

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ
СІКОРСЬКОГО»**

**Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки**

«На правах рукопису»
УДК 004.77

ДО ЗАХИСТУ ДОПУЩЕНО:
В.О. Завідувача кафедри
_____ Михайло НОВОТАРСЬКИЙ
«___» _____ 2025 р.

**Магістерська дисертація
на здобуття ступеня магістра
за освітньою-професійною програмою «Інженерія програмного
забезпечення» зі спеціальності 121 «Інженерія програмного забезпечення
комп'ютерних систем»
на тему: «Система підтримки волонтерської діяльності»**

Виконав:

Студент II курсу, групи ІМ-41мп
БАРГАМІН Максим Максимович _____

Керівник:

доцент, к.т.н.
ПОРЄВ Віктор Миколайович _____

Консультант з нормоконтролю:

проф., д.т.н.
КУЛАКОВ Юрій Олексійович _____

Рецензент:

доцент, к.т.н.
ШИМКОВИЧ Володимир Миколайович _____

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

Студент _____

Київ – 2025

Факультет/ННІ інформатики та обчислювальної техніки
(повна назва)

Кафедра обчислювальної техніки
(повна назва)

Рівень вищої освіти – другий (магістерський)

Спеціальність 121 «Інженерія програмного забезпечення»
(код і назва)

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних систем»

ЗАТВЕРДЖУЮ:

В.О. Завідувача кафедри

_____ Михайло НОВОТАРСЬКИЙ

«___» _____ 2025 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Баргамін Максиму Максимовичу

1. Тема дисертації «Система підтримки волонтерської діяльності»
науковий керівник дисертації Порєв Віктор Миколайович, доцент, кандидат технічних наук
затверджені наказом 4841-с по університету від «06» листопада 2025 р. дисертації
2. Об'єкт дослідження система координації та взаємодії між учасниками волонтерської діяльності в Україні
3. Предмет дослідження (Вихідні дані – для магістерської дисертації за ОПП/ОНП) моделі, методи та інформаційні технології для розробки уніфікованої цифрової платформи для підтримки волонтерського руху.
4. Перелік завдань, які потрібно розробити:
 1. Аналітичний огляд сучасного стану волонтерського руху в Україні та існуючих цифрових рішень.
 2. Проектування архітектури та функціоналу системи підтримки волонтерської діяльності.
 3. Програмна реалізація ключових компонентів системи.

4. Тестування та верифікація розробленої системи.

5. Розробка стартап-проєкту.

6. Консультанти розділів проєкту:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	проф. Кулаков Ю.О.		

7. Дата видачі завдання 01.09.2025 року

Календарний план

№ з/п	Найменування етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
	Затвердження теми проєкту	01.09.25 - 07.09.25	
	Вивчення та аналіз завдання	22.09.25 - 28.09.25	
	Вивчення та аналіз існуючих	29.09.25 - 05.10.25	
	Опис та обґрунтування рішення	06.10.25 - 10.10.25	
	Розробка алгоритму маршрутизації	10.10.25 - 19.10.25	
	Програмна реалізація запропонованого способу	10.11.25 - 13.11.25	
	Розробка стартап-проєкту	14.11.25 - 18.11.25	
	Оформлення магістерської	19.11.25 - 24.11.25	

Студент

Науковий керівник

(підпис)

(підпис)

Максим БАРГАМІН

Віктор ПОРЄВ

Реферат
на магістерську дисертацію
виконану на тему: Система підтримки волонтерської діяльності
студентом: Баргаміном Максимом Максимовичем

Робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг основного тексту становить 117 аркушів, робота містить 16 ілюстрацій, 38 таблиць. При підготовці використано літературу з 32 джерел.

Об'єкт дослідження – процес координації та взаємодії між учасниками волонтерської діяльності в Україні.

Предмет дослідження – моделі, методи та інформаційні технології для розробки уніфікованої цифрової платформи для підтримки волонтерського руху.

Мета роботи – підвищення ефективності волонтерської діяльності в Україні шляхом розробки та обґрунтування архітектури уніфікованої цифрової платформи, що забезпечує ефективну координацію між волонтерами, реципієнтами допомоги та профільними організаціями.

Актуальність теми. Повномасштабне вторгнення 2022 року спричинило безпрецедентний сплеск волонтерського руху в Україні, до якого, за даними соціологічних досліджень, долучалося до 74% населення [1]. Однак стрімка позитивна динаміка виявила низку системних проблем: діяльність залишається переважно хаотичною та децентралізованою, бракує ефективної координації, що призводить до дублювання зусиль та нераціонального розподілу ресурсів. Учасники стикаються з інформаційним вакуумом, логістичними труднощами, юридичними перешкодами та високим ризиком емоційного вигорання. В цих умовах розробка централізованої цифрової платформи є актуальним та ефективним інструментом для систематизації та оптимізації волонтерської діяльності в країні.

Наукова новизна одержаних результатів полягає у наступному:

1. Запропоновано комплексну мікросервісну архітектуру для цифрової платформи координації волонтерської діяльності, яка, на відміну від існуючих монолітних рішень, забезпечує високу масштабованість, відмовостійкість та гнучкість у додаванні нового функціоналу.
2. Розроблено унікальну модель взаємодії, що поєднує гнучку систему ролей (індивідуальний волонтер, організація, реципієнт), багатфакторну систему фільтрації запитів та пропозицій, а також інтегрований комунікаційний модуль. Таке поєднання дозволяє створити єдиний інформаційний простір для всіх учасників та закрити нішу універсальних волонтерських платформ в Україні.

Практична цінність роботи полягає в розробці готового до впровадження програмного продукту — вебзастосунку, який дозволяє:

- волонтерам — швидко знаходити актуальні запити на допомогу відповідно до своїх навичок та ресурсів;
- реципієнтам — створювати верифіковані запити та оперативно отримувати необхідну допомогу;
- волонтерським організаціям — ефективно керувати проєктами, координувати команди та залучати нових учасників волонтерського руху.

Розроблена платформа має значну соціальну цінність, сприяючи підвищенню прозорості, ефективності та керованості волонтерського руху в Україні. Також в рамках роботи розроблено бізнес-модель стартап-проєкту, що обґрунтовує шляхи його сталого розвитку та монетизації.

Ключові слова: волонтерство, цифрова платформа, координація, соціальний проєкт, мікросервісна архітектура, Spring Boot, PostgreSQL, стартап-проєкт.

ABSTRACT

The work consists of an introduction, four chapters, conclusions, a list of references, and appendices. The total volume of the main text is 117 pages, containing 16 figures and 38 tables. The preparation involved the use of literature from 32 sources.

The object of the study is the process of coordination and interaction between participants in volunteer activities in Ukraine.

The subject of the study is models, methods, and information technologies for the development of a unified digital platform to support the volunteer movement.

The purpose of the work is to increase the efficiency of volunteer activities in Ukraine by developing and substantiating the architecture of a unified digital platform that ensures effective coordination between volunteers, aid recipients, and specialized organizations.

The relevance of the topic. The full-scale invasion of 2022 caused an unprecedented surge in the volunteer movement in Ukraine, in which, according to sociological research, up to 74% of the population was involved. However, this explosive growth revealed a number of systemic problems: activities remain mostly chaotic and decentralized, there is a lack of effective coordination, which leads to duplication of effort and irrational distribution of resources. Participants face an information vacuum, logistical difficulties, legal obstacles, and a substantial risk of emotional burnout. In these conditions, the development of a centralized digital platform is a relevant and effective tool for systematizing and optimizing volunteer activities in the country.

The scientific novelty of the obtained results is as follows:

1. Comprehensive microservice architecture for a digital platform for coordinating volunteer activities has been proposed, which, unlike existing monolithic solutions, provides high scalability, fault tolerance, and flexibility in adding new functionality.
2. A unique interaction model has been developed that combines a flexible role system (individual volunteer, organization, recipient), a multi-factor system for filtering requests and proposals, and an integrated communication module. This combination allows for the creation of a single information space for all participants and fills the

niche of universal volunteer platforms in Ukraine.

The practical significance of the work lies in the development of a ready-to-implement software product—a web application that allows:

- volunteers – to quickly find relevant requests for help according to their skills and resources.
- recipients – to submit verified requests and promptly receive the necessary assistance.
- volunteer organizations – to effectively manage projects, coordinate teams, and attract new members. The developed platform has significant social value, contributing to increasing the transparency, efficiency, and manageability of the volunteer movement in Ukraine. Also, within the framework of the work, a business model for a startup project has been developed, which substantiates the ways of its sustainable development and monetization.

Keywords: volunteering, digital platform, coordination, social project, microservice architecture, Spring Boot, PostgreSQL, startup project

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, І СКОРОЧЕНЬ ТЕРМІНІВ.....	10
ВСТУП.....	11
РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД СУЧАСНОГО СТАНУ, ВИКЛИКІВ ТА ЦИФРОВИХ ІНСТРУМЕНТІВ ПІДТРИМКИ ВОЛОНТЕРСЬКОГО РУХУ В УКРАЇНІ.....	14
1.1 Феномен волонтерства в умовах повномасштабного вторгнення: кількісний та якісний аналіз.....	14
1.2 Системні виклики та бар'єри для сталого розвитку волонтерської діяльності.....	17
1.3 Аналіз існуючих цифрових платформ та інфраструктурних рішень для підтримки волонтерства.....	20
1.4 Синтез та обґрунтування ніші для розробки інноваційного рішення.....	24
Висновки до розділу 1.....	26
РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ.....	27
2.1 Проєктування функціоналу.....	27
2.1.1 Функціональні та нефункціональні вимоги.....	27
2.1.2 Актори системи та їхні ролі.....	30
2.1.3 Діаграма варіантів використання та їх опис.....	32
2.2 Проєктування архітектури.....	37
2.2.1 Обґрунтування вибору мікросервісної архітектури.....	38
2.2.2 Загальна архітектурна схема та опис компонентів.....	39
2.2.3 Обґрунтування вибору технологічного стеку.....	42
2.3 Проєктування бази даних.....	43
2.3.1 Логічна модель даних та опис сутностей.....	43
2.3.2 ER-діаграма та опис зв'язків.....	48
Висновки до розділу 2.....	50

РОЗДІЛ 3. РОЗРОБКА ТА РЕАЛІЗАЦІЯ СИСТЕМИ.....	51
3.1 Вибір інструментів розробки.....	51
3.2 Розробка бази даних.....	53
3.3 Реалізація мікросервісів.....	59
3.4 Реалізація веб-інтерфейсу користувача.....	71
3.5 Контейнеризація та розгортання.....	82
Висновки до розділу 3.....	84
РОЗДІЛ 4. АНАЛІЗ ГОТОВНОСТІ, ТЕСТУВАННЯ ТА ЗАХИСТУ СИСТЕМИ VOLUNTEER SUPPORT.....	85
4.1 Загальна характеристика рівня готовності.....	85
4.2 Модульне тестування доменної логіки.....	86
4.3 Інтеграційні тести між сервісами та інфраструктурою.....	88
4.4 Безпека автентифікації та авторизації.....	89
4.5 Захист даних та ізоляція компонентів.....	92
4.6 Захист від типових векторів атак.....	93
4.7 Контейнеризація та середовище виконання.....	93
4.8 Висновки до розділу 4	95
РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ.....	97
5.1 Опис ідеї стартап-проєкту.....	97
5.2 Технологічний аудит ідеї проєкту.....	100
5.3 Аналіз ринкових можливостей запуску стартап-проєкту.....	102
5.4 Розроблення ринкової стратегії проєкту.....	105
5.5 Розроблення маркетингової програми стартап-проєкту.....	107
5.6 Економічний аналіз та оцінка ризиків.....	109
Висновки до розділу 5.....	112
ВИСНОВКИ.....	113
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	114

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (Application Programming Interface) – прикладний програмний інтерфейс

БД – База даних

ДПС – Державна податкова служба

ДСТУ – Державний стандарт України

ЄДРПОУ – Єдиний державний реєстр підприємств та організацій України

JWT (JSON Web Token) – стандарт токенів доступу

НКО – Некомерційна організація

REST (Representational State Transfer) – архітектурний стиль взаємодії компонентів [30]

SPA (Single Page Application) – односторінковий застосунок

SQL (Structured Query Language) – мова структурованих запитів

SSR (Server-Side Rendering) – серверний рендеринг

СУБД – Система управління базами даних

УВС – Українська Волонтерська Служба

DDL (Data Definition Language) – мова визначення даних

DDD (Domain-Driven Design) – предметно-орієнтоване проєктування

ER (Entity-Relationship) – сутність-зв'язок (тип діаграми)

ВСТУП

Актуальність теми. Сучасний етап розвитку українського суспільства характеризується безпрецедентним зростанням ролі громадянського суспільства та волонтерського руху. В умовах екзистенційних викликів волонтерство трансформувалося із ситуативної благодійності на потужний соціальний інститут, що забезпечує життєдіяльність держави. Проте, незважаючи на значні обсяги допомоги та масове залучення громадян, організаційна структура цього руху часто залишається фрагментарною, ситуативною та залежною від особистих контактів, що створює ризики неефективності, «вигорання» волонтерів та зловживань.

Існуючі дослідження носять переважно фрагментарний характер, зосереджуючись на окремих кейсах або історичних аспектах, тоді як питання створення єдиного інформаційного простору для волонтерів залишається відкритим. Необхідність переходу від хаотичної самоорганізації до системного управління людськими, матеріальними та інформаційними ресурсами зумовлює актуальність розробки спеціалізованих інформаційних систем. В умовах війни інформація стає критичним ресурсом, а її відсутність або спотворення може мати фатальні наслідки. Тому розробка уніфікованої платформи координації волонтерської діяльності є важливим науково-практичним завданням.

Зв'язок роботи з науковими програмами, планами, темами. Магістерська дисертація виконана відповідно до плану науково-дослідних робіт кафедри обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

Мета і завдання дослідження. Метою роботи є підвищення ефективності волонтерської діяльності в Україні шляхом розробки та обґрунтування архітектури уніфікованої цифрової платформи, що забезпечує ефективну координацію між волонтерами, реципієнтами допомоги та профільними організаціями.

Для досягнення поставленої мети необхідно вирішити такі завдання: – провести аналітичний огляд сучасного стану волонтерського руху в Україні та існуючих цифрових рішень;

– спроектувати архітектуру та функціонал системи підтримки волонтерської діяльності;

– виконати програмну реалізацію ключових компонентів системи;

– провести тестування та верифікацію розробленої системи;

– розробити стартап-проект для впровадження системи.

Об'єкт дослідження це процес координації та взаємодії між учасниками волонтерської діяльності в Україні .

Предметом дослідження є моделі, методи та інформаційні технології для розробки уніфікованої цифрової платформи для підтримки волонтерського руху.

Методи дослідження. У роботі використано методи: системного аналізу для визначення вимог до системи та аналізу предметної області; об'єктно-орієнтованого проєктування для розробки архітектури програмного забезпечення; моделювання баз даних для проєктування структури зберігання даних; емпіричні методи – для тестування та верифікації розробленої системи.

Наукова новизна одержаних результатів полягає у наступному:

– запропоновано комплексну мікросервісну архітектуру для цифрової платформи координації волонтерської діяльності, яка, на відміну від існуючих монолітних рішень, забезпечує високу масштабованість, відмовостійкість та гнучкість ;

– розроблено унікальну модель взаємодії, що поєднує гнучку систему ролей (індивідуальний волонтер, організація, реципієнт), багатофакторну систему фільтрації запитів та інтегрований комунікаційний модуль, що дозволяє створити єдиний інформаційний простір для всіх учасників .

Практичне значення одержаних результатів полягає в розробці готового до впровадження програмного продукту – вебзастосунку, який дозволяє волонтерам

швидко знаходити актуальні запити, реципієнтам – створювати верифіковані запити, а організаціям – ефективно керувати проєктами. Розроблена платформа сприяє підвищенню прозорості та керованості волонтерського руху в Україні . Також розроблено бізнес-модель стартап-проєкту, що обґрунтовує шляхи його сталого розвитку.

Розділ 1

Аналітичний огляд сучасного стану, викликів та цифрових інструментів підтримки волонтерського руху в Україні

Даний розділ присвячено комплексному аналізу волонтерського руху в Україні як значущого елемента громадянського суспільства після повномасштабного вторгнення Російської Федерації у 2022 році. У розділі досліджуються кількісні та якісні параметри руху, ідентифікуються системні виклики, що впливають на його сталість, проводиться оцінка наявних інфраструктурних рішень для підтримки волонтерів та обґрунтовується необхідність дослідження нових, інтегрованих інструментів для підвищення ефективності волонтерської діяльності.

1.1. Феномен волонтерства в умовах повномасштабного вторгнення: кількісний та якісний аналіз

Повномасштабне вторгнення 24 лютого 2022 року стало чинником значної мобілізації українського громадянського суспільства, в якій волонтерський рух посів центральне місце. Цей рух трансформувався з нішевої громадської активності на загальнонаціональне явище, ставши значущим ресурсом для держави у реагуванні на гуманітарні та військові виклики. Аналіз цього феномену вимагає розгляду як його кількісних масштабів, так і якісних структурних та демографічних характеристик.=

1.1.1. Кількісні масштаби та динаміка

Масштаби залучення громадян до волонтерської діяльності після лютого 2022 року є значними. Соціологічні дослідження показують, що 74 % українців долучалися до волонтерства у тій чи іншій формі [2], причому 26 % робили це на постійній основі, а 36 % – декілька разів. Цей сплеск громадської активності знайшов відображення і в офіційній статистиці. Протягом 2022 року кількість організацій та установ, що

залучали волонтерів, зросла майже на 80 % (+522 організації), що стало найбільшим показником приросту за весь період спостережень [3].

Особливо показовою є динаміка зростання кількості офіційно зареєстрованих волонтерів у Реєстрі волонтерів, який веде Державна податкова служба (ДПС). Якщо до 2022 року щорічно реєструвалося в середньому 40 осіб, то з квітня 2022 року щомісячна кількість реєстрацій перевищувала цей показник. Загалом, від початку повномасштабної війни до грудня 2023 року кількість офіційних волонтерів зросла у 21 раз, досягнувши 6734 особи.

Тенденція до формалізації волонтерської діяльності продовжилася і у 2024 році. Станом на кінець листопада 2024 року в реєстрі було зареєстровано 10 454 волонтери, що в 1,5 раза більше, ніж на початку року. Середньомісячний приріст у 2024 році становив близько 310 осіб [4], що свідчить про сталий процес офіційної реєстрації, як показано на рис. 1.1.

Ця динаміка відображає два паралельні процеси: масову мобілізацію громадян у відповідь на кризу та поступовий процес інституціоналізації їхньої діяльності. Основною причиною для офіційної реєстрації є необхідність правового та фінансового захисту: реєстрація в Реєстрі ДПС дозволяє волонтерам не включати зібрані на особисті картки пожертви до щорічного оподаткованого доходу, що в іншому випадку призвело б до нарахування 18 % податку на доходи фізичних осіб та 1,5 % військового збору [5].

1.1.2. Структура волонтерського руху: офіційний та неофіційний сектори

Характерною рисою українського волонтерства є значна перевага неофіційного сектору над офіційним. Дані показують, що на початку повномасштабного вторгнення кількість неофіційних волонтерів у 5,6 разів перевищувала кількість

офіційно зареєстрованих. До червня-листопада 2022 року співвідношення сягнуло від 20 до майже 30 неофіційних волонтерів на одного офіційно зареєстрованого [6].

Ця диспропорція свідчить про проблему правової незахищеності. З одного боку, держава та суспільство значною мірою покладаються на роботу сотень тисяч громадян, які діють поза офіційною статистикою. З іншого боку, відсутність реєстрації створює ризики для самих волонтерів. Оскільки головною причиною для реєстрації є уникнення оподаткування пожертв, десятки тисяч незареєстрованих волонтерів, які збирають кошти на особисті банківські картки, потенційно перебувають під загрозою податкових зобов'язань. Це може стати суттєвим фінансовим тягарем і демотивуючим фактором. Таким чином, виникає ситуація, коли найбільш чисельна та гнучка частина волонтерського руху є водночас найбільш вразливою, що створює системний ризик для стабільності руху.

1.1.3. Демографічний зріз: залученість молоді та соціальний портрет волонтера

Волонтерський рух в Україні охоплює різні соціально-демографічні групи. Дослідження показують, що до волонтерства більшою мірою долучаються мешканці західних та центральних областей, а також люди з вищим рівнем матеріального забезпечення та ті, хто ідентифікує українську мову як рідну та використовує її у повсякденному спілкуванні [7].

Особливо значущою є роль молоді. Частка молодих людей (віком 14-35 років), що займаються волонтерством, зросла більш ніж удвічі: з 20 % до повномасштабного вторгнення до 42 % у 2022 році. При цьому 30 % молоді вперше долучилися до волонтерської діяльності саме у 2022 році, порівняно з лише 6 % новачків у 2021 році[8]. Дослідження, проведене громадською організацією «Українська Волонтерська Служба» (УВС) та U-Report серед 4621 молодих людей, підтверджує високий рівень залученості: 26 % респондентів волонтерять на постійній основі, 36 % – періодично, і 12 % мали одноразовий досвід[9].

Мотивація молодих волонтерів, зокрема школярів, має глибоке соціальне коріння. Головним рушієм для них є бажання допомогти людям, які цього потребують (65 %), позитивно вплинути на свою громаду (27,4 %), а також розвинути соціальні навички, такі як емпатія та комунікація (28,8 %)[10].

Такий сплеск молодіжного волонтерства є важливим формуючим досвідом, який свідчить про формування нової громадянської ідентичності у молодого покоління українців. Це явище може мати довгострокові наслідки для соціального капіталу країни. Статистика демонструє не просто кількісне зростання, а структурну зміну: діяльність, що раніше була характерною для меншості, стала звичною для значної частини молоді. Мотивація, що базується на емпатії та відповідальності за громаду, свідчить про формування стійких ціннісних орієнтирів. Згідно з соціологічними теоріями, досвід, отриманий у молодому віці, закладає основи для поведінкових моделей на все життя. Отже, ця мобілізація може інтегрувати просоціальні та громадянсько-орієнтовані цінності в покоління, яке братиме участь у повоєнній відбудові України. Це може створити значний резервуар соціального капіталу, що стане важливим активом на роки вперед.

1.2. Системні виклики та бар'єри для сталого розвитку волонтерської діяльності

Попри масштаби та суспільну значущість, волонтерський рух в Україні функціонує в умовах значного навантаження та стикається з низкою системних викликів, які впливають на його довгострокову стабільність та ефективність. Ці виклики мають як внутрішній (психологічний), так і зовнішній (структурний) характер.

1.2.1. Психологічне вигорання як поширена проблема

Однією з найбільш поширених загроз для волонтерів є синдром емоційного вигорання. Це стан емоційного, фізичного та ментального виснаження, спричинений тривалим стресом, який характеризується втратою енергії, цинізмом та зниженням відчуття особистих досягнень. Поширеність цієї проблеми є значною: згідно з опитуваннями, 59 % волонтерів відчувають, що їхній ресурс для допомоги іншим зменшується або закінчується, а ще 16 % зазначають, що вже не мають такого ресурсу[11].

Вигорання є процесом, що розвивається у кілька стадій: від початкового ентузіазму («медовий місяць») до стадії кризи та повного виснаження, на якій, за оцінками експертів, зараз перебуває більшість українських волонтерів. Його симптомами є хронічна втома, проблеми зі сном, дратівливість та соматичні прояви.

Причини вигорання є комплексними. Вони включають роботу в режимі «нон-стоп», ігнорування власних базових потреб, а також психологічний тиск від постійного контакту з людським горем. Одним із ключових зовнішніх факторів, який, за словами самих волонтерів, виснажує, є зростаюча байдужість суспільства. Фраза «Люди звикли до війни. Запити є, а відгуку – все менше» стала поширеною серед багатьох активістів[12].

Емоційне вигорання волонтерів слід розглядати не лише як індивідуальну проблему. Це є системною проблемою, що впливає на стійкість волонтерського руху. Волонтери виконують важливі функції, які держава не завжди може повністю забезпечити. Враховуючи, що до 75 % волонтерів відчувають виснаження ресурсів, рух перебуває під загрозою кадрових втрат. Вигорання призводить до того, що досвідчені активісти можуть залишати діяльність. Це створює ситуацію, коли нові люди можуть долучатися, але накопичені знання та досвід втрачаються. Таким чином, ігнорування проблеми вигорання на системному рівні може призвести до поступової втрати людського капіталу в одному з ключових секторів громадянського суспільства.

1.2.2. Бюрократичні та юридичні перешкоди

Діяльність волонтерів ускладнюється структурними бар'єрами. Волонтерські ініціативи та громадські організації стикаються зі складними процедурами реєстрації, отримання дозволів та фінансової звітності, що вимагає витрачання значного часу.

Навіть для відносно простих завдань, як-от збір коштів на автомобіль для військових, волонтери змушені готувати велику кількість документів[13]. Це не лише ускладнює процес, а й створює додаткове навантаження. До цих проблем додаються ризики під час перевезення вантажів та потреба у кваліфікованому юридичному супроводі, який можуть собі дозволити переважно великі фонди.² Як зазначалося раніше, невизначений юридичний статус незареєстрованих волонтерів та пов'язані з цим податкові ризики створюють правову непевність.

1.2.3. Проблема шахрайства та криза довіри

Сталість волонтерського руху залежить від рівня суспільної довіри. Ця довіра зазнає ударів через діяльність шахраїв. Зростання кількості шахрайських ініціатив є одним з факторів, що знижує активність донорів. Схеми можуть включати збір коштів на неіснуючі потреби, завищення вартості товарів або привласнення пожертв[14].

Проблема полягає в тому, що гучні випадки шахрайства негативно впливають на репутацію всього волонтерського руху, що призводить до зниження рівня довіри до всіх благодійних ініціатив. Чесні волонтери відчують це через зменшення пожертв та посилення скептицизму.

Шахрайство та бюрократія перебувають у циклічному процесі з негативними наслідками, що шкодить добросамовісним волонтерам, як це зображено на рис. 1.2. Кожен викритий випадок шахрайства³ призводить до падіння суспільної довіри. У відповідь на це держава та донори можуть вимагати посилення контролю, що виливається у

створення нових, складніших бюрократичних процедур. Ці нові вимоги лягають тягарем на плечі справжніх волонтерів, уповільнюючи надання допомоги. Водночас професійні шахраї можуть бути більш вправними у навігації цими процедурами. У результаті виникає система, в якій ціну довіри сплачують найбільш вмотивовані учасники руху, що робить їхню роботу складнішою.

1.3. Аналіз існуючих цифрових платформ та інфраструктурних рішень для підтримки волонтерства

Для підвищення ефективності волонтерської діяльності в Україні та світі створено низку цифрових та фізичних інфраструктур. Порівняльний аналіз цих рішень, наведений в табл. 1.1 [23], дозволяє визначити їхні сильні та слабкі сторони в контексті українських реалій.

1.3.1. Міжнародний досвід: Модель VolunteerMatch/Idealist

Платформа VolunteerMatch, яка об'єдналася з Idealist, є одним із найбільших у світі цифрових ринків для волонтерства . Вона об'єднує понад 130 000 неприбуткових організацій (НКО) з мільйонами потенційних волонтерів[15]. Основними характеристиками платформи є:

- функціонал для волонтерів, що включає складні фільтри пошуку можливостей за місцем розташування, сферою діяльності, навичками та форматом, а також можливість створювати профілі, відстежувати години та залишати відгуки ⁴;
- інструменти для організацій, що дозволяють публікувати вакансії, управляти волонтерами та проводити аналітику[16].

Модель VolunteerMatch/Idealist орієнтована на потреби стабільного громадянського суспільства, де волонтерство є структурованою діяльністю. Платформа добре

справляється із завданням збалансування попиту та пропозиції в передбачуваному середовищі. Однак її модель не повністю відповідає кризовому контексту воєнної України, оскільки їй бракує інструментів для швидкого гуманітарного реагування або навігації у воєнній бюрократії.

1.3.2. Українські цифрові екосистеми: «Волонтерська Платформа» та «єДопомога»

В Україні функціонують власні цифрові рішення, адаптовані до місцевих потреб.

«Волонтерська Платформа» (volunteer.country)

Ця платформа є екосистемою, створеною УВС у співпраці з ЮНІСЕФ та Міністерством молоді та спорту України[17]. Основними характеристиками платформи є:

- функціонал, що включає онлайн-портал для агрегації можливостей, чат-бот для організацій та мережу локальних каналів;
- наявність освітнього компонента з онлайн-курсами та статтями для початківців[18].
- інтегрованість в український контекст, що робить її важливою точкою входу у волонтерство для молоді.⁷

Водночас платформа зосереджена переважно на пошуку можливостей та освіти; їй може бракувати інтегрованих інструментів для вирішення проблем вигорання, юридичної навігації або надійних механізмів верифікації, окрім стандартної модерації.⁸

«єДопомога» (edopomoga.gov.ua)

Це державна цифрова платформа під егідою Міністерства соціальної політики, яка має на меті об'єднати зусилля держави, гуманітарних організацій та волонтерів.⁹ Вона працює за трьома напрямками: державна грошова допомога, виплати від міжнародних організацій та волонтерська допомога[19]. Основними характеристиками платформи є:

- функціонал волонтерського компоненту, що працює за принципом «людина-людині» (P2P), де особа, що потребує допомоги, залишає заявку на конкретні товари, а благодійник може її виконати[20];
- інтеграція з державними сервісами, що надає платформі високий рівень довіри.

Слабкою стороною є те, що це інструмент для надання прямої матеріальної допомоги, який не вирішує організаційних, психологічних чи юридичних проблем самих волонтерів.

1.3.3. Роль фізичних просторів: Мережа «Vcentri HUB»

На противагу цифровим рішенням, у Києві розвивається мережа муніципальних просторів Vcentri HUB, що є майданчиками для комунікації між владою та громадою[21].

Основними характеристиками мережі є:

- модель фізичної інфраструктури, що підтримує громадську активність, надаючи безкоштовні коворкінги та приміщення для заходів;
- надання на базі хабів безкоштовних юридичних та психологічних консультацій.

Модель Vcentri HUB демонструє, що виключно цифрові рішення можуть бути недостатніми. Аналіз визначив психологічне вигорання та бюрократичні перешкоди як ключові проблеми. Цифрові платформи є здебільшого транзакційними і не завжди сприяють формуванню спільнот. Натомість Vcentri HUB закриває ці прогалини, пропонуючи простір для взаємопідтримки.¹¹ Надання очних консультацій психологів та юристів є прямим інструментом протидії стресорам[22]. Отже, оптимальна система підтримки може бути гібридною моделлю, яка інтегрує ефективність цифрової платформи з підтримкою мережі фізичних хабів.

Порівняльний аналіз функціональності існуючих платформ для волонтерів

Критерій	VolunteerMatch/Idealist (Міжнародний)	Волонтерська Платформа (Україна)	єДопомога (Україна, Державна)	Vcentri HUB (Україна, Гібридна)
Тип платформи	Цифровий ринок (marketplace)	Цифрова екосистема	Цифрова P2P- платформа прямої допомоги	Фізична мережа з цифровим компонентом (портал)
Основна мета	Пошук та підбір волонтерів за навичками та інтересами	Агрегація волонтерських можливостей, освіта	Задоволення нагальних матеріальних потреб	Створення простору для комунікації та підтримки громадянського суспільства
Цільова аудиторія	Волонтери, НКО, соціально- відповідальний бізнес	Волонтери (особливо молодь), громадські та благодійні організації	Люди, що потребують допомоги; благодійники («єДопоможці»)	Інститути громадянського суспільства (ІГС), активні громадяни, представники влади та бізнесу
Інструменти для волонтерів	Розширений пошук, профіль, трекінг годин, відгуки	Пошук можливостей, профіль, облік годин, сертифікати	Пошук та вибір заявки на допомогу	Доступ до безкоштовних заходів, коворкінгу, консультацій
Інструменти для організацій	Публікація вакансій, інструменти менеджменту, аналітика	Створення сторінки організації, публікація можливостей	Відсутні (не є платформою для НКО)	Безкоштовне бронювання приміщень для заходів, промо- підтримка
Підтримка психічного здоров'я	Відсутня (не є фокусом)	Відсутня (не є фокусом)	Відсутня	Ключова функція (безкоштовні консультації психологів)
Юридична/бюрократична підтримка	Відсутня	Відсутня (окрім загальних порад)	Відсутня	Ключова функція (безкоштовні юридичні консультації)
Механізми протидії шахрайству	Верифікація НКО, система відгуків	Модерація можливостей	Реєстрація через державні сервіси	Непряма (через побудову довіри в спільноті)
Освітній компонент	Ресурси для НКО	Сильний компонент (курси, статті для початківців)	Відсутній	Освітні заходи, тренінги, семінари
Ключова слабкість/Прогалина	Не адаптована до кризового контексту України	Немає інтегрованих інструментів проти вигорання та бюрократії	Вузька спрямованість (лише матеріальна допомога)	Обмежена географія (Київ), не є платформою для онлайн-координації

1.4. Синтез та обґрунтування ніші для розробки інноваційного рішення

Проведений аналіз дозволяє зробити низку узагальнюючих висновків щодо стану, викликів та інфраструктурної підтримки волонтерського руху в Україні, а також окреслити нішу для подальших досліджень.

