас, для важливих об’єктів, як-от запити на допомогу або пропозиції, що мають значення для архівування та аналітики, використовується обмежувальна поведінка. Це забезпечує збереження історії взаємодій навіть після деактивації чи видалення користувача, зберігаючи цінні дані для оцінки ефективності роботи платформи та її розвитку.

Зведена логічна модель включає ключові сутності як користувачі, запити про допомогу, пропозиції допомоги, категорії послуг, географічні локації, медіа-ресурси, статуси виконання, а також таблиці для забезпечення зв’язків між об’єктами. Усі сутності і зв’язки спроєктовано з урахуванням принципів нормалізації, зокрема досягнуто третьої нормальної форми, що дозволяє уникнути надлишковості, забезпечити цілісність даних та спростити супровід структури в майбутньому.

Кожен атрибут є атомарним і відповідає вимогам однозначності та несуперечності. Було проведено ретельну перевірку функціональних залежностей, і всі нетривіальні залежності приведено до ключових. Це дозволило мінімізувати можливість аномалій при оновленні чи вставці даних та забезпечити логічну чистоту моделі.

Особлива увага приділена оптимізації структури під пошук, сортування та фільтрацію. Для цього проаналізовано основні сценарії взаємодії користувача з інтерфейсом, зокрема, пошук запитів за категоріями, статусами та локацією, фільтрація актуальних або завершених ініціатив, відображення історії взаємодій. На підставі цього проведено індексацію критично важливих полів, таких як ідентифікатори користувачів, географічні координати, статуси запитів і пропозицій. Це дозволяє суттєво скоротити час виконання запитів і забезпечити стабільну продуктивність системи навіть при значному навантаженні.

Загалом, сформована інфологічна модель повністю відповідає функціональним і нефункціональним вимогам платформи підтримки біженців.

Вона стала надійною основою для реалізації фізичної моделі бази даних, адаптованої до умов подальшого розширення системи, забезпечення захищеного зберігання чутливих даних, а також підтримки масштабованості, відмовостійкості та високої доступності платформи.

3.1.2 Обґрунтування вибору СКБД та проєктування фізичної моделі

Під час розробки інформаційної системи платформи підтримки біженців ключовими критеріями вибору системи керування базами даних були як надійність трансакційної моделі, так і масштабованість, висока продуктивність під час виконання паралельних запитів, простота інтеграції з інструментами розробки та адміністрування, а також сумісність із технологічним стеком .NET.

На основі аналізу функціональних, технічних та експлуатаційних критеріїв для зберігання даних було обрано Microsoft SQL Server як основну систему керування базами даних. Це рішення виявилося оптимальним для платформи, яка потребує високої стабільності, гнучкості у розширенні, а також гарантій збереження чутливої інформації.

Серед ключових переваг SQL Server варто відзначити повну підтримку транзакцій, що дозволяє виконувати складні операції з даними атомарно, гарантує узгодженість і знижує ризик виникнення логічних помилок при паралельному доступі до даних. Це критично важливо в умовах багатокористувацького середовища, де щосекунди можуть виконуватися сотні взаємозалежних операцій.

СКБД також надає широкі можливості роботи з індексами, включно з кластеризованими, некластеризованими, фільтрованими та колоночними індексами, що дозволяє значно підвищити продуктивність запитів, особливо при фільтрації великих наборів даних або побудові звітів у режимі реального часу. До цього додаються вбудовані механізми оптимізації запитів, які дозволяють автоматично формувати ефективні плани виконання на основі статистики.

Крім того, SQL Server має розвинену систему адміністративного контролю за допомогою SQL Server Management Studio (SSMS), Extended Events, Profiler та інших засобів адміністратор може детально відстежувати навантаження, виявляти «вузькі місця» в роботі БД, аналізувати блокування, логіку транзакцій та інші аспекти продуктивності [38].

Окрему увагу приділено засобам безпеки, серед яких багаторівневе управління доступом, шифрування даних як на рівні таблиць, так і на рівні транспортного каналу, аудит операцій з чутливою інформацією та підтримка інструментів аутентифікації через Active Directory [39]. Важливо й те, що SQL Server підтримує масштабовану схему збереження даних, що дозволяє ефективно працювати як у локальному середовищі, так і в хмарних інфраструктурах із використанням реплік, розділення навантаження та резервного копіювання.

Завдяки цим характеристикам Microsoft SQL Server забезпечує стабільну роботу системи, захист критичних даних та високу продуктивність навіть у складних умовах функціонування платформи підтримки біженців.

Фізична модель бази даних проєктувалася з урахуванням майбутнього зростання навантаження та дотримання принципів ефективної нормалізації. Кожна таблиця має первинні ключі, зовнішні ключі з каскадною поведінкою, а критичні поля індексуються для оптимізації операцій фільтрації, пошуку та з’єднань між таблицями.

Довгі текстові поля, що зберігають описи запитів та пропозицій, реалізовані як типи NVARCHAR(MAX) з обмеженням на введення з боку застосунку, аби уникнути надлишкових витрат на зберігання. Таблиці, що містять часто використовувану інформацію містять кластерні індекси для покращення продуктивності при виконанні запитів зі складною фільтрацією. З метою підвищення продуктивності, журнал транзакцій розміщується на окремому фізичному носії.

Уся структура БД узгоджена із третьою нормальною формою, що дозволяє уникати надмірностей, забезпечує узгодженість, цілісність та логічну зв’язаність усіх сутностей.

3.1.3 Контрольний приклад та обрахунок прогнозованих обсягів

Для обґрунтування обраної структури бази даних, а також оцінки потенційного навантаження на інформаційну систему в умовах її експлуатації, було розроблено контрольний приклад. Він охоплює базові сутності як користувачі, запити на допомогу та пропозиції допомоги. Контрольний приклад дозволив зафіксувати середній обсяг даних на один запис у кожній з ключових колекцій, а також спрогнозувати загальне навантаження на СКБД в умовах реального використання платформи.

Для кожної сутності були створені тестові екземпляри, що відтворюють типову структуру з відповідними полями. Наприклад, користувач включає такі поля, як ім’я, електронна адреса, номер телефону, дата створення, роль, статус активності та URL до фотографії профілю. У середньому такий запис займає приблизно 220-250 байтів, залежно від довжини текстових полів. Для запитів про допомогу та пропозицій допомоги структура включає заголовок, опис ситуації, категорію, дату створення, статус, локацію та посилання на медіа, що в середньому становить 500-600 байтів на документ. При цьому важливо підкреслити, що безпосередньо медіа-файли не зберігаються в самій базі даних, а розміщуються у зовнішньому об’єктному сховищі. У самій базі даних зберігаються лише текстові метадані, що зменшує навантаження на систему зберігання та оптимізує час доступу до даних.

Враховуючи цільову аудиторію платформи, було спрогнозовано очікувані обсяги даних у довгостроковій перспективі. Зокрема, орієнтовно передбачається до 100 тисяч зареєстрованих користувачів, з яких кожен потенційно може створити хоча б один запит про допомогу або пропозицію. На основі аналізу подібних систем і гуманітарних платформ було прийнято припущення про наявність приблизно 80 тисяч запитів про допомогу та 40 тисяч пропозицій. Це дозволяє сформувати первинний прогноз кількості записів у кожній колекції на етапі повноцінного розгортання системи.

На базі зібраної інформації було здійснено обрахунок загального обсягу зберігання для кожної сутності. Колекція користувачів за вказаної кількості записів і середнього розміру одного документа займає близько 22-25 мегабайт. Запити на допомогу, найбільша за обсягом колекція займуть орієнтовно 45-50 мегабайт, а пропозиції допомоги, ще близько 20-25 мегабайт.. Також частина простору буде зарезервована під системні лог-файли, журнал транзакцій, тимчасові таблиці, а в подальшому на історію змін та архівні дані.

Таким чином, загальний початковий обсяг бази даних, включаючи всі основні сутності та супутні дані, складає орієнтовно 90-100 мегабайт. Проте з урахуванням природного зростання системи, додавання нових функціональних модулів, збільшення активності користувачів, накопичення логів, періодичного збереження резервних копій і журналів змін, рекомендовано передбачити запас щонайменше до 500 мегабайт для забезпечення стабільної роботи системи в середньо та довгостроковій перспективі. Це дозволить уникнути критичних ситуацій, пов’язаних із нестачею ресурсів на рівні СКБД, а також забезпечить безперервність надання сервісу навіть у пікові періоди навантаження.

Проведене оцінювання підтверджує правильність вибору документно-орієнтованої моделі даних, зокрема у середовищі Microsoft SQL Server, яке дозволяє працювати як із табличною структурою, так і з JSON-полями, якщо буде потрібно зберігати частково гнучкі записи. Обрана структура відповідає вимогам щодо ефективності доступу до даних, швидкості фільтрації, сортування, а також забезпечує високу продуктивність при пошуку за індексованими полями. У разі зростання навантаження можливе як вертикальне масштабування, так і горизонтальне масштабування з реплікацією бази для підвищення доступності та швидкості обробки запитів.

Таким чином, контрольний приклад та прогнозовані обсяги дозволяють зробити висновок про придатність обраної структури даних до реальних умов експлуатації. Система готова до масштабування, зберігаючи при цьому високу стабільність і швидкодію в умовах зростаючого попиту з боку цільової аудиторії.

3.2 Реалізація логіки призначення

Підсистема призначення допомоги є серцем платформи. Саме тут заявка біженця зустрічається з волонтерською пропозицією, після чого сторона отримувача отримує контакти іншої сторон, після вони можуть перейти до особистого спілкування. Уся логіка побудована так, аби ця зустріч була атомарною, безпечною і такою, що не допускає дублювань або помилок.

У сховищі даних для цього використано сутність HelpAssignment. Вона має зовнішні ключі на HelpRequest і HelpOffer, а також на довідник статусів HelpAssignmentStatus. У фрагменті конфігурації моделі кожен зв’язок оголошено з видаленням Restrict, завдяки чому історія взаємодій не зникає навіть під час каскадних операцій над заявками чи оферами.

Точка входу у бек-енд це маршрут /api/HelpAssignment, який обробляє POST-запит. Контролер, захищений атрибутом [Authorize]. Він одразу ж дістає ідентифікатор автентифікованого користувача через UserManager.GetUserId(User) і делегує всю подальшу роботу сервісному шару. Таким чином роль контролера зведено до мінімуму. Він лиш перевіряє авторизацію й валідність моделі, а бізнес-правила не «закопані» в HTTP-рівень, а зібрані в сервісній логіці.

Сервіс HelpAssignmentService формує об’єкт призначення, присвоює йому стартовий статус Pending, фіксує автора операції і передає готову сутність у репозиторій CreateAsync. Після збереження вона одразу проєктується назад у DTO і повертається клієнтові у тілі відповіді. Саме тут прописані й правила переходу станів. Коли призначення перейшло в Completed, автоматично закриваються пов’язані заявка і офер, при InProgress обидві сторони фіксуються у стані «в процесі».

Репозиторій інкапсулює роботу з EF Core й виконує складне фільтрування, сортування та пагінацію. Він «підтягує» всі необхідні навігаційні властивості через Include і далі ThenInclude, використовує AsNoTracking для колекційних запитів і вміє будувати динамічні Where-вирази за будь-якою комбінацією

параметрів. Це дозволяє фронтенду запитувати лише релевантні призначення без додаткових витрат на мережевий трафік чи серверну пам’ять.

На клієнтському боці логіка зібрана у thunk addHelpAssignment та slice helpAssignmentSlice. Після успішного запиту POST нове призначення миттєво потрапляє в Redux-стан, а глобальний спінер вимикається, щоби користувач одразу побачив результат. Асинхронні операції винесено у файл helpAssignmentThunks.ts, де для кожної з них за допомогою утиліти createThunk генерується типізований action-creator ї, а HTTP-запити інкапсульовано у невеликому API-шарі.

На практиці процес виглядає так. Коли користувач натискає кнопку «Прийняти», інтерфейс формує DTO і викликає thunk. Axios автоматично додає до заголовка JWT-токен і відправляє POST на бек-енд. Контролер перевіряє підпис, через сервіс передає об’єкт у репозиторій, де він потрапляє до транзакції SQL Server’а. Після коміту база повертає згенерований Guid, а відповідь 200 OK разом із першим статусом Pending прилітає назад у браузер. Стан Redux оновлюється, компонент React перерендерюється, а користувач бачить toast із підтвердженням успіху.

Особливу увагу приділено безпеці. Усі маршрути захищені ролями «Helper» та «Refugee», авторизація відбувається через JWT у зв’язці з Google OAuth.

Реалізована логіка призначення допомоги забезпечує прозорий та надійний життєвий цикл від моменту надсилання запиту до фінального закриття. Вона поєднує сувору типізацію, транзакційну цілісність і масштабовану архітектуру, з якої легко розвивати нові можливості, такі як push-нотифікації та ML-рекомендації, що автоматично підбиратимуть найкращі офери чи запити під конкретні потреби користувачів.

3.3 Розробка програмного забезпечення платформи

У даному розділі послідовно викладено процес реалізації веб-платформи для підтримки біженців. Від формування логіки взаємодії користувачів до розробки інтерфейсів і підготовки експлуатаційної документації. Всі етапи проєктування й реалізації програмного забезпечення виконувались відповідно до принципів модульності, масштабованості, тестованості та відповідності сучасним вимогам безпеки й зручності користування.

3.3.1 Проєктування програмного забезпечення

При проєктування серверної частини веб-платформи для підтримки біженців було обрано підхід «чистої архітектури» з інверсією залежностей і шаровою структурою, що забезпечує високу модульність і тестованість рішення. Центром конфігурації слугує файл Program.cs, у якому формується DI-контейнер, реєструються контексти БД, сервіси, репозиторії, механізми аутентифікації та політики безпеки [40].

Підключення до бази даних сконфігуровано викликом:

```
builder.Services.AddDbContext<RefuConnectContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"
)));
```

це гарантує узгодженість EF Core-схеми й можливість міграцій без втручання в код сервісів. Для керування користувачами й ролями «волонтер» та «біженець» було застосовано ASP.NET Identity у поєднанні з JWT-аутентифікацією [41]:

```
builder.Services.AddIdentity<ApplicationUser, IdentityRole>()
    .AddEntityFrameworkStores<RefuConnectContext>()
```

```
    .AddDefaultTokenProviders();

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options => {... });
```

CRUD-операції для сутностей, такі запити допомоги, пропозиції, користувачі, фото інкапсулюються у відповідних репозиторіях, зареєстрованих у контейнері:

```
builder.Services.AddScoped<IHelpRequestRepository,
HelpRequestRepository>();
builder.Services.AddScoped<IHelpRequestService, HelpRequestService>();
// аналогічно для HelpOffer, UserPhoto, Location
```

сервіси ж згуртовують репозиторії, виконують валідацію вхідних даних та формують DTO для контролерів [42].

Для локального розроблення і тестування фронтенду, реалізованого на React, визначено CORS-політику:

```
builder.Services.AddCors(o => o.AddPolicy("CorsPolicy", p =>

p.AllowAnyHeader().AllowAnyMethod().WithOrigins("http://localhost:3000")));
```

документацію REST API автоматизовано через Swagger:

```
builder.Services.AddSwaggerGen(c => {
    c.SwaggerDoc("v1", new OpenApiInfo { Title = "RefuConnect API", Version
= "v1" });
    // безпека через Bearer-Auth
});
```

Цикл ініціалізації даних викликається при старті, створюючи базові ролі, категорії та локації, що спрощує розгортання у новому середовищі.

Контейнеризація через Docker забезпечує ідентичність розробницького та продакшн-середовищ. Образ API містить ASP.NET Core, автоматичне виконання міграцій і налаштований порт 80/443, а docker-compose.yml піднімає залежні сервіси. CI/CD на GitHub Actions охоплює збірку, запуск юніт-тестів, валідацію

конфігурацій і автоматичний деплой. Це дозволяє виявляти проблеми ще на стадії Pull Request і гарантує стабільність продакшн-релізів.

Серверна логіка побудована з акцентом на чітке розділення обов'язків, високу розширюваність і простоту підтримки. Кожен компонент, від контексту EF Core до сервісів обробки запитів і пропозицій допомоги відповідає за свою відповідальність, що значно полегшує подальший розвиток платформи в умовах зростання навантаження та нових вимог до функціональності.

3.3.2 Вибір архітектури програмного забезпечення

При проєктуванні архітектури веб-платформи для підтримки біженців було визначено низку критичних вимог, таких як простота розгортання, можливість горизонтального масштабування, чітке розділення відповідальності між компонентами, а також забезпечення високої надійності та безпеки. Після аналізу монолітної та мікросервісної моделей прийнято рішення обрати монолітну клієнт-серверну архітектуру з розбиттям на логічні шари, що суттєво пришвидшує розробку і знижує складність CI/CD.

Платформа складається з трьох основних шарів:

- на рівні представлення фронтальна частина архітектури реалізована за допомогою HTTP API побудованого на основі технології ASP.NET Core. Цей рівень відповідає за прийом HTTP-запитів, що надходять із клієнтського застосунку, розробленого з використанням бібліотеки React. Контролери, розміщені на цьому рівні, відповідають за маршрутизацію запитів, здійснення початкової валідації вхідних даних, а також за передачу управління відповідним сервісам бізнес-логіки. З метою поліпшення зручності тестування та документування API було інтегровано Swagger;
- рівень бізнес-логіки відповідає за обробку основних сценаріїв взаємодії користувача із системою. Логіка взаємодії реалізована у вигляді сервісів. Кожен з них виконує окрему функціональну роль, наприклад автентифікація

користувачів за допомогою JWT або Google OAuth 2.0, створення і керування запитами про допомогу або пропозиціями, управління особистими профілями користувачів та їх ролями. Використання інтерфейсів для опису сервісів дозволяє дотримуватись принципів слабкого зв’язування. Це є важливим для забезпечення масштабованості, гнучкості й можливості модульного тестування. Такий підхід також ізолює бізнес-логіку від інфраструктурних залежностей, підвищуючи стабільність і надійність системи;

- рівень доступу до даних реалізований з використанням шаблонів проєктування Repository та Unit of Work. Вони дозволяють організувати централізоване керування транзакціями та спрощують логіку роботи з даними. Для кожного доменного об'єкта створено відповідні репозиторії, які здійснюють взаємодію з контекстом Entity Framework Core. Останній конфігурується на рівні запуску застосунку в Program.cs, що забезпечує централізоване управління залежностями. Система підтримує використання LINQ-запитів для ефективного вилучення та маніпуляції даними, а також керує процесами міграцій для актуалізації структури бази даних. Забезпечено підтримку транзакцій, що дозволяє гарантувати цілісність і консистентність даних при виконанні складних бізнес-операцій.

Усі поперечні, що не пов’язані безпосередньо з бізнес-логікою, але впливають на роботу всієї системи були грамотно винесені в Middleware-компоненти ASP.NET Core [43]. Це дозволяє централізовано керувати ключовими механізмами захисту, конфігурації запитів і контролю доступу, а також значно спрощує супровід і тестування програми, оскільки логіка обробки HTTP-запитів структурована і прозора.

Одним із таких компонентів є налаштування CORS. Це політика, яка регулює, з яких доменів дозволено звертатися до API. У процесі розробки застосовується конфігурація, що дозволяє доступ із локального середовища (http://localhost:3000), тоді як у продакшн-режимі дозволяються лише чітко визначені авторизовані домени. Це захищає сервер від небажаних крос-доменних запитів і несанкціонованого доступу зі сторонніх ресурсів, що є ключовим у контексті захисту персональних даних.

Аутентифікація та авторизація в системі реалізовані через перевірені стандарти JWT. Після успішного входу користувача як через стандартну форму логіну, так і через інтеграцію Google OAuth 2.0 сервер формує цифровий підписаний токен, у якому закодовані ID користувача, його роль, права доступу та термін дії. Цей токен зберігається у сторі.

При кожному запиті клієнта до API Middleware перевіряє наявність токена, правильність його підпису, чинність строку дії, а також витягує інформацію про роль користувача. На основі цього застосовується політика доступу. Якщо роль або права не відповідають вимогам конкретного ресурсу, запит блокується на рівні середнього шару без навантаження контролерів.

Крім того, для кожного API-контролера або окремих методів використовуються атрибути авторизації ([Authorize(Roles = "Admin")], [AllowAnonymous], тощо), що забезпечують детальне розмежування доступу відповідно до ролі користувача. Це дозволяє чітко вказати, які частини системи можуть бути використані тільки адміністраторами, які волонтерами, а які біженцями або неавторизованими відвідувачами.

Хоча проєкт реалізовано як єдиний монолітний контейнер, така архітектура не обмежує систему у масштабованості чи гнучкості. В основі архітектурного рішення лежить принцип багаторівневого розподілу обов’язків і дотримання чіткої модульності, що дозволяє ефективно підтримувати та розвивати систему в умовах зростаючих вимог і навантажень.

Монолітна реалізація зібрана у вигляді одного цільового артефакту контейнера, який включає серверну частину та всі внутрішні сервіси, необхідні для функціонування системи. Проте навіть у межах моноліту архітектура враховує можливість горизонтального масштабування. У разі підвищення навантаження, наприклад, при великій кількості одночасних запитів від користувачів, передбачено запуск декількох однотипних інстансів цього контейнера за балансувальником навантаження. Такий підхід дозволяє рівномірно розподіляти вхідний трафік, забезпечуючи стабільну продуктивність і уникнення "вузьких місць".

Окрім того, архітектура враховує масштабування на рівні бази даних. У разі значного збільшення обсягу інформації або зростання кількості транзакцій можливе впровадження механізмів реплікації або шардінгу. Це дозволить розподіляти навантаження між кількома екземплярами SQL Server, підвищити доступність даних і знизити час відповіді на критичні запити. Таким чином, хоч на початковому етапі проєкт працює в умовах централізованої БД, його структура передбачає плавний перехід до масштабованої інфраструктури без необхідності повного перепроєктування системи.

Для локальної розробки, тестування та налагодження використано docker-compose як стандартне рішення для створення ізольованого середовища. Завдяки цьому одночасно піднімаються контейнери для API-серверу, Microsoft SQL Server, а також клієнтської частини, реалізованої на React. Такий підхід не тільки автоматизує процес підготовки середовища, але й гарантує консистентність між розробкою та продакшн-оточенням, мінімізуючи ймовірність конфігураційних помилок.

Сама архітектура вирізняється низкою вагомих переваг, які зумовили її вибір для проєкту. Передусім, це простота розгортання, яка забезпечується завдяки єдиному контейнеру, що легко розгортається як у хмарі, так і на локальному сервері. Наступною важливою перевагою є чітке логічне розділення відповідальностей у межах архітектури, таких як інтерфейс користувача, API-логіка, бізнес-сервіси й доступ до бази даних мають своє окреме представлення в кодовій структурі, що сприяє полегшенню супроводу та розширенню функціоналу. Завдяки цьому команда розробки може паралельно працювати над різними модулями, зберігаючи цілісність продукту.

Крім того, архітектура забезпечує високу надійність, оскільки мінімізує кількість точок відмови, та безпеку, зокрема завдяки ізоляції компонентів у контейнерах, суворому контролю доступу до БД і застосуванню сучасних протоколів автентифікації й шифрування. Також важливо відзначити, що обраний підхід забезпечує готовність системи до подальшого розвитку. У разі потреби моноліт легко можна модульно декомпонувати або поступово перевести окремі функціональні блоки на мікросервісну архітектуру.

Таким чином, обрана монолітна архітектура із багаторівневим логічним розділенням забезпечує стійкий баланс між швидкістю розробки, простотою експлуатації та здатністю до масштабування. Вона є оптимальним вибором на поточному етапі проєкту, коли пріоритетом є швидке впровадження, стабільність роботи та мінімізація складності інфраструктури без втрати потенціалу до росту.

3.3.3 Інструментальні засоби розробки

При створенні веб-платформи для підтримки біженців у двох ролях «волонтер» та «біженець» було використано набір сучасних інструментів і технологій, які забезпечують високу продуктивність, зручність налагодження, гнучкість у розширенні та чітке розділення відповідальностей між компонентами системи.

Серверна частина платформи розроблена на базі .NET 8 із використанням мови програмування C# 11, що забезпечує сучасний синтаксис, покращену продуктивність і підтримку новітніх мовних конструкцій. Розробка здійснювалася в середовищі Visual Studio 2022, яке надало повний набір інструментів для зручної та ефективної роботи, включно з інтелектуальним автодоповненням коду (IntelliSense), інтерактивним дебагером, вбудованим профілюванням продуктивності та інтеграцією з системами контролю версій [44-45]. Конфігурація застосунку виконувалась у файлі Program.cs, що відповідає новій моделі хостингу ASP.NET Core. У ньому послідовно реєструються сервіси, репозиторії, політики безпеки, middleware-компоненти та параметри залежностей, формуючи повноцінний контейнер інверсії керування.

Для взаємодії з базою даних використовується Entity Framework Core 8.0 з підходом Code-First, що дозволяє проектувати доменну модель безпосередньо в C# і автоматично генерувати міграції. Це забезпечує швидке створення, оновлення та підтримку схеми бази без необхідності ручного написання SQL-запитів. Потужні можливості LINQ-запитів активно використовуються для реалізації

фільтрації, сортування та пагінації даних у запитах і пропозиціях допомоги, що суттєво спрощує логіку на рівні API.

Механізми аутентифікації та авторизації побудовано на основі ASP.NET Core Identity, інтегрованої з JWT-Bearer Authentication та Google OAuth 2.0. Це дає змогу поєднувати локальну автентифікацію з можливістю входу через Google-акаунт, зберігаючи при цьому високий рівень безпеки. Такий підхід дозволяє гнучко розширювати модель ролей та обмежувати доступ до окремих маршрутів на рівні атрибутів контролерів, зменшуючи кількість повторень логіки перевірки прав вручну.

Усі ключові параметри безпеки, такі як строки підключення, секретні ключі для підпису токенів, політики CORS, налаштування тривалості сесій та дозволених доменів, винесені до централізованих файлів конфігурації appsettings.json та Program.cs. Це спрощує обслуговування системи, дозволяє легко змінювати середовище без повторної збірки застосунку та забезпечує кращу читаність і структурованість проекту. Загалом, така архітектура забезпечує стабільність, масштабованість і безпеку, що є критичним для платформи, яка працює з чутливими даними в гуманітарному контексті.

З метою перевірки коректності обробки даних і підтримки регресійних тестів створено набір юніт та інтеграційних тестів на xUnit із використанням Moq для імітації залежностей репозиторіїв і сервісів [46-47]. Розгортання аспектів cross-cutting в middleware забезпечило однорідність поведінки API та зниження дублювання коду.

Клієнтська частина платформи реалізована як односторінковий застосунок з використанням React 18 у поєднанні з TypeScript, що забезпечує статичну типізацію та дозволяє виявляти потенційні помилки ще на етапі розробки. Такий підхід значно підвищує надійність та передбачуваність поведінки коду, особливо у великих багатокомпонентних проєктах. Для реалізації маршрутизації всередині додатку використовується React Router v6, що дає змогу ефективно керувати переходами між сторінками без перезавантаження, у тому числі з підтримкою захищених маршрутів [48]. Центральне керування станом здійснюється за допомогою Redux Toolkit, зокрема його API createSlice і createAsyncThunk, які

дозволяють спрощено описувати редюсери та керувати асинхронними викликами до бекенду. Візуальне оформлення інтерфейсу реалізовано на основі Material UI, що забезпечує гнучку підтримку тем, адаптивність до мобільних пристроїв і швидке складання UI з готових компонентів. Для підтримки єдиного стилю коду, уникнення синтаксичних помилок і зручності командної роботи у фронтенд-частину проєкту інтегровано ESLint і Prettier, які автоматично перевіряють і форматують код відповідно до обраних стандартів [35-36].

Автоматизацію процесів CI/CD у проєкті реалізовано за допомогою GitHub Actions, що забезпечує послідовне виконання сценаріїв на кожен push або pull-request у репозиторій [24, 49]. Пайплайн включає перевірку коду, тестування, збірку та публікацію артефактів. Для локальної розробки та тестування застосовується Docker Compose, який дозволяє піднімати повноцінне середовище з контейнерами для API, бази даних SQL Server та інших необхідних сервісів, завдяки чому досягається ідентичність між локальним, тестовим і продакшн-оточенням. Такий підхід дозволяє уникати проблем, пов’язаних із «ефектом різного середовища», та спрощує пошук і виправлення помилок. У продакшен-оточенні рекомендовано використовувати Kubernetes, який дає змогу автоматизовано масштабувати сервіси залежно від навантаження, балансувати трафік між репліками, а також розгортати оновлення, що забезпечує безперервну доступність системи навіть під час випуску нових версій [50].

Для тестування й документування REST-інтерфейсів було застосовано Postman, зберігаючи колекції запитів та використовуючи Newman у рамках CI для контролю стабільності ендпоінтів [51-52]. Адміністрування бази даних здійснювалося через SQL Server Management Studio, що полегшило виконання складних запитів і оптимізацію індексів.

Завдяки ретельному добору інструментів розробки та комплексній інтеграції DevOps-практик проєкт досяг високої швидкості випуску релізів, стабільної продуктивності під час пікових навантажень і гнучкості в подальшому розвитку функціоналу, зокрема для нових сценаріїв підтримки біженців і волонтерів.

3.3.4 Тестування програмного забезпечення

Забезпечення якості і стабільності функціонування створеного програмного забезпечення здійснювалося завдяки впровадженню комплексної багаторівневої стратегії тестування. Вона охоплювала юніт-тести, інтеграційні перевірки, сценарії наскрізного тестування.

На етапі модульного тестування окремий сервіс і репозиторій супроводжувався відповідним набором юніт-тестів, реалізованих із використанням фреймворка xUnit. Зовнішні залежності, такі як контекст бази даних, сервіси автентифікації та зовнішні API імітувалися за допомогою бібліотеки Moq. Це дозволило ізолювати тестовану бізнес-логіку. Для кожного тестованого класу створювався окремий клас із методами, назви яких відображали тестові умови та очікувану поведінку. Використання strict mock дозволяє ефективно виявляти спроби доступу до неініціалізованих об’єктів і тим самим підвищувати точність перевірки.

Інтеграційне тестування здійснювалося шляхом об’єднання Web API з in-memory-контекстом Entity Framework Core. Цей підхід надавав змогу тестувати повну логіку взаємодії контролерів, middleware-компонентів і конфігурацій автентифікації без залучення реальної бази даних. Застосування TestServer із пакету дозволяло запускати у пам’яті повноцінне веб-середовище, надсилати HTTP-запити та перевіряти коректність відповіді. Особливу увагу приділяла перевірці доступу до захищених маршрутів як для авторизованих, так і для неавторизованих користувачів.

На наступному рівні реалізовано набір наскрізних тестів, які моделювали типову поведінку користувача у браузер. Від автентифікації до створення реквестів, оферів, завантаження фото тощо.

Крім того, до конвеєра CI/CD було інтегровано регулярне виконання колекцій Postman із використанням Newman. Це забезпечувало перевірку стабільності API в тестовому середовищі. Автоматично генеровані HTML-звіти

про виконання тестів зберігалися як артефакти збірки, а аналіз покриття коду здійснювався за допомогою Coverlet і ReportGenerator.

