

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Вебплатформа координації гуманітарної допомоги»

Виконав студент групи ПЗ-22-4

Тороп Матвій Дмитрович

Керівник роботи:

ЧИЖМОТРА Олексій Володимирович

Рецензент:

ЄФРЕМОВ Юрій Миколайович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. каф. інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«27» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ТОРОП Матвій Дмитрович**

Керівник роботи: **ЧИЖМОТРЯ Олексій Володимирович**

Тема роботи: **«Вебплатформа координації гуманітарної допомоги»**,
затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №92/с**

Вихідні дані для роботи: **розробка вебплатформи для координації
гуманітарної та адресної допомоги**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів,
що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Чижмотря О.В.	1.03.2026р.	1.03.2026р.
2	Чижмотря О.В.	1.03.2026р.	1.03.2026р.
3	Чижмотря О.В.	1.03.2026р.	1.03.2026р.

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи виконана на тему «Платформа координації гуманітарної допомоги».

Випускна кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел із 43 найменувань, 5 додатків (17 сторінок), 39 таблиць, та 39 ілюстрацій по тексту. Повний обсяг роботи становить 109 сторінок, з них 92 сторінок основного тексту.

Метою роботи є проєктування та розробка вебплатформи для координації адресної допомоги між користувачами. У роботі досліджено предметну область та проаналізовано існуючі рішення у сфері організації гуманітарної підтримки. На основі проведеного аналізу сформовано вимоги до системи, спроєктовано її архітектуру та виконано повноцінну програмну реалізацію платформи. Результатом роботи є готовий програмний продукт, що забезпечує прозору та структуровану взаємодію між учасниками процесу надання допомоги.

КЛЮЧОВІ СЛОВА: АДРЕСНА ДОПОМОГА, КООРДИНАЦІЯ ВОЛОНТЕРІВ, ВЕБ-ПЛАТФОРМА, ЖИТТЄВИЙ ЦИКЛ ЗАЯВКИ, ПРОЗОРИСТЬ, РЕПУТАЦІЙНА СИСТЕМА.

					ІПЗ.КР.Б – 121 – 26 – ПЗ				
Змн.	Арк.	№ докум.	Підпис	Дата					
Розроб.		Торон М.Д.			Вебплатформа координації гуманітарної допомоги	Літ.		Арк.	Аркушів
Керівник		Чижмостря О.В.						3	109
Рецензент		Єфремов Ю.М.			Пояснювальна записка	Житомирська політехніка, група ІПЗ-22-4			
Зав. каф.		Вакалюк Т.А.							

ABSTRACT

The explanatory note to the qualification work is devoted to the topic "Platform for Coordinating Targeted Assistance".

The final thesis consists of an introduction, three chapters, conclusions, a bibliography of 43 references, 5 appendices (17 pages), 39 tables, and 39 illustrations throughout the text. The total length of the thesis is 109 pages, of which 92 pages are main text.

The goal of the work is to design and develop the web platform for coordinating targeted assistance between users. The work examines the subject area and analyzes existing solutions in the field of humanitarian aid organization. Based on the analysis, system requirements were defined, the architecture was designed, and a full software implementation of the platform was carried out. The result of the work is a complete software product that ensures transparent and structured interaction between participants in the assistance process.

KEYWORDS: TARGETED ASSISTANCE, VOLUNTEER COORDINATION, WEB PLATFORM, TRANSPARENCY, REPUTATION SYSTEM.

		Торон М.Д.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	7
ВСТУП	8
РОЗДІЛ 1. АНАЛІЗ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ...	10
1.1. Аналіз аналогів програмного продукту	10
1.2. Постановка задачі	15
1.3. Вибір архітектури системи.....	16
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	19
Висновки до першого розділу	22
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБПЛАТФОРМИ КООРДИНАЦІЇ ГУМАНІТАРНОЇ ДОПОМОГИ	23
2.1. Визначення варіантів використання та об'єктної структури системи	23
2.2. Розробка бази даних системи.....	35
2.3. Проєктування та реалізація алгоритмів системи	51
2.4. Реалізація функціоналу.....	58
Висновки до другого розділу.....	65
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ КООРДИНАЦІЇ ГУМАНІТАРНОЇ ДОПОМОГИ	66
3.1. Реалізація функціоналу.....	66
3.2. Структура інтерфейсу платформи.....	69
3.3. Тестування платформи	83
Висновки до третього розділу	87
ВИСНОВКИ.....	88
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	89
ДОДАТКИ.....	93
Додаток А	94
Додаток Б	101

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		5

Додаток В.....	102
Додаток Г.....	103
Додаток Г.....	104

		Торон М.Д.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface (інтерфейс програмування застосунків);

БД – база даних;

ER – Entity-Relationship (сутність-зв'язок);

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту);

MVP – Minimum Viable Product (мінімально життєздатний продукт);

P2P – Peer-to-Peer (однорангова взаємодія);

ПЗ – програмне забезпечення;

SPA – Single Page Application (односторінковий застосунок);

UI – User Interface (інтерфейс користувача);

UML – Unified Modeling Language (уніфікована мова моделювання);

URL – Uniform Resource Locator (уніфікований локатор ресурсів);

UX – User Experience (користувацький досвід);

ПЗ – Програмне забезпечення

		Торон М.Д.			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми зумовлена відсутністю спеціалізованих інструментів для організації прозорої адресної допомоги між користувачами. В умовах воєнного стану інформаційний простір перенасичений неструктурованими запитами, що розповсюджуються через соціальні мережі, месенджери та дошки оголошень. Після публікації запиту подальша взаємодія між сторонами відбувається у приватних повідомленнях, що унеможливорює відстеження статусу виконання, підтвердження отримання допомоги та формування достовірної історії взаємодій. Результатом є низький рівень довіри між учасниками та неможливість оцінити ефективність наданої підтримки, що визначає актуальність розробки платформи «Ethy».

Метою роботи є розробка вебплатформи «Ethy», яка забезпечує керований і прозорий процес координації адресної допомоги між користувачами шляхом структуризації життєвого циклу заявки, фіксації етапів її виконання та формування репутаційного профілю учасників.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати існуючі платформи та визначити відсутні в них механізми прозорості й відстеження процесу виконання запитів;
- сформулювати функціональні вимоги до системи та спроектувати модель взаємодії між учасниками;
- спроектувати архітектуру платформи та структуру даних;
- реалізувати основні модулі системи, що забезпечують повний життєвий цикл заявки на допомогу.

Об'єктом дослідження є процеси організації та координації адресної гуманітарної і матеріальної допомоги між користувачами в умовах неструктурованого інформаційного середовища.

Предметом дослідження є методи структурування взаємодії між учасниками процесу надання допомоги, механізми фіксації етапів виконання заявок та формування репутації користувачів у межах платформи «Ethy».

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				8
Змн.	Арк.	№ докум.	Підпис	Дата		

Методи дослідження. У роботі використано методи системного аналізу для дослідження предметної області та існуючих аналогів, об'єктно-орієнтованого проєктування при побудові доменної моделі системи, структурного моделювання для розробки діаграм UML, а також методи вебпрограмування та проєктування реляційних баз даних при реалізації платформи.

Практичне значення отриманих результатів. Розроблена платформа «Ethy» є готовим до розгортання вебзастосунком, що може застосовуватися людьми, котрі потребують допомоги, або бажають надати її, для організації прозорої адресної допомоги. Реалізований програмний продукт забезпечує повний цикл взаємодії між учасниками – від публікації заявки до підтвердженого завершення – та може бути розширений додатковими модулями відповідно до потреб конкретної організації.

Апробація. За темою кваліфікаційної роботи було опубліковано 2 тези конференцій [1, 2].

		Тороп М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				9
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1. Аналіз аналогів програмного продукту

У зв'язку з потребою в ефективних інструментах координації адресної допомоги між звичайними користувачами було виявлено кілька цифрових платформ, що частково охоплюють функціонал планованої системи «Ethy». Серед них – вітчизняна дошка оголошень, міжнародна мережа безкоштовного обміну речами, платформа координації волонтерів після стихійних лих та агрегатор мереж взаємодопомоги. Жодна з них, однак, не є прямим конкурентом, оскільки кожна орієнтована на вузький або суміжний сценарій використання.

OLX (розділ «Віддам безкоштовно»)

Найпопулярніший в Україні сервіс оголошень із широкою аудиторією по всій країні (рис. 1.1). Серед численних розділів платформи наявний окремий розділ «Віддам безкоштовно», де користувачі можуть публікувати оголошення про речі, які готові передати іншим без будь-якої оплати. OLX На відміну від колишнього розділу «Допомога», поточна модель орієнтована виключно на пропозицію – розмістити запит на отримання допомоги в структурованому вигляді неможливо.

Офіційний сайт: <https://www.olx.ua/uk/otdam-darom/>

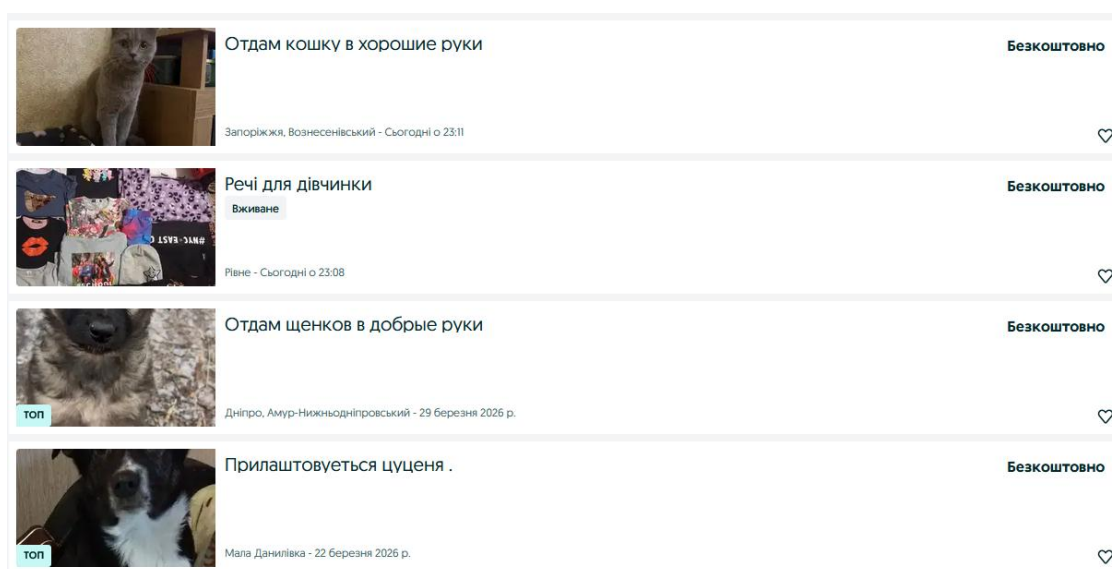


Рис. 1.1. Розділ «Віддам безкоштовно» на сайті OLX

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

Платформа має велику активну аудиторію по всій Україні, звичний інтерфейс, зрозумілий більшості користувачів, підтримує додавання фото, детального опису та геолокацію за містом, а наявність мобільного застосунку додатково розширює доступність сервісу.

Водночас розділ орієнтований виключно на пропозицію речей – потребує не може розмістити структурований запит на отримання допомоги, а більшість оголошень стосується домашніх тварин, яким шукають нового господаря, тоді як предметів інших категорій значно менше. Після публікації вся взаємодія між сторонами переходить у приватний чат і стає невидимою для системи: відсутній будь-який життєвий цикл заявки, немає підтвердження факту передачі речі, а рейтинг продавця не відображає досвід саме у сфері безкоштовної передачі.

The Freecycle Network

Некомерційна глобальна мережа, що об'єднує людей, які безкоштовно віддають непотрібні речі тим, хто їх потребує (рис. 1.2). Мережа забезпечує всесвітній онлайн-реєстр, організовуючи локальні групи та форуми, де фізичні особи та некомерційні організації можуть пропонувати або запитувати безкоштовні речі для повторного використання. Станом на сьогодні Freecycle налічує понад 5000 місцевих груп та понад 9 мільйонів учасників у понад 110 країнах світу.

Офіційний сайт: <https://www.freecycle.org>

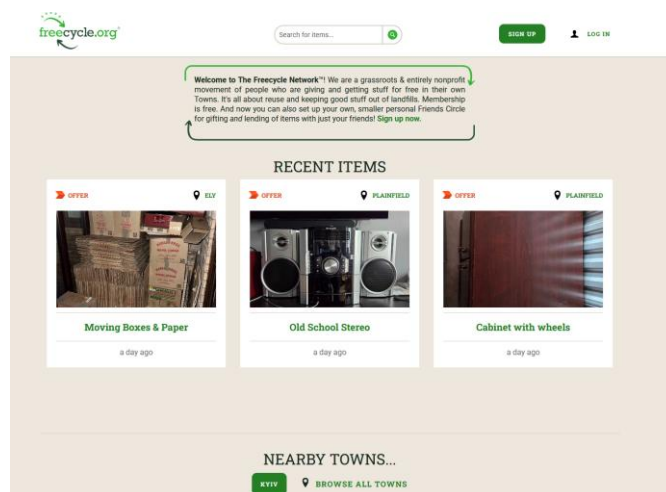


Рис. 1.2. Інтерфейс сайту Freecycle

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

Мережа має глобальне охоплення з локальною прив'язкою до міст і районів, просту модель взаємодії, повністю безкоштовну участь для всіх членів, а орієнтація на фізичні речі робить її концептуально суміжною з поняттям матеріальної допомоги.

Проте весь процес координації відбувається через електронну пошту або приватні повідомлення поза будь-якою структурованою системою: після того як зацікавлена сторона забирає предмет, відправник лише надсилає повідомлення з позначкою «TAKEN», щоб інші учасники знали про недоступність речі. Платформа не має статусів виконання, підтвердження отримання, публічної історії взаємодії та репутаційної системи для учасників.

Crisis Cleanup

Відкрита платформа для координації волонтерської допомоги постраждалим від стихійних лих (рис. 1.3). Crisis Cleanup є колаборативною платформою управління заявками на допомогу після катастроф, що покращує координацію, зменшує дублювання зусиль та підвищує ефективність роботи волонтерів. Платформа є онлайн-системою управління заявками для відстеження запитів на допомогу та їх вирішення.

Офіційний сайт: <https://www.crisiscleanup.org>

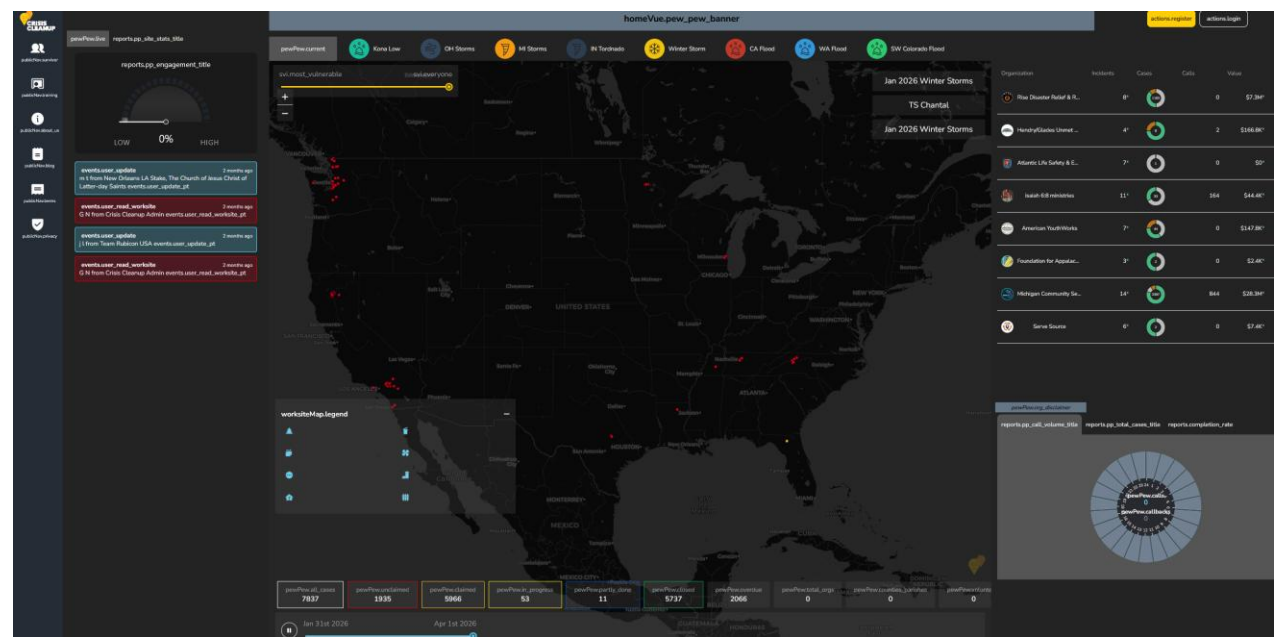


Рис. 1.3. Інтерфейс сайту CrisisCleanup

Платформа вирізняється наявністю структурованих статусів заявок, механізмом запобігання дублюванню запитів від однієї особи до різних організацій, відкритим вихідним кодом та відстеженням охоплення території волонтерами в режимі реального часу.

Однак сервіс орієнтований виключно на координацію між організаціями, а не на індивідуальних користувачів – звичайна людина не може самостійно зареєструватися та відгукнутися на запит. Платформа обмежена сферою реагування на стихійні лиха, працює переважно на території США, має складний процес верифікації для організацій, відсутню репутаційну систему, а інтерфейс місцями відображає назви полів і кнопок некоректно.

Mutual Aid Hub

Американська платформа-агрегатор, створена для збору та відображення на карті мереж взаємодопомоги, щоб допомогти людям знайти групи поблизу та з'єднати їх між собою (рис. 1.4). Інтерактивна карта дозволяє користувачам знаходити мережі взаємодопомоги та інші проєкти підтримки громади, зв'язуватися з групами, пропонувати ресурси та подавати запити на допомогу.

Офіційний сайт: <https://www.mutualaidhub.org>

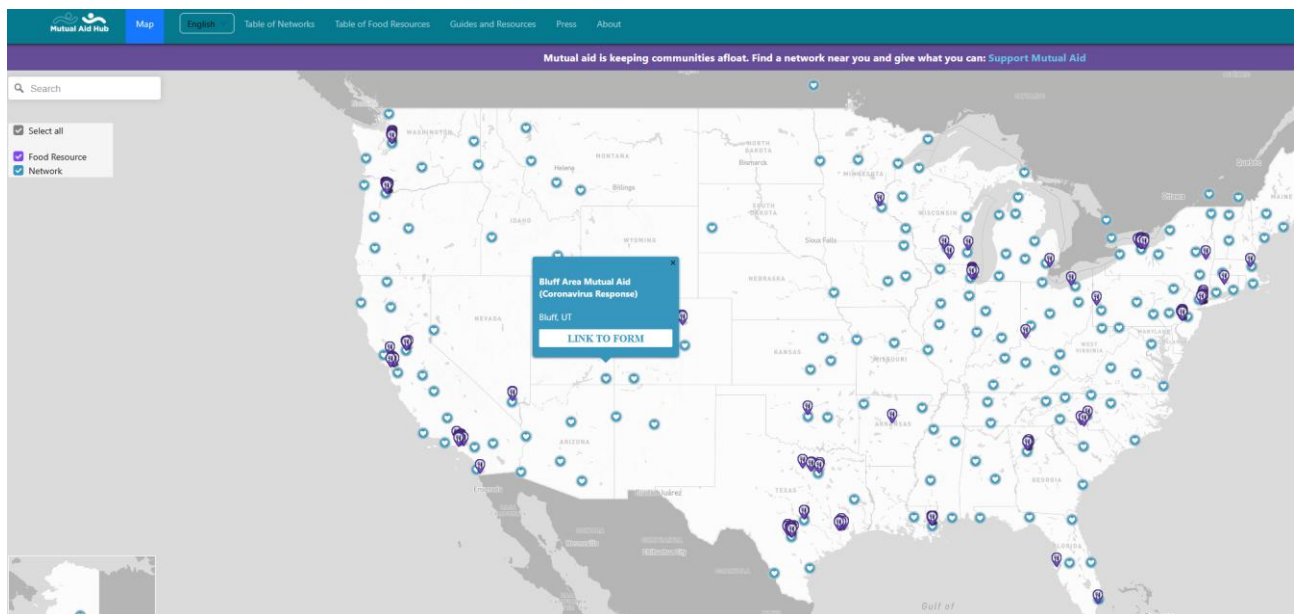


Рис. 1.4. Інтерфейс сайту Mutual Aid Hub

Концептуально це найближча до «Ethy» модель: платформа з'єднує людей, яким потрібна допомога, з тими, хто готовий її надати в межах громади, має

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				13
Змн.	Арк.	№ докум.	Підпис	Дата		

зручну карту для пошуку груп за геолокацією та охоплює різні типи потреб без прив'язки до конкретного виду кризи.

Разом з тим Mutual Aid Hub є лише агрегатором груп, а не самостійним середовищем для Р2Р-взаємодії. Платформа географічно обмежена переважно США та Канадою, позбавлена структурованого життєвого циклу заявки, публічної історії виконання, репутаційної системи та механізму безпосереднього призначення виконавця для конкретного запиту.

Таким чином, проаналізовані рішення охоплюють лише окремі аспекти координації допомоги, однак жодне з них не забезпечує повного циклу взаємодії між двома конкретними користувачами з фіксацією етапів виконання та формуванням публічної репутації учасників (табл. 1.1). Це підтверджує доцільність розробки платформи «Ethy».

Таблиця 1.1.

Порівняння аналогів відносно проєкту «Ethy»

Характеристика	OLX	Freecycle	Crisis Clean up	Mutual Aid Hub	Ethy
Запит на допомогу від фізичної особи	+/-	+/-	-	+	+
Відгук на запит	+	+	+	-	+
Структурований статус виконання	-	-	+	-	+
Публічна історія виконання заявки	-	-	-	-	+
Підтвердження отримання допомоги	-	+/-	-	-	+
Репутаційна система учасників	+/-	-	-	-	+
Орієнтація на матеріальну допомогу	+	+	-	+	+
Адаптація для українського регіону	+	+/-	-	-	+

1.2. Постановка задачі

У сучасному інформаційному суспільстві соціальні мережі та месенджери є основними інструментами комунікації, що дозволяють оперативно поширювати інформацію про потреби та події. Водночас стрімке зростання обсягів неструктурованих даних у цих каналах суттєво ускладнює фільтрування та своєчасне реагування на важливі повідомлення. В умовах воєнного стану ця проблема загострюється: інформаційний простір перенасичений запитами різного характеру, через що критично важливі звернення про допомогу губляться серед другорядного контенту [3].

Децентралізований обмін інформацією через загальні платформи створює принципову проблему координації: після публікації запиту вся подальша взаємодія між учасниками переходить у приватні повідомлення, де вона стає невидимою для системи. Це унеможливорює відстеження статусу виконання запиту, підтвердження факту передачі допомоги та формування достовірної історії взаємодій. Особливо гостро це стосується матеріальної допомоги, коли передача речей здійснюється через поштові служби або особисті домовленості, проте відповідні логістичні дані не інтегровані в жодну інформаційну систему. Результатом є низький рівень довіри між учасниками, дублювання запитів та неможливість оцінити реальну ефективність наданої підтримки [4].

Для вирішення окреслених проблем доцільно розробити спеціалізовану вебплатформу, яка перетворює хаотичну взаємодію між користувачами на керований і прозорий процес [5]. На відміну від соціальних мереж та дошок оголошень, така система має розглядати запит на допомогу не як статичне оголошення, а як об'єкт із чітко визначеним життєвим циклом, кожен етап якого фіксується та підтверджується учасниками взаємодії.

Саме тому розробляється платформа координації адресної допомоги «Ethy». Вона забезпечує централізоване середовище, в якому користувачі можуть формувати структуровані запити на допомогу, а інші учасники – переглядати їх, відгукуватись та брати участь у процесі виконання. Ключовою особливістю платформи є публічна історія кожної заявки, що формується через

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

систему подій, які ініціюються одним учасником та підтверджуються іншим. Це забезпечує достовірність інформації про перебіг виконання запиту та мінімізує можливість одностороннього внесення недостовірних даних. Додатково система накопичує статистику діяльності користувачів та формує їх репутаційний профіль, що підвищує рівень довіри між учасниками.

Першочерговим завданням системи, як вебплатформи це система облікових записів. Управління профілями та доступом користувачів передбачає реєстрацію та авторизацію учасників платформи, реалізацію рольової моделі з трьома рівнями доступу – «Користувач», «Волонтер» та «Адміністратор», а також можливість перегляду та редагування власного профілю.

Також важливою частиною є наповнення основним матеріалом. Координація запитів на допомогу є центральним функціональним блоком системи та включає створення заявок із зазначенням опису потреби, геолокації та фотоматеріалів, відображення й пошук активних запитів із фільтрацією, підтримку структурованого життєвого циклу заявки зі статусами «активний», «у процесі виконання», «виконаний» та «закритий», а також узгодження етапів виконання між учасниками зі збереженням публічної історії заявки.

Взаємодія між учасниками забезпечується через механізм відгуку на запит, після якого власник заявки має можливість призначити конкретного виконавця. Між сторонами передбачено обмін повідомленнями, а система сповіщень інформує учасників про зміни статусів заявки та нові відгуки.

Адміністрування та модерація платформи охоплюють перегляд усіх активних запитів адміністратором з можливістю їх модерації у разі порушення правил, блокування користувачів та управління їхніми ролями в системі.