1.4.1. Синтез ключових висновків

Волонтерський рух в Україні є значним соціальним явищем, що демонструє високий рівень мобілізації (1.1). Однак він функціонує в умовах навантаження і зазнає тиску з боку психологічного вигорання, бюрократичних перешкод та кризи довіри (1.2). Існуючі інфраструктурні рішення пропонують цінні інструменти, але не вирішують ці проблеми в інтегрований спосіб (1.3). Цифрові платформи ефективні для залучення волонтерів, але їм може бракувати механізмів підтримки. Фізичні хаби надають таку підтримку, проте їм бракує масштабованості.

1.4.2. Ідентифікація незадоволених потреб та «білих плям»

На основі проведеного аналізу можна припустити, що одна з прогалин в існуючій екосистемі полягає у відсутності цілісної платформи, розробленої не лише для **мобілізації** волонтерів, а й для **підтримки** їхнього потенціалу. Ключовими відсутніми компонентами можуть бути:

- інтегровані інструменти підтримки психічного здоров'я: ресурси для боротьби з вигоранням, вбудовані у функціонал платформи (наприклад, доступ до психологів, групи взаємопідтримки);
- спрощення бюрократичних процесів: інструменти, що допомагають волонтерам орієнтуватися в питаннях реєстрації та звітності (наприклад, шаблони документів, база знань);
- надійні механізми верифікації: багаторівнева система перевірки для протидії шахрайству.

1.4.3. Обґрунтування актуальності та науково-практичної значущості дипломного проєкту

Таким чином, синтез проведеного аналізу підводить до обґрунтування актуальності та науково-практичної значущості даного дипломного проєкту. Метою дослідження є розробка та обґрунтування моделі цифрової платформи, яка б інтегрувала інструменти для залучення волонтерів із механізмами їхньої психологічної, юридичної та адміністративної підтримки. Запропонована робота спрямована на заповнення чітко ідентифікованої, на основі аналізу даних, прогалини в існуючій інфраструктурі. Її наукова та практична цінність полягає у формулюванні вимог, проєктуванні архітектури та визначенні функціоналу системи, здатної підвищити стійкість та ефективність волонтерського руху в Україні

Висновки до розділу 1

У першому розділі проведено комплексний аналіз предметної області, що дозволило зробити такі висновки:

1. Волонтерський рух в Україні після 2022 року набув значних масштабів, залучивши суттєву частину населення, зокрема молодь. Спостерігається стійка тенденція до формалізації діяльності, що підтверджується багатократним зростанням кількості офіційно зареєстрованих волонтерів.
2. Ідентифіковано ключові системні проблеми, що впливають на сталість руху: високий рівень психологічного вигорання активістів, наявність значних бюрократичних та юридичних перешкод, а також зниження суспільної довіри через діяльність шахраїв. Ці фактори створюють кумулятивний негативний ефект, виснажуючи людський капітал руху.
3. Аналіз існуючих цифрових та інфраструктурних рішень (VolunteerMatch, «Волонтерська Платформа», «Допомога», Vcentri HUB) показав, що вони є ефективними у вирішенні вузькоспеціалізованих завдань, таких як пошук волонтерських можливостей або надання прямої матеріальної допомоги. Однак жодне з існуючих рішень не пропонує інтегрованого підходу до одночасного вирішення проблем вигорання, юридичної незахищеності та координації.

Отже, аналіз предметної області виявив незаповнену нішу для універсальної цифрової платформи, яка б поєднувала функціонал для мобілізації волонтерів з інструментами для підтримки їхньої діяльності. Обґрунтована необхідність у такій системі є підставою для переходу до наступного етапу дослідження – проєктування архітектури та функціоналу такої платформи, що буде детально розглянуто в другому розділі дисертації.

РОЗДІЛ 2

ПРОЄКТУВАННЯ СИСТЕМИ

Цей розділ представляє комплексний технічний проєкт «Системи підтримки волонтерської діяльності». Метою є перетворення аналітичних висновків, зроблених у Розділі 1, у конкретну, обґрунтовану та реалізовану архітектуру програмного забезпечення. Проєктування системи буде проведено у трьох послідовних етапах: визначення функціональних вимог та поведінки системи, розробка високорівневої архітектури та обґрунтування технологічного стеку, і, нарешті, детальне проєктування структури бази даних. Кожне проєктне рішення буде тісно пов'язане з викликами предметної області, такими як необхідність масштабування, забезпечення відмовостійкості та протидія шахрайству, що були ідентифіковані в аналітичному огляді.

2.1. Проєктування функціоналу

На цьому етапі закладається фундамент для всього проєкту, визначаючи, що саме система повинна робити та які характеристики вона повинна мати. Детальний аналіз функціональних та нефункціональних вимог, визначення ключових акторів та опис сценаріїв їх взаємодії з системою дозволяють сформулювати чітке бачення майбутнього продукту.

2.1.1. Функціональні та нефункціональні вимоги

Визначення вимог є фундаментальним етапом, що закладає основу для всіх подальших проєктних рішень. Функціональні вимоги визначають, що система повинна робити, тоді як нефункціональні вимоги визначають, як вона повинна це робити, встановлюючи критерії якості, продуктивності та безпеки. Ці вимоги безпосередньо впливають з проблем, окреслених у Розділі 1, таких як хаотичність, дублювання зусиль, вигорання та шахрайство.

Функціональні вимоги

Функціональні вимоги описують конкретні дії та операції, які користувачі можуть виконувати в системі. Вони є прямою відповіддю на виклики, ідентифіковані в аналітичному розділі. Наприклад, проблема «дублювання зусиль» породжує вимогу до ефективного пошуку та фільтрації існуючих запитів, а ризик «шахрайства» — вимогу до багаторівневої верифікації користувачів та організацій. Систематизований перелік функціональних вимог наведено в табл. 2.1.

Таблиця 2.1

Функціональні вимоги до системи

ID	Назва	Опис	Пріоритет
FR1	Управління профілями	Користувачі всіх ролей (Волонтер, Реципієнт, Організація) повинні мати можливість реєструватися, автентифікуватися, редагувати та видаляти свої профілі.	Високий
FR2	Створення та управління запитам	Реципієнти та Організації повинні мати можливість створювати запити на допомогу, вказуючи категорію, опис, місцезнаходження, терміновість та додаючи підтверджуючі документи.	Високий
FR3	Розширений пошук та фільтрація	Волонтери повинні мати можливість шукати запити за ключовими словами, фільтрувати їх за категорією, регіоном, статусом, терміновістю та рейтингом автора запиту.	Високий
FR4	Верифікація користувачів	Система повинна надавати адміністраторам інструменти для верифікації особи волонтерів та реципієнтів, а також для перевірки юридичного статусу організацій.	Високий
FR5	Система рейтингів та відгуків	Після виконання запиту волонтери та реципієнти повинні мати можливість оцінити один одного та залишити відгук для формування репутації та підвищення довіри в системі.	Високий
FR6	Внутрішня система обміну повідомленнями	Система повинна забезпечувати безпечний канал комунікації між волонтерами та реципієнтами в межах конкретного запиту на допомогу для уточнення деталей.	Середній

Таблиця 2.1 (продовження)

Функціональні вимоги до системи

FR7	Система сповіщень	Користувачі повинні отримувати сповіщення (внутрішні та/або на email) про важливі події: нові відгуки на їхні запити, зміни статусів, повідомлення від адміністрації.	Середній
FR8	Управління членством в організаціях	Організації повинні мати можливість запрошувати волонтерів до своїх команд, а волонтери — приймати або відхиляти ці запрошення.	Середній
FR9	Адміністративна панель	Адміністратори повинні мати доступ до панелі керування для модерації контенту (запитів, відгуків), управління користувачами (верифікація, блокування) та перегляду системної аналітики.	Високий

Нефункціональні вимоги

Нефункціональні вимоги визначають атрибути якості системи та є ключовим фактором, що впливає на вибір архітектури. Для платформи національного масштабу, яка повинна витримувати значні та часто непередбачувані навантаження, такі вимоги, як масштабованість та надійність, є критично важливими. Саме вони обґрунтовують відмову від традиційної монолітної архітектури на користь більш гнучкого та стійкого підходу. Перелік нефункціональних вимог представлено в табл. 2.2.

Таблиця 2.2

Нефункціональні вимоги до системи

ID	Категорія	Вимога	Метрика
NFR1	Продуктивність	Система повинна мати малий час відгуку на дії користувача, особливо під час пікових навантажень.	Час відповіді сервера на 95% ключових запитів (пошук, завантаження списку запитів) не повинен перевищувати 2 секунди при 10,000 одночасних активних користувачів.
NFR2	Масштабованість	Архітектура повинна підтримувати горизонтальне масштабування для обробки зростання	Система повинна дозволяти незалежне масштабування окремих функціональних компонентів (наприклад, сервісу пошуку) без простою всієї платформи.

Таблиця 2.2 (продовження)

Нефункціональні вимоги до системи

NFR3	Надійність та доступність	Система повинна бути доступною для користувачів 24/7 і стійкою до збоїв окремих її частин.	Коефіцієнт доступності системи (uptime) повинен становити не менше 99.9%. Збій одного компонента не повинен призводити до відмови всієї системи (fault isolation).
NFR4	Безпека	Персональні дані користувачів та їх комунікація повинні бути надійно захищені.	Усі дані, що передаються між клієнтом та сервером, повинні шифруватися за допомогою протоколу TLS. Чутливі дані в базі даних (наприклад, паролі) повинні зберігатися у хешованому вигляді.
NFR5	Підтримуваність	Архітектура повинна спрощувати процес внесення змін, виправлення помилок та додавання нового функціоналу.	Кожен функціональний компонент системи має бути незалежно розроблюваним, тестованим та розгортуваним, що дозволяє скоротити час виходу нових версій (time-to-market).
NFR6	Сумісність	Веб-інтерфейс системи повинен коректно відображатися та функціонувати в останніх двох версіях популярних браузерів (Chrome, Firefox, Safari, Edge).	Успішне проходження 98% функціональних тестів на всіх цільових браузерах.

2.1.2. Актори системи та їхні ролі

Система розрахована на взаємодію чотирьох основних типів користувачів (акторів), кожен з яких має унікальний набір прав та обов'язків, що відповідає його ролі у волонтерському процесі:

– Волонтер. Це індивідуальний користувач, який є рушійною силою платформи та прагне надати допомогу. Його основна мотивація — знайти релевантні та перевірені можливості для застосування своїх навичок, часу та ресурсів. Для цього система надає йому інструменти для створення детального профілю, де можна вказати сфери інтересів (допомога тваринам, логістика, психологічна підтримка), наявні ресурси (автомобіль, спеціальне обладнання) та географію діяльності. Ключовою функцією

для волонтера є доступ до розширеного пошуку та фільтрації запитів, що дозволяє швидко знаходити завдання, які відповідають його критеріям. Після знаходження відповідного запиту волонтер може відгукнутися на нього та розпочати безпечну комунікацію з реципієнтом через внутрішню систему повідомлень для уточнення деталей. Для підвищення рівня довіри та власної репутації на платформі, після виконання завдання волонтер може отримати відгук та оцінку від реципієнта. Також система передбачає можливість для волонтера приєднуватися до команд, створених верифікованими організаціями, для участі у більш масштабних та скоординованих проєктах;

– Реципієнт. Це фізична або юридична особа (наприклад, лікарня, притулок для тварин, внутрішньо переміщена особа), яка потребує допомоги. Головна задача реципієнта — максимально просто, швидко та безпечно донести свою потребу до волонтерської спільноти. Система дозволяє реципієнту зареєструватися та створити запит на допомогу, детально описавши суть проблеми, вказавши категорію, місцезнаходження та терміновість. Для боротьби з шахрайством та підвищення довіри до запиту, реципієнт має можливість завантажити підтверджуючі документи або фотографії. Після публікації запиту (який може проходити попередню модерацію) реципієнт отримує відгуки від волонтерів, може переглядати їхні профілі та рейтинги, і на основі цієї інформації обрати найбільш відповідного виконавця. Після того, як допомогу було надано, реципієнт закриває запит та залишає оцінку і відгук волонтеру, що завершує цикл взаємодії та сприяє формуванню прозорості системи репутації на платформі;

– Організація. Це юридично зареєстрована некомерційна організація, благодійний фонд або громадське об'єднання, що виступає як координатор та агрегатор волонтерських зусиль. Ця роль надає розширені можливості для управління масштабною діяльністю. Профіль організації проходить обов'язкову верифікацію з перевіркою юридичних документів, що забезпечує високий рівень довіри до її діяльності на платформі. Організація може створювати власні проєкти та збори, які

можуть об'єднувати декілька менших завдань, а також запрошувати індивідуальних волонтерів до своєї команди, формуючи постійний актив. Система надає інструменти для управління складом команди та призначення конкретних волонтерів на виконання запитів. Важливою функцією для організацій є можливість генерації звітів про свою діяльність, що забезпечує прозорість перед донорами та суспільством;

– Адміністратор. Це користувач з найвищим рівнем доступу, відповідальний за підтримку працездатності, безпеки та порядку на платформі. Роль адміністратора є ключовою для забезпечення довіри та ефективного функціонування всієї екосистеми. До його обов'язків входить управління користувачами, що включає верифікацію профілів реципієнтів та організацій, розгляд скарг від користувачів та блокування акаунтів, що порушують правила платформи. Адміністратор здійснює модерацію контенту, перевіряючи та схвалюючи нові запити на допомогу для уникнення шахрайства, а також видаляючи некоректні відгуки чи коментарі. Крім того, адміністратор управляє системними довідниками (наприклад, категоріями допомоги) та має доступ до панелі з аналітикою, що дозволяє відстежувати ключові показники роботи платформи та приймати рішення щодо її подальшого розвитку.

2.1.3. Діаграма варіантів використання та їх опис

Діаграма варіантів використання (Use-Case Diagram) візуалізує взаємодію акторів із ключовими функціями системи. Вона є важливим інструментом для розуміння загальної функціональності та меж системи. Загальну схему взаємодії акторів та варіантів використання зображено на рис. 2.1.

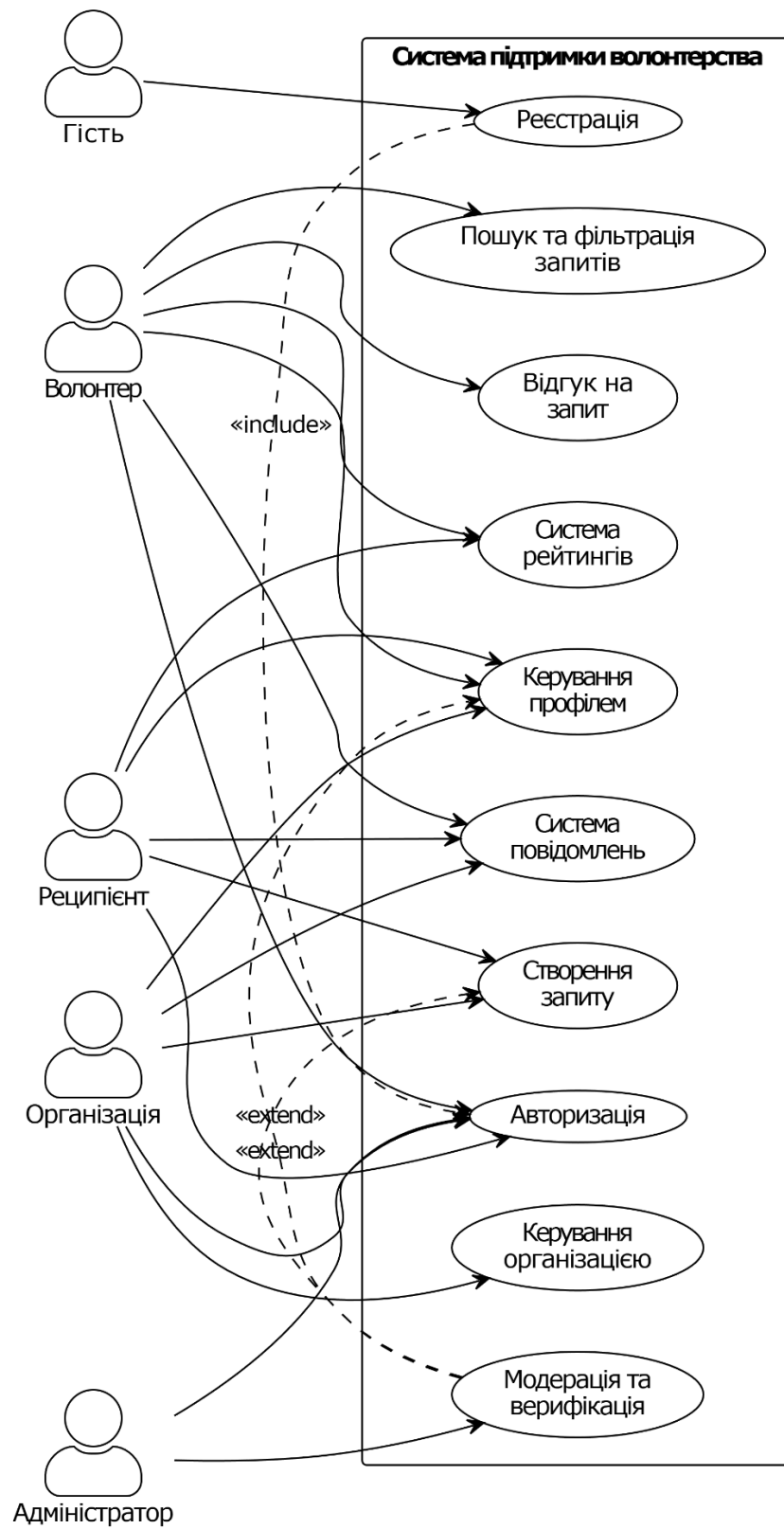


Рис.2.1. – Діаграма варіантів використання

На діаграмі використовуються зв'язки $\diamond$ для позначення обов'язкової функціональності (наприклад, будь-яка дія, що вимагає прав, включає варіант використання «Авторизація») та $\diamond$ для опціональної функціональності (наприклад, варіант використання «Верифікація профілю» розширює «Редагування профілю», але не є обов'язковим на початковому етапі). Для глибшого розуміння логіки системи нижче наведено детальний опис трьох ключових сценаріїв взаємодії.

Сценарій 1: Реєстрація нового волонтера

Назва: Реєстрація нового волонтера.

Актор: Гість (неавторизований користувач).

Передумова: Користувач знаходиться на головній сторінці платформи.

Основний потік:

1. Користувач натискає кнопку «Зареєструватися».
2. Система відображає сторінку вибору ролі.
3. Користувач обирає роль «Волонтер».
4. Система відображає форму реєстрації, що вимагає введення імені, прізвища, адреси електронної пошти та пароля.
5. Користувач заповнює форму та підтверджує свою згоду з правилами платформи.
6. Система виконує валідацію введених даних на стороні сервера (перевіряє унікальність email, складність пароля).
7. У разі успішної валідації, система створює новий обліковий запис у базі даних зі статусом «Очікує підтвердження email».
8. Система генерує унікальне посилання для підтвердження та надсилає його на вказану електронну пошту.

9. Користувач отримує лист, переходить за посиланням.
10. Система верифікує токен з посилання, активує обліковий запис, змінюючи його статус на «Активний», та автоматично авторизує користувача, перенаправляючи його в особистий кабінет.

Альтернативний потік (Email вже існує):

На кроці 6, якщо система виявляє, що введена електронна пошта вже зареєстрована, вона відображає повідомлення про помилку та пропонує користувачеві увійти в систему або відновити пароль.

Сценарій 2: Створення запиту на допомогу Реципієнтом

Назва: Створення запиту на допомогу.

Актор: Реципієнт.

Передумова: Реципієнт успішно авторизований у системі.

Основний потік:

1. Реципієнт переходить до розділу «Мої запити» в особистому кабінеті.
2. Реципієнт натискає кнопку «Створити новий запит».
3. Система відображає багатокрокову форму створення запиту.
4. На першому кроці реципієнт вводить назву запиту, обирає категорію допомоги з випадаючого списку (наприклад, «Гуманітарна допомога», «Транспортні послуги») та вказує рівень терміновості.
5. На другому кроці реципієнт надає детальний опис потреби та вказує точну адресу або регіон, де потрібна допомога.

6. На третьому кроці система пропонує завантажити фотографії або документи, що підтверджують потребу (опціонально, але підвищує довіру).
7. Реципієнт перевіряє введені дані та натискає кнопку «Опублікувати».
8. Система валідує всі поля форми та зберігає запит у базі даних зі статусом «Очікує модерації».
9. Система автоматично надсилає сповіщення адміністраторам про появу нового запиту, що потребує перевірки.
10. Після успішної модерації та схвалення адміністратором, статус запиту змінюється на «Відкритий», і він стає видимим для всіх волонтерів на платформі.

Альтернативний потік (Некоректні дані):

На кроці 8, якщо валідація не пройдена (наприклад, не заповнено обов'язкове поле), система підсвічує поля з помилками та виводить відповідні повідомлення, не зберігаючи запит.

Сценарій 3: Пошук та відгук на запит Волонтером

Назва: Пошук та відгук на запит.

Актор: Волонтер.

Передумова: Волонтер успішно авторизований у системі.

Основний потік:

1. Волонтер переходить на сторінку «Знайти допомогу» (або «Пошук запитів»).
2. Система відображає список усіх активних запитів, відсортованих за датою створення, та панель з інструментами фільтрації.

3. Волонтер використовує фільтри, щоб звужити пошук, обираючи потрібну категорію допомоги, регіон (наприклад, за допомогою інтерактивної карти) та терміновість.
4. Система динамічно оновлює список запитів відповідно до застосованих фільтрів.
5. Волонтер обирає запит, що його зацікавив, та натискає на нього, щоб перейти на сторінку з детальною інформацією.
6. На сторінці запиту волонтер бачить повний опис, фото, місцезнаходження на карті та інформацію про автора запиту (реципієнта).
7. Волонтер натискає кнопку «Відгукнутися» або «Запропонувати допомогу».
8. Система відображає модальне вікно, де волонтер може написати короткий супровідний текст (наприклад, «Маю автомобіль, можу допомогти з перевезенням у вказаний час»).
9. Волонтер підтверджує відправку відгуку.
10. Система зберігає відгук у базі даних, пов'язуючи його з конкретним запитом та волонтером, і надсилає реципієнту сповіщення про нову пропозицію допомоги.

2.2. Проектування архітектури

На цьому етапі визначається загальна структура системи, її основні компоненти та принципи їх взаємодії. Вибір архітектури є стратегічним рішенням, що безпосередньо впливає на здатність системи відповідати нефункціональним вимогам, таким як масштабованість, надійність та гнучкість.

2.2.1. Обґрунтування вибору мікросервісної архітектури

Вибір архітектури є стратегічним рішенням, що визначає довгострокову життєздатність, масштабованість та гнучкість системи. Для проєкту національного масштабу, що має на меті координацію волонтерської діяльності, традиційна монолітна архітектура є ризикованим вибором через її суттєві обмеження. У монолітній системі весь функціонал упакований в єдиний розгортаємий артефакт, що робить її складною для масштабування, оновлення та підтримки. Натомість, мікросервісна архітектура пропонує підхід, що ідеально відповідає нефункціональним вимогам проєкту, розбиваючи складну систему на набір невеликих, незалежних та слабо зв'язаних сервісів [24; 25; 30].

Однією з найвагоміших переваг є масштабованість. На відміну від моноліту, де для обробки підвищеного навантаження необхідно масштабувати весь застосунок, мікросервіси дозволяють масштабувати лише ті компоненти, що зазнають найбільшого навантаження. Наприклад, під час гуманітарної кризи або інформаційної кампанії очікується сплеск активності у створенні та пошуку запитів. У цьому випадку можна горизонтально масштабувати лише request-service та search-service, запустивши додаткові їх екземпляри, тоді як identity-service або notification-service можуть продовжувати працювати з мінімальними ресурсами. Такий підхід забезпечує надзвичайно ефективне використання обчислювальних ресурсів та підтримує високу продуктивність системи під непередбачуваними навантаженнями.

Другим важливим аспектом є надійність та відмовостійкість. У мікросервісній архітектурі збій одного сервісу не призводить до відмови всієї системи. Якщо, наприклад, notification-service вийде з ладу через проблеми з поштовим сервером, користувачі все одно зможуть реєструватися, створювати запити та шукати допомогу. Основний функціонал платформи залишиться доступним. Ця ізоляція збоїв є критично важливою для системи, від якої залежить надання своєчасної допомоги людям. У монолітній архітектурі помилка в некритичному модулі може призвести до зупинки всього застосунку.

Третя перевага полягає у гнучкості та швидкості розробки. Система розбивається на невеликі, незалежні сервіси, кожен з яких відповідає за конкретну бізнес-логіку. Це дозволяє організувати роботу невеликих автономних команд, кожна з яких може зосередитись на розробці, тестуванні та розгортанні свого сервісу. Такий підхід значно прискорює цикли розробки та розгортання (CI/CD). Новий функціонал, наприклад, модуль для збору аналітики або інтеграція з картографічними сервісами, може бути розроблений та впроваджений як новий мікросервіс без необхідності перебудовувати та повторно розгорнути всю систему.

Нарешті, мікросервісний підхід забезпечує технологічну гетерогенність. Кожен мікросервіс може бути реалізований з використанням найбільш відповідного для його завдань технологічного стеку. Хоча основним стеком для даного проєкту обрано Java та Spring Boot, у майбутньому для специфічних завдань можна буде використати інші технології. Наприклад, search-service може бути оптимізований за допомогою Elasticsearch, а потенційний сервіс для аналізу даних та виявлення шахрайських патернів може бути реалізований на Python з використанням бібліотек машинного навчання. Така гнучкість дозволяє обирати найкращий інструмент для кожної конкретної задачі без впливу на решту системи.

Таким чином, вибір мікросервісної архітектури є не просто слідуванням сучасним трендам, а обґрунтованим інженерним рішенням, спрямованим на управління ризиками та забезпечення довгострокового розвитку проєкту. Цей підхід дозволяє створити систему, яка є одночасно потужною, стійкою до збоїв, гнучкою до змін та ефективною у використанні ресурсів, що повністю відповідає вимогам до платформи національного рівня.

2.2.2. Загальна архітектурна схема та опис компонентів

Загальна архітектура системи, зображена на рис. 2.2, побудована на принципах мікросервісного підходу. Вона складається з кількох ключових компонентів, що

взаємодіють між собою через чітко визначені інтерфейси (API) та асинхронні повідомлення:

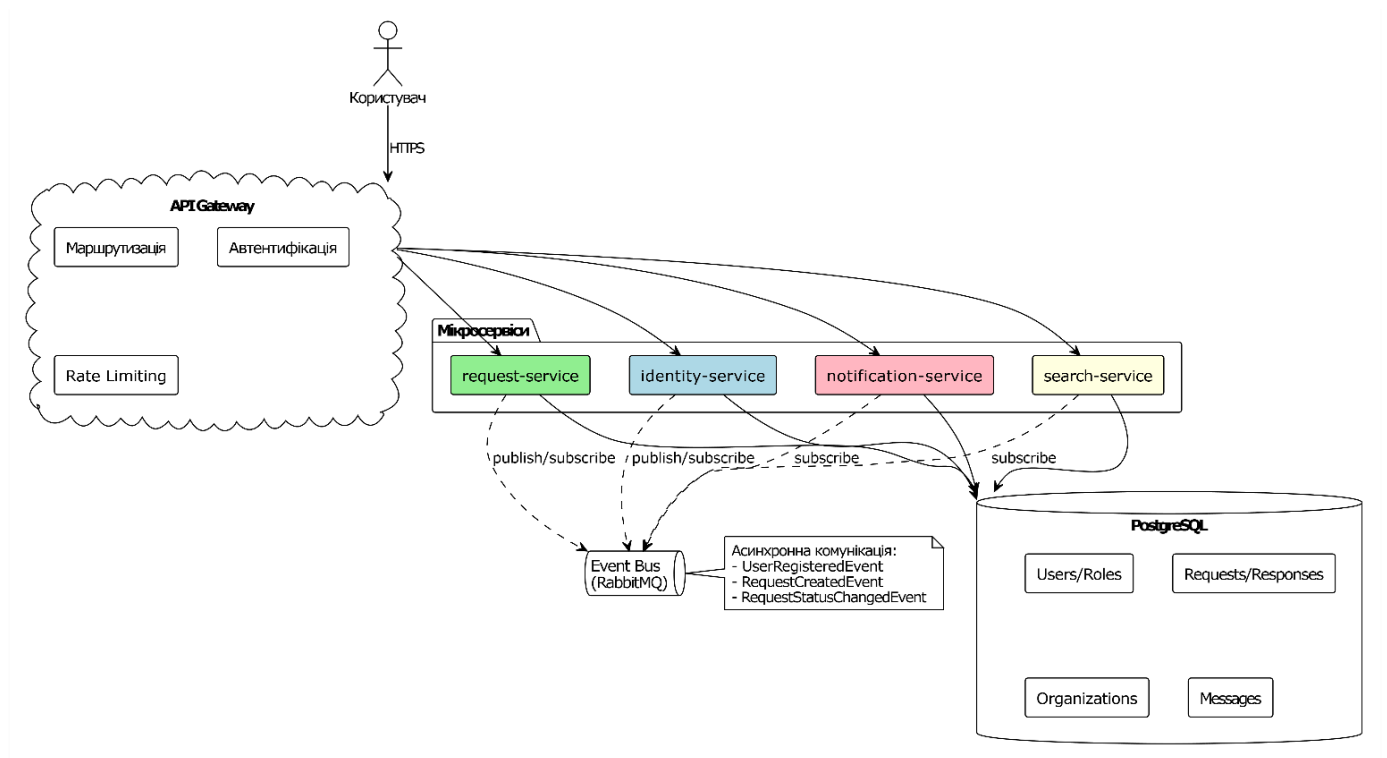


Рис.2.2. Архітектурна діаграма

– API Gateway. Це центральний компонент архітектури, що виступає в ролі посередника між клієнтськими застосунками та внутрішніми мікросервісами. Його впровадження дозволяє інкапсулювати складність внутрішньої структури системи та надає єдину точку для управління наскрізними задачами. Ключовими обов'язками API Gateway є маршрутизація запитів до відповідних сервісів, централізована автентифікація та авторизація користувачів, а також агрегація даних з кількох сервісів для оптимізації комунікації з клієнтом. Крім того, він реалізує такі наскрізні задачі, як логування, моніторинг та обмеження частоти запитів (rate limiting) для захисту від DDoS-атак;

– Мікросервіси. Розбиття системи на мікросервіси виконано за принципами предметно-орієнтованого проєктування (Domain-Driven Design, DDD) [26], де кожен сервіс відповідає за чітко визначену бізнес-область. Це забезпечує низьку зв'язаність

та високу згуртованість компонентів. identity-service відповідає за все, що пов'язано з ідентифікацією та профілями користувачів. request-service керує повним життєвим циклом запитів на допомогу. search-service надає функціонал для складного повнотекстового пошуку та фільтрації. notification-service відповідає за надсилання всіх типів сповіщень користувачам. Детальний опис зон відповідальності кожного сервісу наведено в табл. 2.3;

– Шина подій (Event Bus). Для забезпечення асинхронної комунікації та слабкої зв'язаності між сервісами використовується шина подій, реалізована за допомогою брокера повідомлень, наприклад, RabbitMQ. Замість прямих синхронних викликів, один сервіс публікує подію про те, що відбулася певна дія, а інші сервіси, зацікавлені в цій події, підписуються на неї та реагують відповідним чином. Наприклад, коли в identity-service реєструється новий користувач, він публікує подію UserRegisteredEvent. notification-service, підписаний на цю подію, отримує її та ініціює надсилання вітального листа. Перевага такого підходу полягає в тому, що identity-service не залежить від доступності notification-service.