Загалом підхід дозволив не лише виявити й усунути купу дефектів ще до розгортання системи, а й сформувати практику автоматизованого тестування, що істотно знижує ризики виникнення критичних помилок у продакшн-середовищі. Крім того, запроваджені стандарти структурування тестового коду створили надійну основу для подальшого розширення системи та підтримки її якості в майбутньому.

3.3.5 Керівництво користувача

Керівництво користувача веб-платформи покликане забезпечити максимальну зручність та ефективність роботи зі системою, що слугує інструментом координації взаємодії між біженцями та волонтерами. У ньому викладено покрокову інструкцію щодо реєстрації, створення та обробки заявок, використання фільтрів пошуку, а також організації безпечної роботи з персональними даними.

Початковий етап взаємодії з платформою полягає в реєстрації користувача. Система підтримує два альтернативні механізми створення облікового запису, як традиційну реєстрацію через електронну пошту та пароль із подальшою верифікацією через JWT-токен, а також швидку авторизацію за допомогою Google OAuth 2.0. У другому випадку користувач натискає кнопку «Увійти через Google», обирає свій акаунт у спливаючому вікні та миттєво потрапляє в систему без потреби додаткового введення пароля. Після успішної аутентифікації облікові дані генеруються та зберігаються відповідно до найсуворіших стандартів безпеки, а користувача автоматично перенаправляють на головну сторінку.

Персональний кабінет є центральним елементом взаємодії та представляє собою інтегровану консоль управління, що складається з наступних компонентів:

- розділ «Профіль» дозволяє змінювати особисті дані, оновлювати фотографію, переглядати історію входів та налаштовувати параметри облікового запису;
- розділ «Мої заявки або пропозиції» відображає створені користувачем запити на допомогу для біженця та власні пропозиції для волонтера, надаючи можливість їх редагувати або видаляти;
- розділ «Підтвердження» відображає надіслані чи отримані згідно ролі запити чи пропозиції.

Усі ці компоненти реалізовані з використанням адаптивного інтерфейсу на базі React і TypeScript, що гарантує належний рівень швидкодії та доступність з будь-яких пристроїв.

Для формування нового запиту або пропозиції користувач обирає відповідний пункт меню в персональному кабінеті. Далі слідує заповнення форми, що містить поля для заголовка, детального опису, вибору категорії допомоги і вказівки географічної локації. Додатково можна прикріпити одну або кілька фотографій, які будуть збережені на сервері й відображені в картці заявки. Після натискання кнопки «Опублікувати» запит стає видимим у загальному каталозі, де його можуть переглянути волонтери або біженці залежно від ролі.

На окремій частині платформи реалізовано розгорнуті інструменти фільтрації - за категоріями, статусом, географічним розташуванням і датою публікації. Користувач може деталізовано налаштувати запити, щоб швидко знайти релевантні пропозиції чи заявки. Після вибору конкретного запису відкривається детальна картка із повною інформацією та контактними даними автора.

Статус кожного запиту змінюється відповідно до етапів надання допомоги. Ці зміни відображаються як у картках на загальних сторінках, так і в особистому кабінеті. Відповідні сповіщення надсилаються користувачам-учасникам процесу, що гарантує оперативність обміну інформацією про хід виконання.

Весь обмін даними між клієнтським застосунком і сервером відбувається через зашифроване з’єднання HTTPS. Аутентифікація реалізована на основі JWT із короткостроковими токенами доступу. Альтернативна авторизація через Google

OAuth 2.0 дозволяє користувачам уникнути ризику зберігання паролів на стороні платформи. Крім того, застосовані серверні політики CORS, захист від CSRF, XSS та SQL-ін’єкцій, що гарантує високий рівень інформаційної безпеки.

Для оперативного вирішення запитань було інтегровано кілька каналів підтримки. Розділ FAQ містить відповіді на найчастіші питання. Спеціальна електронна адреса призначена для детальних запитів.

Користувачі можуть самостійно перевірити консистентність функцій платформи, створивши тестові заявки, взаємодіючи між різними акаунтами та відстежуючи коректність зміни статусів.

3.4 Визначення технічного забезпечення

3.4.1 Обґрунтування вибору технічного забезпечення

Розроблення веб-платформи, орієнтованої на підтримку біженців і волонтерів, потребувало ретельного вибору технічного забезпечення, яке гарантує безперервну роботу, високу доступність сервісів і можливість масштабування при збільшенні кількості користувачів. Платформа включає реєстрацію та автентифікацію користувачів, створення запитів і пропозицій допомоги, персоналізований кабінет, зберігання медіа та взаємодію з базою даних тощо. Це зумовлює необхідність стабільної серверної інфраструктури, низької затримки при обробці запитів і підтримки багатопотокової роботи.

Серверна частина, яка реалізує бізнес-логіку на основі ASP.NET Core потребує високої обчислювальної потужності для обробки API-запитів, генерації JWT-токенів, виконання перевірок прав доступу та обслуговування взаємодії з фронтендом. Для цієї мети рекомендовано використання процесора класу AMD EPYC 7302P або еквівалент, який забезпечує багатозадачність і стабільну продуктивність при високому навантаженні. Обсяг оперативної пам’яті становить не менше 32 GB DDR4 з перспективою масштабування до 64 GB. Для систем зберігання даних доцільним є використання двох NVMe SSD по 1TB у RAID 1. Це

гарантує одночасно високу швидкість читання та запису і відмовостійкість. Сервер функціонує під керуванням Windows Server 2022 Standard, який сумісний з усіма компонентами .NET стеку. Для забезпечення високої доступності й резервування обрано дві мережеві карти Ethernet 10 Gbps.

Сервер бази даних, на якому працює Microsoft SQL Server, виконує критично важливі операції. Наприклад, збереження запитів, пропозицій, користувацьких профілів, журналів активності, медіафайлів тощо. Тому для забезпечення стабільної та швидкої роботи СКБД було обрано процесор AMD EPYC 7402P з 24 ядрами, частотою 2.8 GHz, що оптимізовано для інтенсивного використання в базах даних і обчислювальних навантаженнях. Обсяг оперативної пам’яті обирається 64GB ECC DDR4 з можливістю подальшого масштабування до 128 GB. Це відповідає потребам в оперативному кешуванні запитів та обробці великих обсягів даних. Система зберігання має 2 x 1TB NVMe SSD в RAID 1. Вони забезпечують як надійність, так і швидкий доступ до даних. Серверна ОС обрана під назвою Windows Server 2022 Datacenter. Вона дає змогу управляти масштабованими віртуальними середовищами, при цьому гарантуючи стабільність роботи сервісів. Як і для додатка, мережевий інтерфейс організований на основі двох адаптерів 10 Gbps із підтримкою резервування та балансування навантаження.

Мережева інфраструктура також є важливим компонентом надійності системи. Для з’єднання між компонентами було обрано керований комутатор HPE Aruba 2930F або його функціональний аналог із 24-48 портами і підтримкою швидкості 10 Gbps, що забезпечує достатню пропускну здатність для обміну між вузлами. Для захисту, маршрутизації трафіку, підключення віддалених клієнтів та підтримки QoS-засобів застосовується маршрутизатор MikroTik CCR2004, який підтримує VPN, DDoS-захист, резервування каналів і розширені політики безпеки.

Дивлячись на все обране технічне забезпечення платформи забезпечує:

- стабільну роботу під навантаженням;
- високу швидкість обробки запитів;
- безпечну взаємодію з даними;

- можливість масштабування за рахунок горизонтального та вертикального розширення;
- відмовостійкість і підтримку безперервного доступу.

Ці рішення повністю узгоджуються з архітектурною структурою системи, DevOps-практиками та сучасними вимогами до захищеного й надійного веб-сервісу для підтримки соціально вразливих груп населення.

3.4.2 Ресурсні вимоги до технічного забезпечення

Для забезпечення надійної, безперебійної та безпечної роботи веб-платформи необхідно враховувати низку технічних вимог, які охоплюють як клієнтську частину, так і серверну інфраструктуру. Платформа орієнтована на широку аудиторію, включаючи біженців, волонтерів, адміністративний персонал, тому технічні вимоги сформульовані з урахуванням доступності та адаптивності до різних умов використання.

На клієнтському рівні особливу увагу приділено тому, щоб система була доступною навіть для користувачів із мінімальними апаратними ресурсами. Це передбачає оптимізацію інтерфейсу, мінімальне використання важких скриптів, адаптивну верстку, підтримку офлайн-режимів і зменшене споживання трафіку, що критично важливо в умовах поганого або нестабільного інтернет-з’єднання. Основні функції платформи, як-от створення та перегляд запитів про допомогу, фільтрація категорій, заповнення форм, вхід через Google OAuth, доступні навіть при використанні застарілих пристроїв.

Для стабільної роботи на персональних комп’ютерах рекомендується використовувати пристрої з процесорами щонайменше рівня Intel Core i3 або AMD Ryzen 3 починаючи з 8-го покоління, які забезпечують базову обчислювальну потужність для відтворення SPA-додатку, розробленого на основі React. Обсяг оперативної пам’яті повинен бути не менше 4 GB типу DDR4, що дозволить комфортно взаємодіяти з платформою навіть у випадку відкриття

кількох вкладок браузера одночасно. Операційна система має бути актуальною: Windows 10, Windows 11 або Linux-дистрибутив на зразок Ubuntu 22.04 LTS із встановленими оновленнями безпеки. Для зручної роботи з формами та таблицями бажано мати дисплей з роздільною здатністю Full HD, який дозволить уникнути проблем зі скролінгом і масштабуванням.

Системні вимоги до браузерів передбачають використання сучасних версій Google Chrome, Mozilla Firefox або Microsoft Edge, які мають повну підтримку Web API, ECMAScript 2022, Fetch API, Flexbox і CSS Grid. Усе це необхідно для правильного рендерингу динамічного інтерфейсу користувача.

Особливо важливим аспектом є оптимізація для мобільних пристроїв, оскільки значна частина цільової аудиторії має доступ переважно до смартфонів або планшетів. З цієї причини фронтенд платформи спроєктований за принципами Mobile First і адаптований до екранів малого розміру. Мінімальні технічні вимоги для мобільних пристроїв включають операційні системи Android 9.0 або iOS 13 і вище, 2 GB оперативної пам’яті, базовий процесор середнього або бюджетного рівня, а також браузери Chrome Mobile, Safari або Edge із підтримкою JavaScript ES6+ та сучасних CSS-можливостей. Адаптивний дизайн та зменшення навантаження на GPU гарантують працездатність навіть на бюджетних моделях.

На серверному рівні система проектується з урахуванням потенційного розширення масштабів, збільшення обсягів даних та кількості одночасних підключень. Платформа передбачає розгортання у хмарному середовищі, що дозволяє використовувати гнучкі обчислювальні ресурси, масштабуватися відповідно до навантаження, а також забезпечити високу доступність і безперервність сервісу. Основними платформами для хостингу можуть бути Microsoft Azure, Amazon Web Services або Google Cloud Platform. Застосовуються механізми автоматичного масштабування, які динамічно збільшують кількість реплік API або SQL Server у відповідь на збільшення запитів від користувачів. Балансувальники навантаження дозволяють рівномірно розподіляти трафік між усіма інстансами, запобігаючи перевантаженню окремих вузлів.

Для локального розгортання та CI/CD процесів платформа використовує Docker та docker-compose, що дозволяє ізолювати середовище, спростити

конфігурацію та забезпечити передбачувану поведінку застосунку в різних середовищах.

Важливо, що ресурсні вимоги не є статичними, вони масштабуються залежно від розміру аудиторії, активності користувачів та обсягів даних. Наприклад, зростання кількості запитів на допомогу, підвантаження фото та збільшення логістичних операцій з часом вимагатиме додаткових дискових ресурсів і розширення пулу API-серверів. Завдяки гнучкій архітектурі та адаптивному масштабуванню така еволюція системи є передбачуваною та контрольованою.

Отже, сформульовані технічні вимоги до платформи є результатом комплексного підходу до проєктування. Вони поєднують доступність для малопотужних пристроїв із забезпеченням стабільної роботи в умовах високих навантажень. Це дозволяє платформі залишатися ефективним інструментом підтримки вразливих груп населення, гарантуючи зручність, надійність і безпеку на кожному етапі використання.

3.5 Реалізація веб-платформи та перевірка її працездатності

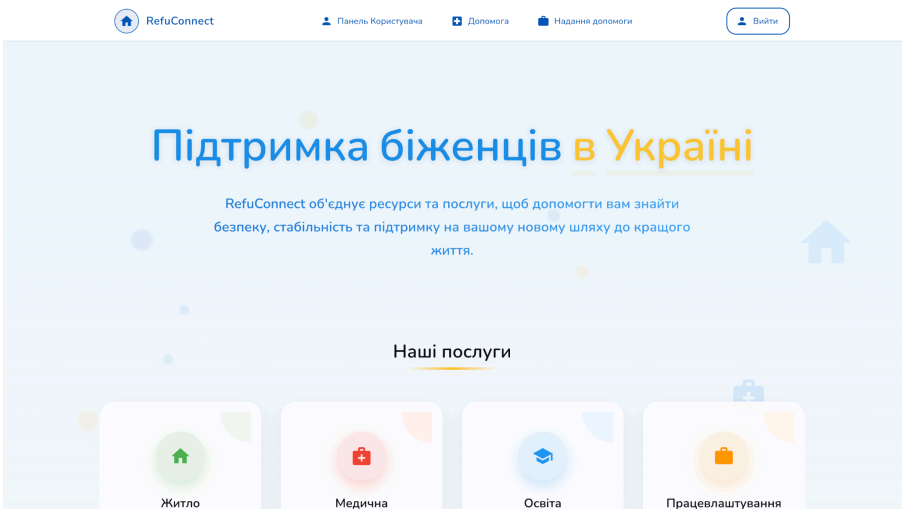


Рисунок 3.6 – Головна сторінка

Джерело: розроблено автором самостійно

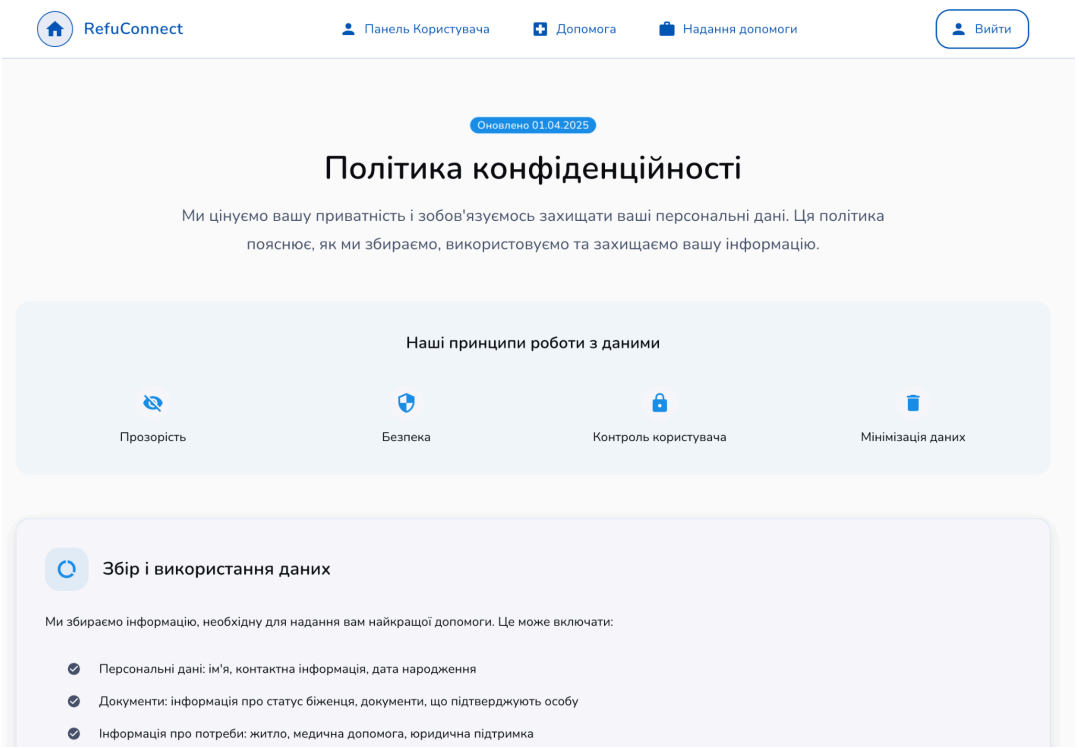


Рисунок 3.7 – Сторінка конфіденційності

Джерело: розроблено автором самостійно

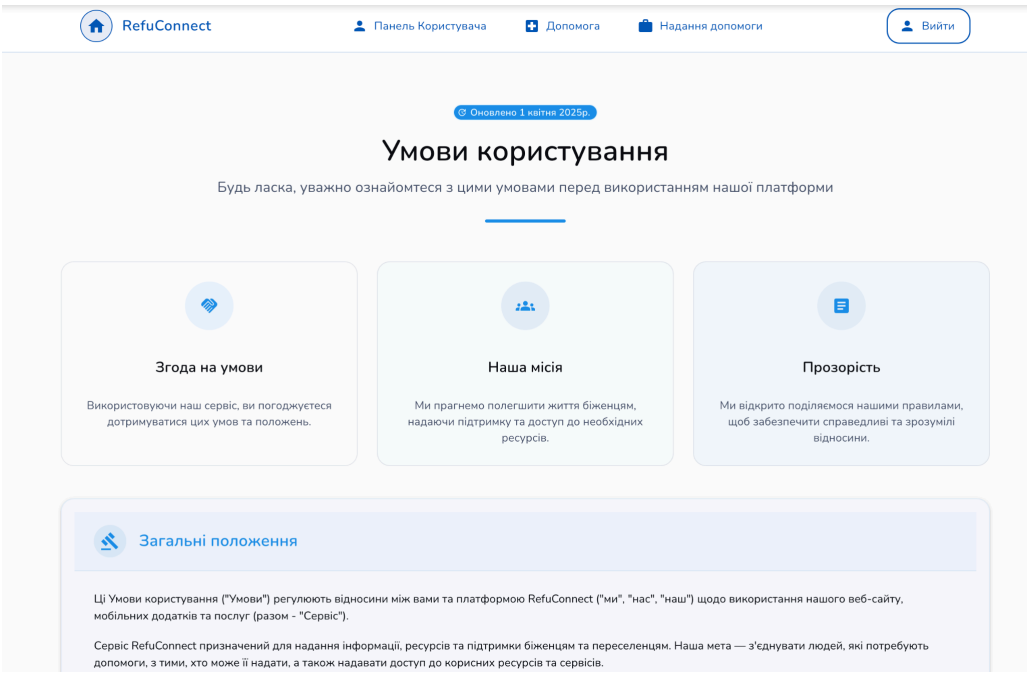


Рисунок 3.8 – Сторінка умов користування

Джерело: розроблено автором самостійно

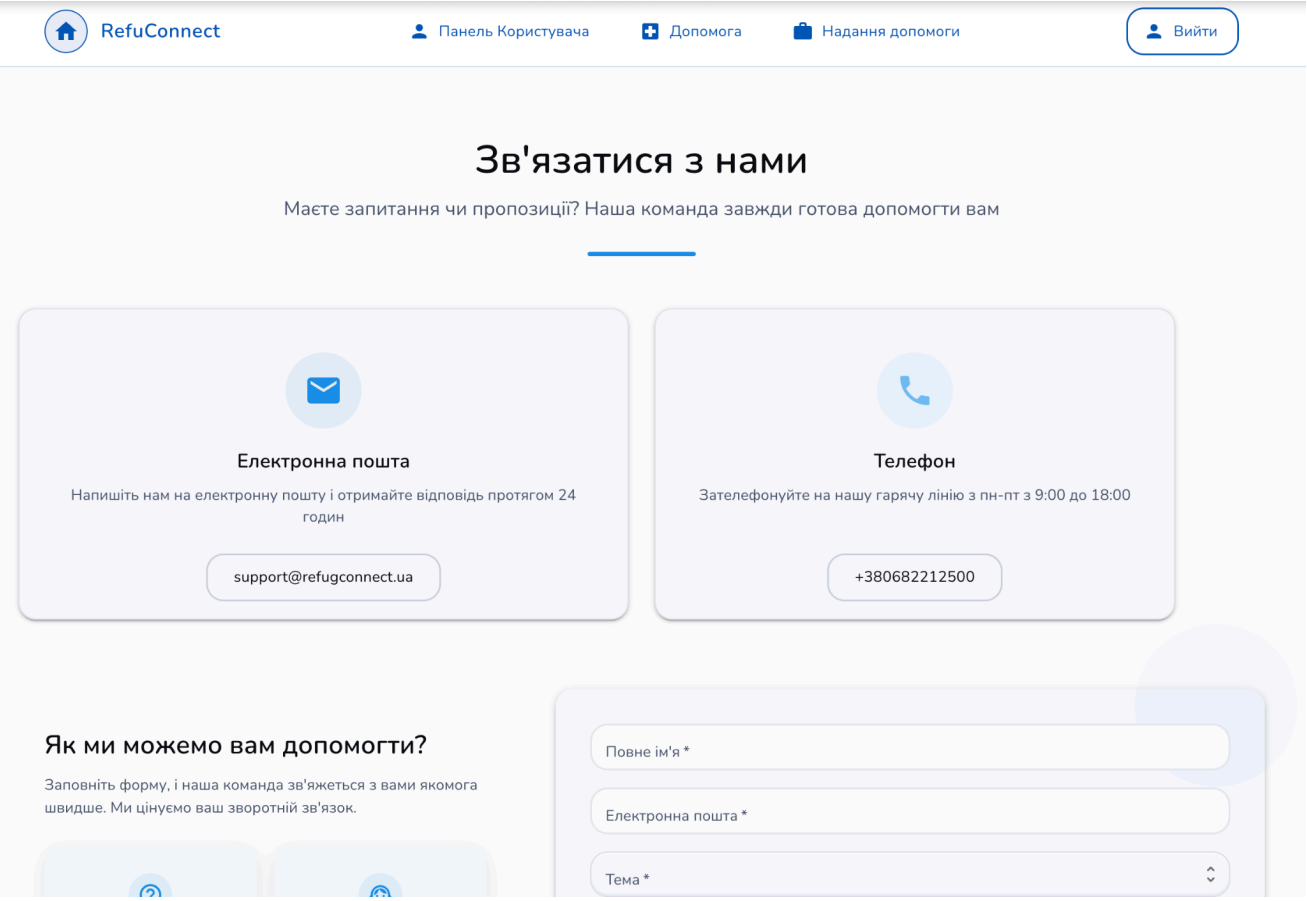


Рисунок 3.9 – Сторінка зв’язатися з нами

Джерело: розроблено автором самостійно

На рисунках 3.6 - 3.9 показано основні інформаційні сторінки веб-платформи, які відіграють важливу роль у забезпеченні прозорості, законності та довіри користувачів до системи. Ці сторінки включають Політику конфіденційності, Умови користування, Контактну інформацію та сторінку з формою зворотного зв’язку.

На сторінці Політики конфіденційності детально описано, які саме персональні дані збираються під час користування платформою, з якою метою це відбувається, а також у яких випадках інформація може передаватися третім сторонам. Сторінка умов користування регламентує порядок взаємодії з платформою, визначає права та обов’язки користувачів, адміністрації та третіх осіб. На контактній сторінці наведено ключові канали зв’язку для отримання допомоги, подання запитів або уточнення технічних деталей. Окремим функціональним блоком є форма зворотного зв’язку, яка інтегрована на сторінці. Вона дозволяє користувачу швидко звернутися до адміністрації, залишивши запит,

пропозицію або повідомлення про проблему. Форма передбачає введення імені, електронної пошти, теми звернення та тексту повідомлення.

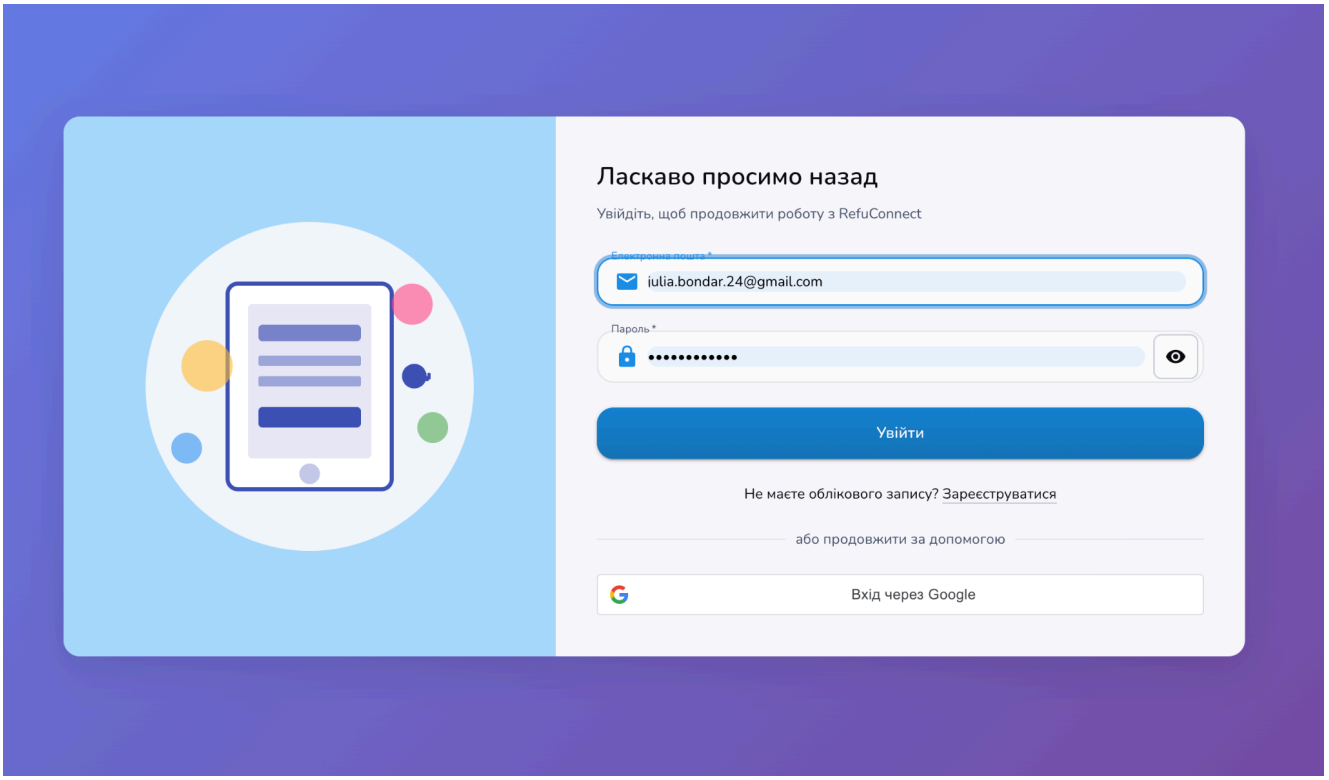


Рисунок 3.10 – Форма логіну

Джерело: розроблено автором самостійно

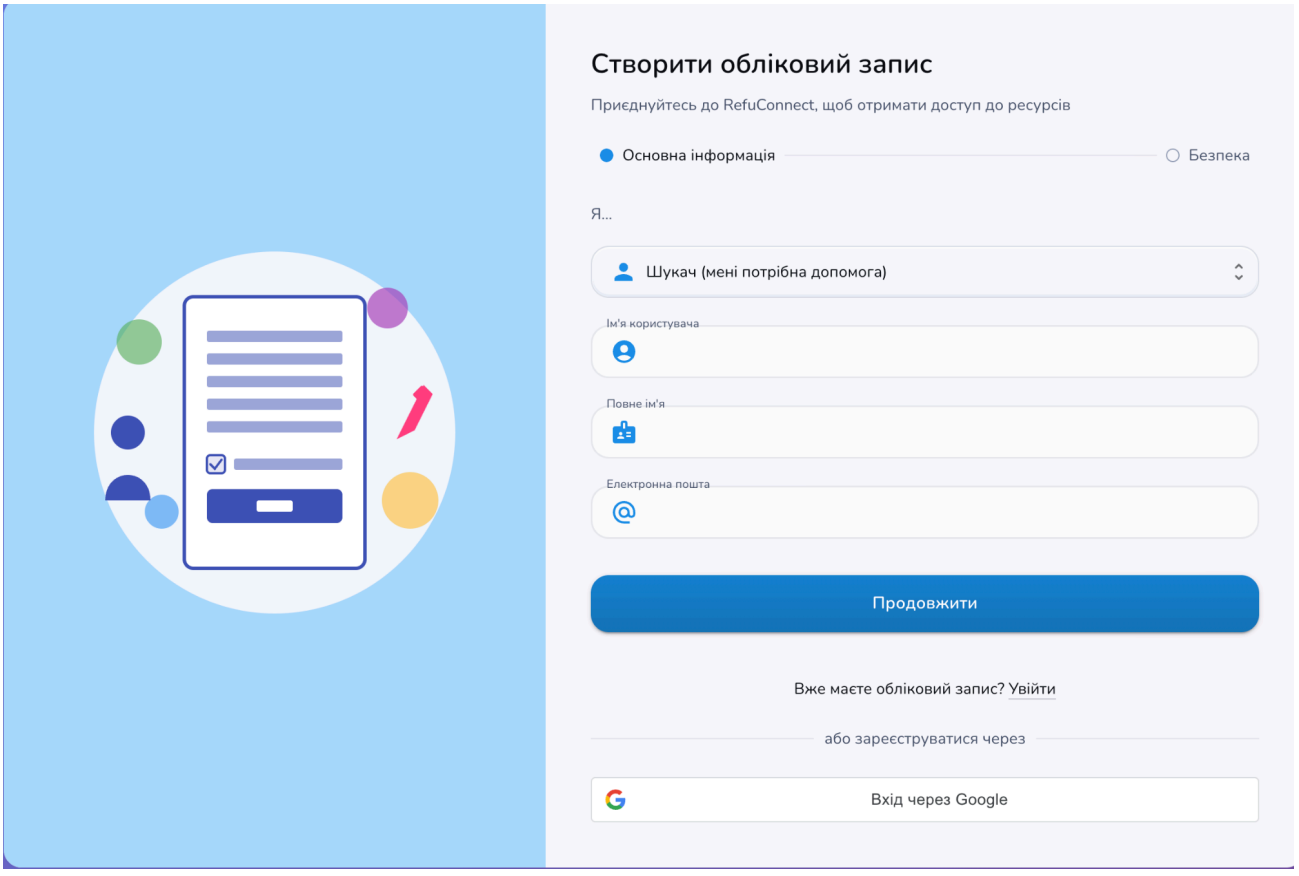


Рисунок 3.11 – Форма реєстрації

На рисунках 3.10 - 3.11 показано ключові інтерфейсні елементи процесу аутентифікації користувачів на веб-платформі, зокрема форми реєстрації та входу. Вони є адаптовані під потреби двох основних ролей - Волонтерів і Біженців. Ці форми є критично важливим елементом загальної структури інформаційної системи, оскільки забезпечують початкову ідентифікацію користувача та налаштування доступу до функціональних можливостей платформи відповідно до його ролі.