1.3. Вибір архітектури системи

При проєктуванні вебплатформи «Ethy» ключовими вимогами до архітектури були масштабованість, можливість незалежного тестування компонентів та чітке розмежування відповідальності між модулями системи.

На початковому етапі проєктування було розглянуто два основні підходи до організації серверної частини: моноліт та мікросервіси. Мікросервісна

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				16
Змн.	Арк.	№ докум.	Підпис	Дата		

архітектура передбачає розбиття системи на окремі незалежно розгорнуті сервіси, кожен з яких відповідає за свою функціональну область [6]. Попри очевидні переваги у масштабуванні та незалежному розгортанні окремих модулів, мікросервісний підхід несе значні операційні витрати: необхідність оркестрації контейнерів, налаштування міжсервісної комунікації, розподілені транзакції та суттєво ускладнений процес налагодження. Для проєкту на стадії MVP з командою одного розробника такі витрати є невиправданими.

Натомість обрано модульний моноліт – підхід, який поєднує простоту розгортання монолітної архітектури зі структурованістю, характерною для мікросервісів [7]. Система фізично є єдиним застосунком, однак внутрішньо розділена на чітко відокремлені модулі з визначеними межами відповідальності. Це дає платформі «Ethy» такі конкретні переваги:

- простота розгортання та налагодження на відміну від мікросервісів;
- відсутність мережових затримок між компонентами системи;
- можливість у майбутньому виділити окремі модулі в мікросервіси без кардинальної переробки коду, оскільки межі між ними вже визначені.

В рамках модульного моноліту для організації внутрішньої структури кожного модуля обрано архітектурний підхід Clean Architecture («Чиста архітектура»), запропонований Робертом Мартіном [8]. Серед альтернатив розглядалися традиційна N-Tier (багаторівнева) архітектура та підхід MVC. N-Tier передбачає просте розшарування на рівні представлення, бізнес-логіки та даних, однак на практиці часто призводить до щільного зв'язування між шарами, коли зміна бази даних або фреймворку вимагає переробки бізнес-логіки. MVC добре підходить для простих CRUD-застосунків, але погано масштабується зі зростанням складності предметної області.

Основна ідея Clean Architecture полягає в інверсії залежностей: усі зовнішні компоненти системи – фреймворки, бази даних, інтерфейси користувача – залежать від внутрішнього доменного ядра, але не навпаки [9]. Для платформи «Ethy» це означає конкретні практичні переваги:

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				17
Змн.	Арк.	№ докум.	Підпис	Дата		

- бізнес-логіка координації заявок та їх життєвий цикл залишаються незмінними незалежно від того, яку базу даних чи API-технологію використовує система;
- заміна або оновлення будь-якого зовнішнього компонента – наприклад, перехід з одного ORM на інший – не зачіпає доменну логіку.

Архітектура платформи складається з чотирьох основних шарів.

Domain Layer (Доменний шар) є ядром системи та не має жодних зовнішніх залежностей. Тут зосереджені основні сутності платформи: користувач, заявка на допомогу, відгук на заявку, подія в журналі заявки. Також на цьому рівні визначено Value Objects – геолокація запиту, статус заявки – та доменні винятки для обробки порушень бізнес-правил.

Application Layer (Прикладний шар) відповідає за оркестрацію сценаріїв використання системи. На цьому рівні реалізовано патерн CQRS (Command Query Responsibility Segregation), який передбачає розділення операцій читання (Queries) та модифікації даних (Commands) [10]. Такий підхід дозволяє оптимізувати кожен тип операцій незалежно: запити на читання спрощуються та можуть кешуватись, тоді як команди проходять повну валідацію бізнес-правил. Взаємодія між компонентами здійснюється через патерн Mediator, що усуває пряму залежність між обробниками команд і контролерами. На цьому шарі також визначено інтерфейси репозиторіїв та сервісів, реалізація яких винесена в інфраструктурний шар.

Infrastructure Layer (Інфраструктурний шар) містить технічні деталі взаємодії з зовнішніми ресурсами. Тут розміщено реалізації репозиторіїв для роботи з базою даних Microsoft SQL Server через мікро-ORM Dapper у поєднанні з ADO.NET. На відміну від «важких» ORM на кшталт Entity Framework, Dapper виконує прямі SQL-запити з мінімальними накладними витратами, що особливо важливо для операцій читання на стороні Query в межах CQRS [11]. Також тут реалізовано сервіси хешування паролів та керування з'єднаннями з базою даних.

Presentation Layer (Шар представлення) є точкою входу в систему для зовнішніх клієнтів. Архітектурно цей шар повністю відокремлений від бізнес-

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				18
Змн.	Арк.	№ докум.	Підпис	Дата		

логіки – він лише приймає запити, передає їх у прикладний шар через Mediator та повертає відформатовану відповідь. Клієнтська частина реалізована як SPA (Single Page Application), що забезпечує безперервний користувацький досвід без повного перезавантаження сторінок. Для забезпечення функціоналу реального часу – сповіщень про зміну статусів заявок та обміну повідомленнями – передбачено окремий канал двостороннього зв'язку між клієнтом і сервером.

Таким чином, поєднання модульного моноліту, Clean Architecture та патерну CQRS забезпечує оптимальний баланс між простотою розробки на поточному етапі та можливістю масштабування системи в майбутньому без кардинальної переробки існуючого коду.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Вибір технологічного стеку для платформи координації гуманітарної допомоги «Ethy» обумовлений архітектурними рішеннями, прийнятими в попередньому пункті, а також вимогами до надійності, продуктивності та можливості підтримки системи в довгостроковій перспективі.

Для реалізації серверної частини обрано фреймворк **ASP.NET Core** з мовою програмування C#. Платформа забезпечує асинхронну обробку запитів та оптимізований конвеєр HTTP, що дозволяє ефективно обслуговувати значну кількість одночасних з'єднань – це особливо важливо для платформи з функціоналом реального часу [12]. Вбудовані механізми автентифікації, авторизації та захисту від поширених вебзагроз підвищують безпеку системи, що є критичним для платформи, яка працює з персональними даними користувачів та їхніми запитами на допомогу.

Порівняно з альтернативою у вигляді Node.js, ASP.NET Core забезпечує сувору типізацію мови C#, що знижує ймовірність помилок при роботі зі складною доменною логікою – зокрема при реалізації життєвого циклу заявки та системи узгодження подій між учасниками. Крім того, ASP.NET Core нативно інтегрується з бібліотекою MediatR, яка реалізує паттерн Mediator в рамках

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				19
Змн.	Арк.	№ докум.	Підпис	Дата		

CQRS, та із SignalR для двостороннього зв'язку в реальному часі, що робить його природним вибором для обраної архітектури [13].

Для зберігання даних використовується реляційна СУБД **Microsoft SQL Server** (MSSQL). Архітектура платформи передбачає роботу з багатьма взаємопов'язаними сутностями: користувачі, заявки, відгуки на заявки, події в журналі, повідомлення – що вимагає чітко визначених зв'язків між таблицями та підтримки транзакційності. SQL Server забезпечує ACID-транзакції, розвинені механізми індексації та цілісність даних, що є принципово важливим для достовірності публічної історії заявок – ключової особливості платформи [14].

Порівняно з MySQL, SQL Server забезпечує глибшу інтеграцію з екосистемою .NET та інструментами розробки, що спрощує налагодження та підтримку серверної частини. SQLite не розглядалась як варіант, оскільки вона орієнтована на локальні або вбудовані застосунки і не оптимізована для багатокористувацьких серверних систем з одночасними запитами. PostgreSQL є сильною альтернативою, однак перевага SQL Server у даному контексті визначається його нативною інтеграцією з інструментарієм .NET та зрілими засобами адміністрування.

Для реалізації шару доступу до даних обрано мікро-ORM **Dapper** у поєднанні з ADO.NET замість більш популярного Entity Framework. Dapper виконує прямі SQL-запити з мінімальними накладними витратами, що забезпечує максимальну продуктивність операцій читання – особливо важливо для відображення списків заявок та їх публічної історії подій. Entity Framework, попри зручність у простих CRUD-сценаріях, генерує складні SQL-запити, які важко контролювати та оптимізувати, тоді як Dapper надає повний контроль над запитами до бази даних, що відповідає принципам обраного підходу CQRS.

Для організації взаємодії між клієнтською та серверною частинами обрано **GraphQL** замість традиційного REST API. Платформа «Ethy» має сторінки з комплексними даними – наприклад, сторінка заявки одночасно відображає деталі запиту, список відгуків, публічну історію подій та інформацію про учасників. У REST-підході це потребувало б кількох окремих запитів до різних

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				20
Змн.	Арк.	№ докум.	Підпис	Дата		

ендпоінтів або створення специфічних ендпоінтів під кожен екран, що ускладнює підтримку API. GraphQL вирішує цю проблему, дозволяючи клієнту отримати всі необхідні дані одним запитом із точним переліком полів, що зменшує навантаження на мережу та підвищує швидкодію [15].

Для забезпечення двостороннього зв'язку між клієнтом і сервером у режимі реального часу використовується бібліотека **SignalR**. Вона забезпечує миттєві сповіщення про зміну статусів заявок, нові відгуки та повідомлення між учасниками взаємодії – без необхідності постійного опитування сервера клієнтом (polling). SignalR автоматично обирає оптимальний транспортний протокол залежно від можливостей середовища, надаючи перевагу WebSocket як найефективнішому варіанту [16].

Інтерфейс користувача реалізується як SPA з використанням бібліотеки React. Компонентна архітектура **React** дозволяє розбити інтерфейс на незалежні модулі – картка заявки, журнал подій, список відгуків – що спрощує їх повторне використання та підтримку. Механізм Virtual DOM забезпечує ефективне оновлення лише змінених елементів сторінки, що є важливим при динамічній зміні статусів заявок та надходженні нових повідомлень у реальному часі [18]. Як альтернативи розглядались Vue.js та Angular: Vue є простішим у освоєнні, однак має менший ринок праці та вужчу екосистему; Angular надає більш жорстку структуру, яка виправдана у великих корпоративних проєктах, але надлишкова для платформи поточного масштабу. Для збірки проєкту використовується **Vite**, який забезпечує значно швидший запуск середовища розробки порівняно з застарілим Create React App.

Для управління глобальним станом застосунку використовується **Redux Toolkit** – офіційний інструментарій для роботи з Redux, що спрощує написання редюсерів та забезпечує передбачувану обробку даних. Візуалізація заявок на інтерактивній карті реалізована через **Leaflet.js** – легковагову бібліотеку з відкритим вихідним кодом, яка не потребує платних ключів API на відміну від Google Maps

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				21
Змн.	Арк.	№ докум.	Підпис	Дата		

Для забезпечення ізолюваності середовищ розробки та виробничої версії використовується платформа контейнеризації Docker. Сервіси системи описуються у файлі конфігурації docker-compose, що дозволяє розгорнути всю платформу – API, базу даних – єдиною командою на будь-якому хостингу незалежно від операційної системи [17].

Вимоги до апаратного забезпечення

З урахуванням обраного технологічного стеку та необхідності обробки з'єднань у реальному часі через SignalR сформовано мінімальні системні вимоги для серверного середовища:

- оперативна пам'ять: 8 ГБ RAM (рекомендовано 16 ГБ для стабільної роботи SQL Server та кешування);
- процесор: 2–4 ядра з тактовою частотою від 2.5 ГГц;
- дискове сховище: від 40 ГБ на SSD для забезпечення швидкого доступу до бази даних;
- пропускна здатність мережі: від 100 Мбіт/с;
- операційна система: Linux (Ubuntu/Debian) або Windows Server з підтримкою контейнеризації Docker.

Висновки до першого розділу

У першому розділі досліджено предметну область координації гуманітарної допомоги та проаналізовано проблему фрагментованості існуючих каналів комунікації, яка унеможливорює прозоре відстеження процесу виконання запитів між користувачами. Огляд аналогів показав, що жодна з платформ не забезпечує повного циклу P2P-взаємодії з фіксацією етапів виконання та формуванням репутації учасників, що підтверджує доцільність розробки платформи «Ethy». На основі проведеного аналізу обґрунтовано архітектурні рішення – модульний моноліт, Clean Architecture та CQRS – як оптимальний баланс, а також визначено технологічний стек, що включає ASP.NET Core, Microsoft SQL Server, GraphQL, SignalR та React, який відповідає вимогам до продуктивності, надійності та підтримуваності платформи.

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				22
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБПЛАТФОРМИ КООРДИНАЦІЇ ГУМАНІТАРНОЇ ДОПОМОГИ

2.1. Визначення варіантів використання та об'єктної структури системи

Проектування вебплатформи координації гуманітарної допомоги розпочинається з формалізації способів взаємодії користувачів із системою. Виявлення варіантів використання дозволяє зафіксувати функціональні межі платформи, визначити, які дії доступні кожній категорії учасників, та уникнути неоднозначностей на етапі реалізації. Отримана модель прецедентів слугує відправною точкою для проектування бази даних, алгоритмів виконання ключових сценаріїв та об'єктно-орієнтованої структури системи.

У межах платформи координації гуманітарної допомоги передбачено взаємодію двох категорій учасників. До першої категорії належить **Користувач** – будь-яка особа, що зареєструвалася в системі та може створювати власні запити, відгукуватися на запити інших, брати участь у процесі виконання допомоги, а за потреби – ініціювати процедуру верифікації для отримання статусу «Волонтер». Друга категорія – **Адміністратор** – представник персоналу платформи, що відповідає за дотримання правил спільноти, модерацію опублікованого контенту та розгляд заявок на верифікацію. Повний набір сценаріїв взаємодії для обох акторів представлено на діаграмі варіантів використання (Додаток Б). Нижче наведено детальний опис кожного з виявлених прецедентів системи у вигляді таблиць 2.1–2.19.

Таблиця 2.1

Опис прецеденту «Реєстрація/авторизація»

Назва варіанту використання	Реєстрація/авторизація
Основні актори	Користувач
Короткий опис	Користувач може створити обліковий запис на платформі, вказавши дані, або увійти до системи за допомогою запису

Таблиця 2.2

Опис прецеденту «Перегляд профілю»

Назва варіанту використання	Перегляд профілю
Основні актори	Користувач, Адміністратор
Короткий опис	Користувач може переглядати власний профіль або профіль іншого користувача, що містить основну інформацію, статистику діяльності та репутаційний показник

Таблиця 2.3

Опис прецеденту «Зміна даних профілю»

Назва варіанту використання	Зміна даних профілю
Основні актори	Користувач
Короткий опис	Користувач може редагувати особисті дані свого профілю – ім'я, контактну інформацію, фотографію, додатковий опис

Таблиця 2.4

Опис прецеденту «Створення запиту на допомогу»

Назва варіанту використання	Створення запиту на допомогу
Основні актори	Користувач
Короткий опис	Користувач може створити новий запит на отримання допомоги із зазначенням категорії, опису потреби, геолокації, фотоматеріалів та інших необхідних параметрів

Таблиця 2.5

Опис прецеденту «Редагування даних про запит»

Назва варіанту використання	Редагування даних про запит
Основні актори	Користувач
Короткий опис	Автор запиту може вносити зміни до раніше створеної заявки за умови, що вона ще не прийнята до виконання

Таблиця 2.6

Опис прецеденту «Перегляд запитів»

Назва варіанту використання	Перегляд запитів
Основні актори	Користувач
Короткий опис	Користувач може переглядати список активних та виконаних запитів на платформі з можливістю фільтрації за категорією, геолокацією та іншими критеріями, а також детально ознайомлюватися з конкретним запитом

Таблиця 2.7

Опис прецеденту «Відгукнутись на запит»

Назва варіанту використання	Відгукнутись на запит, надавши допомогу
Основні актори	Користувач
Короткий опис	Користувач, переглядаючи активний запит, може залишити відгук із пропозицією надати допомогу автору запиту

Таблиця 2.8

Опис прецеденту «Призначення виконавця запиту»

Назва варіанту використання	Призначення власником виконавця запиту
Основні актори	Користувач
Короткий опис	Автор запиту може обрати одного з користувачів, що залишили відгук, як виконавця, після чого запит переходить до наступного етапу життєвого циклу

Таблиця 2.9

Опис прецеденту «Узгодження етапів заявки»

Назва варіанту використання	Узгодження етапів заявки між учасниками заявки
Основні актори	Користувач
Короткий опис	Власник запиту та призначений виконавець можуть ініціювати події щодо перебігу виконання допомоги, які стають частиною публічної історії заявки лише після взаємного підтвердження обома сторонами

Таблиця 2.10

Опис прецеденту «Перегляд історії етапів запиту»

Назва варіанту використання	Перегляд історії етапів запиту
Основні актори	Користувач, Адміністратор
Короткий опис	Будь-який користувач платформи може переглянути публічну історію виконання конкретного запиту із відображенням усіх узгоджених між учасниками подій

Таблиця 2.11

Опис прецеденту «Листування з користувачами»

Назва варіанту використання	Листування з користувачами
Основні актори	Користувач, Адміністратор
Короткий опис	Учасники взаємодії можуть обмінюватися приватними повідомленнями в межах платформи – як у процесі виконання заявки, так і поза ним, включаючи звернення до адміністрації

Таблиця 2.12

Опис прецеденту «Залишати відгуки про заявку та користувача»

Назва варіанту використання	Залишати відгуки про заявку та користувача
Основні актори	Користувач
Короткий опис	Після завершення взаємодії учасники можуть залишити відгук з оцінкою та текстовим коментарем, що формує репутаційний профіль іншого користувача

Таблиця 2.13

Опис прецеденту «Подання заявки на статус Волонтер»

Назва варіанту використання	Подання заявки на статус «Волонтер»
Основні актори	Користувач
Короткий опис	Користувач може ініціювати процедуру верифікації для отримання статусу «Волонтер», що підвищує рівень довіри до його облікового запису

Таблиця 2.14

Опис прецеденту «Перегляд загальної статистики»

Назва варіанту використання	Перегляд загальної статистики
Основні актори	Користувач, Адміністратор
Короткий опис	Користувач може переглядати загальну статистику активності платформи – кількість виконаних запитів, найактивніших учасників, географію допомоги тощо

Таблиця 2.15

Опис прецеденту «Перегляд запитів та користувачів у панелі адміністрування»

Назва варіанту використання	Перегляд запитів та користувачів у панелі адміністрування
Основні актори	Адміністратор
Короткий опис	Адміністратор отримує повний доступ до переліку всіх запитів та облікових записів користувачів платформи для здійснення модерації

Таблиця 2.16

Опис прецеденту «Видалення запитів / блокування користувачів»

Назва варіанту використання	Видалення запитів / блокування користувачів
Основні актори	Адміністратор
Короткий опис	Адміністратор може видаляти запити та блокувати облікові записи користувачів, що порушують правила платформи

Таблиця 2.17

Опис прецеденту «Розгляд заявок на статус Волонтер»

Назва варіанту використання	Розгляд заявок на статус «Волонтер»
Основні актори	Адміністратор
Короткий опис	Адміністратор розглядає подані користувачами заявки на верифікацію, аналізує надані документи та ухвалює рішення про надання або відмову в статусі «Волонтер»

Таблиця 2.18

Опис прецеденту «Заповнення даних про запит»

Назва варіанту використання	Заповнення даних про тип допомоги, категорії, опис, термін
Основні актори	Користувач
Короткий опис	Користувач заповнює форму запиту, вказуючи тип необхідної допомоги, категорію, детальний опис потреби, термін актуальності та додаючи фотоматеріали. Цей прецедент є обов'язковою складовою процесу створення запиту

Таблиця 2.19

Опис прецеденту «Відправка особистих даних для ідентифікації»

Назва варіанту використання	Відправка необхідних особистих даних для ідентифікації
Основні актори	Користувач
Короткий опис	Користувач надсилає особисті документи (паспорт, посвідчення особи тощо), необхідні для підтвердження своєї особи в межах процедури верифікації на отримання статусу «Волонтер»

З огляду на обраний у першому розділі технологічний стек, що передбачає використання засобів об'єктно-орієнтованого програмування мовою C#, ключовим завданням при проєктуванні системи є створення об'єктно-орієнтованої моделі. Для її відображення використовується діаграма класів, яка демонструє структуру системи через зображення класів, їх атрибутів, методів та зв'язків між ними [19]. Розглянемо доменний шар вебплатформи «Ethy», побудований відповідно до принципів предметно-орієнтованого проєктування (DDD) [20], у межах якого виділено декілька агрегатів – груп пов'язаних сутностей, що утворюють межу транзакційної цілісності.

З огляду на обраний технологічний стек, що передбачає використання засобів об'єктно-орієнтованого програмування мовою C#, ключовим завданням при проєктуванні системи є створення об'єктно-орієнтованої моделі. Для її відображення використовується діаграма класів, яка демонструє структуру системи через зображення класів, їх атрибутів, методів та зв'язків між ними [21]. Розглянемо доменний шар вебплатформи «Ethy», побудований відповідно до принципів предметно-орієнтованого проєктування (DDD), у межах якого виділено кілька агрегатів – груп пов'язаних сутностей, що утворюють межу транзакційної цілісності.

Агрегат користувача (рис. 2.1). Клас User є корінною сутністю агрегату і представляє обліковий запис у системі. Містить унікальний ідентифікатор, ім'я, електронну пошту, хеш і сім'я пароля, роль у системі (перерахування UserRole: Admin, User, Volunteer), дату реєстрації, прапорці підтвердження пошти, м'якого видалення та обмеження активних заявок. До агрегату входять також допоміжні сутності: EmailVerificationToken – токен підтвердження електронної пошти з перевіркою терміну дії та методом MarkAsUsed(); VolunteerApplication – заявка на отримання статусу волонтера зі статусами Pending, Approved, Rejected та методами Approve() і Reject(); UserBlockRecord – незмінний запис про факт блокування облікового запису адміністратором.

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		30

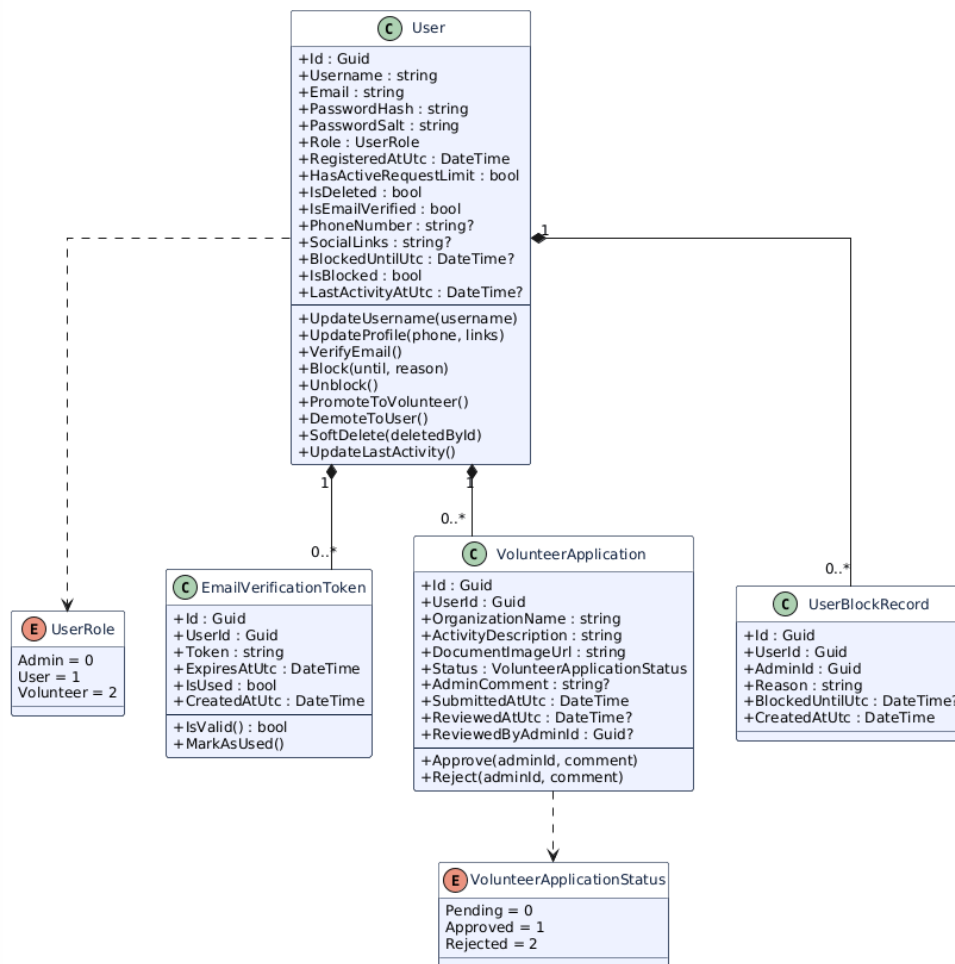


Рис. 2.1. Діаграма класів користувача доменного рівня

Агрегат заявки на допомогу (рис. 2.2). Центральним елементом системи є клас `HelpRequest`, який об'єднує заявку та всі пов'язані з нею об'єкти. До складу агрегату входять: `HelpRequestGeoPoint` – Value Object, що інкапсулює географічні координати заявки з валідацією допустимих меж (широта $-90..90$, довгота $-180..180$); `RequestImage` – Value Object, який представляє зображення, прикріплене до заявки, з порядковим номером для сортування в галереї; `HelpRequestResponse` – сутність відгуку кандидата на заявку зі статусами Pending, Accepted, Rejected, Cancelled та внутрішніми методами Accept(), Reject(), Cancel(), що викликаються виключно через кореневий клас; `HelpRequestStage` – сутність етапу виконання заявки зі статусами Proposed, Confirmed, Rejected, Deleted та методами Confirm(), Reject(), Delete(), кожен з яких містить перевірку що ініціатор не може підтвердити або відхилити власний етап; `HelpRequestEvent` – незмінний запис журналу подій із перерахуванням

HelpRequestEventType, що охоплює одинадцять типів подій: ExecutorAssigned, StatusChanged, StageProposed, StageConfirmed, StageRejected, StageDeleted, HelpRequestEdited, ExecutorResigned, ExecutorRemoved, ModerationApproved, ModerationRejected; HelpRequestReport – звіт про результат виконання заявки з текстовим коментарем та необов'язковим зображенням.