Таблиця 2.3

Опис мікросервісів та їхніх зон відповідальності

Назва сервісу	Основна відповідальність	Ключові сутності	Основні технології
identity-service	Управління користувачами, автентифікація, авторизація, ролі, верифікація.	User, Role, Organization, UserProfile	Java, Spring Boot, Spring Security, PostgreSQL
request-service	Управління життєвим циклом запитів на допомогу та відгуків на них.	Request, RequestCategory, Response, Review	Java, Spring Boot, PostgreSQL
search-service	Індексація даних та надання API для повнотекстового пошуку та фільтрації.	RequestIndex, UserIndex	Java, Spring Boot
notification-service	Надсилання email-сповіщень та інших типів повідомлень.	Notification, Template	Java, Spring Boot, RabbitMQ

2.2.3. Обґрунтування вибору технологічного стеку

Вибір технологій є важливим кроком, що визначає продуктивність розробки, надійність та вартість підтримки системи в майбутньому. Обраний стек базується на перевірених, потужних та добре документованих технологіях з активною спільнотою:

– Backend: Java та Spring Boot. Java є однією з найнадійніших та найпродуктивніших платформ для створення складних веб-застосунків. Фреймворк Spring Boot, побудований на основі Spring, значно спрощує та прискорює розробку на Java, надаючи потужні інструменти для швидкого створення готових до розгортання мікросервісів. Зріла екосистема Spring, зокрема проєкти Spring Cloud, Spring Security та Spring Data, надає готові рішення для більшості завдань, що виникають при розробці розподілених систем, від сервіс-дискавері до безпеки та роботи з даними. Висока продуктивність віртуальної машини Java (JVM) та вбудована підтримка багатопоточності роблять цей стек ідеальним для обробки великої кількості одночасних запитів, що є критичним для високонавантаженої платформи;

– База даних: PostgreSQL. PostgreSQL є потужною об'єктно-реляційною СУБД з відкритим вихідним кодом, що здобула репутацію надзвичайно надійної, безпечної та гнучкої системи. Суворі відповідності принципам ACID гарантує цілісність даних, що є критично важливим для системи, яка обробляє персональні дані. Розширюваність PostgreSQL є однією з її ключових переваг. Нативна підтримка типів даних JSON/JSONB дозволяє ефективно зберігати гнучкі, частково структуровані дані, а розширення PostGIS перетворює PostgreSQL на повноцінну геопросторову базу даних, що може бути використано для реалізації складних геопросторових запитів (наприклад, "знайти всі запити в радіусі 5 км");

– Frontend: Thymeleaf та Bootstrap (Server-Side Rendering). Вибір технології для фронтенду є свідомим компромісом на користь простоти, швидкості розробки та оптимальної пошукової індексації (SEO). Thymeleaf є серверним шаблонізатором, що

безшовно інтегрується у Spring Boot, дозволяючи генерувати готовий HTML на сервері. Такий підхід (SSR) значно спрощує архітектуру, оскільки не вимагає окремого стеку технологій (Node.js, Webpack), що є типовим для Single-Page Application (SPA). Крім того, згенеровані на сервері сторінки легко та ефективно індексуються пошуковими системами, що є критично важливою перевагою для публічної платформи, яка має залучати нових користувачів через пошук. Використання CSS-фреймворку Bootstrap 5 дозволяє швидко створювати адаптивний та візуально привабливий користувацький інтерфейс завдяки великій бібліотеці готових компонентів.

2.3. Проектування бази даних

Проектування бази даних є ключовим етапом, що визначає структуру зберігання даних, їх цілісність та ефективність доступу до них. Для даної системи обрано реляційну модель, оскільки вона найкраще підходить для структурованих даних з чітко визначеними зв'язками.

2.3.1. Логічна модель даних та опис сутностей

Логічна модель даних описує основні сутності (таблиці), їх атрибути (колони) та зв'язки між ними. Детальна структура ключових сутностей представлена в таблицях 2.4–2.12.

Таблиця 2.4

Структура сутності Roles (Ролі)

Назва атрибута	Тип даних	Обмеження	Опис
id	INT	PK	Унікальний ідентифікатор ролі
name	ENUM	UNIQUE, NOT NULL	Назва ролі ('VOLUNTEER', 'RECIPIENT', 'ADMIN')

Таблиця 2.5

Структура сутності Users (Користувачі)

Назва атрибута	Тип даних	Обмеження	Опис
id	BIGINT	PK, auto-increment	Унікальний ідентифікатор користувача
email	VARCHAR(255)	UNIQUE, NOT NULL	Електронна пошта, використовується для входу
password_hash	VARCHAR(255)	NOT NULL	Хеш пароля користувача
first_name	VARCHAR(100)		Ім'я користувача
last_name	VARCHAR(100)		Прізвище користувача
phone_number	VARCHAR(20)	UNIQUE	Контактний номер телефону
status	ENUM	NOT NULL	Поточний статус акаунту ('ACTIVE', 'BANNED')
created_at	TIMESTAMP		Час створення акаунту

Таблиця 2.6

Структура сутності Responses (Відгуки на запити)

Назва атрибута	Тип даних	Обмеження	Опис
id	BIGINT	PK, auto-increment	Унікальний ідентифікатор відгуку
request_id	BIGINT	FK to Requests.id	Ідентифікатор запиту
volunteer_id	BIGINT	FK to Users.id	Ідентифікатор волонтера
message	TEXT		Супровідне повідомлення

Таблиця 2.7

Структура сутності Responses (Відгуки на запити)

status	ENUM	NOT NULL	Статус ('PENDING', 'ACCEPTED', 'REJECTED')
created_at	TIMESTAMP		Час створення відгуку

Таблиця 2.8

Структура сутності Organizations (Організації)

Назва атрибута	Тип даних	Обмеження	Опис
id	BIGINT	PK, auto-increment	Унікальний ідентифікатор організації
name	VARCHAR(255)	NOT NULL	Офіційна назва організації
edrpou_code	VARCHAR(10)	UNIQUE	Код ЄДРПОУ для верифікації
description	TEXT		Детальний опис діяльності
verification_status	ENUM	NOT NULL	Статус верифікації ('VERIFIED', 'NOT_VERIFIED')
owner_id	BIGINT	FK to Users.id, UNIQUE	Ідентифікатор користувача-власника

Таблиця 2.9

Структура сутності OrganizationMembers (Члени організацій)

Назва атрибута	Тип даних	Обмеження	Опис
user_id	BIGINT	FK to Users.id	Ідентифікатор користувача (волонтера)
organization_id	BIGINT	FK to Organizations.id	Ідентифікатор організації

Таблиця 2.10

Структура сутності RequestCategories (Категорії запитів)

Назва атрибута	Тип даних	Обмеження	Опис
id	INT	PK, auto-increment	Унікальний ідентифікатор категорії
name	VARCHAR(100)	UNIQUE, NOT NULL	Назва категорії

Таблиця 2.11

Структура сутності UserRoles (Ролі користувачів)

Назва атрибута	Тип даних	Обмеження	Опис
user_id	BIGINT	FK to Users.id	Ідентифікатор користувача
role_id	INT	FK to Roles.id	Ідентифікатор ролі

Структура сутності Requests (Запити на допомогу)

Назва атрибута	Тип даних	Обмеження	Опис
id	BIGINT	PK, auto-increment	Унікальний ідентифікатор запиту
title	VARCHAR(255)	NOT NULL	Заголовок запиту
description	TEXT	NOT NULL	Детальний опис потреби
status	ENUM	NOT NULL	Статус ('OPEN', 'CLOSED', 'PENDING_MODERATION')
category_id	INT	FK to RequestCategories.id	Ідентифікатор категорії
creator_id	BIGINT	FK to Users.id	Ідентифікатор автора запиту
location_lat	DECIMAL(10, 8)		Широта місцезнаходження
location_lon	DECIMAL(11, 8)		Довгота місцезнаходження
created_at	TIMESTAMP		Час створення запиту
id	BIGINT	PK, auto-increment	Унікальний ідентифікатор запиту
title	VARCHAR(255)	NOT NULL	Заголовок запиту
description	TEXT	NOT NULL	Детальний опис потреби
status	ENUM	NOT NULL	Статус ('OPEN', 'CLOSED', 'PENDING_MODERATION')
category_id	INT	FK to RequestCategories.id	Ідентифікатор категорії
creator_id	BIGINT	FK to Users.id	Ідентифікатор автора запиту
location_lat	DECIMAL(10, 8)		Широта місцезнаходження
location_lon	DECIMAL(11, 8)		Довгота місцезнаходження

Структура сутності Messages (Повідомлення)

Назва атрибута	Тип даних	Обмеження	Опис
id	BIGINT	PK, auto-increment	Унікальний ідентифікатор повідомлення
sender_id	BIGINT	FK to Users.id	Ідентифікатор відправника
recipient_id	BIGINT	FK to Users.id	Ідентифікатор отримувача
content	TEXT	NOT NULL	Текст повідомлення
sent_at	TIMESTAMP		Час відправлення
is_read	BOOLEAN		Статус прочитання

2.3.2 ER-діаграма та опис зв'язків ER-діаграма (Entity-Relationship Diagram) візуалізує структуру бази даних, показуючи таблиці та зв'язки між ними. Схематичне представлення моделі даних наведено на рис. 2.3.

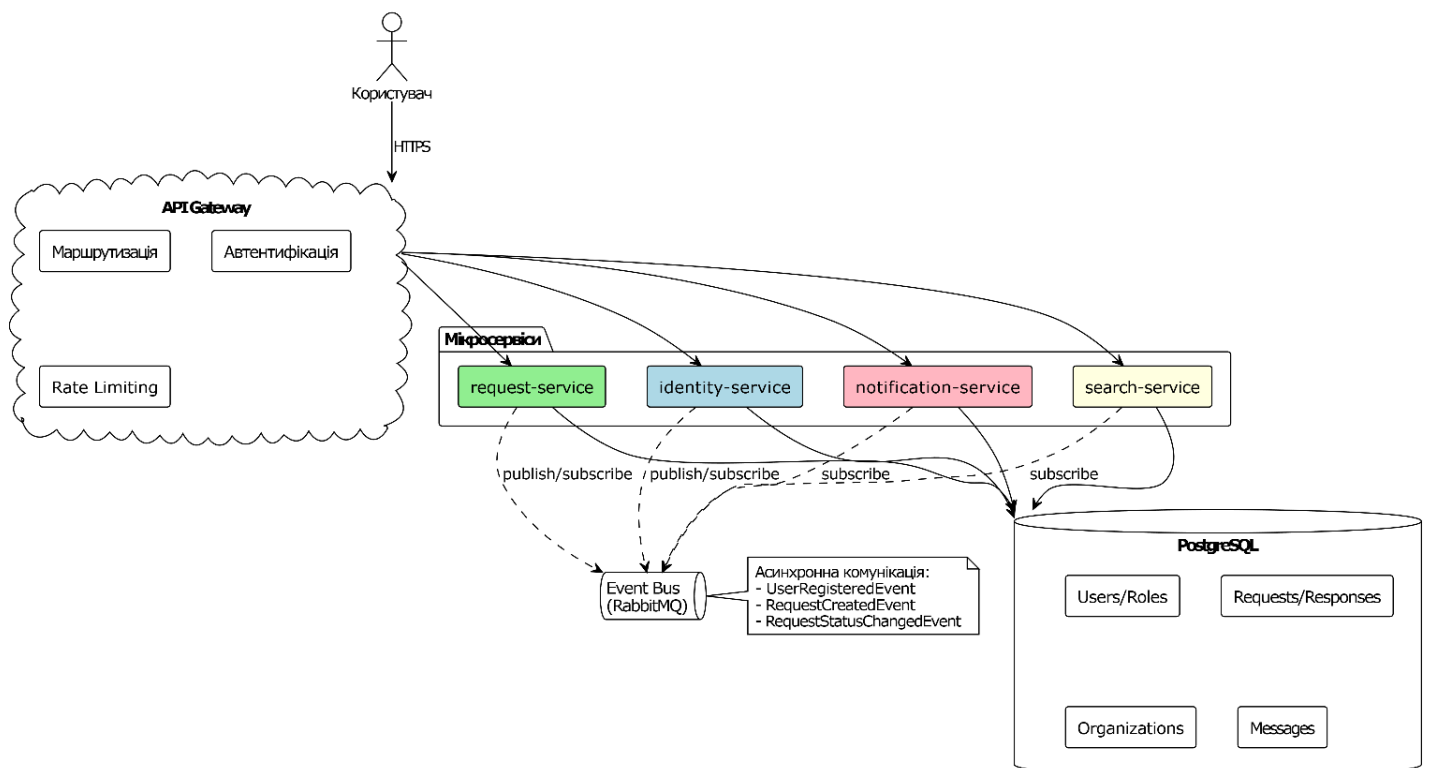


Рис. 2.3. Архітектурна діаграма .

Зв'язок "багато-до-багатьох" між сутностями Users та Roles реалізується через проміжну таблицю UserRoles. Такий дизайн забезпечує гнучкість, дозволяючи одному користувачеві мати декілька ролей (наприклад, бути одночасно і волонтером, і представником організації), а одній ролі бути присвоєною багатьом користувачам. Аналогічно, зв'язок "багато-до-багатьох" між Users та Organizations реалізовано через таблицю OrganizationMembers, що дозволяє волонтерам бути членами кількох організацій.

Зв'язок "один-до-багатьох" є найпоширенішим у спроектованій моделі. Наприклад, один користувач (в ролі реципієнта) може створити багато запитів, але кожен запит має лише одного автора. Це реалізовано за допомогою зовнішнього ключа creator_id в таблиці Requests, що посилається на Users.id. Подібним чином реалізовано зв'язки між Requests та Responses (один запит може мати багато відгуків), Users та Responses (один волонтер може залишити багато відгуків), а також між довідником RequestCategories та таблицею Requests.

Висновки до розділу 2

У другому розділі було розроблено детальний технічний проєкт системи підтримки волонтерської діяльності, що є фундаментом для її подальшої програмної реалізації:

- проведено ретельне проєктування функціоналу. На основі аналізу предметної області, викладеного в першому розділі, були визначені та формалізовані функціональні та нефункціональні вимоги до системи. Було детально описано ролі чотирьох ключових акторів — Волонтера, Реципієнта, Організації та Адміністратора — та створено діаграму варіантів використання з детальним описом основних сценаріїв їх взаємодії з платформою;
- спроектовано архітектуру системи. Зроблено обґрунтований вибір на користь гнучкої та масштабованої мікросервісної архітектури, що найкращим чином відповідає нефункціональним вимогам проєкту, таким як висока доступність та надійність. Було представлено загальну архітектурну схему та детально описано відповідальність кожного компонента: API Gateway, набору мікросервісів (identity-service, request-service, search-service, notification-service) та шини подій. Також було надано всебічне обґрунтування вибору технологічного стеку, що включає Java та Spring Boot для бекенду, PostgreSQL як систему управління базами даних, та комбінацію Thymeleaf і Bootstrap для реалізації фронтенду за моделлю серверного рендерингу;
- виконано проєктування бази даних. Розроблено логічну модель даних, описано структуру ключових сутностей та їх атрибутів у табличному вигляді. Створено концептуальну ER-діаграму та детально описано зв'язки між таблицями, включаючи реалізацію зв'язків "один-до-багатьох" та "багато-до-багатьох".

Таким чином, створений у цьому розділі проєкт є комплексною технічною специфікацією, що охоплює всі аспекти майбутньої системи та слугує міцною основою для переходу до етапів розробки та тестування.

РОЗДІЛ 3

РОЗРОБКА ТА РЕАЛІЗАЦІЯ СИСТЕМИ

Після завершення етапу аналізу (Розділ 1) та детального проєктування (Розділ 2), цей розділ присвячений практичній реалізації «Системи підтримки волонтерської діяльності». Метою є демонстрація перетворення архітектурних рішень, функціональних вимог та логічних моделей даних у конкретний програмний продукт. Розділ 3 детально описує вибір інструментарію, процес створення фізичної структури бази даних та наводить приклади програмного коду ключових компонентів системи. Кожен етап реалізації безпосередньо базується на проєктних рішеннях, обґрунтованих у попередньому розділі, зокрема на виборі мікросервісної архітектури та технологічного стеку, спрямованого на забезпечення масштабованості, надійності та гнучкості.

3.1. Вибір інструментів розробки

Вибір інструментів розробки є логічним продовженням обґрунтування технологічного стеку (див. підрозділ 2.2.3) і визначає конкретні програмні засоби, що були використані для втілення спроектованої архітектури. Кожен інструмент обрано з огляду на його відповідність нефункціональним вимогам, зрілість екосистеми та ефективність у вирішенні завдань проєкту.

Java (версія 17 LTS): В якості основної мови програмування для серверної частини обрано Java. Це обґрунтовано її винятковою надійністю, продуктивністю (завдяки віртуальній машині JVM) та величезною екосистемою. Java є стандартом для створення складних, високонавантажених корпоративних систем, що повністю відповідає вимогам до розробки платформи національного масштабу. Використання версії 17 (Long-Term Support) гарантує довгострокову підтримку, стабільність та доступ до сучасних мовних конструкцій, що підвищують читабельність коду та продуктивність розробки.

Spring Boot (версія 3.x): Для реалізації мікросервісної архітектури, обґрунтованої в підрозділі 2.2.1, було обрано фреймворк Spring Boot [27; 28]. Він є де-факто

стандартом для розробки мікросервісів на Java, оскільки радикально спрощує конфігурацію та прискорює процес розробки. Ключові модулі, що використовуються в проєкті:

Spring Web (MVC): Використовується для створення REST API ендпоінтів у кожному мікросервісі (наприклад, identity-service, request-service), а також для обробки веб-запитів та рендерингу HTML-сторінок за допомогою Thymeleaf, відповідно до обраної моделі Server-Side Rendering (SSR).

Spring Data JPA: Забезпечує шар абстракції для взаємодії з базою даних PostgreSQL. Цей модуль дозволяє значно скоротити обсяг шаблонного коду (boilerplate) для операцій CRUD (Create, Read, Update, Delete) шляхом визначення інтерфейсів-репозиторіїв, що автоматично реалізуються фреймворком.

Spring Security: Ключовий компонент для реалізації функціоналу identity-service. Він надає комплексне рішення для захисту ендпоінтів, автентифікації (перевірки особи) та авторизації (перевірки прав доступу) користувачів, що є критичним для виконання вимог FR1, FR4 та NFR4.

PostgreSQL (версія 15.x): Як було обґрунтовано в підрозділі 2.2.3, PostgreSQL обрано як основну систему управління базами даних. Це потужна, об'єктно-реляційна СУБД з відкритим кодом, що забезпечує сувору відповідність принципам ACID та гарантує цілісність даних [29]. Її розширені можливості, такі як підтримка JSONB та геопросторових даних (через PostGIS), закладають фундамент для майбутнього розширення функціоналу системи, наприклад, для реалізації складного гео-пошуку.

Apache Maven: Обрано як інструмент для автоматизації збірки проєкту та управління його залежностями. Maven стандартизує життєвий цикл проєкту (компіляція, тестування, пакування, розгортання) та керує підключенням всіх необхідних бібліотек (Spring Boot, PostgreSQL driver тощо) через декларативний pom.xml файл. Це забезпечує відтворюваність збірки у будь-якому середовищі, що є критично важливим для CI/CD процесів.

Git: В якості системи контролю версій використовується Git. У проєкті з мікросервісною архітектурою, де розробка може вестися паралельно над різними

сервісами, Git є незамінним інструментом. Він дозволяє ефективно відстежувати зміни, керувати гілками для розробки нового функціоналу, вирішувати конфлікти при злитті коду та підтримувати цілісність кодової бази.

IDE (IntelliJ IDEA Ultimate): В якості основного інтегрованого середовища розробки (IDE) обрано IntelliJ IDEA. Її розширені можливості для роботи з Java та Spring Boot, такі як інтелектуальне автодоповнення коду, рефакторинг, вбудовані інструменти для роботи з Git, Maven та базами даних, значно підвищують продуктивність розробників та якість коду.

3.2. Розробка бази даних

Цей підрозділ представляє фізичну реалізацію моделі даних, спроектованої у підрозділі 2.3. Наведені нижче SQL-скрипти (Data Definition Language, DDL) написані на діалекті PostgreSQL і точно відповідають ER-діаграмі (рис. 2.3) та структурам таблиць (таблиці 2.4 – 2.12).

Скрипти організовані логічно: спочатку створюються користувацькі типи даних (ENUM), потім основні («батьківські») сутності, і наостанок — залежні сутності та таблиці зв'язків.

3.2.1. Створення перелічуваних типів (ENUMs)

Для забезпечення цілісності даних та оптимізації зберігання для полів зі строго визначеним набором значень (наприклад, статуси) створюються окремі типи даних.

SQL

```
-- Статуси користувачів (для табл. Users)
```

```
-- Додано PENDING_EMAIL_VERIFICATION згідно зі Сценарієм 1 (2.1.3)
```

```
CREATE TYPE user_status_enum AS ENUM (
```

```
    'ACTIVE',
```

```
    'BANNED',
```

```
    'PENDING_EMAIL_VERIFICATION'
```

```
);
```

-- Ролі користувачів (для табл. Roles)

-- Додано 'ORGANIZATION' для власників організацій

```
CREATE TYPE role_name_enum AS ENUM (  
    'VOLUNTEER',  
    'RECIPIENT',  
    'ADMIN',  
    'ORGANIZATION'  
);
```

-- Статуси верифікації (для табл. Organizations)

```
CREATE TYPE org_verification_enum AS ENUM (  
    'VERIFIED',  
    'NOT_VERIFIED',  
    'PENDING_VERIFICATION'  
);
```

-- Статуси запитів (для табл. Requests)

-- Додано 'IN_PROGRESS' та оновлено 'PENDING_MODERATION' згідно зі
Сценарієм 2 (2.1.3)

```
CREATE TYPE request_status_enum AS ENUM (  
    'OPEN',  
    'CLOSED',  
    'PENDING_MODERATION',  
    'IN_PROGRESS'  
);
```

-- Статуси відгуків (для табл. Responses)

```
CREATE TYPE response_status_enum AS ENUM (  

```

```
'PENDING',  
'ACCEPTED',  
'REJECTED'
```

```
);
```

3.2.2. Скрипти створення таблиць (DDL)

Послідовність створення таблиць враховує залежності зовнішніх ключів (Foreign Key).

SQL

-- Таблиця 2.4: Users (Користувачі)

```
CREATE TABLE users (  
    id BIGSERIAL PRIMARY KEY,  
    email VARCHAR(255) UNIQUE NOT NULL,  
    password_hash VARCHAR(255) NOT NULL,  
    first_name VARCHAR(100),  
    last_name VARCHAR(100),  
    phone_number VARCHAR(20) UNIQUE,  
    status          user_status_enum          NOT          NULL          DEFAULT  
'PENDING_EMAIL_VERIFICATION',  
    created_at TIMESTAMP NOT NULL DEFAULT current_timestamp  
);
```

-- Таблиця 2.5: Roles (Ролі)

```
CREATE TABLE roles (  
    id SERIAL PRIMARY KEY,  
    name role_name_enum UNIQUE NOT NULL  
);
```

-- Таблиця 2.6: UserRoles (Ролі користувачів)

-- Проміжна таблиця для зв'язку "багато-до-багатьох"

```
CREATE TABLE user_roles (  
    user_id BIGINT NOT NULL,  
    role_id INT NOT NULL,  
    PRIMARY KEY (user_id, role_id),  
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,  
    FOREIGN KEY (role_id) REFERENCES roles(id) ON DELETE RESTRICT  
);
```

-- Таблиця 2.7: Organizations (Організації)

```
CREATE TABLE organizations (  
    id BIGSERIAL PRIMARY KEY,  
    name VARCHAR(255) NOT NULL,  
    edrpou_code VARCHAR(10) UNIQUE,  
    description TEXT,  
    verification_status org_verification_enum NOT NULL DEFAULT 'NOT_VERIFIED',  
    owner_id BIGINT UNIQUE NOT NULL,  
    created_at TIMESTAMP NOT NULL DEFAULT current_timestamp,  
    FOREIGN KEY (owner_id) REFERENCES users(id) ON DELETE RESTRICT  
);
```

-- Таблиця 2.8: OrganizationMembers (Члени організацій)

-- Проміжна таблиця для зв'язку "багато-до-багатьох"

```
CREATE TABLE organization_members (  
    user_id BIGINT NOT NULL,  
    organization_id BIGINT NOT NULL,  
    joined_at TIMESTAMP NOT NULL DEFAULT current_timestamp,  
    PRIMARY KEY (user_id, organization_id),  
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,  
    FOREIGN KEY (organization_id) REFERENCES organizations(id) ON DELETE
```

CASCADE

);

-- Таблиця 2.9: RequestCategories (Категорії запитів)

```
CREATE TABLE request_categories (  
    id SERIAL PRIMARY KEY,  
    name VARCHAR(100) UNIQUE NOT NULL,  
    parent_category_id INT,  
    FOREIGN KEY (parent_category_id) REFERENCES request_categories(id) ON  
DELETE SET NULL  
);
```

-- Таблиця 2.10: Requests (Запити на допомогу)

```
CREATE TABLE requests (  
    id BIGSERIAL PRIMARY KEY,  
    title VARCHAR(255) NOT NULL,  
    description TEXT NOT NULL,  
    status request_status_enum NOT NULL DEFAULT 'PENDING_MODERATION',  
    category_id INT,  
    creator_id BIGINT NOT NULL,  
    location_lat DECIMAL(10, 8),  
    location_lon DECIMAL(11, 8),  
    address_details VARCHAR(512),  
    created_at TIMESTAMP NOT NULL DEFAULT current_timestamp,  
    updated_at TIMESTAMP NOT NULL DEFAULT current_timestamp,  
    FOREIGN KEY (category_id) REFERENCES request_categories(id) ON DELETE SET  
NULL,  
    FOREIGN KEY (creator_id) REFERENCES users(id) ON DELETE CASCADE  
);
```

-- Таблиця 2.11: Responses (Відгуки на запити)

```
CREATE TABLE responses (  
    id BIGSERIAL PRIMARY KEY,  
    request_id BIGINT NOT NULL,  
    volunteer_id BIGINT NOT NULL,  
    message TEXT,  
    status response_status_enum NOT NULL DEFAULT 'PENDING',  
    created_at TIMESTAMP NOT NULL DEFAULT current_timestamp,  
    FOREIGN KEY (request_id) REFERENCES requests(id) ON DELETE CASCADE,  
    FOREIGN KEY (volunteer_id) REFERENCES users(id) ON DELETE CASCADE,  
    -- Запобігання дублюванню відгуків від одного волонтера на той самий запит  
    UNIQUE (request_id, volunteer_id)  
);
```

-- Таблиця 2.12: Messages (Повідомлення)

-- Для реалізації внутрішньої системи обміну повідомленнями (FR6)

```
CREATE TABLE messages (  
    id BIGSERIAL PRIMARY KEY,  
    sender_id BIGINT NOT NULL,  
    recipient_id BIGINT NOT NULL,  
    -- Опціональний зв'язок із запитом, щоб групувати чати за контекстом  
    request_id BIGINT,  
    content TEXT NOT NULL,  
    sent_at TIMESTAMP NOT NULL DEFAULT current_timestamp,  
    is_read BOOLEAN NOT NULL DEFAULT false,  
    FOREIGN KEY (sender_id) REFERENCES users(id) ON DELETE SET NULL,  
    FOREIGN KEY (recipient_id) REFERENCES users(id) ON DELETE SET NULL,  
    FOREIGN KEY (request_id) REFERENCES requests(id) ON DELETE SET NULL
```

);

-- Індеси для прискорення вибірок за зовнішніми ключами

CREATE INDEX idx_requests_creator_id ON requests(creator_id);

CREATE INDEX idx_requests_category_id ON requests(category_id);

CREATE INDEX idx_responses_request_id ON responses(request_id);

CREATE INDEX idx_responses_volunteer_id ON responses(volunteer_id);

CREATE INDEX idx_messages_sender_id ON messages(sender_id);

CREATE INDEX idx_messages_recipient_id ON messages(recipient_id);

CREATE INDEX idx_messages_request_id ON messages(request_id);

3.3. Реалізація мікросервісів

У цьому підрозділі описано практичну реалізацію серверної частини системи архітектура якої була спроектована у другому розділі. Відповідно до обраного архітектурного стилю, система складається з набору незалежних мікросервісів, реалізованих на платформі Java Spring Boot. Кожен сервіс виконує чітко окреслену бізнес-функцію та володіє власною схемою бази даних, що забезпечує ізоляцію даних та відмовостійкість.

3.3.1. Identity Service: управління користувачами та автентифікація

Сервіс ідентифікації (Identity Service) виступає центральним елементом підсистеми безпеки. Його функціональне призначення полягає у забезпеченні повного життєвого циклу облікових записів: від реєстрації та верифікації email до автентифікації та управління правами доступу.

З технічної точки зору, сервіс реалізовано з використанням механізму JSON Web Token (JWT), що дозволяє забезпечити автентифікацію без збереження стану сесії на сервері (stateless). Це рішення є критично важливим для забезпечення горизонтальної масштабованості системи.

Структура доменної моделі

Основною сутністю сервісу є обліковий запис користувача. При проектуванні моделі даних було застосовано архітектурний патерн розділення ідентифікаторів (табл. 3.1).

Таблиця 3.1

Типи ідентифікаторів користувача в системі

Ідентифікатор	Тип даних	Призначення та сфера видимості
id	BIGINT	Внутрішній первинний ключ. Використовується виключно для забезпечення зв'язків (Foreign Keys) у базі даних та оптимізації індексів. Не передається клієнту.
publicId	UUID	Публічний ідентифікатор. Використовується у зовнішніх REST API. Забезпечує безпеку системи, унеможливаючи перебір профілів зломисниками (ID enumeration attack).

Алгоритм автентифікації

Процес входу користувача в систему реалізовано за наступним алгоритмом:

1. Клієнтський застосунок надсилає запит на ендпоінт /auth/login з обліковими даними.
2. Сервіс виконує пошук запису в базі даних та перевірку хешу пароля (використовується алгоритм BCrypt).
3. Перевіряється статус облікового запису (дозволено вхід лише зі статусом ACTIVE).
4. У разі успішної перевірки генерується JWT-токен, що містить зашифровані дані про користувача (Claims). Структуру токена наведено в табл. 3.2.

Структура корисного навантаження (Payload) JWT-токена

Атрибут (Claim)	Опис значення	Приклад
sub (subject)	Унікальний ідентифікатор користувача	"42"
email	Електронна адреса	"user@example.com"
roles	Масив призначених ролей	``
iat	Час емісії токена (UNIX timestamp)	1701234567
exp	Час закінчення дії токена	1701238167

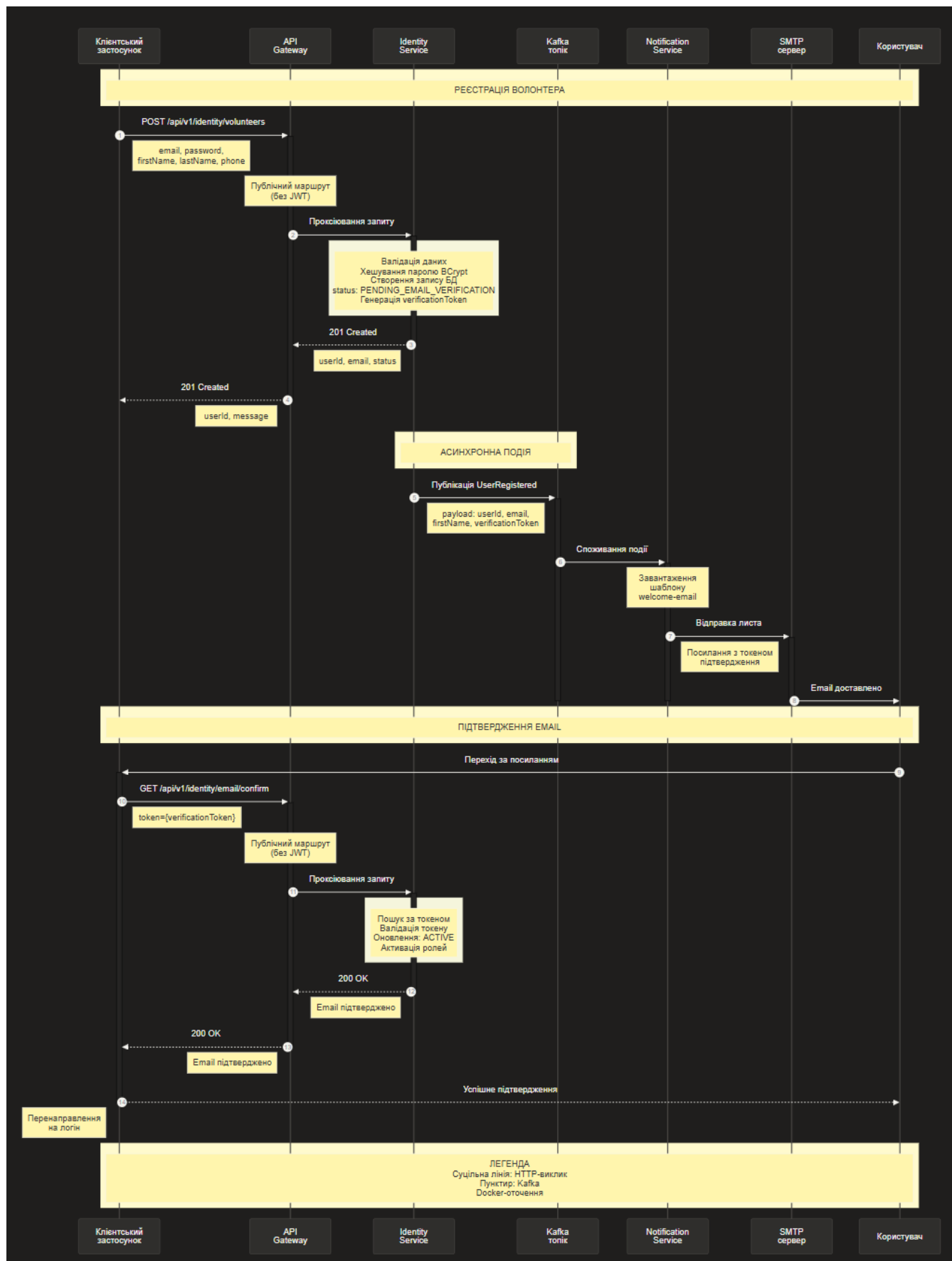


Рис. 3.1. Послідовність реєстрації та верифікації волонтера

Рольова модель доступу

Система підтримує гнучку модель управління доступом (RBAC), де права визначаються ролями користувача. Характеристику ролей наведено в табл. 3.3.