Особливістю реалізації є наявність механізму входу через обліковий запис Google, що значно спрощує процес аутентифікації для користувачів, які не бажають щоразу вводити дані. Це рішення не лише покращує користувацький досвід, але й знижує ризик помилок при введенні даних, забезпечує вищий рівень безпеки завдяки двофакторній автентифікації Google, а також дає змогу оперативно інтегрувати основну інформацію користувача у профіль системи.

Джерело: розроблено автором самостійно

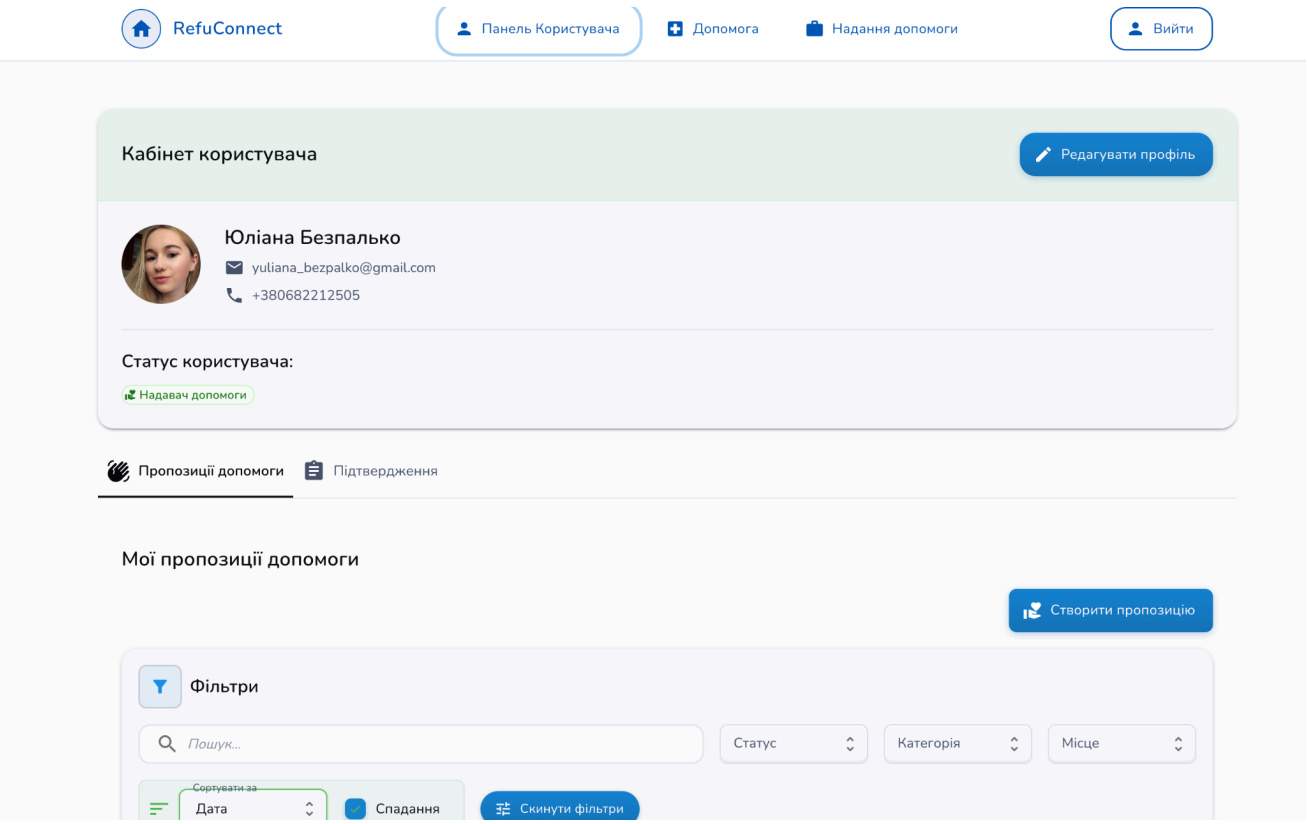


Рисунок 3.12 – Кабінет користувача (Волонтер)

Джерело: розроблено автором самостійно

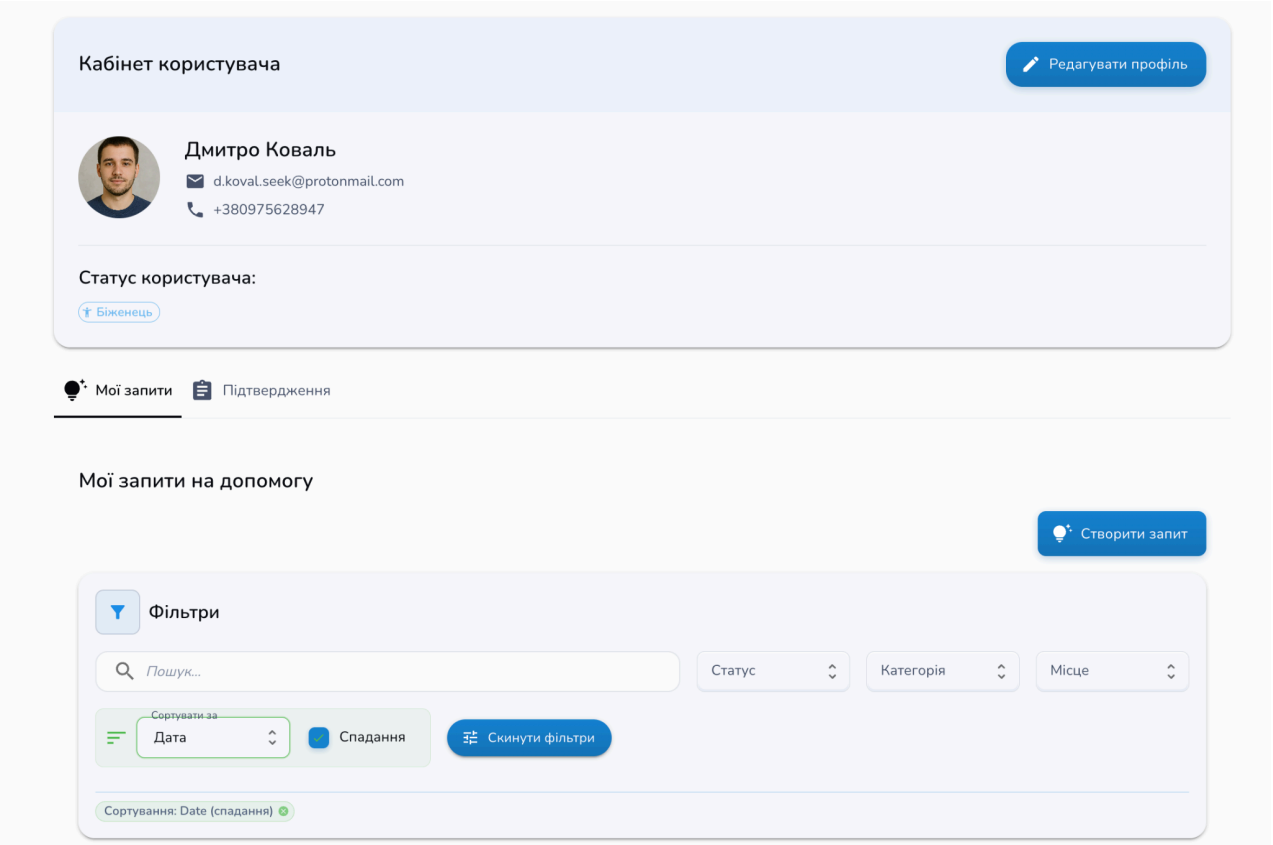


Рисунок 3.13 – Кабінет користувача (Біженець)

Джерело: розроблено автором самостійно

На рисунках 3.12 - 3.13 представлено інтерфейс персонального кабінету користувача в двох ключових ролях - Біженець та Надавач допомоги. Кожна з цих ролей має власний набір функціональних можливостей, адаптованих під відповідний сценарій використання, що було реалізовано з урахуванням принципів зручності, інтуїтивності та ефективності взаємодії.

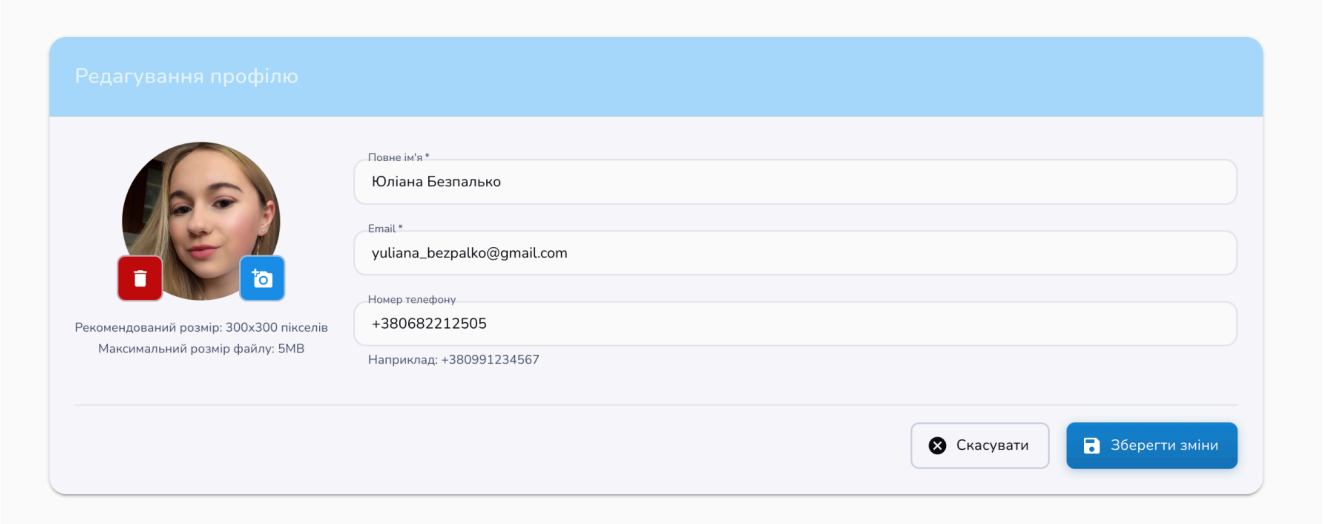


Рисунок 3.14 – Форма редагування профілю

Джерело: розроблено автором самостійно

На рисунку 3.14 зображено інтерфейс форми редагування персональних даних користувача, що є невід’ємним елементом персонального кабінету. Цей функціональний модуль реалізовано з метою забезпечення гнучкості керування особистою інформацією та підтримки актуальності профілю користувача, що особливо важливо в контексті гуманітарної платформи, де достовірність і швидкий зв’язок є критично важливими.

Форма дозволяє користувачеві в будь-який момент змінити повне ім’я, адресу електронної пошти або номер телефону, що є основними каналами комунікації між біженцями та надавачами допомоги. Для зручності використання кожне поле супроводжується підписами, валідаційними повідомленнями у разі введення некоректних даних, а також іконками, які полегшують візуальне сприйняття функціоналу.

Окрему увагу приділено зміні фотографії профілю. Користувач може завантажити нове зображення, яке автоматично адаптується під необхідний формат і зберігається в системі.

Інтерфейс форми побудовано на базі бібліотеки Material UI, що дозволяє забезпечити єдність стилістики та адаптивність компонентів під різні розміри екранів. Усі зміни передбачають попередню перевірку даних, а також підтвердження дій, що мінімізує ризик помилок або випадкового редагування.

Загалом, форма редагування особистих даних виконує важливу роль у підтриманні актуальності інформації в системі, сприяє довірі між користувачами платформи та забезпечує належний рівень самостійності й контролю з боку кінцевого користувача.

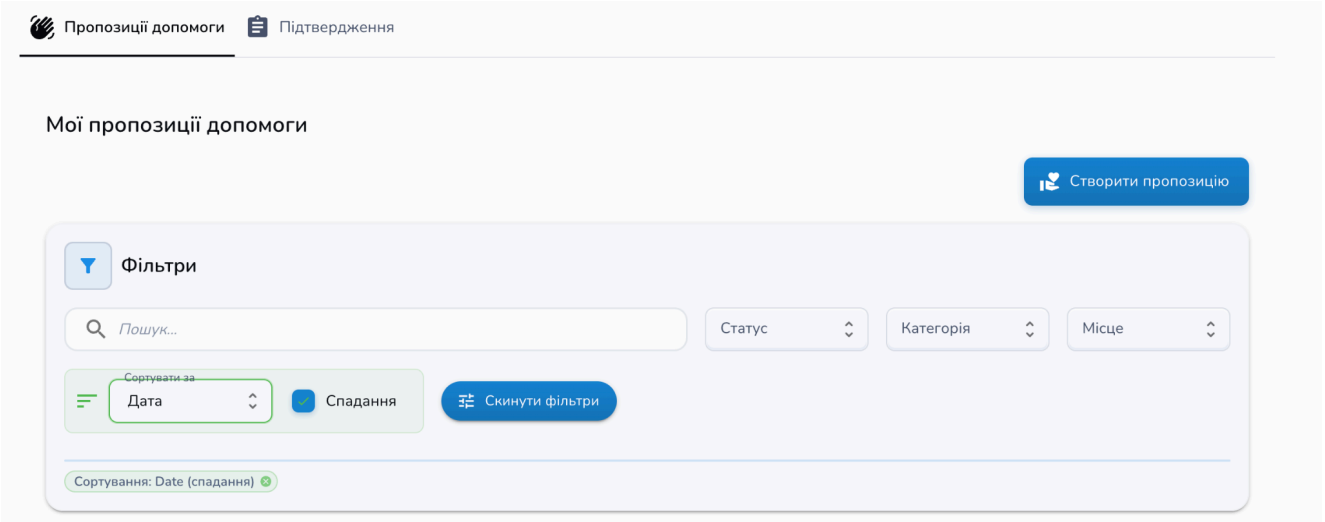


Рисунок 3.15 – Форма з фільтрами

Джерело: розроблено автором самостійно

На рисунку 3.15 представлено універсальну форму фільтрації, яка є ключовим інструментом для зручного та ефективного пошуку інформації в межах платформи. Ця форма реалізована як багаторазово використовуваний компонент, що вбудовується в різні секції системи, зокрема у списки запитів, пропозицій допомоги, історію взаємодій та модулі модерації.

Форма фільтрації надає користувачам гнучкі засоби для точного пошуку інформації за кількома параметрами:

- пошук дозволяє швидко знаходити записи за назвою, описом або іншими релевантними атрибутами. Пошукове поле реалізоване з підтримкою динамічної підстановки та підказок;
- фільтрація за статусом дає змогу обмежити відображення лише активних, завершених або архівних запитів/пропозицій, що є особливо корисним для волонтерів і модераторів, які працюють із великими масивами даних;
- фільтрація за категорією допомоги забезпечує тематичне групування, що дозволяє зосередитися на конкретних видах потреб;
- фільтрація за місцем розташування дає можливість обмежити результати певним регіоном, містом або адміністративною одиницею, що особливо актуально при кризовому розподілі ресурсів у регіональному контексті.

Крім того, форма передбачає можливість сортування результатів за різними критеріями, як наприклад, за датою створення, пріоритетністю чи віддаленістю. Усі параметри фільтрації реалізовані через елементи інтерфейсу Material UI:

селектори, чекбокси, текстові поля з автозаповненням, що забезпечує високу швидкодію, адаптивність до мобільних пристроїв і доступність.

Технічно форма побудована як окремий компонент React з використанням локального стану та Redux Toolkit для глобального контролю фільтрованих даних. Це дозволяє зберігати обрані параметри між переглядами, ділитися ними через URL-параметри, а також легко інтегрувати цей компонент у нові розділи системи без дублювання логіки.

У підсумку, форма фільтрації, зображена на рисунку 3.15, є важливим елементом UX, що значно покращує навігацію, зменшує когнітивне навантаження на користувача та сприяє швидшому реагуванню на актуальні потреби у складних гуманітарних умовах.

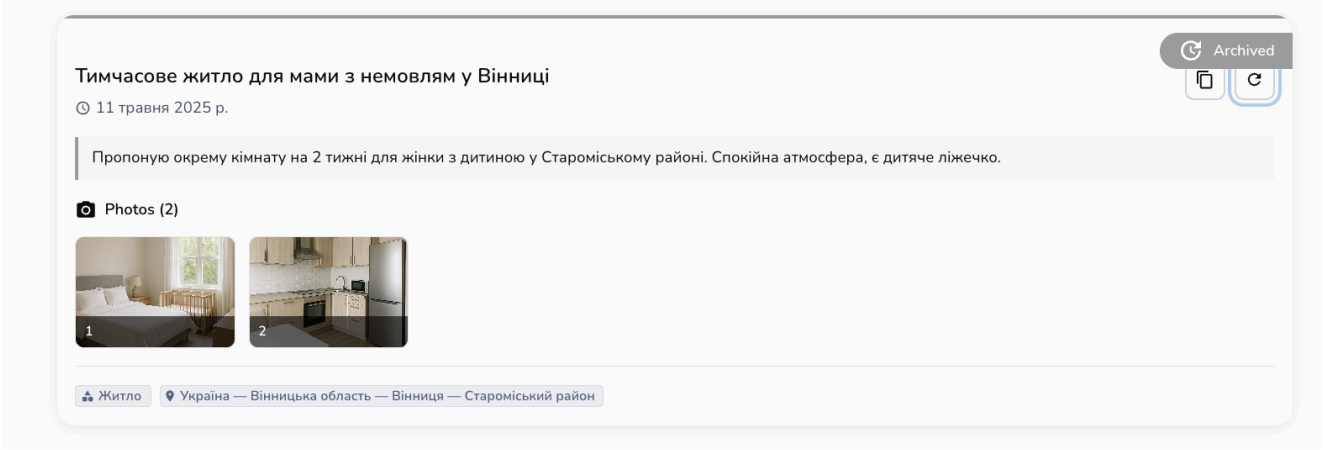


Рисунок 3.16 – Карта пропозиції (архівована)

Джерело: розроблено автором самостійно

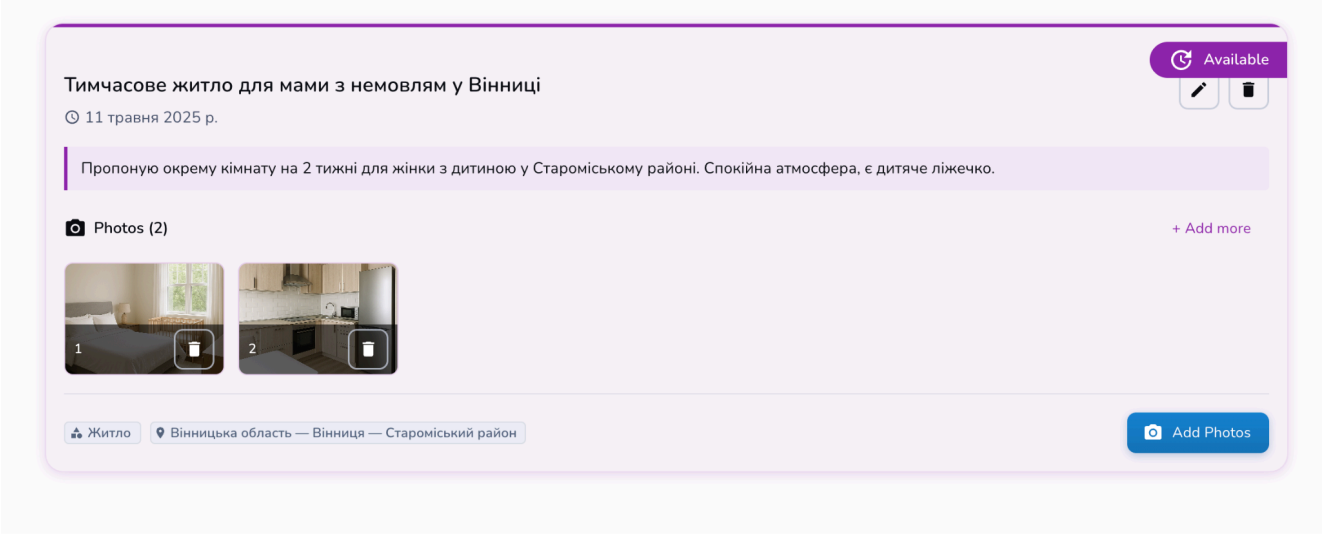


Рисунок 3.17 – Карта пропозиції (доступна)

Джерело: розроблено автором самостійно

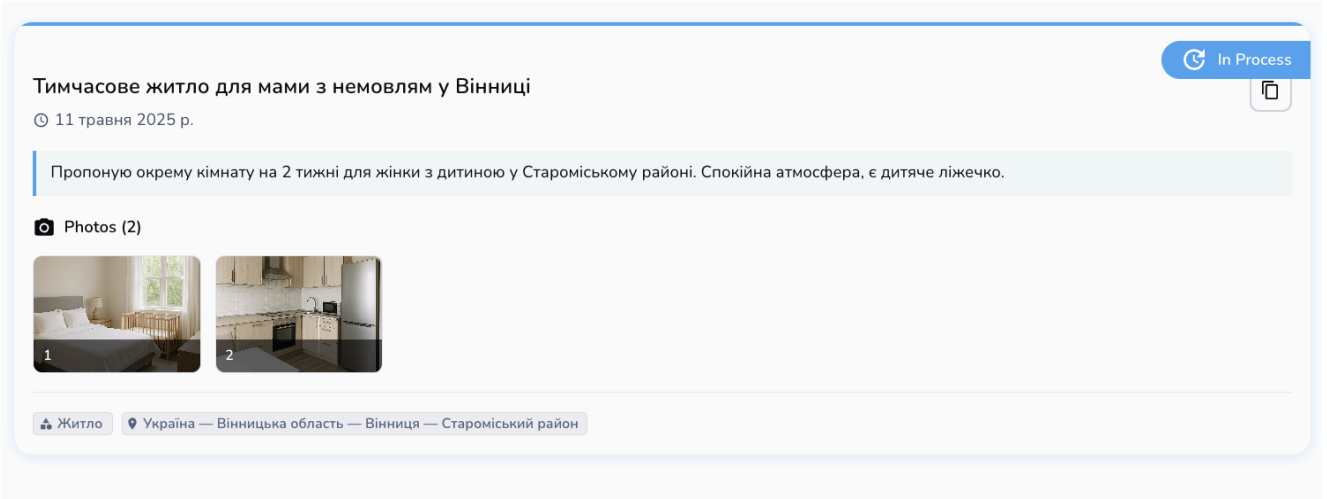


Рисунок 3.18 – Карта пропозиції (в процесі)

Джерело: розроблено автором самостійно

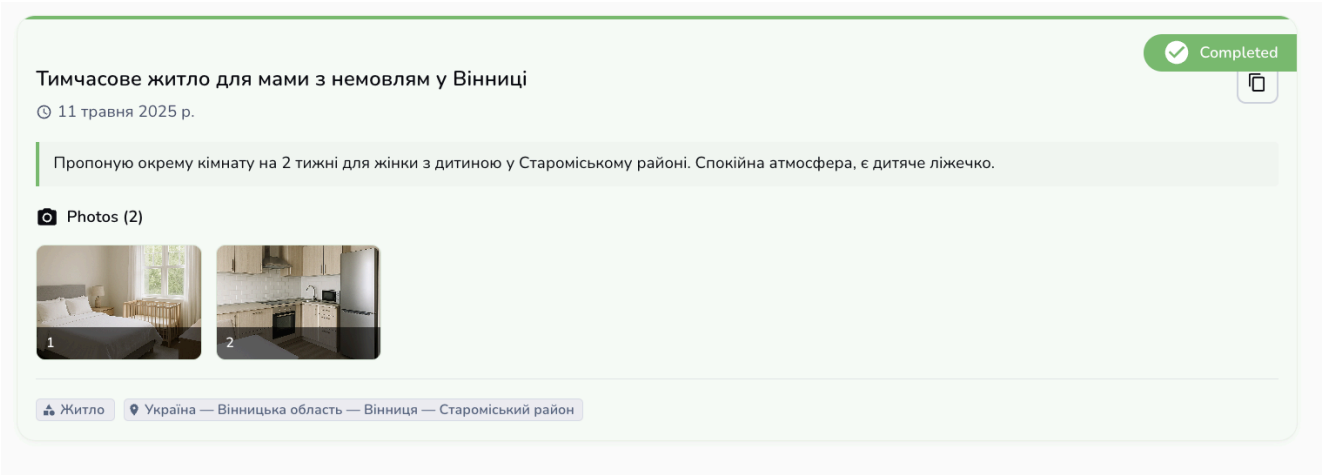


Рисунок 3.19 – Карта пропозиції (виконана)

Джерело: розроблено автором самостійно

На рисунках 3.16–3.19 зображено різні стани картки пропозиції допомоги, що реалізовані в системі. Кожен із цих станів відповідає певному етапу життєвого циклу пропозиції та супроводжується візуальною індикацією, що дозволяє користувачам швидко орієнтуватися у статусах.

Рисунок 3.16 - Карта пропозиції (архівована) демонструє завершену взаємодію, яка більше не доступна для редагування або відгуків. Такий статус допомагає зберігати історію дій і забезпечує прозорість комунікації.

Рисунок 3.17 - Карта пропозиції (доступна) відображає активну пропозицію, яка відкрита для відгуків з боку біженців.

Рисунок 3.18 – Карта пропозиції (в процесі) означає, що на пропозицію вже є активна взаємодія з боку одного користувача, але вона ще не завершена. Це

дозволяє іншим користувачам бачити, що допомога вже надається, але процес триває.

Рисунок 3.19 - Карта пропозиції ілюструє завершення процесу надання допомоги, з відповідною відміткою про успішне виконання. Такі карти можуть бути використані для статистичного аналізу й мотивації волонтерів.

Кожна карта побудована за особливим візуальним шаблоном. Колірна індикація статусу, а також піктограми, що супроводжують кожен запис, сприяють швидкому візуальному сприйняттю інформації, зменшуючи час на її обробку. Всі елементи реалізовано з використанням бібліотеки Material UI з додаванням анімацій через Framer Motion, що створює приємний користувацький досвід.

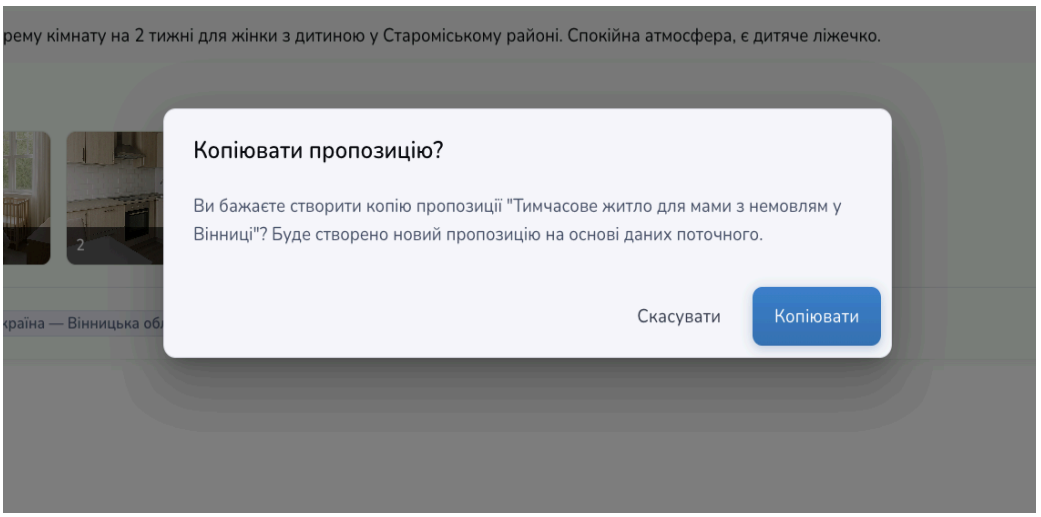


Рисунок 3.20 – Вікно для копіювання пропозиції

Джерело: розроблено автором самостійно

Рисунок 3.20 демонструє вікно для копіювання пропозиції, яке реалізоване для спрощення повторного створення аналогічних записів. Це особливо зручно для волонтерів або організацій, які регулярно надають однакову або подібну допомогу. Користувачеві надається можливість відкрити раніше створену пропозицію, змінити ключові поля за потреби й зберегти її як нову.

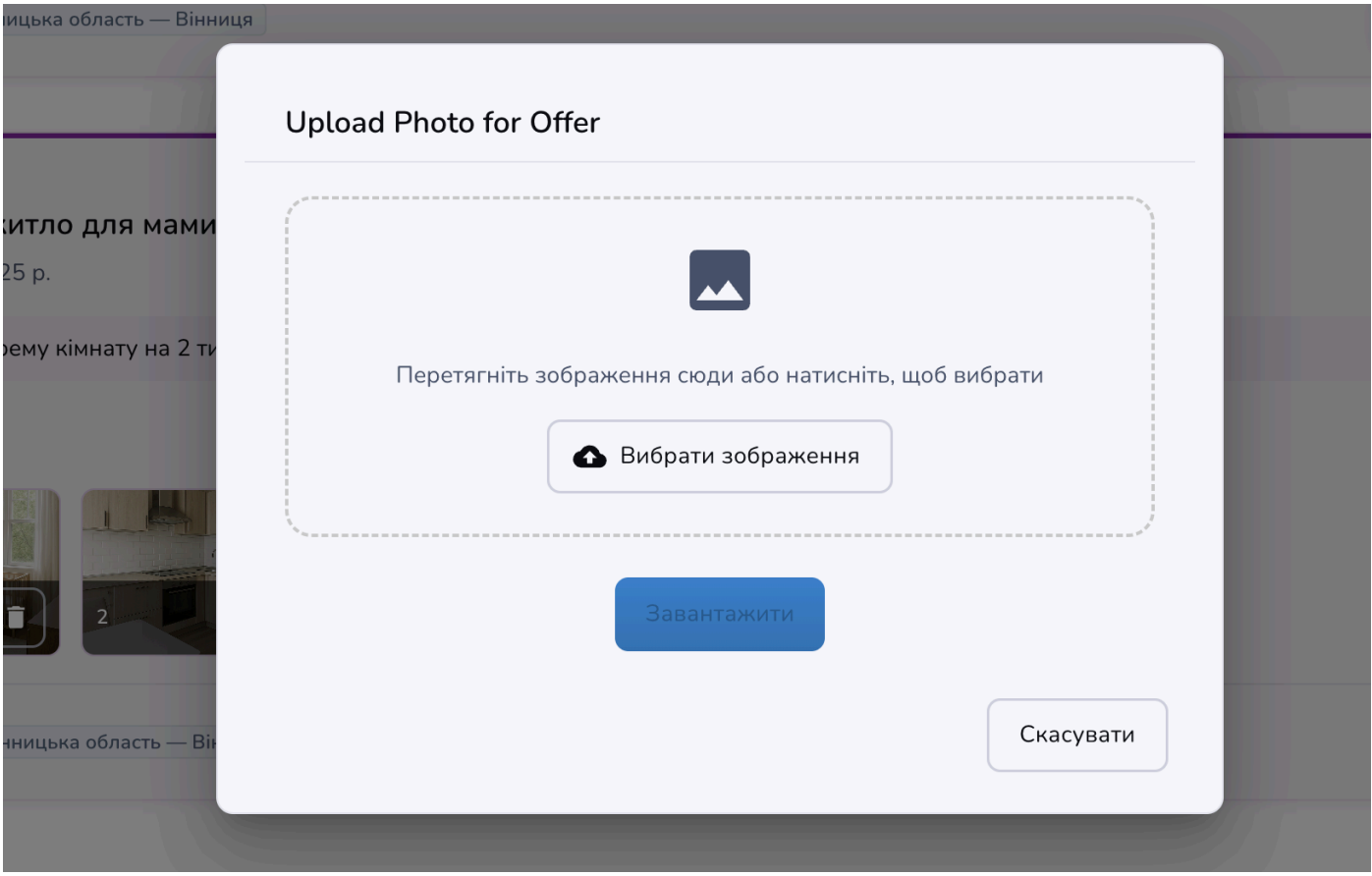


Рисунок 3.21 – Форма для завантаження фото для пропозиції

Джерело: розроблено автором самостійно

Рисунок 3.21 ілюструє форму завантаження фотографій до пропозиції, що дозволяє візуально підтвердити тип чи обсяг наданої допомоги. Завантажені фото проходять перевірку типу файлу, після чого додаються до пропозиції у вигляді галереї.