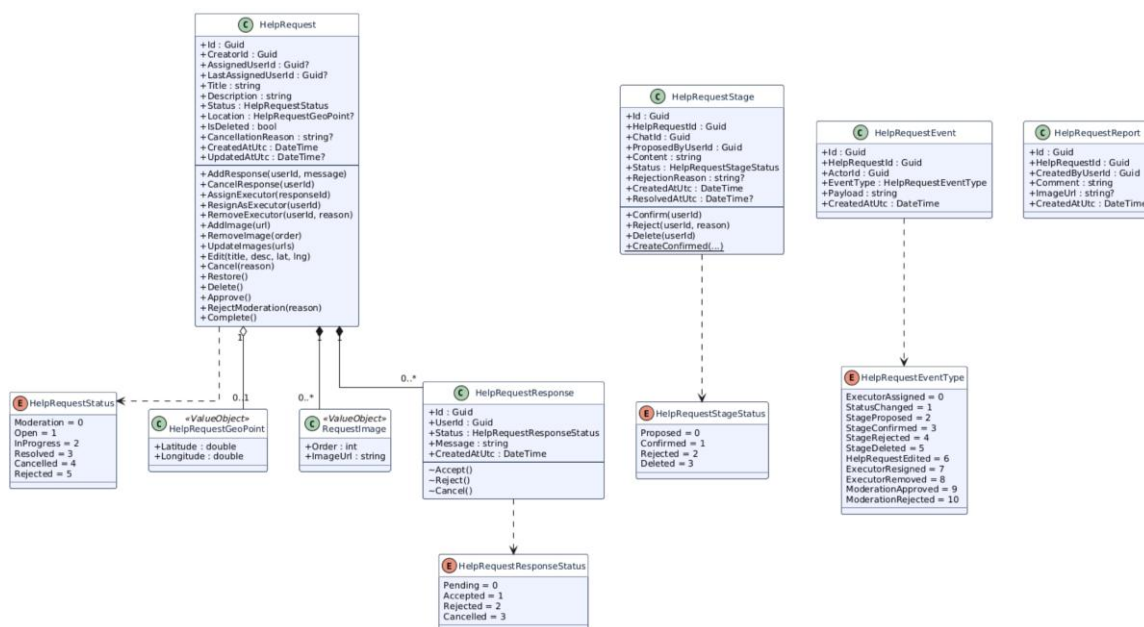


Рис. 2.2. Діаграма класів заявок доменного рівня

Агрегат чату (рис. 2.3). Клас Chat представляє приватний канал спілкування між власником заявки та призначеним виконавцем, прив'язаний до конкретної заявки. Містить ідентифікатори обох учасників, дату створення та прапорець активності. Метод Deactivate() викликається при відмові або відхиленні виконавця. Сутність ChatMessage представляє окреме повідомлення в межах чату з вбудованою валідацією вмісту: обов'язковість тексту та обмеження у 2000 символів.

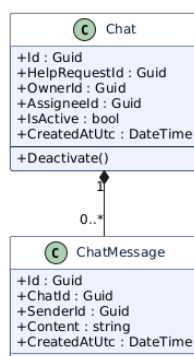


Рис. 2.3. Діаграма класів чату доменного рівня

Агрегат відгуків та скарг (рис. 2.4). Клас UserReview реалізує систему репутації – кожен відгук посилається на автора, цільового користувача та заявку, в межах якої відбулася взаємодія, і містить булеву оцінку (позитивна/негативна) та необов'язковий коментар. Клас UserComplaint представляє скаргу на конкретного користувача з текстовим обґрунтуванням.

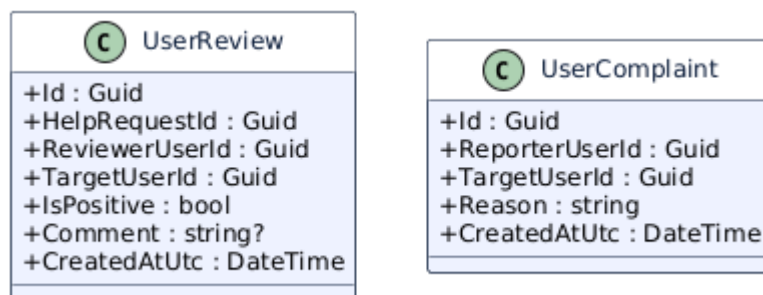


Рис. 2.4. Діаграма класів відгуків та скарг доменного рівня

Сповіщення (рис. 2.5). Клас Notification є самостійною сутністю, що зберігає сповіщення для конкретного користувача. Містить заголовок, текст, тип (перерахування NotificationType: Info, Chat, HelpRequest, Volunteer, Warning), прапорець прочитання та необов'язкові поля для навігації до пов'язаної сутності. Метод MarkAsRead() позначає сповіщення як прочитане.

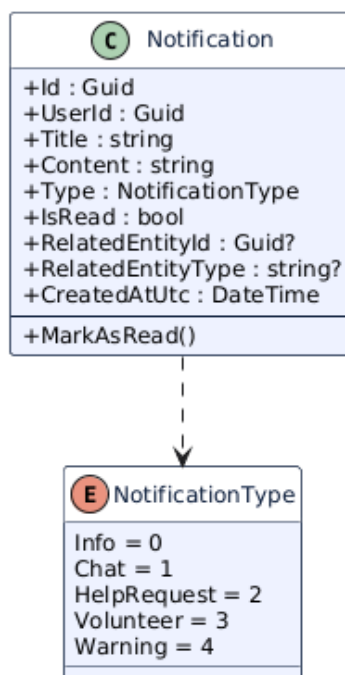


Рис. 2.5. Діаграма класів сповіщень доменного рівня

Примітиви доменного шару (рис. 2.6). Для уніфікованої обробки результатів операцій та помилок використовуються допоміжні типи: Result та його узагальнений варіант Result<T> для явного повернення успіху або помилки без використання винятків у штатних сценаріях, запис Error для опису помилки з кодом і повідомленням, а також клас DomainException для винятків, що виникають при порушенні доменних бізнес-правил.

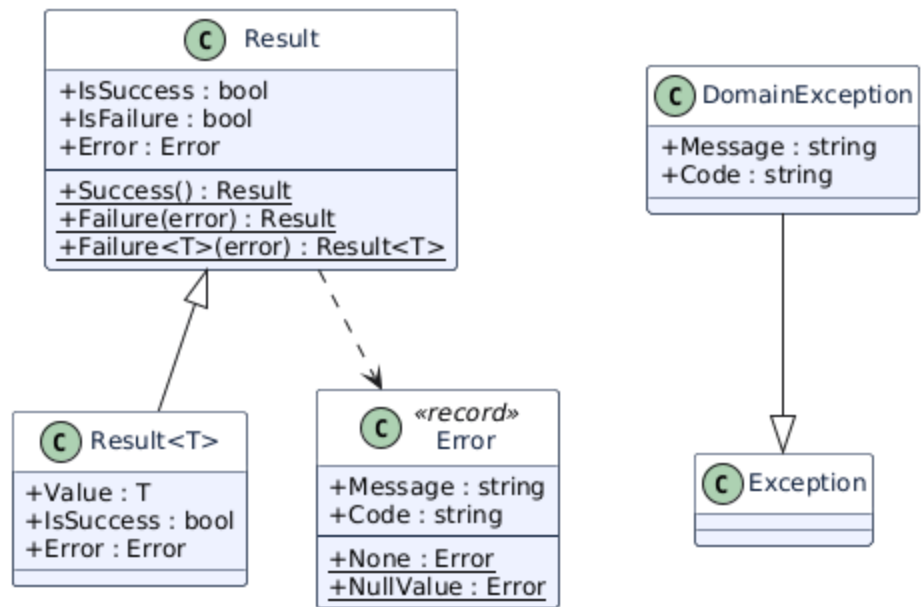


Рис. 2.6. Діаграма примітивів доменного рівня

2.2. Розробка бази даних системи

Оснoву збереження даних платформи «Ethy» складає реляційна база даних Microsoft SQL Server. Вибір реляційної моделі обумовлений чітко структурованим предметним доменом із розвинуеною мережею зв'язків між сутностями, необхідністю забезпечення цілісності даних через декларативні обмеження та зовнішні ключі, а також підтримкою транзакцій, що є критично важливим для операцій, які зачіпають кілька таблиць одночасно. Схему бази даних побудовано в результаті покрокового застосування міграцій засобами бібліотеки DbUp, що дозволяє відстежувати версію схеми у системі контролю версій і відтворювати середовище з нуля.

Повну структуру реляційної схеми відображено у вигляді ER-діаграми (Додаток В).

Нижче наведено детальний опис кожної таблиці бази даних (полів, типів, обмежень, зв'язків) у вигляді таблиць 2.20-2.36.

Таблиця 2.20

Опис таблиці “User”

Поле	Тип	Обмеження	Опис
Id	UNIQUE IDENTIFIER	PK, NOT NULL	Унікальний ідентифікатор
Username	NVARCHAR	NOT NULL	Ім'я користувача
Email	NVARCHAR	NOT NULL, UNIQUE	Електронна адреса;
PasswordHash	NVARCHAR	NOT NULL	Хеш пароля користувача
PasswordSalt	NVARCHAR	NOT NULL	Сіль для хешування пароля
Role	INT	NOT NULL	Роль у системі
RegisteredAtUtc	DATETIME2	NOT NULL, SYSUTCDATE TIME()	Дата та час реєстрації облікового запису (UTC)

HasActiveRequestLimit	BIT	NOT NULL, DEFAULT 1	Прапорець обмеження кількості активних заявок
IsDeleted	BIT	NOT NULL, DEFAULT 0	Ознака м'якого видалення облікового запису
DeletedAtUtc	DATETIME2	NULL	Дата та час м'якого видалення
DeletedById	UNIQUE IDENTIFIER	NULL	Ідентифікатор адміністратора, що видалив запис
PhoneNumber	NVARCHAR	NULL	Номер телефону користувача (необов'язковий)
SocialLinks	NVARCHAR	NULL	Посилання на профілі в соціальних мережах
IsEmailVerified	BIT	NOT NULL, DEFAULT 0	Ознака підтвердженої електронної адреси
BlockedUntilUtc	DATETIME2	NULL	Дата закінчення тимчасового блокування
BlockReason	NVARCHAR	NULL	Причина блокування облікового запису
LastVolunteerApplicationAtUtc	DATETIME2	NULL	Дата останньої поданої заявки на статус волонтера
LastActivityAtUtc	DATETIME2	NULL	Дата та час останньої активності користувача

Таблиця 2.21

Опис таблиці “HelpRequestImages”

Поле	Тип	Обмеження	Опис
Id	UNIQUEIDENTIFIER	PK, NOT NULL	Унікальний ідентифікатор
HelpRequestId	UNIQUEIDENTIFIER	NOT NULL, FK → HelpRequests(Id) ON DELETE CASCADE	Ідентифікатор заявки, до якої належить зображення
Order	INT	NOT NULL	Порядковий номер зображення
ImageUrl	NVARCHAR	NOT NULL	Шлях до файлу

Таблиця 2.22

Опис таблиці “HelpRequestResponses”

Поле	Тип	Обмеження	Опис
Id	UNIQUEIDENTIFIER	PK, NOT NULL	Унікальний ідентифікатор
HelpRequestId	UNIQUEIDENTIFIER	NOT NULL, FK → HelpRequests(Id)	Ідентифікатор заявки, на яку залишено відгук
UserId	UNIQUEIDENTIFIER	NOT NULL	Ідентифікатор користувача, що відгукнувся
Status	INT	NOT NULL	Статус відгуку
Message	NVARCHAR	NOT NULL	Текстове повідомлення кандидата
CreatedAtUtc	DATETIME2	NOT NULL	Дата та час створення відгуку (UTC)

Опис таблиці “HelpRequests”

Поле	Тип	Обмеження	Опис
Id	UNIQUE IDENTIFIER	PK, NOT NULL	Унікальний ідентифікатор
CreatorId	UNIQUEIDENTIFIER	NOT NULL, FK → Users(Id)	Ідентифікатор користувача, який створив заявку
Title	NVARCHAR	NOT NULL	Заголовок заявки
Description	NVARCHAR	NOT NULL	Детальний опис потреби
Status	INT	NOT NULL	Статус
Latitude	FLOAT	NULL	Географічна широта
Longitude	FLOAT	NULL	Географічна довгота
CreatedAtUtc	DATETIME2	NOT NULL	Дата/час створення заявки
UpdatedAtUtc	DATETIME2	NULL	Дата та час останнього оновлення
AssignedUserId	UNIQUEIDENTIFIER	NULL	Ідентифікатор призначеного виконавця
LastAssignedUserId	UNIQUE IDENTIFIER	NULL	Ідентифікатор попереднього призначеного виконавця
IsDeleted	BIT	NOT NULL, DEFAULT 0	Ознака м'якого видалення заявки
CancellationReason	NVARCHAR	NULL	Причина скасування заявки
IsHidden	BIT	NOT NULL, DEFAULT 0	Ознака приховання заявки адміністратором
ResolvedAtUtc	DATETIME2	NULL	Дата та час позначення заявки як виконаної

Таблиця 2.24

Опис таблиці “Chats”

Поле	Тип	Обмеження	Опис
Id	UNIQUEIDENTIFIER	PK, NOT NULL	Унікальний ідентифікатор
HelpRequestId	UNIQUEIDENTIFIER	NOT NULL, FK → HelpRequests(Id)	Заявка, у межах якої відкрито чат
OwnerId	UNIQUEIDENTIFIER	NOT NULL	Ідентифікатор власника заявки
AssigneeId	UNIQUEIDENTIFIER	NOT NULL	Ідентифікатор призначеного виконавця
CreatedAtUtc	DATETIME2	NOT NULL	Дата та час створення чату (UTC)
IsActive	BIT	NOT NULL, DEFAULT 1	Ознака активності чату

Таблиця 2.25

Опис таблиці “ChatMessages”

Поле	Тип	Обмеження	Опис
Id	UNIQUEIDENTIFIER	PK, NOT NULL	Унікальний ідентифікатор
ChatId	UNIQUEIDENTIFIER	NOT NULL, FK → Chats(Id)	Ідентифікатор чату, до якого належить повідомлення
SenderId	UNIQUEIDENTIFIER	NOT NULL	Ідентифікатор відправника
Content	NVARCHAR	NOT NULL	Текст повідомлення
CreatedAtUtc	DATETIME2	NOT NULL	Дата та час надсилання повідомлення (UTC)

Опис таблиці “HelpRequestStages”

Поле	Тип	Обмеження	Опис
Id	UNIQUEIDENTIFIER	PK, NOT NULL	Унікальний ідентифікатор
HelpRequestId	UNIQUEIDENTIFIER	NOT NULL, FK HelpRequests(Id)	Ідентифікатор заявки, до якої належить етап
ChatId	UNIQUEIDENTIFIER	NOT NULL, FK → Chats(Id)	Ідентифікатор чату, у межах якого запропоновано етап
ProposedBy UserId	UNIQUEIDENTIFIER	NOT NULL	Ідентифікатор користувача, що запропонував етап
Content	NVARCHAR	NOT NULL	Текстовий опис етапу виконання
Status	INT	NOT NULL	Статус
Rejection Reason	NVARCHAR	NULL	Причина відхилення етапу
CreatedAtUtc	DATETIME2	NOT NULL	Дата та час пропозиції етапу (UTC)
ResolvedAtUtc	DATETIME2	NULL	Дата та час підтвердження або відхилення етапу

Таблиця 2.27

Опис таблиці “HelpRequestEventLog”

Поле	Тип	Обмеження	Опис
Id	UNIQUEIDENTIFIER	PK, NOT NULL	Унікальний ідентифікатор
HelpRequestId	UNIQUEIDENTIFIER	NOT NULL, FK → HelpRequests(Id)	Ідентифікатор заявки, до якої відноситься подія
ActorId	UNIQUEIDENTIFIER	NOT NULL	Ідентифікатор користувача, що ініціював подію
EventType	INT	NOT NULL	Тип події
Payload	NVARCHAR	NOT NULL	JSON-рядок із додатковими даними події
CreatedAtUtc	DATETIME2	NOT NULL	Дата та час фіксації події (UTC)

Таблиця 2.28

Опис таблиці “StageTemplates”

Поле	Тип	Обмеження	Опис
Id	UNIQUE IDENTIFIER	PK, NOT NULL	Унікальний ідентифікатор шаблону
Content	NVARCHAR	NOT NULL	Текст шаблонного опису етапу
IsAutomatic	BIT	NOT NULL, DEFAULT 0	Ознака автоматично застосовуваного шаблону

Опис таблиці “HelpRequestReports”

Поле	Тип	Обмеження	Опис
Id	UNIQUEIDENTIFIER	PK, NOT NULL	Унікальний ідентифікатор
HelpRequestId	UNIQUEIDENTIFIER	NOT NULL, FK → HelpRequests(Id)	Ідентифікатор заявки, до якої прикріплено звіт
CreatedByUserId	UNIQUEIDENTIFIER	NOT NULL	Ідентифікатор користувача, що склав звіт
Comment	NVARCHAR	NOT NULL	Текстовий коментар до результатів виконання
ImageUrl	NVARCHAR	NULL	Шлях до прикріпленого зображення звіту
CreatedAtUtc	DATETIME2	NOT NULL	Дата та час створення звіту (UTC)

Опис таблиці “UserReviews”

Поле	Тип	Обмеження	Опис
Id	UNIQUEIDENTIFIER	PK, NOT NULL	Унікальний ідентифікатор
HelpRequestId	UNIQUEIDENTIFIER	NOT NULL, FK → HelpRequests(Id)	Заявка, у межах якої залишено відгук
ReviewerUserId	UNIQUEIDENTIFIER	NOT NULL, FK → Users(Id)	Ідентифікатор автора відгуку
TargetUserId	UNIQUEIDENTIFIER	NOT NULL, FK → Users(Id)	Ідентифікатор користувача, якому залишено відгук
IsPositive	BIT	NOT NULL	Тип оцінки
Comment	NVARCHAR	NULL	Необов'язковий текстовий коментар
CreatedAtUtc	DATETIME2	NOT NULL	Дата та час залишення відгуку (UTC)

Опис таблиці “UserComplaints”

Поле	Тип	Обмеження	Опис
Id	UNIQUEIDENTIFIER	PK, NOT NULL	Унікальний ідентифікатор
ReporterUserId	UNIQUEIDENTIFIER	NOT NULL, FK → Users(Id)	Ідентифікатор користувача, що подав скаргу
TargetUserId	UNIQUEIDENTIFIER	NOT NULL, FK → Users(Id)	Ідентифікатор користувача, на якого подано скаргу
Reason	NVARCHAR	NOT NULL	Текстове обґрунтування скарги
CreatedAtUtc	DATETIME2	NOT NULL	Дата та час подачі скарги (UTC)
IsResolved	BIT	NOT NULL, DEFAULT 0	Ознака розгляду скарги адміністратором
AdminComment	NVARCHAR	NULL	Коментар адміністратора за результатами розгляду

Опис таблиці “EmailVerificationTokens”

Поле	Тип	Обмеження	Опис
Id	UNIQUEIDENTIFIER	PK, NOT NULL, DEFAULT NEWID()	Унікальний ідентифікатор токена
UserId	UNIQUEIDENTIFIER	NOT NULL, FK → Users(Id)	Ідентифікатор користувача, якому видано токен
Token	NVARCHAR	NOT NULL	Рядок токена підтвердження електронної пошти
ExpiresAtUtc	DATETIME2	NOT NULL	Дата та час закінчення дії токена (UTC)
IsUsed	BIT	NOT NULL, DEFAULT 0	Ознака використаного токена
CreatedAtUtc	DATETIME2	NOT NULL, DEFAULT GETUTCDATE()	Дата та час генерації токена (UTC)

Опис таблиці “VolunteerApplications”

Поле	Тип	Обмеження	Опис
Id	UNIQUE IDENTIFIER	PK, NOT NULL, DEFAULT NEWID()	Унікальний ідентифікатор заявки на верифікацію
UserId	UNIQUE IDENTIFIER	NOT NULL, FK → Users(Id)	Ідентифікатор користувача-заявника
Organization Name	NVARCHAR	NOT NULL	Назва організації або ініціативи
Activity Description	NVARCHAR	NOT NULL	Опис волонтерської діяльності
Document ImageUrl	NVARCHAR	NULL	Шлях до завантаженого підтверджуючого документа
Status	INT	NOT NULL, DEFAULT 0	Статус розгляду
Admin Comment	NVARCHAR	NULL	Коментар адміністратора при ухваленні рішення
Submitted AtUtc	DATETIME2	NOT NULL, DEFAULT GETUTCDATE()	Дата та час подачі заявки (UTC)
Reviewed AtUtc	DATETIME2	NULL	Дата та час розгляду заявки адміністратором
ReviewedBy AdminId	UNIQUE IDENTIFIER	NULL, FK → Users(Id)	Ідентифікатор адміністратора, що розглянув заявку

Опис таблиці “UserBlockHistory”

Поле	Тип	Обмеження	Опис
Id	UNIQUEIDENTIFIER	PK, NOT NULL, DEFAULT NEWID()	Унікальний ідентифікатор
UserId	UNIQUEIDENTIFIER	NOT NULL, FK → Users(Id)	Ідентифікатор заблокованого користувача
AdminId	UNIQUEIDENTIFIER	NOT NULL, FK → Users(Id)	Ідентифікатор адміністратора, що здійснив блокування
Reason	NVARCHAR	NOT NULL	Причина блокування
BlockedUntilUtc	DATETIME2	NULL	Дата закінчення блокування (NULL – постійне блокування)
CreatedAtUtc	DATETIME2	NOT NULL, DEFAULT GETUTCDATE()	Дата та час застосування блокування (UTC)

Таблиця 2.35

Опис таблиці “Notifications”

Поле	Тип	Обмеження	Опис
Id	UNIQUE IDENTIFIER	PK, NOT NULL	Унікальний ідентифікатор
UserId	UNIQUE IDENTIFIER	NOT NULL, FK → Users(Id) ON DELETE CASCADE	Ідентифікатор отримувача сповіщення
Title	NVARCHAR	NOT NULL, FK → Users(Id)	Заголовок сповіщення
Content	NVARCHAR	NOT NULL	Текст сповіщення
Type	INT	NULL	Тип сповіщення (кодується числом)
IsRead	BIT	NOT NULL, DEFAULT GETUTCDATE()	Ознака прочитаного сповіщення
RelatedEntityId	UNIQUE IDENTIFIER	NULL	Ідентифікатор пов'язаної сутності (заявки, чату тощо)
RelatedEntity Type	NVARCHAR	NULL	Тип пов'язаної сутності у вигляді рядка
CreatedAtUtc	DATETIME2	NOT NULL	Ознака прочитаного сповіщення

Для прискорення виконання найбільш поширених запитів у базі даних визначено набір індексів. Таблиця 2.36 містить повний перелік створених індексів із зазначенням їх призначення.

Індекси бази даних платформи «Ethy»

Назва індексу	Таблиця	Поля	Призначення
IX_HelpRequests _CreatorId	HelpRequests	CreatorId	Прискорює вибірку заявок конкретного автора
IX_HelpRequestImages _RequestId	HelpRequestImages	HelpRequestId	Прискорює отримання зображень
IX_EmailVerificationTokens _Token	EmailVerificationTokens	Token	Прискорює пошук токена при підтвердженні email
IX_EmailVerificationTokens _UserId	EmailVerificationTokens	UserId	Прискорює вибірку токенів
IX_VolunteerApplications _UserId	VolunteerApplications	UserId	Прискорює перегляд заявок на верифікацію
IX_VolunteerApplications _Status	VolunteerApplications	Status	Прискорює фільтрацію заявок за статусом розгляду

IX_UserBlockHistory _UserId	UserBlockHistory	UserId	Прискорює отримання історії блокувань конкретного користувача
IX_Notifications_UserId _IsRead	Notifications	UserId, IsRead	Прискорює вибірку непрочитаних сповіщень користувача
IX_Notifications _CreatedAt	Notifications	CreatedAtUtc DESC	Прискорює сортування сповіщень за хронологією

Схема відрізняється послідовним використанням типу UNIQUEIDENTIFIER як первинних ключів замість автоінкрементних цілих чисел. Такий підхід дозволяє генерувати ідентифікатори на стороні клієнта або прикладного рівня ще до збереження запису в базі даних, що спрощує роботу з тимчасовими файлами та знижує ймовірність конфліктів при паралельних операціях вставки. Дати та часові мітки зберігаються виключно у форматі UTC (суфікс Utc у назвах полів) та з використанням типу DATETIME2, що забезпечує мікросекундну точність і сумісність із часовими поясами. М'яке видалення (soft delete) реалізовано через прапорець IsDeleted у таблицях Users та HelpRequests, що дозволяє зберігати аудиторський слід усіх дій без фізичного знищення даних.

