НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

"На правах рукопису"
УДК_____

«До захисту допущено»
Завідувач кафедри
_____ Наталія АУШЕВА
“___”_____2025р.

Магістерська дисертація
на здобуття ступеня другого (магістерського) рівня вищої освіти
за освітньою програмою “Цифрові технології в енергетиці”
зі спеціальності 122 “Комп’ютерні науки”

на тему: Методи створення веб-платформи для організації волонтерської діяльності

Виконала : студентка _2__курсу, групи ТР-41мп
_____ЧИЖ Єлизавета Олегівна_____ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник к.т.н., доцент Світлана ШАПОВАЛОВА _____
(науковий ступінь, вчене звання, ім’я ПРІЗВИЩЕ) (підпис)

Рецензент доцент кафедри ТАЕ, к.т.н., доцент
Артур РАЧИНСЬКИЙ _____
(посада, науковий ступінь, вчене звання, ім’я ПРІЗВИЩЕ) (підпис)

Н.контроль старший викладач Ольга БЕСПАЛА _____
(посада, ім’я ПРІЗВИЩЕ) (підпис)

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

Київ – 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ
Рівень вищої освіти – другий (магістерський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА

«_____»_____2025 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

ЧИЖ Єлизаветі Олегівні
(прізвище, ім’я, по батькові)

1. Тема роботи Методи створення веб-платформи для організації волонтерської діяльності

керівник роботи Шаповалова Світлана Ігорівна, к.т.н., доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «03» листопада 2025 р. № 4779-с

2. Термін подання студентом дисертації 06.12.2025

1. Об’єкт дослідження : інформаційна модель процесів організації та координації волонтерської діяльності.
2. Вихідні дані: Nuxt.js, Vue.js, Pinia, TypeScript, Tailwind CSS, PrimeVue, Node.js
3. Перелік завдань, які потрібно розробити:

1) провести аналіз існуючих методів та програмних засобів реалізації сучасних веб-платформ;

2) спроєктувати та програмно реалізувати модель розмежування доступу на основі ролей (RBAC);

3) розробити функціональну веб-платформу для координації волонтерської діяльності з використанням обраних методів.

4. Орієнтовний перелік графічного (ілюстративного) матеріалу: концептуальна схема, фізична ER-діаграма, діаграма послідовності потоку даних, інтерфейс платформи.

5. Орієнтовний перелік публікацій:

1) Чиж Є. О. Методи створення веб-платформи для організації волонтерської діяльності // Modern Perspectives on Science and Economic Progress: Collection of Scientific Papers with Proceedings of the 2nd International Scientific and Practical Conference. – 2025.

2) Shapovalova S., Chyzh Ye. Architectural approaches to implementing a Role-Based Access Control (RBAC) model for modern web platforms // Progressive Approaches in Science and Engineering: Proceedings of the 2nd International Scientific and Practical Conference. – 2025.

6. Дата видачі завдання *16.09.2024 р.*

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	1. Прибуття студента на базу проведення практики	01-07.09.25	Виконано
2	Ознайомлення з історією та діяльністю підприємства. Узгодження завдання переддипломної практики.	01-07.09.25	Виконано
3	Вивчення відкритих інформаційних технологій та опанування спеціалізованого програмного забезпечення.	01-14.09.25	Виконано
4	Виконання завдання переддипломної практики. Робота щодо підготовки матеріалів за темою дипломної роботи.	15.09-26.10.25	Виконано
5	Постановка задачі. Формування ТЗ.	15-21.09.25	Виконано
6	Підбір та ознайомлення зі спеціальною літературою.	15-28.09.25	Виконано
7	Розробка дизайну веб-продукту. Збірка програми	29.09-12.10.25	Виконано
8	Тестування та усунення недоліків у роботі програмного продукту.	06-19.10.25	Виконано
9	Захист програмного продукту за темою дипломної роботи.	20-26.10.25	Виконано
10	Оформлення щоденника і звіту з практики.	20-26.10.25	Виконано
11	Складання заліку з практики.	21.10.25	Виклнано

Студентка _____ Єлизавета ЧИЖ
(підпис) (ім'я, ПРІЗВИЩЕ)

Науковий керівник _____ Світлана ШАПОВАЛОВА
(підпис) (ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Актуальність теми магістерської роботи обумовлена необхідністю застосування передових методів розробки для створення гнучкого та надійного веб-інструментарію для координації волонтерських процесів. Забезпечення централізованої та автоматизованої взаємодії вимагає впровадження сучасних методів комп'ютерних наук: мікросервісної архітектури для підвищення стійкості, ефективного управління базами даних та передового UX/UI-дизайну. Такий підхід дозволяє значно підвищити ефективність, прозорість та керованість волонтерського руху.

Мета роботи полягає у розробці та реалізації оптимальної за характеристиками веб-платформи, яка забезпечує підвищену надійність функціонування та потребує мінімальної кількості ресурсів з боку користувача.

Для досягнення поставленої мети вирішувалися такі завдання дослідження:

провести аналіз існуючих методів та програмних засобів реалізації сучасних веб-платформ;

спроєктувати та програмно реалізувати модель розмежування доступу на основі ролей (RBAC);

розробити функціональну веб-платформу для координації волонтерської діяльності з використанням обраних методів;

провести комплексне тестування розробленої веб-платформи для перевірки її надійності, продуктивності та відповідності вимогам.

Об’єкт дослідження – інформаційна модель процесів організації та координації волонтерської діяльності.

Предмет дослідження – методи реалізації веб-платформ, засновані на рольовій моделі доступу.

Методи дослідження. У роботі використано комплекс методів: метод системного аналізу для вивчення предметної області; метод порівняльного аналізу для обґрунтування вибору технологічних рішень; метод об'єктно-орієнтованого моделювання для розробки логічної структури платформи. На етапі реалізації

застосовувалися методи розробки програмного забезпечення, а для підтвердження коректності – методи експериментального тестування.

Апробація результатів магістерської дисертації. Основні положення та результати дослідження доповідалися та обговорювалися на міжнародних науково-практичних конференціях: II Міжнародній науково-практичній конференції «Modern Perspectives on Science and Economic Progress» (Вільнюс, Литва, 2025 р.) та II Міжнародній науково-практичній конференції «Progressive Approaches in Science and Engineering» (Копенгаген, Данія, 2025 р.).

За результатами конференцій опубліковано такі праці:

1. Чиж Є. О. Методи створення веб-платформи для організації волонтерської діяльності / Є. О. Чиж // Modern Perspectives on Science and Economic Progress : Collection of Scientific Papers with Proceedings of the 2nd International Scientific and Practical Conference : міжнародна наукова конференція. – Вільнюс, 2025. – С. 74–77.

2. Shapovalova S. Architectural approaches to implementing a Role-Based Access Control (RBAC) model for modern web platforms / S. Shapovalova, Ye. Chyzh // Progressive Approaches in Science and Engineering : Proceedings of the 2nd International Scientific and Practical Conference. – Copenhagen, 2025. – P. 314–315.

Структура та обсяг магістерської дисертації. Робота складається зі вступу, 5 розділів, висновків, списку використаних джерел із 27 найменувань. Повний обсяг роботи становить 80 сторінок.

Ключові слова: веб-платформа, волонтерська діяльність, рольова модель доступу (RBAC), надійність системи, тестування продуктивності, програмна реалізація, архітектура системи.

ABSTRACT

The relevance of the topic of the master's thesis is due to the need to apply advanced development methods to create flexible and reliable web tools for coordinating volunteer processes. Ensuring centralized and automated coordination requires the implementation of modern computer science methods: microservice architecture for increased resilience, effective database management, and advanced UX/UI design. This approach significantly increases the efficiency, transparency, and manageability of the volunteer movement.

The purpose of the work is to develop and implement an optimal web platform based on its characteristics, which ensures increased operational reliability and requires a minimum amount of user resources.

To achieve this goal, the following research tasks were solved:

to analyze existing methods and software tools for implementing modern web platforms;

to design and software implement a role-based access control (RBAC) model;

to develop a functional web platform for coordinating volunteer activities using the selected methods;

to conduct comprehensive testing of the developed web platform to verify its reliability, performance, and compliance with requirements.

The object of research is the information model of processes for organizing and coordinating volunteer activities. The subject of research is methods of web platform implementation based on a role-based access model.

Research methods. A complex of methods was used in the work: system analysis method for studying the subject area; comparative analysis method for justifying the choice of technological solutions; object-oriented modeling method for developing the logical structure of the platform. Software development methods were used at the implementation stage, and experimental testing methods were used to confirm correctness.

Approbation of the results of the master's thesis. The main provisions and results of the research were reported and discussed at international scientific and practical

conferences: the 2nd International Scientific and Practical Conference "Modern Perspectives on Science and Economic Progress" (Vilnius, Lithuania, 2025) and the 2nd International Scientific and Practical Conference "Progressive Approaches in Science and Engineering" (Copenhagen, Denmark, 2025).

Based on the results of the conferences, the following papers were published:

1. Chyzh, Ye. O. (2025). Methods of creating a web platform for organizing volunteer activities. In Modern Perspectives on Science and Economic Progress: Collection of Scientific Papers with Proceedings of the 2nd International Scientific and Practical Conference (pp. 74–77). Vilnius.

2. Shapovalova, S., & Chyzh, Ye. (2025). Architectural approaches to implementing a Role-Based Access Control (RBAC) model for modern web platforms. In Progressive Approaches in Science and Engineering: Proceedings of the 2nd International Scientific and Practical Conference (pp. 314–315). Copenhagen.

Structure and volume of the master's thesis. The work consists of an introduction, 5 chapters, conclusions, a list of used sources containing 27 titles. The full volume of the work is 80 pages.

Keywords: web platform, volunteer activity, role-based access control (RBAC), system reliability, performance testing, software implementation, system architecture.

ЗМІСТ

ВСТУП

У сучасному суспільстві волонтерська діяльність набула стратегічного значення, виступаючи ключовим чинником у подоланні соціальних, економічних та гуманітарних криз. Ефективність цієї діяльності критично залежить від оперативності, прозорості та якості координації між волонтерами, координаторами та бенефіціарами. Саме тут виникає наукова проблема, яка полягає у відсутності універсальних, високонадійних та масштабованих цифрових інструментів, здатних централізувати та автоматизувати складні багаторівневі волонтерські процеси. Існуючі методи та IT-рішення часто характеризуються фрагментацією даних, низьким рівнем інтеграції та обмеженою здатністю до адаптації в умовах швидко зростаючого попиту на допомогу. Критичний аналіз показує, що більшість платформ або зосереджені лише на одній функції, або мають застарілу архітектуру, яка ускладнює їх масштабування.

Актуальність теми магістерської роботи обумовлена необхідністю застосування передових методів розробки для створення гнучкого, надійного веб-інструментарію. Забезпечення централізованої та автоматизованої координації волонтерських процесів вимагає впровадження сучасних методів комп'ютерних наук, зокрема: мікросервісної архітектури для підвищення стійкості та незалежного розгортання модулів, ефективного управління базами даних для швидкої обробки транзакцій та передового UX/UI-дизайну для максимальної залученості користувачів. Такий підхід дозволить значно підвищити ефективність, прозорість та керованість волонтерського руху.