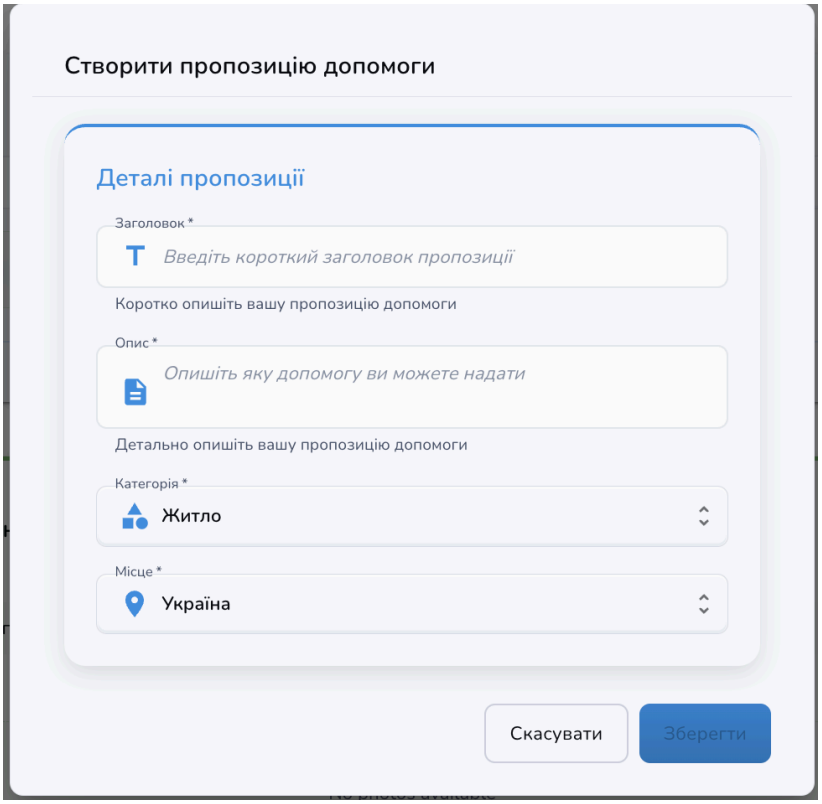


Рисунок 3.22 – Форма додавання пропозиції

Джерело: розроблено автором самостійно

Рисунок 3.22 відображає форму додавання нової пропозиції, яка включає обов'язкові поля. Інтерфейс підтримує валідацію введених даних.

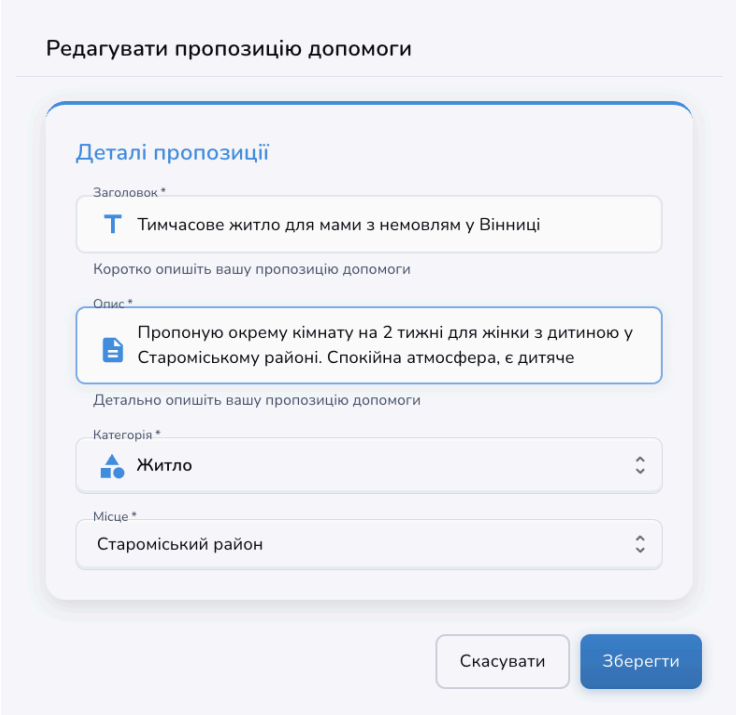


Рисунок 3.23 – Форма редагування пропозиції

Джерело: розроблено автором самостійно

Рисунок 3.23 демонструє форму редагування пропозиції, що дозволяє внести зміни до вже створеної заявки у разі зміни умов, помилок у введених даних або оновлення статусу. Форма містить ті ж елементи, що й форма створення.

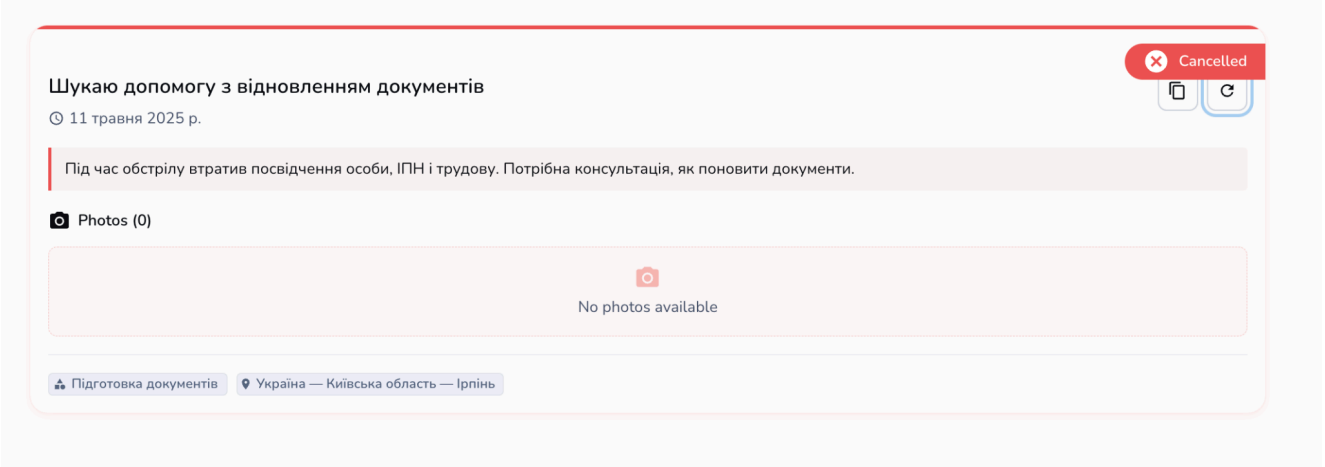


Рисунок 3.24 – Карта запиту (архівовано)

Джерело: розроблено автором самостійно

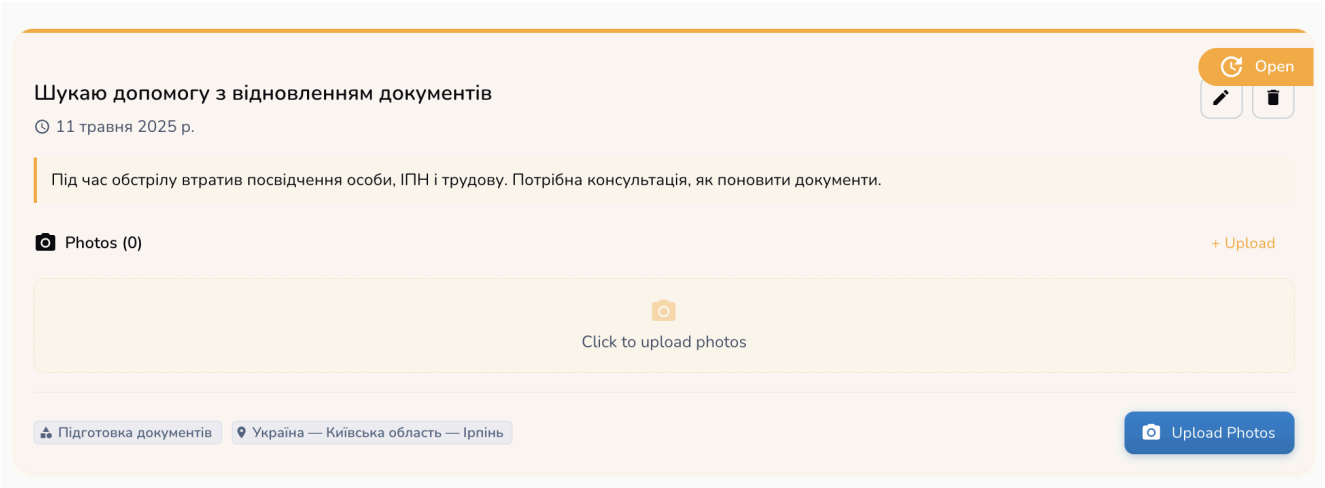


Рисунок 3.25 – Карта запиту (відкрита)

Джерело: розроблено автором самостійно

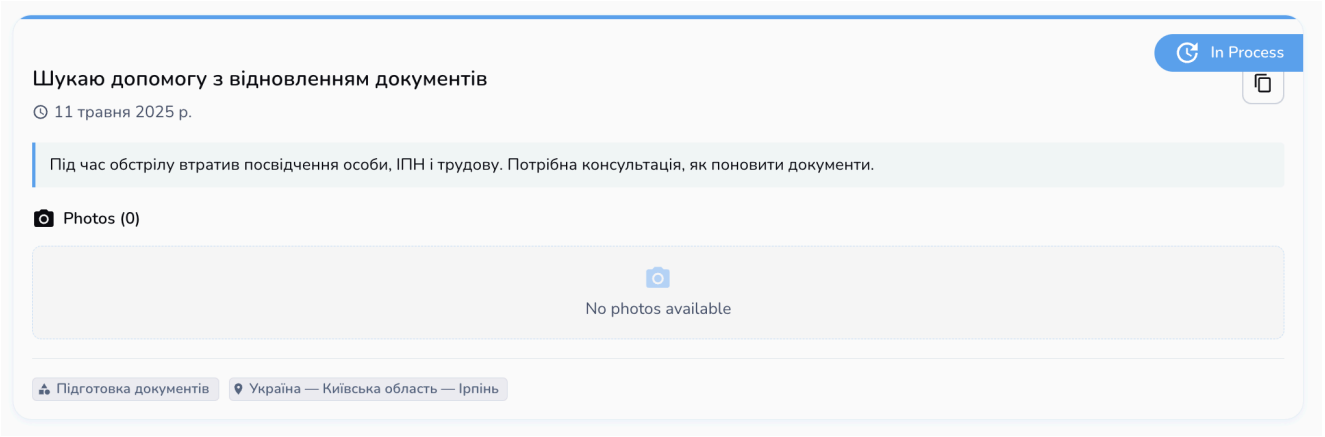


Рисунок 3.26 – Карта запиту (в процесі)

Джерело: розроблено автором самостійно

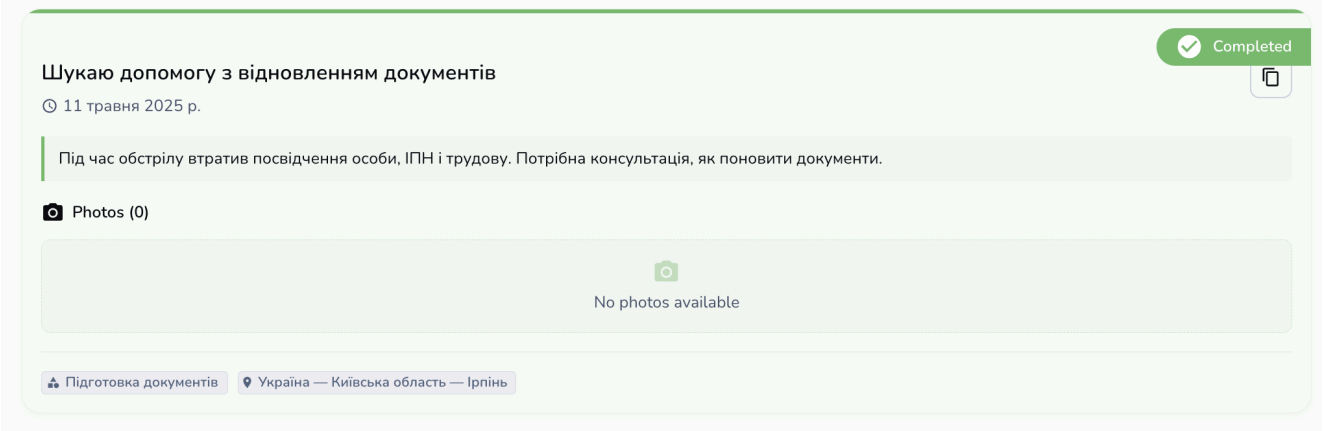


Рисунок 3.27 – Карта запиту (виконана)

Джерело: розроблено автором самостійно

На рисунках 3.24 - 3.27 представлено різні стани картки запиту, які відображають етапи взаємодії біженця з платформою. Кожен стан оформлено з урахуванням чіткої візуальної ідентифікації, що дозволяє користувачам системи швидко орієнтуватися в статусі своїх звернень і реагувати відповідно до обставин.

Рисунок 3.24 - Карта запиту (архівовано) демонструє завершене звернення, що вже не підлягає редагуванню або новим відгукам. Такий тип картки призначений для збереження історії запитів і створення аналітичної бази щодо частоти та типів потреб.

Рисунок 3.25 - Карта запиту (відкрита) відображає активне звернення, яке може переглядатися надавачами допомоги й бути доступним для відгуку.

Рисунок 3.26 – Карта запиту позначає, що на звернення вже надано відповідь, і триває процес узгодження або виконання. Такий статус сигналізує про те, що інші волонтери можуть спостерігати, але не дублювати дії.

Рисунок 3.27 - Карта запиту (виконана) ілюструє завершену допомогу, яка була надана успішно. Наявність такого статусу підвищує прозорість платформи та дозволяє іншим користувачам бачити приклади ефективної взаємодії.

Кожна карта має адаптивну структуру з ключовими метаданими Індикатори статусу реалізовані за допомогою кольорової гами та піктограм, що відповідають принципам UX-дизайну й роблять взаємодію інтуїтивною.

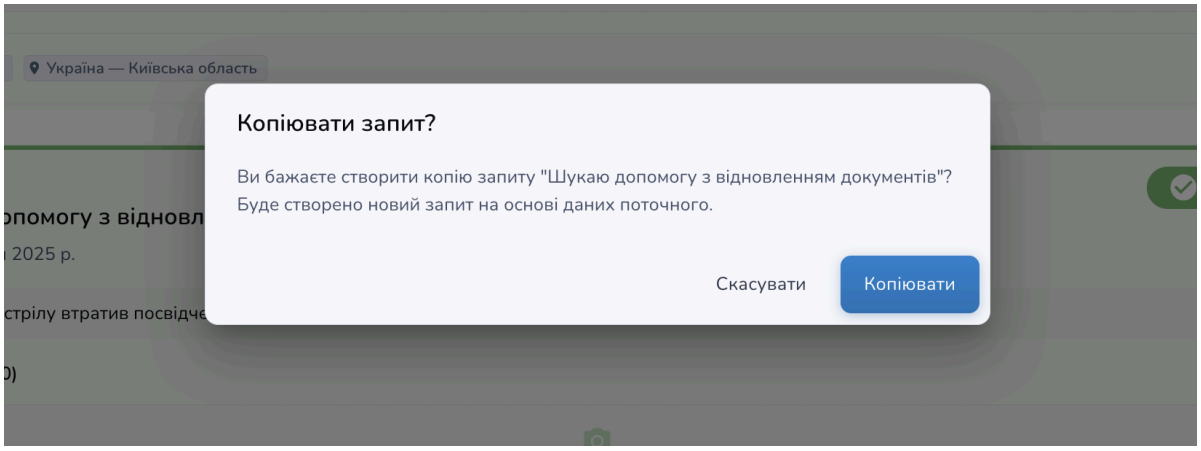


Рисунок 3.28 – Вікно для копіювання запиту

Джерело: розроблено автором самостійно

Рисунок 3.28 демонструє вікно копіювання запиту, яке дозволяє біженцям швидко повторити звернення з тими самими або подібними даними. Такий функціонал суттєво знижує час на повторне створення запитів, що особливо корисно в умовах, коли потреба є регулярною.

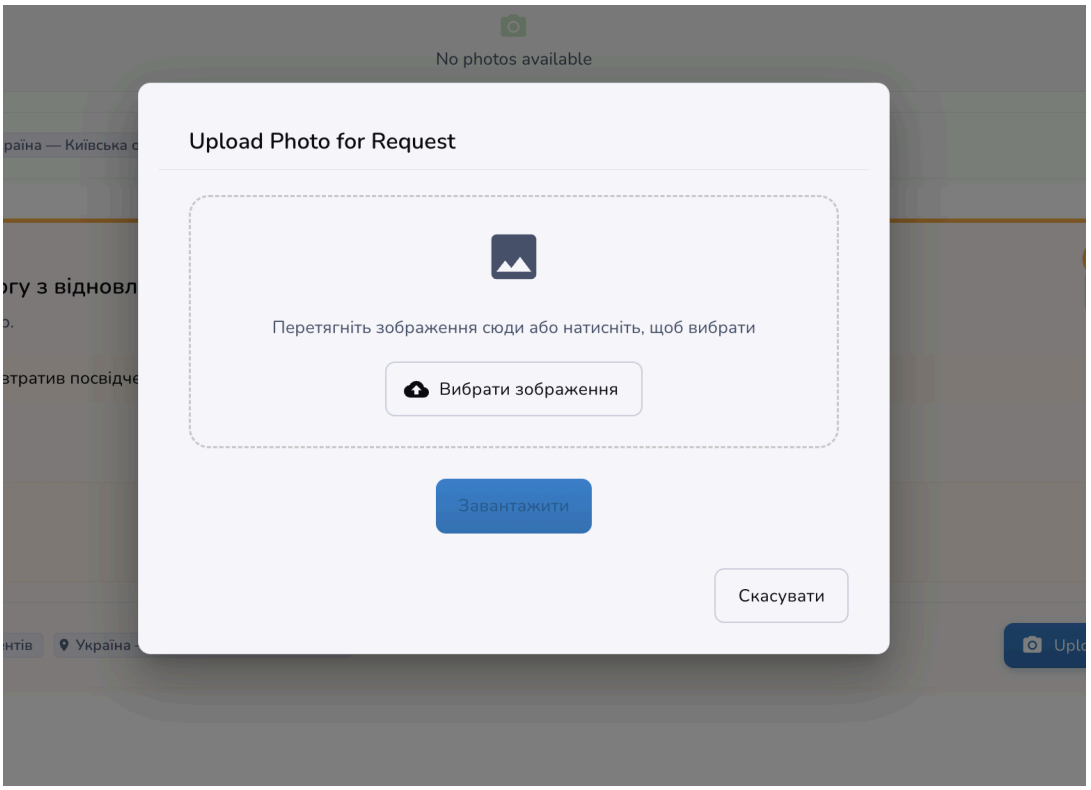


Рисунок 3.29 – Форма для завантаження фото для запиту

Джерело: розроблено автором самостійно

Рисунок 3.29 ілюструє форму завантаження фотографії до запиту, яка дає змогу додати візуальне підтвердження потреби. Завантажені зображення автоматично обробляються системою, зберігаються у безпечному сховищі.

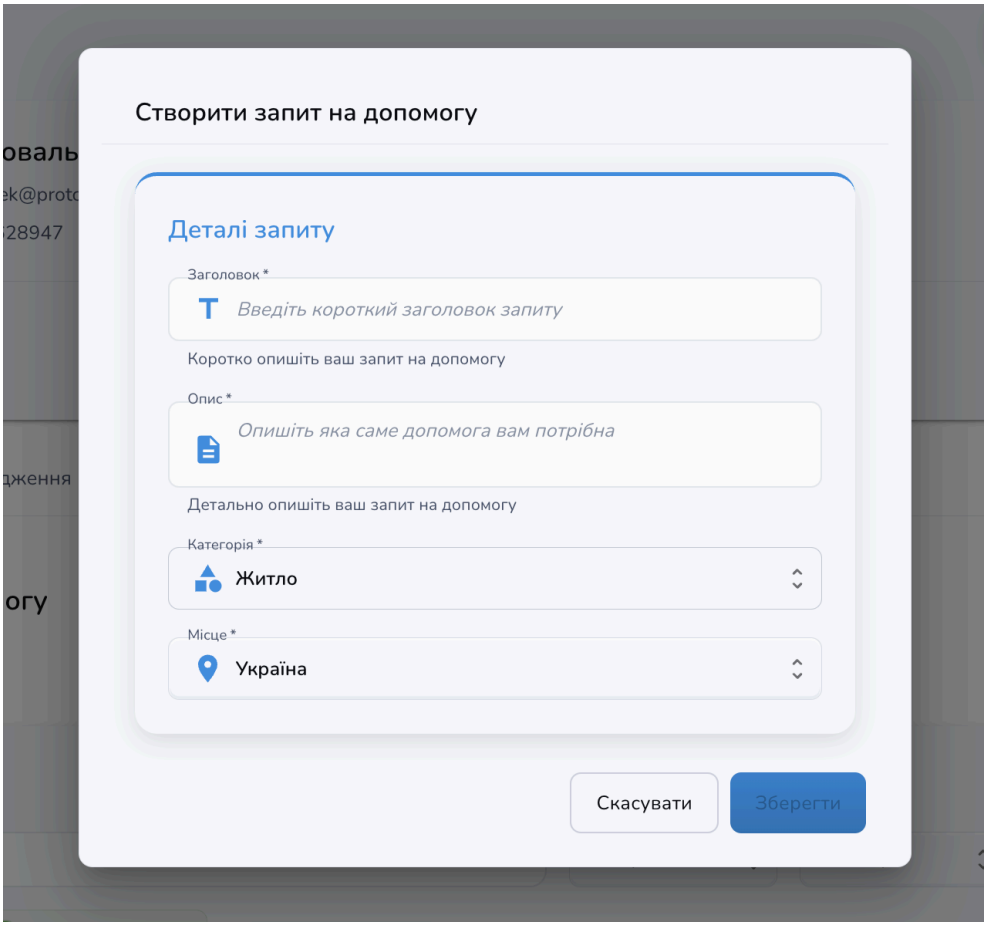


Рисунок 3.30 – Форма додавання запиту

Джерело: розроблено автором самостійно

Рисунок 3.30 відображає форму створення нового запиту, яка побудована за логікою простоти та зрозумілості. Користувач має змогу обрати категорію, вказати опис потреби, додати фото, зазначити місцезнаходження. Усі поля супроводжуються валідацією для запобігання помилкам введення.

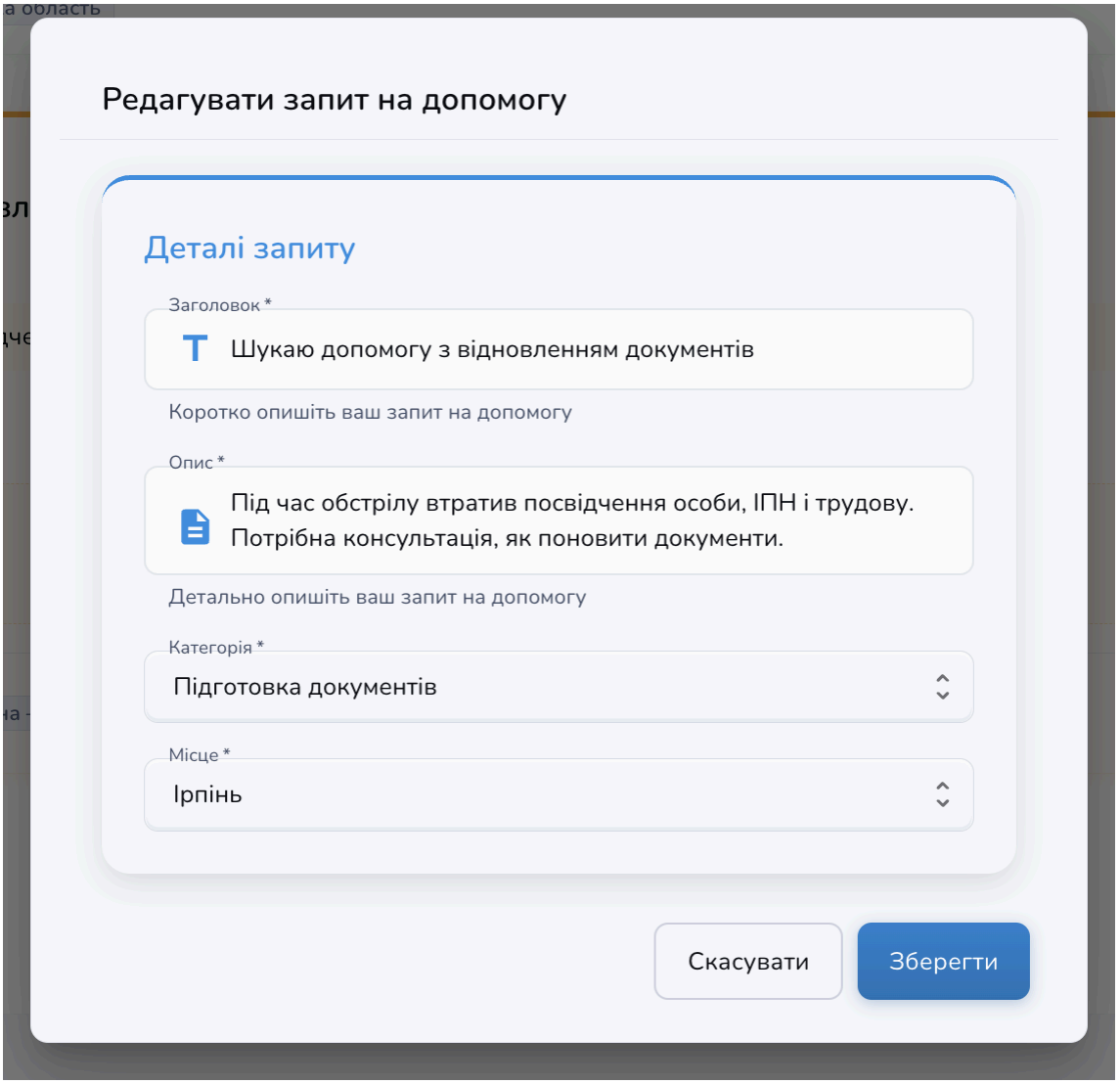


Рисунок 3.31 – Форма редагування запиту

Джерело: розроблено автором самостійно

Рисунок 3.31 демонструє форму редагування запиту, яка відкривається при потребі оновити інформацію або виправити неточності.