2.3. Проєктування та реалізація алгоритмів системи

Функціональність платформи «Ethy» ґрунтується на кількох ключових алгоритмах, що реалізують основні бізнес-процеси системи. На відміну від типових CRUD-операцій, ці алгоритми охоплюють багатокрокові сценарії взаємодії між кількома акторами, супроводжуються зміною станів сутностей та формують аудиторський слід усіх значущих подій. У підрозділі розглянуто три ключові алгоритми: створення та модерація заявки, призначення виконавця, а також узгодження етапів виконання.

Процес публікації заявки на платформі є двофазним: перша фаза – заповнення та надсилання форми користувачем, друга – модерація адміністратором. Такий підхід забезпечує контроль якості контенту до його потрапляння у загальний доступ і є принциповою відмінністю платформи від дошок оголошень, де публікація відбувається миттєво.

На першому етапі користувач заповнює форму, вказуючи назву, опис, за бажанням, прикріплює зображення та координати місця надання допомог. Перед відправкою на сервер React-клієнт виконує локальну валідацію форми – перевіряє наявність обов'язкових полів та коректність введених даних. У разі виявлення помилок запит на сервер не надсилається, а користувачеві відображаються відповідні повідомлення. Лише після успішної валідації формується GraphQL-запит до сервера.

Серверна частина, отримавши запит, перевіряє дійсність JWT-токена та права користувача. Зображення завантажуються у тимчасову директорію і лише після успішного збереження основного запису в базі даних переміщуються у постійне сховище. Заявка зберігається зі статусом Moderation, що позначає її як «на опрацюванні», недоступну для публічного перегляду.

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		51

Друга фаза – модерація – відбувається на стороні адміністратора. Після збереження заявки система автоматично формує сповіщення для адміністратора. Переглянувши деталі, адміністратор ухвалює рішення: у разі схвалення статус заявки змінюється на Open і вона потрапляє у загальний список; у разі відхилення – статус встановлюється як Cancelled, і заявка залишається недоступною для інших користувачів. В обох випадках автор заявки отримує відповідне сповіщення. Повна послідовність взаємодії між клієнтом, сервером, базою даних та адміністратором відображена на діаграмі активності алгоритму створення та модерації заявок (Додаток Г).

Після публікації заявки вона стає доступною іншим користувачам, які можуть висловити готовність надати допомогу. Алгоритм призначення виконавця охоплює весь шлях від подання відгуку до встановлення робочого каналу зв'язку між учасниками.

Кандидат, переглянувши деталі заявки, надсилає відгук із вступним повідомленням. Сервер зберігає відгук зі статусом Pending та формує сповіщення власнику заявки. Власник у будь-який момент, поки заявка має статус Open, може переглянути список кандидатів та ознайомитися з їхніми повідомленнями. Вибравши відповідного кандидата, власник ініціює призначення.

Призначення виконавця є атомарною операцією, що зачіпає кілька таблиць бази даних одночасно: статус заявки змінюється на InProgress, обраному кандидату присвоюється роль виконавця, решті відгуків встановлюється статус Rejected, у таблиці Chats створюється запис із прив'язкою до заявки та обох учасників, а в журнал подій HelpRequestEventLog додається запис типу ExecutorAssigned. Атомарність цієї операції забезпечує узгодженість даних: неможлива ситуація, коли заявка перейшла у статус InProgress, але чат між учасниками не було створено, або коли кілька кандидатів одночасно мають статус Accepted. Після завершення операції виконавець отримує сповіщення про призначення, а на клієнті власника стає активним перехід до чату.

Важливо зазначити, що створення запису чату в базі даних не рівнозначне встановленню з'єднання SignalR. Запис у таблиці Chats фіксує лише факт

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		52

існування каналу зв'язку між двома учасниками та його прив'язку до конкретної заявки. Реальне підключення відбувається лише тоді, коли один із учасників відкриває модальне вікно чату – в цей момент клієнт ініціює з'єднання з хабом та підписується на відповідну групу повідомлень за ідентифікатором заявки. Такий підхід дозволяє уникнути зайвих постійних з'єднань і зменшити навантаження на сервер, оскільки SignalR-з'єднання існує лише протягом активної сесії роботи з чатом.

Алгоритм також передбачає зворотні сценарії дострокового завершення співпраці. Якщо призначений виконавець вирішує відмовитися від виконання, він ініціює відмову із зазначенням причини: заявка повертається у статус Open, поточний чат деактивується, а власник отримує відповідне сповіщення. Симетричний сценарій передбачений і для власника заявки – він може відхилити призначеного виконавця із зазначенням обґрунтування, після чого заявка так само повертається у статус Open і стає доступною для нових відгуків. В обох випадках система зберігає запис у журналі подій HelpRequestEventLog, що забезпечує повну аудиторську стежку незалежно від того, хто ініціював припинення співпраці. Повну послідовність описаного алгоритму зображено на рис. 2.7.

		Торон М.Д.			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		53

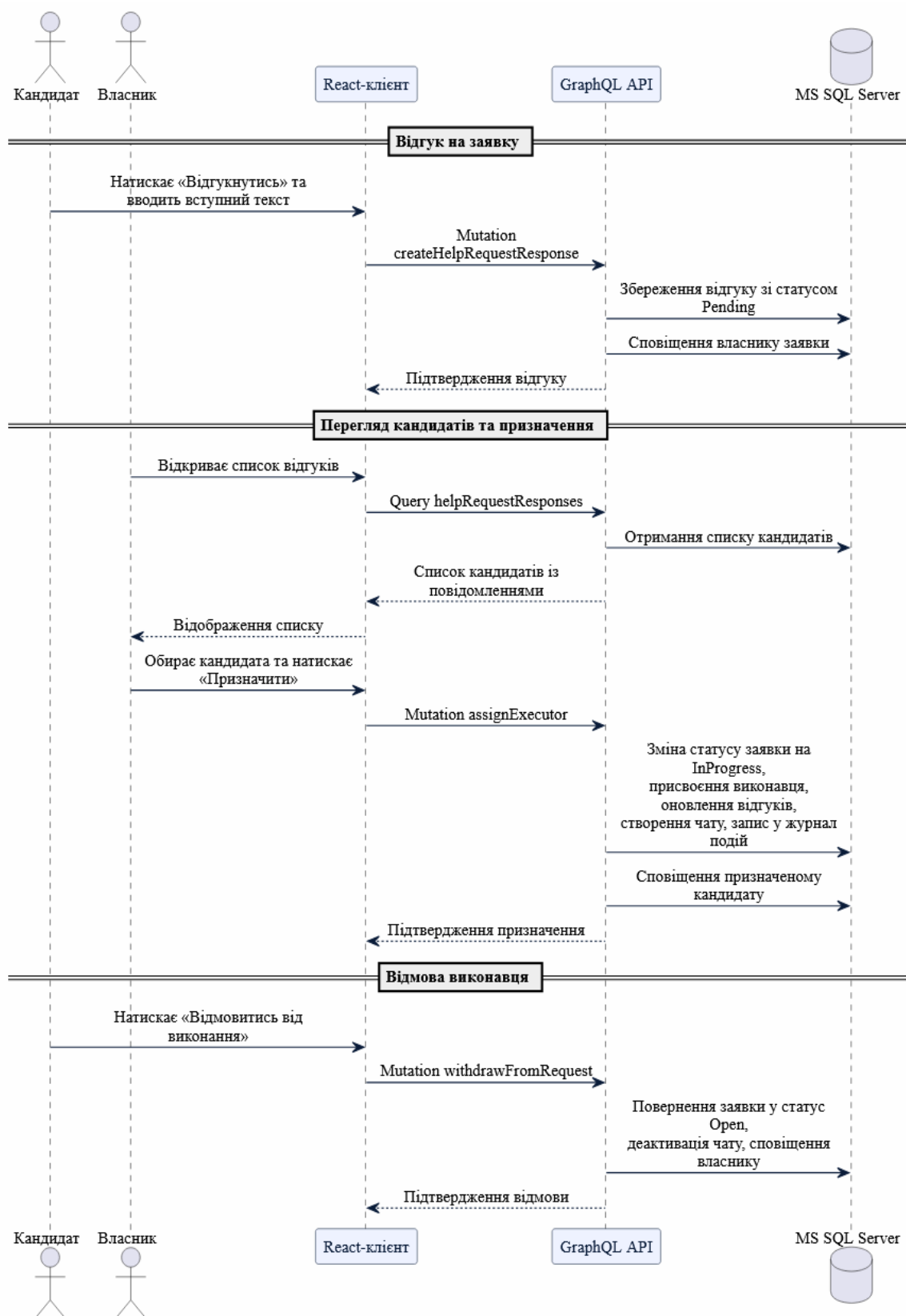


Рис. 2.7. Діаграма послідовності алгоритму призначення виконавця заявки

Після призначення виконавця учасники отримують доступ до механізму узгодження етапів – одного з ключових інструментів, що відрізняє платформу «Ethy» від звичайних чатів або дошок оголошень. Суть механізму полягає в тому, що будь-яка із двох сторін може запропонувати фіксацію певного проміжного результату виконання, однак ця фіксація набуває чинності лише після підтвердження протилежною стороною. Підтверджені етапи формують публічну, чисту та незмінну хронологію виконання заявки.

Ініціатором пропозиції може виступати як власник заявки, так і виконавець. Після надсилання пропозиції вона зберігається в таблиці HelpRequestStages зі статусом Pending та стає видимою іншій стороні безпосередньо в інтерфейсі чату. Інша сторона розглядає пропозицію та ухвалює рішення.

У разі підтвердження статус етапу змінюється на Confirmed, а в журнал подій HelpRequestEventLog вноситься запис типу StageConfirmed. Цей запис є незмінним і формує публічну хронологію заявки, доступну будь-якому відвідувачу платформи. У разі відхилення від іншої сторони вимагається обов'язкове зазначення причини – поле RejectionReason – після чого статус змінюється на Rejected, а в журнал подій вноситься запис типу StageRejected із зазначенням причини. Ініціатор отримує сповіщення в обох випадках.

Обов'язковість причини відхилення є свідомим проєктним рішенням: вона мінімізує ризик зловживань механізмом блокування прогресу та забезпечує прозорість процесу для обох сторін. Діаграму, що відображає повну послідовність взаємодії при узгодженні етапу, наведено на рис. 2.8.

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ІПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		55

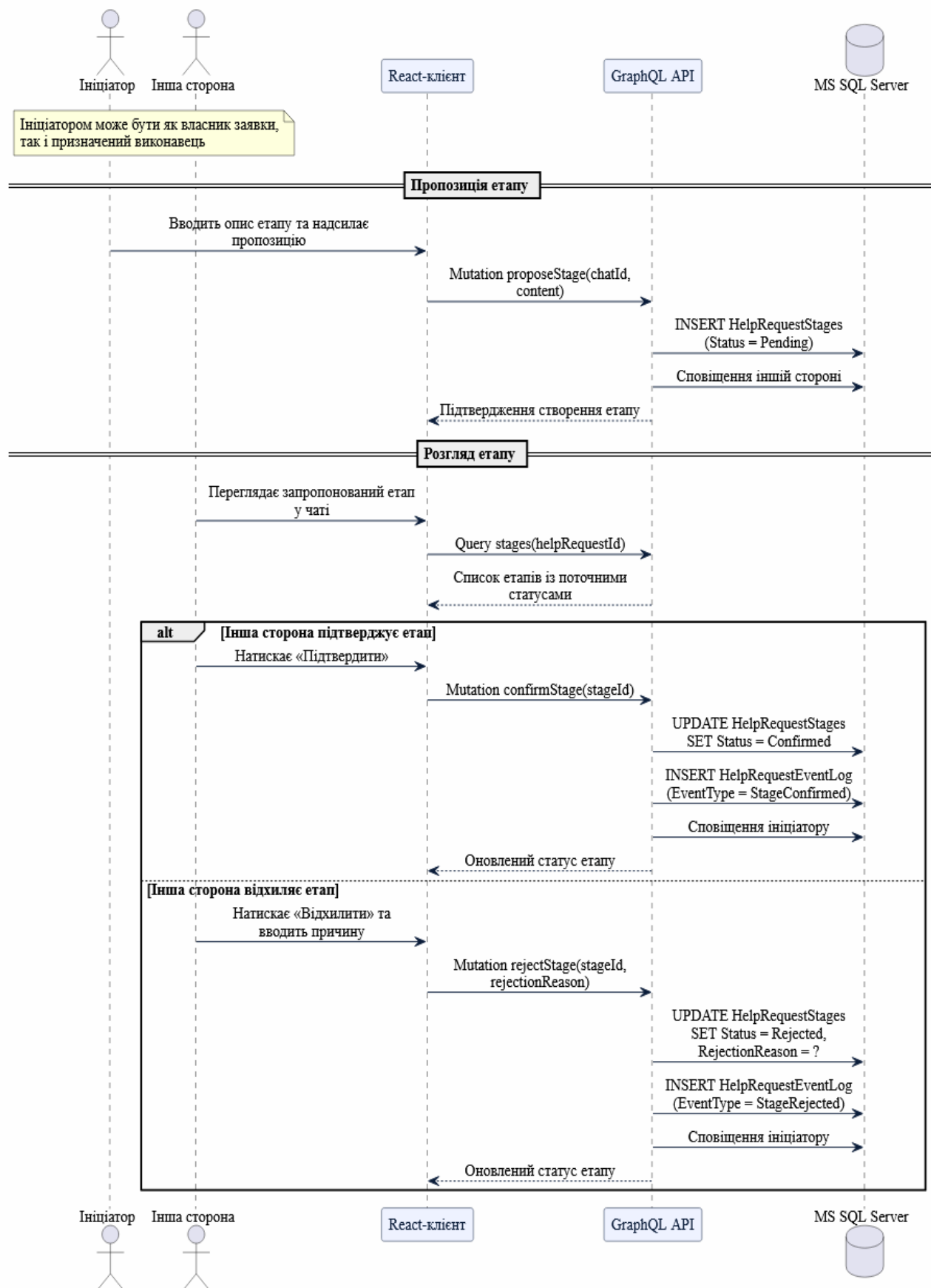


Рис. 2.8. Діаграма послідовності алгоритму узгодження етапу виконання заявки

Заявка створюється зі статусом Moderation і недоступна до рішення адміністратора: після схвалення переходить у Open і потрапляє у публічний список, після відхилення отримує термінальний статус Rejected. Зі статусу Open заявка переходить у InProgress при призначенні виконавця або у Cancelled при скасуванні власником. На відміну від Rejected, статус Cancelled не є термінальним – власник може відновити заявку. Зі статусу InProgress заявка повертається у Open при відмові виконавця чи його відхиленні власником.

Окрім основного статусу, поведінку заявки визначають два незалежні прапорці. Прапорець isHidden встановлюється адміністратором і приховує заявку з публічного списку без зміни її статусу. Прапорець isDeleted реалізує м'яке видалення на стороні власника. Повну діаграму станів із усіма переходами наведено на рис. 2.9.

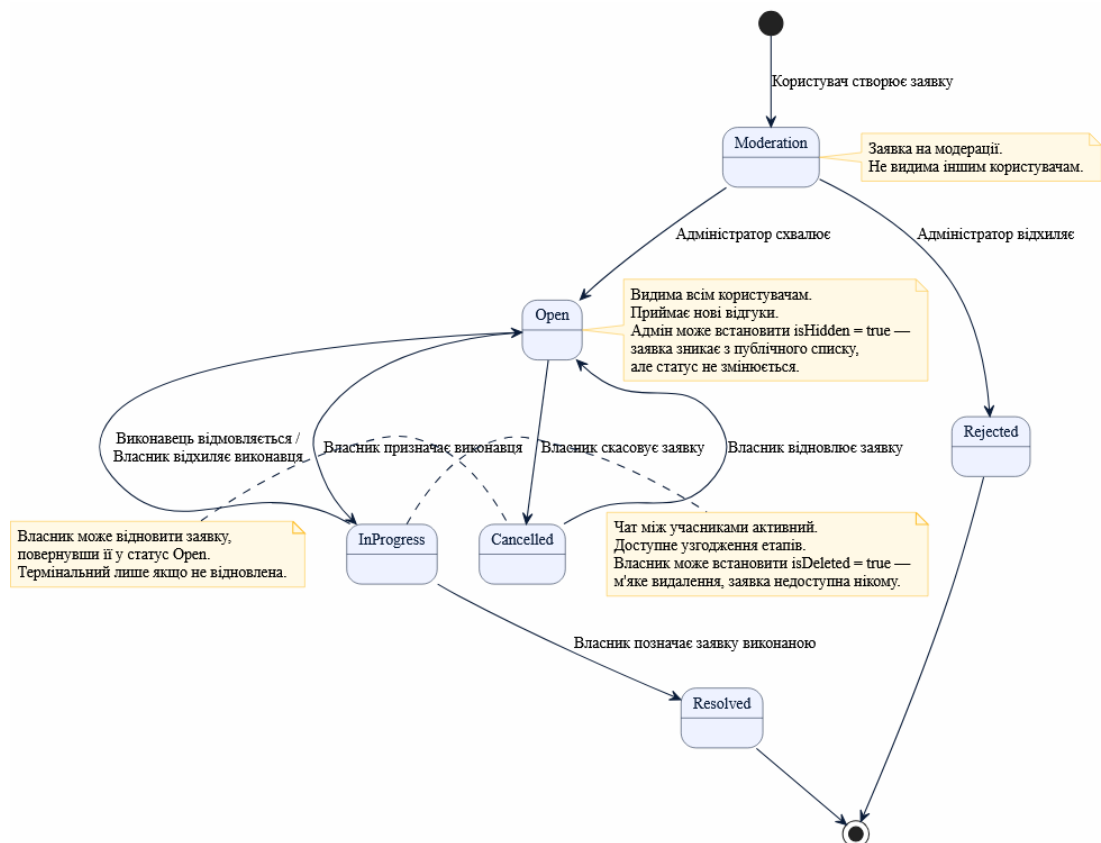


Рис. 2.9. Діаграма станів заявки на допомогу вебплатформи

2.4. Реалізація функціоналу

Реалізація функціональної частини платформи «Ethy» охоплює серверну та клієнтську сторони застосунку. На серверній стороні ключовими аспектами є організація обробки GraphQL-запитів через pipeline MediatR, реалізація двоетапного збереження медіафайлів засобами хмарного сховища Cloudinary, а також забезпечення реального часу взаємодії через SignalR. На клієнтській стороні – налаштування Apollo Client для роботи з GraphQL API та інтеграція SignalR-з'єднання у React-застосунок. Обробка запитів на серверній стороні побудована за принципом чіткого розподілу відповідальності між трьома шарами: GraphQL-схема приймає запит і делегує його прикладному рівню через MediatR, прикладний рівень виконує бізнес-логіку та звертається до репозиторіїв, репозиторії взаємодіють з базою даних через Dapper. Такий підхід відповідає принципам чистої архітектури [22] і дозволяє змінювати кожен шар незалежно від інших. Resolver у GraphQL-схемі відповідає виключно за отримання аргументів запиту, формування команди та передачу її до MediatR. Нижче наведено лістинг резолвера для операції створення заявки.

Лістинг резолвера createHelpRequests:

```
Field<AddHelpRequestPayloadType>("createHelpRequest")
    .AuthorizeWithPolicy("Verified")
    .Arguments(
        new QueryArguments(
            new QueryArgument<NonNullGraphType<StringGraphType>> { Name = "title" },
            new QueryArgument<NonNullGraphType<StringGraphType>> { Name = "description" },
            new QueryArgument<FloatGraphType> { Name = "latitude" },
            new QueryArgument<FloatGraphType> { Name = "longitude" },
            new QueryArgument<ListGraphType<StringGraphType>> { Name = "imageUrls" }
        )
    )
    .ResolveAsync(async context =>
    {
        var userId = context.GetUserId();
        var command = new AddHelpRequestCommand(
            CreatorId: userId,
            Title: context.GetArgument<string>("title"),
            Description: context.GetArgument<string>("description"),
            Latitude: context.GetArgument<double?>("latitude"),
            Longitude: context.GetArgument<double?>("longitude"),
            ImageUrls: context.GetArgument<List<string>>("imageUrls") ?? []
        );
```

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				58
Змн.	Арк.	№ докум.	Підпис	Дата		

```
var result = await mediator.Send(command);
return result.ToPayload((value, error) => new AddHelpRequestPayload(value, error));
});
```

Атрибут `.AuthorizeWithPolicy("Verified")` гарантує, що операція доступна лише користувачам із підтвердженою електронною поштою. Результат обробки команди повертається через узагальнений тип `Result<T>`, що дозволяє однаково обробляти як успішний результат, так і бізнес-помилку без використання винятків для керування потоком [23]. Метод-розширення `ToPayload` перетворює `Result<T>` у GraphQL payload-тип, приймаючи фабричну функцію, яка приймає або значення, або об'єкт помилки.

Команда `AddHelpRequestCommand` є незмінним записом (record), що містить усі вхідні дані операції та реалізує інтерфейс `IRequest<Result<AddHelpRequestResult>>` бібліотеки MediatR [24]. Обробник команди `AddHelpRequestHandler` отримує її через механізм впровадження залежностей і виконує повний сценарій створення заявки.

Лістинг обробника `AddHelpRequestHandler`:

```
public async Task<Result<AddHelpRequestResult>> Handle(
    AddHelpRequestCommand request, CancellationToken ct)
{
    var user = await _userRepository.GetByIdAsync(request.CreatorId, ct);
    if (user != null && user.Role == UserRole.User)
    {
        var activeCount = await _repository
            .CountActiveRequestsByCreatorAsync(request.CreatorId, ct);
        if (activeCount >= 3)
            return Result.Failure<AddHelpRequestResult>(
                new Error("Regular users can have a maximum of 3 active help requests.",
                    "HelpRequest.LIMIT_EXCEEDED"));
    }
    var permanentImageUrls =
        await _imageStorage.CommitHelpRequestImagesAsync(request.ImageUrls);
    var helpRequest = new HelpRequest(
        request.CreatorId, request.Title,
        request.Description, location);
    foreach (var url in permanentImageUrls)
        helpRequest.AddImage(url);
    await _repository.AddAsync(helpRequest, ct);
    return Result<AddHelpRequestResult>.Success(result);
}
```

Обробник реалізує кілька рівнів перевірок. Спочатку перевіряється ліміт активних заявок для звичайних користувачів – не більше трьох одночасно.

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		59

Волонтери цього обмеження не мають, що є частиною системи привілеїв платформи. Після успішних перевірок зображення переміщуються з тимчасового сховища в постійне, формується доменний об'єкт `'HelpRequest'` і зберігається через репозиторій. Усі бізнес-помилки повертаються як `'Result.Failure'` з відповідним кодом, а системні винятки перехоплюються блоком `'catch'` і також перетворюються на `'Result.Failure'`, що унеможливорює неконтрольоване поширення помилок на рівень GraphQL.

Доменний клас `'HelpRequest'` інкапсулює всі правила цілісності сутності. Публічні методи класу – `'AssignExecutor'`, `'Cancel'`, `'Restore'`, `'Edit'`, `'ResignAsExecutor'`, `'RemoveExecutor'`, `'Approve'`, `'RejectModeration'` – містять перевірки допустимих переходів між статусами і генерують `'DomainException'` з машинозчитуваним кодом помилки при порушенні інваріантів. Наприклад, метод `'AssignExecutor'` перевіряє, що заявка має статус `'Open'`, що відгук існує і має статус `'Pending'`, після чого підтверджує обраний відгук, відхиляє решту та переводить заявку у `'InProgress'`. Такий підхід до проєктування доменної моделі [25] гарантує, що жоден код поза класом не може перевести заявку у недопустимий стан.

Для зберігання зображень заявок, звітів та документів волонтерів використовується хмарний сервіс Cloudinary [26]. Процес збереження зображень розділено на два етапи, що дозволяє уникнути ситуації, коли файли завантажені, але пов'язаний запис у базі даних не було створено через помилку.

На першому етапі клієнт завантажує файли через REST-ендпоінт `'/api/files/help-requests'`. Сервіс `'CloudinaryImageStorageService'` валідує кожен файл двічі: перевіряє розширення зі списку дозволених та зчитує перші байти файлу для порівняння з еталонними сигнатурами відповідного формату. Перевірка сигнатури реалізована у статичному класі `'FileSignatureValidator'`.

Лістинг `FileSignaruteValidator`:

```
private static readonly Dictionary<string, List<byte[]>> _fileSignatures = new()
{
    { ".jpeg", new List<byte[]> { new byte[] { 0xFF, 0xD8, 0xFF } } },
    { ".jpg", new List<byte[]> { new byte[] { 0xFF, 0xD8, 0xFF } } },
}
```

		Торон М.Д.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		60

```

    { ".png", new List<byte[]> { new byte[] { 0x89, 0x50, 0x4E, 0x47, 0x0D, 0x0A, 0x1A, 0x0A } } },
    { ".gif", new List<byte[]> { new byte[] { 0x47, 0x49, 0x46, 0x38 } } },
    { ".webp", new List<byte[]> { new byte[] { 0x52, 0x49, 0x46, 0x46 } } }
};
public static bool IsValidImage(Stream fileStream, string extension)
{
    var ext = extension.ToLowerInvariant();
    if (!_fileSignatures.ContainsKey(ext)) return false;
    fileStream.Position = 0;
    using var reader = new BinaryReader(fileStream, Encoding.Default, leaveOpen: true);
    var headerBytes = reader.ReadBytes(_fileSignatures[ext].Max(m => m.Length));
    fileStream.Position = 0;
    return _fileSignatures[ext].Any(signature =>
        headerBytes.Take(signature.Length).SequenceEqual(signature));
}

```

Перевірка сигнатури є важливим заходом безпеки: зловмисник може перейменувати файл довільного формату, надавши йому допустиме розширення, однак підробити байтову сигнатуру без зміни вмісту файлу неможливо [27]. Після успішної валідації файл завантажується до тимчасової папки `ethy/temp/` у Cloudinary під іменем, що генерується як GUID.