Мета роботи: розробка та реалізація оптимальної за характеристиками веб-платформи, яка забезпечує підвищену надійність функціонування та потребує мінімальної кількості ресурсів з боку користувача.

Об'єктом дослідження є інформаційна модель процесів організації та координації волонтерської діяльності.

Предмет дослідження: методи реалізації веб-платформ, засновані на рольовій моделі доступу.

Для досягнення поставленої мети використовувався комплекс методів дослідження, включаючи метод системного аналізу для вивчення предметної області та метод порівняльного аналізу для обґрунтування вибору технологічних рішень. Метод об'єктно-орієнтованого моделювання був задіяний для розробки логічної структури платформи. На етапі реалізації застосовувалися методи розробки програмного забезпечення, а для підтвердження надійності та коректності – методи експериментального тестування та апробації.

Завдання, які потрібно виконати:

1. Провести аналіз існуючих методів та програмних засобів реалізації сучасних веб-платформ.

2. Спроєктувати та програмно реалізувати модель розмежування доступу на основі ролей (RBAC).

3. Розробити функціональну веб-платформу для координації волонтерської діяльності з використанням обраних методів.

4. Провести комплексне тестування розробленої веб-платформи для перевірки її надійності, продуктивності та відповідності вимогам.

Основні положення та результати дослідження доповідалися та обговорювалися на міжнародних науково-практичних конференціях: II Міжнародній науково-практичній конференції «Modern Perspectives on Science and Economic Progress» (Вільнюс, Литва, 2025 р.) та II Міжнародній науково-практичній конференції «Progressive Approaches in Science and Engineering» (Копенгаген, Данія, 2025 р.).

За результатами конференцій опубліковано такі праці:

1. Чиж Є. О. Методи створення веб-платформи для організації волонтерської діяльності / Є. О. Чиж // Modern Perspectives on Science and Economic Progress : Collection of Scientific Papers with Proceedings of the 2nd International Scientific and Practical Conference : міжнародна наукова конференція. – Вільнюс, 2025. – С. 74–77.

2. Shapovalova S. Architectural approaches to implementing a Role-Based Access Control (RBAC) model for modern web platforms / S. Shapovalova, Ye. Chyzh //

Progressive Approaches in Science and Engineering : Proceedings of the 2nd International Scientific and Practical Conference. – Copenhagen, 2025. – P. 314–315.

Загальний зміст роботи відображено в її структурі та обсязі. Магістерська дисертація складається зі вступу, чотирьох розділів, висновків до кожного розділу, загальних висновків, списку використаних джерел (27 джерел). Загальний обсяг дисертації – 80 сторінок. Роботу проілюстровано 9 таблицями та 17 рисунками.

1 МЕТОДИ РЕАЛІЗАЦІЇ ВЕБ-ПЛАТФОРМ

У сучасному суспільстві, веб-технології стали ключовим інструментом для соціальної мобілізації та координації. Вони забезпечують миттєвий обмін інформацією, дозволяють керувати складними процесами та об'єднувати спільноти. Однією з найважливіших сфер, де ці технології відіграють критичну, життєво необхідну роль, є волонтерська діяльність. Ефективність гуманітарної допомоги та громадських ініціатив, особливо в кризових умовах, напряму залежить від швидкості, прозорості та якості комунікації. Саме веб-платформи для організації волонтерської діяльності стають тією цифровою інфраструктурою, яка дозволяє автоматизувати ці процеси, ефективно розподіляти ресурси та масштабувати зусилля тисяч людей.

1.1 Завдання та вимоги для розробки веб-застосунку

Формулювання завдання для розробки будь-якого комплексного веб-застосунку починається з визначення його ключової мети та архітектурних основ. У контексті соціально-орієнтованих платформ, таких як системи для організації волонтерської діяльності, головним завданням є створення єдиного, надійного цифрового простору. Цей простір має забезпечувати ефективний зв'язок між різними групами користувачів, зокрема між тими, хто пропонує допомогу (волонтерами), та тими, хто її потребує або організовує (організаціями). Кінцевою метою є спрощення процесів пошуку, управління та участі у відповідній діяльності, забезпечуючи позитивний та безперебійний досвід для всіх учасників.

Для успішного вирішення цього завдання, фундаментальною вимогою є проектування системи на основі рольової моделі доступу (Role-Based Access Control, RBAC). Такий підхід є критично важливим для складних платформ, оскільки він дозволяє чітко сегментувати функціонал та права доступу. Надання виділених порталів або інтерфейсів для кожної ролі користувачів суттєво спрощує

взаємодію з системою та підвищує її безпеку. У типовій системі координації можна виділити щонайменше три основні ролі. [1]

Перша роль — це кінцевий учасник. Функціональні вимоги для цієї ролі зосереджені на персоналізації та взаємодії. Система повинна надавати повний набір інструментів для управління ідентифікацією, включаючи безпечні механізми реєстрації нового облікового запису, автентифікації (входу) та процедури відновлення доступу у випадку втрати пароля. Не менш важливою є можливість управління профілем, де учасник може вказувати та оновлювати свою особисту інформацію, і що особливо важливо — деталізувати свої навичкита компетенції. Саме ця інформація є ключовою для подальших алгоритмів зіставлення. Нарешті, основна функція для цієї ролі — це доступ до каталогу можливостей, де учасник може переглядати, фільтрувати та знаходити актуальні завдання або події, до яких він може долучитися.

Друга роль — це координатор або менеджер організації. Ця роль є постачальником завдань у системі. Вимоги для неї включають не лише базові функції автентифікації та управління профілем, але й значно ширший набір інструментів управління. Ключовим завданням для цієї ролі є управління життєвим циклом сутностей "проєкт" та "подія". Це означає, що координатори повинні мати можливість створювати нові волонтерські проєкти (довгострокові ініціативи) та події (конкретні заходи), редагувати їхні деталі, встановлювати вимоги та керувати їхніми статусами. Окрім цього, система повинна надавати їм інструменти для управління учасниками, тобто можливість переглядати список волонтерів, які відгукнулися або були призначені на їхні заходи.

Третя, найвища роль — це адміністратор або менеджер платформи. Ця роль відповідає за загальний стан, якість та безпеку всієї системи. Вимоги для цієї ролі є виключно адміністративними. По-перше, це наявність централізованої панелі приладівб, яка надає огляд ключових метрик та загальної активності на платформі в режимі реального часу. По-друге, це функції модерації та верифікації. Адміністратор повинен мати повноваження переглядати, затверджувати або відхиляти нові організації та проєкти, що створюються в системі, забезпечуючи таким чином дотримання стандартів якості та безпеки. По-третє, це глобальне

управління користувачами, що включає можливість керувати обліковими записами всіх типів, вирішувати конфлікти та керувати загальноплатформенними налаштуваннями. [2] Для наглядності функціональні вимоги описані у таблиці 1.1.

Таблиця 1.1 – Функціональні вимоги

Категорія/Роль	Ключові вимоги
Волонтер	- Автентифікація: реєстрація, вхід, відновлення паролю. - Управління профілем: редагування особистої інформації та навичок. - Пошук: перегляд та фільтрація списку доступних подій.
Організація	- Автентифікація: реєстрація організації, вхід. - Управління профілем: редагування профілю організації та контактної інформації. - Управління діяльністю: створення та керування проєктами та подіями. - Управління волонтерами: перегляд та керування залученими волонтерами.
Менеджер платформи	- Моніторинг: доступ до Dashboard з оглядом активності. - Модерація: затвердження або відхилення нових організацій та проєктів. - Адміністрування: керування всіма обліковими записами користувачів. - Налаштування: керування загальноплатформенними параметрами.

Окрім цього деталізованого переліку функціональних вимог, що описують, що система повинна робити, не менш важливими є нефункціональні вимоги, які визначають, як вона повинна це робити.

Якість користувацького досвіду (UX) та доступність - інтерфейс має бути інтуїтивно зрозумілим, професійним та приємним у використанні, оскільки низька якість UX є головним фактором відтоку користувачів. Це також включає вимоги до доступності, щоб платформою могли користуватися люди з різними можливостями. [1,2]

Зручність обслуговування та масштабованість - архітектура має бути структурованою та модульною, оскільки веб-застосунок вимагає постійних оновлень. Це спрощує процес розробки, підтримки та дозволяє легко масштабувати кодову базу в майбутньому.

Економічна ефективність та швидкість розробки - вибір програмних засобів має бути спрямований на прискорення процесу розробки. Використання сучасних фреймворків та бібліотек компонентів дозволяє зменшити витрати на робочу силу та швидко реагувати на потреби спільноти.

Продуктивність та оптимізація для пошукових систем - платформа повинна бути видимою в пошукових системах для залучення нових користувачів. Це висуває жорсткі вимоги до методів рендерингу. Висока швидкість завантаження та переваги у SEO є критичними для забезпечення органічного трафіку. Для наглядності нефункціональні вимоги описані у таблиці 1.2.

Таблиця 1.2 – Нефункціональні вимоги

Категорія/Роль	Ключові вимоги
1. UX / Доступність	- Забезпечення високої якості користувацького досвіду (UX). - Інтерфейс має бути інтуїтивно зрозумілим, професійним та доступним (accessibility).
2.Обслуговування/ Масштабованість	- Архітектура має бути структурованою та модульною. - Забезпечення зручності підтримки та легкого масштабування кодової бази в майбутньому.
3.Ефективність/ Швидкість розробки	- Використання інструментів, що прискорюють розробку (Time-to-Market). - Зменшення витрат на робочу силу для швидкої реакції на потреби спільноти.
4.Продуктивність/ SEO	- Висока швидкість завантаження сторінок. - Оптимізація для пошукових систем (SEO) для залучення органічного трафіку (наприклад, через SSR).

Таким чином, комплексне завдання розробки веб-платформи для волонтерства полягає у проектуванні системи, яка не лише задовольняє складний набір функціональних вимог для трьох різних ролей, але й відповідає суворим нефункціональним вимогам щодо UX, продуктивності, SEO та зручності обслуговування. [2,3]

1.2 Аналіз методів та програмних архітектур для реалізації веб-платформи

Проектування сучасної цифрової екосистеми для координації волонтерської діяльності базується на необхідності вирішення фундаментального протиріччя між забезпеченням динамічного, інтерактивного користувацького досвіду та жорсткими вимогами щодо пошукової оптимізації та швидкодії. Комплексний підхід до створення такої системи передбачає глибинний аналіз існуючих парадигм рендерингу веб-застосунків, принципів побудови масштабованої компонентної архітектури інтерфейсу, а також методологій організації та оптимізації стилів. Вибір конкретних технологічних рішень має спиратися на теоретичне обґрунтування їх здатності задовольнити специфічні вимоги предметної області.

1.2.1 Архітектурні патерни веб-застосунків

В інженерії веб-систем критично важливим рішенням є вибір стратегії відтворення контенту, оскільки це безпосередньо впливає на метрики продуктивності та оптимізацію для пошукових систем (SEO). Існує кілька базових підходів до вирішення цього завдання.

Перший — це клієнтський рендеринг (Client-Side Rendering, CSR), модель якого передбачає завантаження браузером мінімального HTML-каркаса та об'ємного JavaScript-файлу, що динамічно генерує контент.

Другий підхід — серверний рендеринг (Server-Side Rendering, SSR), при якому HTML-код сторінки повністю формується на сервері для кожного окремого запиту.