Таблиця 3.3

Матриця ролей та повноважень користувачів

Роль	Опис суб'єкта	Ключові функціональні можливості
VOLUNTEER	Волонтер	Пошук та фільтрація запитів, створення відгуків, комунікація з реципієнтами.
RECIPIENT	Реципієнт	Створення запитів на допомогу, модерація вхідних відгуків, підтвердження виконання.
ORGANIZATION	Організація	Розширені права волонтера: створення команд, управління проєктами, звітність.
ADMIN	Адміністратор	Повний доступ до системи, верифікація користувачів, модерація контенту.

3.3.2. Request Service: управління запитами на допомогу

Сервіс запитів (Request Service) реалізує основну бізнес-логіку платформи, забезпечуючи взаємодію між тими, хто потребує допомоги, та тими, хто може її надати.

Управління життєвим циклом

Для забезпечення коректності бізнес-процесів розроблено машину станів (State Machine), яка контролює переходи запиту між різними етапами виконання (табл. 3.4).

Таблиця 3.4

Стани запиту та логіка переходів

Поточний стан	Дозволені переходи	Умова переходу / Ініціатор
PENDING_MODERATION	OPEN, REJECTED	Модерація адміністратором після створення.
OPEN	IN_PROGRESS, CLOSED	Автор приймає відгук волонтера або закриває запит.
IN_PROGRESS	COMPLETED, OPEN	Підтвердження отримання допомоги або відновлення пошуку (якщо волонтер відмовився).
COMPLETED	—	Фінальний стан (успіх).
CLOSED	OPEN	Відновлення актуальності запиту автором.

Таблиця 3.5

Типологія запитів у системі

Тип запиту	Функціональне призначення
HELP_REQUEST	Класичний запит на допомогу (потреба у речах, послугах).
VOLUNTEER_OFFER	Проактивна пропозиція допомоги від волонтера (наприклад, "можу перевезти вантаж").
FUNDRAISER	Організація збору коштів (містить платіжні реквізити та прогрес-бар).



Рис. 3.2. Життєвий цикл запиту та відгуку в Request Service

Механізм відгуків

Комунікація ініціюється через створення відгуку (Response). Цей процес реалізовано транзакційно: система перевіряє унікальність відгуку (один волонтер — один відгук на конкретний запит) та генерує відповідні події для сервісу сповіщень. Стани відгуків описано в табл. 3.6.

Таблиця 3.6

Життєвий цикл відгуку волонтера

Стан	Опис процесу
PENDING	Відгук створено, очікує рішення автора запиту.
ACCEPTED	Автор прийняв пропозицію. Контактні дані сторін відкриваються.
REJECTED	Автор відхилив пропозицію. Волонтер отримує сповіщення.

3.3.3 API Gateway: маршрутизація та безпека

Для уніфікації доступу до мікросервісів та підвищення рівня безпеки впроваджено шлюз API (API Gateway) на базі Spring Cloud Gateway. Він приховує внутрішню топологію мережі від клієнта та виконує наскрізні функції (табл. 3.7).

Таблиця 3.7

Функціональні обов'язки API Gateway

Функція	Реалізація
Маршрутизація	Динамічне перенаправлення запитів на основі URL-префіксів.
Автентифікація	Валідація підпису JWT-токена на вході в систему.
Передача контексту	Вилучення User-Id та Roles з токена і передача їх у заголовках до мікросервісів (табл. 3.9).
Rate Limiting	Захист від DDoS-атак шляхом обмеження кількості запитів (Redis).

Таблиця 3.8

Карта маршрутизації запитів

URL-шаблон	Цільовий сервіс	Рівень доступу
/api/v1/identity/auth/**	identity-service	Публічний (без токена)
/api/v1/requests/**	request-service	Змішаний (GET — публічний, інші — захищений)
/api/v1/search/**	search-service	Публічний
/api/v1/notifications/**	notification-service	Захищений

Після успішної перевірки токена, Gateway збагачує запит внутрішніми заголовками (табл. 3.9), що дозволяє мікросервісам довіряти даним про користувача без повторної перевірки токена.

Таблиця 3.9

Внутрішні заголовки контексту безпеки

Заголовок	Призначення
X-User-Id	Унікальний ідентифікатор ініціатора запиту.
X-User-Roles	Перелік ролей для перевірки прав доступу на рівні сервісів.

3.3.4 Notification Service: система асинхронних сповіщень

Для забезпечення реактивності системи та своєчасного інформування користувачів реалізовано сервіс сповіщень. Архітектурно він побудований на принципах подієво-орієнтованої взаємодії (Event-Driven Architecture). Сервіс підписується на черги повідомлень у брокері RabbitMQ.

Таблиця 3.10

Карта подій системи та сценаріїв сповіщень

Подія (Event)	Джерело	Дія сервісу сповіщень
UserRegistered	Identity Service	Відправка листа з посиланням активації акаунту.
RequestApproved	Request Service	Інформування автора про публікацію запиту.
VolunteerResponded	Request Service	Сповіщення автора про новий відгук на запит.
ResponseAccepted	Request Service	Інформування волонтера, що його допомогу прийнято.

Таблиця 3.11

Реєстр шаблонів електронних листів

Назва шаблону	Параметри контексту	Призначення
welcome-email	Ім'я, посилання верифікації	Привітання нового користувача.
new-response	Ім'я волонтера, назва запиту	Оперативне інформування про відгук.
request-approved	Назва запиту, посилання	Підтвердження проходження модерації.

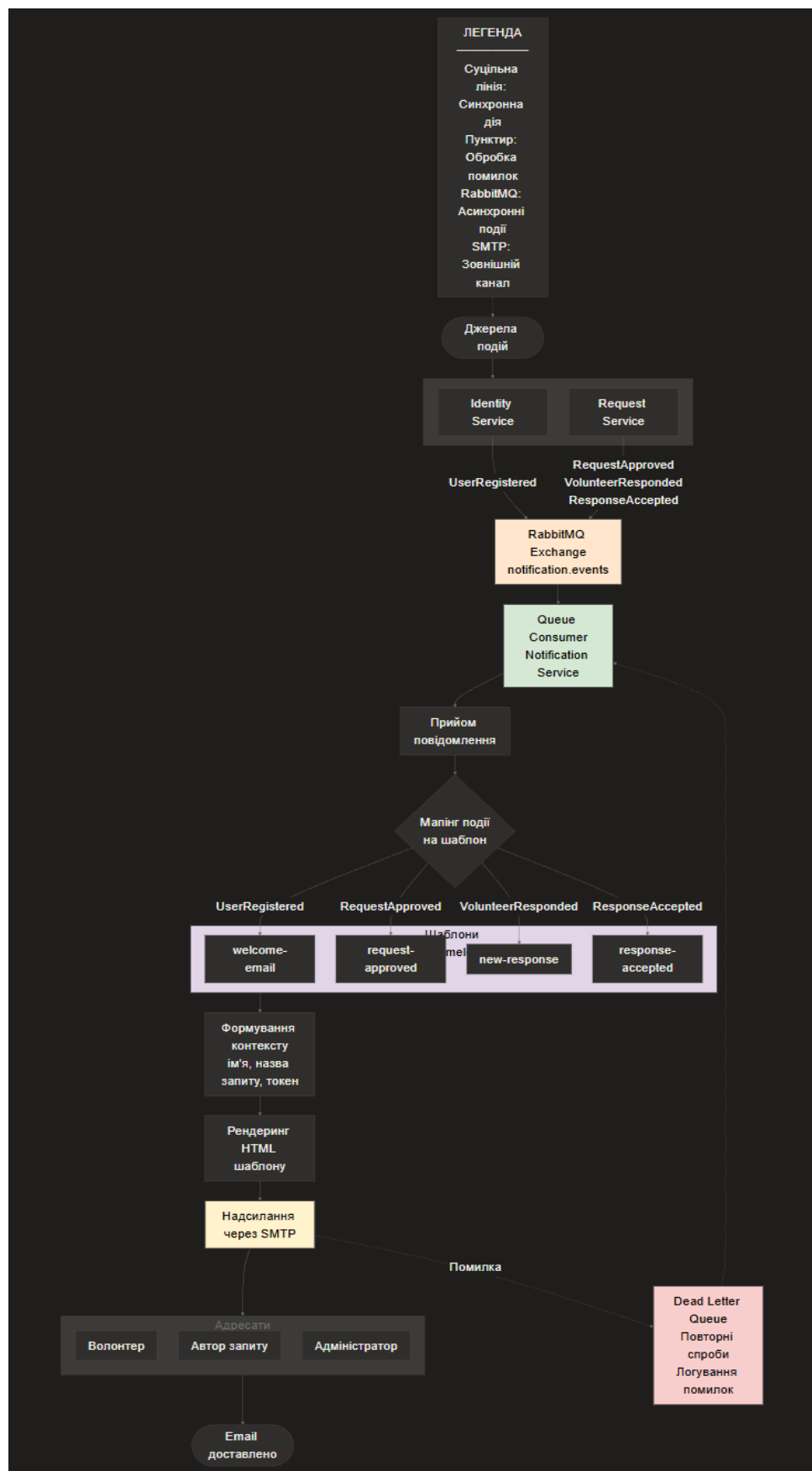


Рис. 3.3. Обробка подій у Notification Service

3.3.5. Search Service: підсистема повнотекстового пошуку

Для реалізації вимоги щодо швидкого пошуку релевантних запитів (FR3) впроваджено окремий сервіс пошуку на базі Elasticsearch. Використано патерн CQRS, де операції запису йдуть у PostgreSQL, а операції читання та складного пошуку — в Elasticsearch (табл. 3.12).

Таблиця 3.12

Розподіл відповідальності між сховищами даних

Характеристика	PostgreSQL	Elasticsearch
Тип навантаження	OLTP (Транзакції)	OLAP (Пошук та аналітика)
Тип даних	Нормалізовані дані	Денормалізовані документи (індекс)
Синхронізація	Первинне джерело	Асинхронне оновлення через RabbitMQ

Таблиця 3.13

Структура пошукового індексу (Mapping)

Поле	Тип даних	Аналізатор	Призначення
title, description	text	ukrainian	Повнотекстовий пошук з урахуванням відмінків.
location	geo_point	—	Геопросторовий пошук (фільтрація за радіусом).
type, status	keyword	—	Точна фільтрація (фасетний пошук).

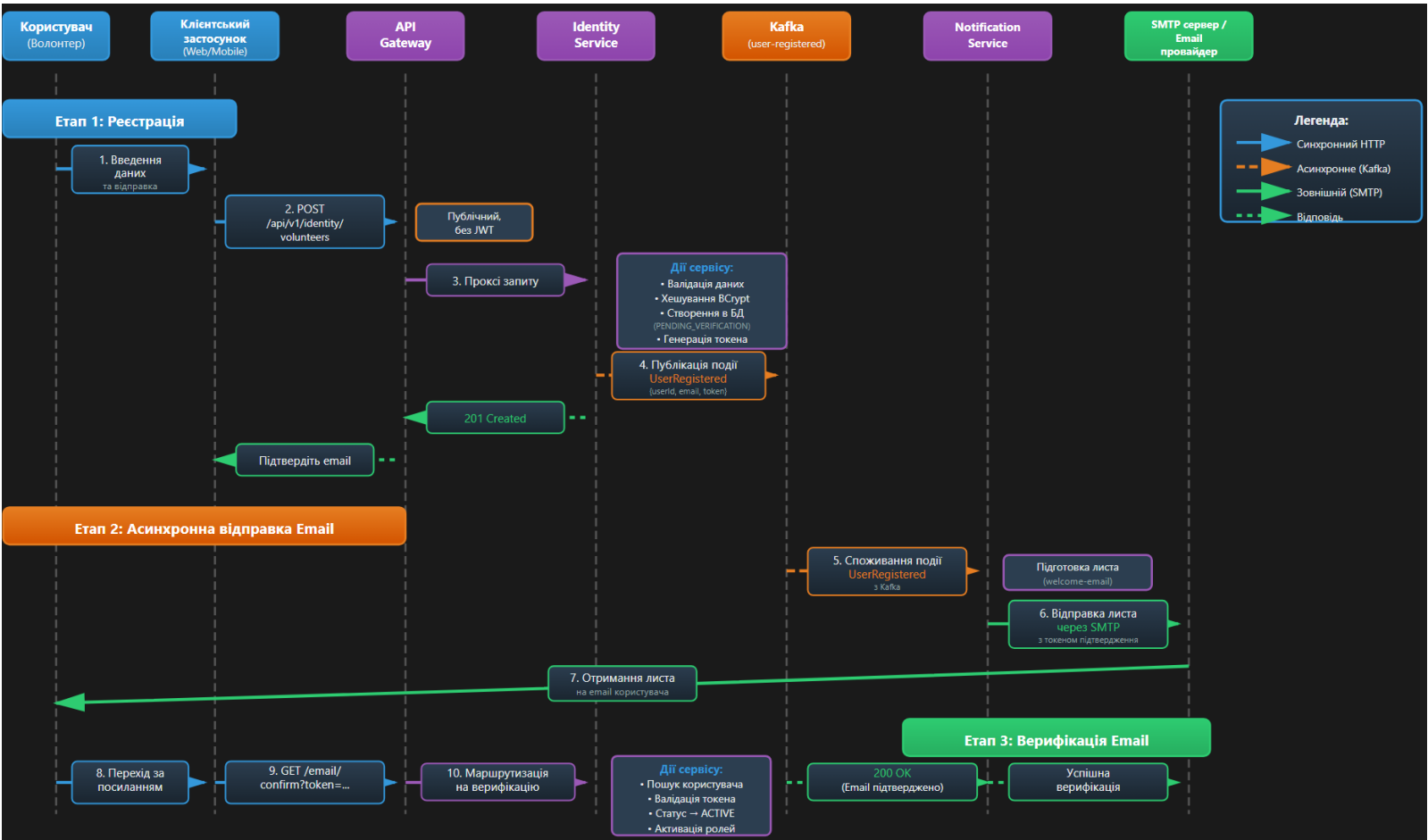


Рис. 3.4. Синхронізація даних між Request Service і Search Service (CQRS)

Така архітектура дозволяє реалізувати складні сценарії пошуку, наведені в табл. 3.14.

Таблиця 3.14

Реалізовані сценарії пошуку

Сценарій	Приклад запиту	Технічна реалізація
Fuzzy-пошук	"волонтер"	Знаходить "волонтер" (стійкість до одруківок).
Гео-пошук	"у радіусі 10 км"	Geo-distance query по координатах.
Сортування	"найновіші"	Сортування за полем createdAt.

3.4. Реалізація веб-інтерфейсу користувача

Клієнтська частина системи розроблена як односторінковий застосунок (SPA) з використанням бібліотеки React [32]. Це забезпечує високу швидкість відгуку інтерфейсу та зменшує навантаження на сервер, оскільки завантажуються лише необхідні дані у форматі JSON.

3.4.1. Технологічний стек фронтенду

Вибір інструментів фронтенду (табл. 3.15) базується на вимогах до продуктивності та сучасності інтерфейсу.

Таблиця 3.15

Інструментарій клієнтської розробки

Компонент	Технологія / Бібліотека	Обґрунтування вибору
Core	React 18.x	Компонентний підхід, віртуальний DOM.
State Management	TanStack Query	Ефективне кешування серверних даних, deduplication запитів.
UI Kit	Mantine UI 7.x	Готовий набір адаптивних компонентів, підтримка тем.
Routing	React Router 6.x	Клієнтська маршрутизація без перезавантаження сторінки.

3.4.2. Структура та інтерфейс застосунку

Навігаційна структура застосунку побудована за рольовим принципом (табл. 3.16).

Карта основних екранів застосунку

Екран	URL-шлях	Доступність
Dashboard	/dashboard	Головна сторінка з віджетами статистики.
Profile	/profile	Редагування особистих даних.
My Requests	/my-requests	Управління власними запитами (для реципієнтів).
Search	/search	Пошук запитів (доступний усім).

3.4.3. Сторінка входу та реєстрації

Забезпечує доступ до системи. Реалізовано валідацію полів на льоту та обробку помилок авторизації.

Volunteer Support

Home Fundraisers Sign In Get Started

Welcome back

Sign in to your Volunteer Support account

Email *
@recipient@demo.com

Password *

Sign in

or

Don't have an account? Register

Рис. 3.5. Інтерфейс сторінки входу в систему

3.4.4. Головна панель

Головну панель системи реалізовано як центральний інформаційний хаб, що забезпечує миттєвий доступ до агрегованих даних користувача (рис. 3.6). Інтерфейс адаптується залежно від ролі: для координаторів відображається статистика створених запитів та їх поточних статусів, для волонтерів — персоналізована стрічка рекомендованих запитів на основі зазначених навичок та геолокації.

Статистичні картки у верхній частині екрану відображають кількість створених запитів, опублікованих оголошень, чернеток та отриманих відгуків волонтерів. Кругова діаграма візуалізує розподіл запитів за статусами, що дозволяє координатору оцінити завантаженість одним поглядом. Блок активності волонтерів у нижній частині панелі демонструє останні відгуки з можливістю швидкого переходу до їх обробки. Побудову інтерфейсу виконано на принципах "інформація на один погляд", що мінімізує кількість взаємодій для отримання критично важливих даних.

3.4.5. Сторінка профілю

Сторінку профілю призначено для відображення персональних даних користувача, статусу верифікації та призначеної ролі в системі (рис. 3.7). Кожен обліковий запис ідентифікується унікальним UUID, що забезпечує конфіденційність внутрішніх ідентифікаторів бази даних та відповідає вимогам безпеки.

Інформаційна структура сторінки включає блок основних даних із зазначенням імені, електронної пошти та аватару користувача. Окремо винесено статус облікового запису, що містить роль (волонтер, координатор, адміністратор) та стан активації. Метадані профілю включають публічний ідентифікатор та дату реєстрації в системі. Блок налаштувань безпеки забезпечує керування email-сповіщеннями та налаштування двофакторної автентифікації.

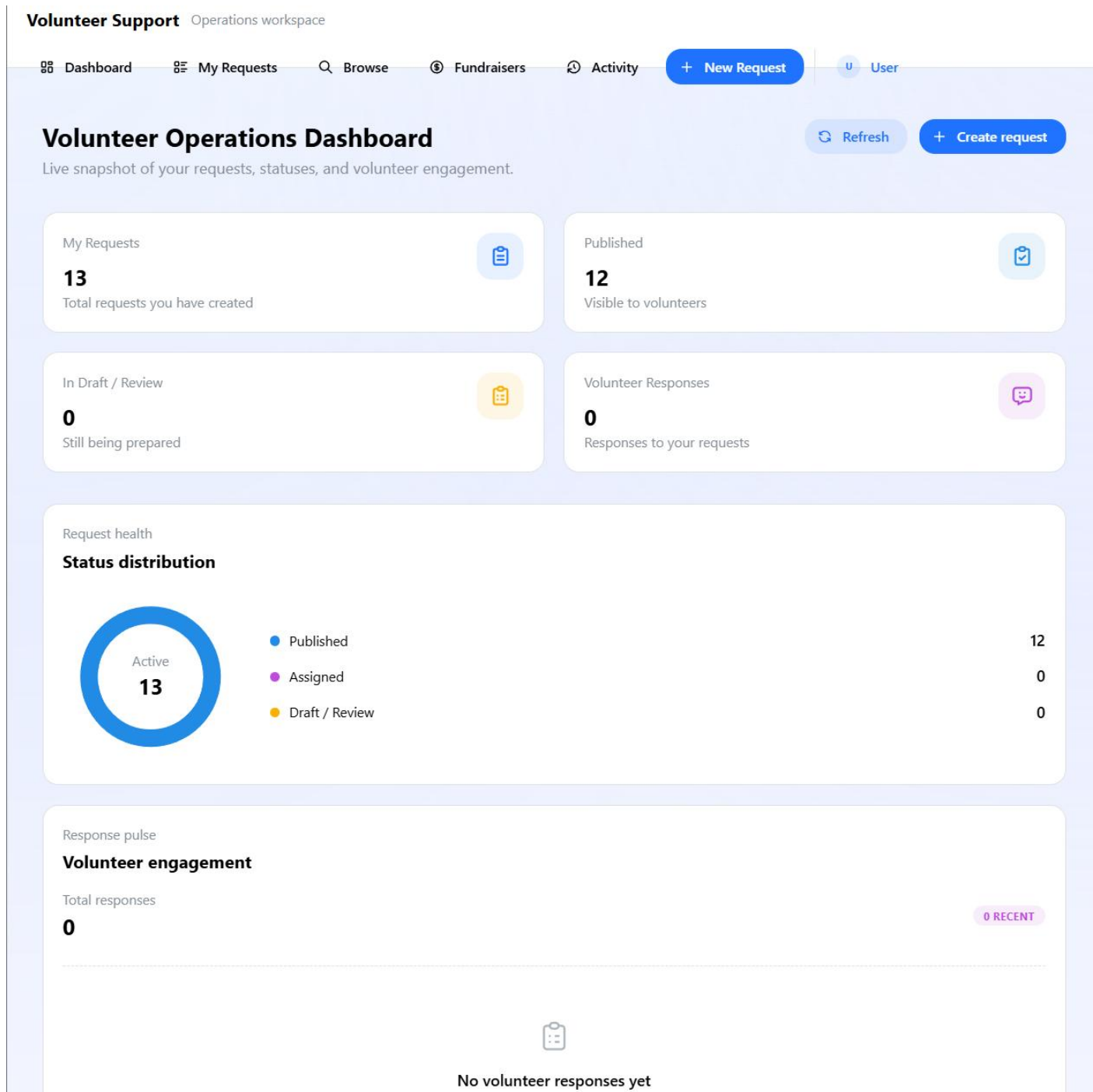


Рис. 3.6. Головна сторінка особистого кабінету

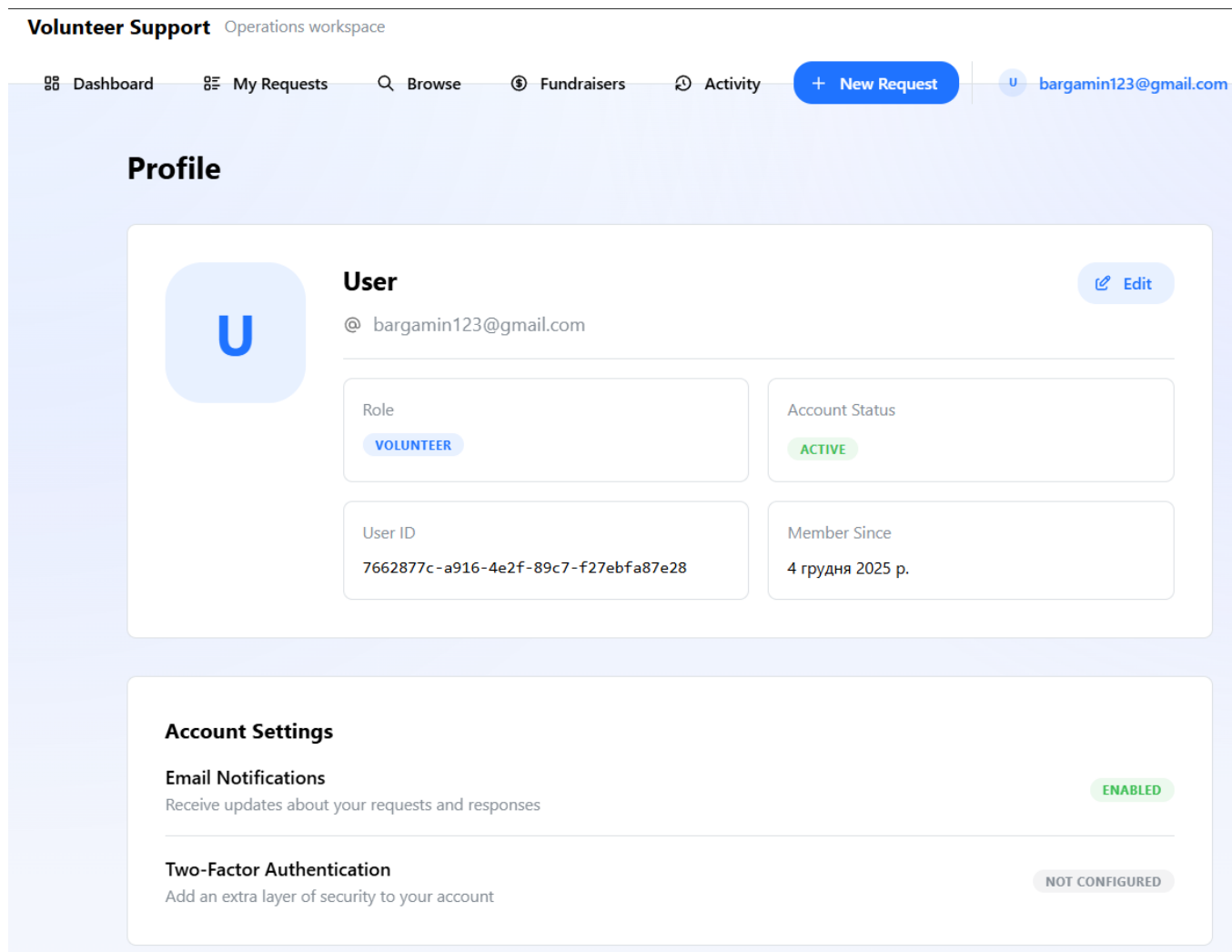


Рис. 3.7. Сторінка редагування профілю користувача

3.4.6. Пошук запитів

Інтерфейс пошуку запитів реалізує багаторівневу систему фільтрації для ефективного знаходження релевантних волонтерських можливостей (рис. 3.8). Панель фільтрів розташовано у верхній частині сторінки, що забезпечує фільтрацію за текстовим запитом, географічним розташуванням, статусом та пріоритетом.

Результати пошуку відображаються у вигляді карток із ключовою інформацією: назвою, локацією, коротким описом, індикатором пріоритету та терміном виконання. Статистичні індикатори над результатами демонструють загальну кількість знайдених запитів, кількість запитів з високим пріоритетом та кількість відкритих для волонтерів позицій. Для обробки великих обсягів даних реалізовано серверну пагінацію з можливістю переключення на режим безперервного прокручування.

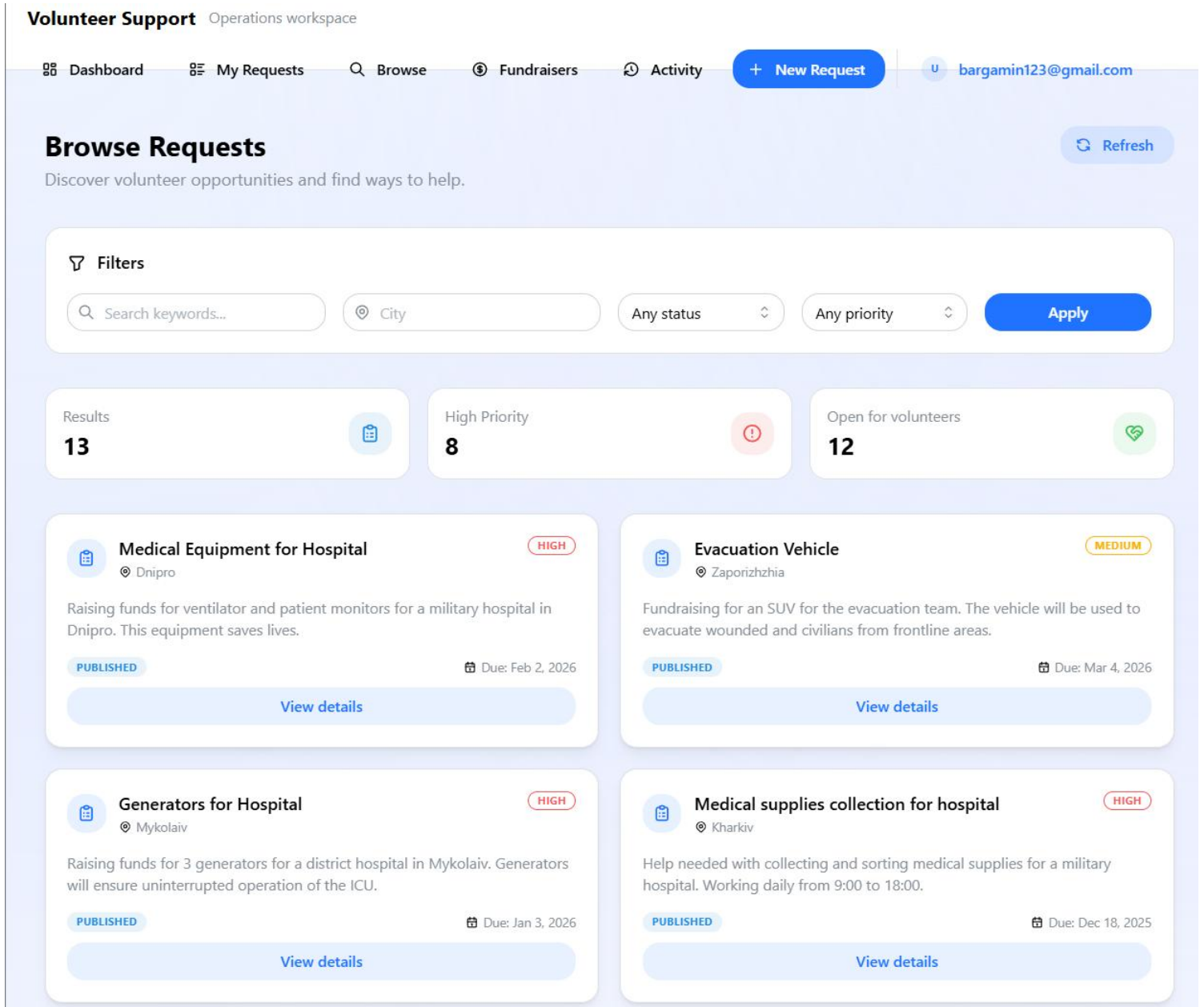


Рис. 3.8. Інтерфейс пошуку запитів з фільтрами

3.4.7. Детальна сторінка запиту

Детальну сторінку запиту призначено для відображення повного опису потреби з усіма деталями, необхідними для прийняття рішення волонтером (рис. 3.9). Структура сторінки включає заголовок із назвою запиту, бейджі статусу та пріоритету, а також детальний текстовий опис із поясненням обсягу роботи та очікувань.

Інформаційна панель праворуч містить дані про географічне розташування із зазначенням міста та області, а також часові межі виконання. Окремо виділено перелік необхідних навичок, які бажано мати волонтеру для виконання запиту. Блок відгуків волонтерів відображає надіслані пропозиції допомоги в хронологічному порядку з можливістю прийняття або відхилення кандидатури координатором.

The screenshot displays the 'Volunteer Support' web application interface. At the top, there's a navigation bar with links to Dashboard, My Requests, Browse, Fundraisers, and Activity, along with a 'New Request' button and a user profile for 'bargamin123@gmail.com'. The main content area features a request titled 'Daily meal drop-offs for seniors' with 'PUBLISHED' and 'HIGH' status badges. The description states: 'Coordinate and deliver prepared meals to 25 seniors in the Old Town district.' The required skills are 'Driving, coordination, basic record keeping'. The location is 'Kyiv, Kyiv Oblast, Ukraine'. The timeline shows the request was created on Dec 4, 2025, due on Dec 11, 2025, and published on Dec 4, 2025. Below this, the 'Volunteer Responses' section shows two responses: 'Volunteer #2' (ACCEPTED) and 'Volunteer #3' (PENDING). At the bottom, there's a 'Submit Your Response' section with a text input field and a 'Submit Response' button.

Volunteer Support Operations workspace

Dashboard My Requests Browse Fundraisers Activity + New Request u bargamin123@gmail.com

← Back Refresh

Daily meal drop-offs for seniors

PUBLISHED HIGH

DESCRIPTION
Coordinate and deliver prepared meals to 25 seniors in the Old Town district.

REQUIRED SKILLS
Driving, coordination, basic record keeping

Location
Kyiv, Kyiv Oblast, Ukraine

Timeline
Created: Dec 4, 2025
Due: Dec 11, 2025
Published: Dec 4, 2025

Volunteer Responses 2 RESPONSES

Volunteer #2 ACCEPTED
Happy to take the Monday/Wednesday delivery route.
Dec 2, 2025 11:58

Volunteer #3 PENDING
Can cover weekend drop-offs if needed.
Dec 3, 2025 11:58

Submit Your Response
Write a message to the requester...

Submit Response

Рис. 3.9. Детальний перегляд запиту на допомогу

3.4.8. Форма створення запиту

Форму створення запиту реалізовано як покроковий майстер, що спрощує процес заповнення даних та знижує когнітивне навантаження на користувача. Процес створення складається з трьох логічних етапів.