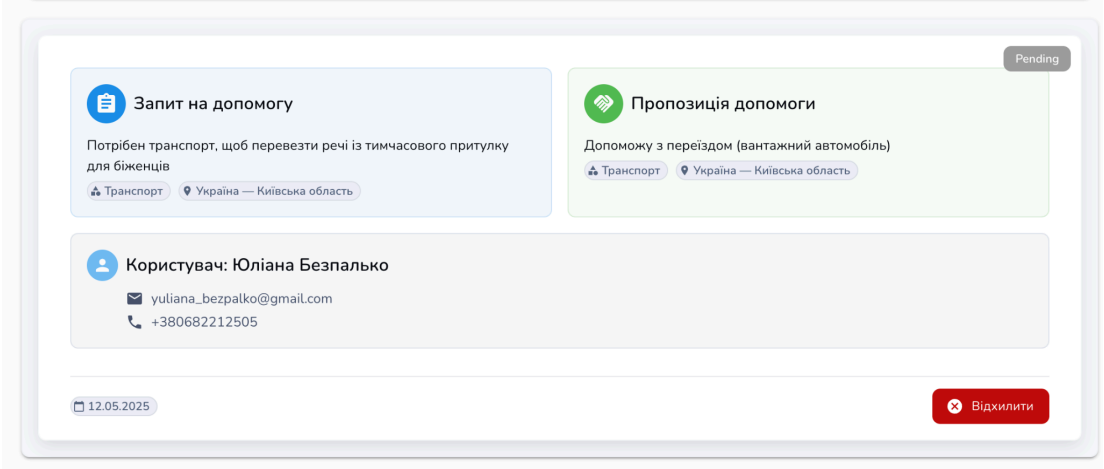


Рисунок 3.32 – Картка призначення у відправника (відправлена)

Джерело: розроблено автором самостійно

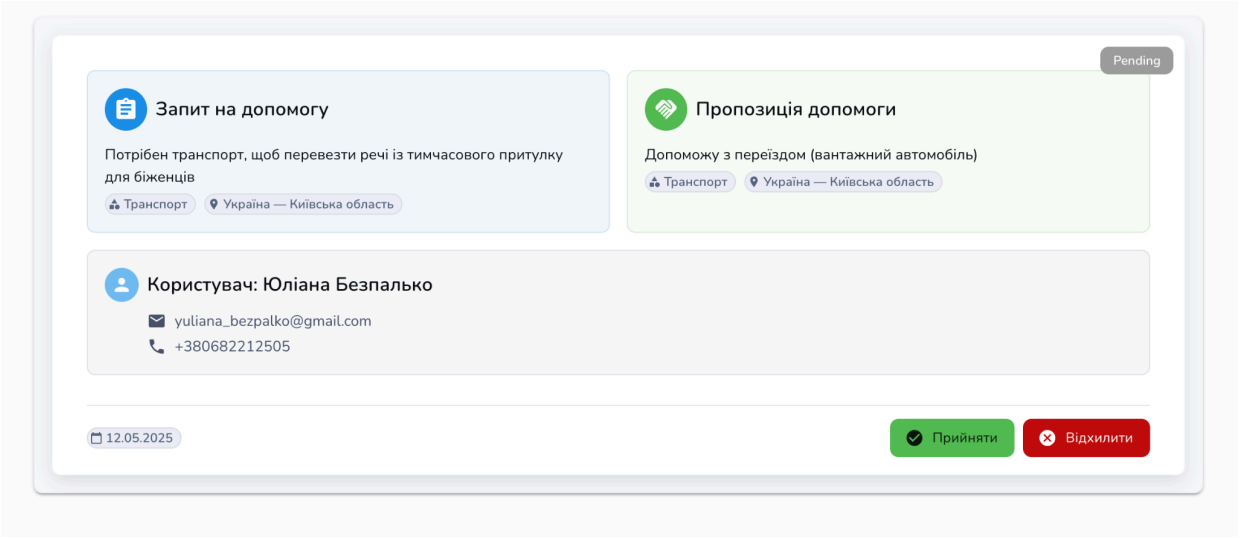


Рисунок 3.33 – Картка призначення у отримувача (відправлена)

Джерело: розроблено автором самостійно

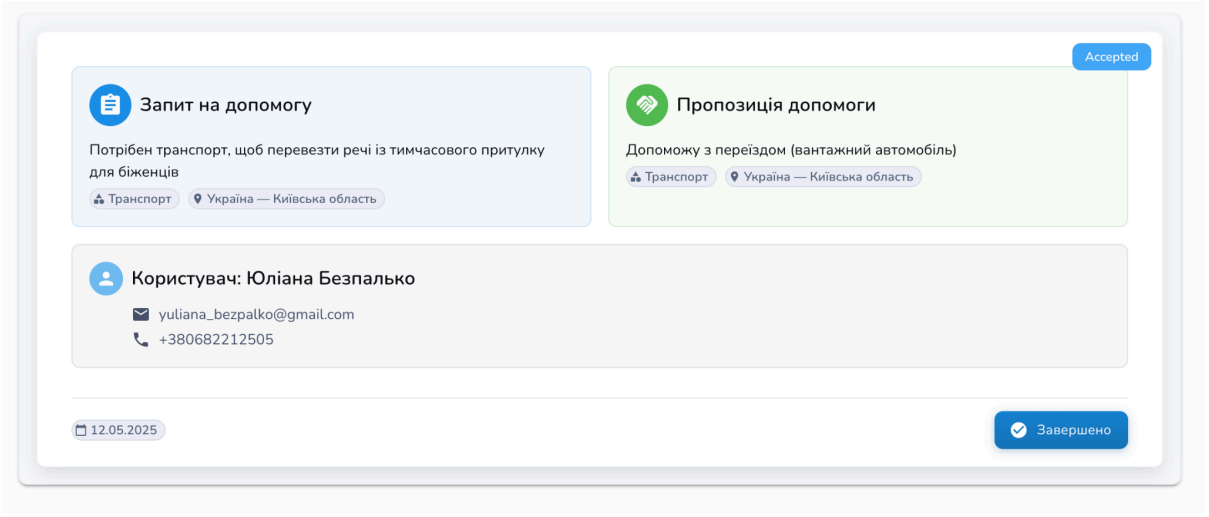


Рисунок 3.34 – Картка призначення (в роботі)

Джерело: розроблено автором самостійно

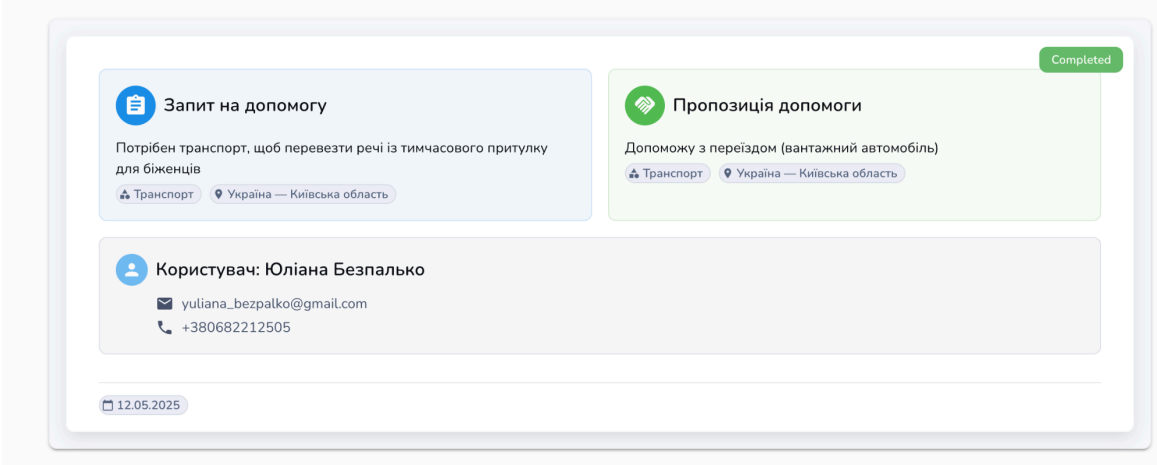


Рисунок 3.35 – Картка призначення (завершена)

Джерело: розроблено автором самостійно

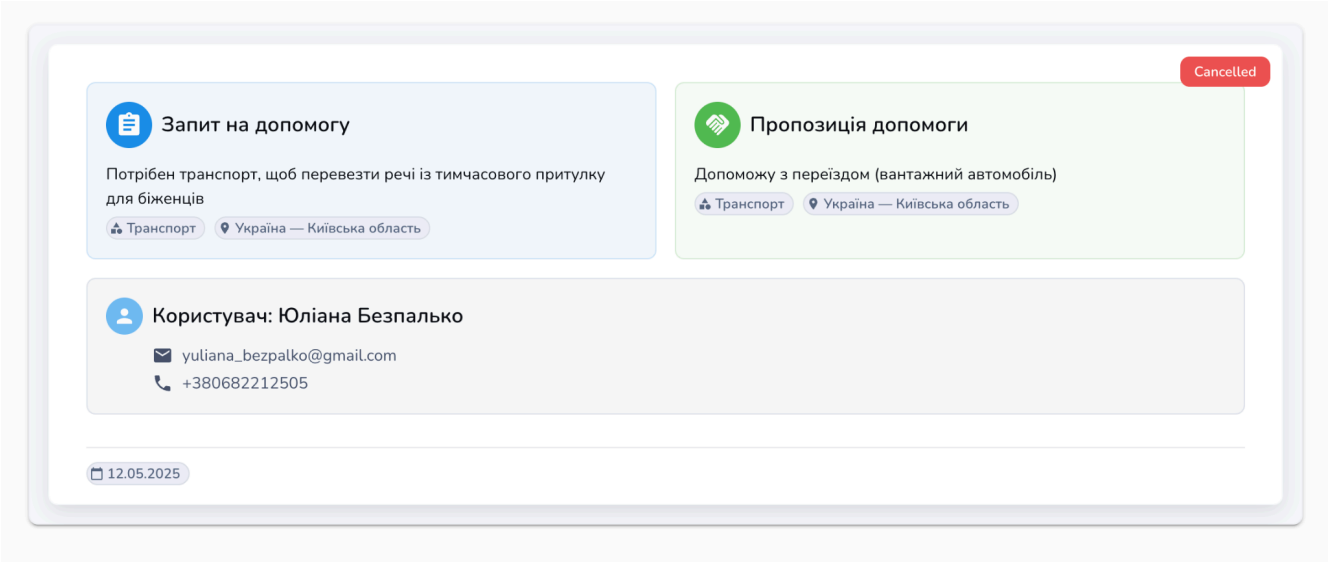


Рисунок 3.36 – Картка призначення (відмінена)

Джерело: розроблено автором самостійно

На рисунках 3.32 - 3.36 зображено картки призначень, що є ключовим елементом у реалізації механізму взаємодії між біженцями та волонтерами в системі. Призначення виникає в результаті відгуку на запит або пропозицію допомоги, і відображає процес погодження та виконання конкретного акту взаємодії між сторонами.

Рисунок 3.32 – Картка призначення у відправника (відправлена) відображає стан, у якому користувач, що ініціював взаємодію, вже надіслав відповідну заявку на контакт або допомогу. У картці відображається інформація про одержувача, тип допомоги, дата призначення та очікуваний статус. Доступні дії - скасування призначення або перегляд деталей.

Рисунок 3.33 - Картка призначення у отримувача (відправлена) показує дзеркальну картину, інший учасник отримав повідомлення про ініційовану взаємодію. У цьому стані він має змогу прийняти, відхилити. Це забезпечує обопільне погодження дій перед переходом до фактичного виконання.

Рисунок 3.34 - Картка призначення (в роботі) демонструє активний стан, у якому обидві сторони погодили взаємодію, і триває процес виконання домовленості. Система фіксує цей стан як поточний, що дозволяє іншим користувачам бачити, що відповідний запит або пропозиція вже обробляються. У картці доступні опції оновлення статусу, перегляду спільного чату або уточнення умов.

Рисунок 3.35 - Картка призначення (завершена) означає, що допомога була надана, і всі сторони підтвердили її виконання. Такий статус зберігається в історії облікового запису кожного учасника.

Рисунок 3.36 - Картка призначення (відмінена) показує призначення, яке було припинено однією зі сторін до завершення взаємодії.

Усі картки побудовані з урахуванням візуальної послідовності. Кожна має заголовок, короткий опис домовленості, контактні дані сторін, статус, а також інтерактивні елементи для змін або підтвердження. Використання іконок, кольорових індикаторів і анімацій допомагає підвищити зрозумілість і доступність інтерфейсу навіть для недосвідчених користувачів.

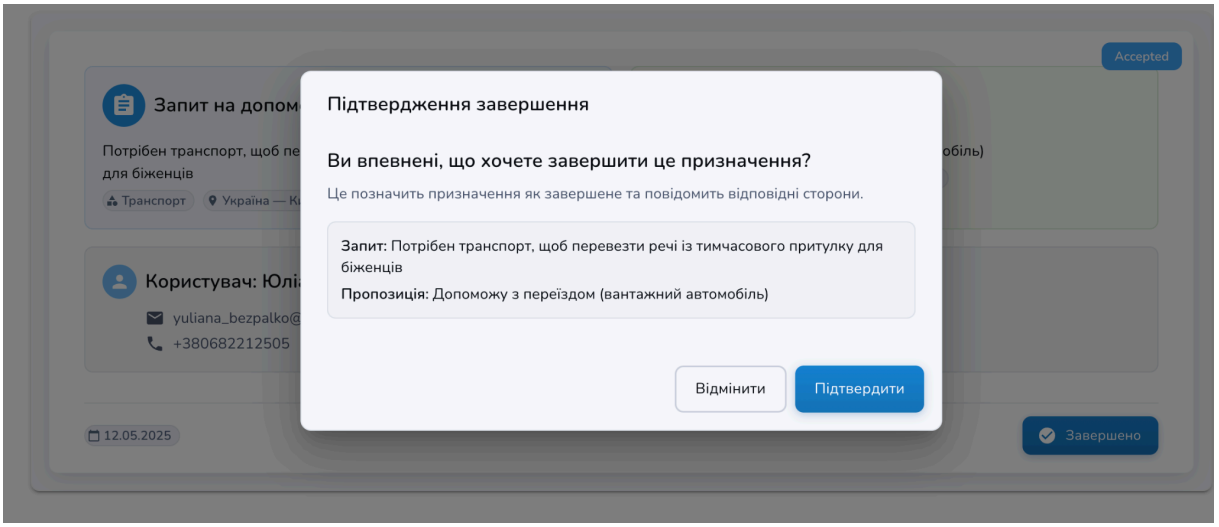


Рисунок 3.37 – Форма підтвердження призначення

Джерело: розроблено автором самостійно

Рисунок 3.37 демонструє форму підтвердження призначення, яка відкривається після досягнення домовленості між сторонами. Ця форма містить коротке резюме домовленостей, а також кнопки для остаточного підтвердження чи відмови.

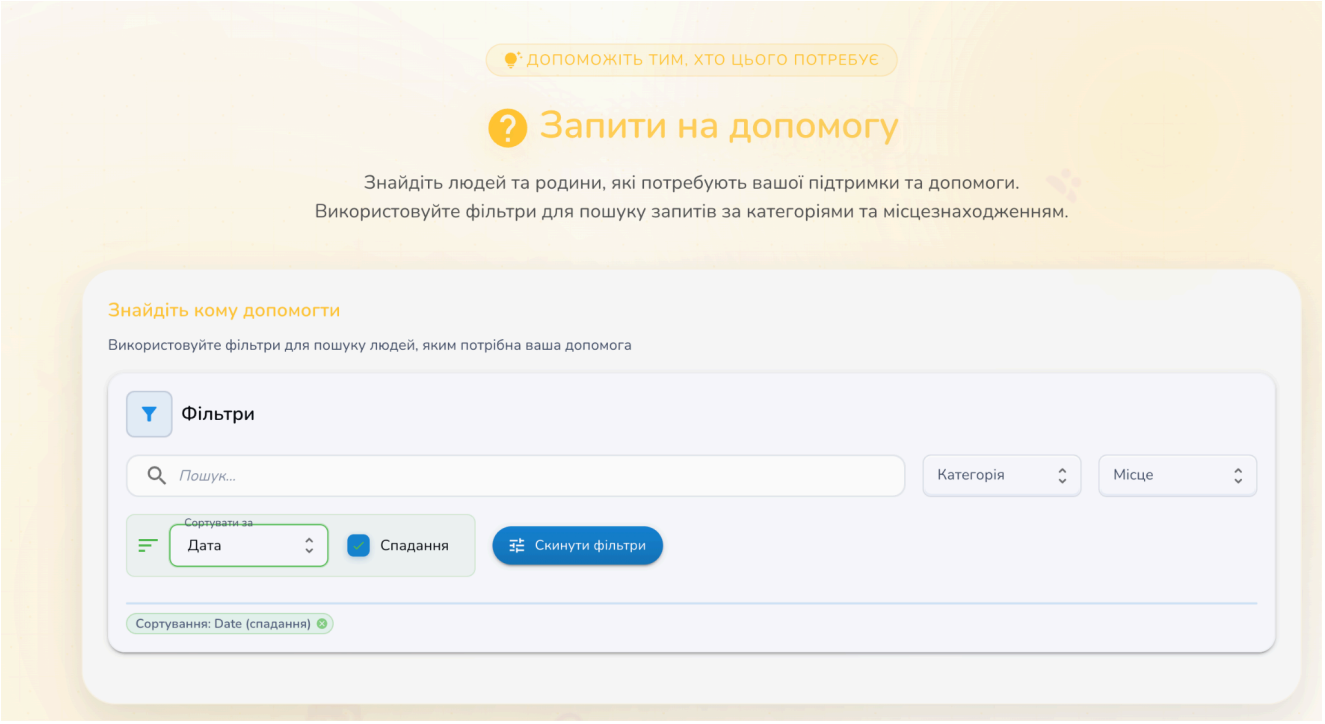


Рисунок 3.38 – Фільтрація на сторінці з запитами про допомогу

Джерело: розроблено автором самостійно

Рисунок 3.38 ілюструє функціонал фільтрації на сторінці з відкритими запитами про допомогу, що є одним з найважливіших інструментів у навігації користувачів серед великої кількості звернень.

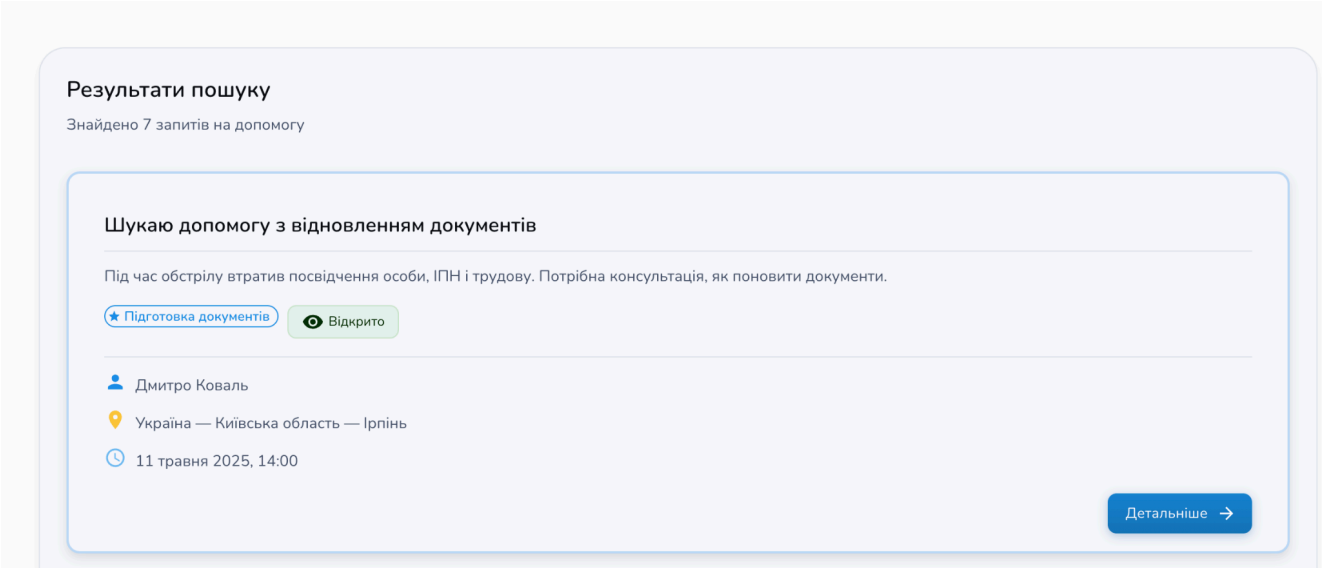


Рисунок 3.39 – Картка запиту на допомогу

Джерело: розроблено автором самостійно

Рисунок 3.39 демонструє картку запиту на допомогу, яка є короткою анотацією звернення, поданого біженцем. Картка містить заголовок, категорію, опис, дату створення, географічну прив’язку та кнопку для перегляду деталей.

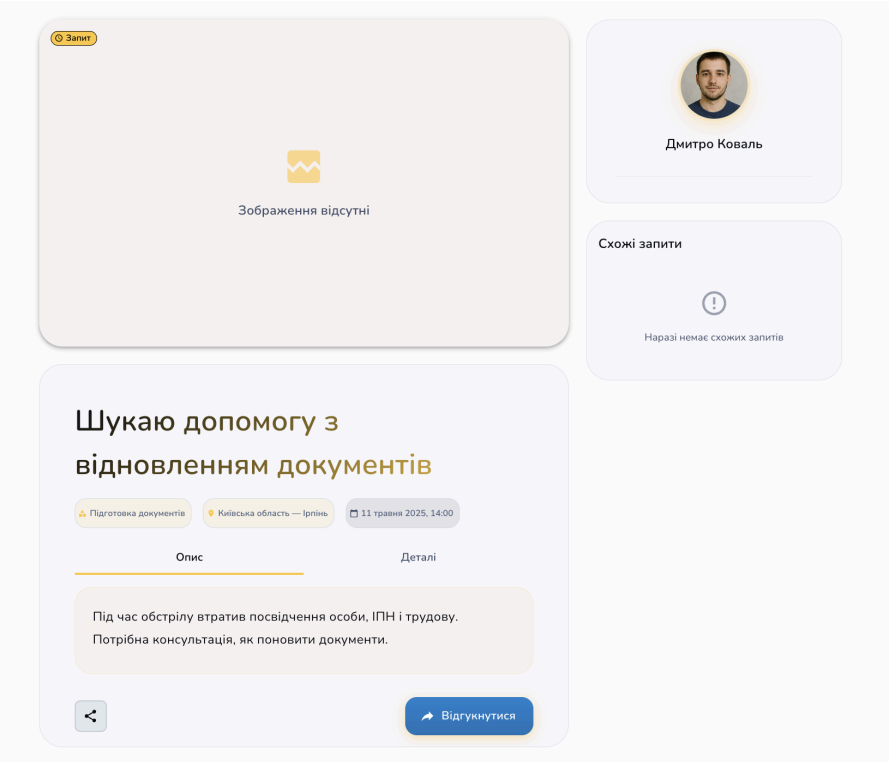


Рисунок 3.40 – Сторінка з запитом на допомогу

Джерело: розроблено автором самостійно

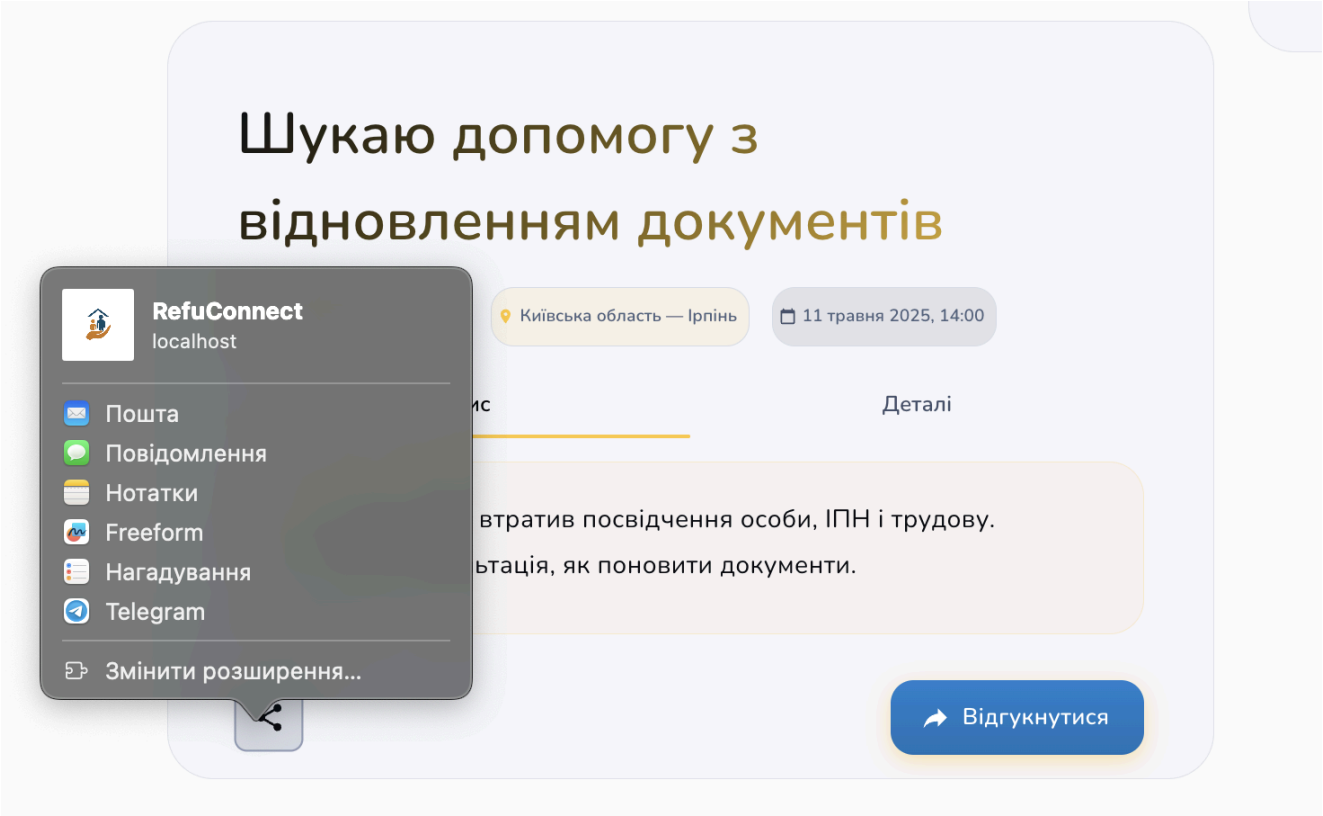


Рисунок 3.41 – Можливість поділитись запитом

Джерело: розроблено автором самостійно

Рисунок 3.41 показує можливість поділитися запитом. Користувач може скопіювати пряме посилання або надіслати запит через месенджер або соціальні

мережі. Це особливо актуально у випадках, коли заявник самостійно не має змоги привернути увагу до своєї потреби, а спільнота може допомогти у розповсюдженні запиту.

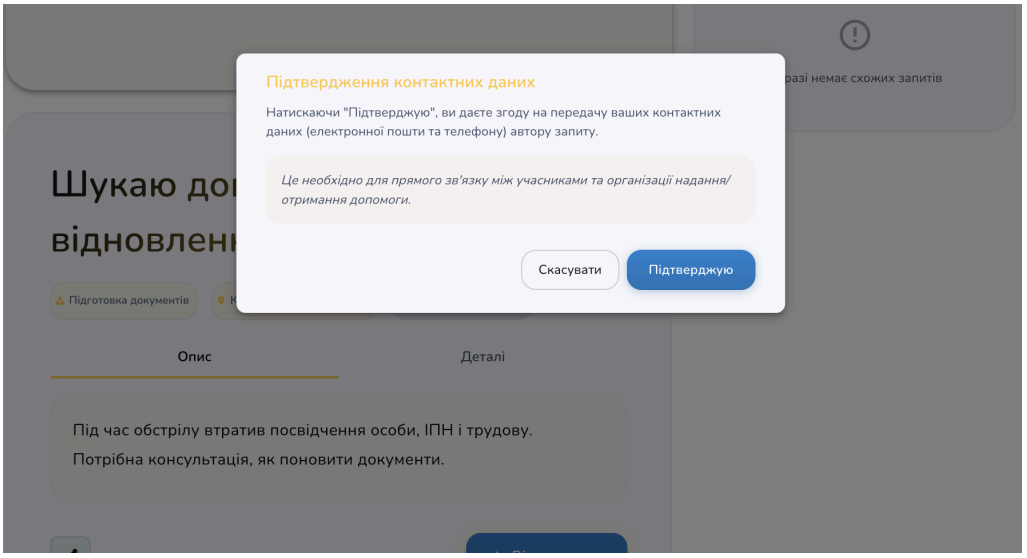


Рисунок 3.42 – підтвердження передачі конт. даних (для запиту)

Джерело: розроблено автором самостійно

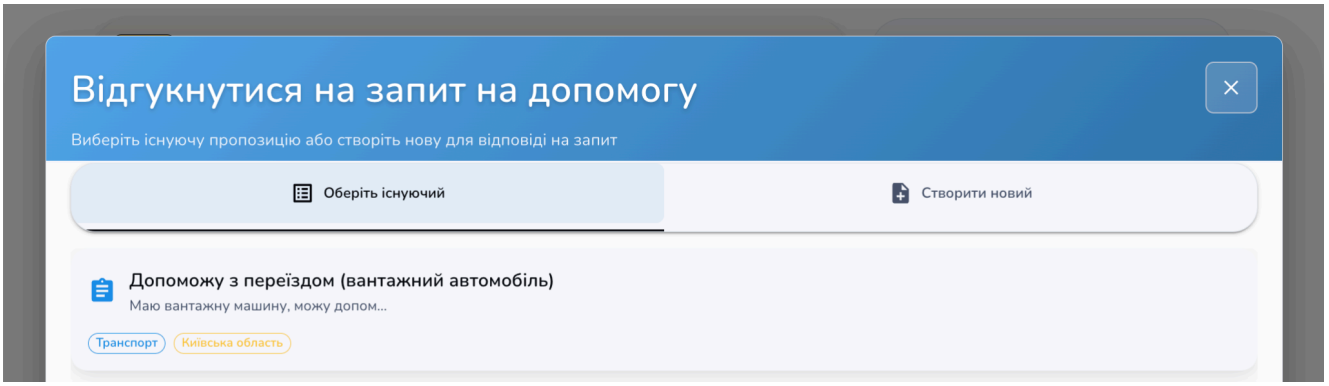


Рисунок 3.43 –Вікно для відгуку на запит (таба 1)

Джерело: розроблено автором самостійно

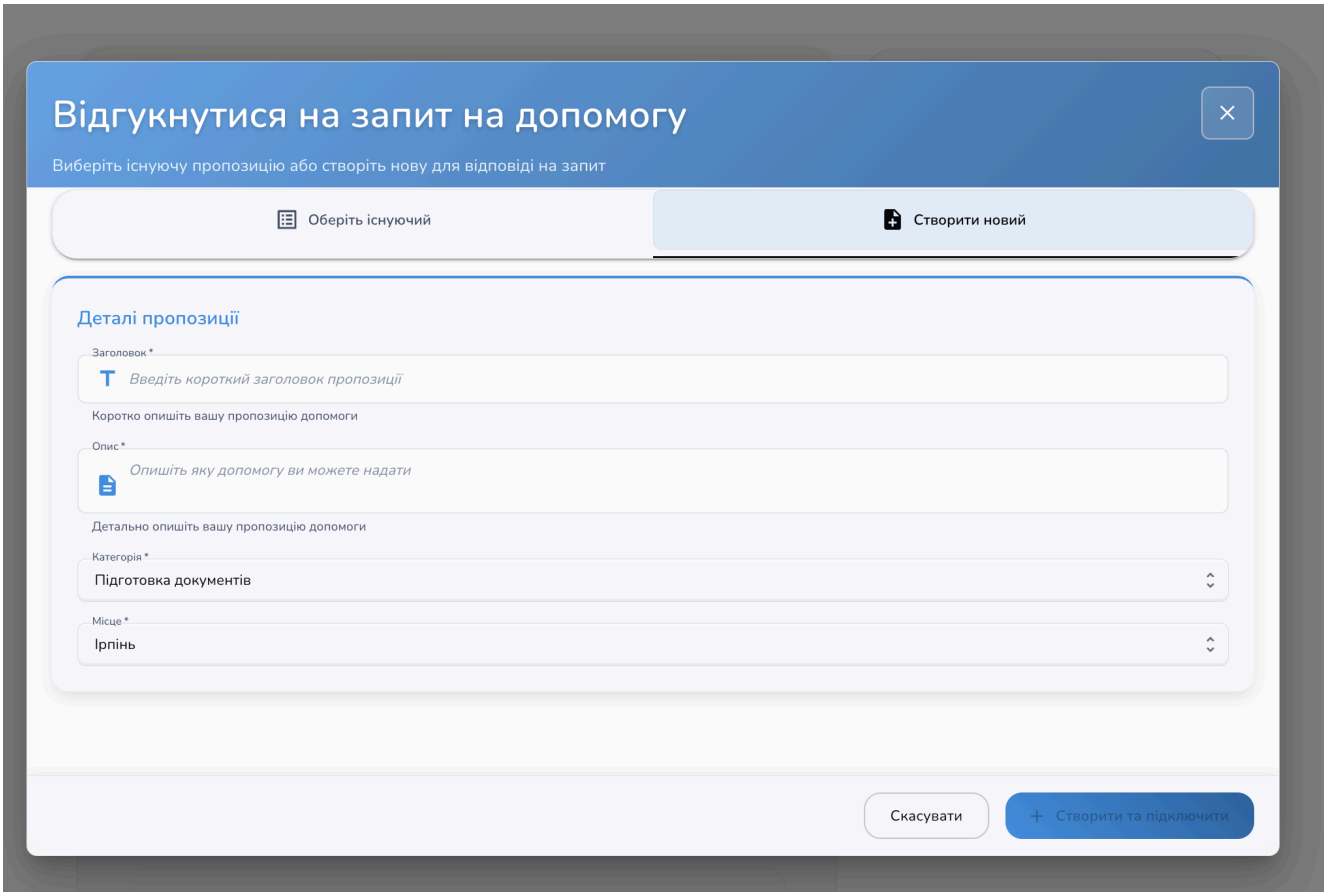


Рисунок 3.44 –Вікно для відгуку на запит (таба 2)

Джерело: розроблено автором самостійно

На рисунках 3.43–3.44 представлено діалогове вікно, що відкривається у відповідь на запит про допомогу. Користувачеві надається два варіанти дій - обрати одну з наявних пропозицій або створити нову, відповідно до характеру запиту. Категорія і місце беруться з самого запиту.