Третім, найбільш прогресивним методом для соціально значущих платформ, є гібридний підхід (Universal Rendering або Isomorphic JavaScript). Він поєднує переваги обох попередніх методів: початкове завантаження сторінки відбувається через SSR, що забезпечує швидке відображення та якісне SEO, після чого відбувається процес гідратації. Гідратація перетворює отриманий від сервера статичний HTML у динамічний застосунок шляхом прикріплення обробників подій, що дозволяє поєднати видимість для пошукових систем із високою інтерактивністю односторінкових застосунків (SPA). [4]

Таблиця 1.3 - Порівняльний аналіз базових моделей управління доступом

Критерій порівняння	SPA (Client-Side Rendering)	SSR (Server-Side Rendering)	Гібридний підхід (Universal/Isomorphic)
Оптимізація для пошукових систем (SEO)	Низька (потребує виконання JS ботами)	Висока (контент доступний одразу)	Висока (поєднує переваги SSR)
Час до першого відмальовування (FCP)	Повільно (чекає завантаження JS-бандлу)	Швидко (браузер отримує готовий HTML)	Швидко
Інтерактивність після завантаження	Висока (сторінка не перезавантажується)	Низька (перезавантаження при навігації)	Висока (після процесу гідратації)
Навантаження на сервер	Мінімальне (лише API запити)	Високе (генерація кожної сторінки)	Середнє/Високе (SSR для першого входу + API)
Складність розробки та підтримки	Середня	Середня	Висока (потребує спеціалізованих фреймворків)
Відповідність вимогам платформи	Часткова (підходить для адмін-панелі)	Часткова (підходить для статичних сайтів)	Повна (забезпечує і SEO, і UX)

У таблиці 1.3 зроблений порівняльний аналіз цих моделей управління доступом.

1.2.2 Методи організації інтерфейсу користувача

Створення інтерфейсу вимагає застосування компонентно-орієнтованого підходу. Суть методу полягає у декомпозиції інтерфейсу на незалежні, ізольовані модулі, які містять власну логіку, структуру та стилі.

В основі цього методу базуються три принципи, такі як:

1) інкапсуляція - кожен компонент містить власну логіку, представлення та, в деяких підходах, стилі. Це локалізує зміни та запобігає побічним ефектам при оновленні коду.

2) декларативність - розробник описує, *як* має виглядати інтерфейс у певному стані даних, а не покроково інструктує браузер, як змінювати DOM. Використання Virtual DOM дозволяє мінімізувати дорогі операції перемальовування сторінки, обчислюючи різницю між станами та вносячи лише необхідні зміни.

3) односпрямований потік даних - дані передаються від батьківських компонентів до дочірніх, що робить поведінку системи передбачуваною та спрощує відлагодження.

Застосування готових рішень у вигляді бібліотек компонентів позбавляє від необхідності реалізації рутинної низькорівневої логіки складних елементів інтерфейсу, вивільняючи час для фокусування на ключовій бізнес-логіці застосунку. [5]

1.3 Підходи до розробки веб-платформи

Визначені у попередніх підрозділах вимоги до веб-платформи, зокрема необхідність поєднання високої інтерактивності користувацького інтерфейсу з ефективною пошуковою оптимізацією через гібридний рендеринг, суттєво

звужують коло потенційних технологічних рішень. Традиційні серверні фреймворки не забезпечують необхідного рівня реактивності інтерфейсу, тоді як чисті SPA-бібліотеки без додаткової інфраструктури не вирішують проблем SEO. [6]

Отже, аналіз інструментальних засобів доцільно зосередити на сучасних JavaScript-екосистемах, які підтримують компонентний підхід та мають зрілі рішення для серверного рендерингу. Основними кандидатами на роль технологічної бази платформи є екосистеми React, Angular та Vue.js. Розглянемо кожну з них детальніше:

1. Екосистема React (з використанням Next.js).

React, розроблений компанією Meta (Facebook), є найпопулярнішою у світі бібліотекою для побудови користувацьких інтерфейсів. Для реалізації задач, поставлених у цій роботі, React використовується в тандемі з мета-фреймворком Next.js.

Перевагами є величезна спільнота та екосистема готових рішень. Next.js надає потужні інструменти для гібридного рендерингу, статичної генерації та оптимізації зображень. Використання JSX дозволяє гнучко описувати структуру компонентів безпосередньо в JavaScript-коді.

Недоліки в контексті проєкту є високий поріг входження, особливо при необхідності вивчення супутніх бібліотек для управління станом та маршрутизації, оскільки сам React є лише бібліотекою відображення, а не повноцінним фреймворком. Надмірна гнучкість може призводити до неузгодженості коду в команді за відсутності суворих гайдлайнів.

2. Фреймворк Angular.

Angular, що розвивається компанією Google, є повноцінним MVC-фреймворком корпоративного рівня. Він пропонує "все в одному": власну систему модулів, маршрутизацію, управління формами та HTTP-клієнт.

Жорстка стандартизація архітектури та використання TypeScript є перевагами, що за замовчуванням забезпечують високу надійність та передбачуваність коду, що є критичним для великих довгострокових проєктів. Для реалізації SSR використовується технологія Angular Universal.

Недоліками є найвищий поріг входження серед аналогів через складність концепцій. Angular часто критикують за надмірну багатослівність коду (verbosity) та більший розмір підсумкового JS-бандлу, що може негативно вплинути на швидкість першого завантаження на мобільних пристроях волонтерів.

3. Екосистема Vue.js (з використанням Nuxt.js).

Vue.js — це прогресивний фреймворк, який позиціонується як золота середина між гнучкістю React та структурованістю Angular. Його ключовою особливістю є інтуїтивно зрозумілий синтаксис шаблонів, заснований на звичайному HTML, та реактивна система, що автоматично відстежує залежності.

Для побудови масштабованих застосунків із підтримкою SSR використовується мета-фреймворк Nuxt.js, який надає стандартизовану структуру директорій, автоматичну маршрутизацію на основі файлової системи та вбудовану підтримку серверного рендерингу. [7,8]

Найнижчий поріг входження, що дозволяє прискорити процес розробки. Чітке розділення логіки, шаблону та стилів в межах одного файлу компонента спрощує підтримку коду, що є перевагою для розробників. Nuxt.js автоматизує складні конфігурації Webpack/Vite та SSR, дозволяючи зосередитися на бізнес-логіці.

Менша частка ринку порівняно з React, що може означати меншу кількість готових вузькоспеціалізованих бібліотек, хоча для потреб волонтерської платформи існуючих рішень достатньо.

Таблиця 1.4 - Порівняння платформ

Критерій порівняння	React + Next.js	Angular + Universal	Vue.js + Nuxt.js (Обраний стек)
Підтримка гібридного рендерингу (SSR)	Відмінна (нативна в Next.js)	Добра (через Angular Universal)	Відмінна (нативна в Nuxt.js)
Швидкість розробки та поріг входження	Середня (потребує вивчення екосистеми)	Низька (високий поріг входження, багатослівність)	Висока (інтуїтивний синтаксис, "магія" Nuxt)

Продовження таблиці 1.4

Структурованість та стандартизація	Низька (залежить від команди)	Висока (закладена фреймворком)	Висока (завдяки конвенціям Nuxt.js)
Екосистема UI-бібліотек	Величезна (MUI, AntD)	Велика (Angular Material)	Велика (PrimeVue, Vuetify)
Відповідність вимогам проєкту	Висока	Середня (надлишкова складність)	Найвища (баланс швидкості та можливостей)

Для наочного порівняння розглянутих платформ у таблиці 1.4 оберемо ключові критерії, що відповідають нефункціональним вимогам проєкту: підтримка SSR, швидкість розробки, продуктивність та екосистема UI-компонентів.

1.3.1 Фреймворк для побудови веб-застосунку Nuxt.js

В якості основи для побудови клієнтської та серверної частини платформи обрано мета-фреймворк Nuxt.js, що базується на екосистемі Vue.js. Цей вибір обумовлений рядом переваг як з точки зору процесу розробки, так і з погляду бізнес-ефективності.

Інженерні переваги - Nuxt.js надає потужну та структуровану архітектуру "з коробки". Його підхід "конвенція понад конфігурацією" (convention-over-configuration), модульна структура та вбудовані функції, такі як автоматична маршрутизація та гібридний рендеринг (SSR + SPA), значно спрощують процес розробки. Це робить кодову базу більш зрозумілою, легшою для підтримки та масштабування в майбутньому. Велика екосистема готових модулів дозволяє швидко інтегрувати необхідну функціональність (наприклад, для автентифікації чи роботи з PWA) без написання зайвого коду. [9]

Бізнес-переваги - використання Nuxt.js дозволяє прискорити вихід продукту на ринок (Time-to-Market) завдяки швидкій розробці нових функцій. Будучи фреймворком з відкритим кодом та великою спільнотою розробників Vue.js, він є

економічно ефективним вибором. Критично важливою бізнес-перевагою є підтримка серверного рендерингу (SSR), що покращує продуктивність завантаження сторінок та забезпечує якісну пошукову оптимізацію (SEO), напряму впливаючи на залучення органічного трафіку та задоволеність користувачів.

Обраний технологічний стек, що включає Nuxt.js, TypeScript, Tailwind CSS, PrimeVue та Pinia, є збалансованим рішенням, яке відповідає сучасним вимогам до розробки високонавантажених веб-платформ. Цей комплекс інструментів дозволяє не лише ефективно вирішити інженерні задачі (продуктивність, надійність, масштабованість), але й забезпечує вагомі бізнес-переваги, такі як прискорення виходу на ринок, економічна ефективність розробки, висока якість користувацького досвіду та відповідність стандартам SEO. [9]

1.3.2 Мова програмування TypeScript та забезпечення надійності

TypeScript (TS) — це мова програмування, яка розширює JavaScript, додаючи до нього статичну типізацію. Вона була розроблена і підтримується компанією Microsoft. Основна ідея TypeScript полягає в тому, щоб покращити процес розробки великих додатків, роблячи код більш надійним, зрозумілим і легким для підтримки. [10]

Основні характеристики TypeScript:

1. Статична типізація - це найголовніша особливість TypeScript. Вона дозволяє вам явно вказувати типи змінних, функцій, аргументів і повертаних значень. Це допомагає виявити помилки на ранніх етапах розробки, ще до запуску програми.

2. Інтерфейси в TypeScript дозволяють вам визначати структуру об'єктів. Це допомагає зробити ваш код більш читабельним і зрозумілим, а також полегшує взаємодію з іншими розробниками.

3. Класи та об'єкти - TypeScript підтримує об'єктно-орієнтоване програмування, включаючи класи, об'єкти, успадкування, поліморфізм та інші концепції. Це робить його ідеальним вибором для розробки складних додатків.

4. Модульність - TypeScript підтримує модульну архітектуру, що дозволяє вам розбивати ваш код на менші, незалежні частини. Це полегшує керування великими проектами і робить код більш придатним для повторного використання.

5. Сумісність з JavaScript - TypeScript повністю сумісний з JavaScript. Ви можете використовувати будь-яку бібліотеку або фреймворк JavaScript в своєму проекті TypeScript.