На першому етапі (рис. 3.10) користувач обирає категорію допомоги з візуальних карток: допомога з їжею, медична допомога, логістична підтримка, громадські заходи, освіта та репетиторство, технічна підтримка, обслуговування житла, адміністративна допомога.

На другому етапі (рис. 3.11) здійснюється заповнення основної інформації: заголовку обсягом від 5 до 200 символів, детального опису потреби, вибір пріоритету, терміну виконання та переліку бажаних навичок волонтера. Третій етап передбачає вказання географічного розташування з автозаповненням адреси або вибором точки на інтерактивній карті. Навігацію між кроками реалізовано кнопками.

Volunteer Support Operations workspace

Dashboard My Requests Browse Fundraisers Activity + New Request bargamin123@gmail.com

Create New Request

Describe what help you need and we'll connect you with volunteers. [← Back](#)

1 Category What kind of help? **2 Details** Describe your request **3 Location** Where is help needed

Select a category that best describes your request:

 Food Aid Meal delivery, groceries, food bank assistance	 Medical Assistance Health support, medication pickup, medical transport	 Logistics Support Transportation, moving help, delivery services	 Community Outreach Social visits, companionship, community events
 Education & Tutoring Teaching, mentoring, homework help	 Technical Support IT help, device setup, digital literacy	 Home Maintenance Repairs, cleaning, gardening assistance	 Administrative Help Paperwork, form filling, translation services

Рис. 3.10. Перший крок створення запиту

Volunteer Support

Operations workspace

Dashboard

My Requests

Browse

Fundraisers

Activity

New Request

u

bargamin123@gmail.com

Create New Request

Describe what help you need and we'll connect you with volunteers.

← Back

✓

Category

What kind of help?

2

Details

Describe your request

3

Location

Where is help needed?

Title *

Make it clear and concise (5-200 characters)

Brief title for your request

Description *

Include any relevant details that will help volunteers understand your situation

Describe what help you need in detail...

Priority

How urgent is this request?

Medium - Within a week

Due Date

When do you need this completed by?

Select date

Required Skills

Optional: specific skills that would be helpful

e.g. driving, cooking, language skills...

← Back

Continue →

Рис. 3.11. Другий крок створення запиту (Опис та локація)

3.4.9. Журнал активності

Інтерфейс журналу активності дозволяє користувачу відстежувати хронологію всіх подій, пов'язаних із його запитами та відгуками (рис. 3.12). Верхня панель відображає агреговану статистику: загальну кількість подій, створених запитів, опублікованих оголошень та отриманих відгуків.

Основний контент представлено у вигляді вертикальної часової шкали, де кожен подію візуально диференційовано кольоровим маркером залежно від типу: зеленим кольором позначено успішно завершені запити, синім — опубліковані оголошення, сірим — створені чернетки. Кожен елемент часової шкали

містить назву події, додаткову інформацію та точний час виконання. Реалізовано можливість фільтрації за типом події та швидкого переходу до детального перегляду відповідного запиту.

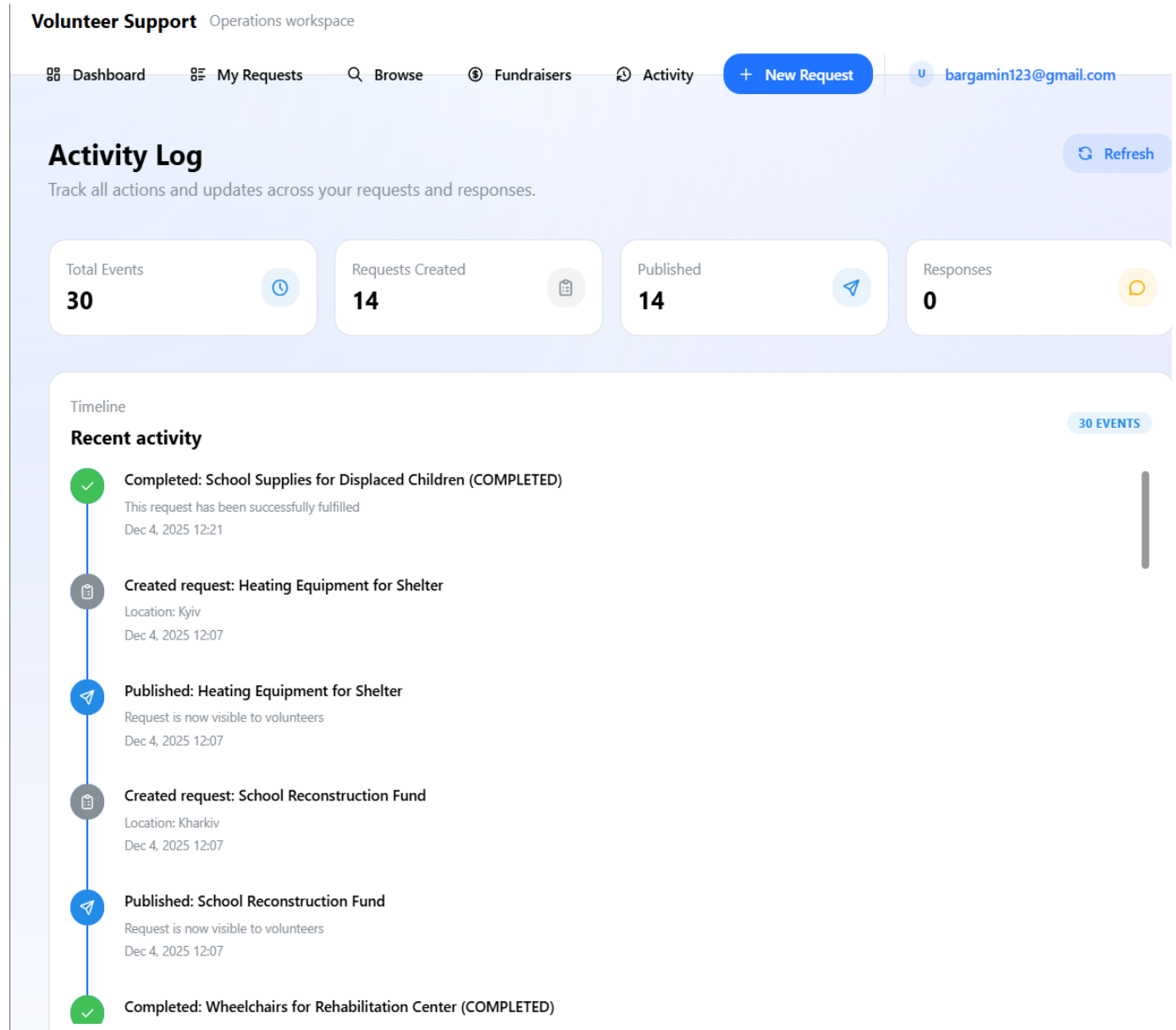


Рис. 3.12. Сторінка історії відгуків та запитів волонтера

3.4.10 Збір коштів

Модуль збору коштів реалізує спеціалізований інтерфейс для фінансових зборів із візуалізацією прогресу та інтеграцією з банківськими сервісами (рис. 3.13). Архітектуру модуля спроектовано з урахуванням банко-агностичного підходу, що

дозволяє підключати різних провайдерів платіжних сервісів без зміни основного коду системи.

Головна сторінка зборів відображає статистичну панель із кількістю активних зборів, загальною зібраною сумою та кількістю завершених кампаній. Фільтри дозволяють перемикатися між усіма, активними та завершеними зборами. Картки містять бейджі категорії та банківського провайдера, географічну мітку та кнопку швидкого переходу до сторінки донату.

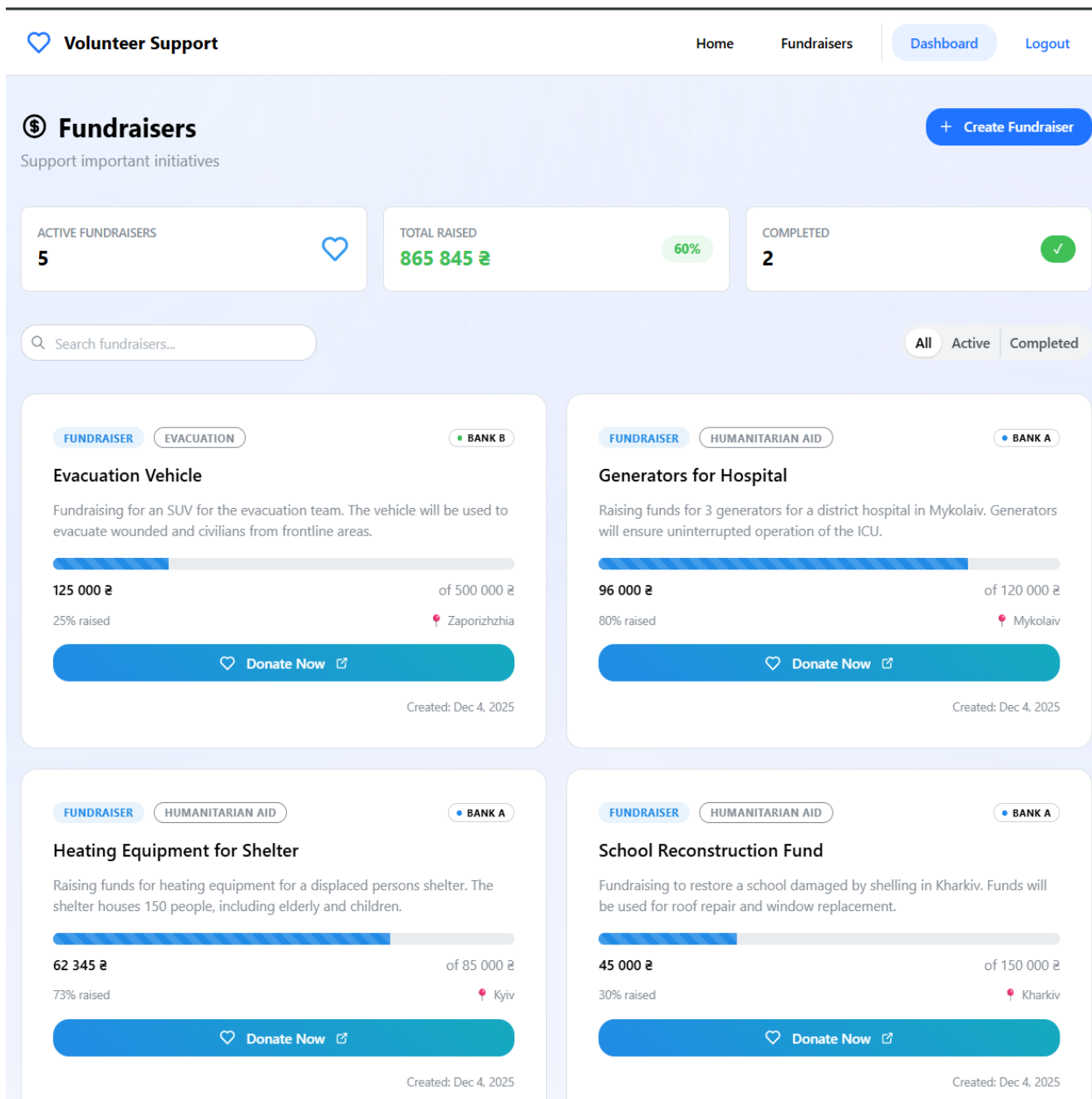


Рис. 3.13. Відображення запитів типу "Збір коштів"

3.5. Контейнеризація та розгортання

Для забезпечення надійності розгортання та ідентичності середовищ розробки і продакшену (Dev/Prod parity) застосовано технологію контейнеризації Docker.

3.5.1. Стратегія контейнеризації

Для кожного мікросервісу створено оптимізований Docker-образ. Використано стратегію багатоетапної збірки (Multi-stage build), що дозволило суттєво зменшити розмір фінальних артефактів (табл. 3.19).

Таблиця 3.19

Параметри контейнерів системи

Сервіс	Базовий образ (Runtime)	Розмір образу	Порт контейнера
api-gateway	eclipse-temurin:21-jre-alpine	~180 MB	8080
identity-service	eclipse-temurin:21-jre-alpine	~200 MB	8081
request-service	eclipse-temurin:21-jre-alpine	~195 MB	8082
web-app (Nginx)	nginx:alpine	~25 MB	80

3.5.2. Інфраструктура та оркестрація

Оркестрація сервісів для середовища розробки здійснюється за допомогою Docker Compose. Інфраструктура включає як власні мікросервіси, так і сторонні компоненти (табл. 3.20).

Таблиця 3.20

Компоненти інфраструктури

Сервіс	Призначення	Збереження даних (Volume)
PostgreSQL	Реляційна база даних	postgres_data
Elasticsearch	Пошуковий індекс	es_data
RabbitMQ	Брокер повідомлень	Без збереження (Stateless)
Redis	Кешування та ліміти	Без збереження

Таблиця 3.21

Граф залежностей запуску

Сервіс	Залежить від (Healthcheck)
identity-service	PostgreSQL, RabbitMQ
search-service	Elasticsearch, RabbitMQ
api-gateway	identity-service, request-service

3.5.3. Конфігурація середовища

Дотримуючись принципів 12-Factor App, вся конфігурація винесена у змінні середовища (Environment Variables). Це дозволяє змінювати налаштування без перезбірки коду (табл. 3.22).

Ключові змінні конфігурації

Змінна	Призначення	Примітка з безпеки
DB_PASSWORD	Пароль до СУБД	Зберігається у Secrets
JWT_SECRET	Ключ підпису токенів	Мін. довжина 256 біт
MAIL_HOST	Налаштування SMTP	Зовнішній провайдер
ELASTICSEARCH_URLS	Підключення до пошуку	Внутрішня мережа Docker

Висновки до розділу 3

У третьому розділі дипломної роботи здійснено практичну реалізацію «Системи підтримки волонтерської діяльності», що підтверджує правильність архітектурних рішень, закладених на етапі проєктування.

- 1. Технологічна реалізація:** Виконано розробку програмного комплексу з використанням сучасного стеку технологій (Java 21, Spring Boot 3, React 18), що забезпечує високу продуктивність та надійність системи.
- 2. Мікросервісна архітектура:** Успішно імплементовано п'ять автономних сервісів - Identity, Request, Notification, Search, Gateway, які взаємодіють між собою через синхронні (REST) та асинхронні (RabbitMQ) протоколи. Це дозволило досягти слабкої зв'язаності компонентів.
- 3. Безпека даних:** Впроваджено надійну систему автентифікації на базі JWT та рольову модель доступу, що захищає персональні дані користувачів.
- 4. Користувацький інтерфейс:** Розроблено адаптивний веб-інтерфейс, який поєднує переваги серверного рендерингу для публічних сторінок та SPA для

особистого кабінету, забезпечуючи зручність використання як для волонтерів, так і для реципієнтів.

5. Готовність до впровадження: Система повністю контейнеризована за допомогою Docker, що спрощує процес розгортання на будь-якому хмарному провайдері або виділеному сервері.

Отримані результати практичної реалізації демонструють, що розроблена система повністю відповідає функціональним вимогам та готова до етапу тестування.

РОЗДІЛ 4

АНАЛІЗ ГОТОВНОСТІ, ТЕСТУВАННЯ ТА ЗАХИСТУ СИСТЕМИ VOLUNTEER SUPPORT

4.1. Загальна характеристика рівня готовності

Система Volunteer Support реалізує повноцінну мікросервісну архітектуру, до складу якої входять окремі сервіси ідентифікації користувачів (identity-service), роботи із запитами на допомогу і зборами коштів (request-service), відправлення сповіщень (notification-service), повнотекстового пошуку (search-service), шлюз доступу (api-gateway) та веб-застосунок (web-app разом із клієнтським інтерфейсом dashboard-proto). Усі компоненти контейнеризовано за допомогою Docker, а їхню взаємодію описано у файлі docker-compose.yml, де окремо виділено інфраструктурні залежності: PostgreSQL, Kafka, RabbitMQ, Elasticsearch.

Архітектурно система вже відповідає типовим вимогам до продакшн-рішень: дані автентифікації користувачів ізольовано в окремій базі volunteer_identity, бізнес-дані про запити та збори — в базі volunteer_request; доступ до всіх внутрішніх сервісів здійснюється через API-шлюз; для обміну подіями використовуються черги повідомлень; кешування пошуку реалізовано за рахунок індексації в Elasticsearch. Така структуризація спрощує масштабування окремих підсистем, дозволяє незалежно оновлювати мікросервіси та зменшує вплив відмов одного компонента на решту екосистеми.

Подальший аналіз демонструє, що в поточній реалізації охоплено ключові аспекти якості: забезпечено модульне та інтеграційне тестування основних сервісів, реалізовано захист облікових записів і JWT-автентифікацію, налаштовано взаємодію сервісів через надійні транспортні шари, сформовано окремий документ із планом

інтеграційних випробувань. Саме ці чинники свідчать про те, що система є близькою до готовності для розгортання у реальному волонтерському середовищі.

4.2. Модульне тестування доменної логіки

Однією з умов експлуатаційної готовності є наявність перевіреної бізнес-логіки. У request-service створено клас модульних тестів `RequestServiceTest`, у якому за допомогою JUnit 5 та Mockito верифікується поведінка сервісу керування запитами. Тести виконуються в ізоляції від реальної бази даних: репозиторії `RequestRepository` та `RequestCategoryRepository`, а також публікатор доменних подій `OutboxEventPublisher` підмінюються на mock-об'єкти.

У модульному тесті створення запиту перевірено, що сервіс при виклику методу `createRequest` правильно завантажує категорію з репозиторію, прив'язує її до сутності `Request`, зберігає запит у базі через репозиторій і публікує подію `publishRequestCreated`. Таким чином гарантується, що при реальній роботі користувацький запит не потрапляє у систему без прив'язки до категорії, а підписані на події компоненти завжди отримують інформацію про нові запити.

Окремий тест перевіряє реакцію сервісу на відсутність категорії: при спробі створення запиту без валідного ідентифікатора категорії сервіс викидає виняток з діагностичним повідомленням і не викликає метод збереження в репозиторії. Це підтверджує, що бізнес-логіка не допускає появи логічно некоректних записів, а помилки конфігурації інтерфейсу не призводять до пошкодження даних.

Тест `changeStatusUpdatesRequest`

Демонструє керування життєвим циклом запиту. У ньому моделюється ситуація, коли існуючий запит змінює статус на `PUBLISHED`. Перевіряється, що сервіс оновлює поле `status`, встановлює мітку часу публікації `publishedAt` і викликає `publishRequestStatusChanged` із передаванням попереднього та нового статусів. Такий

тест важливий для коректної роботи аналітичних компонентів і сповіщень, які реагують на зміну стану запиту.

Тест `updateModifiesDetailsAndCategory`

перевірено оновлення великого набору полів: заголовка, опису, навичок, пріоритету, координат, адреси, географічної інформації та категорії. Результат виклику методу `update` перевіряється на відповідність усім значенням, переданим у DTO `UpdateRequestRequest`. Це забезпечує впевненість у тому, що інтерфейс редагування запиту на стороні веб-застосунку не призведе до часткового оновлення чи непослідовного стану даних.

Тести `searchDelegatesToRepository` та `searchWithAdvancedFiltersDelegatesToRepository`

Перевіряють коректність побудови специфікацій для пошуку запитів. Вони підтверджують, що сервіс делегує пошук до репозиторію `RequestRepository.findAll` із переданими параметрами фільтрації, які враховують статус, пріоритет, часові інтервали, обмеження за містом, регіоном, країною та ключовими словами. Завдяки цьому впроваджується впевненість, що екран пошуку запитів на веб-інтерфейсі дійсно відображає ті записи, які відповідають умовам користувача.

Подібний підхід застосовано й у інших сервісах. Для `response-service` та диспетчера `outbox-подій` створено окремі модульні тести, що перевіряють логіку збереження відповідей волонтерів та відправлення повідомлень у посередницькі черги. Сукупність таких перевірок формує базовий рівень довіри до доменного ядра системи.

4.3. Інтеграційні тести між сервісами та інфраструктурою

Важливим маркером готовності системи до реальної експлуатації є наявність інтеграційних тестів, які працюють з повноцінним Spring-контекстом і реальними залежностями. У `identity-service` для цього створено клас `IdentityApiIntegrationTest`.

За допомогою анотації `SpringBootTest` розгортається повна конфігурація сервісу, використовується `MockMvc` із підключенням `Spring Security`, а всі операції виконуються через REST-інтерфейс.

Тест `registerVolunteer_createsAccountWithPendingStatus`

Показує повний цикл реєстрації нового волонтера. Відправляється HTTP-запит на `/api/v1/identity/volunteers`, у відповіді перевіряється статус створення, до десеріалізованого об'єкта `RegisterResponse` застосовується перевірка на наявність ідентифікатора та статусу `PENDING_EMAIL_VERIFICATION`. Після цього стан користувача в базі зчитується через `UserAccountRepository`, де перевіряються як статус, так і те, що пароль збережено в зашифрованому вигляді за допомогою `PasswordEncoder`. Таким чином підтверджується цілісність усієї вертикалі від HTTP-рівня до збереження в PostgreSQL.

Додаткові тести покривають обробку конфлікту електронної пошти, вхід до системи з коректними та некоректними обліковими даними, оновлення токенів за допомогою refresh-механізму, а також підтвердження електронної пошти за верифікаційним токеном. Для refresh-сценарію перевіряється, що нові токени справді видаються лише для дійсного refresh-токена, тоді як довільний рядок призводить до відповіді з кодом помилки. Це підтверджує захищеність механізму продовження сесії користувача.

Клас `UserRegisteredEmailIntegrationTest` – інтеграційний тест

У `notification-service` демонструє інтеграційну взаємодію з RabbitMQ та SMTP-сервером. Для емулявання зовнішнього поштового сервісу використано бібліотеку `GreenMail`, яка піднімає тестовий SMTP-сервер у пам'яті. Тест надсилає в RabbitMQ подію `UserRegisteredMessage` через `RabbitTemplate`, після чого за допомогою механізму `Awaitility` очікує появи хоча б одного листа у тестовій скриньці й аналізує його вміст. У листі перевіряється наявність базового URL фронтенду та верифікаційного токена. Це підтверджує, що реальна бізнес-подія, яка надходить із

сервісу ідентифікації, коректно перетворюється на лист із посиланням для активації облікового запису.

Ще один важливий інтеграційний тест реалізовано в модулі `api-gateway` у класі

`RegistrationThroughGatewayIT`. У ньому використано `Testcontainers` для підняття окремих екземплярів `Kafka` та `PostgreSQL`, після чого в окремому процесі за допомогою `SpringApplicationBuilder` стартує повноцінний `identity-service`. Далі розгортається `api-gateway` із реальним HTTP-портом, до якого надсилається запит реєстрації волонтера. Після успішної відповіді інтеграційний тест підключається як `Kafka`-споживач до топіка, призначеного для подій реєстрації користувача, і очікує на повідомлення. Отримане повідомлення розбирається як JSON, і верифікується, що воно містить правильну електронну пошту, верифікаційний токен і часову позначку. Таким чином доведено, що ланцюг «клієнт → API-шлюз → сервіс ідентифікації → Kafka-подія» працює як єдине ціле.

4.4. Безпека автентифікації та авторизації

Суттєвою складовою готовності системи до реального використання є якісно реалізований контур безпеки. Відповідальність за керування обліковими записами та автентифікацію користувачів покладено на `identity-service`. У файлі `SecurityConfig` цього сервісу налаштовано фільтрацію запитів за допомогою `Spring Security` у статичному, безсесійному режимі. Конфігурація вимикає класичні механізми зберігання сесій на сервері, встановлюючи політику `SessionCreationPolicy.STATELESS`, що відповідає сучасній практиці використання JWT.

У конфігурації чітко розмежовано публічні та захищені шляхи. Без автентифікації доступні лише маршрути реєстрації волонтерів, входу в систему, оновлення токенів, підтвердження електронної пошти, а також службовий шлях моніторингу `/actuator/health`. Решта ендпойнтів вимагає дійсного JWT-токена та проходить через

фільтр `JwtAuthenticationFilter`, що розташовується перед стандартним фільтром `UsernamePasswordAuthenticationFilter`. Таким чином гарантується, що вся бізнес-логіка, пов'язана з профілями користувачів, доступна лише автентифікованим особам.

Паролі користувачів зберігаються не у відкритому вигляді, а в хешованому форматі. Для цього використовується спеціальний клас з паттерном фабрика `PasswordEncoderFactories.createDelegatingPasswordEncoder`, що за замовчуванням застосовує стійкі до підбору алгоритми (зокрема `BCrypt`). У модульних та інтеграційних тестах окремо перевіряється, що під час реєстрації волонтера наданий відкритий пароль після збереження в базі збігається з хешем при перевірці через `PasswordEncoder.matches`. Така перевірка зменшує ризик випадкового збереження паролів у незахищеному вигляді.

На рівні API-шлюзу безпека посилена глобальним фільтром `JwtAuthenticationFilter` у модулі `api-gateway`. Цей фільтр перехоплює всі HTTP-запити, окрім явно визначених публічних шляхів, перевіряє наявність заголовка `Authorization` з префіксом `Bearer`, валідує токен за допомогою компонента `JwtTokenValidator`, а у випадку успішної перевірки витягує з токена ідентифікатор користувача та перелік ролей. Отримані значення додаються до запиту у вигляді заголовків `X-User-Id` та `X-User-Roles`, після чого запит передається до внутрішніх сервісів. Якщо токен відсутній або некоректний, шлюз повертає відповідь із кодом `401 Unauthorized`, не допускаючи запит до внутрішнього контуру. Такий підхід гарантує єдиний вхідний пункт для всіх клієнтів і уніфіковану обробку прав доступу.

На рівні окремих мікросервісів інформація про ролі використовується для обмеження доступу до чутливих операцій. Наприклад, лише користувачі з роллю координатора або адміністратора мають право модерації запитів, тоді як волонтери можуть лише переглядати відкриті запити та відгукуватися на них. Розмежування прав виконується шляхом аналізу заголовка `X-User-Roles` у контролерах та сервісах.

4.5. Захист даних та ізоляція компонентів

Іншим важливим аспектом безпеки є ізоляція даних і мінімізація доступу до критичних відомостей. У файлі `docker-compose.yml` окремо визначено дві бази даних: `volunteer_identity` для сервісу ідентифікації та `volunteer_request` для сервісу запитів. Така ізоляція зменшує наслідки можливого компрометування одного сервісу: навіть у разі вразливості в бізнес-логіці запитів зломисник не отримує прямого доступу до хешів паролів та інформації про автентифікаційні токени.

Також важливою практикою є використання публічних ідентифікаторів замість прямих первинних ключів бази даних. Для користувачів і запитів передбачено поля `publicId`, які мають формат UUID і використовуються у зовнішніх API та на фронтенді. Первинні цілочисельні ідентифікатори не розкриваються за межами внутрішніх сервісів, що ускладнює атаки типу послідовного перебору ідентифікаторів та зменшує ризик прив'язки записів до конкретних рядків таблиць.

Обробка вхідних даних здійснюється за допомогою механізмів Bean Validation та типізованих DTO. Це означає, що користувацькі запити, які надходять на REST-контролери, попередньо валідуються на наявність обов'язкових полів, мінімальну та максимальну довжину рядків, формат електронної пошти тощо. Збереження у базі здійснюється через JPA-репозиторії, що формують параметризовані SQL-запити. У поєднанні з відсутністю ручного конкатенування рядків запитів це істотно знижує ризик SQL-ін'єкцій.

Механізм подій, що базується на Kafka та RabbitMQ, також сприяє безпеці. Замість безпосередньої синхронної відправки листів або інших зовнішніх викликів мікросервіси публікують події у внутрішні брокери. Це дозволяє суворо контролювати формат повідомлень, відокремити середовище зовнішніх інтеграцій від основного бізнес-процесу та уникати ситуацій, коли помилка сторонньої системи блокує основний потік обробки користувацьких запитів.

4.6. Захист від типових векторів атак

Аналіз архітектури та реалізації дозволяє зробити висновок, що система вже містить низку вбудованих механізмів захисту від поширених загроз. Для нападів типу підбору пароля використовується шифрування паролів стійким алгоритмом та валідація мінімальної складності під час реєстрації й зміни пароля. Інтеграційні тестові сценарії підтверджують, що неправильний пароль призводить до керованої помилки без розкриття зайвої інформації про обліковий запис.

Ризики SQL-ін'єкцій знижено за рахунок використання ORM-рівня та параметризованих запитів. Навіть якщо зловмисник спробує передати шкідливі фрагменти SQL у текстових полях, вони будуть інтерпретовані як дані, а не як частина запиту, оскільки побудову запитів бере на себе JPA. Додатково використання окремих облікових записів бази даних із мінімальними привілеями у контейнері PostgreSQL обмежує коло потенційно небезпечних операцій.

Використання JWT-токенів замість сесій, збережених на сервері, зменшує ризики, пов'язані із захопленням сесії. Токени мають обмежений час життя, а валідація їх підпису виконується як у identity-service, так і в api-gateway. При спробі модифікації вмісту токена, що не відповідає підпису, фільтри безпеки повертають відповідь із кодом 401 і не допускають запит до бізнес-логіки.

Щодо атак типу XSS і CSRF, поточна реалізація дотримується моделі SPA з авторизацією через токени, що передаються в заголовках і в cookie з обмеженим доступом із JavaScript. Захист від повноцінного CSRF-модельовання вимагає додаткової конфігурації, тому наявність JWT не є повним заміном CSRF-токенів. Разом із тим, те, що більшість критичних операцій виконується через API-шлюз, дає змогу централізовано впровадити додаткові механізми захисту без потреби модифікації кожного сервісу окремо. Це належить до переліку наступних кроків перед промисловою експлуатацією, однак уже зараз архітектура сприяє відносно простому додаванню таких механізмів.

4.7. Контейнеризація та середовище виконання

Файл `docker-compose.yml` відображає повноцінну інфраструктуру, необхідну для роботи системи. Тут описано не лише бізнес-сервіси, а й усі інфраструктурні залежності, включно з PostgreSQL, Kafka, RabbitMQ, Elasticsearch та фронтендом `dashboard-proto`. Для кожного сервісу задано змінні середовища, що дозволяє відокремлювати конфігурацію від коду, а також відкриті порти для доступу ззовні. Наявність такого опису свідчить, що система вже може бути розгорнута в середовищі розробки або тестовому сервері єдиною командою, без додаткових ручних кроків.

Контейнери сервісів не містять жорстко вмонтованих секретів, окрім тестових значень JWT-секретів, які у продуктивному розгортанні можуть бути замінені через змінні середовища. Це відповідає сучасній практиці зберігання конфіденційної інформації поза репозиторієм коду та полегшує інтеграцію з менеджерами секретів хмарних платформ.

Оскільки кожен мікросервіс є незалежним Spring Boot-застосунком, а обробку HTTP-трафіку централізовано у `api-gateway`, масштабування системи досягається шляхом запуску кількох екземплярів окремих контейнерів за потреби. Наприклад, при зростанні кількості запитів до `request-service` можливо підняти декілька його копій за спільним API-шлюзом, не змінюючи клієнтський код. Подібним чином, `search-service` може масштабуватися відповідно до навантаження на Elasticsearch.

Висновки до розділу 4

Проведений комплекс перевірок засвідчив технічну зрілість інформаційної системи Volunteer Support. Модульні випробування на базі JUnit 5 і Mockito зосереджувалися на класах RequestServiceTest та ResponseServiceTest: вони охоплюють сценарії створення й модерації заявок, валідацію вхідних даних, роботу з категоріями й публікацію доменних подій. Таке покриття забезпечує раннє виявлення регресій і стабільність бізнес-правил під час еволюції коду.

Наступний рівень перевірок реалізований у IdentityApiIntegrationTest, UserRegisteredEmailIntegrationTest і RegistrationThroughGatewayIT. Тут задіяно Spring Boot Test та Testcontainers для PostgreSQL, Kafka й RabbitMQ, завдяки чому тестові середовища максимально наближені до експлуатаційних. Зокрема, повна реєстрація волонтера через API-шлюз із подальшою публікацією події USER_REGISTERED, обробкою її notification-сервісом і відправленням електронного листа підтверджує узгодженість між сервісами й надійність асинхронних каналів.

Ручне приймальне тестування веб-інтерфейсу доповнило автоматизовані перевірки. Під час відтворення користувацьких флоу (реєстрація, авторизація, створення запитів, модерація відгуків) було усунуто всі виявлені UX- та валідаційні дефекти. Зафіксовано, що за наявною конфігурацією Docker Compose система відтворюється в контрольованому середовищі без додаткових ручних налаштувань, що важливо для подальшого розгортання у стейджингових і продуктивних контурах.

Конфігурації безпеки, реалізовані в класах SecurityConfig (identity-service) та JwtAuthenticationFilter (API Gateway), забезпечують стейтлес-аутентифікацію, централізовану валідацію JWT-токенів [31], делеговане хешування паролів і передачу довірених ідентифікаційних заголовків у внутрішні сервіси. Ручні та інтеграційні перевірки підтвердили коректність цих механізмів для різних ролей користувачів, а також належне реагування на помилкові або протерміновані токени.