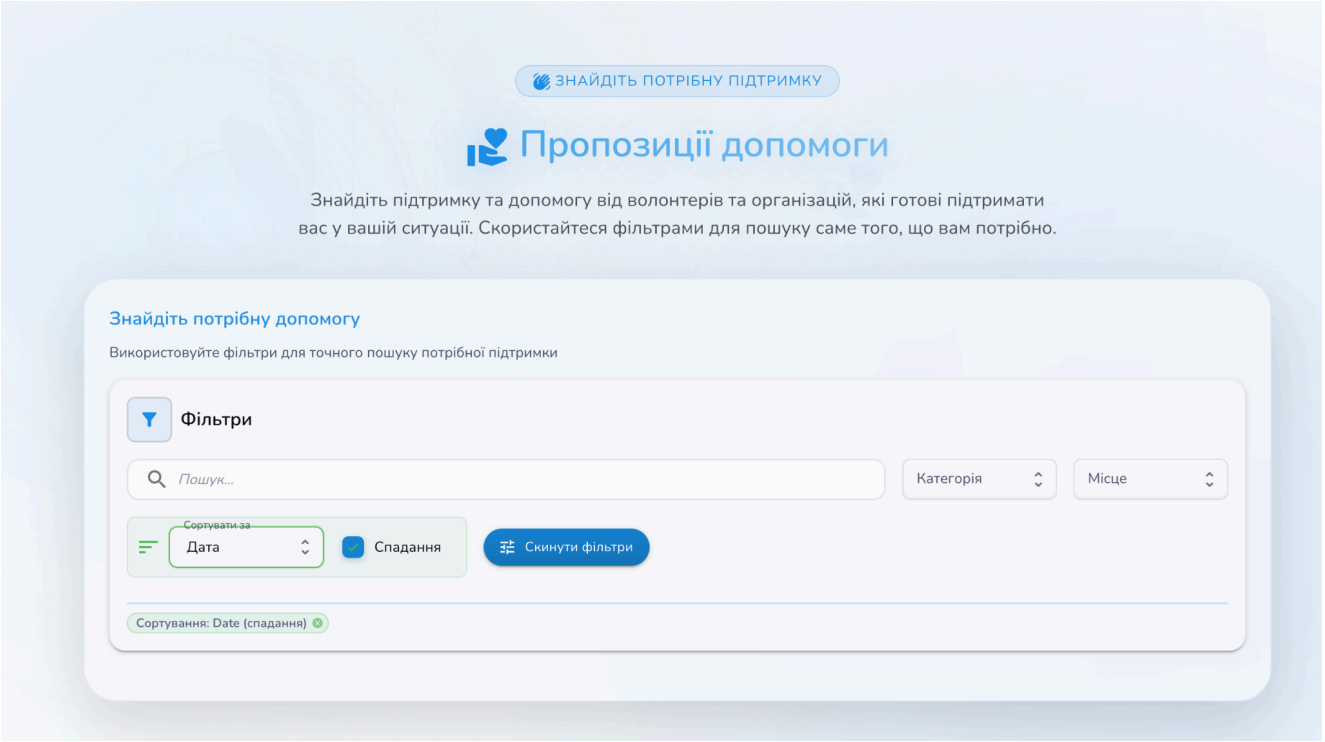


Рисунок 3.45 – Фільтрація на сторінці з пропозиціями про допомогу

Джерело: розроблено автором самостійно

Рисунок 3.45 ілюструє функціонал фільтрації на сторінці з відкритими пропозиціями на допомогу, що є одним з найважливіших інструментів у навігації біженців серед великої кількості звернень.

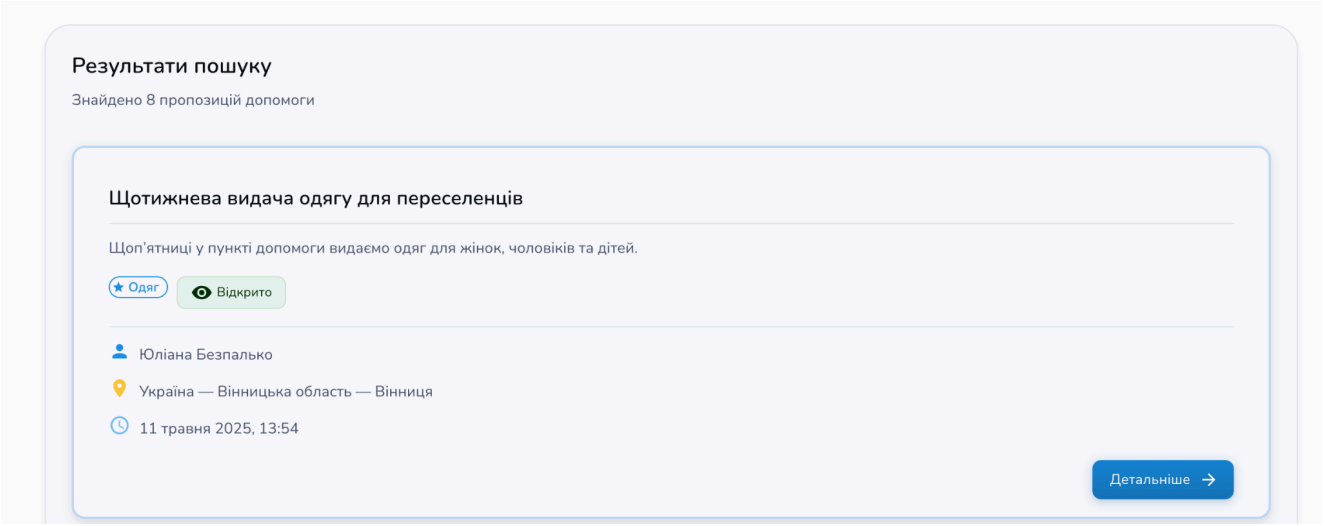


Рисунок 3.46 – Картка пропозиції на допомогу

Джерело: розроблено автором самостійно

Рисунок 3.46 демонструє картку пропозиції на допомогу, яка є короткою анотацією звернення, поданого волонтером. Картка містить заголовок, категорію, опис, дату створення, географічну прив’язку та кнопку для перегляду деталей.

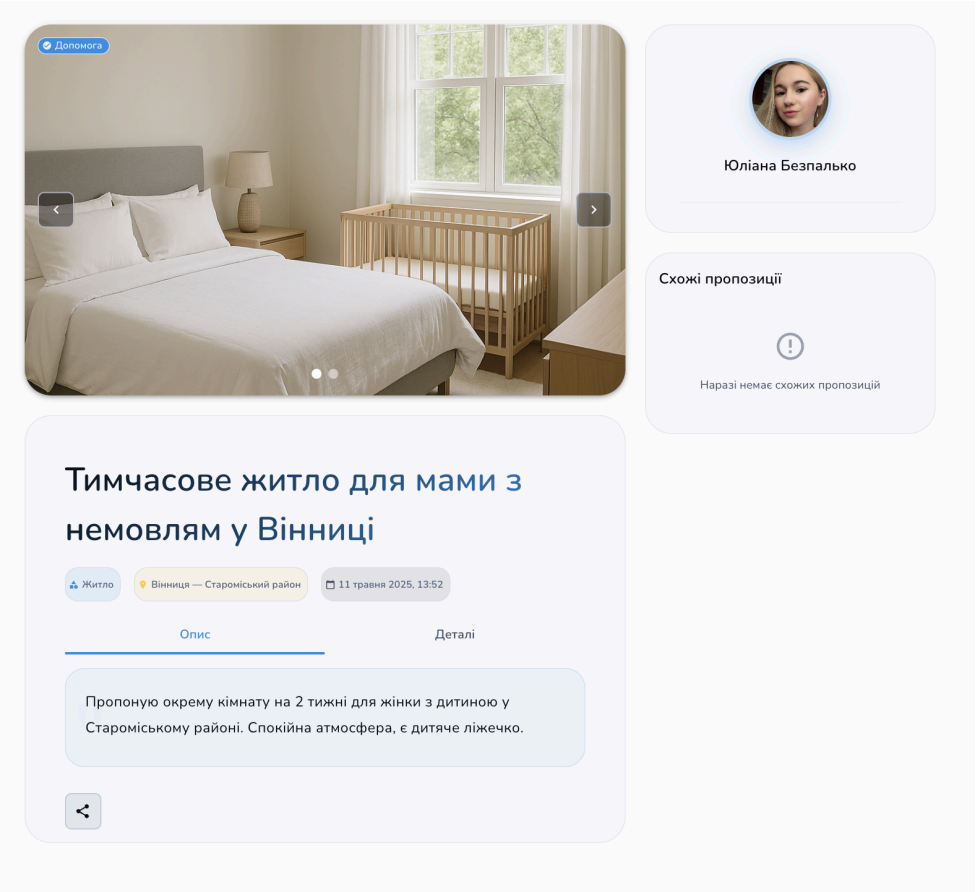


Рисунок 3.47 – Сторінка з пропозицією на допомогу

Джерело: розроблено автором самостійно

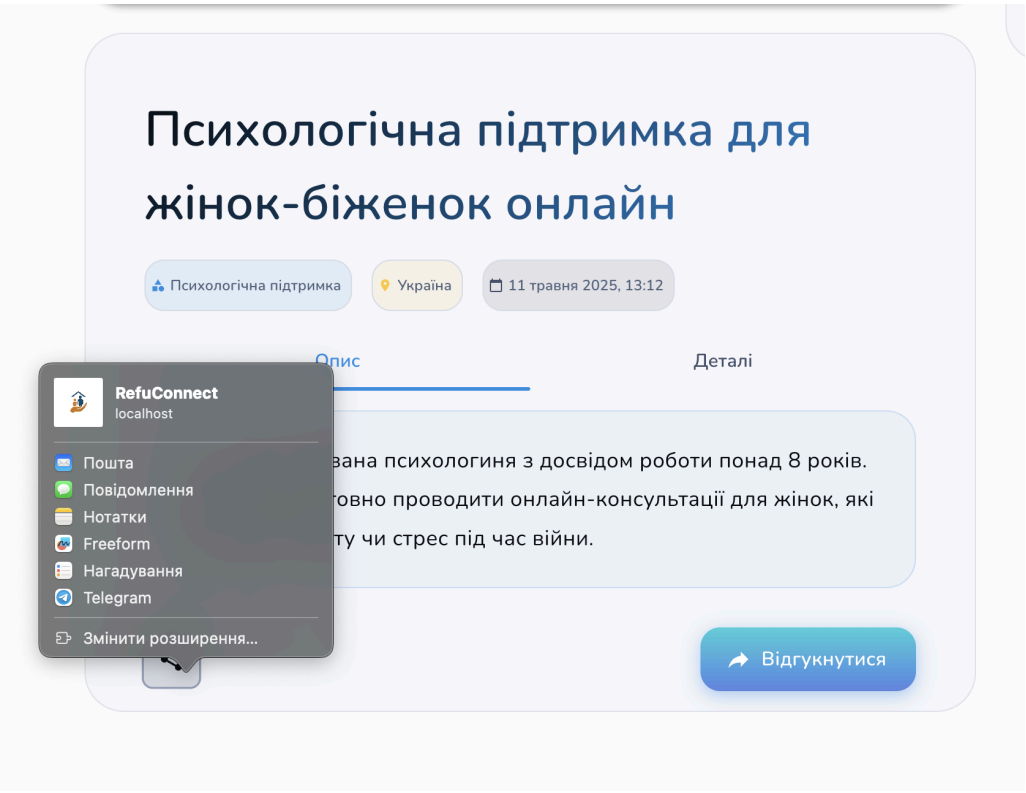


Рисунок 3.48 – Можливість поділитись пропозицією

Джерело: розроблено автором самостійно

Рисунок 3.48 показує можливість поділитися пропозицією. Користувач може скопіювати пряме посилання або надіслати запит через месенджер або соціальні мережі.

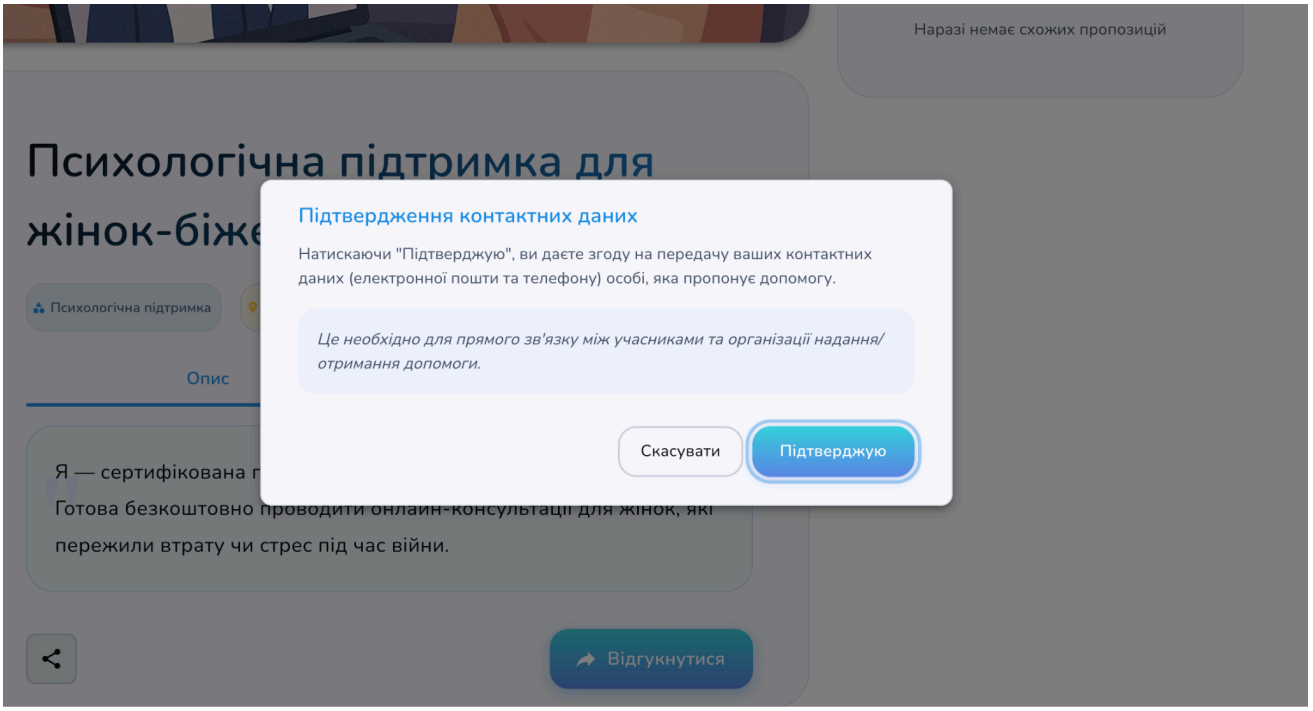


Рисунок 3.49 – підтвердження передачі конт. даних (для пропозиції)

Джерело: розроблено автором самостійно

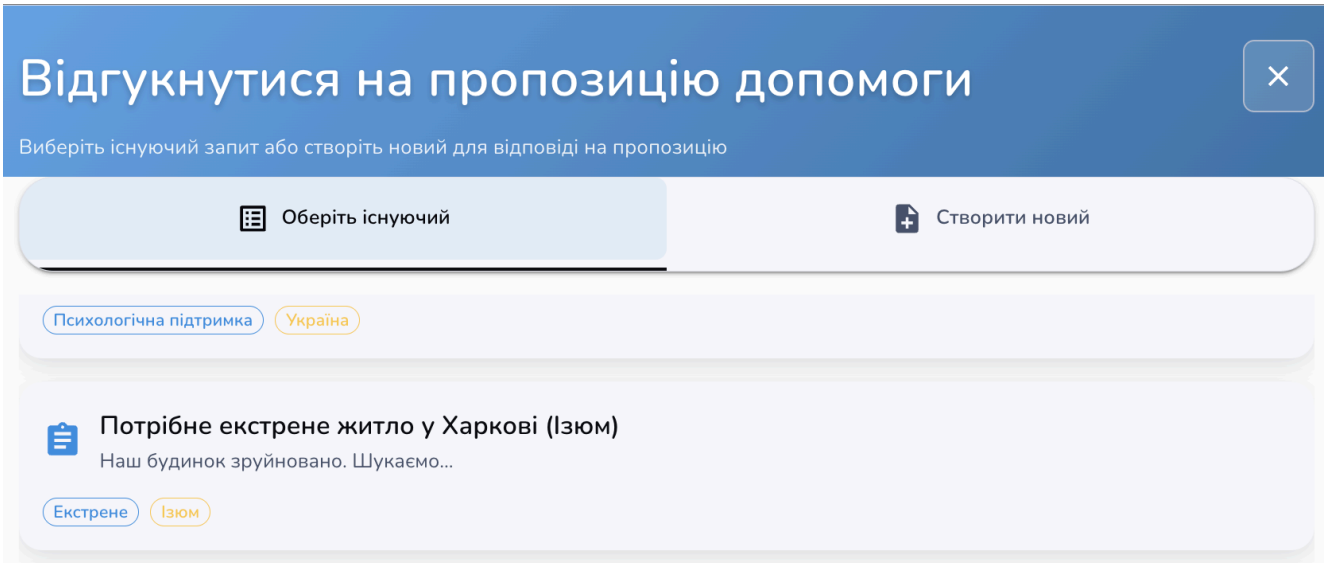


Рисунок 3.50 – Вікно для відгуку на пропозицію(таба 1)

Джерело: розроблено автором самостійно

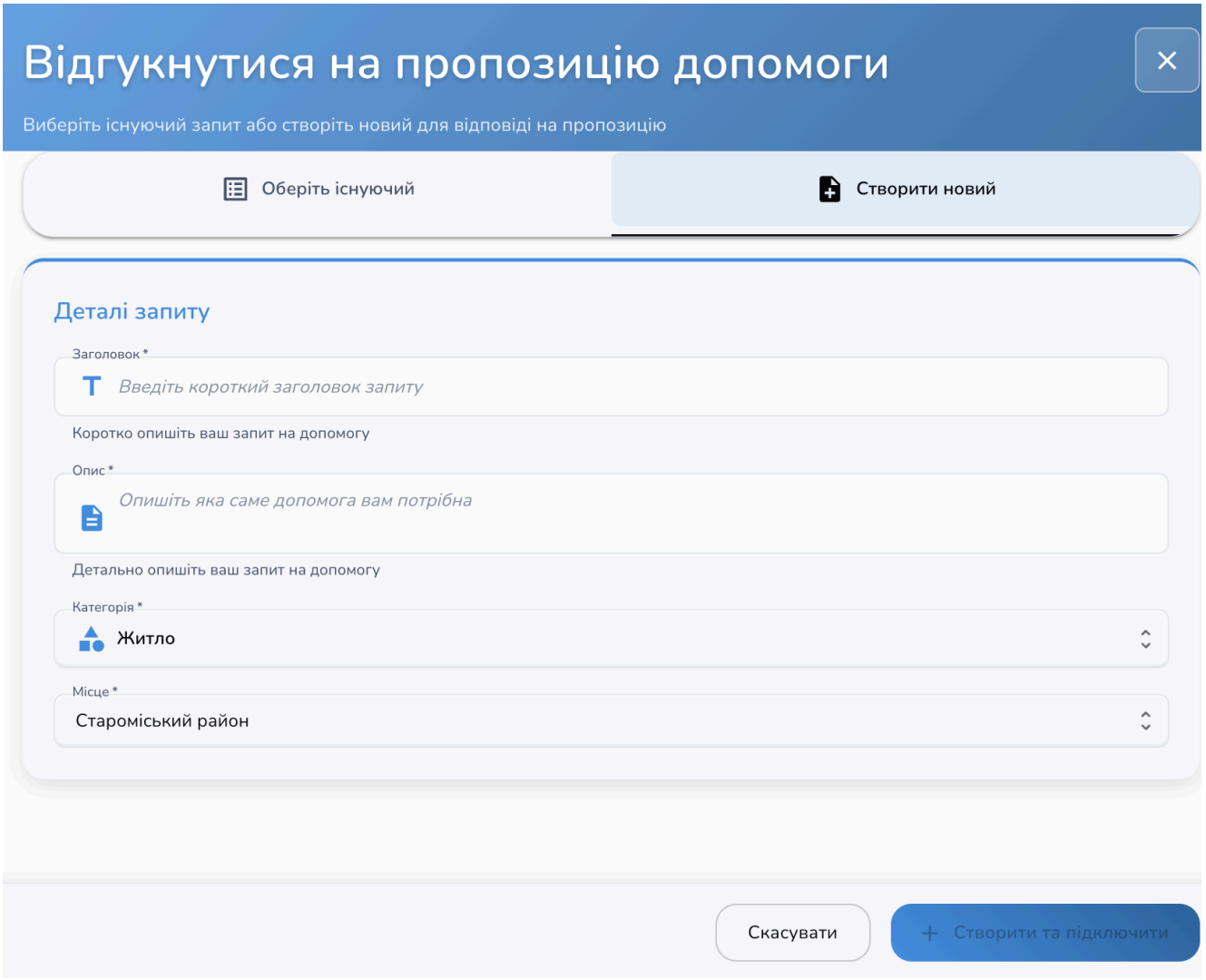


Рисунок 3.51 – Вікно для відгуку на пропозицію (таба 2)

Джерело: розроблено автором самостійно

На рисунках 3.43–3.44 представлено діалогове вікно, що відкривається у відповідь на пропозицію на допомогу. Користувачеві надається два варіанти дій - обрати один з наявних запитів або створити новий, відповідно до характеру пропозиції. Категорія і місце беруться з самої пропозиції.

ВИСНОВКИ

У межах виконання кваліфікаційної магістерської роботи здійснено комплексне дослідження проблеми цифрової координації гуманітарної допомоги та створено веб-платформу, що забезпечує безпечну, масштабовану й інклюзивну взаємодію між біженцями та волонтерами. Об’єктом дослідження виступили інформаційні управляючі системи у сфері гуманітарної підтримки, а предметом архітектурні, функціональні й технологічні рішення, здатні реалізувати двосторонню модель «запит та пропозиція» з дотриманням принципів Data Responsibility та рекомендацій OWASP.

У вступі обґрунтовано актуальність теми, що зумовлена зростанням масштабів примусової міграції та недостатньою ефективністю наявних онлайн-сервісів. Було сформульовано мету - це дослідити сучасні підходи до побудови гуманітарних ІУС та розробити прототип платформи, який дозволить біженцям публікувати структуровані запити на допомогу, а волонтерам пропозиції, гарантувавши високу доступність і захист персональних даних. Для досягнення мети було поставлено завдання проаналізувати нормативно-правову базу і практику роботи існуючих систем, спроєктувати концептуальну модель, обґрунтувати вибір технологічного стеку, реалізувати прототип і перевірити його відповідність функціональним та безпековим вимогам.

Перший розділ містить системний огляд міжнародних і локальних гуманітарних платформ UNHCR Refugee App, Flüchtlinge Willkommen, Choose Love тощо. Показано, що жоден із проаналізованих сервісів не поєднує дворольову модель, горизонтальне масштабування та наскрізне шифрування даних. Виявлено мінуси, такі як відсутність персоналізованих фільтрів, складна процедура ідентифікації, нестача аналітики поведінкових патернів. На цій основі сформульовано вимоги до нової системи, такі як адаптивний SPA-інтерфейс, розширювана категорійна модель, прозорий аудит даних і можливість роботи на малопотужних пристроях.

Другий розділ присвячено методології та архітектурі. За результатами порівняльного аналізу обрано багатошарову Clean-архітектуру, у якій фронтенд реалізовано на React 18 з TypeScript, а бекенд на ASP.NET Core 8 Web API з використанням Entity Framework Core. Для зберігання даних обґрунтовано вибір Microsoft SQL Server; спроєктовано логічну та фізичну модель бази, нормалізовані до 3НФ. Ключовою інновацією стала атомарна сутність HelpAssignment, яка синхронізує стани запиту й пропозиції та зберігає повну історію взаємодій. Архітектура підтримує контейнеризацію, автоматизовані міграції БД і CI/CD-конвеєр GitHub Actions, що гарантує безперервну інтеграцію та розгортання.

Третій розділ висвітлює практичну реалізацію та результати тестування. Розроблено модулі реєстрації, керування запитами й пропозиціями, механізм призначень, систему сповіщень, особисті кабінети та універсальну форму фільтрації. Усі інтерфейсні компоненти побудовано на Material UI. Проведено юніт, інтеграційне та наскрізне тестування, що підтвердило коректність бізнес-логіки й відповідність вимогам безпеки. Контрольний приклад засвідчив, що стартовий обсяг БД з можливістю резерву забезпечує стабільну роботу системи при прогнозованих навантаженнях.

Наукова новизна роботи полягає у поєднанні принципів Data Responsibility, Clean Architecture та OWASP-орієнтованої моделі безпеки в єдиній гуманітарній платформі, а також у запропонованій моделі узгодження «запит - пропозиція» на базі сутності HelpAssignment, що може бути адаптована до інших соціально орієнтованих ІУС. Практична значущість полягає у можливості оперативного впровадження проєкту у діяльність неурядових організацій, муніципальних служб і волонтерських мереж. Контейнеризована інфраструктура спрощує масштабування, а багатомовний адаптивний інтерфейс забезпечує доступність для користувачів з обмеженим інтернетом і низькою цифровою грамотністю.

Перспективи розвитку платформи передбачають інтеграцію модуля машинного навчання для автоматичної пріоритизації запитів і прогнозування потреб, створення офлайн-мобільного застосунку, перехід до мікросервісної

архітектури з GraphQL-API та впровадження аналітичного дашборда поведінкових патернів користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. UNHCR. Digital Transformation Strategy 2022–2026. — Женева: UNHCR, 2022. — URL: https://emergency.unhcr.org/sites/default/files/2024-06/Digital-Transformation-Strategy-2022-2026-UNHCR-Web.pdf
2. IOM. Digital Identity Toolkit: Immigration and Border Governance Unit. — Женева: IOM, 2023. — URL: https://publications.iom.int/books/iom-digital-identity-toolkit-immigration-and-border-governance-unit
3. Digital Public Goods Alliance. 2024 State of the Digital Public Goods Ecosystem. — Осло: DPGA, 2024. — URL: https://www.digitalpublicgoods.net/2024-DPG-Ecosystem-Report
4. UNHCR. The 1951 Refugee Convention. — Женева: UNHCR, 2023. — URL: https://www.unhcr.org/about-unhcr/overview/1951-refugee-convention
5. Cambridge University Press. Robert E. Park's Theory of Assimilation and Beyond. — Cambridge: Cambridge University Press, 2017. — URL: https://www.cambridge.org/core/books/anthem-companion-to-robert-park/robert-e-parks-theory-of-assimilation-and-beyond/3A37239C4BB734CD06F5AD6BF1C3F6EC
6. Comparative Population Studies. Ravenstein Revisited: The Analysis of Migration, Then and Now. — 2019. — URL: https://comparativepopulationstudies.de/index.php/CPoS/article/view/369
7. European Parliament and Council. Regulation (EU) 2016/679 of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation – GDPR). — Official Journal of the European Union, L119, 04.05.2016, p. 1–88. — URL: https://eur-lex.europa.eu/eli/reg/2016/679/oj
8. Botha, R. A., & Thomson, K.-L. An Overview of Access Control Practices: Guidance from ITIL, COBIT 5 and ISO/IEC 27002. — Information Institute

Conferences, Las Vegas, NV, March 29-31, 2016. — URL: https://029e2c6.netsolhost.com/II-Proceedings/2016/Tekeni_Botha_and_Thomson.pdf

9. SEPA (Single Euro Payments Area). Офіційна інформація про єдину зону платежів у євро. — Європейський центральний банк, 2025. — URL: https://www.ecb.europa.eu/paym/integration/retail/sepa/html/index.en.html
10. SWIFT (Society for Worldwide Interbank Financial Telecommunication). Офіційна документація глобальної платіжної системи. — SWIFT, 2025. — URL: https://www.swift.com/about-us
11. European Parliament & Council. Directive (EU) 2015/849 of 20 May 2015 on the prevention of the use of the financial system for the purposes of money laundering or terrorist financing (4th Anti-Money Laundering Directive – AMLD4). — Official Journal of the EU, L141, 05.06.2015, pp. 73–117. — URL: https://eur-lex.europa.eu/eli/dir/2015/849/oj
12. Refugees Welcome Organization. Refugees Welcome – Deine Hilfe für Flüchtlinge in Deutschland. — Берлін: Refugees Welcome, 2023. — URL: https://refugees-welcome.org/
13. Choose Love. Choose Love – гуманітарна підтримка біженців та переміщених осіб. — Лондон: Choose Love, 2024. — URL: https://chooselove.org
14. UNHCR Innovation Service. Project Jetson: Predictive Analytics for Displacement. — Женева: UNHCR, 2023. — URL: https://www.unhcr.org/innovation/project-jetson/
15. React. Офіційний сайт бібліотеки React. — Meta Platforms, 2025. — URL: https://react.dev
16. TypeScript. Офіційна документація мови програмування TypeScript. — Microsoft, 2025. — URL: https://www.typescriptlang.org/docs/
17. React Training. React Router: Declarative Routing for React. — 2024. — URL: https://reactrouter.com
18. Axios Contributors. Axios: Promise Based HTTP Client for the Browser and Node.js. — 2024. — URL: https://axios-http.com

19. Microsoft. ASP.NET Core Web API with .NET 8. — Redmond: Microsoft Docs, 2024. — URL: https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-8.0
20. Microsoft. Entity Framework Core Documentation. — Redmond: Microsoft Docs, 2024. — URL: https://learn.microsoft.com/en-us/ef/core/
21. Fowler, M. Patterns of Enterprise Application Architecture. — Boston: Addison-Wesley, 2003. — ISBN: 978-0321127426. — URL: https://martinfowler.com/eaaCatalog/
22. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). — RFC 7519. — IETF, 2015. — URL: https://datatracker.ietf.org/doc/html/rfc7519
23. Google Identity. OAuth 2.0 for Web Server Applications. — Mountain View: Google Developers, 2024. — URL: https://developers.google.com/identity/protocols/oauth2
24. GitHub. GitHub Actions: Automate your workflow from idea to production. — San Francisco: GitHub, 2024. — URL: https://docs.github.com/en/actions
25. Docker Inc. Docker Documentation: Empowering App Development for Developers. — Palo Alto: Docker, 2024. — URL: https://docs.docker.com
26. W3C. Cross-Origin Resource Sharing (CORS) — W3C Recommendation. — World Wide Web Consortium, 2014. — URL: https://www.w3.org/TR/cors/
27. Internet Engineering Task Force (IETF). The Transport Layer Security (TLS) Protocol Version 1.3 (RFC 8446). — IETF, 2018. — URL: https://datatracker.ietf.org/doc/html/rfc8446
28. MUI. MUI: React components for faster and easier web development. — San Francisco: MUI Inc., 2024. — URL: https://mui.com
29. Microsoft. Routing to controller actions in ASP.NET Core. — Redmond: Microsoft Docs, 2024. — URL: https://learn.microsoft.com/en-us/aspnet/core/mvc/controllers/routing?view=aspnetcore-8.0
30. Nottingham, M., & Wilde, E. Problem Details for HTTP APIs (RFC 7807). — Internet Engineering Task Force (IETF), May 2016. — URL: https://datatracker.ietf.org/doc/html/rfc7807

31. SmartBear Software. Swagger UI: Visualize and Interact with Your API's Resources. — Somerville, MA: SmartBear, 2024. — URL: https://swagger.io/tools/swagger-ui/
32. Microsoft. Domain-driven design (DDD) and .NET. — Redmond: Microsoft Docs, 2024. — URL: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures#domain-driven-design
33. Microsoft. SQL Server 2022 Documentation. — Redmond: Microsoft Docs, 2024. — URL: https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16
34. AutoMapper Project. AutoMapper: A convention-based object-object mapper. — 2024. — URL: https://automapper.org
35. ESLint Contributors. ESLint: Find and fix problems in your JavaScript code. — 2024. — URL: https://eslint.org
36. Prettier Contributors. Prettier: An opinionated code formatter. — 2024. — URL: https://prettier.io
37. Microsoft. Language-Integrated Query (LINQ) in C#. — Redmond: Microsoft Learn, 2024. — URL: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/
38. Microsoft. SQL Server Management Studio (SSMS) Documentation. — Redmond: Microsoft, 2024. — URL: https://learn.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms
39. Microsoft. Active Directory Documentation. — Redmond: Microsoft Docs, 2024. — URL: https://learn.microsoft.com/en-us/windows-server/identity/active-directory-domain-services
40. Microsoft. Dependency injection in ASP.NET Core. — Redmond: Microsoft Docs, 2024. — URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection

41. Microsoft. Introduction to ASP.NET Core Identity. — Redmond: Microsoft Learn, 2024. — URL: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity
42. Fowler, M. Patterns of Enterprise Application Architecture. — Boston: Addison-Wesley, 2003. — ISBN: 978-0321127426. — URL: https://martinfowler.com/eaaCatalog/dataTransferObject.html
43. Microsoft. Middleware in ASP.NET Core. — Redmond: Microsoft Learn, 2024. — URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/middleware
44. Microsoft. Visual Studio 2022 Documentation. — Redmond: Microsoft Docs, 2024. — URL: https://learn.microsoft.com/en-us/visualstudio/releases/2022
45. Microsoft. IntelliSense in Visual Studio. — Redmond: Microsoft Docs, 2024. — URL: https://learn.microsoft.com/en-us/visualstudio/ide/using-intellisense
46. xUnit.net Team. xUnit.net Documentation. — 2024. — URL: https://xunit.net
47. Moq Contributors. Moq: The most popular and friendly mocking library for .NET. — 2024. — URL: https://github.com/moq/moq4
48. Remix Software. React Router v6 Documentation. — 2024. — URL: https://reactrouter.com/en/main
49. Microsoft. What is CI/CD?. — Redmond: Microsoft Learn, 2024. — URL: https://learn.microsoft.com/en-us/devops/deliver/what-is-cicd
50. The Kubernetes Authors. Kubernetes Documentation. — Cloud Native Computing Foundation, 2024. — URL: https://kubernetes.io/docs/
51. Postman, Inc. Postman API Platform Documentation. — San Francisco: Postman, 2024. — URL: https://www.postman.com/product/api-client/
52. Postman, Inc. Newman: Command-line Collection Runner for Postman. — San Francisco: Postman, 2024. — URL: https://www.npmjs.com/package/newman
53. Формування сучасної науки: методика та практика: тези доповідей IV Всеукраїнської студентської наукової конференції (м. Київ, 20 жовтня 2023 р.) / ред. кол.: І. В. Левченко та ін. — Київ: Ліга Наук, 2023. — 242 с. — URL: https://archive.liga.science/index.php/conference-proceedings/issue/view/ukr-20.10.2023?utm_source=eSputnik-promo&utm_medium=email&utm_campaign=M

NL_Konferencia_%7C_Status_:_opublikovano&utm_content=2247777662&etm_term=Y2lkPTIyNDc3Nzc2NjI=.