Використання TypeScript пропонує ряд суттєвих переваг для процесу розробки. Завдяки статичній типізації стає можливим виявлення багатьох помилок ще на ранніх етапах, до запуску програми, що значно економить час та зусилля на подальше налагодження. Крім того, явна типізація та використання інтерфейсів роблять код більш зрозумілим та легким для читання, покращуючи його загальну якість. Важливою перевагою є також підтримка всіх сучасних можливостей JavaScript, включаючи стандарти ES6, ES7 та інші, що доповнюється наявністю великої спільноти розробників та багатої екосистеми інструментів і бібліотек. [10]

Однак, застосування TypeScript пов'язане і з певними недоліками. Мова має крутіший поріг входження порівняно з JavaScript, що вимагає додаткового часу на вивчення нових концепцій, таких як статична типізація та інтерфейси. Також TypeScript може генерувати більший обсяг коду, ніж чистий JavaScript, що потенційно здатне збільшити час завантаження додатку. Окрім цього, можуть виникати потенційні проблеми з сумісністю при спробі інтеграції деяких старих бібліотек або фреймворків, які не були розраховані на роботу з TypeScript.

1.3.3 Бібліотека компонентів інтерфейсу PrimeVue

Для оптимізації процесу розробки клієнтської частини, забезпечення високої якості та візуальної єдності користувацького інтерфейсу (UI) було прийнято рішення використовувати бібліотеку готових компонентів PrimeVue. Цей вибір базується на комплексному аналізі інженерних та бізнес-переваг, які надає даний інструмент у контексті створення сучасної веб-платформи. [11]

Інженерні переваги включають в себе:

1. Широкий набір готових компонентів - PrimeVue пропонує вичерпний набір із понад 90 якісних компонентів (від базових кнопок та полів введення до складних таблиць даних з фільтрацією, календарів та модальних вікон). Це дозволяє значно зекономити час розробки, який інакше довелося б витратити на проєктування, верстку та тестування цих складних елементів з нуля.

2. Повна підтримка сучасного стеку - бібліотека розроблена з урахуванням особливостей Vue 3, повністю підтримує Composition API та надає офіційні визначення типів для TypeScript. Це забезпечує сувору типізацію пропсів та подій компонентів, покращує досвід розробки (IntelliSense) та запобігає потенційним помилкам на етапі компіляції.

3. Уніфікація та консистентність UI - икористання єдиної екосистеми компонентів гарантує, що всі елементи інтерфейсу матимуть узгоджений зовнішній вигляд та передбачувану поведінку по всьому додатку, що спрощує підтримку кодової бази.

4. Гнучкість та документація - наявність детальної документації з прикладами та потужної системи тем (theming API) дозволяє легко інтегрувати компоненти та глибоко адаптувати їхній візуальний стиль під специфічні дизайн-вимоги проєкту без необхідності змінювати їхню внутрішню логіку. Активна спільнота та регулярні оновлення сприяють стабільності та безпеці використовуваних рішень.

Бізнес-перевагами є економічна ефективність та прискорення Time-to-Market: Використання професійної, готової до використання бібліотеки суттєво скорочує терміни розробки MVP та подальших ітерацій продукту, що прямо транслюється у зниження загальної вартості проєкту. Підвищення якості користувацького досвіду - PrimeVue гарантує створення професійного, відшліфованого та інтуїтивно зрозумілого інтерфейсу, що підвищує задоволеність та лояльність кінцевих користувачів. Відповідність стандартам доступності - критично важливим аспектом для соціально-орієнтованої платформи є відповідність компонентів стандартам веб-доступності (WCAG). Це дозволяє охопити ширшу аудиторію, включаючи користувачів з особливими потребами. Зменшення ризиків та легша підтримка - використання перевіреного часом рішення зменшує технологічні ризики, пов'язані з багами у самописних UI-елементах. Крім того,

стандартизований підхід прискорює адаптацію (onboarding) нових розробників у команді. [11,12]

1.4 Аналіз існуючих рішень та визначення конкурентних переваг розробленої платформи

Для об'єктивного визначення місця розробленої системи в сучасному ландшафті цифрових інструментів для волонтерства необхідно провести порівняльний аналіз конкретних рішень, що активно використовуються на міжнародному та українському ринках. Як було зазначено раніше, критичними проблемами багатьох існуючих IT-рішень є фрагментація даних, недостатня глибина інструментів управління процесами та застарілі архітектурні підходи. Аналіз проводиться крізь призму цих проблем, а також вимог до повноти рольової моделі та якості користувацького досвіду.

У ході дослідження було виявлено ряд системних недоліків у існуючих рішеннях, які обмежують їх ефективність у сучасних умовах.

Значна частина існуючих платформ, особливо тих, що розвиваються протягом тривалого часу, побудована на базі застарілих монолітних архітектур. У таких системах клієнтський інтерфейс та серверна логіка тісно пов'язані, що ускладнює впровадження нових функцій та масштабування окремих компонентів системи під навантаженням.

Крім того, багато аналогів використовують традиційний підхід до рендерингу сторінок на сервері без використання сучасних реактивних фреймворків на клієнті. Це призводить до повільнішої взаємодії з інтерфейсом (повне перезавантаження сторінки при кожній дії) та гіршого користувацького досвіду порівняно із сучасними односторінковими додатками (SPA). [13]

На противагу цьому, розроблювана платформа базується на сучасному фреймворку Nuxt.js, що дозволяє використовувати гібридну модель рендерингу

(SSR + SPA). Це забезпечує як швидке початкове завантаження та SEO-оптимізацію, так і високу інтерактивність інтерфейсу.

Критичним аспектом для платформ такого типу є якісне розмежування прав доступу між волонтерами, організаторами та адміністраторами. У багатьох проаналізованих аналогах реалізація RBAC є поверхневою: часто різні ролі використовують один і той самий перевантажений інтерфейс, де доступ до певних функцій просто приховується на рівні UI, що не гарантує належного рівня безпеки.

Пропоноване рішення ставить RBAC в основу архітектури. Платформа надає виділені, захищені маршрути та макети для кожної з трьох ролей (Волонтер, Організація, Менеджер). Це не лише підвищує безпеку даних, але й дозволяє створити спеціалізовані робочі простори, оптимізовані під конкретні задачі кожної групи користувачів, наприклад, централізовану панель управління (Dashboard) для менеджерів платформи.[13,14]

В умовах, коли більшість користувачів отримують доступ до веб-ресурсів через мобільні пристрої, якість мобільної версії є критичною. Багато існуючих платформ мають "адаптивний" дизайн, який реалізовано за залишковим принципом — десктопна версія просто стискається під менший екран, часто залишаючись незручною.

Власна розробка використовує підхід Mobile-First Responsive Design із застосуванням утилітарного фреймворку Tailwind CSS. Це гарантує, що інтерфейс проєктується в першу чергу для зручності використання на смартфонах і планшетах, забезпечуючи сучасний та чистий UI на будь-якому пристрої.

Узагальнені результати порівняльного аналізу наведено в таблиці 1.5.

Таблиця 1.5 – Порівняльна характеристика аналогів та власної розробки

Критерій порівняння	Типові існуючі аналоги	Власна розробка (Пропоноване рішення)
Архітектура	Переважно монолітна, застарілі стеки	Модульна, на базі Nuxt.js (Vue.js ecosystem)
Метод рендерингу	Традиційний серверний (MPA) або чистий SPA (проблеми з SEO)	Гібридний (SSR + SPA) для балансу продуктивності та SEO
Реалізація RBAC	Уніфікований інтерфейс із приховуванням елементів	Виділені захищені портали та маршрути для кожної ролі
Управління станом	Часто фрагментоване або відсутнє на клієнті	Централізоване, типізоване (Pinia) з модульними сховищами
Мобільна адаптація	Базова адаптивність (Desktop-first)	Mobile-First, сучасний UI (Tailwind CSS)
Типізація коду	Часто відсутня (JavaScript, PHP без суворих типів)	Повна типізація (TypeScript) для надійності

На міжнародному рівні одним із найвідоміших рішень є платформа VolunteerMatch (США). На рисунку 1.1 можемо побачити їх логотип.

Рисунок 1.1 - Платформа VolunteerMatch

Вона володіє найбільшою базою можливостей та потужними пошуковими механізмами. Однак, її архітектура несе ознаки застарілих підходів, інтерфейс часто не відповідає сучасним стандартам адаптивності, а функціонально платформа діє переважно як агрегатор вакансій ("дошка оголошень"), де

комунікація та управління волонтерами після подачі заявки виводяться за межі системи. Наглядно бачимо це на рисунку 1.2.

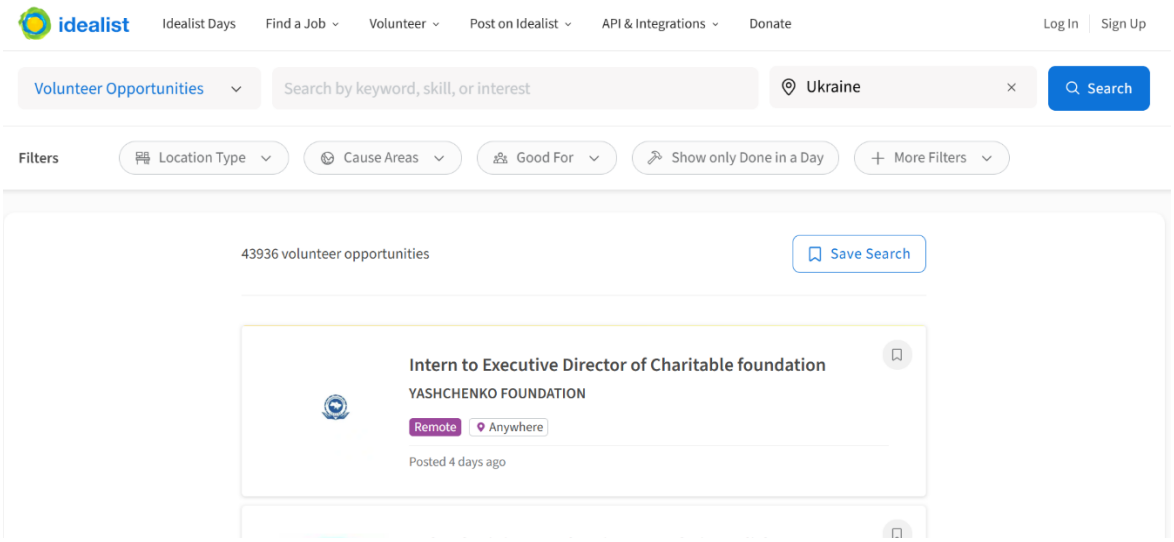

Рисунок 1.2 – Інтерфейс платформи VolunteerMatch

Більш сучасним міжнародним аналогом є платформа Golden Volunteer, зображена на рисунку 1.3.

Рисунок 1.3 – Платформа Golden Volunteer

Вона фокусується на покращеному UX/UI та гейміфікації процесу. Хоча платформа пропонує зручніший інтерфейс для волонтерів, її функціонал для організацій все ще зосереджений переважно на публікації можливостей та базовому відстеженні годин, не надаючи комплексного середовища для управління довгостроковими проєктами та командою всередині єдиної екосистеми. Інтерфейс платформи бачимо на рисунку 1.4.