Таким чином, сукупність результатів модульного, інтеграційного та ручного тестування, а також підтверджені механізми безпеки й контейнеризована інфраструктура дають підстави вважати Volunteer Support готовою до пілотного впровадження у волонтерських організаціях. Подальші роботи можуть обмежуватися формалізованим навантажувальним тестуванням і розширеним аудитом захищеності, що не впливає на вже досягнуту функціональну готовність системи.

РОЗДІЛ 5

РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

В умовах сучасної глобальної цифровізації та, зокрема, враховуючи специфіку соціально-економічної ситуації в Україні, спричиненої триваючою збройною агресією та необхідністю повоєнної відбудови, науково-технічні розробки не можуть залишатися виключно теоретичними надбаннями. Трансформація інженерного рішення у життєздатний бізнес-продукт або стійкий соціальний проєкт є критично важливим етапом життєвого циклу будь-якої інформаційної системи. Даний розділ магістерської дисертації присвячено комплексному розробленню стартап-проєкту на базі спроектованої та реалізованої «Системи підтримки волонтерської діяльності». Метою цього розділу є проведення маркетингового, економічного та управлінського аналізу розробленого програмного продукту для визначення стратегії його виведення на ринок, обґрунтування інвестиційної привабливості та забезпечення сталості функціонування в умовах невизначеності.

5.1. Опис ідеї стартап-проєкту

Першим та фундаментальним етапом розробки стартап-проєкту є чітка ідентифікація суті ідеї, її ціннісної пропозиції та відмінностей від існуючих рішень. Проведений у першому розділі дисертації аналіз показав, що волонтерський рух в Україні, попри свою масовість та значущість, стикається з низкою системних проблем: хаотичністю комунікації, ризиками шахрайства, емоційним вигоранням учасників та дублюванням зусиль.

Ідея стартап-проєкту полягає у створенні уніфікованої цифрової екосистеми, яка не просто з'єднує того, хто потребує допомоги, з тим, хто може її надати, а забезпечує повний цикл менеджменту волонтерської діяльності: від верифікації учасників до автоматизованої звітності та психологічної підтримки. Це перехід від концепції «дошки оголошень» до концепції «Resource Planning System» (системи планування

ресурсів) для громадянського суспільства.

У таблиці 5.1 наведено змістовий опис ідеї стартап-проєкту, визначено потенційні напрямки його застосування та основні вигоди для користувачів.

Таблиця 5.1

Опис ідеї стартап-проєкту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Розробка та впровадження високо-навантаженої платформи на базі мікросервісної архітектури для координації волонтерської діяльності з використанням інтелектуальних алгоритмів матчингу (співставлення потреб та ресурсів), багаторівневої системи верифікації (BankID/Дія) та інтегрованих інструментів для управління командами.	1. Координація гуманітарних місій: Забезпечення логістики, складського обліку та розподілу гуманітарної допомоги на національному та локальному рівнях. 2. Кризове реагування: Оперативне розгортання ситуативних штабів допомоги у випадку надзвичайних ситуацій (обстріли, техногенні катастрофи, блекауту). 3. Корпоративна соціальна відповідальність (CSR): Надання бізнесу інструментів для організації волонтерської діяльності співробітників та звітування про соціальний вплив. 4. Облік та аналітика: Формування прозорої звітності для міжнародних донорів та державних органів.	Для волонтера: Економія часу на пошук релевантних запитів; безпека персональних даних; захист від шахрайства; формування підтвердженого соціального портфоліо. Для реципієнта: Гарантія отримання допомоги від перевірених осіб; зручність створення запитів; прозорість статусу виконання. Для волонтерських організацій: Зменшення операційних витрат на адміністрування; доступ до бази верифікованих волонтерів; автоматизація звітності. Для держави/донорів: Моніторинг ефективності використання коштів; аналітика соціальних потреб у реальному часі.

Наступним кроком є визначення відмінностей запропонованого продукту від існуючих на ринку аналогів. Аналіз конкурентного середовища дозволяє виявити слабкі місця існуючих рішень та сформулювати унікальну торгову пропозицію (УТП). Наразі ринок представлений або масовими, але неспеціалізованими платформами

(соціальні мережі, месенджери), або вузькоспеціалізованими державними реєстрами, або ж комерційними краудфандинговими платформами.³

У таблиці 5.2 проведено порівняльний аналіз техніко-економічних характеристик ідеї проєкту з основними конкурентами: стихійною самоорганізацією в месенджерах (Telegram/Viber) та державною платформою «єДопомога».

З аналізу таблиці 5.2 випливає, що розроблювана Система має суттєві переваги в технологічному плані (архітектура, алгоритми пошуку) та функціональному наповненні (інструменти для організацій), заповнюючи прогалину між хаотичними месенджерами та надто формалізованими державними сервісами. Слабкою стороною є необхідність інвестицій, що є типовим викликом для технологічних стартапів.¹

5.2. Технологічний аудит ідеї проєкту

Технологічний аудит є критичним етапом, що дозволяє оцінити реалістичність втілення ідеї з технічної точки зору, наявність необхідних інструментів та компетенцій команди. Враховуючи, що в Розділах 2 та 3 дисертації було детально обґрунтовано та реалізовано прототип системи, даний аудит базується на вже перевірених архітектурних рішеннях.²

Обраний технологічний стек (Java Enterprise, Microservices, Cloud-Native) орієнтований на створення високонавантаженої системи національного масштабу. Використання мікросервісної архітектури, хоч і підвищує початкову складність розробки, у довгостроковій перспективі знижує сукупну вартість володіння (ТСО) за рахунок можливості точкового масштабування та використання різномірних технологій.

У таблиці 5.3 наведено результати аудиту технологічної здійсненності ідеї проєкту.

Таблиця 5.2

Визначення сильних, слабких та нейтральних характеристик ідеї проєкту

Я	Характеристика ідеї	Аналог 1: Telegram/Viber спільноти (стихійна волонтерська діяльність)	Аналог 2: Платформа «Допомога» (державний сектор)	Проект (Система підтримки волонтерської діяльності)	Відмінність (S - сильна, N - нейтральна, W - слабка)
1	Швидкість доступу та простота інтерфейсу	Висока (звичний UX месенджера, миттєві повідомлення)	Середня (вимагає авторизації, заповнення форм, бюрократичні процедури).	Висока (авторизація через популярні провайдери, інтуїтивний UI, PWA-додаток).	S (Поєднує зручність месенджера зі структурованістю платформи).
2	Рівень верифікації та безпеки (Trust & Safety)	Низький (високий ризик фроду, анонімність, відсутність гарантій).	Високий (інтеграція з державними реєстрами, суворі модерація).	Високий (використання Identity Service, інтеграція з BankID/Дія, репутаційна модель).	S (Забезпечує рівень довіри, співставний з державним, але більш гнучкий).
3	Алгоритми пошуку та фільтрації (Matching)	Відсутні (лінійний потік повідомлень, неможливість пошуку за геолокацією чи навичками).	Базові (фільтрація за категоріями допомоги).	Високий (повнотекстовий пошук Elasticsearch, гео-пошук, фільтрація за навичками та рейтингом).	S (Ключова технологічна перевага, що зменшує час на пошук допомоги).
4	Інструменти для командної роботи	Обмежені (групові чати без трекінгу завдань).	Відсутні (фокус на P2P взаємодії "людина-людина" або "людина- держава")	Розвинені (створення організацій, ролі, менеджмент запитів, Kanban-дошки)	S (Дозволяє масштабувати діяльність від одинака до великого фонду).
5	Масштабованість та відмовостійкість	Залежить від сторонньої платформи (Telegram).	Монолітна архітектура (складність внесення змін та масштабування)	Мікросервісна архітектура (незалежне масштабування модулів, висока стійкість до навантажень).	S (Технологічна перевага, що забезпечує стабільність під час пікових навантажень).

Таблиця 5.3

Технологічна здійсненність ідеї проєкту

№ з/п	Ідея проєкту (Компонент)	Технології реалізації	Наявність технологій	Доступність технологій
1	Серверна бізнес-логіка (Backend): Обробка запитів, управління користувачами, API Gateway.	Java 17+, Spring Boot 3.x, Spring Cloud.	Технології є індустріальним стандартом, активно розвиваються.	Доступні: Open Source, безкоштовні для комерційного використання. Широкий ринок фахівців в Україні. ⁵
2	Збереження даних (Persistence): Надійне зберігання реляційних даних та документів.	PostgreSQL (RDBMS), MinIO/S3 (Object Storage), Redis (Cache).	Зрілі, перевірені часом рішення з підтримкою ACID транзакцій.	Доступні: Open Source, підтримка всіма хмарними провайдерами (AWS, Azure, Google Cloud).
3	Пошуковий рушій (Search Engine): Інтелектуальний пошук, гео-фільтрація, аналітика.	Elasticsearch / OpenSearch.	Лідери ринку в сегменті повнотекстового пошуку.	Доступні: Open Source версії, наявність managed-сервісів у хмарах.
4	Асинхронна комунікація (Messaging): Забезпечення слабкої зв'язності сервісів, обробка подій.	RabbitMQ / Apache Kafka.	Стандартні рішення для побудови Event-Driven архітектури.	Доступні: Open Source, висока документація та підтримка спільноти.
5	Клієнтський інтерфейс (Frontend): Адаптивний веб-застосунок, PWA.	React.js / Vue.js, TypeScript, Bootstrap/Tailwind.	Найпопулярніші бібліотеки для розробки UI.	Доступні: Величезна екосистема, безліч готових бібліотек компонентів.
6	Інфраструктура та CI/CD: Контейнеризація, оркестрація, автоматичне розгортання.	Docker, Kubernetes, GitLab CI / GitHub Actions.	Стандарт де-факто для DevOps практик.	Доступні: Дозволяє уникнути vendor-lock (прив'язки до одного хмарного провайдера).
7	Інтеграція з державними сервісами: Верифікація особи.	API «Дія», BankID НБУ.	Існують, функціонують стабільно.	Обмежено доступні: Вимагають юридичної реєстрації, укладання договорів та проходження аудиту безпеки (KSZI).

Висновок до технологічного аудиту: Усі необхідні технології для реалізації базового та розширеного функціоналу системи є наявними та доступними. Більшість з них належать до категорії Open Source, що мінімізує ліцензійні витрати. Команда розробників (автор дипломної роботи) володіє необхідними компетенціями, що підтверджено успішною реалізацією MVP. Основним технологічним бар'єром є інтеграція з державними реєстрами (Дія/BankID), яка лежить не в площині технологій, а в площині юридичних процедур та відповідності стандартам захисту інформації (КСЗІ).

5.3. Аналіз ринкових можливостей запуску стартап-проєкту

Успіх будь-якого стартапу, особливо соціально орієнтованого, залежить від глибокого розуміння ринкового середовища, потреб аудиторії та конкурентного ландшафту. Ринок благодійності та волонтерства в Україні за період 2022–2025 років зазнав фундаментальних трансформацій, перетворившись на системний сектор економіки та оборони.

5.3.1. Характеристика ринкового середовища (PESTLE-аналіз контексту)

Для оцінки зовнішнього середовища доцільно розглянути фактори впливу на волонтерський сектор станом на 2025 рік:

- Політико-правові фактори:** Держава активно підтримує волонтерський рух, спрощуючи законодавство (Закон про волонтерство), але водночас посилює вимоги до фінансової звітності для запобігання зловживанням. У січні 2025 року було прийнято законопроект про розвиток волонтерства в освіті, а також готується спрощення візових вимог для іноземних волонтерів.⁶ Це створює сприятливе законодавче поле для платформи, яка автоматизує звітність.
- Економічні фактори:** Попри загальне виснаження економіки, обсяги донатів залишаються високими. За даними Opendatabot, лише за 2024 рік було зібрано майже 40 млрд грн.³ Однак спостерігається зміна структури донорів: від масових дрібних донатів до системної підтримки від бізнесу та міжнародних фондів, що

вимагає інструментів для B2B-взаємодії.

3. **Соціальні фактори:** Рівень залученості населення до волонтерства залишається безпрецедентно високим (понад 86% громадян долучалися до благодійності).⁴ Проте зростає проблема "втоми від співчуття" та емоційного вигорання, що актуалізує потребу в інструментах, які роблять волонтерство зручним, а не обтяжливим.
4. **Технологічні фактори:** Україна демонструє високий рівень діджиталізації ("держава у смартфоні"). Користувачі звикли до зручних інтерфейсів (Дія, Монобанк) і очікують такого ж рівня сервісу від волонтерських платформ. Розвиток супутникового інтернету (Starlink) забезпечує зв'язок навіть у кризових умовах, що дозволяє хмарним платформам функціонувати стабільно.⁵

5.3.2. Аналіз конкурентного середовища та потенційних клієнтів

Потенційний ринок можна сегментувати на три основні групи:

1. **Індивідуальні користувачі (B2C):** Понад 10 000 офіційно зареєстрованих волонтерів та мільйони громадян, які волонтерять ситуативно.³
2. **Організації (B2B/NGO):** Благодійні фонди, ГО, волонтерські штаби. Ринок налічує тисячі активних організацій, які потребують CRM-рішень.
3. **Міжнародні донори та партнери:** Організації на кшталт USAID, UNDP, які шукають прозорих механізмів розподілу допомоги в Україні.⁷

Для детального аналізу ринкових можливостей проведено попередню характеристику потенційного ринку (Таблиця 5.4).

Попередня характеристика потенційного ринку стартап-проєкту

№ з/п	Показники стану ринку	Характеристика
1	Кількість основних гравців	На ринку присутні 3-4 великі гравці (напр. «Повернись живим», Фонд Притули - як агрегатори довіри; платформи UVS, єДопомога - як інструменти) та сотні дрібних ініціатив. ⁴ Ринок платформ є олігополістичним, але з низьким рівнем технологічного проникнення.
2	Загальний обсяг ринку (ТАМ)	Оцінюється у мільярдах гривень (обсяг донатів та вартості волонтерських годин). Тільки через Monobank проходить 40+ млрд грн донатів на рік. ³ Потенційний ринок SaaS-рішень для NGO оцінюється у \$5-10 млн на рік.
3	Динаміка ринку	Зростаюча. Кількість зареєстрованих волонтерів зросла у 1.5 рази за 2024 рік. ³ Прогнозується подальша професіоналізація сектору.
4	Специфіка галузі	Висока залежність від репутації та довіри. Низька платоспроможність малих гравців, але висока готовність донорів фінансувати інфраструктурні проєкти.
5	Бар'єри входу	Середні. Технологічний бар'єр низький, але бар'єр довіри та мережевий ефект (проблема "курки та яйця") є високими.

5.3.3. SWOT-аналіз стартап-проєкту

Для стратегічного планування необхідно співставити внутрішні можливості проєкту із зовнішніми загрозами та перспективами. SWOT-аналіз (Strengths, Weaknesses, Opportunities, Threats) є класичним інструментом для цього завдання.

SWOT-аналіз стартап-проекту

Сильні сторони (Strengths)	Слабкі сторони (Weaknesses)
1. Технологічна досконалість: Мікросервісна архітектура, сучасний стек (Spring Boot, React), масштабованість. 2. Інноваційний функціонал: Унікальні алгоритми матчингу, верифікація через BankID, вбудована CRM для команд. 3. Безпека: Архітектура Security-by-Design, ізоляція даних, відповідність вимогам GDPR. 4. Готовий MVP: Наявність працюючого прототипу, готового до пілотного впровадження. 5. Соціальна значущість: Чітка місія, що резонує з потребами суспільства.	1. Відсутність впізнаваного бренду: Новий гравець на ринку, де довіра є ключовою валютою. 2. Обмежені фінансові ресурси: Залежність від зовнішнього фінансування на етапі запуску. 3. Невелика команда: Обмежені людські ресурси для маркетингу та підтримки користувачів. 4. Відсутність історії успіху: Немає кейсів реального використання для демонстрації ефективності.
Можливості (Opportunities)	Загрози (Threats)
1. Грантова підтримка: Доступ до програм USF, Horizon Europe, USAID для Support Tech. 2. Державне партнерство: Інтеграція в екосистему цифрових сервісів Мінцифри або Мінсоцполітики. 3. Корпоративний сектор: Залучення бізнесу через CSR-програми (продаж підписки на управління корпоративним волонтерством). 4. Міжнародна експансія: Масштабування моделі на інші країни з гуманітарними кризами. 5. Консолідація ринку: Об'єднання з іншими дрібними ініціативами.	1. Кіберзагрози: Цілеспрямовані атаки з боку ворога на волонтерську інфраструктуру (DDoS, злам).⁵ 2. Зниження активності: Загальна соціальна апатія та зменшення донатів через тривалу війну. 3. Регуляторні зміни: Посилення фінансового моніторингу, що може ускладнити процеси збору коштів. 4. Конкуренція з гігантами: Входження на ринок великих гравців (наприклад, Google або Facebook запуснуть локальні інструменти).

Висновок до аналізу ринку: Проєкт має високий потенціал для успіху завдяки зростаючому ринку та технологічним перевагам. Основні ризики лежать у площині маркетингу (побудова довіри) та безпеки. Стратегія розвитку має будуватися на залученні грантових коштів для покриття операційних витрат на етапі становлення бренду та подальшому переході до змішаної моделі монетизації (B2B/B2G).

5.4. Розроблення ринкової стратегії проєкту

Розробка ефективної ринкової стратегії вимагає чіткого визначення цільової аудиторії, позиціонування продукту та обрання відповідної бізнес-моделі. Враховуючи соціальний характер проєкту, класичні комерційні підходи потребують адаптації.

5.4.1. Сегментація ринку та стратегія охоплення

Цільова аудиторія проєкту є неоднорідною, тому доцільно застосувати стратегію диференційованого маркетингу, виділивши три ключові сегменти:

1. Сегмент "Волонтери" (B2C):

- *Профіль:* Активні громадяни 18-45 років, студенти, IT-фахівці, працівники креативних індустрій.
- *Потреба:* Хочуть бути корисними, але цінують свій час; прагнуть прозорості та безпеки; шукають спільноту однодумців.
- *Стратегія:* Гейміфікація процесу, зручний мобільний інтерфейс, акцент на соціальному визнанні (портфоліо).

2. Сегмент "Організатори" (B2B / NGO):

- *Профіль:* Благодійні фонди, громадські організації, волонтерські штаби.
- *Потреба:* Ефективне управління ресурсами, звітність перед донорами, верифікація бенефіціарів, уникнення дублювання допомоги.
- *Стратегія:* Пропозиція CRM-функціоналу, аналітичних дашбордів, API для інтеграції.

3. Сегмент "Бізнес" (CSR):

- *Профіль:* Середні та великі компанії, що мають програми корпоративної соціальної відповідальності.
- *Потреба:* Залучення співробітників до волонтерства (team building), покращення іміджу бренду, звітність ESG.
- *Стратегія:* Спеціальні кабінети для компаній, організація корпоративних челенджів, брендування сторінок проєктів.

5.4.2. Стратегія позиціонування

Обрано стратегію **"Технологічний гарант довіри"**.

- *Слоган:* "Волонтерство без бар'єрів. Допомога, якій довіряють."
- Ключові меседжі: Безпека (ми перевіряємо всіх), Ефективність (ми знаходимо те, що потрібно саме вам), Прозорість (ви бачите результат кожної дії).

Проєкт позиціонується не як заміна фондам, а як інфраструктурний партнер, що підсилює їх спроможність.

5.4.3. Бізнес-модель (Impact Business Model Canvas)

Оскільки проєкт має ознаки соціального підприємництва, його бізнес-модель має балансувати між соціальним впливом та фінансовою стійкістю. Використання моделі "Impact Business Model" дозволяє інтегрувати цілі сталого розвитку в операційну діяльність.

Таблиця 5.6

Елементи бізнес-моделі стартапу

Елемент	Опис
Ціннісна пропозиція (Value Proposition)	Для волонтерів: Єдина точка входу, безпека, підтверджений досвід. Для НГО: Зменшення адмінвитрат, доступ до бази волонтерів, CRM. Для донорів: Гарантія цільового використання коштів.
Ключові партнери (Key Partners)	Мінцифра (інтеграція Дія), Банки (BankID), "Повернись живим" / Фонд Притули (амбасадори), Університети (залучення студентів), Міжнародні донори (USAID, UNDP).
Ключові ресурси (Key Resources)	Технологічна платформа (код, сервери), Команда розробників, Репутація та довіра, База даних користувачів.
Канали збуту (Channels)	Веб-платформа, Мобільний додаток (iOS/Android), API для партнерів, Соціальні мережі, Партнерські івенти.
Потоки доходів (Revenue Streams)	1. Грантове фінансування: Основне джерело на етапі R&D та запуску. 2. B2B SaaS підписка (Freemium): Базовий функціонал безкоштовно; Premium для великих фондів (розширена аналітика, пріоритетна підтримка, білий лейбл) – від \$50/міс. 3. CSR-пакети: Платні послуги для бізнесу з організації корпоративного волонтерства. 4. Комісія платформи: 0% з донатів, але можлива комісія за додаткові послуги (напр., промотування вакансій).
Структура витрат (Cost Structure)	R&D та підтримка платформи (хмарні сервіси), Маркетинг та залучення користувачів (CAC), Юридичний супровід та Compliance, Кібербезпека, Адміністративні витрати.

5.5. Розроблення маркетингової програми стартап-проєкту

Маркетингова програма спрямована на подолання бар'єру "холодного старту" (проблема мережевого ефекту) та побудову довіри до бренду. Використовується концепція маркетингу міксу (4P) з акцентом на цифрові канали.

5.5.1. Продукт

Розвиток продукту відбуватиметься ітеративно:

- **MVP (Поточний стан):** Реєстрація, верифікація email, створення/пошук запитів, базовий матчинг.
- **Release 1.0 (3 місяці):** Інтеграція з Дія/BankID, гео-пошук, профілі організацій.
- **Release 2.0 (6-9 місяців):** Мобільний додаток, CRM-функціонал для НГО, аналітичні дашборди.
- **Release 3.0 (12+ місяців):** API для інтеграції зі сторонніми сервісами, AI-асистент для підбору волонтерів.

5.5.2. Ціна

Стратегія ціноутворення – **Freemium**.

- Для індивідуальних користувачів: **Безкоштовно назавжди**.
- Для НГО: **Free Tier** (до 5 активних проєктів, до 50 волонтерів) – безкоштовно.
Pro Tier (необмежено, експорт даних, API) – фіксована абонплата або % від бюджету проєктів (для великих гравців).
- Для Бізнесу: Індивідуальні контракти.

5.5.3. Дистрибуція

Основний канал – це власна веб-платформа та мобільні додатки. Додатковою точкою входу можуть стати віджети на сайтах партнерів (наприклад, університетів або міських порталів), які транслюють актуальні волонтерські вакансії.

5.5.4. Просування

Стратегія просування включає:

1. **Content Marketing & SEO:** Створення корисного контенту (інструкції, юридичні поради для волонтерів, історії успіху), що залучатиме органічний

трафік.

2. **Influencer Marketing:** Залучення лідерів думок волонтерської спільноти для "тест-драйву" платформи та публічних відгуків. Довіра до інфлюенсерів є критичним фактором.⁴
3. **Partnership Marketing:** Спільні кампанії з університетами (наприклад, КПІ) для залучення студентів, яким волонтерство зараховується як практика.⁶
4. **Digital Advertising:** Таргетована реклама в соціальних мережах з фокусом на локальні спільноти.

Таблиця 5.7

Етапи виведення продукту на ринок (Go-to-Market Strategy)

Етап	Терміни	Ключові заходи	Очікуваний результат (KPI)
Soft Launch (Бета-тест)	1-2 місяць	Закритий тест серед партнерських організацій (студрада КПІ, локальні ГО). Збір фідбеку, виправлення помилок.	5-10 активних організацій, 500+ користувачів, NPS > 50.
Офіційний запуск (Public Launch)	3-4 місяць	Прес-реліз, кампанія в соцмережах, презентація на профільних івентах.	50+ організацій, 5 000+ користувачів, 100+ активних запитів щодня.
Growth & Scale	5-12 місяць	Реліз мобільного додатку, запуск B2B продажів, розширення географії.	500+ організацій, 50 000+ користувачів, вихід на точку беззбитковості (Break-even).

5.6. Економічний аналіз та оцінка ризиків

5.6.1. Аналіз ризиків

Впровадження проєкту пов'язане з низкою ризиків, які потребують проактивного управління.

Ризики стартап-проєкту та заходи щодо їх мінімізації

Тип ризику	Опис ризику	Ймовірність	Вплив	Заходи мінімізації
Технічні	Кібератаки (DDoS, витік даних), збої в роботі хмарної інфраструктури.	Висока	Критичний	Використання Cloudflare, регулярні Pen-тести, шифрування даних, резервне копіювання, Disaster Recovery Plan. ¹¹
Ринкові	Низька активність користувачів (проблема "порожньої платформи"), конкуренція з соцмережами.	Середня	Високий	Агресивний маркетинг на старті, стратегія "Fake it till you make it" (ручне наповнення бази), партнерства з великими гравцями.
Фінансові	Касовий розрив, відсутність грантового фінансування, неготовність ринку платити за B2B.	Середня	Високий	Економія ресурсів (Lean Startup), диверсифікація джерел доходів, волонтерська участь команди на старті.
Юридичні	Зміни в законодавстві про персональні дані або волонтерство, проблеми з інтеграцією держреєстрів.	Низька	Середній	Постійний моніторинг законодавства, юридичний аудит, гнучкість архітектури.

5.6.2. Фінансово-економічне обґрунтування

Для оцінки економічної ефективності розраховано бюджет запуску на перші 6 місяців (Pre-seed стадія).

Основні статті витрат (CAPEX + OPEX):

1. **Розробка (ФОТ):** \$15,000 (умовна вартість, на старті може бути внеском засновників).
2. **Інфраструктура:** \$3,000 (хостинг, домени, сервіси верифікації).
3. **Маркетинг:** \$5,000 (реклама, контент).
4. **Адміністративні:** \$2,000 (реєстрація юрособи, юридичні послуги).

Разом: ~\$25,000.

Джерела фінансування:

- Гранти (USF, міжнародні фонди): 70%.
- Власні кошти / Angel investments: 20%.
- Доходи від ранніх B2B продажів: 10%.

Точка безбитковості (Break-even point) при оптимістичному сценарії досягається на 9-12 місяць роботи за умови залучення 20-30 платних B2B клієнтів (фондів) та реалізації 2-3 спецпроектів для бізнесу.

Висновки до розділу 5

У п'ятому розділі магістерської дисертації проведено детальне розроблення стартап-проєкту на базі «Системи підтримки волонтерської діяльності».

1. **Актуальність та життєздатність:** Доведено, що розроблений продукт вирішує нагальні проблеми ринку (хаос, недовіра, неефективність) та має чітку ціннісну пропозицію, яка відрізняє його від конкурентів (Telegram, єДопомога).
2. **Технологічна готовність:** Технологічний аудит підтвердив правильність вибору архітектурних рішень (Microservices, Java), які забезпечують необхідну гнучкість, безпеку та масштабованість для запуску проєкту. Всі необхідні технології є доступними.
3. **Ринковий потенціал:** Аналіз ринку показав наявність платоспроможного попиту в сегментах B2B (НГО) та CSR (бізнес), а також високий інтерес з боку міжнародних донорів до інструментів цифрової прозорості.
4. **Стратегія успіху:** Розроблена бізнес-модель "Impact Business" та маркетингова стратегія дозволяють поєднати соціальну місію з фінансовою стійкістю. Ключем до успіху є побудова екосистеми довіри та партнерство з ключовими стейкхолдерами.

Таким чином, стартап-проєкт є економічно обґрунтованим, технічно реалізованим та має високі шанси на успішне впровадження в реальному секторі економіки України.

ВИСНОВКИ

У магістерській дисертації проведено дослідження методів та засобів підвищення ефективності організації волонтерської діяльності в Україні. Основна увага була зосереджена на теоретичному обґрунтуванні та проектуванні архітектури спеціалізованої цифрової платформи, здатної мінімізувати існуючі організаційні недоліки.

На етапі аналізу предметної області було ідентифіковано ключові бар'єри у взаємодії волонтерів та організацій, зокрема фрагментарність комунікації та ризики шахрайства. Дослідження існуючих аналогів показало, що вони вирішують здебільшого локальні задачі, що підтвердило доцільність розробки комплексного архітектурного рішення.

У рамках роботи спроектовано мікросервісну архітектуру системи, яка теоретично дозволяє забезпечити гнучкість та масштабованість платформи в умовах змінного навантаження. Було розроблено моделі взаємодії користувачів та структуру бази даних, що враховують специфічні вимоги до безпеки та верифікації учасників волонтерського руху.

Для перевірки життєздатності запропонованих теоретичних рішень було реалізовано програмний прототип системи із використанням сучасного технологічного стеку (Java, Spring Boot, React). Створений прототип дозволив на практиці дослідити механізми асинхронної взаємодії сервісів та перевірити ефективність обраних підходів до автентифікації користувачів.

Проведене тестування прототипу підтвердило технічну коректність спроектованої архітектури та працездатність основних алгоритмів бізнес-логіки. Розроблений стартап-проект продемонстрував потенційну економічну привабливість такої системи для донорів та громадських організацій, що свідчить про перспективність подальшого розвитку дослідження у напрямку повноцінного впровадження.

Список використаних джерел

1. Міністерство соціальної політики України. Нові державні підходи до розвитку волонтерства та підтримки ОГС до 2030 року : прес-реліз Мінсоцполітики від 05.12.2025 [Електронний ресурс]. – Режим доступу: <https://www.msp.gov.ua/press-center/news/novi-derzhavni-pidkhody-do-rozvytku-volonterstva>.
2. Українська Волонтерська Служба. Дослідження волонтерства в Україні: цифри та висновки [Електронний ресурс] // Бібліотека УВС, 29.01.2024. – Режим доступу: <https://volunteer.country/library/74-ukrayinciv-dolucalis-do-volonterstva-pislia-24-liutogo-2022-roku-shho-pokazali-rezultati-doslidzennia>.
3. ІСАР «Єднання». Громадянське суспільство України в умовах війни 2022–2025 рр. : звіт комплексного соціологічного дослідження. – Київ, 2025. – 190 с.
4. Опендатабот. Реєстр волонтерів виріс у 1,5 рази цьогоріч [Електронний ресурс] : аналітична публікація від 05.12.2024. – Режим доступу: <https://opendatabot.ua/analytics/volunteers-2024>.
5. Податковий кодекс України : Закон України від 02.12.2010 р. № 2755-VI (зі змінами) // Відомості Верховної Ради України. – 2011. – № 13–17. – Ст. 112.
6. Фонд «Демократичні ініціативи» ім. І. Кучеріва. Опитування: понад 70 % українців із початку повномасштабної війни долучалися до волонтерської допомоги [Електронний ресурс] : новина від 15.10.2024. – Режим доступу: <https://reinform.in.ua/51474/>.
7. Українське суспільство в умовах війни 2022 : колективна монографія / за ред. Є. Головахи, С. Макеєва. – Київ : Ін-т соціології НАН України, 2022. – 392 с.
8. Волосевич І., Когут І., Жерьобкіна Т., Назаренко Ю. Вплив війни на молодь в Україні : аналітичний звіт дослідження. – Київ : Cedoss ; Info

Sapiens, 2023. – 72 с. – Режим доступу: <https://cedos.org.ua/researches/vpliv-vijni-na-molod-v-ukrayini/>.