54. Сучасні інформаційні технології та системи в управлінні [Електронний ресурс]: зб. матеріалів V Міжнар. наук.-практ. конф. молодих вчених, аспірантів і студентів (м. Київ, 18–19 квітня 2024 р.) / редкол.: В. П. Пилипенко та ін. — Київ: КНЕУ, 2024. — 308 с. — URL: https://drive.google.com/file/d/1D8vstTNHJjaXL-5oH8cprgO8CDgE1GCS/view

ДОДАТКИ

Додаток А

Код створення бази даних через EF з міграціями

```
public class RefuConnectContext : IdentityDbContext<ApplicationUser>
{
    public RefuConnectContext(DbContextOptions<RefuConnectContext> options)
        : base(options)
    {
    }

    public DbSet<ApplicationUserPhoto> ApplicationUserPhotos { get; set; }

    public DbSet<Location> Locations { get; set; }
    public DbSet<HelpCategory> HelpCategories { get; set; }
    public DbSet<HelpOfferStatus> HelpOfferStatuses { get; set; }
    public DbSet<HelpRequestStatus> HelpRequestStatuses { get; set; }
    public DbSet<HelpAssignmentStatus> HelpAssignmentStatuses { get; set; }

    public DbSet<HelpOffer> HelpOffers { get; set; }
    public DbSet<HelpOfferPhoto> HelpOfferPhotos { get; set; }
    public DbSet<HelpRequest> HelpRequests { get; set; }
    public DbSet<HelpRequestPhoto> HelpRequestPhotos { get; set; }
    public DbSet<HelpAssignment> HelpAssignments { get; set; }

    protected override void OnModelCreating(ModelBuilder builder)
    {
        base.OnModelCreating(builder);

        builder.Entity<ApplicationUser>(entity =>
        {
```

```
    entity.Property(u => u.FullName)
        .IsRequired()
        .HasMaxLength(100);
});

builder.Entity<ApplicationUserPhoto>(entity =>
{
    entity.Property(p => p.PhotoPath)
        .IsRequired()
        .HasMaxLength(300);
    entity.HasOne(p => p.ApplicationUser)
         .WithOne(u => u.Photo)
         .HasForeignKey<ApplicationUserPhoto>(p => p.UserId)
         .IsRequired()
         .OnDelete(DeleteBehavior.Cascade);
});

builder.Entity<Location>(entity =>
{
    entity.Property(l => l.Name)
        .IsRequired()
        .HasMaxLength(100);

    entity.HasOne(l => l.ParentLocation)
        .WithMany(l => l.SubLocations)
        .HasForeignKey(l => l.ParentLocationId)
        .OnDelete(DeleteBehavior.Restrict);
});

builder.Entity<HelpCategory>(entity =>
{
```

```
    entity.HasOne(hc => hc.ParentCategory)
          .WithMany(hc => hc.SubCategories)
          .HasForeignKey(hc => hc.ParentCategoryId)
          .OnDelete(DeleteBehavior.Restrict);
});

builder.Entity<HelpOfferStatus>(entity =>
{
    entity.Property(s => s.Name)
          .IsRequired()
          .HasMaxLength(50);
});

builder.Entity<HelpRequestStatus>(entity =>
{
    entity.Property(s => s.Name)
          .IsRequired()
          .HasMaxLength(50);
});

builder.Entity<HelpAssignmentStatus>(entity =>
{
    entity.Property(s => s.Name)
          .IsRequired()
          .HasMaxLength(50);
});

builder.Entity<HelpOffer>(entity =>
{
    entity.Property(h => h.Title)
          .IsRequired()
```

```
            .HasMaxLength(200);
        entity.Property(h => h.Description)
            .IsRequired();

        entity.HasOne(h => h.ApplicationUser)
            .WithMany(u => u.HelpOffers)
            .HasForeignKey(h => h.ApplicationUserId)
            .IsRequired()
            .OnDelete(DeleteBehavior.Cascade);

        entity.HasOne(h => h.HelpCategory)
            .WithMany(c => c.HelpOffers)
            .HasForeignKey(h => h.HelpCategoryId)
            .IsRequired()
            .OnDelete(DeleteBehavior.Restrict);

        entity.HasOne(h => h.HelpOfferStatus)
            .WithMany(s => s.HelpOffers)
            .HasForeignKey(h => h.HelpOfferStatusId)
            .IsRequired()
            .OnDelete(DeleteBehavior.Restrict);

        entity.HasOne(h => h.Location)
            .WithMany()
            .HasForeignKey(h => h.LocationId)
            .IsRequired()
            .OnDelete(DeleteBehavior.Restrict);
    });

    builder.Entity<HelpOfferPhoto>(entity =>
    {
```

```
    entity.Property(p => p.PhotoPath)
        .IsRequired()
        .HasMaxLength(300);

    entity.HasOne(p => p.HelpOffer)
        .WithMany(o => o.Photos)
        .HasForeignKey(p => p.HelpOfferId)
        .IsRequired()
        .OnDelete(DeleteBehavior.Cascade);
});

builder.Entity<HelpRequest>(entity =>
{
    entity.Property(r => r.Title)
        .IsRequired()
        .HasMaxLength(200);
    entity.Property(r => r.Description)
        .IsRequired();

    entity.HasOne(r => r.ApplicationUser)
        .WithMany(u => u.HelpRequests)
        .HasForeignKey(r => r.ApplicationUserId)
        .IsRequired()
        .OnDelete(DeleteBehavior.Cascade);

    entity.HasOne(r => r.HelpCategory)
        .WithMany(c => c.HelpRequests)
        .HasForeignKey(r => r.HelpCategoryId)
        .IsRequired()
        .OnDelete(DeleteBehavior.Restrict);
```

```
    entity.HasOne(r => r.HelpRequestStatus)
        .WithMany(s => s.HelpRequests)
        .HasForeignKey(r => r.HelpRequestStatusId)
        .IsRequired()
        .OnDelete(DeleteBehavior.Restrict);

    entity.HasOne(r => r.Location)
        .WithMany()
        .HasForeignKey(r => r.LocationId)
        .IsRequired()
        .OnDelete(DeleteBehavior.Restrict);
});

builder.Entity<HelpRequestPhoto>(entity =>
{
    entity.Property(p => p.PhotoPath)
        .IsRequired()
        .HasMaxLength(300);
    entity.HasOne(p => p.HelpRequest)
        .WithMany(r => r.Photos)
        .HasForeignKey(p => p.HelpRequestId)
        .IsRequired()
        .OnDelete(DeleteBehavior.Cascade);
});

builder.Entity<HelpAssignment>(entity =>
{
    entity.HasOne(a => a.HelpRequest)
        .WithMany(r => r.HelpAssignments)
        .HasForeignKey(a => a.HelpRequestId)
        .IsRequired()
```

```
            .OnDelete(DeleteBehavior.Restrict);

        entity.HasOne(a => a.HelpOffer)
            .WithMany(o => o.HelpAssignments)
            .HasForeignKey(a => a.HelpOfferId)
            .IsRequired()
            .OnDelete(DeleteBehavior.Restrict);

        entity.HasOne(a => a.HelpAssignmentStatus)
            .WithMany(s => s.HelpAssignments)
            .HasForeignKey(a => a.HelpAssignmentStatusId)
            .IsRequired()
            .OnDelete(DeleteBehavior.Restrict);
    });
  }
}
```

Додаток Б

Код контроллера для пропозицій на серверній частині

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using RefuConnect.DTOs.HelpOffer;
using RefuConnect.DTOs.Parametrs;
using RefuConnect.DTOs.User;
using RefuConnect.Models;
using RefuConnect.Services.Interfaces;

namespace RefuConnect.Controllers
{
   [Route("api/[controller]")]
   [ApiController]
   public class HelpOfferController : ControllerBase
   {
      private readonly IHelpOfferService _helpOfferService;
      private readonly UserManager<ApplicationUser> _userManager;

                        public HelpOfferController(IHelpOfferService helpOfferService,
UserManager<ApplicationUser> userManager)
      {
         _helpOfferService = helpOfferService;
         _userManager = userManager;
      }

      [Authorize]
      [HttpGet]
                  public async Task<IActionResult> GetAllHelpOffers([FromQuery]
HelpOfferQueryParameters queryParameters)
```

```
        {
            var response = await _helpOfferService.GetAllHelpOffers(queryParameters);
            if (response.Data == null)
            {
                return BadRequest(new { message = response.Message });
            }

            return Ok(new { data = response.Data, totalCount = response.TotalCount });
        }

        [Authorize]
        [HttpGet("{id}")]
        public async Task<IActionResult> GetHelpOfferById(Guid id)
        {
            var response = await _helpOfferService.GetDetailsHelpOfferById(id);
            if (response.Data == null)
            {
                return BadRequest(new { message = response.Message });
            }

            return Ok(response.Data);
        }

        [Authorize(Roles = "Helper")]
        [HttpPost]
         public async Task<IActionResult> AddHelpOffer([FromBody] HelpOfferAddDto
newHelpOffer)
        {
            var userId = _userManager.GetUserId(User);
            if (userId == null)
            {
```

```
            return BadRequest(new { message = "User not found" });
        }

        var response = await _helpOfferService.AddHelpOffer(newHelpOffer, userId);
        if (!response.Success)
        {
            return BadRequest(new { message = response.Message });
        }


        return Ok(response.Data);
    }

    [Authorize(Roles= "Helper,Admin")]
    [HttpPut("{id}")]
        public async Task<IActionResult> UpdateHelpOffer(Guid id, [FromBody]
HelpOfferUpdateDto updatedHelpOffer)
    {
        var userId = _userManager.GetUserId(User);
        if (userId == null)
        {
            return BadRequest(new { message = "User not found" });
        }

         var response = await _helpOfferService.UpdateHelpOffer(updatedHelpOffer,
userId);
        if (!response.Success)
        {
            return BadRequest(new { message = response.Message });
        }
```

```
            return Ok(response.Data);
        }

        [Authorize(Roles = "Helper,Admin")]
        [HttpPut("{id}/status")]
          public async Task<IActionResult> UpdateHelpOfferStatus(Guid id, [FromBody]
UpdateHelpOfferStatusDto dto)
        {
            var userId = _userManager.GetUserId(User);
            if (userId == null)
            {
                return BadRequest(new { message = "User not found" });
            }

                var response = await _helpOfferService.UpdateHelpOfferStatus(id, userId,
dto.Status);
            if (!response.Success)
            {
                return BadRequest(new { message = response.Message });
            }

            return Ok(response.Data);
        }



        [Authorize(Roles = "Helper,Admin")]
        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteHelpOffer(Guid id)
        {
            var userId = _userManager.GetUserId(User);
            if (userId == null)
```

```
            {
                return BadRequest(new { message = "User not found" });
            }

            var response = await _helpOfferService.CancelHelpOffer(id, userId);
            if (!response.Success)
            {
                return BadRequest(new { message = response.Message });
            }

            return Ok(response.Data);
        }
    }
}
```

Додаток В

Код сервісу для пропозицій на серверній частині

```
using AutoMapper;
using RefuConnect.DTOs.HelpOffer;
using RefuConnect.DTOs.HelpOffer.RefuConnect.DTOs.HelpOffer;
using RefuConnect.DTOs.Parametrs;
using RefuConnect.DTOs.User;
using RefuConnect.Enums;
using RefuConnect.Models;
using RefuConnect.Repositories.Interfaces;
using RefuConnect.Services.Interfaces;

namespace RefuConnect.Services
{
   public class HelpOfferService : IHelpOfferService
   {
      private readonly IHelpOfferRepository _helpOfferRepository;
      private readonly IMapper _mapper;

         public HelpOfferService(IHelpOfferRepository helpOfferRepository, IMapper
mapper)
      {
         _helpOfferRepository = helpOfferRepository;
         _mapper = mapper;
      }

            public async Task<ServiceQueryResponse<IEnumerable<HelpOfferDto>>>
GetAllHelpOffers(HelpOfferQueryParameters queryParameters)
      {
                                          var (offers, totalCount) = await
_helpOfferRepository.GetAllAsync(queryParameters);
```

```
        var offersDto = _mapper.Map<IEnumerable<HelpOfferDto>>(offers);

        if (offersDto == null)
            return new ServiceQueryResponse<IEnumerable<HelpOfferDto>>
            {
                Success = false,
                Message = "No offers found"
            };

        return new ServiceQueryResponse<IEnumerable<HelpOfferDto>>
        {
            Data = offersDto,
            TotalCount = totalCount,
            Message = "Offers found"
        };
    }

                    public async Task<ServiceResponse<HelpOfferDetailsDto>>
GetDetailsHelpOfferById(Guid id)
    {
        var offer = await _helpOfferRepository.GetByIdAsync(id);
        if (offer == null)
        {
            return new ServiceResponse<HelpOfferDetailsDto>
            {
                Success = false,
                Message = "Offer not found"
            };
        }
        var offerDetailsDto = _mapper.Map<HelpOfferDetailsDto>(offer);
```

```
        var similarOffers = await _helpOfferRepository.GetSimilarOffersAsync(offer.Id,
offer.HelpCategoryId, offer.LocationId);
                                                        var similarOffersDto =
_mapper.Map<IEnumerable<HelpOfferShortDto>>(similarOffers);
        offerDetailsDto.SimilarOffers = similarOffersDto.ToList();
        offerDetailsDto.Creator = _mapper.Map<UserDto>(offer.ApplicationUser);
        return new ServiceResponse<HelpOfferDetailsDto>
        {
            Data = offerDetailsDto,
            Message = "Offer found"
        };
    }

    public async Task<ServiceResponse<HelpOfferDto>> GetHelpOfferById(Guid id)
    {
        var offer = await _helpOfferRepository.GetByIdAsync(id);
        if (offer == null)
        {
            return new ServiceResponse<HelpOfferDto>
            {
                Success = false,
                Message = "Offer not found"
            };
        }
        var offerDto = _mapper.Map<HelpOfferDto>(offer);
        return new ServiceResponse<HelpOfferDto>
        {
            Data = offerDto,
            Message = "Offer found"
        };
    }
```

```
                public async Task<ServiceResponse<HelpOfferDto>>
AddHelpOffer(HelpOfferAddDto newHelpOffer, string userId)
    {
        var offer = _mapper.Map<HelpOffer>(newHelpOffer);
        offer.ApplicationUserId = userId;
        offer.HelpOfferStatusId = (int)HelpOfferStatusEnum.Pending;
        await _helpOfferRepository.CreateAsync(offer);
        var offerDto = _mapper.Map<HelpOfferDto>(offer);

        return new ServiceResponse<HelpOfferDto>
        {
            Data = offerDto,
            Message = "Offer created"
        };
    }

                public async Task<ServiceResponse<HelpOfferDto>>
UpdateHelpOffer(HelpOfferUpdateDto updatedHelpOffer, string userId)
    {
        var offer = await _helpOfferRepository.GetByIdAsync(updatedHelpOffer.Id);

        if (offer == null)
            return new ServiceResponse<HelpOfferDto>
            {
                Success = false,
                Message = "Offer not found",
            };

        if (offer.ApplicationUserId != userId)
            return new ServiceResponse<HelpOfferDto>
```

```
            {
                Success = false,
                Message = "You are not the owner of this offer"
            };

        _mapper.Map(updatedHelpOffer, offer);
        await _helpOfferRepository.UpdateAsync(offer);
        var offerDto = _mapper.Map<HelpOfferDto>(offer);

        return new ServiceResponse<HelpOfferDto>
        {
            Data = offerDto,
            Message = "Offer updated"
        };
    }

                                public async Task<ServiceResponse<HelpOfferDto>>
UpdateHelpOfferStatus(Guid id, string userId, HelpOfferStatusEnum status)
    {
        var offer = await _helpOfferRepository.GetByIdAsync(id);
        if (offer == null)
            return new ServiceResponse<HelpOfferDto>
            {
                Success = false,
                Message = "Offer not found"
            };
        if (offer.ApplicationUserId != userId)
            return new ServiceResponse<HelpOfferDto>
            {
                Success = false,
                Message = "You are not the owner of this offer"
```

```
            };
        offer.HelpOfferStatusId = (int)status;
        await _helpOfferRepository.UpdateAsync(offer);
        var offerDto = _mapper.Map<HelpOfferDto>(offer);
        return new ServiceResponse<HelpOfferDto>
        {
            Data = offerDto,
            Message = "Offer status updated"
        };
    }

     public async Task<ServiceResponse<Guid>> CancelHelpOffer(Guid id, string
userId)
    {
        var offer = await _helpOfferRepository.GetByIdAsync(id);

        if (offer == null)
            return new ServiceResponse<Guid>
            {
                Success = false,
                Message = "Offer not found"
            };

        if (offer.ApplicationUserId != userId)
        {
            return new ServiceResponse<Guid>
            {
                Success = false,
                Message = "You are not the owner of this offer"
            };
        }
```

```
            offer.HelpOfferStatusId = (int)HelpOfferStatusEnum.Cancelled;

            await _helpOfferRepository.SoftDeleteAsync(offer);
            return new ServiceResponse<Guid>
            {
                Data = id,
                Message = "Offer Cancelled"
            };
        }
    }
}
```

Додаток Г

Код слайсу для пропозицій на клієнтській частині

```
import { AnyAction, createSlice, isAnyOf, PayloadAction } from "@reduxjs/toolkit";
import { HelpOfferState } from "./types";
import { getAllHelpOffers, getHelpOffer, addHelpOffer, updateHelpOffer, deleteHelpOffer, updateHelpOfferStatus } from "./helpOfferThunks";
import { HelpOfferDto } from "../../DTOs/HelpOffer/HelpOfferDto";
import { HelpOfferDetailsDto } from "../../DTOs/HelpOffer/HelpOfferDetailsDto";


const initialState: HelpOfferState = {
  helpOffers: [],
  helpOffer: null,
  isLoading: false,
  totalCount: 0,
  error: null,
};

// Pending handler (sets isLoading and resets error)
const handlePending = (state: HelpOfferState) => {
  state.isLoading = true;
  state.error = null;
};

// Rejected handler (sets error message and disables loading)
const handleRejected = (state: HelpOfferState, action: AnyAction) => {
  state.isLoading = false;
  state.error = action.payload || "Operation failed";
};

// Slice
```

```
const helpOfferSlice = createSlice({
  name: "helpOffer",
  initialState,
  reducers: {
    resetHelpOffer: (state) => {
      state.helpOffer = null;
    },
    resetError: (state) => {
      state.error = null;
    },
    resetHelpOffers: (state) => {
      state.helpOffers = [];
    },
    resetHelpOffersLoading: (state) => {
      state.isLoading = false;
    },
  },

  extraReducers: (builder) => {
    // GET ALL HELP OFFERS
    builder.addCase(getAllHelpOffers.fulfilled,
      (state, action: PayloadAction<{ data: HelpOfferDto[], totalCount: number }>) => {
        state.helpOffers = action.payload.data;
        state.isLoading = false;
        state.totalCount = action.payload.totalCount;
      }
    )

    // GET HELP OFFER
    builder.addCase(getHelpOffer.fulfilled,
      (state, action: PayloadAction<HelpOfferDetailsDto>) => {
```

```
    state.helpOffer = action.payload;
    state.isLoading = false;
  }
)

// ADD HELP OFFER
builder.addCase(addHelpOffer.fulfilled,
  (state, action: PayloadAction<HelpOfferDto>) => {
    state.helpOffers.push(action.payload);
    state.isLoading = false;
  }
)

// UPDATE HELP OFFER
builder.addCase(updateHelpOffer.fulfilled,
  (state, action: PayloadAction<HelpOfferDto>) => {
    const index = state.helpOffers.findIndex((x) => x.id === action.payload.id);
    state.helpOffers[index] = action.payload;
    state.isLoading = false;
  }
)

// UPDATE HELP OFFER STATUS
builder.addCase(updateHelpOfferStatus.fulfilled,
  (state, action: PayloadAction<HelpOfferDto>) => {
    const index = state.helpOffers.findIndex((x) => x.id === action.payload.id);
    if (index !== -1) {
      state.helpOffers[index] = action.payload;
    }
    state.isLoading = false;
  }
```

```
)

// DELETE HELP OFFER
builder.addCase(deleteHelpOffer.fulfilled,
  (state, action: PayloadAction<string>) => {
    state.isLoading = false;
    state.helpOffer = null;
    if (state.helpOffers) {
      state.helpOffers = state.helpOffers.filter((x) => x.id !== action.payload);
      state.totalCount = Math.max(0, state.totalCount - 1);
    }
  }
)

// Apply pending/rejected handlers to all async thunks
builder.addMatcher(
  isAnyOf(
    getAllHelpOffers.pending,
    getHelpOffer.pending,
    addHelpOffer.pending,
    updateHelpOffer.pending,
    updateHelpOfferStatus.pending,
    deleteHelpOffer.pending
  ),
  handlePending
);

builder.addMatcher(
  isAnyOf(
    getAllHelpOffers.rejected,
    getHelpOffer.rejected,
```

```
        addHelpOffer.rejected,
        updateHelpOffer.rejected,
        updateHelpOfferStatus.rejected,
        deleteHelpOffer.rejected
      ),
      handleRejected
    );
  }
});

export const { resetHelpOffer, resetError, resetHelpOffers, resetHelpOffersLoading } =
helpOfferSlice.actions;
export default helpOfferSlice.reducer;
```

Додаток Д

Код апі для пропозицій на клієнтській частині

```
import { HelpOfferAddDto } from "../../DTOs/HelpOffer/HelpOfferAddDto";
import { HelpOfferUpdateDto } from "../../DTOs/HelpOffer/HelpOfferUpdateDto";
import { HelpOfferQueryParameters } from "../../DTOs/Parametrs/HelpOfferQueryParameters";
import api from "../../services/api/api";


export const getAllHelpOffersAsync = async (query: HelpOfferQueryParameters) => {
   const response = await api.get("/api/HelpOffer", { params: query });
   return response.data;
}

export const getHelpOfferAsync = async (id: string) => {
   const response = await api.get(`/api/HelpOffer/${id}`);
   return response.data;
}

export const addHelpOfferAsync = async (data: HelpOfferAddDto) => {
   const response = await api.post("/api/HelpOffer", data);
   return response.data;
}

export const updateHelpOfferAsync = async (data: HelpOfferUpdateDto) => {
   const response = await api.put("/api/HelpOffer", data);
   return response.data;
}

export const updateHelpOfferStatusAsync = async (arg: { id: string, status: number }) => {
```

```
    const response = await api.put(`/api/HelpOffer/${arg.id}/status`, { status: arg.status
});
  return response.data;
}

export const deleteHelpOfferAsync = async (id: string) => {
  const response = await api.delete(`/api/HelpOffer/${id}`);
  return response.data;
}
```

Додаток Е

Код для CI/CD (frontend) пайплайну

```
name: Frontend CI/CD

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

concurrency:
  group: frontend-ci-${{ github.ref }}
  cancel-in-progress: true

jobs:
  build-and-test:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Cache node_modules
        uses: actions/cache@v4
        with:
          path: ./frontend/node_modules
          key: ${{ runner.OS }}-node-${{ hashFiles('**/package-lock.json') }}
          restore-keys: ${{ runner.OS }}-node-

      - name: Setup Node.js
        uses: actions/setup-node@v1
```

```
  with:
    node-version: 20

- name: Install dependencies
  run: npm install
  working-directory: ./frontend
```

Додаток Є

Код Dockerfile (frontend)

```
# frontend/Dockerfile

# ============================================================
# 1. STAGE: BUILD
# ============================================================
FROM node:18-alpine AS build

WORKDIR /app

COPY package.json package-lock.json ./
RUN npm ci

COPY . .

RUN npm run build



# ============================================================
# 2. STAGE: RUNTIME (Nginx)
# ============================================================
FROM nginx:alpine

COPY --from=build /app/build /usr/share/nginx/html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

МОЛОДІЖНА НАУКОВА ЛІГА

МАТЕРІАЛИ КОНФЕРЕНЦІЇ

IV ВСЕУКРАЇНСЬКА СТУДЕНТСЬКА НАУКОВА КОНФЕРЕНЦІЯ

ФОРМУВАННЯ СУЧАСНОЇ НАУКИ: МЕТОДИКА ТА ПРАКТИКА

20 ЖОВТНЯ **2023** РІК м. КИЇВ, УКРАЇНА

Вінниця, Україна
«UKRLOGOS Group»
2023

УДК **082:001**
Е 79

Голова оргкомітету: Коренюк І.О.
Верстка: Зрада С.І.
Дизайн: Бондаренко І.В.

Конференцію зареєстровано Державною науковою установою «УкрІНТЕІ» в базі даних науково-технічних заходів України та інформаційному бюлетені «План проведення наукових, науково- технічних заходів в Україні» (Посвідчення №323 від 16.06.2023).

Ф 79

Формування сучасної науки: методика та практика: матеріали IV Всеукраїнської студентської наукової конференції, м. Київ, 20 жовтня, 2023 рік / ГО «Молодіжна наукова ліга». — Вінниця: ТОВ «УКРЛОГОС Груп», 2023. — 242 с.

ISBN 978-617-8126-71-1
DOI 10.36074/liga-ukr-20.10.2023

Викладено матеріали учасників IV Всеукраїнської мультидисциплінарної студентської наукової конференції «Формування сучасної науки: методика та практика», яка відбулася 20 жовтня 2023 року у місті Київ, Україна.

УДК 082:001

ISBN 978-617-8126-63-6

СЕКЦІЯ 14. ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА СИСТЕМИ

АНАЛІЗ НАПРЯМІВ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
Новак К.М., ***Науковий керівник: Устенко С.В***
.....
122

АНАЛІЗ ТА РОЗВИТОК ІНФОРМАЦІЙНИХ СИСТЕМ ДЛЯ ІНВЕСТУВАННЯ
Датко А.П., ***Науковий керівник: Устенко С.В***
.....
125

ЗАБЕЗПЕЧЕННЯ КОНФІДЕНЦІЙНОСТІ ДАНИХ ПРИ ВИКОРИСТАННІ AR-ТЕХНОЛОГІЙ
Бондарчук Я.В., ***Науковий керівник: Казимира І.Я***
.....
128

ІНТЕЛЕКТУАЛЬНИЙ ПІДХІД ДО РОЗШИРЕНОЇ РЕАЛЬНОСТІ ДЛЯ ЗБАГАЧЕННЯ КУЛЬТУРНОЇ СПАДЩИНИ: ПЕРЕВАГИ ПОРІВНЯНО З АНАЛОГАМИ
Пісний О.В., ***Науковий керівник: Ліпяніна-Гончаренко Х***
.....
130

ІНФОРМАЦІЙНА СИСТЕМА УПРАВЛІННЯ ПРОЄКТАМИ
Бондаренко М.Д., ***Науковий керівник: Устенко С.В.***
.....
132

МЕСЕНДЖЕР З ПІДТРИМКОЮ ВІРТУАЛЬНОЇ РЕАЛЬНОСТІ: ІНТЕРФЕЙСИ ТА МОЖЛИВОСТІ ВЗАЄМОДІЇ
Василькевич М.В., ***Науковий керівник: Ріпак Н.С***
.....
135

ОПТИМІЗАЦІЯ ВЕБ-ДОДАТКІВ ЗА ДОПОМОГОЮ PWA
Бондар Ю.С., ***Науковий керівник: Устенко С.В***
.....
137

ОПТИМІЗАЦІЯ ОБРОБКИ ЗОВНІШНІХ НЕ СТРУКТУРОВАНИХ ПАПЕРОВИХ ТА ЕЛЕКТРОННИХ ФІНАНСОВИХ ДОКУМЕНТІВ
Урдін А.Г., ***Науковий Керівник: Устенко С.В***
.....
140

РОЗРОБКА ВЕБ-ДОДАТКІВ З ВИКОРИСТАННЯМ АРХІТЕКТУРНОГО СТИЛЮ RESTFUL API
Безпалько В.М., ***Науковий керівник: Устенко С.В***
.....
145

Напрям конференції: Інформаційні системи та технології.