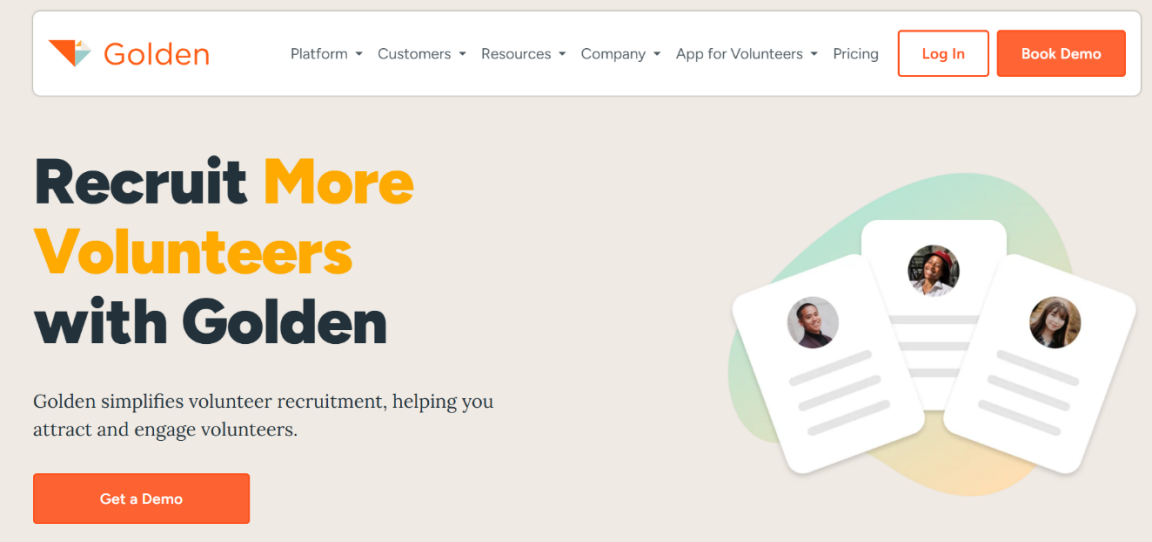

Рисунок 1.4 – Інтерфейс Golden Volunteer

На українському ринку яскравим прикладом є Платформа Української Волонтерської Служби, що зображена на рисунку 1.5.

Рисунок 1.5 - Платформа Української Волонтерської Служби

Її сильною стороною є глибока адаптація до локального контексту та оперативність реагування на кризові потреби. Проте критичний аналіз архітектури показує високий рівень фрагментації процесів. Веб-ресурс часто виконує роль вітрини, що зображено на рисунку 1.6, тоді як критичні бізнес-процеси (реєстрація волонтерів, збір потреб, координація) реалізуються через зовнішні, неінтегровані інструменти, такі як Google Forms або Telegram-чати. Це призводить до децентралізації даних, ускладнює створення єдиного профілю волонтера та не дає організаціям повноцінного інструментарію для менеджменту ресурсів.

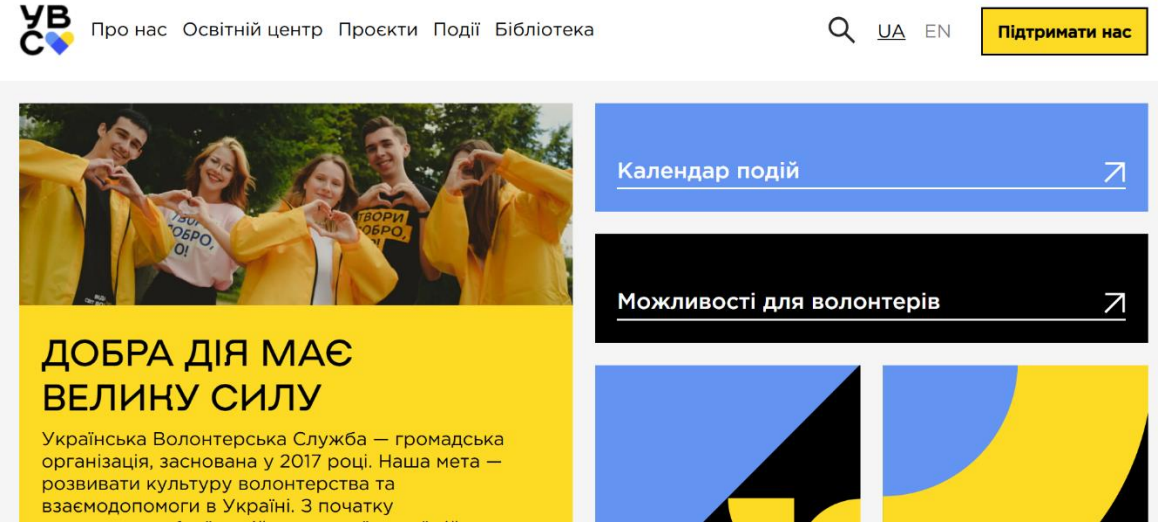

Рисунок 1.6 – Інтерфейс УВС

Розроблена у рамках магістерської роботи "Платформа управління волонтерством" позиціонується як комплексне рішення. На відміну від розглянутих аналогів, система базується на сучасній гібридній архітектурі та надає виділені спеціалізовані портали для всіх трьох ролей. Вона забезпечує не лише пошук можливостей, але й повний цикл менеджменту проєктів, подій та бази волонтерів для організацій, а також централізовану модерацію для адміністраторів у межах єдиної системи. Порівняльний аналіз ключових характеристик розглянутих платформ наведено у таблиці 1.6.

Таблиця 1.6 – Порівняння платформ-аналогів

Критерій порівняння	VolunteerMatch (США)	Golden Volunteer (США)	Платформа УВС (Україна)	Розроблена платформа
Основна модель роботи	Агрегатор вакансій (Legacy)	Сучасний агрегатор з гейміфікацією	Агрегатор потреб + зовнішні інструменти	Інтегроване середовище управління процесами
Повнота рольової моделі (RBAC)	Часткова (Акцент на пошуку для волонтера)	Середня (Покращений кабінет волонтера)	Слабка (Взаємодія часто через зовнішні канали)	Повна (Виділені портали: Волонтер, Організація, Менеджер)

Продовження таблиці 1.6

Інструменти для організацій	Базові (Публікація, отримання заявок на email)	Середні (Публікація, трекінг годин)	Мінімальні вбудовані (Покладання на зовнішні форми)	Розширені (Управління проєктами, подіями, базою волонтерів в системі)
Централізація даних	Висока, але застаріла структура	Висока	Низька (Фрагментація між сервісами)	Висока (Єдине джерело істини для всіх процесів)
Архітектура та UX/UI	Застаріла монолітна, слабкий мобільний UX	Сучасна SPA, якісний UX	Різна (часто прості статичні сторінки)	Сучасна гібридна (SSR+SPA), висока продуктивність, адаптивний UI

Результати порівняльного аналізу, наведені в таблиці 1.6, наочно демонструють, що розроблена платформа пропонує якісно новий підхід до автоматизації волонтерської діяльності. Відмова від моделі простого агрегатора вакансій на користь створення інтегрованого середовища управління дозволяє подолати ключові недоліки існуючих рішень — фрагментацію даних та обмеженість інструментарію для організацій. Поєднання повної рольової моделі із сучасною гібридною архітектурою забезпечує системі конкурентні переваги як у функціональній площині, так і в аспектах продуктивності та користувацького досвіду.

Висновки до розділу 1

У першому розділі було проведено комплексний аналіз предметної області координації волонтерської діяльності. Виявлено ключові проблеми існуючих процесів, зокрема фрагментацію даних та низьку ефективність комунікації між учасниками. Критичний огляд сучасних вітчизняних та закордонних веб-платформ показав, що більшість аналогів мають суттєві архітектурні обмеження, застарілий стек технологій, недостатній рівень реалізації рольової моделі доступу та не відповідають сучасним вимогам щодо мобільного користувацького досвіду.

На основі проведеного аналізу було сформульовано чітку постановку задачі, визначено функціональні та нефункціональні вимоги до нової системи. Обґрунтовано вибір сучасного технологічного стеку, що включає фреймворк Nuxt.js (Vue 3), мову TypeScript, бібліотеки Pinia, PrimeVue та Tailwind CSS. Доведено, що цей комплекс інструментів є оптимальним для створення високопродуктивної, масштабованої та безпечної веб-платформи з гібридною архітектурою рендерингу.

2 МОДЕЛІ ТА МЕТОДИ ПРЕДСТАВЛЕННЯ ДАННИХ ДОСТУПУ ТА УПРАВЛІННЯ ЗАСТОСУНКУ

Проєктування та реалізація сучасної веб-платформи для організації волонтерської діяльності вимагає переходу від сформульованих вимог до побудови чітких архітектурних та алгоритмічних моделей. Основою інженерного рішення є концептуальна модель даних, що відображає структуру предметної області, та деталізована архітектура безпеки, яка визначає правила взаємодії користувачів з системою. Важливим етапом розробки є вибір та обґрунтування конкретних методів реалізації клієнт-серверної взаємодії, забезпечення реактивності інтерфейсу та управління потоками даних у розподіленому середовищі, що дозволить створити масштабовану та ефективну систему.[15]

2.1 Проєктування концептуальної моделі даних та архітектури доступу

Першим кроком у розробці складної програмної системи є перетворення бізнес-вимог, сформульованих у першому розділі, на чітку структурну модель. Ця модель визначає основні інформаційні об'єкти системи, їхні характеристики та правила взаємодії між ними. Паралельно з цим проєктується архітектура безпеки, яка визначає, хто і як може взаємодіяти з цими даними.

2.1.1 Концептуальна модель сутностей предметної області

На основі аналізу вимог до платформи було виділено ключові сутності, які формують інформаційний каркас системи. Модель побудована з урахуванням необхідності розділення користувачів на різні типи та забезпечення зв'язку між волонтерами та заходами.

Основні сутності системи:

1. Користувач (User) - базова абстрактна сутність, що містить спільні атрибути для всіх учасників платформи: унікальний ідентифікатор, електронну пошту (для автентифікації), хеш пароля та дату реєстрації. Ця сутність є основою для конкретних ролей.

2. Волонтер (Volunteer) - сутність, що розширює базового користувача. Вона містить специфічні дані, необхідні для участі у волонтерській діяльності: особисту інформацію (ім'я, прізвище), контактні дані та, що критично важливо для функціоналу платформи, — набір навичок (Skills). Навички є ключовим атрибутом для подальшого алгоритму підбору відповідних завдань (матчингу).

3. Організація (Organization) - сутність, що представляє координаторів волонтерської діяльності. Вона містить атрибути профілю організації: назву, опис діяльності, контактну інформацію та статус верифікації адміністратором платформи.