На другому етапі, під час обробки команди створення заявки, метод `CommitHelpRequestImagesAsync` переміщує файли з `ethy/temp/` до `ethy/help-requests/` за допомогою операції перейменування Cloudinary Rename API. Якщо файл у тимчасовій папці не знайдено – наприклад, через закінчення сесії або повторну відправку форми – операція завершується винятком `FileNotFoundException`, який перехоплюється в обробнику команди і повертається клієнту як структурована помилка `HelpRequest.IMAGE\_WAS\_EXPIRED`. Аналогічна двоетапна схема застосовується для зображень звітів та документів верифікації волонтерів, з розміщенням у відповідних папках `ethy/reports/` та `ethy/volunteer-documents/`.

Чат між власником заявки та виконавцем, а також операції над етапами реалізовані через SignalR – бібліотеку ASP.NET Core для двосторонньої комунікації між сервером і клієнтом у реальному часі [28]. Усі методи хабу захищені атрибутом `[Authorize]`, тому підключення без дійсного JWT-токена відхиляється на рівні інфраструктури.

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		61

Центральним елементом є клас `ChatHub`, що успадковує `Hub` і отримує `IMediator` через конструктор. Такий підхід дозволяє хабу залишатися тонким транспортним шаром: він не містить бізнес-логіки, а лише отримує ідентифікатор користувача з claims, формує команду MediatR і транслює результат підключеним клієнтам.

Лістинг фрагменту ChatHub:

```
public async Task JoinChat(string helpRequestId)
{
    await Groups.AddToGroupAsync(Context.ConnectionId, helpRequestId);
}
public async Task SendMessage(string helpRequestId, string content)
{
    var senderId = GetUserId();
    var result = await _mediator.Send(
        new SendMessageCommand(Guid.Parse(helpRequestId), senderId, content));
    if (result.IsFailure)
    {
        await Clients.Caller.SendAsync("Error", result.Error.Code, result.Error.Message);
        return;
    }
    await Clients.Group(helpRequestId).SendAsync("ReceiveMessage", new
    {
        messageId = result.Value,
        senderId,
        content,
        createdAtUtc = DateTime.UtcNow
    });
}
public async Task ConfirmStage(string helpRequestId, string stageId)
{
    var userId = GetUserId();
    var result = await _mediator.Send(
        new ConfirmStageCommand(Guid.Parse(stageId), userId));
    if (result.IsFailure)
    {
        await Clients.Caller.SendAsync("Error", result.Error.Code, result.Error.Message);
        return;
    }
    await Clients.Group(helpRequestId).SendAsync("StageConfirmed", new
    {
        stageId,
        confirmedByUserId = userId,
        resolvedAtUtc = DateTime.UtcNow
    });
}
```

Ізоляція учасників чату забезпечується через механізм груп SignalR: при відкритті модального вікна чату клієнт викликає метод `JoinChat` з

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

ідентифікатором заявки, після чого з'єднання додається до відповідної групи. Усі наступні повідомлення та події етапів транслюються лише членам цієї групи через `Clients.Group(helpRequestId)`. При виникненні помилки сервер викликає `Clients.Caller.SendAsync("Error", ...)` – таким чином лише ініціатор операції отримує повідомлення про помилку, тоді як решта учасників групи не отримує зайвих сповіщень.

На клієнтській стороні взаємодія з GraphQL API реалізована через бібліотеку Apollo Client [29], а підключення до SignalR хабу – через офіційний пакет `@microsoft/signalr`.

Apollo Client налаштований із політиками кешування, що забезпечують баланс між актуальністю даних та мінімізацією мережових запитів.

Лістинг конфігурації Apollo Client:

```
const httpLink = new HttpLink({
  uri: `${import.meta.env.VITE_API_BASE_URL}/graphql`,
  credentials: 'include',
})
export const apolloClient = new ApolloClient({
  link: httpLink,
  cache: new InMemoryCache(),
  defaultOptions: {
    watchQuery: {
      fetchPolicy: 'cache-and-network',
      errorPolicy: 'all',
    },
    query: {
      fetchPolicy: 'cache-first',
      errorPolicy: 'all',
    },
    mutate: {
      errorPolicy: 'all',
    },
  },
})
```

Параметр `credentials: 'include'` забезпечує передачу HTTP-кук із JWT-токеном при кожному запиті до GraphQL API. Для підписок на дані в реальному часі (`watchQuery`) застосовується політика `cache-and-network`: клієнт негайно повертає кешовані дані для відображення, паралельно надсилаючи мережовий запит для їх оновлення. Для разових запитів (`query`) використовується `cache-first` – запит іде в мережу лише якщо даних у кеші немає. Значення `errorPolicy`:

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				63
Змн.	Арк.	№ докум.	Підпис	Дата		

'all' для всіх типів операцій дозволяє Apollo повертати часткові дані разом із помилками, а не відкидати весь результат при наявності будь-якої помилки у відповіді.

Підключення до SignalR реалізовано як singleton-з'єднання, що створюється при першому зверненні та повторно використовується впродовж сесії.

Лістинг підключення до SignalR:

```
let connection: signalR.HubConnection | null = null
export function getChatConnection(): signalR.HubConnection {
  if (!connection) {
    connection = new signalR.HubConnectionBuilder()
      .withUrl(`${API_BASE_URL}/hubs/chat`, { withCredentials: true })
      .withAutomaticReconnect()
      .build()
  }
  return connection
}
export async function startChatConnection(): Promise<void> {
  const conn = getChatConnection()
  if (conn.state === signalR.HubConnectionState.Disconnected) {
    await conn.start()
  } – asd
}
```

Метод `'withAutomaticReconnect()'` вмикає вбудований механізм автоматичного відновлення з'єднання при короткочасних мережевих збоях [30]. Перевірка стану `'HubConnectionState.Disconnected'` перед викликом `'start()'` запобігає спробам повторного підключення на вже активному з'єднанні. Підписка на конкретні події чату (`'ReceiveMessage'`, `'StageProposed'`, `'StageConfirmed'` тощо) виконується у React-компонентах через виклик `'connection.on(eventName, handler)'` після успішного встановлення з'єднання.

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		64

Висновки до другого розділу

У другому розділі виконано повний цикл проєктування та реалізації платформи «Ethy». Досліджено принципи побудови багатошарової веб-архітектури та обґрунтовано вибір технологічного стеку, що поєднує GraphQL API на базі ASP.NET Core, шаблон MediatR для організації прикладного рівня, Dapper для доступу до даних та React з Apollo Client на клієнтській стороні. Спроектовано реляційну схему бази даних, що охоплює повну доменну модель платформи, визначено стратегію індексування для прискорення ключових запитів та обґрунтовано рішення щодо організації зберігання та видалення даних. Досліджено підходи до моделювання бізнес-процесів засобами UML і побудовано діаграми послідовності та станів, що відображають ключові сценарії взаємодії між акторами системи та повний життєвий цикл центральної сутності платформи. Реалізовано функціональну частину системи із застосуванням патернів чистої архітектури та предметно-орієнтованого проєктування: обробку запитів через pipeline MediatR з доменною моделлю, що захищає інваріанти сутностей, двоетапну схему збереження медіафайлів із перевіркою байтових сигнатур, двосторонню комунікацію в реальному часі через SignalR, а також реактивне отримання та кешування даних на клієнтській стороні засобами Apollo Client.

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				65
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ КООРДИНАЦІЇ ГУМАНІТАРНОЇ ДОПОМОГИ

3.1. Реалізація функціоналу

Платформа «Ethy» складається з двох основних компонентів: серверного застосунку на базі ASP.NET Core та клієнтського застосунку на базі React. Обидва компоненти контейнеризовані за допомогою Docker [31], що забезпечує відтворюваність середовища виконання незалежно від цільової інфраструктури. Загальну топологію розгортання відображено на рис. 3.1.

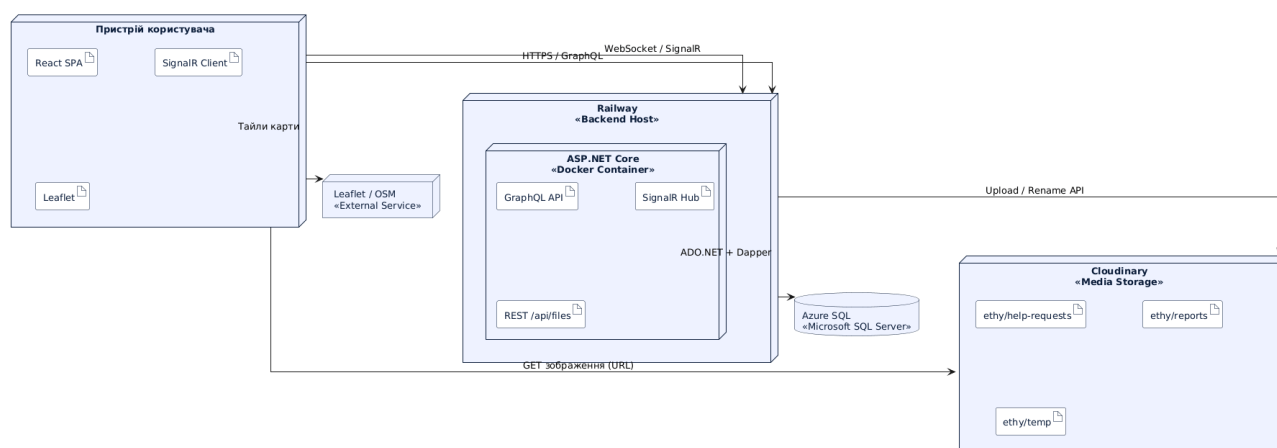


Рис. 3.1. Діаграма розгортання платформи «Ethy»

Серверний застосунок розгортається на хмарній платформі Railway [32], клієнтський застосунок – також на Railway як окремий сервіс. База даних Microsoft SQL Server розміщується на хмарній платформі Azure SQL [33]. Медіафайли (зображення заявок, звітів та документів верифікації) зберігаються у хмарному сервісі Cloudinary [34]. Для відображення карти використовується сервіс Mapbox [35].

Для локального розгортання та розробки необхідне наступне програмне забезпечення: .NET SDK 10 або вище, Node.js версії 20 або вище, MS SQL Server або доступ до хмарного екземпляру Azure SQL, Git для клонування репозиторію. Для розгортання у хмарному середовищі достатньо наявності Docker та облікових записів на Railway, Azure та Cloudinary.

Взаємодія з платформою здійснюється через будь-який сучасний веб-браузер із підтримкою WebSocket – зокрема Google Chrome версії 80 і вище, Mozilla Firefox версії 75 і вище, Microsoft Edge версії 80 і вище або Safari версії 14 і вище. Підтримка WebSocket є обов'язковою умовою для коректної роботи чату в реальному часі через SignalR.

Перед розгортанням необхідно налаштувати змінні середовища для обох компонентів. На серверній стороні конфігурація визначається через файл `appsettings.json` або змінні середовища, що мають пріоритет над файлом конфігурації. Таблиця 3.1 містить перелік обов'язкових змінних серверного застосунку.

Таблиця 3.1

Змінні середовища

Змінна	Опис
ConnectionStrings__DefaultConnection	Рядок підключення до MS SQL Server
Jwt_Key	Секретний ключ для підпису токенів
Jwt__Issuer	Ідентифікатор видавця токена
Jwt__Audience	Ідентифікатор аудиторії токена
Cloudinary__CloudName	Назва хмари Cloudinary
Cloudinary__ApiKey	API-ключ Cloudinary
Cloudinary__ApiSecret	Секрет API Cloudinary
Email__SmtpHost	Адреса SMTP-сервера
Email__SmtpPort	Порт SMTP-сервера
Email__Username	Логін облікового запису електронної пошти
Email__Password	Пароль облікового запису електронної пошти
Email__From	Адреса відправника листів
ClientUrl	URL клієнтського застосунку

На клієнтській стороні змінні середовища визначаються у файлі ``.env`` у кореневій директорії фронтенд-проєкту. Таблиця 3.2 містить перелік обов'язкових змінних клієнтського застосунку.

Таблиця 3.2

Змінні середовища клієнтського застосунку

Змінна	Опис
VITE_API_BASE_URL	Базова URL-адреса серверного API

Процес розгортання побудований на основі CI/CD пайплайну, реалізованого засобами GitHub Actions [36]. При кожному пуші у гілку ``master`` автоматично запускається послідовність кроків: збірка проєкту, виконання тестів, побудова Docker-образу та його публікація у реєстр контейнерів. Після успішної публікації образу Railway автоматично підтягує нову версію і перезапускає сервіс без простою.

Для першого розгортання необхідно клонувати репозиторій та створити проєкт на Railway з двома окремими сервісами – для серверного і клієнтського застосунків. У налаштуваннях кожного сервісу вказується шлях до відповідного ``Dockerfile``, після чого заповнюються змінні середовища згідно з таблицями 3.1 і 3.2. Паралельно створюється екземпляр Azure SQL Database, рядок підключення до якого вказується у змінній ``ConnectionStrings__DefaultConnection`` серверного сервісу. У Cloudinary отримуються значення ``CloudName``, ``ApiKey`` та ``ApiSecret`` у розділі Dashboard і вказуються у відповідних змінних.

Завершальним кроком налаштування є конфігурація секретів GitHub Actions у розділі Secrets репозиторію: ``DOCKER_USERNAME`` і ``DOCKER_PASSWORD`` для публікації образів у Docker Hub, а також токен Railway для автоматичного деплою. Після цього достатньо виконати перший пуш у гілку ``main`` – CI/CD пайплайн автоматично збере образи, опублікує їх і задеплойть на Railway. Міграції бази даних застосовуються автоматично при запуску серверного застосунку засобами DbUp, тому окремого кроку для ініціалізації схеми не потрібно.

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		68

Після успішного розгортання серверний застосунок стає доступним за адресою, яку Railway призначає автоматично у форматі `https://<назва-сервісу>.railway.app`. GraphQL API доступний за шляхом `/graphql`, SignalR хаб – за шляхом `/hubs/chat`.

3.2. Структура інтерфейсу платформи

Інтерфейс платформи «Ethy» реалізовано як односторінковий застосунок (SPA) [37] засобами бібліотеки React. Навігація між розділами відбувається без перезавантаження сторінки через React Router [38]. Дизайн-система побудована на Tailwind CSS [39] з підтримкою світлої та темної теми, що перемикається через налаштування у верхній панелі навігації. Основний шрифт інтерфейсу – Manrope, заголовки – Jura. Кольорова схема базується на темно-синьому (#0B1D3A) та жовтому акцентному (#FEC130) кольорах.

Для користувачів, що не авторизовані, доступна публічна частина платформи – перегляд загальної інформації та статистики платформи (рис. 3.2). Для отримання повного доступу до функціоналу необхідна реєстрація або вхід в обліковий запис. Форма реєстрації містить поля для введення імені користувача, електронної адреси та пароля (рис. 3.3). Після успішної реєстрації на вказану адресу надсилається лист із посиланням для підтвердження пошти – без підтвердження користувач не може виконувати операції, що потребують політики `Verified`.

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		69

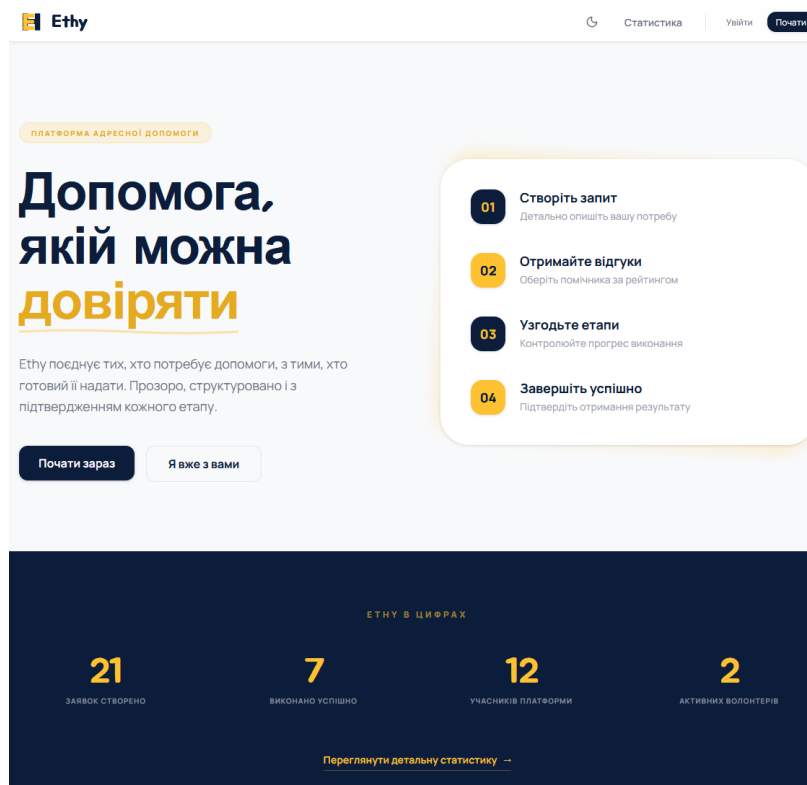


Рис. 3.2. Публічне вікно при вході на вебплатформу

Рис. 3.3. Вікно реєстрації

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

Головний екран авторизованого користувача – список відкритих заявок на допомогу (рис. 3.4). Кожна картка заявки відображає назву, короткий опис, дату створення, кількість кандидатів та мітку геолокації, якщо вказано координати. У верхній частині сторінки розміщено панель пошуку та фільтрації: користувач може фільтрувати заявки за статусом, а також здійснювати текстовий пошук за назвою та id. Список підтримує пагінацію.

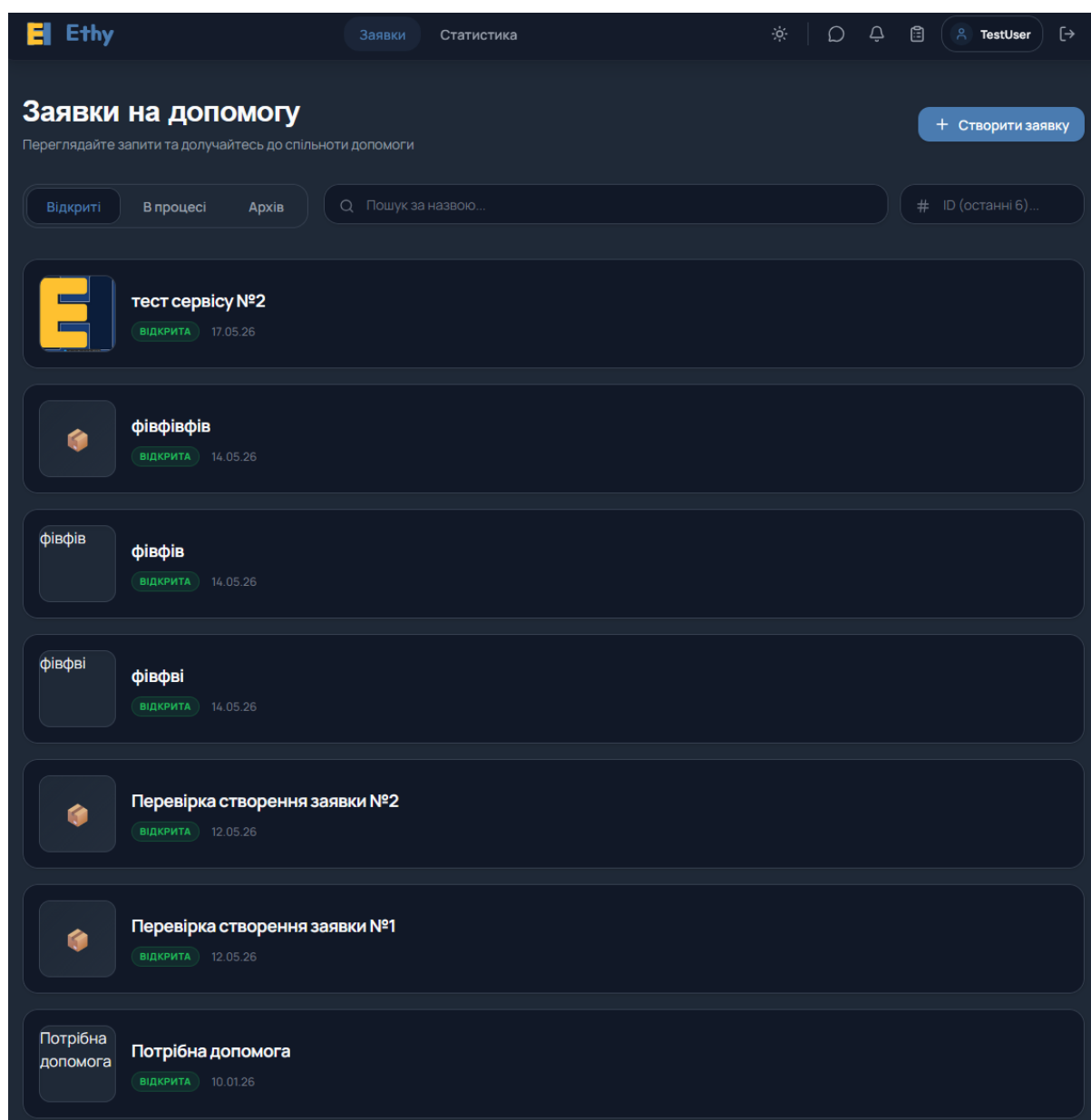


Рис. 3.4. Головний екран авторизованого користувача

		Торон М.Д.			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		71

Створення нової заявки доступне через кнопку у верхній навігаційній панелі (рис. 3.5). Форма містить поля для введення назви та опису заявки, інтерактивну карту на базі Leaflet для вибору місця надання допомоги шляхом встановлення маркера, а також завантаження до п'яти зображень. Зображення проходять попередню валідацію на клієнті та завантажуються у тимчасове сховище Cloudinary до надсилання основної форми. Після відправки форми заявка отримує статус Moderation і очікує на розгляд адміністратором (рис. 3.6).

Нова заявка

НАЗВА *

>Lorem Ipsum

11/200

ОПИС *

>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer eu sapien mollis, convallis mauris vel, auctor dolor. Aliquam quam massa, viverra vel elit ut, finibus elementum eros. Phasellus quis sodales ipsum. Duis tempus orci orci, a consectetur odio vestibulum quis. Vivamus eleifend enim at elit consequat, non posuere velit semper. Cras aliquet, sem sit amet bibendum fringilla, mi urna vehicula arcu, non porta quam metus id velit. Cras et vehicula odio, in eleifend est. Suspendisse rutrum ipsum.

827/4000

ФОТО (ДО 5)

Додати

ЛОКАЦІЯ (НЕОБОВ'ЯЗКОВО)

Мое місцезнаходження

+

-

Житомир

Leaflet | © OpenStreetMap

50.03597, 28.63037

Очистити

Скасувати

Створити заявку

Рис. 3.5. Вікно створення заявки

		Торон М.Д.			ПІЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				72
Змн.	Арк.	№ докум.	Підпис	Дата		

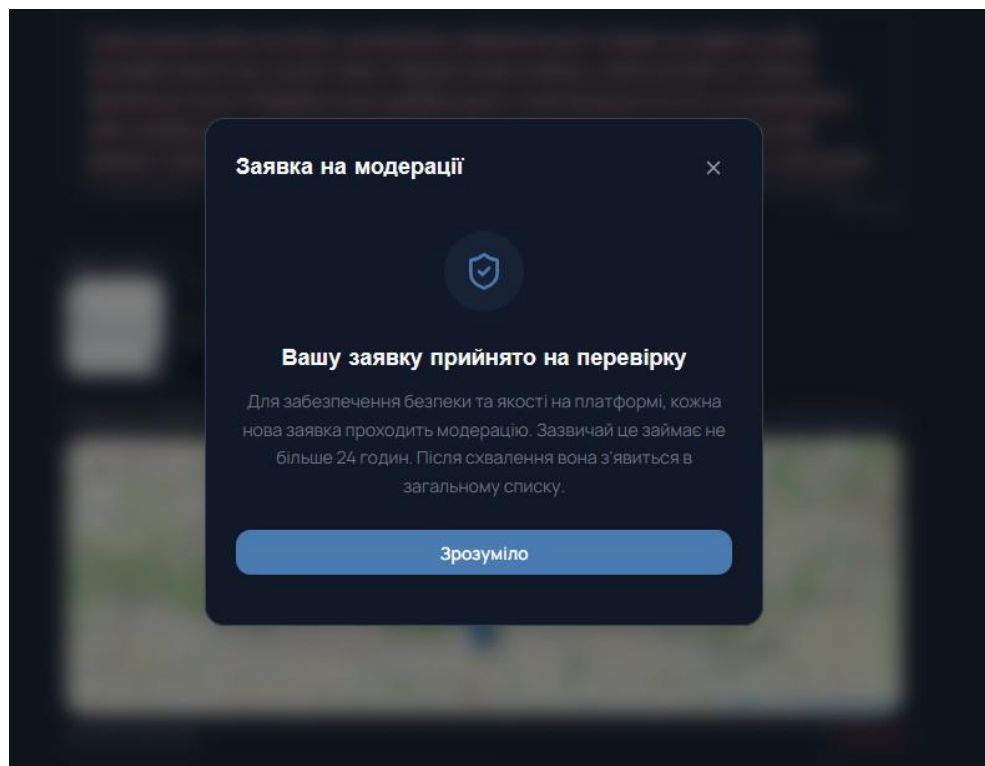


Рис. 3.6. Повідомлення після створення заявки

Сторінка деталей заявки є центральним екраном платформи і містить кілька функціональних зон (рис. 3.7). У верхній частині відображається повна інформація про заявку: назва, опис, галерея зображень, геолокація на карті, статус та дата створення. Нижче розміщено блок дій, доступний набір яких залежить від ролі поточного користувача та статусу заявки. Власник заявки зі статусом Open може редагувати її, скасувати із зазначенням причини або видалити. При статусі Cancelled доступна дія відновлення заявки.