Предметна область: Розробка інформаційних систем.

Тема дослідження: Оптимізація веб-додатків за допомогою PWA

Бондар Юліана Сергіївна

здобувач вищої освіти інституту інформаційних технологій в економіці

Київський національний економічний університет ім.Вадима Гетьмана, Україна

Науковий керівник: Устенко Станіслав Веніамінович

док. екон. наук, проф., професор кафедри інформаційних систем в економіці,

Київський національний економічний університет ім.Вадима Гетьмана, Україна

Оптимізація веб-додатків за допомогою PWA

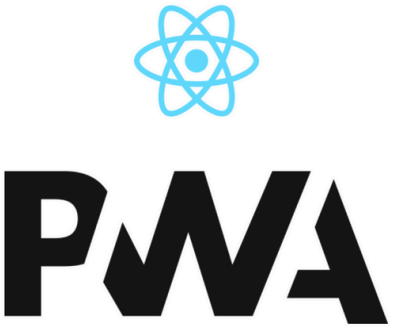


Рис. 1: **LOGO PWA and React**

Прогресивні веб-додатки (PWA) стають все більш популярними в світі веб-розробки [1]. Вони дозволяють створювати високоякісні, надійні та швидкі веб-додатки, які можуть працювати офлайн та надавати користувачам досвід, подібний до роботи з нативними додатками. У рамках цієї тези буде розглянуто оптимізацію веб-додатків за допомогою PWA з використанням таких технологій, як React [4].

React: це одна з найпопулярніших бібліотек для розробки інтерфейсу користувача. Вона дозволяє створювати інтерактивні користувацькі інтерфейси з використанням компонентного підходу, що полегшує розробку та підтримку коду.

Крім того, React має потужну екосистему та активну спільноту, що робить його ідеальною основою для розробки PWA.

Оптимізація веб-додатків за допомогою PWA полягає у використанні сучасних веб-технологій для створення додатків, які можуть працювати офлайн, завантажуватися швидко та надавати користувачам досвід, подібний до роботи з нативними додатками. Це досягається за допомогою таких технологій, як сервіс-працівники, кешування, push-сповіщення та інших [1].

PWA може бути особливо корисним для наступних сфер:

- Електронна комерція: PWA може допомогти в покращенні швидкості завантаження, забезпечити офлайн-доступ до каталогу продуктів та поліпшити загальний досвід користувача.
- Медіа та новини: PWA може забезпечити швидкий доступ до контенту, навіть при повільному або відсутньому інтернет-з'єднанні.
- Освіта: PWA може допомогти студентам і викладачам отримати доступ до навчальних матеріалів офлайн.
- Соціальні медіа: PWA може поліпшити швидкість завантаження та забезпечити кращий досвід користувача, наприклад як зробив Facebook, створивши Facebook Lite (рис. 2) [5].

Рис. 2: **Facebook PWA [5]**

Основні концепції та переваги PWA включають:

- Вартісна ефективність: PWA - це більш економічне рішення для бізнесу, оскільки розробники можуть легко налаштувати розширену

розробку веб-додатків за короткий час, з меншими вимогами до людських ресурсів та бюджету [1].

- Режим офлайн: PWA можуть легко кешуватися веб-браузером, що є чудовою новиною для бізнесу з офлайновими сценаріями використання [1].
- Покращена продуктивність: За даними Marketing Dive, 53% користувачів покидають сайт, якщо його завантаження триває довше 3 секунд [1].
- Безпека: PWA безпечні для користувачів, оскільки вони використовують безпечні кінцеві точки HTTPS та інші заходи безпеки. Адаптивність: PWA адаптуються до розміру екрана або орієнтації користувача та методу введення [1].

Рис. 3: **Переваги PWA**

Щодо створення PWA з використанням React, ось кроки, як це можна зробити [3]:

1. **Створення проекту PWA React:** Використовуйте Create React App (CRA) для створення нового проекту з шаблоном PWA. Наприклад: npx create-react-app my-app --template cra-template-pwa
2. **Реєстрація сервісного працівника:** CRA надає всі необхідні інструменти для роботи з сервіс-працівниками. Вам потрібно буде зареєструвати сервісного працівника у вашому файлі src/index.js, змінивши serviceWorker.unregister() на serviceWorker.register()
3. **Налаштування манифесту веб-додатка:** Манифест веб-додатку - це JSON-файл, який містить метадані про ваш веб-додаток, такі як ім’я, автор, опис тощо.

Рис. 3: Оглядовий рисунок роботи Pwa з Service Worker та Manifest

Висновок:

Оптимізація веб-додатків за допомогою PWA та використання React для їх розробки стають все більш важливими у світі веб-розробки. Ця стратегія дозволяє досягти великої кількості переваг, які включають економічну вигідність, роботу в офлайн-режимі, поліпшену продуктивність, безпеку та адаптивність до різних умов користувача.

За даними Marketing Dive, 53% користувачів покидають сайт, якщо його завантаження триває довше 3 секунд, тому важливо мати швидкий та надійний

веб-додаток [1]. PWA дозволяють досягти цієї мети та надати користувачам якісний досвід роботи з вашим додатком.

Використання React для розробки PWA надає додаткові переваги, такі як гнучкість та масштабованість, завдяки компонентному підходу. Щобільше, наявність активної спільноти та потужної екосистеми робить React ідеальним інструментом для створення високоякісних веб-додатків.

Отже, оптимізація веб-додатків за допомогою PWA та використання React - це сучасна та ефективна стратегія, яка допомагає підвищити якість продукту і задовольнення користувачів.

Список використаних джерел

1) Pros and Cons of PWAs: Advantages and Disadvantages of Progressive Web Apps?: Pros and Cons of PWAs: Advantages and Disadvantages of Progressive Web Apps? (ailoitte.com)
2) Progressive web apps: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps
3) Making a Progressive Web App: https://create-react-app.dev/docs/making-a-progressive-web-app/
4) React: https://ru.legacy.reactjs.org/
5) Facebook PWA: https://devlom.com/en/blog/facebook-launches-pwa

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Київський національний економічний університет імені Вадима Гетьмана
Ярославська Державна Вища Технічно-Економічна Школа
ім. о. Броніслава Маркевича (Польща) Академія
фінансів та бізнесу Vistula (Польща) Європейський
університет у Варшаві (Польща) Компанія Pro Insight
sp. z o.o. (Польща)
Вроцлавський університет економіки та бізнесу (Польща) Інститут
кібернетики імені В.М. Глушкова НАН України Київський
національний університет імені Тараса Шевченка
Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Харківський національний економічний університет ім.Семена Кузнеця
Національний технічний університет «ХПІ»
Харківський національний університет імені В. Н. Каразіна
Харківський національний університет радіоелектроніки
Харківський національний університет міського господарства ім. О.М. Бекетова Херсонський
національний технічний університет
Запорізький національний університет Хмельницький
національний університет Криворізький державний
педагогічний університет
Східноукраїнський національний університет ім. Володимира Даля Львівський
національний університет імені Івана Франка
Львівський національний технічний університет «Львівська політехніка»
Кременчуцький національний університет ім. М. Остроградського

СУЧАСНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА СИСТЕМИ В УПРАВЛІННІ

Збірник матеріалів

V Міжнародної науково-практичної конференції молодих вчених, аспірантів і студентів (Інтернет-конференція)

18–19 квітня 2024 р.

КИЇВ
2024

Ковтунович Д. О.

РОЗРОБКА ПРОЕКТУ ДІАГРАМ КЛАСІВ ВЕБ-ЗАСТОСУВАННЯ АВТОМАТИЗОВАНОГО ПРОЕКТУВАННЯ ПАРАМЕТРИЧНИХ ГРАФІЧНИХ ОБ'ЄКТІВ 275

Лебеденко Д. В.
РОЗРОБКА ПРОЕКТУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МОНІТОРИНГУ ТА УПРАВЛІННЯ ПРОЦЕСАМИ ОПЕРАЦІЙНОЇ СИСТЕМИ WINDOWS 277

Маринич В.Ю.
ПЕРЕВАГИ ТА НЕДОЛІКИ JAVA ENTERPRISE 280

Мормуль Є.А.

МОВА ПРОГРАМУВАННЯ JAVA: СУЧАСНИЙ СТАН ТА ПЕРСПЕКТИВИ РОЗВИТКУ 282

Мосунов Д. В.

РОЗРОБКА UML-ДІАГРАМ ПРОЕКТУ ІНФОРМАЦІЙНОЇ СИСТЕМИ ПІДТРИМКИ ДІЯЛЬНОСТІ МЕНЕДЖЕРА БАНКУ 284

Музика С. О.

АКТУАЛЬНІ ІНСТРУМЕНТИ ВЕБ-РОЗРОБКИ: NODE.JS, REACT, VUE.JS 287

Савчук О. О.
РОЗРОБКА СТРУКТУРИ ПРОГРАМНОГО МОДУЛЯ ДОСЛІДЖЕННЯ ПОШУКОВИХ АЛГОРИТМІВ НА ГРАФАХ 288

Скідан В.В, Бунда Н. В.

ОСОБЛИВОСТІ ЗАСТОСУВАННЯ RFID ТЕХНОЛОГІЇ В КОНТЕКСТІ УПРАВЛІННЯ ЗАПАСАМИ 290

Снігур А.Є., Назаренко Ю.В.

JAVA ENTERPRISE EDITION VS SPRING 292

Снігур А.Є.
АКТУАЛЬНІ ПИТАННЯ ВИКОРИСТАННЯ МОВ ПРОГРАМУВАННЯ ДЛЯ СТВОРЕННЯ ВЕБСАЙТІВ 293

Троян Ю.В.
ВАЖЛИВІСТЬ ЗРУЧНОСТІ ВИКОРИСТАННЯ ПРИ СТВОРЕННІ ВЕБ-САЙТУ 295

Устенко С.В., Безпалько В.М., Безпалько Ю.С.
ОПТИМІЗАЦІЯ УПРАВЛІННЯ ОСВІТНІМ КОНТЕНТОМ ЗА ДОПОМОГОЮ ВЕБ-ТЕХНОЛОГІЙ 297

Устенко С.В., д.е.н., професор
Безпалько В.М., магістр,
Безпалько Ю.С., магістр
Київський національний економічний
університет імені Вадима Гетьмана
stasustenko@ukr.net
spike723224@gmail.com
julia.bondar.24@gmail.com

ОПТИМІЗАЦІЯ УПРАВЛІННЯ ОСВІТНІМ КОНТЕНТОМ ЗА ДОПОМОГОЮ ВЕБ-ТЕХНОЛОГІЙ

Актуальність дослідження зумовлена необхідністю модернізації методів управління освітнім контентом у зв'язку зі зростанням вимог до гнучкості, доступності та інтерактивності навчальних матеріалів. Впровадження сучасних веб-технологій дозволяє оптимізувати процеси створення, розповсюдження та управління освітніми ресурсами, зробити їх більш адаптованими до потреб сучасних навчальних закладів та студентів.

Метою даного дослідження є аналіз можливостей використання веб-технологій для оптимізації управління освітнім контентом. Більша увага приділяється вивченню платформ та технологій, які забезпечують розробку інтерактивних та гнучких систем управління навчанням. Технологічний розвиток вимагає сучасних підходів до сучасних навчальних систем освіти. Освіта набуває масовості за рахунок збільшення онлайн курсів та можливості отримати знання у кращих провідних фахівців України та світу. З організаційно-технічної точки зору та сфер використання набули розвитку такі системи як LMS - Learning Management System (система управління навчанням), CMS - Course Management System (система управління курсами), LCMS - Learning Content Management System (система управління навчальним матеріалом), VLE - Virtual Learning Environments (система віртуального середовища навчання) та інші. Оптимізація освітнього контенту з базовою LMS дозволяє викладачам розподіляти завдання, слідкувати за прогресом та керувати курсами. Ці сайти можуть включати квізи, тести та інші інтерактивні елементи.. Гарними прикладами - є платформи такі як Google Classroom та Microsoft Education, вони надають інструменти для спільної роботи та оцінювання, спрощуючи взаємодію в освітньому процесі [1,2]. Окрім згаданих платформ, існує багато інших інструментів та сервісів для управління освітнім контентом. Для прикладу Moodle - який є широко поширеною платформою для навчання[6]. Платформа пропонує велику кількість плагінів, доповнень, які дозволяють адаптувати систему до специфічних потреб навчальних закладів. Також платформа підтримує інтеграцію з багатьма сторонніми інструментами і сервісами, що дозволяє розширити її функціональність і робить більш гнучкою для різних освітніх моделей. Якщо переходити до технічної частини, то, технологій, які дозволяють розробити такий проєкт, існує

достатньо. Для приладу візьмемо React, ASP.NET Core, та Umbraco CMS [3-5]. Вони вибрані через їхні унікальні переваги для розробки ефективних освітніх платформ, а саме тому, що:

1. **React** є JavaScript бібліотекою для створення користувацьких інтерфейсів, вона дозволяє розробникам ефективно створювати динамічні веб-додатки та значно підвищує швидкість відгуку застосунків [3].
2. **ASP.NET Core** є дуже потужним фреймворком для створення безпечних та модульних веб-додатків на серверній стороні. Переваги - це висока продуктивність, підтримка асинхронних операцій, та високий рівень безпеки [4].
3. **Umbraco CMS** використовується для управління контентом, він надає можливість закладами легко оновлювати та розповсюджувати матеріали[5].

Поєднання цих технологій, дає змогу створити єдину платформу, яка забезпечить ефективну взаємодію між усіма користувачами. Для розробки такої системи потрібне глибоке розуміння не лише технічних аспектів, але і потреб користувачів. Також важливо включити думку представників навчальних засобів, щоб забезпечити, що система відповідає їхнім потребам. Реалізація такої системи також вимагає забезпечення підтримки та обслуговування на запуску. В цілому, застосування інноваційних підходів, таких як використання LMS та зазначених інших технологій, забезпечує значні переваги. Дослідження показує важливість використання веб-технологій для оптимізації управління освітнім контентом.

Список використаних джерел:

1. Google Wiki. The Google encyclopedia. Google Classroom URL: https://google.fandom.com/wiki/Google_Classroom
2. Microsoft Education URL: https://www.microsoft.com/en-us/education
3. React documentation URL: https://react.dev/
4. What is ASP.NET Core? URL: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core
5. Umbraco CMS: Empowering your digital vision URL: https://umbraco.com/
6. Moodle URL: https://moodle.org/?lang=uk

КОПІЯ ТЕЗ ДОПОВІДІ

XIII Міжнародна науково-практична конференція

«ГЛУШКОВСЬКІ ЧИТАННЯ. СУЧАСНА КІБЕРНЕТИКА 2024»

Устенко С.В., д.е.н., професор,
Тішков Б.О., зав. кафедри, к.е.н., доцент,
Безпалько В.М., магістр,
Безпалько Ю.С., магістр,
Київський національний економічний
університет імені Вадима Гетьмана
stasustenko@ukr.net
tishcov_b@ukr.net
spike723224@gmail.com
julia.bondar.24@gmail.com

ВИКОРИСТАННЯ CRM-СИСТЕМ У ПІДВИЩЕННІ КЛІЄНТООРІЄНТОВАНОСТІ БІЗНЕСУ

У сучасному бізнес-середовищі, де конкуренція стає все більш жорсткою, клієнтоорієнтованість виступає одним із ключових факторів успіху компанії. Орієнтація на клієнта дозволяє не лише задовольнити поточні потреби споживачів, але й передбачити їхні майбутні очікування, створюючи тим самим довгострокову лояльність [1]. Використання CRM-систем (Customer Relationship Management) стає незамінним інструментом у цьому процесі, допомагаючи бізнесу ефективно керувати взаємовідносинами з клієнтами та підвищувати рівень їх задоволеності [2].

Роль CRM-систем у клієнтоорієнтованості бізнесу

1. Збір та аналіз даних про клієнтів

CRM-системи дозволяють компаніям збирати та аналізувати велику кількість даних про клієнтів, включаючи їхні вподобання, поведінку та історію покупок [3]. Це дає можливість створювати персоналізовані пропозиції та комунікації, що підвищує ефективність маркетингових зусиль [4].

2. Персоналізація взаємодії

Завдяки CRM-системам бізнес може забезпечити індивідуальний підхід до кожного клієнта. Персоналізація взаємодії сприяє підвищенню рівня задоволеності клієнтів та їхньої лояльності до бренду [5].

3. Підвищення якості обслуговування

CRM-системи надають співробітникам доступ до всієї історії взаємодії з клієнтом, що дозволяє швидко реагувати на запити та вирішувати проблеми [6]. Це покращує якість обслуговування та сприяє формуванню позитивного іміджу компанії [7].

4. Оптимізація бізнес-процесів

CRM-системи автоматизують бізнес-процеси, знижуючи помилки та підвищуючи ефективність, що дозволяє співробітникам зосередитися на розвитку клієнтських відносин [8, 9].

Переваги використання CRM-систем

- **Збільшення лояльності клієнтів**: персоналізований підхід та високий рівень обслуговування сприяють формуванню довгострокових відносин з клієнтами [10].
- **Підвищення продажів**: краще розуміння потреб клієнтів дозволяє пропонувати їм відповідні продукти та послуги, що збільшує обсяги продажів [11].
- **Покращення маркетингових стратегій**: аналіз даних про клієнтів допомагає ефективніше сегментувати ринок та розробляти цілеспрямовані маркетингові кампанії [12].

Виклики при впровадженні CRM-систем

- **Інвестиційні витрати**: впровадження CRM-систем потребує значних фінансових вкладень, як у придбання програмного забезпечення, так і в навчання персоналу [13].

- **Спротив змінам**: співробітники можуть опиратися новим технологіям та змінам у процесах [14].
- **Якість даних**: для ефективної роботи CRM-системи необхідно забезпечити точність та актуальність даних про клієнтів [15].

Стратегії успішного впровадження CRM-систем

- **Планування та підготовка**: ретельне планування процесу впровадження, включаючи визначення цілей та очікуваних результатів [16].
- **Навчання персоналу**: забезпечення належного навчання співробітників щодо використання CRM-системи [17].
- **Моніторинг та оцінка**: постійний контроль за роботою CRM-системи та оцінка її ефективності [18].

Висновки

CRM-системи підвищують клієнтоорієнтованість бізнесу, допомагаючи краще розуміти потреби клієнтів, забезпечувати якісне обслуговування та будувати довгострокові відносини. Попри виклики впровадження, їхні переваги сприяють зростанню конкурентоспроможності та прибутковості компанії [2,10].

Список використаних джерел:

1. Kotler, P., & Keller, K. L. (2016). *Marketing Management*. Pearson Education.
2. Buttle, F. (2009). *Customer Relationship Management: Concepts and Technologies*. Routledge.
3. Peppers, D., & Rogers, M. (2011). *Managing Customer Relationships: A Strategic Framework*. John Wiley & Sons.
4. Payne, A., & Frow, P. (2005). A Strategic Framework for Customer Relationship Management. *Journal of Marketing*, 69(4).
5. Chen, I. J., & Popovich, K. (2003). Understanding customer relationship management (CRM): People, process and technology. *Business Process Management Journal*, 9(5).

6. Mendoza, L. E., Marius, A., Pérez, M., & Grimán, A. C. (2007). Critical success factors for a customer relationship management strategy. *Information and Software Technology*, 49(8).
7. Reinartz, W., Krafft, M., & Hoyer, W. D. (2004). The customer relationship management process: Its measurement and impact on performance. *Journal of Marketing Research*, 41(3).
8. Rigby, D. K., & Ledingham, D. (2004). CRM done right. *Harvard Business Review*, 82(11).
9. Zablah, A. R., Bellenger, D. N., & Johnston, W. J. (2004). An evaluation of divergent perspectives on customer relationship management: Towards a common understanding of an emerging phenomenon. *Industrial Marketing Management*, 33(6).
10. Reichheld, F. F. (2003). The one number you need to grow. *Harvard Business Review*, 81(12).
11. Winer, R. S. (2001). A framework for customer relationship management. *California Management Review*, 43(4).
12. Xu, M., & Walton, J. (2005). Gaining customer knowledge through analytical CRM. *Industrial Management & Data Systems*, 105(7).
13. Goodhue, D. L., Wixom, B. H., & Watson, H. J. (2002). Realizing business benefits through CRM: Hitting the right target in the right way. *MIS Quarterly Executive*, 1(2).
14. Corner, I., & Hinton, M. (2002). Customer relationship management systems: Implementation risks and relationship dynamics. *Qualitative Market Research: An International Journal*, 5(4).
15. Bose, R. (2002). Customer relationship management: Key components for IT success. *Industrial Management & Data Systems*, 102(2).
16. Kim, J., Suh, E., & Hwang, H. (2003). A model for evaluating the effectiveness of CRM using the balanced scorecard. *Journal of Interactive Marketing*, 17(2).
17. Bull, C. (2003). Strategic issues in customer relationship management (CRM) implementation. *Business Process Management Journal*, 9(5).

18. Foss, B., & Stone, M. (2001). *Successful Customer Relationship Marketing*. Kogan Page.

КОПІЯ ТЕЗ ДОПОВІДІ

VI Міжнародна науково-практична конференція

Безпалько В.М., магістр
Безпалько Ю.С., магістр
Київський національний економічний університет імені Вадима Гетьмана
spike723224@gmail.com
julia.bondar.24@gmail.com

ВИКОРИСТАННЯ ВЕБ-АНАЛІТИКИ ДЛЯ ОПТИМІЗАЦІЇ БІЗНЕС-ПРОЦЕСІВ У МАЛИХ ПІДПРИЄМСТВАХ

Анотація. У роботі розглянуто роль веб-аналітики як ефективного інструменту для оптимізації бізнес-процесів у малих підприємствах. Акцент зроблено на перевагах застосування веб-аналітики, таких як покращення маркетингових стратегій, підвищення конверсії, покращення користувацького досвіду. Проаналізовано основні інструменти, зокрема Google Analytics, Hotjar та Matomo, а також описано методи роботи з аналітичними даними. Висвітлено головні виклики, з якими стикаються малі підприємства, серед яких брак ресурсів, недостатня експертиза, надлишок даних і складність дотримання вимог конфіденційності. Висновки підкреслюють важливість впровадження аналітики навіть у малих компаніях задля сталого розвитку й підвищення ефективності управління.

Ключові слова: веб-аналітика, малий бізнес, оптимізація, Google Analytics, Hotjar, конверсія, бізнес-процеси, дані

Вступ

Цифровізація охопила всі сфери економіки, і малі підприємства дедалі частіше звертаються до цифрових рішень для підвищення конкурентоспроможності. Одним із таких рішень є веб-аналітика — процес збору, обробки та аналізу даних про поведінку користувачів на сайті. Для малого бізнесу, який має обмежені ресурси, веб-аналітика стає критично важливою у прийнятті рішень, оптимізації витрат і покращенні ефективності процесів. У цій роботі розглядається, як саме веб-аналітика може бути інтегрована в діяльність малих підприємств, які інструменти доцільно використовувати та яких результатів можна досягти.

Метою цієї статті є дослідити можливості використання веб-аналітики для оптимізації бізнес-процесів у малих підприємствах. Зокрема, проаналізувати ключові інструменти, методи аналізу даних, вигоди та труднощі, з якими стикаються підприємці у процесі впровадження аналітики.

Переваги веб-аналітики для малого бізнесу

1. **Оптимізація маркетингу.** Веб-аналітика дозволяє визначати ефективність маркетингових каналів і кампаній. За допомогою Google Analytics можна відстежити джерела трафіку, найбільш популярні сторінки, поведінку користувачів, що дозволяє коригувати стратегії реклами та зменшувати витрати на неефективні канали.
2. **Підвищення конверсії.** Веб-аналітика дозволяє виявляти «вузькі місця» на шляху користувача до цільової дії — покупки, замовлення послуги, заповнення форми. Аналіз воронки конверсії допомагає змінити структуру сайту або контент задля досягнення кращих показників.

3. **Покращення користувацького досвіду.** Інструменти на кшталт Hotjar дають змогу бачити теплові карти кліків, запис сесій користувачів, що допомагає виявляти проблеми юзабіліті. Опитування користувачів та A/B-тестування дозволяють оцінити якість змін.

Інструменти та методи веб-аналітики

- **Google Analytics** — безкоштовний і найпопулярніший інструмент для збору кількісних даних (відвідуваність, джерела трафіку, поведінка користувачів).
- **Hotjar** — сервіс для аналізу поведінки, з акцентом на якісні дані: теплові карти, сесії, опитування.
- **Matomo (раніше Piwik)** — альтернатива з відкритим кодом, яка забезпечує контроль над даними та відповідає вимогам конфіденційності (GDPR).

Методика використання включає визначення ключових показників ефективності (KPI), регулярний аналіз звітів, експерименти (A/B-тестування) і впровадження змін для постійного покращення.

Виклики у впровадженні веб-аналітики

- **Брак ресурсів.** Обмеженість у фінансах та персоналі може ускладнити впровадження аналітики.
- **Недостатня експертиза.** Малі підприємства часто не мають достатніх знань для глибокого аналізу даних.
- **Надлишок інформації.** Без фокусування на важливому можна втратити орієнтир у великій кількості показників.
- **Регуляторні обмеження.** Необхідність відповідності законодавству щодо захисту даних (GDPR) створює додаткове навантаження.

Висновки

Веб-аналітика відкриває перед малими підприємствами значні можливості для підвищення ефективності бізнес-процесів, раціонального розподілу ресурсів і поліпшення взаємодії з клієнтами. Попри обмеження, доступність сучасних інструментів і можливість гнучкого використання даних робить веб-аналітику важливим елементом стратегії розвитку малого бізнесу.

Список використаних джерел

1. Almatrafi, A.M., & Alharbi, Z.H. (2023). The Impact of Web Analytics Tools on the Performance of Small and Medium Enterprises. *Engineering, Technology & Applied Science Research*, 13(5), 11753–11762.
2. Biyani, A. (2023). Benefits and Tips of Using Google Analytics for Your Small Business. *NiceJob Blog*. Retrieved from https://get.nicejob.com/
3. Loresco, S. (2023). 10 Ways to Improve Your Data Analytics Processes Using Hotjar. *Hotjar Blog*. Retrieved from https://www.hotjar.com/
4. RoseHosting (2025). Matomo vs Google Analytics: Choosing the Best for Your Business in 2025. Retrieved from https://www.rosehosting.com/
5. Ascend Analytics (2024). Small Business Analytics: Challenges and Considerations. Retrieved from https://www.ascendanalytics.com/
6. Rehnborg, A. (2025). How Web Analytics Works for Small Businesses (In-Depth Analysis). *CyberFolks Blog*. Retrieved from https://cyberfolks.pl/

Звіт подібності

метадані

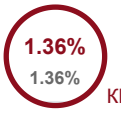
Назва організації
Kyiv National Economic University named after Vadym Hetman KNEU
Заголовок
Розроблення веб-платформи для підтримки біженців

Автор
Безпалько Юліана Сергіївна
Науковий керівник / Експерт
Мозгаллі Ольга Петрівна

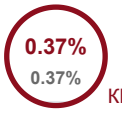
підрозділ
кафедра інформаційних систем в економіці

Обсяг знайдених подібностей

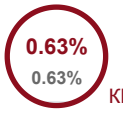
Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1



КП 2



КЦ

25
Довжина фрази для коефіцієнта подібності 2

15120
Кількість слів

121534
Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		1
Інтервали	A→	0
Мікропробіли	·	0
Білі знаки		0
Парафрази (SmartMarks)	a	11

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://ir.kneu.edu.ua/bitstreams/f2d2da61-14eb-4ba9-b82f-2663cc9ae485/download	28 0.19 %
2	https://ir.kneu.edu.ua/bitstreams/f2d2da61-14eb-4ba9-b82f-2663cc9ae485/download	28 0.19 %
3	МР_К23-1м_Рудь А.Є. [redacted] 1/8/2025 University of Customs and Finance (University of Customs and Finance)	14 0.09 %
4	https://ela.kpi.ua/bitstream/123456789/30918/1/Kasianchuk_bakalavr.pdf	12 0.08 %

5	https://ir.kneu.edu.ua/server/api/core/bitstreams/8622815f-827a-4e26-8cee-fc9dc9fd7849/content	10 0.07 %
6	Інформаційна система внутрішньої перевірки підприємства [redacted] 6/16/2020 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра інформаційних систем в економіці)	9 0.06 %
7	https://ir.kneu.edu.ua/server/api/core/bitstreams/8622815f-827a-4e26-8cee-fc9dc9fd7849/content	8 0.05 %
8	Інформаційна система внутрішньої перевірки підприємства [redacted] 6/16/2020 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра інформаційних систем в економіці)	8 0.05 %
9	https://ir.kneu.edu.ua/bitstreams/f2d2da61-14eb-4ba9-b82f-2663cc9ae485/download	7 0.05 %
10	Інформаційна система аналізу CEO показників сайту [redacted] 6/22/2020 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра інформаційних систем в економіці)	7 0.05 %

з бази даних RefBooks (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з домашньої бази даних (0.20 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Інформаційна система внутрішньої перевірки підприємства [redacted] 6/16/2020 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра інформаційних систем в економіці)	17 (2) 0.11 %
2	Інформаційна система аналізу CEO показників сайту [redacted] 6/22/2020 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра інформаційних систем в економіці)	7 (1) 0.05 %
3	Франчайзингові стратегії розвитку міжнародного бізнесу на прикладі «Multi Cook» [redacted] 4/24/2025 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра міжнародного менеджменту)	6 (1) 0.04 %

з програми обміну базами даних (0.09 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	МР_К23-1м_Рудь А.Є. [redacted] 1/8/2025 University of Customs and Finance (University of Customs and Finance)	14 (1) 0.09 %

з Інтернету (1.07 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	**https://ir.kneu.edu.ua/bitstreams/f2d2da61-14eb-4ba9-b82f-2663cc9ae485/download**	**127 (15) 0.84 %**
2	https://ir.kneu.edu.ua/server/api/core/bitstreams/8622815f-827a-4e26-8cee-fc9dc9fd7849/content	23 (3) 0.15 %
3	https://ela.kpi.ua/bitstream/123456789/30918/1/Kasianchuk_bakalavr.pdf	12 (1) 0.08 %