4. Проєкт (Project) та подія (Event) - сутності, що представляють об'єкти волонтерської діяльності. "Проєкт" — це довгострокова ініціатива, яка може об'єднувати декілька конкретних "Подій" (заходів з певною датою та місцем проведення). Обидві сутності містять опис, вимоги до волонтерів та статус (наприклад, "Активний", "Завершений").[15,16]

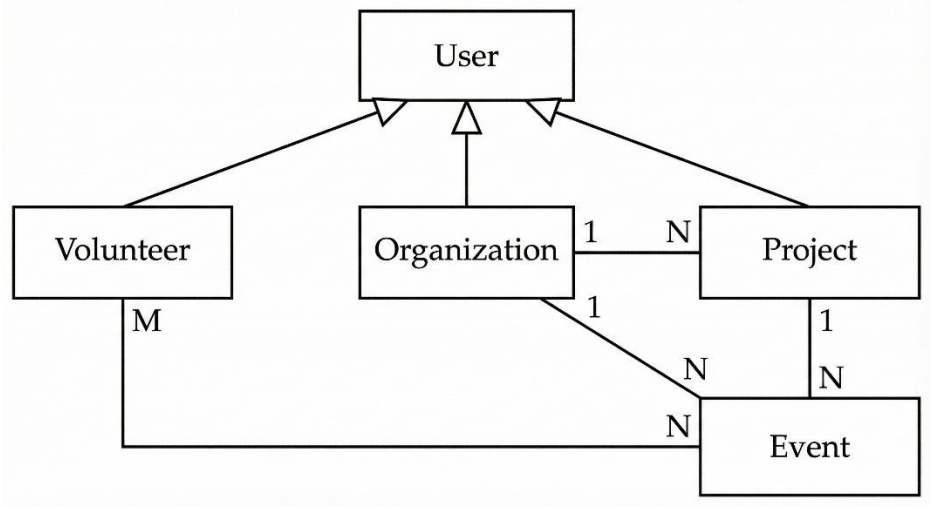

Рисунок 2.1 - Схема зв'язків волонтерської платформи

Між цими сутностями встановлено два ключових зв'язки. Зв'язок один-до-багатьох - одна організація може створити багато проєктів та подій. Цей зв'язок необхідний для розмежування доступу. Організація має право редагувати лише власні заходи.[16]

Зв'язок багато-до-багатьох - один волонтер може брати участь у багатьох подіях, і в одній події можуть брати участь багато волонтерів. Цей зв'язок є центральним для основного функціоналу платформи — координації допомоги.

Ці зв’язки наглядно зображені на рисунку 2.1.

2.1.2 Фізична модель бази даних

Наступним етапом після побудови концептуальної моделі є її трансформація у фізичну схему бази даних. Цей процес передбачає вибір конкретної системи управління базами даних (СУБД), визначення типів даних для атрибутів, проєктування таблиць та встановлення зв'язків між ними з урахуванням вимог до продуктивності та цілісності даних.

Для реалізації сховища даних платформи обрано об'єктно-реляційну систему управління базами даних PostgreSQL. Цей вибір обумовлений наступними чотирма факторами. Надійність та відповідність ACID - PostgreSQL гарантує транзакційну цілісність даних, що є критично важливим для операцій реєстрації користувачів та запису на події. Підтримка складних типів даних - наявність нативних типів для роботи з JSONB дозволяє гнучко зберігати слабкоструктуровані дані (наприклад, динамічний набір навичок волонтера або налаштування профілю) без порушення нормалізації основної схеми. Масштабованість та продуктивність - розвинені механізми індексації та оптимізації запитів дозволяють системі ефективно працювати під високим навантаженням.[17] Інтеграція з екосистемою Node.js - наявність потужних ORM-бібліотек (таких як Prisma або TypeORM) забезпечує зручну та типобезпечну взаємодію з базою даних на рівні коду.

Фізична модель спроєктована відповідно до третьої нормальної форми для уникнення надлишковості даних. Ключові рішення при проєктуванні схеми:

1. Ідентифікатори - у якості первинних ключів (Primary Keys) для всіх таблиць використовуються UUID (Universally Unique Identifier) замість традиційних автоінкрементних цілих чисел. Це підвищує безпеку (ідентифікатори важче підібрати) та спрощує потенційне масштабування або міграцію даних у розподіленому середовищі.

2. Зберігання часу - для всіх полів, що стосуються дати та часу (наприклад, createdAt, eventDate), використовується типу timestamp with time zone, що забезпечує коректну роботу з користувачами в різних часових поясах.

3. Реалізація зв'язків:

- зв'язки "один-до-багатьох" (1:N), наприклад, між Організацією та Подіями, реалізовані через зовнішні ключі (Foreign Keys) у залежній таблиці (наприклад, поле organizerId у таблиці events).

- зв'язок "багато-до-багатьох" (M:N) між Волонтерами та Подіями реалізовано через проміжну таблицю (junction table) registrations, яка містить зовнішні ключі на обидві сутності та додаткові атрибути зв'язку (наприклад, статус заявки, час реєстрації).[18]

Для забезпечення високої швидкості виконання запитів, особливо на сторінках каталогу подій та при автентифікації, розроблено стратегію індексації. Окрім автоматичних індексів на первинних ключах, створено додаткові B-tree індекси для:

1. Полів пошуку та фільтрації - status, city та date у таблиці events проіндексовані для швидкого відбору актуальних подій у каталозі.

2. Зовнішніх ключів - поля, що використовуються для об'єднання таблиць (JOIN), такі як organizerId та userId, проіндексовані для прискорення вибірок пов'язаних даних.

3. Унікальних полів - поле email у таблиці users має унікальний індекс для забезпечення швидкої автентифікації та запобігання дублюванню акаунтів.

2.1.3 Архітектура рольової моделі доступу

Для забезпечення безпеки та виконання вимоги щодо надання виділених порталів для різних груп користувачів, у системі спроєктовано архітектуру управління доступом на основі ролей.[18]

Ця модель вводить прошарок абстракції між користувачем і його правами, що значно спрощує управління безпекою та масштабування системи. Архітектура складається з трьох основних компонентів:

1. Суб'єкти (користувачі) - конкретні облікові записи людей, які пройшли автентифікацію.
2. Ролі - абстрактні контейнери, що відповідають бізнес-функціям у системі. У платформі визначено три основні ролі: волонтер, менеджер організації, менеджер платформи.
3. Дозволи - атомарні права на виконання конкретних дій у системі (наприклад, event:create — право створювати подію, user:ban — право блокувати користувача).

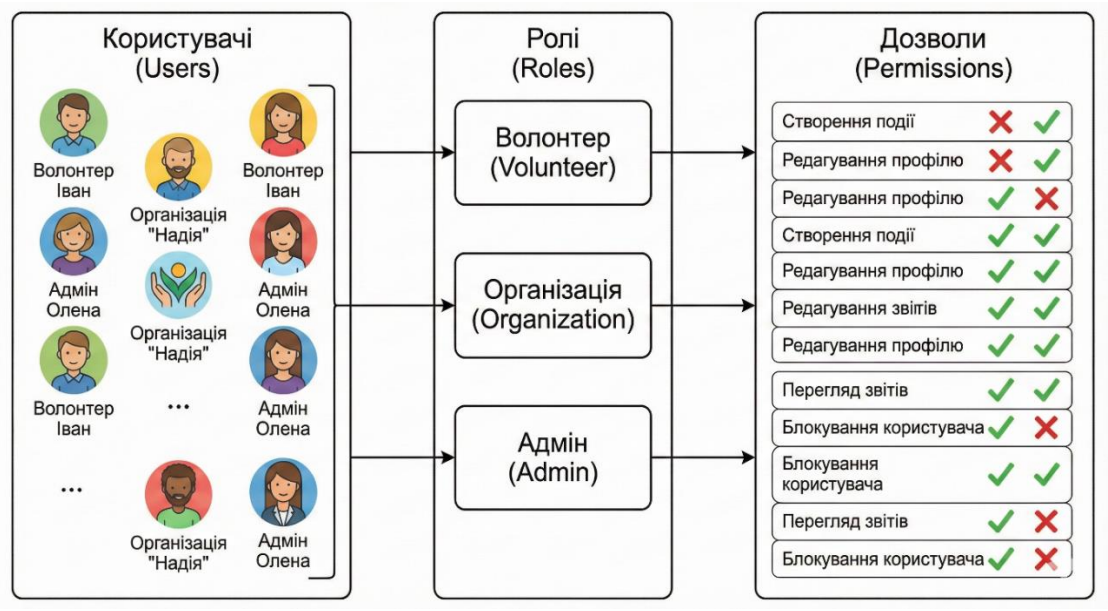

Рисунок 2.2 - Структурна схема рольової моделі контролю доступу

Замість того, щоб призначати дозволи кожному користувачу індивідуально (що неможливо адмініструвати у великій системі), дозволи групуються і

призначаються ролі. Користувачу ж, у свою чергу, призначається одна з ролей при реєстрації або адміністратором. Структуру такого розмежування доступу зображена на рисунку 2.2.

Що до алгоритму перевірки доступу або авторизації, коли користувач намагається виконати захищену дію наприклад, створити подію, система виконує такий алгоритм перевірки:

1. Ідентифікує поточного користувача.

2. Визначає його призначену роль.

3. Перевіряє, чи містить ця роль необхідний дозвіл для запитуваної дії.

4. Якщо дозвіл наявний, дія дозволяється; в іншому випадку — блокується з помилкою доступу.

Як видно з лістингу А1 у додатку А, алгоритм працює наступним чином: при спробі доступу до захищеного маршруту система ідентифікує поточний стан автентифікації та роль користувача. Якщо роль не відповідає вимогам маршруту, виконання запиту переривається, і користувач перенаправляється на відповідну сторінку. Ця програмна реалізація є фундаментом системи безпеки веб-додатку.

Ця концептуальна модель є фундаментом для подальшої програмної реалізації механізмів захисту маршрутів. [18]

2.2 Реалізація гібридної архітектури та компонентів інтерфейсу

Визначення методів побудови клієнтської частини, що забезпечує взаємодію з користувачем, є наступним етапом після проєктування моделей даних і безпеки. Вибір технологічної основи платформи було здійснено з урахуванням вимог до високої продуктивності та SEO-оптимізації. В результаті було обрано гібридну архітектуру на базі Nuxt.js, яка завдяки механізму універсального рендерингу дозволяє комбінувати переваги серверного рендерингу та інтерактивність односторінкових застосунків.

Основою функціонування публічної частини платформи є алгоритм універсального рендерингу, реалізований засобами фреймворку Nuxt.js. Суть цього

підходу полягає в тому, що кодова база інтерфейсу, написана на Vue.js, є спільною для клієнтського та серверного середовищ, проте етапи її виконання різняться залежно від контексту запиту.[19]

Процес обробки першого заходу користувача на сторінку є багатокроковим алгоритмом, що включає виконання коду в середовищі Node.js на сервері та подальшу "активацію" сторінки в браузері. Детальний алгоритм складається з наступних етапів:

1. Отримання запиту та серверна ініціалізація.

При надходженні HTTP-запиту до сервер Nuxt, що працює на базі Node.js, перехоплює цей запит. Для кожного окремого запиту створюється ізольований контекст додатку та ініціалізується новий екземпляр кореневого Vue-компонента.

2. Асинхронне отримання даних.

Перед початком рендерингу HTML, фреймворк визначає, які компоненти будуть відображені на запитуваному маршруті. Він виконує спеціальні асинхронні функції, визначені в цих, що представлено у лістингу А2 у додатку А. На цьому етапі серверна частина платформи виконує запити до внутрішнього або зовнішнього. Важливою особливістю є те, що ці запити виконуються безпосередньо з сервера, що зменшує затримки мережі та навантаження на клієнта.

3. Серверний рендеринг (SSR) та серіалізація стану.