9. U-Report. «Волонтерство під час війни» – результати опитування U-Report спільно з УВС, грудень 2022 р. [Електронний ресурс]. – Режим доступу: <https://ukraine.ureport.in/story/1605/>.
10. Горінов П. В., Драпушко Р. Г. Волонтерська діяльність в Україні: соціально-правове дослідження : монографія. – Київ : Державний інститут сімейної та молодіжної політики, 2022. – 240 с.
11. U-Report. «Стійкість волонтерів» – дослідження Української Волонтерської Служби та U-Report, січень 2024 р. [Електронний ресурс]. – Режим доступу: <https://ukraine.ureport.in/story/1617/>.
12. Рокитна А. Шахраї, бюрократія та вигорання: головні виклики для волонтерів в Україні під час війни [Електронний ресурс] // РБК-Україна, аналітика, 12.02.2025. – Режим доступу: <https://www.rbc.ua/ukr/news/shahrayi-byurokratiya-i-vigorannya-shcho-1739188492.html>.
13. Рокитна А. Шахраї, бюрократія та вигорання: головні виклики для волонтерів в Україні під час війни [Електронний ресурс] // РБК-Україна, аналітика, 12.02.2025. – Режим доступу: <https://daily.rbc.ua/analytics/shahrayi-byurokratiya-vigoryannya-golovni-1676182285.html>.
14. Мацієвський Ю., Лук'янчук Я. Волонтерство та псевдоволонтерство в Україні: новий погляд крізь призму неформальних інститутів // Політичні дослідження. – 2024. – № 1(7). – С. 29–37.
15. VolunteerMatch. Профіль платформи на Idealist.org [Електронний ресурс]. – Режим доступу: <https://www.idealists.org/en/nonprofit/56b7cf10e42b45f0...-volunteermatch-pleasanton>.
16. Barrymore J. How VolunteerMatch Works [Електронний ресурс] // HowStuffWorks, 2008. – Режим доступу:

<https://money.howstuffworks.com/volunteer/organizations/volunteermatch.htm>

17. UNICEF Україна. ЮНІСЕФ підтримує молодіжне волонтерство разом з Міністерством молоді та спорту та ГО «Українська Волонтерська Служба» : прес-реліз від 19.04.2021 [Електронний ресурс]. – Режим доступу: <https://www.unicef.org/ukraine/press-releases/youth-volunteering-platform-launch-2021>.
18. Українська Волонтерська Служба. Освітній центр для волонтерів – навчальні матеріали на платформі УВС [Електронний ресурс]. – Режим доступу: <https://volunteer.country/about-us/misiia-ta-istoriia#education>.
19. Дія.Безбар'єрність. Як отримати гуманітарну допомогу [Електронний ресурс] // Роз'яснення Мінцифри, липень 2022. – Режим доступу: <https://bf.diia.gov.ua/articles/yak-otrymaty-humanitarnu-dopomohu>.
20. Кошеленко К. Гуманітарна платформа «єДопомога»: як вона допомагає підтримати людей та економіку країни [Електронний ресурс] // Блог LIGA.net, 15.04.2023. – Режим доступу: <https://blog.liga.net/user/kkoshelenko/article/50212>.
21. Асоціація міст України. «Vcentri HUB» потрібен кожній громаді [Електронний ресурс] : новина АМУ від 11.12.2023. – Режим доступу: <https://auc.org.ua/novyna/vcentri-hub-potriben-kozhniy-gromadi>.
22. Жуленова О. Стан і перспективи волонтерства в Україні: дані опитування 2025 [Електронний ресурс] // Аналітика Ін-ту соціології НАН України, 24.09.2025. – Режим доступу: <https://isnasu.org.ua/analytics/volunteering-2025-data>.
23. Про волонтерську діяльність : Закон України від 19.04.2011 р. № 3236-VI (ред. від 01.01.2023) // Відомості Верховної Ради України. – 2011. – № 42. – Ст. 435.
24. Newman S. Building Microservices. 2nd ed. Sebastopol : O'Reilly Media, 2021. – 616 p.
25. Richardson C. Microservices Patterns: With Examples in Java. Shelter Island, NY : Manning Publications, 2018. – 520 p.

26. **Evans E.** Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston : Addison-Wesley, 2004. – 529 p.
27. **Walls C.** Spring in Action. Sixth Edition. New York : Simon & Schuster, 2022. – 520 p.
[Google Книги+1](#)
28. **Spring Boot Reference Documentation** [Electronic resource]. – URL: <https://docs.spring.io/spring-boot/docs/current/reference/> [Home+1](#)
29. **PostgreSQL Global Development Group.** PostgreSQL 16 Documentation [Electronic resource]. – URL: <https://www.postgresql.org/docs/16/>
30. **Fielding R. T.** Architectural Styles and the Design of Network-based Software Architectures : PhD thesis. University of California, Irvine, 2000.
31. **Jones M. B., Bradley J., Sakimura N.** JSON Web Token (JWT). RFC 7519 [Electronic resource]. IETF, 2015. – URL: <https://datatracker.ietf.org/doc/html/rfc7519>
32. **React documentation** [Electronic resource]. – URL: <https://react.dev/>

ДОДАТОК А

Лістинги програмного коду

А.1 Підсистема ідентифікації (identity-service)

Лістинг А.1.1 — SecurityConfig.java

```
package org.volunteer.identity.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.factory.PasswordEncoderFactories;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.volunteer.identity.security.JwtAuthenticationFilter;

@Configuration
public class SecurityConfig {

    private final JwtAuthenticationFilter jwtAuthenticationFilter;

    public SecurityConfig(JwtAuthenticationFilter jwtAuthenticationFilter) {
        this.jwtAuthenticationFilter = jwtAuthenticationFilter;
    }

    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -> csrf.disable())
            .sessionManagement(sm ->
                sm.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authorizeHttpRequests(auth -> auth
                .requestMatchers(
                    "/api/v1/identity/volunteers",
                    "/api/v1/identity/auth/login",
                    "/api/v1/identity/auth/refresh",
                    "/api/v1/identity/email/confirm",
                    "/actuator/health"
                ).permitAll()
                .anyRequest().authenticated()
            )
            .addFilterBefore(jwtAuthenticationFilter,
                UsernamePasswordAuthenticationFilter.class)
            .httpBasic(httpBasic -> httpBasic.disable())
            .formLogin(form -> form.disable());
        return http.build();
    }

    @Bean
    PasswordEncoder passwordEncoder() {
        return PasswordEncoderFactories.createDelegatingPasswordEncoder();
    }
}
```

```

        @Bean
        AuthenticationManager authenticationManager(AuthenticationConfiguration config)
        throws Exception {
            return config.getAuthenticationManager();
        }
    }
}

```

Лістинг A.1.2 — JwtProperties.java

```

package org.volunteer.identity.config;

import org.springframework.boot.context.properties.ConfigurationProperties;

@ConfigurationProperties(prefix = "app.jwt")
public record JwtProperties(
    String secret,
    String issuer,
    long accessTokenValidity,
    long refreshTokenValidity
) {
}

```

Лістинг A.1.3 — UserAccount.java

```

package org.volunteer.identity.domain;

import jakarta.persistence.*;
import java.time.Instant;
import java.util.EnumSet;
import java.util.Set;
import java.util.UUID;
import lombok.AccessLevel;
import lombok.Getter;
import lombok.NoArgsConstructor;

@Entity
@Table(name = "users")
@Getter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
public class UserAccount {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "public_id", nullable = false, unique = true, updatable = false)
    private UUID publicId = UUID.randomUUID();

    @Column(nullable = false, unique = true, length = 255)
    private String email;

    @Column(name = "password_hash", nullable = false, length = 255)
    private String passwordHash;

    @Column(name = "first_name", length = 100)
    private String firstName;

    @Column(name = "last_name", length = 100)
    private String lastName;
}

```

```

@Column(name = "phone_number", length = 20, unique = true)
private String phoneNumber;

@Enumerated(EnumType.STRING)
@Column(nullable = false, length = 40)
private UserStatus status = UserStatus.PENDING_EMAIL_VERIFICATION;

@ElementCollection(targetClass = UserRole.class, fetch = FetchType.EAGER)
@CollectionTable(name = "user_roles", joinColumns = @JoinColumn(name = "user_id"))
@Enumerated(EnumType.STRING)
@Column(name = "role", nullable = false, length = 40)
private Set<UserRole> roles = EnumSet.noneOf(UserRole.class);

@Column(name = "created_at", nullable = false)
private Instant createdAt = Instant.now();

public UserAccount(String email, String passwordHash, String firstName,
                    String lastName, String phoneNumber, Set<UserRole> roles) {
    this.email = email;
    this.passwordHash = passwordHash;
    this.firstName = firstName;
    this.lastName = lastName;
    this.phoneNumber = phoneNumber;
    this.roles = roles == null || roles.isEmpty() ? EnumSet.noneOf(UserRole.class) :
EnumSet.copyOf(roles);
    this.status = UserStatus.PENDING_EMAIL_VERIFICATION;
}

public void activate() { this.status = UserStatus.ACTIVE; }
public void ban() { this.status = UserStatus.BANNED; }
public void assignRole(UserRole role) { roles.add(role); }
public boolean isActive() { return this.status == UserStatus.ACTIVE; }

public void updateProfile(String firstName, String lastName, String phoneNumber) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.phoneNumber = phoneNumber;
}
}

```

Лістинг A.1.4 — UserStatus.java

```

package org.volunteer.identity.domain;

public enum UserStatus {
    PENDING_EMAIL_VERIFICATION,
    ACTIVE,
    BANNED
}

```

Лістинг A.1.5 — UserRole.java

```

package org.volunteer.identity.domain;

public enum UserRole {
    VOLUNTEER,
    REQUESTER,
    MODERATOR,
    ADMIN
}

```

```
}
```

Лістинг A.1.6 — EmailVerificationToken.java

```
package org.volunteer.identity.domain;

import jakarta.persistence.*;
import java.time.Instant;
import java.util.UUID;
import lombok.AccessLevel;
import lombok.Getter;
import lombok.NoArgsConstructor;

@Entity
@Table(name = "email_verification_tokens")
@Getter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
public class EmailVerificationToken {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "user_id", nullable = false)
    private UserAccount user;

    @Column(nullable = false, unique = true, length = 64)
    private String token;

    @Column(name = "expires_at", nullable = false)
    private Instant expiresAt;

    @Column(name = "consumed_at")
    private Instant consumedAt;

    public EmailVerificationToken(UserAccount user, Instant expiresAt) {
        this.user = user;
        this.token = UUID.randomUUID().toString().replaceAll("-", "");
        this.expiresAt = expiresAt;
    }

    public boolean isExpired(Instant now) { return now.isAfter(expiresAt); }
    public boolean isConsumed() { return consumedAt != null; }
    public void consume(Instant now) { this.consumedAt = now; }
}
```

Лістинг A.1.7 — JwtService.java

```
package org.volunteer.identity.security;

import io.jsonwebtoken.*;
import io.jsonwebtoken.security.Keys;
import java.nio.charset.StandardCharsets;
import java.security.*;
import java.time.Instant;
import java.util.Date;
import java.util.Map;
import org.springframework.stereotype.Component;
import org.volunteer.identity.config.JwtProperties;
```

```

@Component
public class JwtService {

    private final JwtProperties properties;
    private final Key signingKey;
    private final JwtParser jwtParser;

    public JwtService(JwtProperties properties) {
        this.properties = properties;
        this.signingKey = Keys.hmacShaKeyFor(createSigningKeyBytes(properties.secret()));
        this.jwtParser = Jwts.parserBuilder()
            .setSigningKey(signingKey)
            .requireIssuer(properties.issuer())
            .build();
    }

    private byte[] createSigningKeyBytes(String secret) {
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            return digest.digest(secret.getBytes(StandardCharsets.UTF_8));
        } catch (NoSuchAlgorithmException e) {
            throw new IllegalStateException("SHA-256 algorithm not available", e);
        }
    }

    public String generateAccessToken(AuthenticatedUser user) {
        var now = Instant.now();
        var expiry = now.plusSeconds(properties.accessTokenValidity());
        return Jwts.builder()
            .setSubject(user.getUsername())
            .setIssuer(properties.issuer())
            .setIssuedAt(Date.from(now))
            .setExpiration(Date.from(expiry))
            .claim("uid", user.getId())
            .claim("roles", user.getRoles().stream().map(Enum::name).toList())
            .claim("typ", "access")
            .signWith(signingKey, SignatureAlgorithm.HS256)
            .compact();
    }

    public String generateRefreshToken(AuthenticatedUser user) {
        var now = Instant.now();
        var expiry = now.plusSeconds(properties.refreshTokenValidity());
        return Jwts.builder()
            .setSubject(user.getUsername())
            .setIssuer(properties.issuer())
            .setIssuedAt(Date.from(now))
            .setExpiration(Date.from(expiry))
            .addClaims(Map.of("uid", user.getId(), "typ", "refresh"))
            .signWith(signingKey, SignatureAlgorithm.HS256)
            .compact();
    }

    public Claims extractClaims(String token) {
        return jwtParser.parseClaimsJws(token).getBody();
    }

    public boolean isAccessTokenValid(String token, String username) {
        var claims = extractClaims(token);
        return "access".equals(claims.get("typ")) && username.equals(claims.getSubject())
            && claims.getExpiration().after(new Date());
    }
}

```

```

    }

    public boolean isRefreshTokenValid(String token, String username) {
        var claims = extractClaims(token);
        return "refresh".equals(claims.get("typ")) &&
            username.equals(claims.getSubject())
                && claims.getExpiration().after(new Date());
    }

    public String extractUsername(String token) { return
        extractClaims(token).getSubject(); }
    public long accessTokenTtlSeconds() { return properties.accessTokenValidity(); }
    public long refreshTokenTtlSeconds() { return properties.refreshTokenValidity(); }
}

```

Лістинг A.1.8 — JwtAuthenticationFilter.java

```

package org.volunteer.identity.security;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.List;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.HttpHeaders;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

@Component
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    private static final Logger LOGGER =
        LoggerFactory.getLogger(JwtAuthenticationFilter.class);
    private static final String BEARER_PREFIX = "Bearer ";
    private static final List<String> PUBLIC_PATHS = List.of(
        "/api/v1/identity/volunteers",
        "/api/v1/identity/auth/login",
        "/api/v1/identity/auth/refresh",
        "/actuator/health"
    );

    private final JwtService jwtService;
    private final IdentityUserDetailsService userDetailsService;

    public JwtAuthenticationFilter(JwtService jwtService, IdentityUserDetailsService
        userDetailsService) {
        this.jwtService = jwtService;
        this.userDetailsService = userDetailsService;
    }

    @Override
    protected boolean shouldNotFilter(HttpServletRequest request) {
        var path = request.getRequestURI();
        return PUBLIC_PATHS.stream().anyMatch(path::startsWith);
    }
}

```

```

@Override
protected void doFilterInternal(HttpServletRequest request,
                                HttpServletResponse response,
                                FilterChain filterChain) throws ServletException,
IOException {
    String authHeader = request.getHeader(HttpHeaders.AUTHORIZATION);
    if (authHeader == null || !authHeader.startsWith(BEARER_PREFIX)) {
        filterChain.doFilter(request, response);
        return;
    }

    var token = authHeader.substring(BEARER_PREFIX.length());
    try {
        var username = jwtService.extractUsername(token);
        if (username != null &&
SecurityContextHolder.getContext().getAuthentication() == null) {
            var userDetails = userDetailsService.loadUserByUsername(username);
            if (jwtService.isAccessTokenValid(token, userDetails.getUsername())) {
                var authentication = new UsernamePasswordAuthenticationToken(
                    userDetails, null, userDetails.getAuthorities()
                );
                authentication.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(authentication);
            }
        }
    } catch (Exception ex) {
        LOGGER.warn("JWT authentication failed: {}", ex.getMessage());
    }

    filterChain.doFilter(request, response);
}
}

```

Лістинг A.1.9 — AuthenticatedUser.java

```

package org.volunteer.identity.security;

import java.util.Collection;
import java.util.Set;
import java.util.stream.Collectors;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.volunteer.identity.domain.UserRole;

public class AuthenticatedUser implements UserDetails {

    private final Long id;
    private final String email;
    private final String passwordHash;
    private final boolean active;
    private final Set<UserRole> roles;

    public AuthenticatedUser(Long id, String email, String passwordHash, boolean active,
Set<UserRole> roles) {
        this.id = id;
        this.email = email;
        this.passwordHash = passwordHash;
        this.active = active;
    }
}

```

```

        this.roles = roles;
    }

    public Long getId() { return id; }
    public Set<UserRole> getRoles() { return roles; }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return roles.stream()
            .map(role -> new SimpleGrantedAuthority("ROLE_" + role.name()))
            .collect(Collectors.toUnmodifiableSet());
    }

    @Override
    public String getPassword() { return passwordHash; }

    @Override
    public String getUsername() { return email; }

    @Override
    public boolean isAccountNonExpired() { return true; }

    @Override
    public boolean isAccountNonLocked() { return active; }

    @Override
    public boolean isCredentialsNonExpired() { return true; }

    @Override
    public boolean isEnabled() { return active; }
}

```

Лістинг А.1.10 — IdentityUserDetailsService.java

```

package org.volunteer.identity.security;

import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.volunteer.identity.repository.UserAccountRepository;

@Service
public class IdentityUserDetailsService implements UserDetailsService {

    private final UserAccountRepository userAccountRepository;

    public IdentityUserDetailsService(UserAccountRepository userAccountRepository) {
        this.userAccountRepository = userAccountRepository;
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws
    UsernameNotFoundException {
        var account = userAccountRepository.findByEmail(username)
            .orElseThrow(() -> new UsernameNotFoundException("Користувача з email " +
            username + " не знайдено"));

        return new AuthenticatedUser(
            account.getId(),
            account.getEmail(),

```

```

        account.getPasswordHash(),
        account.isActive(),
        account.getRoles()
    );
}
}

```

Лістинг А.1.11 — AuthenticationService.java

```

package org.volunteer.identity.service;

import io.jsonwebtoken.JwtException;
import org.springframework.security.authentication.*;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.volunteer.identity.security.*;
import org.volunteer.identity.web.dto.*;

@Service
public class AuthenticationService {

    private final AuthenticationManager authenticationManager;
    private final JwtService jwtService;
    private final IdentityUserDetailsService userDetailsService;

    public AuthenticationService(AuthenticationManager authenticationManager,
                                JwtService jwtService,
                                IdentityUserDetailsService userDetailsService) {
        this.authenticationManager = authenticationManager;
        this.jwtService = jwtService;
        this.userDetailsService = userDetailsService;
    }

    public TokenResponse authenticate(LoginRequest request) {
        try {
            var authentication = authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(request.getEmail(),
request.getPassword())
            );
            var user = (AuthenticatedUser) authentication.getPrincipal();
            return buildTokenResponse(user);
        } catch (AuthenticationException ex) {
            throw new IllegalArgumentException("Некоректні облікові дані", ex);
        }
    }

    public TokenResponse refresh(RefreshTokenRequest request) {
        try {
            var token = request.getRefreshToken();
            var username = jwtService.extractUsername(token);
            if (!jwtService.isRefreshTokenValid(token, username)) {
                throw new IllegalArgumentException("Недійсний refresh токен");
            }
            var userDetails = userDetailsService.loadUserByUsername(username);
            if (!(userDetails instanceof AuthenticatedUser authenticatedUser)) {
                throw new UsernameNotFoundException("Користувача " + username + " не
знайдено");
            }
            return buildTokenResponse(authenticatedUser);
        } catch (JwtException ex) {

```

```

        throw new IllegalArgumentException("Недійсний refresh токен", ex);
    }
}

private TokenResponse buildTokenResponse(AuthenticatedUser user) {
    var accessToken = jwtService.generateAccessToken(user);
    var refreshToken = jwtService.generateRefreshToken(user);
    TokenResponse response = new TokenResponse();
    response.setAccessToken(accessToken);
    response.setRefreshToken(refreshToken);
    response.setExpiresIn(jwtService.accessTokenTtlSeconds());
    response.setRefreshExpiresIn(jwtService.refreshTokenTtlSeconds());
    return response;
}
}

```

Лістинг A.1.12 — UserRegistrationService.java

```

package org.volunteer.identity.service;

import java.util.Set;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.volunteer.identity.domain.*;
import org.volunteer.identity.events.UserEventPublisher;
import org.volunteer.identity.repository.UserAccountRepository;

@Service
public class UserRegistrationService {

    private final UserAccountRepository userAccountRepository;
    private final PasswordEncoder passwordEncoder;
    private final UserEventPublisher userEventPublisher;
    private final EmailVerificationService emailVerificationService;

    public UserRegistrationService(UserAccountRepository userAccountRepository,
                                   PasswordEncoder passwordEncoder,
                                   UserEventPublisher userEventPublisher,
                                   EmailVerificationService emailVerificationService) {
        this.userAccountRepository = userAccountRepository;
        this.passwordEncoder = passwordEncoder;
        this.userEventPublisher = userEventPublisher;
        this.emailVerificationService = emailVerificationService;
    }

    @Transactional
    public UserAccount registerVolunteer(String email, String rawPassword,
                                           String firstName, String lastName, String
phoneNumber) {
        if (userAccountRepository.existsByEmail(email)) {
            throw new IllegalArgumentException("Email вже зареєстровано");
        }

        var account = new UserAccount(
            email, passwordEncoder.encode(rawPassword),
            firstName, lastName, phoneNumber, Set.of(UserRole.VOLUNTEER)
        );

        var saved = userAccountRepository.save(account);
        var token = emailVerificationService.createTokenFor(saved);
    }
}

```

```

        userEventPublisher.publishUserRegistered(saved, token);
        return saved;
    }
}

```

Лістинг A.1.13 — EmailVerificationService.java

```

package org.volunteer.identity.service;

import java.time.Clock;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.volunteer.identity.config.EmailVerificationProperties;
import org.volunteer.identity.domain.*;
import org.volunteer.identity.repository.*;

@Service
@Transactional
public class EmailVerificationService {

    private static final Logger LOGGER =
        LoggerFactory.getLogger(EmailVerificationService.class);

    private final EmailVerificationTokenRepository tokenRepository;
    private final UserAccountRepository userAccountRepository;
    private final EmailVerificationProperties properties;
    private final Clock clock;

    public EmailVerificationService(EmailVerificationTokenRepository tokenRepository,
                                   UserAccountRepository userAccountRepository,
                                   EmailVerificationProperties properties) {
        this.tokenRepository = tokenRepository;
        this.userAccountRepository = userAccountRepository;
        this.properties = properties;
        this.clock = Clock.systemUTC();
    }

    public String createTokenFor(UserAccount user) {
        var now = clock.instant();
        var token = new EmailVerificationToken(user, now.plus(properties.getTokenTtl()));
        tokenRepository.save(token);
        return token.getToken();
    }

    public void confirmEmail(String tokenValue) {
        var token = tokenRepository.findByTokenAndConsumedAtIsNull(tokenValue)
            .orElseThrow(() -> new IllegalArgumentException("Маркер підтвердження не знайдено"));

        var now = clock.instant();
        if (token.isExpired(now)) {
            throw new IllegalArgumentException("Маркер підтвердження протерміновано");
        }

        token.consume(now);
        tokenRepository.save(token);

        var user = token.getUser();
    }
}

```

```

        if (user.getStatus() != UserStatus.ACTIVE) {
            user.activate();
            userAccountRepository.save(user);
        }
    }

    @Scheduled(cron = "0 0 * * * *")
    public int purgeExpiredTokens() {
        var removed = tokenRepository.deleteByExpiresAtBefore(clock.instant());
        if (removed > 0) {
            LOGGER.debug("Purged {} expired email verification tokens", removed);
        }
        return removed;
    }
}

```

Лістинг A.1.14 — UserProfileService.java

```

package org.volunteer.identity.service;

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.volunteer.identity.domain.UserAccount;
import org.volunteer.identity.repository.UserAccountRepository;
import org.volunteer.identity.web.dto.*;
import org.volunteer.identity.web.mapper.UserAccountMapper;

@Service
public class UserProfileService {

    private final UserAccountRepository userAccountRepository;
    private final UserAccountMapper userAccountMapper;

    public UserProfileService(UserAccountRepository userAccountRepository,
                             UserAccountMapper userAccountMapper) {
        this.userAccountRepository = userAccountRepository;
        this.userAccountMapper = userAccountMapper;
    }

    @Transactional(readOnly = true)
    public UserProfileResponse getProfile(Long userId) {
        UserAccount account = userAccountRepository.findById(userId)
            .orElseThrow(() -> new IllegalStateException("Користувача не знайдено"));
        return userAccountMapper.toUserProfileResponse(account);
    }

    @Transactional
    public UserProfileResponse updateProfile(Long userId, UpdateUserProfileRequest
request) {
        UserAccount account = userAccountRepository.findById(userId)
            .orElseThrow(() -> new IllegalStateException("Користувача не знайдено"));

        String normalizedFirstName = request.getFirstName().trim();
        String normalizedLastName = request.getLastName().trim();

        if (normalizedFirstName.isEmpty() || normalizedLastName.isEmpty()) {
            throw new IllegalArgumentException("Ім'я та прізвище не можуть бути
порожніми");
        }

        String normalizedPhone = request.getPhoneNumber();
    }
}

```

```

        if (normalizedPhone != null) {
            normalizedPhone = normalizedPhone.trim();
            if (normalizedPhone.isBlank()) {
                normalizedPhone = null;
            }
        }

        if (normalizedPhone != null
            && userAccountRepository.existsByPhoneNumberAndIdNot(normalizedPhone,
account.getId())) {
            throw new PhoneNumberAlreadyUsedException("Номер телефону вже
використовується");
        }

        account.updateProfile(normalizedFirstName, normalizedLastName, normalizedPhone);
        return userAccountMapper.toUserProfileResponse(account);
    }
}

```

Лістинг A.1.15 — AuthController.java

```

package org.volunteer.identity.web;

import jakarta.validation.Valid;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.server.ResponseStatusException;
import org.volunteer.identity.service.AuthenticationService;
import org.volunteer.identity.web.api.AuthenticationApi;
import org.volunteer.identity.web.dto.*;

@RestController
public class AuthController implements AuthenticationApi {

    private final AuthenticationService authenticationService;

    public AuthController(AuthenticationService authenticationService) {
        this.authenticationService = authenticationService;
    }

    @Override
    public ResponseEntity<TokenResponse> login(@Valid LoginRequest request) {
        try {
            return ResponseEntity.ok(authenticationService.authenticate(request));
        } catch (IllegalArgumentException ex) {
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST, ex.getMessage(),
ex);
        }
    }

    @Override
    public ResponseEntity<TokenResponse> refreshToken(@Valid RefreshTokenRequest request)
{
        try {
            return ResponseEntity.ok(authenticationService.refresh(request));
        } catch (IllegalArgumentException ex) {
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST, ex.getMessage(),
ex);
        }
    }
}

```

```
}
```

Лістинг A.1.16 — RegistrationController.java

```
package org.volunteer.identity.web;

import jakarta.validation.Valid;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.server.ResponseStatusException;
import org.volunteer.identity.service.UserRegistrationService;
import org.volunteer.identity.web.api.RegistrationApi;
import org.volunteer.identity.web.dto.*;
import org.volunteer.identity.web.mapper.UserAccountMapper;

@RestController
public class RegistrationController implements RegistrationApi {

    private final UserRegistrationService registrationService;
    private final UserAccountMapper accountMapper;

    public RegistrationController(UserRegistrationService registrationService,
                                  UserAccountMapper accountMapper) {
        this.registrationService = registrationService;
        this.accountMapper = accountMapper;
    }

    @Override
    public ResponseEntity<RegisterResponse> registerVolunteer(@Valid
RegisterVolunteerRequest request) {
        try {
            var created = registrationService.registerVolunteer(
                request.getEmail(),
                request.getPassword(),
                request.getFirstName(),
                request.getLastName(),
                request.getPhoneNumber()
            );
            return
ResponseEntity.status(HttpStatus.CREATED).body(accountMapper.toRegisterResponse(created))
;
        } catch (IllegalArgumentException ex) {
            throw new ResponseStatusException(HttpStatus.CONFLICT, ex.getMessage(), ex);
        }
    }
}
```

Лістинг A.1.17 — LoginRequest.java

```
package org.volunteer.identity.web.dto;

import jakarta.validation.constraints.*;
import lombok.*;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class LoginRequest {
```

```

    @NotBlank(message = "Email is required")
    @Email(message = "Please provide a valid email")
    private String email;

    @NotBlank(message = "Password is required")
    @Size(min = 8, max = 72)
    private String password;
}

```

Лістинг A.1.18 — RefreshTokenRequest.java

```

package org.volunteer.identity.web.dto;

import jakarta.validation.constraints.NotBlank;
import lombok.*;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class RefreshTokenRequest {

    @NotBlank(message = "Refresh token is required")
    private String refreshToken;
}

```

Лістинг A.1.19 — TokenResponse.java

```

package org.volunteer.identity.web.dto;

import lombok.*;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class TokenResponse {
    private String accessToken;
    private String refreshToken;
    private long expiresIn;
    private long refreshExpiresIn;
}

```

Лістинг A.1.20 — RegisterVolunteerRequest.java

```

package org.volunteer.identity.web.dto;

import jakarta.validation.constraints.*;
import lombok.*;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class RegisterVolunteerRequest {

    @NotBlank
    @Email
    private String email;

    @NotBlank
    @Size(min = 8, max = 72)

```

```

    private String password;

    @NotBlank
    @Size(max = 100)
    private String firstName;

    @NotBlank
    @Size(max = 100)
    private String lastName;

    @Size(max = 20)
    private String phoneNumber;
}

```

A.2 Підсистема управління запитами (request-service)

Лістинг A.2.1 — Request.java

```

package org.volunteer.request.domain;

import jakarta.persistence.*;
import java.time.Instant;
import java.time.LocalDate;
import java.util.*;
import lombok.*;

@Entity
@Table(name = "requests")
@Getter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
@AllArgsConstructor(access = AccessLevel.PRIVATE)
@Builder(toBuilder = true)
public class Request {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "public_id", nullable = false, unique = true, updatable = false,
length = 36)
    private String publicId;

    @Column(nullable = false, length = 200)
    private String title;

    @Column(nullable = false, columnDefinition = "text")
    private String description;

    @Column(name = "skills", columnDefinition = "text")
    private String skills;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false, length = 30)
    @Builder.Default
    private RequestStatus status = RequestStatus.DRAFT;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false, length = 20)
    @Builder.Default
    private RequestPriority priority = RequestPriority.MEDIUM;
}

```

```

@Enumerated(EnumType.STRING)
@Column(name = "request_type", nullable = false, length = 30)
@Builder.Default
private RequestType requestType = RequestType.HELP_REQUEST;

@OneToOne(mappedBy = "request", cascade = CascadeType.ALL, orphanRemoval = true,
fetch = FetchType.LAZY)
private FundraiserDetails fundraiserDetails;

@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "category_id")
private RequestCategory category;

@Column(name = "creator_id", nullable = false)
private Long creatorId;

@Column(name = "assigned_volunteer_id")
private Long assignedVolunteerId;

@OneToMany(mappedBy = "request", cascade = CascadeType.ALL, orphanRemoval = true)
@Builder.Default
private List<Response> responses = new ArrayList<>();

@Column(name = "location_lat")
private Double locationLat;

@Column(name = "location_lon")
private Double locationLon;

@Column(name = "address_details", length = 512)
private String addressDetails;

@Column(length = 120)
private String city;

@Column(length = 120)
private String region;

@Column(length = 120)
private String country;

@Column(name = "due_date")
private LocalDate dueDate;

@Column(name = "published_at")
private Instant publishedAt;

@Column(name = "created_at", nullable = false)
private Instant createdAt;

@Column(name = "updated_at", nullable = false)
private Instant updatedAt;

@Column(name = "archived_at")
private Instant archivedAt;

@PrePersist
void onCreate() {
    var now = Instant.now();
    this.publicId = UUID.randomUUID().toString();
    this.createdAt = now;
}

```

```

        this.updatedAt = now;
    }

    @PreUpdate
    void onUpdate() { this.updatedAt = Instant.now(); }

    public void changeStatus(RequestStatus newStatus) {
        this.status = newStatus;
        if (newStatus == RequestStatus.PUBLISHED && publishedAt == null) {
            this.publishedAt = Instant.now();
        }
        if (newStatus.isTerminal()) {
            this.archivedAt = Instant.now();
        } else {
            this.archivedAt = null;
        }
    }

    public void assignCategory(RequestCategory category) { this.category = category; }
    public void assignVolunteer(Long volunteerId) { this.assignedVolunteerId =
volunteerId; }
    public void clearAssignee() { this.assignedVolunteerId = null; }

    public void addResponse(Response response) {
        Objects.requireNonNull(response, "response must not be null");
        responses.add(response);
    }
}

```

Лістинг A.2.2 — Response.java

```

package org.volunteer.request.domain;

import jakarta.persistence.*;
import java.time.Instant;
import lombok.*;

@Entity
@Table(name = "responses")
@Getter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
@AllArgsConstructor(access = AccessLevel.PRIVATE)
@Builder(toBuilder = true)
public class Response {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "request_id", nullable = false)
    private Request request;

    @Column(name = "volunteer_id", nullable = false)
    private Long volunteerId;

    @Column(columnDefinition = "text")
    private String message;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false, length = 30)

```

```

@Builder.Default
private ResponseStatus status = ResponseStatus.PENDING;

@Column(name = "created_at", nullable = false)
private Instant createdAt;

@Column(name = "updated_at", nullable = false)
private Instant updatedAt;

@PrePersist
void onCreate() {
    var now = Instant.now();
    this.createdAt = now;
    this.updatedAt = now;
}

@PreUpdate
void onUpdate() { this.updatedAt = Instant.now(); }

public Response(Request request, Long volunteerId, String message) {
    this.request = request;
    this.volunteerId = volunteerId;
    this.message = message;
    this.status = ResponseStatus.PENDING;
}

public void accept() { this.status = ResponseStatus.ACCEPTED; this.updatedAt =
Instant.now(); }
public void decline() { this.status = ResponseStatus.DECLINED; this.updatedAt =
Instant.now(); }
public void withdraw() { this.status = ResponseStatus.WITHDRAWN; this.updatedAt =
Instant.now(); }
}

```

Лістинг A.2.3 — FundraiserDetails.java

```

package org.volunteer.request.domain;

import jakarta.persistence.*;
import java.time.Instant;
import lombok.*;

@Entity
@Table(name = "fundraiser_details")
@Getter
@Setter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
@AllArgsConstructor(access = AccessLevel.PRIVATE)
@Builder
public class FundraiserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @OneToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "request_id", nullable = false, unique = true)
    private Request request;

    @Column(name = "jar_url", nullable = false, length = 500)
    private String jarUrl;
}