Окрема вкладка «Кандидати» відображає список користувачів, що відгукнулися на заявку, із зазначенням їхнього вступного повідомлення, репутаційних лічильників та посилання на публічний профіль. Власник може призначити будь-якого кандидата виконавцем – після цього решта відгуків автоматично отримують статус Rejected, а заявка переходить у статус InProgress.

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		73

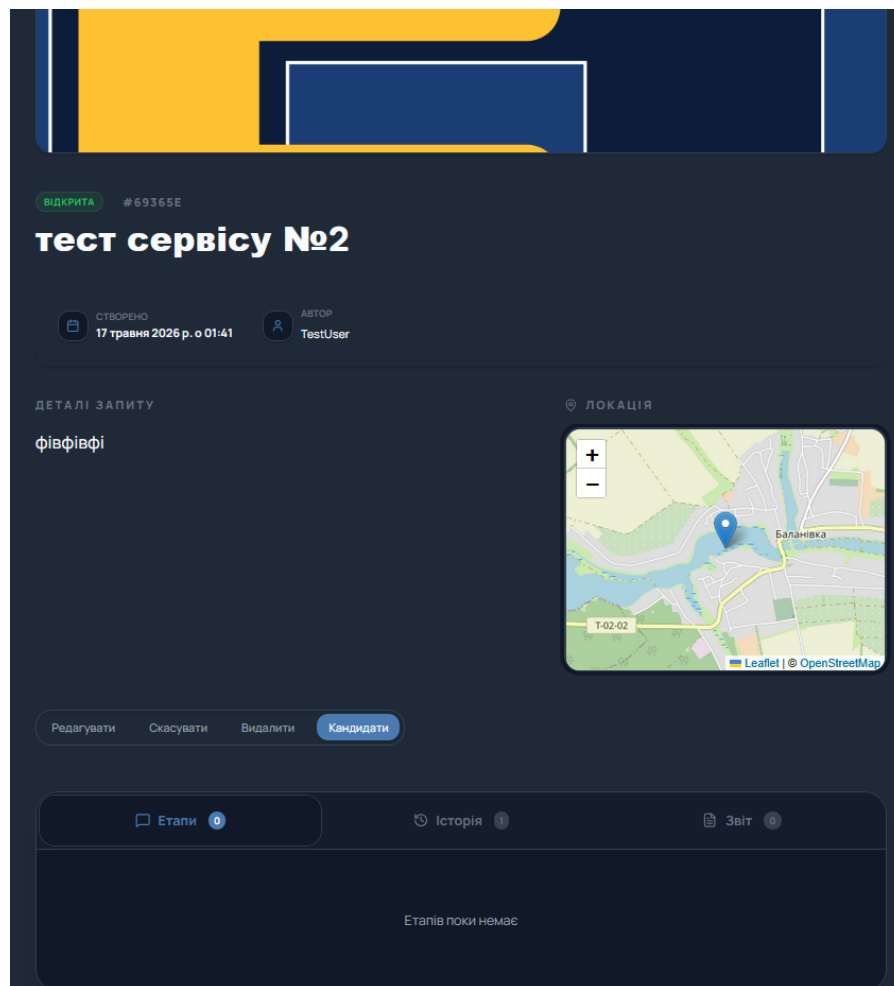


Рис. 3.7. Сторінка з детальним описом заявки

Після призначення виконавця обом учасникам стає доступний чат у реальному часі, реалізований через SignalR. Інтерфейс чату містить стрічку повідомлень із відображенням імені відправника та часу надсилання, адаптованого до часового поясу користувача, а також поле введення нового повідомлення.

У межах того ж вікна розміщено блок узгодження етапів виконання. Будь-який із учасників може запропонувати новий етап, ввівши його опис. Запропонований етап відображається у стрічці зі статусом Pending (рис. 3.8) та кнопками підтвердження і відхилення для іншої сторони (рис. 3.9). При відхиленні обов'язково вказується причина. Підтверджені етапи формують хронологію виконання заявки, видиму всім відвідувачам сторінки. Перед тим як позначити заявку виконаною, власник зобов'язаний вирішити всі незакриті етапи.

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		74

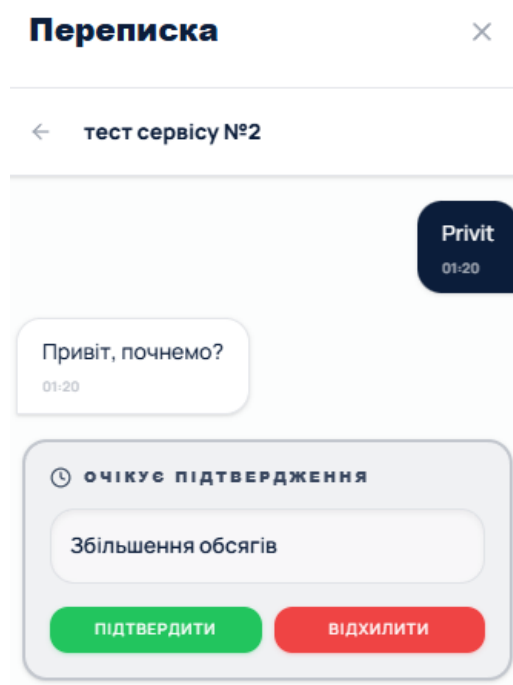


Рис. 3.8. Чат та запропонований етап

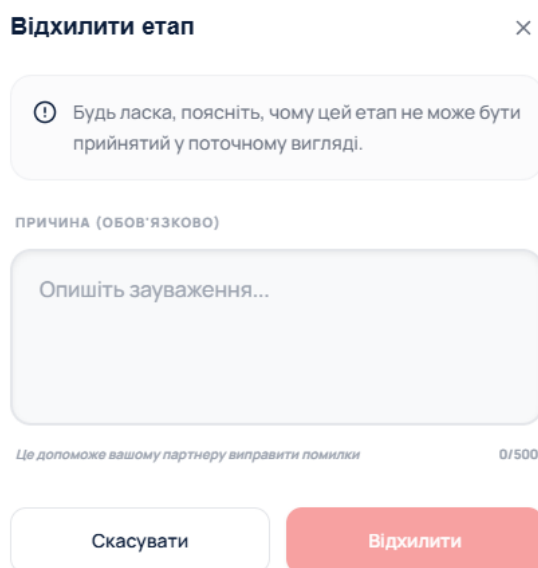


Рис. 3.9. Вікно відхилення етапу

Після завершення заявки власник може додати звіт: короткий текстовий коментар та зображення результату (рис. 3.10). Звіт є публічним і відображається на сторінці заявки для всіх відвідувачів як підтвердження факту надання допомоги.

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				75
Змн.	Арк.	№ докум.	Підпис	Дата		

Окрема вкладка «Історія подій» відображає автоматично сформовану хронологію ключових дій із заявкою: призначення виконавця, зміни статусу, підтвердження та відхилення етапів (рис. 3.11). Кожен запис містить тип події, ім'я актора та часову мітку. Журнал є незмінним і слугує аудиторським слідом усієї історії заявки.

Важливою функцією для створення публічного авторитету та довіреності особи – можливість залишити відгук, як помічнику, так і власнику заявки, після її виконання (рис. 3.12).

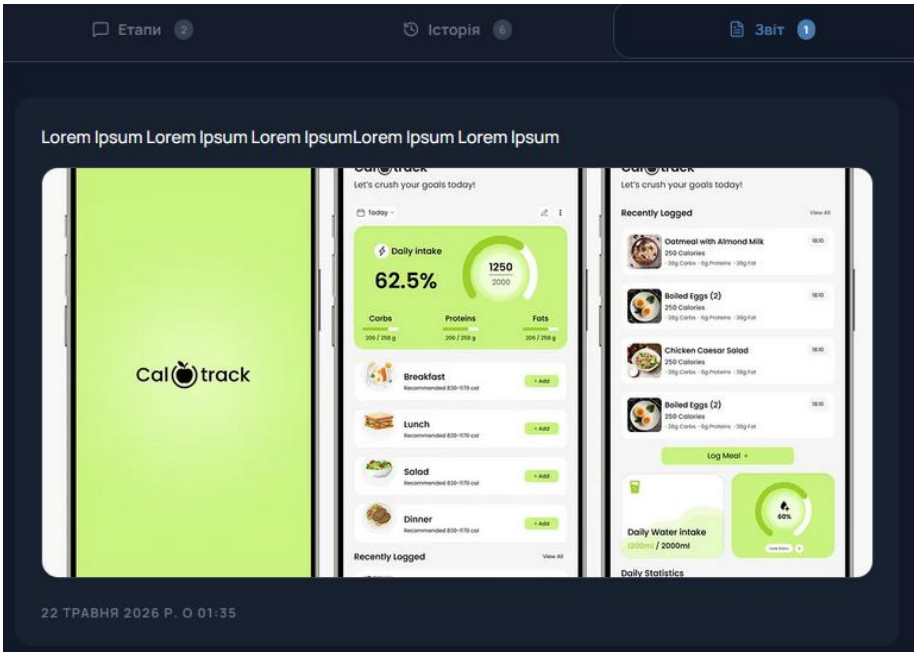


Рис. 3.10. Вкладення звіту заявки



Рис. 3.11. Вкладення історії подій заявки

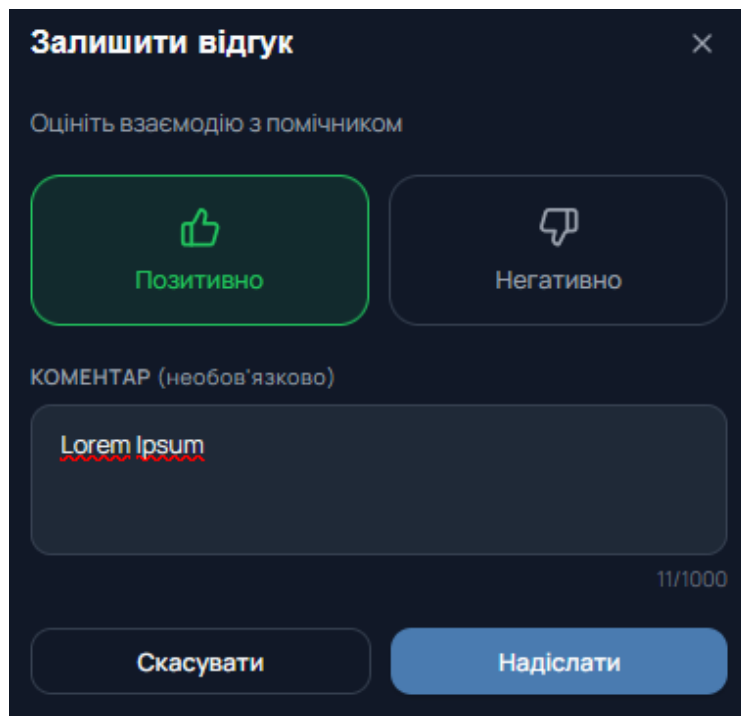


Рис. 3.12. Вкладення відгуку на користувача

Сторінка власного профілю містить дві зони: редагування особистих даних та репутаційний блок та блок пов'язаних із користувачем заявок (рис. 3.13). У зоні редагування користувач може змінити ім'я, номер телефону та посилання на соціальні мережі з відображенням типу мережі. Окремо відображається статус підтвердження електронної пошти та наявність номера телефону у вигляді візуальних міток.

Репутаційний блок містить лічильники позитивних та негативних відгуків, статистику про кількість створених заявок, виконаних, надано допомогу у виконанні і загальний відсоток успішності власних заявок. Нижче розміщено вкладки із власними заявками користувача, розділеними на ті, які він створював, та ті, у виконанні яких брав участь як виконавець. Окремий розділ профілю надає доступ до форми подання заявки на статус волонтера із завантаженням підтверджуючого документа та описом діяльності.

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				77
Змн.	Арк.	№ докум.	Підпис	Дата		

Мій Профіль

#0e4204

НА ПЛАТФОРМІ З 4 ТРАВНЯ 2026 Р. О 13:05

ВОЛОНТЕР

ІМ'Я КОРИСТУВАЧА

TestUser

EMAIL АДРЕСА

gigi@gmail.com

ПІДТВЕРДЖЕНО

БЕЗПЕКА

ЗМІНИТИ ПАРОЛЬ

КОНТАКТИ

РЕДАГУВАТИ

Номер не вказано

Соц. мережі не вказано

3 НАМИ З ТРАВЕНЬ 2026 Р.

ВИДАЛИТИ АКАУНТ

СТАТИСТИКА

12

СТВОРЕНО ЗАПИТІВ

3

ВИКОНАНО ДЛЯ МЕНЕ

3

ДОПОМІГ ІНШИМ

25%

РЕЙТИНГ УСПІШНОСТІ

(ОПІДКРИТЕНО)

РЕПУТАЦІЯ

ОСТАННІ ВІДГУКИ

1

ПОЗИТИВНИХ

1

НЕГАТИВНИХ

testuser

Пишу від імені помічника

15.05.26

testuser

посварились в чаті

12.05.26

АКТИВНІСТЬ

Шукаю допомогу

Надаю допомогу

Всі

Активні

Історія

Рис. 3.13. Вікно особистого профілю

Будь-який авторизований користувач може переглянути публічний профіль іншого учасника платформи (рис. 3.14). Публічний профіль відображає ім'я, репутаційні лічильники, дату реєстрації, останню активність, мітки підтвердження пошти та наявності телефону, а також посилання на соціальні мережі. Нижче відображається список заявок користувача, розділений на створені та виконані. Зі сторінки публічного профілю доступна можливість залишити скаргу на користувача із зазначенням причини – з обмеженням на кількість скарг на добу та на одного користувача (рис. 3.15).

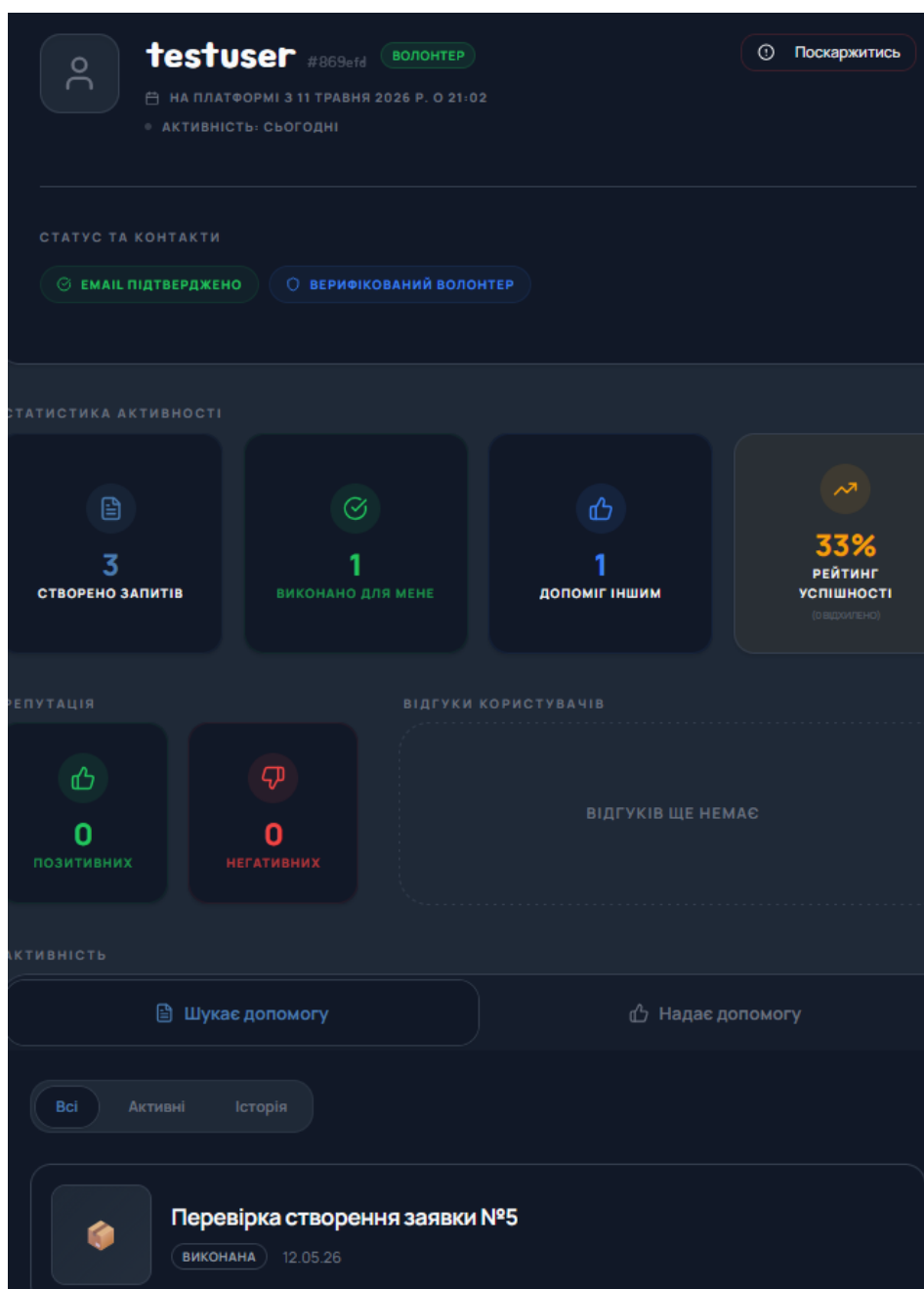


Рис. 3.14. Вікно публічного профілю

		Торон М.Д.			ПІЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				79
Змн.	Арк.	№ докум.	Підпис	Дата		

Поскаржитись на користувача

Скарга на testuser

ПРИЧИНА СКАРГИ

Опишіть що сталося...

0/1000

СКАРГ НА СЬОГОДНІ: 0/15

Скасувати

Надіслати скаргу

Рис. 3.15. Модальне вікно скарги

Вкладка сповіщень доступна через іконку у верхній навігаційній панелі (рис. 3.16). Вона відображає хронологічний список подій, що стосуються поточного користувача: нові повідомлення у чаті, результати модерації заявок, відповіді на скарги, призначення виконавцем, підтвердження або відхилення етапів. Непрочитані сповіщення виділяються візуально та позначаються як прочитані при кліку. Кожне сповіщення є інтерактивним і веде до відповідного розділу платформи.

СПОВІЩЕННЯ

✓ ПОЗНАЧИТИ ВСІ

Нове повідомлення від testuser

3 год тому

Privit

Новий відгук!

3 год тому

Хтось запропонував вам допомогу у заявці. Перегляньте деталі.

Заявку схвалено

123 год тому

Ваша заявка "тест сервісу №2" пройшла модерацію і тепер доступна для всіх.

Припинення співпраці

133 год тому

Власник заявки "Потрібна допомога" припинив вашу участь у ролі помічника.

Рис. 3.16. Вікно сповіщень

		Торон М.Д.			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				80
Змн.	Арк.	№ докум.	Підпис	Дата		

Панель адміністратора доступна лише користувачам із роллю Admin і містить кілька незалежних вкладок (рис. 3.17). Вкладка модерації заявок відображає всі заявки зі статусом Moderation у вигляді таблиці з можливістю переходу до деталей кожної. Переглянувши заявку, адміністратор може схвалити її – заявка переходить у статус Open і стає публічною – або відхилити із зазначенням причини, після чого вона отримує статус Rejected. Для опублікованих заявок адміністратор може встановити прапорець isHidden для тимчасового приховання без зміни статусу.

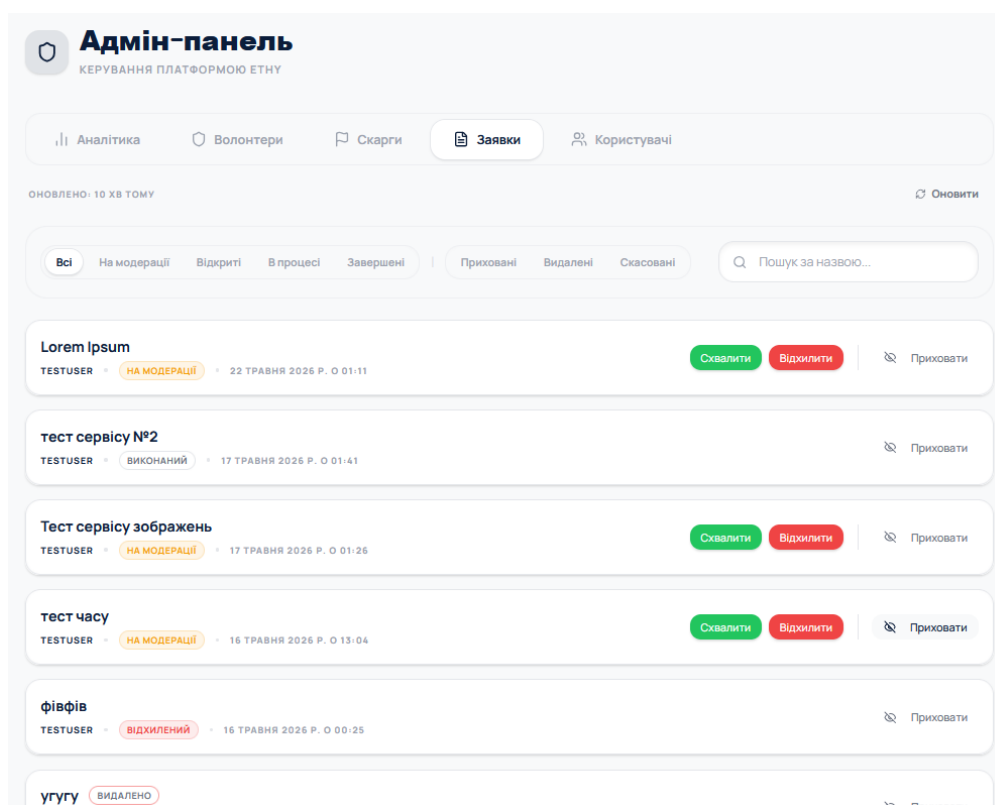


Рис. 3.17. Адмін-панель заявок користувачів

Вкладка користувачів відображає повний список зареєстрованих облікових записів із фільтрацією за роллю та статусом. Для кожного користувача адміністратор може переглянути профіль, заблокувати обліковий запис на визначений термін із зазначенням причини або видалити його (рис. 3.18). Вкладка скарг відображає список поданих скарг із зазначенням автора, цільового користувача та причини (рис. 3.19). Адміністратор розглядає кожну скаргу і залишає коментар при відхиленні.

		Торон М.Д.			ПІЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				81
Змн.	Арк.	№ докум.	Підпис	Дата		

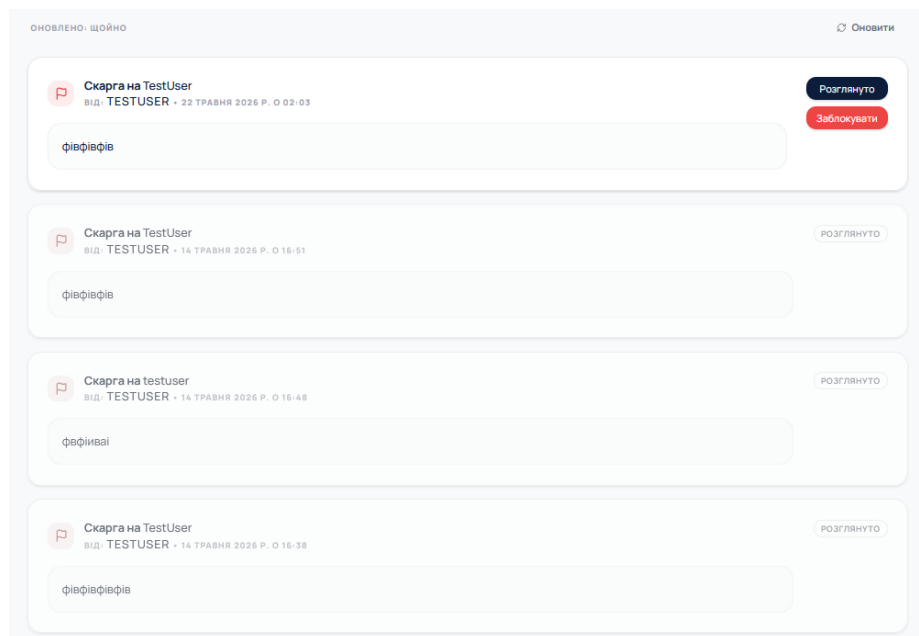


Рис. 3.18. Розділ скарг користувачів

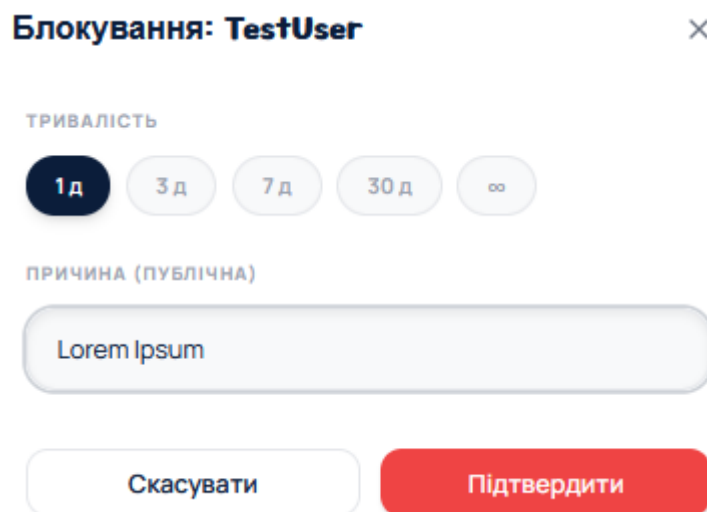


Рис. 3.19. Модальне вікно при блокуванні користувача

Вкладка волонтерів відображає список поданих заявок на отримання статусу волонтера зі статусом Pending. Кожна заявка містить назву організації, опис діяльності та завантажений документ (рис. 3.20). Адміністратор може схвалити або відхилити заявку з коментарем. Вкладка статистики надає зведену інформацію про стан платформи: розподіл заявок за статусами, відсоток успішно виконаних, середній час виконання та рейтинг волонтерів за кількістю виконаних заявок (рис. 3.21).