Отримані асинхронні дані зберігаються у реактивному стані додатку. Після завершення всіх запитів, Nuxt використовує бібліотеку vue/server-renderer для перетворення ієрархії Vue-компонентів у статичний HTML-рядок. Критично важливим кроком на цьому етапі є серіалізація стану: всі дані, отримані під час кроку 2, перетворюються у JSON-формат і вбудовуються у згенерований HTML-документ. Це необхідно для того, щоб клієнтська частина отримала ті самі дані, що використовувалися на сервері, без повторних запитів.

4. Клієнтська гідратація.

Після завантаження HTML браузер починає виконувати клієнтський JavaScript-код. Фреймворк Vue "підхоплює" існуюче DOM-дерево та десеріалізує стан, отриманий із сервера. Процес гідратації полягає у співставленні віртуального DOM, згенерованого на клієнті на основі отриманих даних, з реальним DOM,

отриманим від сервера. Якщо структури співпадають, Vue не перемальовує сторінку, а лише "прикріплює" обробники подій до існуючих елементів. Це перетворює статичну сторінку на повністю інтерактивний односторінковий застосунок.

У разі виникнення розбіжностей між серверним та клієнтським рендерингом Vue змушений відкинути серверний HTML і виконати повне перемальовування на клієнті, що негативно впливає на продуктивність. Тому дотримання ізоморфності коду є важливою вимогою при розробці компонентів.

Після успішної гідратації додаток переходить у режим SPA. Подальша навігація між маршрутами не призводить до перезавантаження сторінки; натомість, Nuxt перехоплює кліки по посиланнях, асинхронно завантажує код та дані для нової сторінки і динамічно оновлює інтерфейс, забезпечуючи плавність роботи.

Реалізація базується на концепції однофайлових компонентів екосистеми Vue. Кожен компонент, наприклад, картка події, форма реєстрації, навігаційна панель, інкапсульований в одному файлі з розширенням .vue, який структурно складається з трьох частин:

- <template> - декларативний опис HTML-структури компонента.

- <script setup lang="ts"> - логіка компонента, написана з використанням TypeScript та Composition API. Тут визначається реактивний стан, методи та обчислювані властивості.

- <style scoped> - стилі CSS, область видимості яких обмежена виключно даним компонентом, що запобігає конфліктам стилів у масштабах всього додатку.

Критично важливим для реалізації бізнес-логіки є розуміння та використання життєвого циклу компонента. Це набір етапів, через які проходить компонент від моменту створення до знищення. У розробленій платформі активно використовуються хуки життєвого циклу Composition API для керування поведінкою інтерфейсу:

onMounted() - вкористовується для виконання дій, що потребують доступу до реального DOM-дерева (наприклад, ініціалізація інтерактивних карт або складних графіків) після того, як компонент було вбудовано в сторінку.

onUpdated() - дозволяє реагувати на реактивні зміни даних після того, як DOM було оновлено.

onUnmounted() - застосовується для очищення ресурсів (наприклад, видалення глобальних слухачів подій або зупинки таймерів) перед видаленням компонента, що запобігає витокам пам'яті.

Застосування цієї архітектури дозволило створити модульну, легко підтримувану кодову базу, де кожен елемент інтерфейсу має чітко визначену зону відповідальності та передбачувану поведінку.

2.3 Проєктування архітектури взаємодії з програмним інтерфейсом сервера

У розподілених веб-системах, побудованих за клієнт-серверною архітектурою, критично важливим аспектом є організація ефективної та надійної взаємодії між фронтенд-додатком та сервером. Пряме виконання HTTP-запитів (наприклад, використовуючи fetch або axios) безпосередньо всередині UI-компонентів призводить до дублювання коду, порушення принципу єдиної відповідальності (Single Responsibility Principle) та ускладнює підтримку системи при зміні контрактів API.

Для вирішення цієї проблеми у розробленій платформі спроєктовано виділений архітектурний шар (abstraction layer) для взаємодії з API, реалізований з використанням патерну "Сервісний шар" (Service Layer), адаптованого під екосистему Vue 3 Composition API.[19]

Архітектура взаємодії базується на інкапсуляції логіки мережевих запитів у спеціалізовані модулі — композиабли (Composables). Кожен композиаб-сервіс відповідає за роботу з конкретною доменною сутністю.

Цей підхід надає три інженерні переваги. Централізація логіки - усі ендпоінти API, специфічні заголовки запитів та базові URL-адреси визначаються в одному місці. Уніфікована обробка помилок - сервісний шар дозволяє реалізувати єдиний механізм перехоплення мережевих помилок та їх стандартизоване логування або

відображення користувачу, перш ніж дані потраплять до Pinia або компонентів. Абстракція від HTTP-клієнта - компоненти та сховища Pinia не залежать від конкретної реалізації HTTP-клієнта, що дозволяє легко замінити його в майбутньому за необхідності.

Критичним аспектом спроєктованої архітектури є використання можливостей TypeScript для забезпечення гарантій типів даних, що передаються мережею. Оскільки серверна та клієнтська частини платформи розроблені в єдиному монорепозиторії з використанням TypeScript, вони спільно використовують набір інтерфейсів даних.[20]

Сервісні композиабли використовують механізм узагальнень (generics) TypeScript при виконанні запитів. Це дозволяє явно вказати очікувану структуру відповіді від сервера. Якщо сервер API (реалізований на Nitro) повертає дані, що не відповідають очікуваному інтерфейсу (наприклад, відсутнє обов'язкове поле), система типізації повідомить про невідповідність ще на етапі компіляції або розробки.

Приклад реалізації сервісного композиаблу для роботи з подіями наведено у лістингу А3 у додатку А.

2.4 Методи управління централізованими даними та станом застосунку

У сучасних односторінкових веб-застосунках (SPA) поняття "стану" (state) охоплює сукупність усіх даних, які визначають поведінку та відображення інтерфейсу в конкретний момент часу. Це можуть бути дані, отримані з сервера (список подій, профіль користувача), або локальні дані інтерфейсу (стан відкриття модального вікна, значення фільтрів).[20]

Із зростанням складності системи та збільшенням кількості компонентів виникає критична архітектурна проблема синхронізації стану. Традиційний підхід передачі даних від батьківських компонентів до дочірніх через механізм властивостей (props) призводить до проблеми "прокидання пропсів" (prop drilling),

коли дані змушені проходити через безліч проміжних рівнів ієрархії, які їх не використовують. Це порушує принцип інкапсуляції, створює жорстку зв'язність між компонентами та значно ускладнює підтримку коду.

Для вирішення цієї проблеми у розробленій платформі застосовано патерн централізованого управління станом (Centralized State Management), реалізований за допомогою бібліотеки Pinia. Цей підхід базується на архітектурному патерні Flux, який пропонує винести спільний стан застосунку за межі дерева компонентів у глобальне сховище, забезпечуючи передбачуваний односпрямований потік даних.

2.4.1 Архітектура централізованого сховища

Архітектура управління даними в Pinia базується на концепції Single Source of Truth. Сховище є автономною сутністю, яка інкапсулює дані та логіку їх обробки. Структурно воно складається з трьох фундаментальних елементів:

1. State (стан) - реактивний об'єкт, що є безпосереднім носієм даних. Це "база даних" на боці клієнта.

2. Getters (геттери) - чисті функції-обчислювачі, які дозволяють отримувати похідні дані зі стану. Вони автоматично кешують результати своїх обчислень і перераховуються лише тоді, коли змінюються реактивні залежності, на яких вони базуються.

3. Actions (дії) - методи, що містять бізнес-логіку для модифікації стану. На відміну від мутацій у попередніх поколіннях менеджерів стану дії в Pinia можуть бути асинхронними, що робить їх ідеальним шаром для інкапсуляції взаємодії із зовнішніми API.[21]

Ключовою технологією, що забезпечує ефективність цього підходу, є система реактивності Vue 3. Стан у Pinia обгортається у реактивні проксі-об'єкти. Коли компонент звертається до властивості стану під час свого рендерингу, він "підписується" на цю властивість. Коли Дія модифікує цю властивість стану, система реактивності автоматично сповіщає всіх підписників, ініціюючи

ефективне, точкове оновлення відповідних частин DOM-дерева без необхідності повного перемальовування сторінки.

Взаємодія між компонентами та сховищем реалізує односпрямований потік даних (One-Way Data Flow), схема якого зображена на рисунку 2.5.

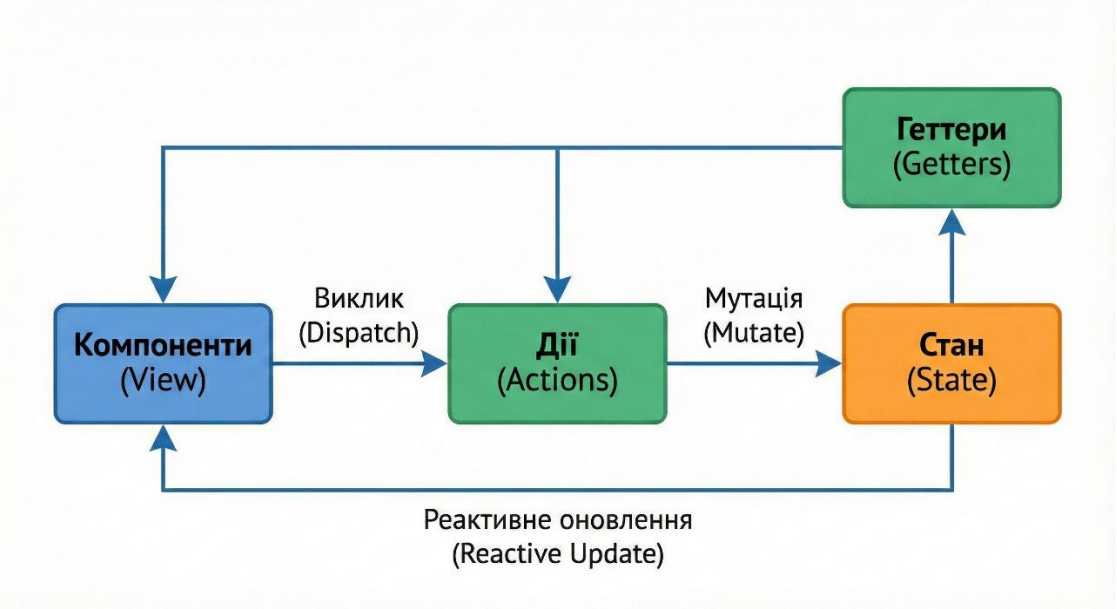

Рисунок 2.3 - Схема односпрямованого потоку даних

Як видно з рисунку 2.3, компоненти ніколи не змінюють стан напряму. Натомість вони викликають дії, які виконують необхідну логіку і мутують стан. Змінений стан реактивно оновлює компоненти. Така циклічна, але строго односпрямована структура робить поведінку системи передбачуваною та значно спрощує відлагодження та тестування бізнес-логіки.

2.4.2 Організація потоку даних між компонентами та сховищем

Практична реалізація цього патерну в платформі продемонстрована на прикладі модуля сховища для управління подіями (events store). Визначення цього модуля з використанням синтаксису "Setup Store" наведено на нижче у фрагменті коду у лістингу А4 у додатку А.