```

```

@Column(name = "jar_id", length = 100)
private String jarId;

@Enumerated(EnumType.STRING)
@Column(name = "bank_provider", nullable = false, length = 50)
@Builder.Default
private BankProvider bankProvider = BankProvider.OTHER;

@Column(name = "goal_amount", nullable = false)
private Long goalAmount;

@Column(name = "current_amount", nullable = false)
@Builder.Default
private Long currentAmount = 0L;

@Column(name = "currency", nullable = false, length = 3)
@Builder.Default
private String currency = "UAH";

@Column(name = "last_synced_at")
private Instant lastSyncedAt;

@Column(name = "created_at", nullable = false)
private Instant createdAt;

@Column(name = "updated_at", nullable = false)
private Instant updatedAt;

@PrePersist
void onCreate() {
    var now = Instant.now();
    this.createdAt = now;
    this.updatedAt = now;
    if (this.jarUrl != null) {
        if (this.bankProvider == null) {
            this.bankProvider = BankProvider.fromUrl(this.jarUrl);
        }
        if (this.jarId == null) {
            this.jarId = BankProvider.extractJarId(this.jarUrl);
        }
    }
}

@PreUpdate
void onUpdate() { this.updatedAt = Instant.now(); }

public int getProgressPercent() {
    if (goalAmount == null || goalAmount == 0) return 0;
    return (int) Math.min(100, (currentAmount * 100) / goalAmount);
}

public boolean isGoalReached() { return currentAmount >= goalAmount; }

public void updateAmount(Long newAmount) {
    this.currentAmount = newAmount;
    this.lastSyncedAt = Instant.now();
}
}

```

```

package org.volunteer.request.domain;

import jakarta.persistence.*;
import lombok.*;

@Entity
@Table(name = "request_categories")
@Getter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
public class RequestCategory {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(nullable = false, unique = true, length = 100)
    private String name;

    @ManyToOne
    @JoinColumn(name = "parent_category_id")
    private RequestCategory parent;

    public RequestCategory(String name, RequestCategory parent) {
        this.name = name;
        this.parent = parent;
    }
}

```

Лістинг A.2.5 — RequestStatus.java

```

package org.volunteer.request.domain;

public enum RequestStatus {
    DRAFT, PENDING_REVIEW, PUBLISHED, ASSIGNED, IN_PROGRESS, COMPLETED, CANCELLED,
    REJECTED;

    public boolean isTerminal() {
        return this == COMPLETED || this == CANCELLED || this == REJECTED;
    }
}

```

Лістинг A.2.6 — RequestPriority.java

```

package org.volunteer.request.domain;

public enum RequestPriority { LOW, MEDIUM, HIGH, CRITICAL }

```

Лістинг A.2.7 — RequestType.java

```

package org.volunteer.request.domain;

public enum RequestType {
    HELP_REQUEST("Запит допомоги"),
    FUNDRAISER("Збір коштів");

    private final String displayName;

    RequestType(String displayName) { this.displayName = displayName; }
    public String getDisplayName() { return displayName; }
}

```

```
}
```

Лістинг A.2.8 — ResponseStatus.java

```
package org.volunteer.request.domain;

public enum ResponseStatus { PENDING, ACCEPTED, DECLINED, WITHDRAWN }
```

Лістинг A.2.9 — BankProvider.java

```
package org.volunteer.request.domain;

public enum BankProvider {
    MONOBANK("Monobank", "send.monobank.ua"),
    PRIVATBANK("PrivatBank", "privatbank.ua"),
    OTHER("Other", null);

    private final String displayName;
    private final String urlPattern;

    BankProvider(String displayName, String urlPattern) {
        this.displayName = displayName;
        this.urlPattern = urlPattern;
    }

    public String getDisplayName() { return displayName; }

    public static BankProvider fromUrl(String url) {
        if (url == null || url.isBlank()) return OTHER;
        String lowerUrl = url.toLowerCase();
        if (lowerUrl.contains("send.monobank.ua")) return MONOBANK;
        if (lowerUrl.contains("privatbank.ua")) return PRIVATBANK;
        return OTHER;
    }

    public static String extractJarId(String url) {
        if (url == null || url.isBlank()) return null;
        if (url.contains("/jar/")) {
            int idx = url.lastIndexOf("/");
            if (idx > 0 && idx < url.length() - 1) {
                return url.substring(idx + 1).split("\\?")[0];
            }
        }
        return null;
    }
}
```

Лістинг A.2.10 — RequestRepository.java

```
package org.volunteer.request.repository;

import java.util.*;
import org.springframework.data.jpa.repository.*;
import org.volunteer.request.domain.*;

public interface RequestRepository extends JpaRepository<Request, Long>,
JpaSpecificationExecutor<Request> {
    Optional<Request> findByPublicId(String publicId);
    List<Request> findByCreatorId(Long creatorId);
}
```

```

        List<Request> findByStatus(RequestStatus status);
        List<Request> findByRequestType(RequestType requestType);
    }

```

Лістинг А.2.11 — ResponseRepository.java

```

package org.volunteer.request.repository;

import java.util.*;
import org.springframework.data.jpa.repository.*;
import org.springframework.data.repository.query.Param;
import org.volunteer.request.domain.*;

public interface ResponseRepository extends JpaRepository<Response, Long> {
    boolean existsByRequestIdAndVolunteerId(Long requestId, Long volunteerId);
    List<Response> findByRequestOrderByCreatedAtAsc(Request request);

    @Query("SELECT r FROM Response r JOIN FETCH r.request WHERE r.volunteerId = :volunteerId")
    List<Response> findByVolunteerIdWithRequest(@Param("volunteerId") Long volunteerId);

    Optional<Response> findByIdAndRequest(Long id, Request request);
}

```

Лістинг А.2.12 — RequestService.java

```

package org.volunteer.request.service;

import java.util.*;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.volunteer.request.domain.*;
import org.volunteer.request.repository.*;
import org.volunteer.request.service.event.OutboxEventPublisher;

@Service
@RequiredArgsConstructor
public class RequestService {

    private final RequestRepository requestRepository;
    private final RequestCategoryRepository categoryRepository;
    private final OutboxEventPublisher outboxEventPublisher;

    @Transactional
    public Request createRequest(Request request, Integer categoryId) {
        RequestCategory category = categoryRepository.findById(categoryId)
            .orElseThrow(() -> new IllegalArgumentException("Категорію не знайдено"));
        request.assignCategory(category);
        Request saved = requestRepository.save(request);
        outboxEventPublisher.publishRequestCreated(saved);
        return saved;
    }

    @Transactional
    public Request changeStatus(String publicId, RequestStatus status) {
        Request request = requestRepository.findById(publicId)
            .orElseThrow(() -> new IllegalArgumentException("Запит не знайдено"));
        RequestStatus previousStatus = request.getStatus();
        request.changeStatus(status);
    }
}

```

```

        outboxEventPublisher.publishRequestStatusChanged(request, previousStatus,
status);
        return request;
    }

    @Transactional(readOnly = true)
    public Request findByPublicId(String publicId) {
        return requestRepository.findByPublicId(publicId)
            .orElseThrow(() -> new IllegalArgumentException("Запит не знайдено"));
    }
}

```

Лістинг А.2.13 — ResponseService.java

```

package org.volunteer.request.service;

import java.util.*;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.server.ResponseStatusException;
import org.volunteer.request.domain.*;
import org.volunteer.request.repository.*;
import org.volunteer.request.web.dto.CreateResponseRequest;

@Service
@RequiredArgsConstructor
public class ResponseService {

    private final RequestRepository requestRepository;
    private final ResponseRepository responseRepository;

    @Transactional
    public Response createResponse(String requestPublicId, CreateResponseRequest payload)
    {
        Request request = requestRepository.findByPublicId(requestPublicId)
            .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND, "Запит
не знайдено"));

        if (Objects.equals(request.getCreatorId(), payload.getVolunteerId())) {
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Волонтер не може
відгукуватися на власний запит");
        }

        if (request.getStatus().isTerminal()) {
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Запит недоступний
для відгуків");
        }

        boolean alreadyExists = responseRepository.existsByRequestIdAndVolunteerId(
            request.getId(), payload.getVolunteerId());
        if (alreadyExists) {
            throw new ResponseStatusException(HttpStatus.CONFLICT, "Волонтер вже залишив
відгук");
        }

        Response response = new Response(request, payload.getVolunteerId(),
payload.getMessage());
        request.addResponse(response);
        return responseRepository.save(response);
    }
}

```

```

    }

    @Transactional(readOnly = true)
    public List<Response> findResponses(String requestPublicId) {
        Request request = requestRepository.findByPublicId(requestPublicId)
            .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND, "Запит
не знайдено"));
        return responseRepository.findByRequestOrderByCreatedAtAsc(request);
    }
}

```

Лістинг A.2.14 — OutboxEvent.java

```

package org.volunteer.request.service.event;

import jakarta.persistence.*;
import java.time.Instant;
import lombok.*;

@Entity
@Table(name = "request_outbox_events")
@Getter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
public class OutboxEvent {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "event_type", nullable = false, length = 100)
    private String eventType;

    @Column(name = "aggregate_id", nullable = false, length = 36)
    private String aggregateId;

    @Column(name = "payload", nullable = false, columnDefinition = "text")
    private String payload;

    @Column(name = "created_at", nullable = false)
    private Instant createdAt;

    @Column(name = "processed_at")
    private Instant processedAt;

    public OutboxEvent(String eventType, String aggregateId, String payload) {
        this.eventType = eventType;
        this.aggregateId = aggregateId;
        this.payload = payload;
    }

    @PrePersist
    void onCreate() { if (createdAt == null) createdAt = Instant.now(); }

    public boolean isProcessed() { return processedAt != null; }
    public void markProcessed() { this.processedAt = Instant.now(); }
}

```

Лістинг A.2.15 — OutboxDispatcher.java

```

package org.volunteer.request.service.event;

```

```

import java.util.ArrayList;
import java.util.List;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.data.domain.PageRequest;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;

@Slf4j
@Component
public class OutboxDispatcher {

    private final OutboxEventRepository outboxEventRepository;
    private final DomainEventPublisher domainEventPublisher;
    private final int batchSize;

    public OutboxDispatcher(OutboxEventRepository outboxEventRepository,
                           DomainEventPublisher domainEventPublisher,
                           @Value("${request.outbox.dispatcher.batch-size:50}") int
batchSize) {
        this.outboxEventRepository = outboxEventRepository;
        this.domainEventPublisher = domainEventPublisher;
        this.batchSize = batchSize;
    }

    @Transactional
    @Scheduled(fixedDelayString = "${request.outbox.dispatcher.fixed-delay:5000}")
    public void dispatch() {
        List<OutboxEvent> events = outboxEventRepository
            .findByProcessedAtIsNullOrderByCreatedAtAsc(PageRequest.of(0, batchSize));

        if (events.isEmpty()) return;

        List<OutboxEvent> processed = new ArrayList<>(events.size());
        for (OutboxEvent event : events) {
            try {
                domainEventPublisher.publish(event.getEventType(),
event.getAggregateId(), event.getPayload());
                event.markProcessed();
                processed.add(event);
            } catch (Exception ex) {
                log.error("Failed to publish outbox event id={}", event.getId(), ex);
                break;
            }
        }

        if (!processed.isEmpty()) {
            outboxEventRepository.saveAll(processed);
        }
    }
}

```

Лістинг A.2.16 — RequestController.java

```

package org.volunteer.request.web;

import java.util.List;
import lombok.RequiredArgsConstructor;
import org.springframework.http.*;

```

```

import org.springframework.web.bind.annotation.*;
import org.volunteer.request.domain.*;
import org.volunteer.request.service.RequestService;
import org.volunteer.request.web.dto.*;
import org.volunteer.request.web.mapper.RequestMapper;

@RestController
@RequestMapping("/api/v1/requests")
@RequiredArgsConstructor
public class RequestController {

    private final RequestService requestService;
    private final RequestMapper requestMapper;

    @PostMapping
    public ResponseEntity<RequestResponse> create(@RequestBody CreateRequestRequest dto)
    {
        Request request = requestMapper.toEntity(dto);
        Request saved = requestService.createRequest(request, dto.getCategoryId());
        return
        ResponseEntity.status(HttpStatus.CREATED).body(requestMapper.toResponse(saved));
    }

    @GetMapping("/{publicId}")
    public ResponseEntity<RequestResponse> getByPublicId(@PathVariable String publicId) {
        Request request = requestService.findByPublicId(publicId);
        return ResponseEntity.ok(requestMapper.toResponse(request));
    }

    @PatchMapping("/{publicId}/status")
    public ResponseEntity<RequestResponse> changeStatus(@PathVariable String publicId,
                                                         @RequestBody ChangeStatusRequest
                                                         dto) {
        Request request = requestService.changeStatus(publicId, dto.getStatus());
        return ResponseEntity.ok(requestMapper.toResponse(request));
    }
}

```

Лістинг A.2.17 — ResponseController.java

```

package org.volunteer.request.web;

import java.util.List;
import lombok.RequiredArgsConstructor;
import org.springframework.http.*;
import org.springframework.web.bind.annotation.*;
import org.volunteer.request.domain.Response;
import org.volunteer.request.service.ResponseService;
import org.volunteer.request.web.dto.*;
import org.volunteer.request.web.mapper.ResponseMapper;

@RestController
@RequestMapping("/api/v1/requests/{publicId}/responses")
@RequiredArgsConstructor
public class ResponseController {

    private final ResponseService responseService;
    private final ResponseMapper responseMapper;

    @PostMapping
    public ResponseEntity<ResponseDto> create(@PathVariable String publicId,

```

```

        @RequestBody CreateResponseRequest dto) {
            Response response = responseService.createResponse(publicId, dto);
            return
                ResponseEntity.status(HttpStatus.CREATED).body(responseMapper.toDto(response));
        }

        @GetMapping
        public ResponseEntity<List<ResponseDto>> list(@PathVariable String publicId) {
            List<Response> responses = responseService.findResponses(publicId);
            return ResponseEntity.ok(responses.stream().map(responseMapper::toDto).toList());
        }
    }
}

```

Лістинг А.2.18 — CreateRequestRequest.java

```

package org.volunteer.request.web.dto;

import jakarta.validation.constraints.*;
import java.time.LocalDate;
import lombok.*;
import org.volunteer.request.domain.RequestPriority;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class CreateRequestRequest {

    @NotBlank
    @Size(max = 200)
    private String title;

    @NotBlank
    private String description;

    private String skills;

    @NotNull
    private Integer categoryId;

    @NotNull
    private Long creatorId;

    private RequestPriority priority;
    private LocalDate dueDate;
    private Double locationLat;
    private Double locationLon;
    private String city;
}

```

Лістинг А.2.19 — RequestResponse.java

```

package org.volunteer.request.web.dto;

import java.time.Instant;
import java.time.LocalDate;
import lombok.*;

@Data
@Builder
@NoArgsConstructor

```

```

@AllArgsConstructor
public class RequestResponse {
    private String publicId;
    private String title;
    private String description;
    private String status;
    private String priority;
    private String requestType;
    private Integer categoryId;
    private String categoryName;
    private Long creatorId;
    private Long assignedVolunteerId;
    private String city;
    private LocalDate dueDate;
    private Instant createdAt;
    private Instant updatedAt;
}

```

Лістинг A.2.20 — ChangeStatusRequest.java

```

package org.volunteer.request.web.dto;

import jakarta.validation.constraints.NotNull;
import lombok.*;
import org.volunteer.request.domain.RequestStatus;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class ChangeStatusRequest {
    @NotNull
    private RequestStatus status;
}

```

Лістинг A.2.21 — CreateResponseRequest.java

```

package org.volunteer.request.web.dto;

import jakarta.validation.constraints.NotNull;
import lombok.*;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class CreateResponseRequest {
    @NotNull
    private Long volunteerId;
    private String message;
}

```

Лістинг A.2.22 — ResponseDto.java

```

package org.volunteer.request.web.dto;

import java.time.Instant;
import lombok.*;

@Data
@Builder

```

```

@NoArgsConstructor
@AllArgsConstructor
public class ResponseDto {
    private Long id;
    private Long volunteerId;
    private String message;
    private String status;
    private Instant createdAt;
    private Instant updatedAt;
}

```

A.3 Підсистема сповіщень (notification-service)

Лістинг A.3.1 — MailPayload.java

```

package org.volunteer.notification.service;

import java.util.List;
import lombok.Builder;
import lombok.Builder.Default;
import lombok.Value;

@Value
@Builder
public class MailPayload {
    String to;
    String subject;
    String htmlBody;
    String textBody;
    @Default
    List<String> bcc = List.of();
}

```

Лістинг A.3.2 — MailComposer.java

```

package org.volunteer.notification.service;

import java.util.Locale;
import java.util.Map;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Component;
import org.thymeleaf.TemplateEngine;
import org.thymeleaf.context.Context;
import org.volunteer.notification.config.MailProperties;

@Component
@RequiredArgsConstructor
public class MailComposer {

    private final TemplateEngine templateEngine;
    private final MailProperties mailProperties;

    public MailPayload composeWelcomeMail(String recipient, String fullName, String token) {
        var displayName = (fullName == null || fullName.isBlank()) ? recipient :
        fullName;
        var frontendUrl = mailProperties.getFrontendBaseUrl();
        var verificationLink = frontendUrl + "/verify-email?token=" + token;
    }
}

```

```

var context = new Context(Locale.getDefault());
context.setVariables(Map.of(
    "fullName", displayName,
    "frontendUrl", frontendUrl,
    "token", token,
    "verificationLink", verificationLink
));

var subject = "Ласкаво просимо до спільноти волонтерів";
var html = templateEngine.process("welcome-email", context);
var text = "Вітаємо, " + displayName + "! Перейдіть за посиланням "
    + verificationLink + " для підтвердження адреси.";

return MailPayload.builder()
    .to(recipient)
    .subject(subject)
    .htmlBody(html)
    .textBody(text)
    .bcc(mailProperties.getAuditBcc())
    .build();
}
}

```

Лістинг А.3.3 — MailSenderService.java

```

package org.volunteer.notification.service;

import jakarta.mail.MessagingException;
import java.io.UnsupportedEncodingException;
import java.nio.charset.StandardCharsets;
import java.util.List;
import lombok.RequiredArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.MimeMessageHelper;
import org.springframework.stereotype.Service;
import org.volunteer.notification.config.MailProperties;

@Service
@RequiredArgsConstructor
public class MailSenderService {

    private static final Logger LOGGER =
        LoggerFactory.getLogger(MailSenderService.class);

    private final JavaMailSender mailSender;
    private final MailProperties mailProperties;

    public void send(MailPayload payload) {
        try {
            var message = mailSender.createMimeMessage();
            var helper = new MimeMessageHelper(message, true,
                StandardCharsets.UTF_8.name());

            helper.setTo(payload.getTo());
            helper.setFrom(helper);
            helper.setBcc(helper, payload.getBcc());
            helper.setSubject(payload.getSubject());
            helper.setText(payload.getTextBody(), payload.getHtmlBody());

```

```

        mailSender.send(message);
    } catch (MessagingException | UnsupportedEncodingException ex) {
        LOGGER.error("Failed to send mail to {}: {}", payload.getTo(),
ex.getMessage(), ex);
        throw new IllegalStateException("Не вдалося надіслати email", ex);
    }
}

private void setFrom(MimeMessageHelper helper) throws MessagingException,
UnsupportedEncodingException {
    var from = mailProperties.getFrom();
    var senderName = mailProperties.getSenderName();
    if (senderName != null && !senderName.isBlank()) {
        helper.setFrom(from, senderName);
    } else {
        helper.setFrom(from);
    }
}

private void setBcc(MimeMessageHelper helper, List<String> bcc) throws
MessagingException {
    if (bcc != null && !bcc.isEmpty()) {
        helper.setBcc(bcc.toArray(String[]::new));
    }
}
}

```

Лістинг A.3.4 — NotificationDispatchService.java

```

package org.volunteer.notification.service;

import java.util.Objects;
import java.util.stream.Stream;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Service;
import org.volunteer.notification.event.UserRegisteredMessage;

@Service
public class NotificationDispatchService {

    private static final Logger LOGGER =
LoggerFactory.getLogger(NotificationDispatchService.class);

    private final MailComposer mailComposer;
    private final MailSenderService mailSenderService;

    public NotificationDispatchService(MailComposer mailComposer, MailSenderService
mailSenderService) {
        this.mailComposer = mailComposer;
        this.mailSenderService = mailSenderService;
    }

    public void scheduleWelcomeNotification(UserRegisteredMessage message) {
        var fullName = Stream.of(message.getFirstName(), message.getLastName())
            .filter(Objects::nonNull)
            .map(String::trim)
            .filter(value -> !value.isEmpty())
            .reduce((first, second) -> first + " " + second)
            .orElse(null);
    }
}

```

```

        var payload = mailComposer.composeWelcomeMail(
            message.getEmail(), fullName, message.getVerificationToken()
        );

        mailSenderService.send(payload);
        LOGGER.info("Welcome email dispatched to {}", message.getEmail());
    }
}

```

A.4 Підсистема пошуку (search-service)

Лістинг A.4.1 — RequestDocument.java

```

package org.volunteer.search.domain;

import java.time.Instant;
import java.util.Set;
import lombok.*;
import org.springframework.data.annotation.Id;
import org.springframework.data.elasticsearch.annotations.*;

@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Document(indexName = "requests")
public class RequestDocument {

    @Id
    private String id;

    @Field(type = FieldType.Keyword)
    private String publicId;

    @Field(type = FieldType.Text)
    private String title;

    @Field(type = FieldType.Text)
    private String description;

    @Field(type = FieldType.Keyword)
    private String status;

    @Field(type = FieldType.Keyword)
    private String priority;

    @Field(type = FieldType.Integer)
    private Integer categoryId;

    @Field(type = FieldType.Long)
    private Long creatorId;

    @Field(type = FieldType.Keyword)
    private String city;

    @Field(type = FieldType.Date)
    private Instant createdAt;

    @Field(type = FieldType.Keyword)

```

```

    @Builder.Default
    private Set<String> tags = Set.of();
}

```

Лістинг А.4.2 — RequestSearchService.java

```

package org.volunteer.search.service;

import java.util.*;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.*;
import org.springframework.data.elasticsearch.core.*;
import org.springframework.data.elasticsearch.core.query.*;
import org.springframework.stereotype.Service;
import org.volunteer.search.domain.RequestDocument;
import reactor.core.publisher.Flux;

@Service
@RequiredArgsConstructor
public class RequestSearchService {

    private final ReactiveElasticsearchOperations operations;

    public Flux<RequestDocument> search(Optional<String> keyword,
                                       Optional<String> status,
                                       Optional<String> priority,
                                       Optional<Integer> categoryId,
                                       Optional<String> city,
                                       Optional<Integer> size) {
        List<Criteria> clauses = new ArrayList<>();

        keyword.filter(v -> !v.isBlank()).ifPresent(value ->
            clauses.add(new Criteria("title").matches(value)
                .or(new Criteria("description").matches(value)))
        );

        status.filter(v -> !v.isBlank()).ifPresent(value -> clauses.add(new
Criteria("status").is(value)));
        priority.filter(v -> !v.isBlank()).ifPresent(value -> clauses.add(new
Criteria("priority").is(value)));
        categoryId.ifPresent(value -> clauses.add(new Criteria("categoryId").is(value)));
        city.filter(v -> !v.isBlank()).ifPresent(value -> clauses.add(new
Criteria("city").is(value)));

        Criteria criteria = null;
        for (Criteria clause : clauses) {
            criteria = criteria == null ? clause : criteria.and(clause);
        }

        CriteriaQuery query = new CriteriaQuery(criteria != null ? criteria : new
Criteria());
        query.addSort(Sort.by(Sort.Direction.DESC, "createdAt"));
        size.ifPresent(value -> query.setPageable(PageRequest.of(0,
Math.min(Math.max(value, 1), 100))));

        return operations.search(query,
RequestDocument.class).map(SearchHit::getContent);
    }
}

```

A.5 API-шлюз (api-gateway)

Лістинг A.5.1 — JwtGatewayProperties.java

```
package org.volunteer.gateway.config;

import java.util.List;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.boot.context.properties.bind.DefaultValue;

@ConfigurationProperties(prefix = "gateway.jwt")
public record JwtGatewayProperties(
    String secret,
    @DefaultValue("volunteer-support-system") String issuer,
    @DefaultValue({"api/v1/identity/auth", "api/v1/identity/volunteers", "actuator"})
    List<String> publicPaths
) {
}
```

Лістинг A.5.2 — JwtTokenValidator.java

```
package org.volunteer.gateway.security;

import io.jsonwebtoken.*;
import io.jsonwebtoken.security.Keys;
import java.nio.charset.StandardCharsets;
import java.security.*;
import java.time.Instant;
import java.util.*;
import org.springframework.stereotype.Component;
import org.volunteer.gateway.config.JwtGatewayProperties;

@Component
public class JwtTokenValidator {

    private final JwtGatewayProperties properties;
    private final JwtParser parser;

    public JwtTokenValidator(JwtGatewayProperties properties) {
        this.properties = properties;
        Key key = Keys.hmacShaKeyFor(createSigningKeyBytes(properties.secret()));
        this.parser = Jwts.parserBuilder()
            .setSigningKey(key)
            .requireIssuer(properties.issuer())
            .build();
    }

    private byte[] createSigningKeyBytes(String secret) {
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            return digest.digest(secret.getBytes(StandardCharsets.UTF_8));
        } catch (NoSuchAlgorithmException e) {
            throw new IllegalStateException("SHA-256 algorithm not available", e);
        }
    }

    public boolean isPublicPath(String path) {
        return properties.publicPaths().stream().anyMatch(path::startsWith);
    }

    public Claims validateAccessToken(String token) {

```

```

        Claims claims = parser.parseClaimsJws(token).getBody();
        if (!"access".equals(claims.get("typ"))) {
            throw new IllegalArgumentException("Invalid token type");
        }
        Date expiration = claims.getExpiration();
        if (expiration == null || expiration.toInstant().isBefore(Instant.now())) {
            throw new IllegalArgumentException("Token expired");
        }
        return claims;
    }

    @SuppressWarnings("unchecked")
    public Set<String> extractRoles(Claims claims) {
        Object roles = claims.get("roles");
        if (roles instanceof List<?> list) {
            return
list.stream().map(String::valueOf).collect(java.util.stream.Collectors.toUnmodifiableSet(
));
        }
        return Set.of();
    }
}

```

Лістинг А.5.3 — JwtAuthenticationFilter.java

```

package org.volunteer.gateway.security;

import io.jsonwebtoken.Claims;
import java.util.Set;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.core.Ordered;
import org.springframework.http.*;
import org.springframework.http.server.reactive.ServerHttpRequest;
import org.springframework.stereotype.Component;
import org.springframework.web.server.ServerWebExchange;
import org.springframework.cloud.gateway.filter.*;
import reactor.core.publisher.Mono;

@Component
public class JwtAuthenticationFilter implements GlobalFilter, Ordered {

    private static final Logger LOGGER =
LoggerFactory.getLogger(JwtAuthenticationFilter.class);
    private static final String BEARER_PREFIX = "Bearer ";

    private final JwtTokenValidator tokenValidator;

    public JwtAuthenticationFilter(JwtTokenValidator tokenValidator) {
        this.tokenValidator = tokenValidator;
    }

    @Override
    public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {
        String path = exchange.getRequest().getURI().getPath();
        if (tokenValidator.isPublicPath(path)) {
            return chain.filter(exchange);
        }

        String authHeader =
exchange.getRequest().getHeaders().getFirst(HttpHeaders.AUTHORIZATION);

```

```

        if (authHeader == null || !authHeader.startsWith(BEARER_PREFIX)) {
            LOGGER.debug("Missing bearer token for path {}", path);
            exchange.getResponse().setStatusCode(HttpStatus.UNAUTHORIZED);
            return exchange.getResponse().setComplete();
        }

        String token = authHeader.substring(BEARER_PREFIX.length());
        try {
            Claims claims = tokenValidator.validateAccessToken(token);
            String userId = String.valueOf(claims.get("uid"));
            Set<String> roles = tokenValidator.extractRoles(claims);

            ServerHttpRequest mutatedRequest = exchange.getRequest().mutate()
                .header("X-User-Id", userId)
                .header("X-User-Roles", String.join(",", roles))
                .build();

            return chain.filter(exchange.mutate().request(mutatedRequest).build());
        } catch (IllegalArgumentException ex) {
            LOGGER.debug("Token validation failed: {}", ex.getMessage());
            exchange.getResponse().setStatusCode(HttpStatus.UNAUTHORIZED);
            return exchange.getResponse().setComplete();
        }
    }

    @Override
    public int getOrder() { return Ordered.HIGHEST_PRECEDENCE + 10; }
}

```

A.6 Веб-застосунок (web-app)

Лістинг A.6.1 — SecurityConfig.java

```

package org.volunteer.webapp.config;

import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.EnableWebSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.volunteer.webapp.security.JwtAuthenticationFilter;

@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityConfig {

    private final JwtAuthenticationFilter jwtAuthenticationFilter;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .csrf(AbstractHttpConfigurer::disable)

```

```

        .authorizeHttpRequests(auth -> auth
            .requestMatchers("/api/dashboard/**", "/api/requests/my").authenticated()
            .requestMatchers("/api/requests/**").authenticated()
            .requestMatchers("/auth/**").permitAll()
            .requestMatchers("/api/categories/**").permitAll()
            .requestMatchers("/", "/assets/**").permitAll()
            .anyRequest().permitAll()
        )
        .formLogin(AbstractHttpConfigurer::disable)
        .httpBasic(AbstractHttpConfigurer::disable)
        .sessionManagement(session ->
            session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .addFilterBefore(jwtAuthenticationFilter,
                UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }
}

```

Лістинг A.6.2 — IdentityServiceClient.java

```

package org.volunteer.webapp.identity;

import lombok.extern.slf4j.Slf4j;
import org.springframework.http.*;
import org.springframework.stereotype.Service;
import org.springframework.web.reactive.function.client.*;
import org.volunteer.webapp.config.ApiGatewayProperties;
import org.volunteer.webapp.identity.dto.UserProfileResponse;

@Slf4j
@Service
public class IdentityServiceClient {

    private final WebClient webClient;

    public IdentityServiceClient(WebClient.Builder webClientBuilder, ApiGatewayProperties props) {
        this.webClient = webClientBuilder
            .baseUrl(props.baseUrl())
            .defaultHeader(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE)
            .build();
    }

    public UserProfileResponse getProfile(String jwtToken) {
        try {
            return webClient.get()
                .uri("/api/v1/identity/profile")
                .header(HttpHeaders.AUTHORIZATION, "Bearer " + jwtToken)
                .retrieve()
                .bodyToMono(UserProfileResponse.class)
                .block();
        } catch (WebClientResponseException e) {
            log.error("Failed to get user profile: {} - {}", e.getStatusCode(),
                e.getResponseBodyAsString());
            throw e;
        }
    }
}

```

Лістинг A.6.3 — RequestServiceClient.java

```
package org.volunteer.webapp.request;

import java.util.List;
import org.springframework.http.*;
import org.springframework.stereotype.Service;
import org.springframework.web.reactive.function.client.WebClient;
import org.volunteer.webapp.config.ApiGatewayProperties;
import org.volunteer.webapp.request.dto.*;

@Service
public class RequestServiceClient {

    private final WebClient webClient;

    public RequestServiceClient(WebClient.Builder webClientBuilder, ApiGatewayProperties
props) {
        this.webClient = webClientBuilder
            .baseUrl(props.baseUrl())
            .defaultHeader(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE)
            .build();
    }

    public List<RequestResponse> findByCreator(Long creatorId) {
        return webClient.get()
            .uri("/api/v1/requests/creator/{creatorId}", creatorId)
            .retrieve()
            .bodyToFlux(RequestResponse.class)
            .collectList()
            .block();
    }

    public RequestResponse createRequest(CreateRequestRequest request) {
        return webClient.post()
            .uri("/api/v1/requests")
            .bodyValue(request)
            .retrieve()
            .bodyToMono(RequestResponse.class)
            .block();
    }

    public RequestResponse getByPublicId(String publicId) {
        return webClient.get()
            .uri("/api/v1/requests/{publicId}", publicId)
            .retrieve()
            .bodyToMono(RequestResponse.class)
            .block();
    }
}
```

Лістинг A.6.4 — LoginRequest.java

```
package org.volunteer.webapp.identity.dto;

import jakarta.validation.constraints.*;
import lombok.*;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class LoginRequest {
```

```
    @NotBlank(message = "Email is required")
    @Email(message = "Please provide a valid email")
    private String email;

    @NotBlank(message = "Password is required")
    @Size(min = 8, max = 72)
    private String password;
}
```

Лістинг A.6.5 — TokenResponse.java

```
package org.volunteer.webapp.identity.dto;

import lombok.*;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class TokenResponse {
    private String accessToken;
    private String refreshToken;
    private long expiresIn;
    private long refreshExpiresIn;
}
```