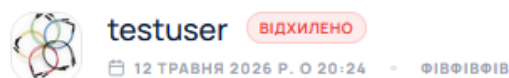
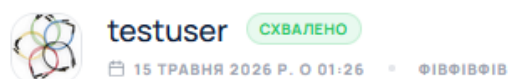


Рис. 3.20. Розділ заявок на статус волонтера

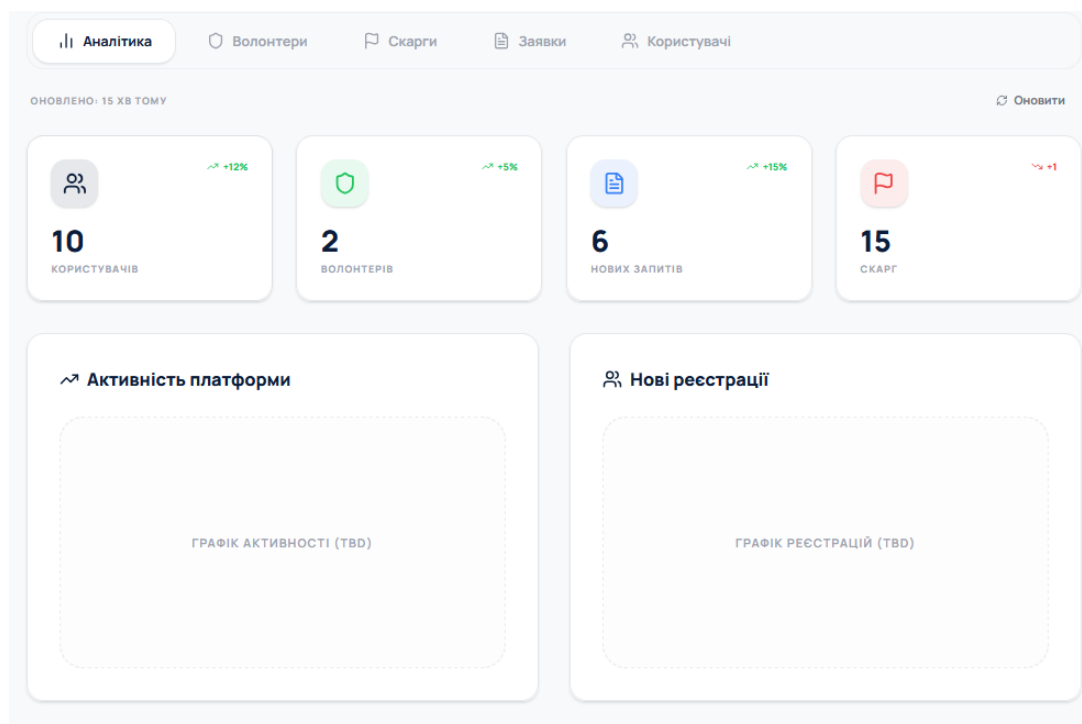


Рис. 3.21. Розділ аналітики в адмін-панелі

3.3. Тестування платформи

Тестування серверної частини платформи «Ethy» охоплює два рівні: модульне тестування ізольованої бізнес-логіки та інтеграційне тестування повного стеку від GraphQL-запиту до бази даних. Такий підхід відповідає піраміді тестування [40], де швидкі ізольовані юніт-тести забезпечують покриття доменних інваріантів, а інтеграційні – перевіряють коректність взаємодії між шарами застосунку в умовах, наближених до реального середовища. Усі тести реалізовані з використанням фреймворку xUnit [41] та бібліотеки FluentAssertions [42] для виразних перевірок.

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				83
Змн.	Арк.	№ докум.	Підпис	Дата		

Модульні тести зосереджені на перевірці обробників команд прикладного рівня в ізоляції від зовнішніх залежностей. Залежності замінюються тестовими замінниками (mock-об'єктами) засобами бібліотеки Moq [43], що дозволяє перевіряти поведінку обробника незалежно від стану бази даних або хмарного сховища.

Тести для «AddHelpRequestHandler» охоплюють три сценарії: успішне створення заявки з коректними даними, повернення помилки «HelpRequest.IMAGE_WAS_EXPIRED» при недоступності тимчасових зображень у Cloudinary, та повернення помилки «HelpRequest.LIMIT_EXCEEDED» при досягненні ліміту активних заявок для звичайного користувача.

Лістинг модульного тесту перевірки ліміту активних заявок:

```
[Fact]
public async Task Handle_ShouldReturnFailure_WhenUserReachesLimit()
{
    // Arrange
    var creatorId = Guid.NewGuid();
    var command = new AddHelpRequestCommand(creatorId, "Title", "Desc", null, null, new
List<string>());
    var user = new User("test", "test@test.com", "hash", "salt", UserRole.User);
    _userRepositoryMock.Setup(u => u.GetByIdAsync(creatorId, It.IsAny<CancellationToken>()))
        .ReturnsAsync(user);
    _repositoryMock.Setup(r => r.CountActiveRequestsByCreatorAsync(creatorId,
It.IsAny<CancellationToken>()))
        .ReturnsAsync(3);
    // Act
    var result = await _handler.Handle(command, Cancellation.Token.None);
    // Assert
    result.IsSuccess.Should().BeFalse();
    result.Error.Code.Should().Be("HelpRequest.LIMIT_EXCEEDED");
    _repositoryMock.Verify(repo => repo.AddAsync(
        It.IsAny<HelpRequest>(), It.IsAny<CancellationToken>()), Times.Never);
}
```

Тести для «RegisterUserHandler» перевіряють два граничні випадки: спробу реєстрації з вже існуючою електронною адресою – очікується помилка «User.USER_ALREADY_EXISTS» і відсутність виклику репозиторію – та успішну реєстрацію нового користувача, де перевіряється що обліковий запис збережено з правильними даними, токен верифікації створено і лист підтвердження надіслано.

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				84
Змн.	Арк.	№ докум.	Підпис	Дата		

Інтеграційні тести запускають повноцінний ASP.NET Core застосунок у тестовому середовищі засобами «WebApplicationFactory» [44] і виконують реальні GraphQL-запити через «HttpClient». База даних розгортається у Docker-контейнері за допомогою бібліотеки Testcontainers [45] – перед кожною тестовою сесією автоматично піднімається ізольований екземпляр MS SQL Server, на якому застосовуються всі міграції DbUp. Це гарантує що тести завжди виконуються на чистій, актуальній схемі бази даних.

Базовий клас «IntegrationTestBase» реалізує спільну інфраструктуру: метод «SendGraphQLAsync» для надсилання GraphQL-запитів із опціональним токеном авторизації, «RegisterAndLoginAsync» для реєстрації тестового користувача та отримання JWT-токена, «CreateHelpRequestAsync» і «ApproveHelpRequestAsync» для підготовки передумов тестів.

Клас «HelpRequestLifecycleTests» містить наскрізний тест повного циклу заявки. Тест послідовно реєструє двох користувачів, створює заявку, проводить її через модерацію адміністратором, оформлює відгук, призначає виконавця та завершує заявку зі звітом – перевіряючи що кожна операція повертає успішний результат без помилок.

Лістинг фрагменту інтеграційного тесту повного циклу заявки:

```
[Fact]
public async Task HappyPath_Create_Respond_Assign_Complete_Report_ShouldWork()
{
    var creator = await RegisterAndLoginAsync("creator_a", "creator_a@test.com",
"Password123!");
    var volunteer = await RegisterAndLoginAsync("volunteer_b", "volunteer_b@test.com",
"Password123!");
    var helpRequestId = await CreateHelpRequestAsync(creator.Token, "Test Help Request", "Need
help.");
    await ApproveHelpRequestAsync(helpRequestId);
    // Volunteer responds
    var respondResult = await SendGraphQLAsync(respondMutation,
        new { id = helpRequestId.ToString(), message = "I can help!" }, volunteer.Token);
    // ...verify no error...

    // Creator assigns volunteer
    var assignResult = await SendGraphQLAsync(assignMutation,
        new { hrId = helpRequestId.ToString(), respId = responseId }, creator.Token);
    // ...verify no error...
    // Creator completes the request
    var changeStatusResult = await SendGraphQLAsync(changeStatusMutation,
```

		Торон М.Д.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмотря О.В.				85
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    new { id = helpRequestId.ToString(), status = "RESOLVED" }, creator.Token);
// ...verify no error...
}

```

Окремий клас «ChatAndStagesTests» тестує сценарії роботи з чатом та етапами через SignalR: підключення до хабу, пропозицію та підтвердження етапу з перевіркою що події доставляються всім учасникам групи. «HelpRequestAssigneeManagementTests» перевіряє сценарії відмови виконавця та відхилення виконавця власником, контролюючи що заявка коректно повертається у статус Open і чат деактивується.

Запуск тестів інтегровано у пайплайн GitHub Actions через workflow «ci.yml», що спрацьовує при кожному пуші у гілки «main» та «dev», а також при створенні pull request (Додаток Г). Пайплайн послідовно виконує відновлення залежностей, збірку рішення та запуск юніт-тестів і інтеграційних тестів окремими кроками. Загальний CI/CD пайплайн «ci.yml» організований у чотири стадії – build, test, docker, deploy – де кожна наступна стадія запускається лише після успішного завершення попередньої (рис. 3.22). Це унеможливлює публікацію Docker-образу або ініціювання деплою при наявності помилок у тестах.

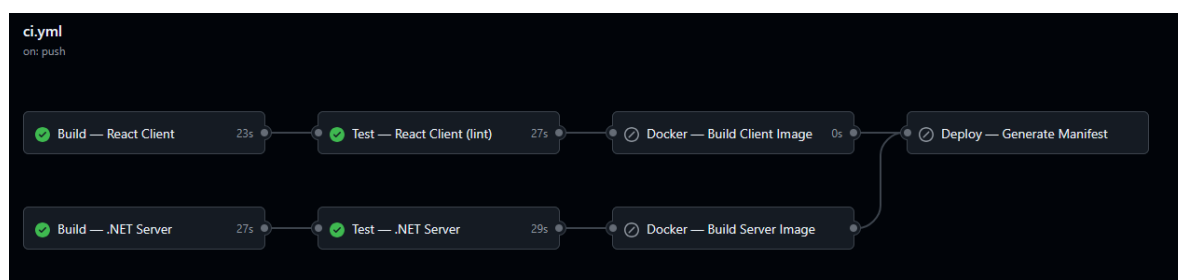


Рис. 3.22. CI/CD пайплайн платформи «Ethy» у GitHub Actions

Паралельно з автоматизованими тестами проводилося мануальне тестування GraphQL API через вбудований GraphQL Playground, доступний за адресою «/graphql» у режимі розробки. Це дозволяло перевіряти схему, тестувати окремі запити та мутації безпосередньо під час розробки нового функціоналу. Для тестування REST-ендпоінтів завантаження файлів використовувався інструмент Postman [46] – зокрема для перевірки коректності обробки multipart/form-data запитів (рис. 3.23), валідації байтових сигнатур зображень та двоетапного переміщення файлів у Cloudinary.

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				86
Змн.	Арк.	№ докум.	Підпис	Дата		

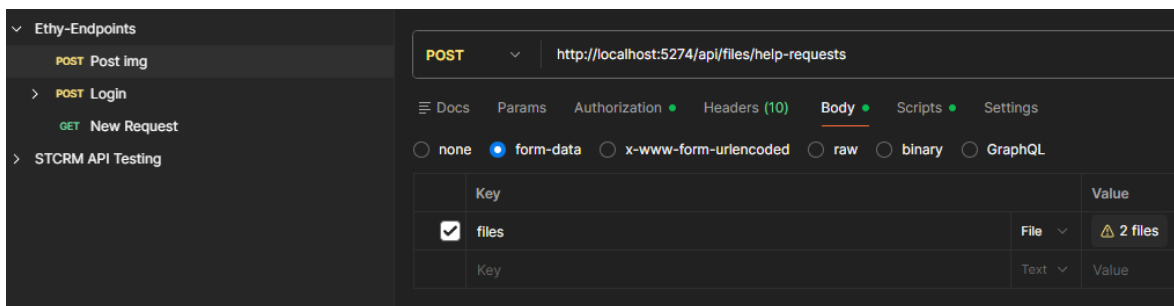


Рис. 3.23. Тестування завантаження файлів у Postman

Висновки до третього розділу

У третьому розділі розглянуто практичні аспекти розгортання та експлуатації вебплатформи «Ethy». Описано топологію хмарної інфраструктури, що включає розміщення серверного та клієнтського застосунків на платформі Railway, бази даних на Azure SQL і медіафайлів у хмарному сховищі Cloudinary. Визначено вимоги до середовища виконання, перелік обов'язкових змінних конфігурації та покроковий порядок першого розгортання через CI/CD пайплайн GitHub Actions. Проведено опис інтерфейсу користувача, що охоплює всі ключові екрани платформи: автентифікацію, список і деталі заявок, чат із механізмом узгодження етапів, профіль користувача, систему сповіщень та панель адміністратора. Верифіковано коректність реалізованого функціоналу на двох рівнях: автоматизованому – через модульні та інтеграційні тести з використанням xUnit, Moq і Testcontainers в ізольованому середовищі з реальною базою даних, та мануальному – через тестування GraphQL API у вбудованому Playground і перевірку завантаження файлів у Postman.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи спроектовано та реалізовано вебплатформу координації адресної допомоги «Ethy», що вирішує актуальну проблему неефективної організації волонтерської взаємодії в умовах інформаційного перевантаження. У першому розділі проведено аналіз предметної галузі, досліджено наявні платформи-аналоги та виявлено їх ключові обмеження: відсутність структурованого підтвердження факту надання допомоги, непрозорість процесу взаємодії між учасниками та відсутність системи довіри. На основі цього сформульовано вимоги до системи та обґрунтовано її концептуальну унікальність. У другому розділі обрано технологічний стек на основі ASP.NET Core, GraphQL.NET, MediatR, Dapper і React з Apollo Client, спроектовано реляційну схему бази даних з п'ятнадцяти таблиць, розроблено діаграми UML для ключових бізнес-процесів і реалізовано функціональну частину системи із застосуванням принципів чистої архітектури та предметно-орієнтованого проєктування. У третьому розділі описано інфраструктуру розгортання на базі Docker і хмарних сервісів Railway та Azure SQL, задокументовано інтерфейс користувача та проведено багаторівневе тестування системи із застосуванням автоматизованих тестів і мануальної верифікації API. Результатом роботи є повнофункціональна вебплатформа, що забезпечує прозорий цикл заявки від публікації до підтвердженого завершення, двосторонню комунікацію учасників у реальному часі, систему репутації та адміністративний контроль над усіма процесами платформи.

		Торон М.Д.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		88

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Тороп М.Д., Чижмотря О.В. Атаки типу Prompt Injection на LLM-системи та методи захисту. Тези XVI Міжнародної науково-технічної конференції «Інформаційно-комп'ютерні технології», м. Житомир, 02-03 квітня 2026 р. – Житомир: Житомирська політехніка, 2026. (подано до друку)
2. Тороп М.Д., Чижмотря О.В. Автоматизація тестування вебзастосунків з використанням LLM-агентів. Тези XVI Міжнародної науково-технічної конференції «Інформаційно-комп'ютерні технології», м. Житомир, 02-03 квітня 2026 р. – Житомир: Житомирська політехніка, 2026. (подано до друку)
3. Starbird K., Palen L. Voluntweeters... Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. New York: ACM, 2011. P. 1071–1080. DOI: <https://doi.org/10.1145/1978942.1979087>
4. Currion P. Not the Tool You're Looking For: A Critical Review of Humanitarian Technology. Journal of Humanitarian Logistics and Supply Chain Management. 2015. Vol. 5, No. 2. P. 236–253. DOI: <https://doi.org/10.1108/JHLSCM-03-2014-0015>
5. Spade D. Mutual Aid: Building Solidarity During This Crisis (and the Next). London: Verso Books, 2020. 160 p.
6. Richardson C. Microservices Patterns: With examples in Java. Shelter Island: Manning Publications, 2018. 520 p.
7. Newman S. Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith. Sebastopol: O'Reilly Media, 2019. 272 p.
8. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston: Prentice Hall, 2017. 432 p.
9. Martin R. C. Agile Software Development, Principles, Patterns, and Practices. Upper Saddle River: Prentice Hall, 2002. 529 p.
10. Fowler M. CQRS [Електронний ресурс] / Martin Fowler. – 2011. – URL: <https://martinfowler.com/bliki/CQRS.html> (дата звернення: 15.01.2026).
11. Date C. J. An Introduction to Database Systems. 8th ed. Boston: Addison-Wesley, 2003. 1024 p.

		Тороп М.Д.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмотря О.В.				89
Змн.	Арк.	№ докум.	Підпис	Дата		

- 12.Freeman A. Pro ASP.NET Core 6: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages. 9th ed. New York: Apress, 2022. 1050 p.
- 13.Troelsen A., Japikse P. Pro C# 10 with .NET 6: Foundational Principles and Practices in Programming. 10th ed. New York: Apress, 2022. 1960 p.
- 14.Churcher C. Beginning Database Design: From Novice to Professional. 2nd ed. New York: Apress, 2012. 266 p.
- 15.Banks A., Porcello E. Learning GraphQL: Declarative Data Fetching for Modern Web Apps. Sebastopol: O'Reilly Media, 2018. 198 p.
- 16.Accomazzo A., Murray N., Lerner A. Fullstack React: The Complete Guide to ReactJS and Friends. San Francisco: Fullstack.io, 2017. 836 p.
- 17.Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Boston: Addison-Wesley Professional, 2003. 208 p.
- 18.Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston: Addison-Wesley Professional, 2003. 560 p.
- 19.Scott Millett, Nick Tune. Patterns, Principles, and Practices of Domain-Driven Design. Indianapolis: Wrox, 2015. 794 p.
- 20.Wlaschin S. Domain Modeling Made Functional. Raleigh: Pragmatic Bookshelf, 2018. 320 p.
- 21.Bogard J. MediatR – Simple, unambitious mediator implementation in .NET [Електронний ресурс] / Jimmy Bogard. – URL: <https://github.com/jbogard/MediatR> (дата звернення: 10.01.2026).
- 22.Vernon V. Implementing Domain-Driven Design. Boston: Addison-Wesley Professional, 2013. 656 p.
- 23.Seemann M., van Deursen S. Dependency Injection Principles, Practices, and Patterns. Shelter Island: Manning Publications, 2019. 552 p.
- 24.Stuttard D., Pinto M. The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws. 2nd ed. Indianapolis: Wiley, 2011. 912 p.
- 25.Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol: O'Reilly Media, 2017. 562 p.

		Торон М.Д.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмоторя О.В.				90
Змн.	Арк.	№ докум.	Підпис	Дата		

26.Stefanov S. React: Up & Running. 2nd ed. Sebastopol: O'Reilly Media, 2021. 230 p.

27.Humble J., Farley D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Boston: Addison-Wesley Professional, 2010. 512 p.

28.Kane S. P., Matthias K. Docker: Up & Running. 3rd ed. Sebastopol: O'Reilly Media, 2023. 416 p.

29.Railway. Railway Documentation [Електронний ресурс] / Railway Corp. – URL: <https://docs.railway.app> (дата звернення: 19.01.2026).

30.Redmond E., Wilson J. Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement. Raleigh: Pragmatic Bookshelf, 2012. 354 p.

31.Turnbull J. The Docker Book: Containerization Is the New Virtualization. New York: James Turnbull, 2014. 261 p.

32.Agafonkin V. Leaflet – an open-source JavaScript library for mobile-friendly interactive maps [Електронний ресурс] / Vladimir Agafonkin. – URL: <https://leafletjs.com/reference.html> (дата звернення: 23.03.2026).

33.Forsgren N., Humble J., Kim G. Accelerate: The Science of Lean Software and DevOps. Portland: IT Revolution Press, 2018. 156 p.

34.Fink G., Flatow I. Pro Single Page Application Development. New York: Apress, 2014. 300 p.

35.Despoudis T. TypeScript 4 Design Patterns and Best Practices. Birmingham: Packt Publishing, 2021. 350 p.

36.Oshero R. The Art of Unit Testing: with Examples in .NET. 2nd ed. Shelter Island: Manning Publications, 2013. 296 p.

37.Fowler M. Test Pyramid [Електронний ресурс] / Martin Fowler. – 2012. – URL: <https://martinfowler.com/bliki/TestPyramid.html> (дата звернення: 20.01.2026).

38.Meszaros G. xUnit Test Patterns: Refactoring Test Code. Boston: Addison-Wesley, 2007. 833 p.

		Торон М.Д.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чишмоторя О.В.				91
Змн.	Арк.	№ докум.	Підпис	Дата		

39.Freeman S., Pryce N. Growing Object-Oriented Software, Guided by Tests. Boston: Addison-Wesley Professional, 2009. 384 p.

40.Kim G., Behr K., Spafford G. The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win. Portland : IT Revolution Press, 2013. 345 p.

41.Nygard M. T. Release It!: Design and Deploy Production-Ready Software. 2nd ed. Raleigh: Pragmatic Bookshelf, 2018. 376 p.

42.Richardson C., Smith F. Microservices: From Design to Deployment. Boston: NGINX Inc., 2016. 74 p.

43.Feathers M. Working Effectively with Legacy Code. Upper Saddle River : Prentice Hall, 2004. 456 p.

		Торон М.Д.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмотря О.В.				92
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

		Торон М.Д.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмотря О.В.				93
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістинг доменного класу HelpRequest

```

using server.Domain.Exceptions;
namespace server.Domain.HelpRequest
{
    public enum HelpRequestStatus
    {
        Moderation = 0,
        Open = 1,
        InProgress = 2,
        Resolved = 3,
        Cancelled = 4,
        Rejected = 5
    }

    public class HelpRequest
    {
        private readonly List<RequestImage> _images = new();
        private readonly List<HelpRequestResponse> _responses = new();
        public IReadOnlyCollection<HelpRequestResponse> Responses =>
            _responses.AsReadOnly();
        public Guid? AssignedUserId { get; private set; }
        public Guid Id { get; private set; }
        public string Title { get; private set; }
        public string Description { get; private set; }
        public HelpRequestStatus Status { get; private set; }
        public Guid CreatorId { get; private set; }
        public HelpRequestGeoPoint? Location { get; private set; }

        public DateTime CreatedAtUtc { get; private set; }
        public IReadOnlyCollection<RequestImage> Images => _images.AsReadOnly();
        public DateTime? UpdatedAtUtc { get; private set; }
        public bool IsDeleted { get; private set; }
        public string? CancellationReason { get; private set; }
        public Guid? LastAssignedUserId { get; private set; }
        private HelpRequest() { }
        public HelpRequest(
            Guid creatorId,
            string title,
            string description,
            HelpRequestGeoPoint? location = null)
        {
            Id = Guid.NewGuid();
            CreatorId = creatorId;
            SetTitle(title);
            SetDescription(description);
            Location = location;
            Status = HelpRequestStatus.Moderation;
            CreatedAtUtc = DateTime.UtcNow;
        }
        internal HelpRequest(

```

		Торон М.Д.			ПІЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				94
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    Guid id,
    Guid creatorId,
    string title,
    string description,
    int status,
    Guid? assignedUserId,
    Guid? lastAssignedUserId,
    double? latitude,
    double? longitude,
    DateTime createdAtUtc,
    DateTime? updatedAtUtc,
    bool isDeleted,
    string? cancellationReason,
    IEnumerable<HelpRequestResponse> responses)
{
    Id = id;
    CreatorId = creatorId;
    Title = title;
    Description = description;
    Status = (HelpRequestStatus)status;
    AssignedUserId = assignedUserId;
    LastAssignedUserId = lastAssignedUserId;
    CreatedAtUtc = createdAtUtc;
    UpdatedAtUtc = updatedAtUtc;
    IsDeleted = isDeleted;
    CancellationReason = cancellationReason;
    if (latitude.HasValue && longitude.HasValue)
        Location = new HelpRequestGeoPoint(latitude.Value, longitude.Value);
    _responses.AddRange(responses);
}
private void SetTitle(string title)
{
    if (string.IsNullOrEmpty(title))
        throw new DomainException("Title is empty","HelpRequest.TITLE_REQUIRED");
    if (title.Length > 200)
        throw new DomainException("Title is too long",
"HelpRequest.TITLE_TOO_LONG");
    Title = title;
}
private void SetDescription(string description)
{
    if (string.IsNullOrEmpty(description))
        throw new DomainException("Description is empty",
"HelpRequest.DESCRPTION_REQUIRED");
    if (description.Length > 4000)
        throw new DomainException("Title is too long",
"HelpRequest.DESCRPTION_TOO_LONG");

    Description = description;
}
public void AddImage(string url)
{

```

		Торон М.Д.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		95

```

        if (Status != HelpRequestStatus.Open && Status != HelpRequestStatus.Moderation)
            throw new DomainException("Wrong help request status for adding image",
"HelpRequest.CANNOT_ADD_IMAGE_IN_CURRENT_STATUS");
        if (_images.Count >= 5)
            throw new DomainException("Too many images to add",
"HelpRequest.IMAGE_LIMIT_EXCEEDED");
        _images.Add(new RequestImage(_images.Count, url));
    }
    public void RemoveImage(int order)
    {
        var image = _images.FirstOrDefault(i => i.Order == order);
        if (image == null)
            throw new DomainException("Image doesnt exist or missing",
"HelpRequest.IMAGE_NOT_FOUND");
        _images.Remove(image);
        for (int i = 0; i < _images.Count; i++)
            _images[i] = new RequestImage(i, _images[i].ImageUrl);
    }

    public void UpdateImages(IEnumerable<string> imageUrls)
    {
        if (Status != HelpRequestStatus.Open)
            throw new DomainException("Wrong help request status for updating images",
"HelpRequest.CANNOT_UPDATE_IMAGES_IN_CURRENT_STATUS");
        _images.Clear();
        int order = 0;
        foreach (var url in imageUrls.Take(5))
        {
            _images.Add(new RequestImage(order++, url));
        }
    }
    internal void MarkInProgress()
    {
        if (Status == HelpRequestStatus.Open || Status == HelpRequestStatus.Resolved)
        {
            Status = HelpRequestStatus.InProgress;
        }
        else
        {
            throw new DomainException("Wrong status for transitioning",
"HelpRequest.INVALID_STATUS_TRANSITION_IN_PROGRESS");
        }
    }
    public void Complete()
    {
        if (Status != HelpRequestStatus.InProgress)
            throw new DomainException("Wrong status",
"HelpRequest.INVALID_STATUS_TRANSITION_COMPLETE");
        Status = HelpRequestStatus.Resolved;
    }
    public void Cancel()
    {

```

```

        if (Status is HelpRequestStatus.Resolved or HelpRequestStatus.Cancelled)
            throw new DomainException("Wrong status",
                "HelpRequest.INVALID_STATUS_TRANSITION_CANCEL");
        Status = HelpRequestStatus.Cancelled;
        foreach (var r in _responses)
        {
            r.Reject();
        }
    }
    public bool HasExecutor => AssignedUserId.HasValue;
    public HelpRequestResponse AddResponse(Guid userId, string message)
    {
        if (CreatorId == userId)
            throw new DomainException("Cannot respond to own request",
                "HelpRequest.SELF_RESPONSE");
        if (Status != HelpRequestStatus.Open)
            throw new DomainException("Cannot respond to inactive request",
                "HelpRequest.NOT_OPEN");
        if (string.IsNullOrEmpty(message))
            throw new DomainException("Message is required",
                "HelpRequestResponse.MESSAGE_REQUIRED");
        if (message.Length > 1000)
            throw new DomainException("Message is too long",
                "HelpRequestResponse.MESSAGE_TOO_LONG");
        if (_responses.Any(r => r.UserId == userId && r.Status !=
            HelpRequestResponseStatus.Cancelled))
            throw new DomainException("User already responded",
                "HelpRequest.ALREADY_RESPONDED");
        var response = new HelpRequestResponse(userId, message);
        _responses.Add(response);
        return response;
    }
    public void CancelResponse(Guid userId)
    {
        if (Status != HelpRequestStatus.Open)
            throw new DomainException(
                "Cannot cancel response on inactive request",
                "HelpRequest.NOT_OPEN");

        var response = _responses
            .FirstOrDefault(r => r.UserId == userId);
        if (response is null)
            throw new DomainException("Response not found",
                "HelpRequestResponse.NOT_FOUND");
        response.Cancel();
    }
    public void AssignExecutor(Guid responseId)
    {
        if (Status != HelpRequestStatus.Open)
            throw new DomainException("Request is not open", "HelpRequest.NOT_OPEN");
        var response = _responses
            .FirstOrDefault(r => r.Id == responseId);

```

```

        if (response is null)
            throw new DomainException("Response not found",
"HelpRequestResponse.NOT_FOUND");
        if (response.Status != HelpRequestResponseStatus.Pending)
            throw new DomainException("Invalid response state",
"HelpRequestResponse.INVALID_STATE");
        response.Accept();
        foreach (var r in _responses.Where(r => r.Id != responseId))
        {
            r.Reject();
        }
        AssignedUserId = response.UserId;
        LastAssignedUserId = response.UserId;
        Status = HelpRequestStatus.InProgress;
    }

    public void Edit(
        string title,
        string description,
        double? latitude,
        double? longitude)
    {
        if (Status != HelpRequestStatus.Open)
            throw new DomainException(
                "Cannot edit current request",
                "HelpRequest.CANNOT_EDIT");
        SetTitle(title);
        SetDescription(description);
        Location = (latitude.HasValue && longitude.HasValue)
            ? new HelpRequestGeoPoint(latitude.Value, longitude.Value)
            : null;
        Status = HelpRequestStatus.Moderation;
        UpdatedAtUtc = DateTime.UtcNow;
    }

    public void Delete()
    {
        if (Status != HelpRequestStatus.Open && Status != HelpRequestStatus.Moderation)
            throw new DomainException(
                "Cannot delete inactive request",
                "HelpRequest.CANNOT_DELETE");
        if (IsDeleted)
            throw new DomainException(
                "Request is already deleted",
                "HelpRequest.ALREADY_DELETED");
        IsDeleted = true;
    }

    public void Cancel(string reason)
    {
        if (Status is HelpRequestStatus.Resolved or HelpRequestStatus.Cancelled)
            throw new DomainException(
                "Wrong status",
                "HelpRequest.INVALID_STATUS_TRANSITION_CANCEL");
    }

```

		Торон М.Д.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмоторя О.В.				98
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        if (string.IsNullOrEmpty(reason))
            throw new DomainException(
                "Cancellation reason is required",
                "HelpRequest.CANCELLATION_REASON_REQUIRED");
        if (reason.Length > 500)
            throw new DomainException(
                "Cancellation reason is too long",
                "HelpRequest.CANCELLATION_REASON_TOO_LONG");
        Status = HelpRequestStatus.Cancelled;
        CancellationReason = reason;
        AssignedUserId = null;
        foreach (var r in _responses)
            r.Reject();
    }
    public void Restore()
    {
        if (Status != HelpRequestStatus.Cancelled)
            throw new DomainException(
                "Only cancelled requests can be restored",
                "HelpRequest.CANNOT_RESTORE");

        Status = HelpRequestStatus.Moderation;
        CancellationReason = null;
    }
    public void ResignAsExecutor(Guid userId)
    {
        if (Status != HelpRequestStatus.InProgress)
            throw new DomainException(
                "Cannot resign from inactive request",
                "HelpRequest.NOT_IN_PROGRESS");

        if (AssignedUserId != userId)
            throw new DomainException(
                "You are not the assigned executor",
                "HelpRequest.NOT_EXECUTOR");
        var response = _responses.FirstOrDefault(r => r.UserId == userId && r.Status ==
HelpRequestResponseStatus.Accepted);
        response?.Reject();
        Status = HelpRequestStatus.Open;
        AssignedUserId = null;
    }
    public void RemoveExecutor(Guid userId, string reason)
    {
        if (Status != HelpRequestStatus.InProgress)
            throw new DomainException(
                "Cannot remove executor from inactive request",
                "HelpRequest.NOT_IN_PROGRESS");
        if (CreatorId != userId)
            throw new DomainException(
                "Only owner can remove executor",
                "HelpRequest.NOT_OWNER");
        if (string.IsNullOrEmpty(reason))

```

```

        throw new DomainException(
            "Reason is required",
            "HelpRequest.REASON_REQUIRED");
    if (reason.Length > 500)
        throw new DomainException(
            "Reason is too long",
            "HelpRequest.REASON_TOO_LONG");

    if (AssignedUserId.HasValue)
    {
        var response = _responses.FirstOrDefault(r => r.UserId == AssignedUserId.Value &&
r.Status == HelpRequestResponseStatus.Accepted);
        response?.Reject();
    }
    Status = HelpRequestStatus.Open;
    AssignedUserId = null;
}
public void Approve()
{
    if (Status != HelpRequestStatus.Moderation)
        throw new DomainException("Only requests in moderation can be approved",
"HelpRequest.NOT_IN_MODERATION");

    Status = HelpRequestStatus.Open;
    UpdatedAtUtc = DateTime.UtcNow;
}
public void RejectModeration(string reason)
{
    if (Status != HelpRequestStatus.Moderation)
        throw new DomainException("Only requests in moderation can be rejected",
"HelpRequest.NOT_IN_MODERATION");

    if (string.IsNullOrEmpty(reason))
        throw new DomainException("Rejection reason is required",
"HelpRequest.REASON_REQUIRED");
    Status = HelpRequestStatus.Rejected;
    CancellationReason = $"Moderation rejected: {reason}";
    UpdatedAtUtc = DateTime.UtcNow;
}

```

Діаграма варіантів використання вебплатформи «Ethy»



Рис. Б.1. Діаграма варіантів використання вебплатформи «Ethy»

ER-діаграма бази даних вебплатформи “Ethy”

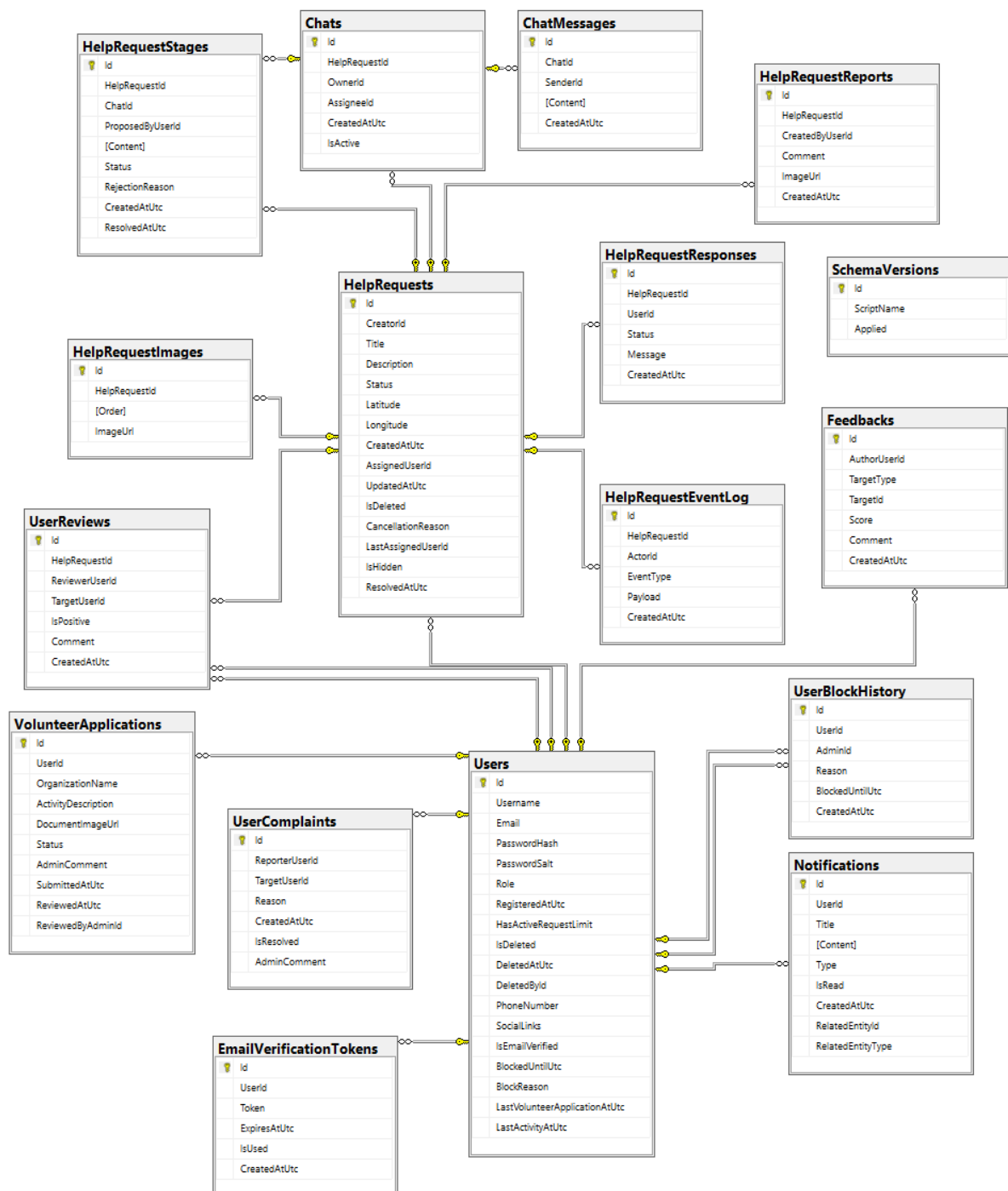


Рис. В.1. ER-діаграма бази даних вебплатформи “Ethy”

Діаграма активності алгоритму створення та модерації заявки

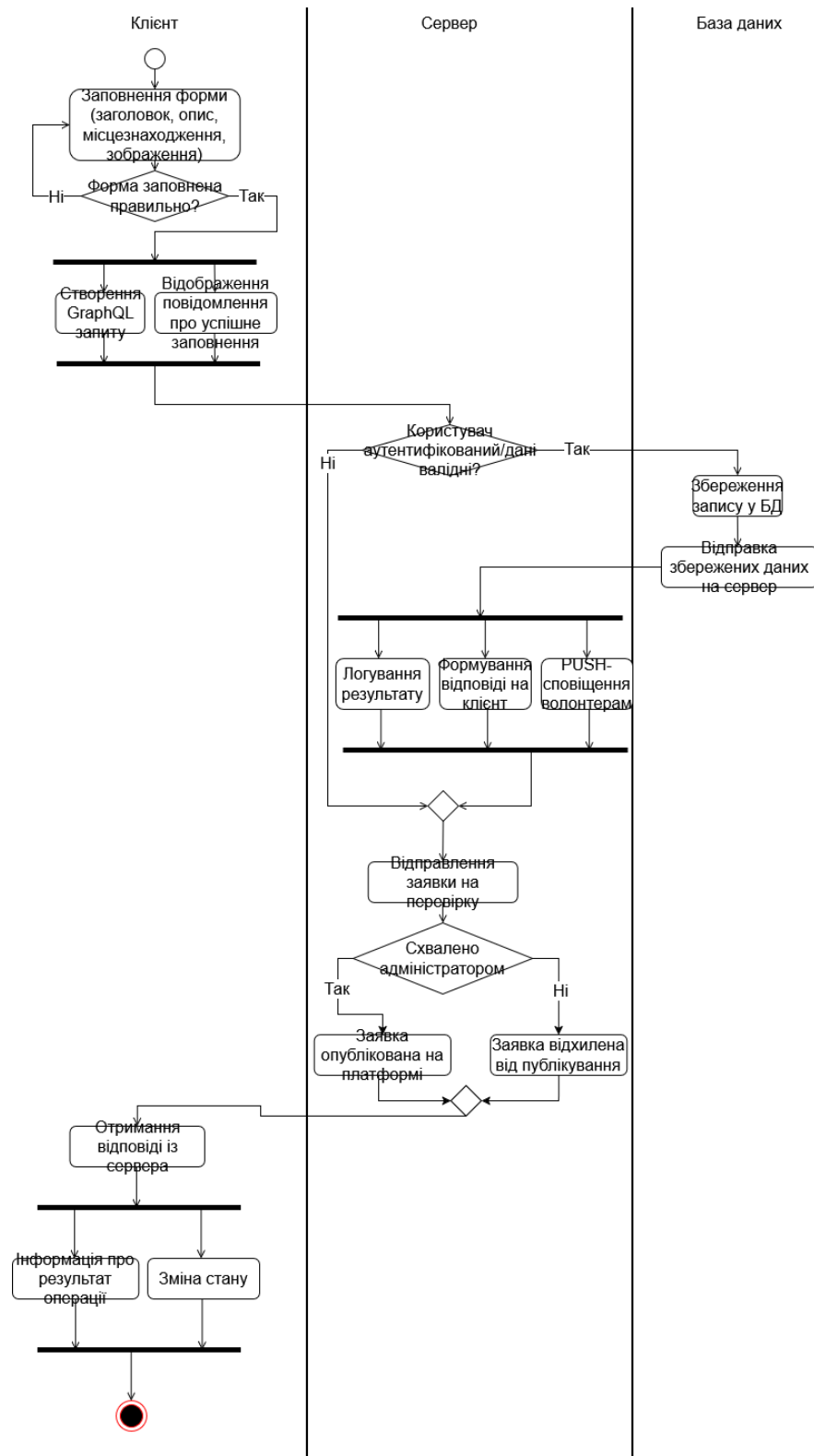


Рис. Г.1. Діаграма активності алгоритму створення та модерації заявки

Лістинг CI/CD пайплайну

```

name: .NET Tests
on:
  push:
    branches: [ "main", "dev" ]
  pull_request:
    branches: [ "main", "dev" ]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Setup .NET 10
        uses: actions/setup-dotnet@v4
        with:
          dotnet-version: 10.0.x
      - name: Restore dependencies
        run: dotnet restore project-ethy.slnx
      - name: Build
        run: dotnet build project-ethy.slnx --no-restore
      - name: Test Unit Tests
        run: dotnet test tests/server.UnitTests/server.UnitTests.csproj --no-build --verbosity normal
      - name: Test Integration Tests
        run: dotnet test tests/server.IntegrationTests/server.IntegrationTests.csproj --no-build --
        verbosity normal

```

Лістинг CI/CD пайплайну (конфігурація інтеграції):

```

name: CI/CD Pipeline

```

```

on:
  push:
    branches: ["**"]
  workflow_dispatch:

```

```

#

```

```

=====
# STAGE ORDER: build → test → docker → deploy
# Кожен наступний stage стартує тільки після успіху попереднього
#
=====

```

```

jobs:
#

```

```

=====
# STAGE: build
#
=====

```

```

build-server:
  name: Build – .NET Server
  runs-on: ubuntu-latest

```

```

  steps:
    - name: Checkout code

```

		Торон М.Д.			ПІЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				104
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    uses: actions/checkout@v4
  - name: Cache NuGet packages
    uses: actions/cache@v4
    with:
      path: ~/.nuget/packages
      key: nuget-${ { hashFiles('**/server/*.csproj') } }
      restore-keys: |
        nuget-
  - name: Setup .NET
    uses: actions/setup-dotnet@v4
    with:
      dotnet-version: "10.0.x"
  - name: Restore dependencies
    run: dotnet restore server/server.csproj
  - name: Build
    run: dotnet build server/server.csproj --no-restore --configuration Release
  - name: Upload build artifact
    uses: actions/upload-artifact@v4
    with:
      name: server-build
      path: server/bin/Release/
      retention-days: 1
build-client:
  name: Build – React Client
  runs-on: ubuntu-latest
  steps:
    - name: Checkout code
      uses: actions/checkout@v4
    - name: Cache node_modules
      uses: actions/cache@v4
      with:
        path: client/node_modules
        key: npm-${ { hashFiles('client/package-lock.json') } }
        restore-keys: |
          npm-
    - name: Setup Node.js
      uses: actions/setup-node@v4
      with:
        node-version: "22"
    - name: Install dependencies
      working-directory: client
      run: npm ci
    - name: Build client
      working-directory: client
      run: npm run build
    - name: Upload build artifact
      uses: actions/upload-artifact@v4
      with:
        name: client-build
        path: client/dist/
        retention-days: 1

```

#

STAGE: test

Залежить від успішного build

#

test-server:

name: Test – .NET Server

runs-on: ubuntu-latest

needs: build-server

steps:

- name: Checkout code

uses: actions/checkout@v4

- name: Cache NuGet packages

uses: actions/cache@v4

with:

path: ~/.nuget/packages

key: nuget-\${{ hashFiles('**/server/*.csproj') }}

restore-keys: |

nuget-

- name: Setup .NET

uses: actions/setup-dotnet@v4

with:

dotnet-version: "10.0.x"

- name: Restore dependencies

run: dotnet restore server/server.csproj

- name: Verify build (no tests yet)

run: |

dotnet build server/server.csproj --no-restore --configuration Release

echo "✓ Server build verified – no test project yet"

- name: Generate build report

run: |

mkdir -p build-report

echo "Server build report" > build-report/server-info.txt

echo "Date: \$(date)" >> build-report/server-info.txt

echo "Commit: \${GITHUB_SHA}" >> build-report/server-info.txt

echo "Branch: \${GITHUB_REF_NAME}" >> build-report/server-info.txt

echo "Status: PASSED" >> build-report/server-info.txt

- name: Upload build report artifact

uses: actions/upload-artifact@v4

with:

name: build-report

path: build-report/

retention-days: 7

test-client:

name: Test – React Client (lint)

runs-on: ubuntu-latest

needs: build-client

steps:

- name: Checkout code

uses: actions/checkout@v4

		Торон М.Д.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		106

```

- name: Cache node_modules
  uses: actions/cache@v4
  with:
    path: client/node_modules
    key: npm-${{ hashFiles('client/package-lock.json') }}
    restore-keys: |
      npm-
- name: Setup Node.js
  uses: actions/setup-node@v4
  with:
    node-version: "22"

- name: Install dependencies
  working-directory: client
  run: npm ci
- name: Run lint
  working-directory: client
  run: npm run lint

```

#

```

# STAGE: docker
# Збірка Docker образів – залежить від test stage
#

```

```

docker-server:
  name: Docker – Build Server Image
  runs-on: ubuntu-latest
  needs: test-server
  if: github.ref == 'refs/heads/main' || github.ref == 'refs/heads/dev'
  steps:
    - name: Checkout code
      uses: actions/checkout@v4
    - name: Build server Docker image
      run: |
        docker build \
          --file server/Dockerfile \
          --tag ethy-server:${{ github.sha }} \
          --tag ethy-server:latest \
          server/
    - name: Verify image
      run: |
        docker images ethy-server
        echo "✓ Server image built successfully"

```

```

docker-client:
  name: Docker – Build Client Image
  runs-on: ubuntu-latest
  needs: test-client
  if: github.ref == 'refs/heads/main' || github.ref == 'refs/heads/dev'
  steps:
    - name: Checkout code
      uses: actions/checkout@v4
    - name: Download client build artifact

```

		Торон М.Д.			ПІЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				107
Змн.	Арк.	№ докум.	Підпис	Дата		

```

uses: actions/download-artifact@v4
with:
  name: client-build
  path: client/dist/
- name: Build client Docker image
run: |
  docker build \
    --file client/Dockerfile \
    --tag ethy-client:${{ github.sha }} \
    --tag ethy-client:latest \
    client/
- name: Verify image
run: |
  docker images ethy-client
  echo "✓ Client image built successfully"

```

#

```

# STAGE: deploy
# Імітація розгортання – залежить від docker stage
#

```

```

deploy:
  name: Deploy – Generate Manifest
  runs-on: ubuntu-latest
  needs: [docker-server, docker-client]
  if: github.ref == 'refs/heads/main' || github.ref == 'refs/heads/dev'
  steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Download build report
      uses: actions/download-artifact@v4
      with:
        name: build-report
        path: build-report/
    - name: Generate deployment manifest
      run: |
        mkdir -p deployment
        echo "===== " > deployment/manifest.txt
        echo " ETHY PLATFORM – DEPLOYMENT MANIFEST " >>
        deployment/manifest.txt
        echo "===== " >>
        deployment/manifest.txt
        echo "" >> deployment/manifest.txt
        echo "Deployment date : $(date)" >> deployment/manifest.txt
        echo "Commit SHA    : ${ github.sha }" >> deployment/manifest.txt
        echo "Branch       : ${ github.ref_name }" >> deployment/manifest.txt
        echo "Triggered by  : ${ github.actor }" >> deployment/manifest.txt
        echo "" >> deployment/manifest.txt
        echo "--- Images ---" >> deployment/manifest.txt
        echo "ethy-server : ${ github.sha }" >> deployment/manifest.txt

```

		Торон М.Д.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмоторя О.В.				108
Змн.	Арк.	№ докум.	Підпис	Дата		

```

echo "ethy-client : ${ { github.sha } }" >> deployment/manifest.txt
echo "" >> deployment/manifest.txt
echo "--- Build report ---" >> deployment/manifest.txt
cat build-report/server-info.txt >> deployment/manifest.txt
echo "" >> deployment/manifest.txt
echo "Status: READY FOR DEPLOYMENT" >> deployment/manifest.txt
- name: Print manifest
  run: cat deployment/manifest.txt
# Зберігаємо manifest як artifact для звіту
- name: Upload deployment manifest
  uses: actions/upload-artifact@v4
  with:
    name: deployment-manifest
    path: deployment/
    retention-days: 7

```

		Торон М.Д.			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		109