Механізм взаємодії базується на системі гранулярної реактивності Vue 3. Коли компонент звертається до певної властивості стану сховища у своєму шаблоні або обчислюваних властивостях, система автоматично реєструє залежність саме від цієї конкретної властивості, а не від усього об'єкта стану. Це гарантує високу ефективність оновлення інтерфейсу: модифікація однієї частини глобального стану спричиняє перемальовування виключно тих компонентів, які безпосередньо використовують цю частину даних, не зачіпаючи решту дерева компонентів.[21]

Важливим інженерним аспектом реалізації є збереження реактивного зв'язку при доступі до даних. Поширена практика деструктуризації об'єктів у JavaScript при наївному застосуванні до сховища Pinia призводить до втрати реактивності, оскільки витягуються примітивні значення, а не реактивні проксі. Для вирішення цієї проблеми застосовується спеціалізована утиліта storeToRefs, яка дозволяє безпечно деструктуризувати стан та геттери, зберігаючи їх реактивну природу. При цьому методи дій деструктуризуються напряму, оскільки вони є функціями і не потребують реактивної обгортки.

Як показано в лістингу, такий підхід забезпечує чітке розділення відповідальності: компонент делегує всю логіку роботи з даними сховищу і лише реактивно відображає поточний стан, ініціюючи необхідні дії. Це робить код компонентів чистим, декларативним та легким для тестування.

Висновки до розділу 2

У другому розділі розроблено теоретико-інженерний фундамент системи. Спроєктовано концептуальну та фізичну моделі бази даних, що забезпечують надійне зберігання та цілісність інформації про користувачів, події та організації. Формалізовано архітектуру Рольової Моделі Доступу (RBAC), яка стала основою системи безпеки платформи.

Детально розроблено архітектуру клієнтської частини на базі Nuxt.js, обґрунтувано застосування гібридного рендерингу для балансу між SEO-оптимізацією та інтерактивністю. Спроєктовано компонентну модель інтерфейсу

та архітектуру централізованого управління станом за допомогою Pinia. Розроблено шар взаємодії з сервером (API service layer) із забезпеченням наскрізної типізації, а також алгоритми безстанової автентифікації на основі JWT-токенів та механізми авторизації запитів через middleware. Розроблені моделі та алгоритми створюють цілісну основу для подальшої програмної реалізації.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА СУПРОВОДЖЕННЯ ВЕБ-ПЛАТФОРМИ

Практична програмна реалізація веб-платформи для організації волонтерської діяльності виконана на основі теоретичних моделей та архітектурних рішень, розроблених у ході проєктування. В основі програмної системи лежить використання фреймворку Nuxt.js, що визначило загальну структуру застосунку та підхід до побудови компонентної моделі інтерфейсу користувача. Об'єктно-орієнтована структура проєкту представлена через систему класів та інтерфейсів TypeScript, що забезпечує надійність та розширюваність коду. Невід'ємною складовою реалізації є розробка технічної документації, що регламентує процеси розгортання та подальшого супроводження програмного забезпечення.

3.1 Загальна архітектура програмної системи

Розроблена веб-платформа є сучасним повним (full-stack) веб-застосунком, реалізованим на базі мета-фреймворку Nuxt 3. Архітектура системи дотримується принципів модульності та розділення відповідальності, що забезпечується стандартизованою структурою директорій Nuxt.

Система складається з двох основних логічних частин, що працюють у єдиному монорепозиторії. Клієнтська частина (Frontend) відповідає за рендеринг інтерфейсу, взаємодію з користувачем та управління станом у браузері. Реалізована на Vue.js 3. Серверна частина (Backend / Nitro Server) відповідає за серверний рендеринг (SSR), обробку API-запитів та взаємодію із зовнішніми сервісами або базою даних. Реалізована на базі рушія Nitro, що вбудований у Nuxt.[22]

Високорівнева архітектура програмної реалізації та взаємодія ключових модулів зображена на рисунку 3.1.

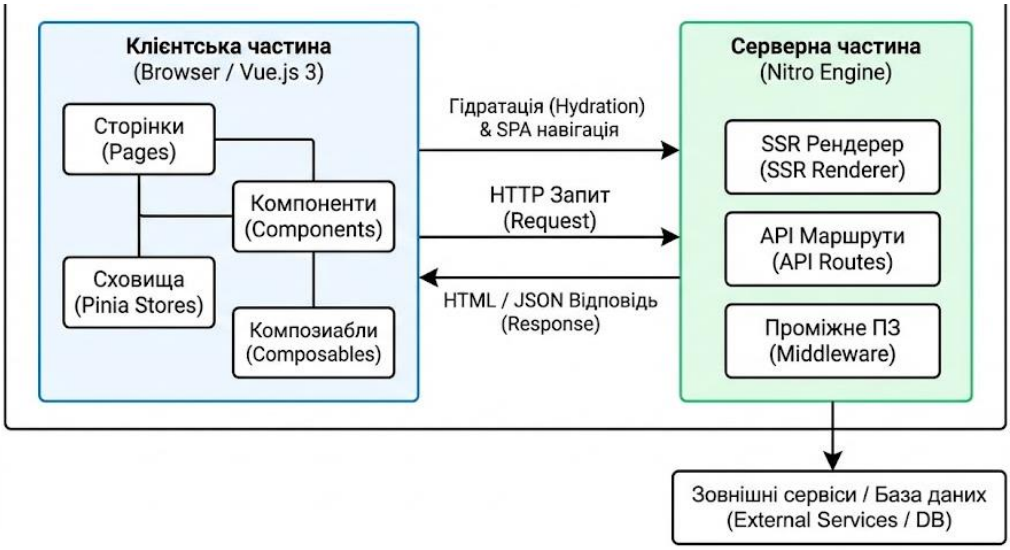

Рисунок 3.1 – Загальна архітектура програмної реалізації веб-платформи на Nuxt

Як видно з рисунку 3.1, ядром системи є серверний рушій Nitro, який приймає вхідні HTTP-запити. Залежно від типу запиту, він або ініціює процес серверного рендерингу Vue-застосунку для сторінок, або обробляє запит як API-ендпоінт для даних.

Ключові архітектурні компоненти реалізації включають:

1. Pages (сторінки) - компоненти в директорії pages/, що формують структуру маршрутизації застосунку. У розробленій платформі вони чітко розділені за рольовим принципом: публічні сторінки pages/index.vue, pages/events/index.vue, захищений кабінет волонтера pages/volunteers/ та панель управління організації pages/organizations/.

2. Components (компоненти інтерфейсу) - багаторазові елементи UI в директорії components/. Вони побудовані з використанням бібліотеки PrimeVue та стилізовані за допомогою утилітарних класів Tailwind CSS. Прикладами є EventCard.vue - картка події, що використовується у списках та специфічні форми реєстрації.

3. Stores (сховища стану) - модулі Pinia в директорії stores/ для централізованого управління даними на клієнті. Реалізовано модулі auth.ts для зберігання стану поточного користувача та токена та events.ts для управління списками подій та фільтрами.

4. Composables (складові) - в директорії composables/ інкапсульована логіка, що використовується повторно. Наприклад, useAPI.ts для уніфікації запитів до серверної частини.

5. Middleware (проміжне ПЗ) - перехоплювачі навігації в middleware/ для реалізації захисту маршрутів. Зокрема, auth.global.ts реалізує перевірку JWT-токена та рольову модель доступу (RBAC) перед входом на захищені сторінки.[23]

Така архітектура забезпечує високу гнучкість розгортання. Систему можна розгорнути як на традиційному Node.js сервері, так і в безсерверному (serverless) середовищі. Для демонстраційної версії платформи було обрано хмарну платформу Vercel, що забезпечує автоматичне масштабування серверної частини Nitro.

Для наочної демонстрації компонентного підходу на рисунку 3.2 наведено схему ієрархії компонентів на прикладі сторінки каталогу подій.

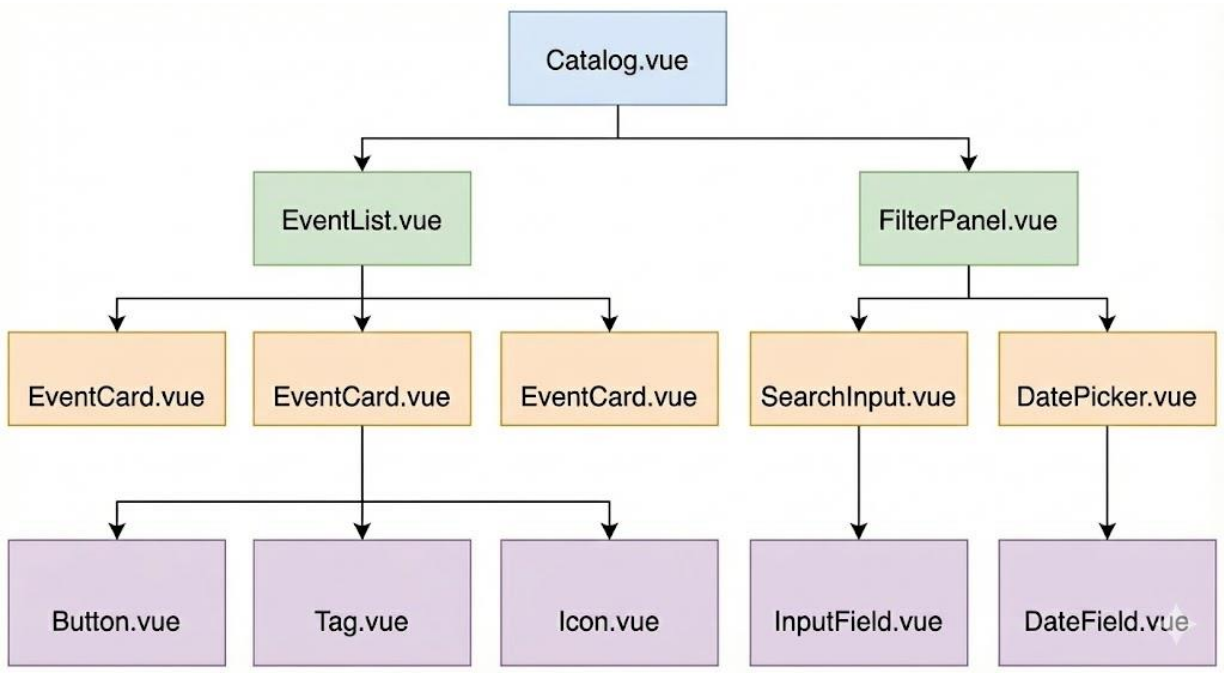

Рисунок 3.2 - Ієрархія компонентів інтерфейсу

Як видно з рисунку 3.2, застосування компонентного підходу дозволяє будувати складні інтерфейси з простих, ізольованих та повторно використовуваних блоків, що значно спрощує підтримку та тестування кодової бази.

3.2 Об’єктно-орієнтоване проєктування та структура даних системи

Для забезпечення надійності програмного коду, зменшення кількості помилок під час виконання та покращення підтримки проєкту, реалізацію платформи виконано з використанням мови програмування TypeScript. Це дозволило формалізувати концептуальну модель даних, розроблену через систему суворих типів та інтерфейсів.

Об’єктно-орієнтована структура основних сутностей системи та зв’язки між ними представлені на діаграмі класів на рисунку 3.3.

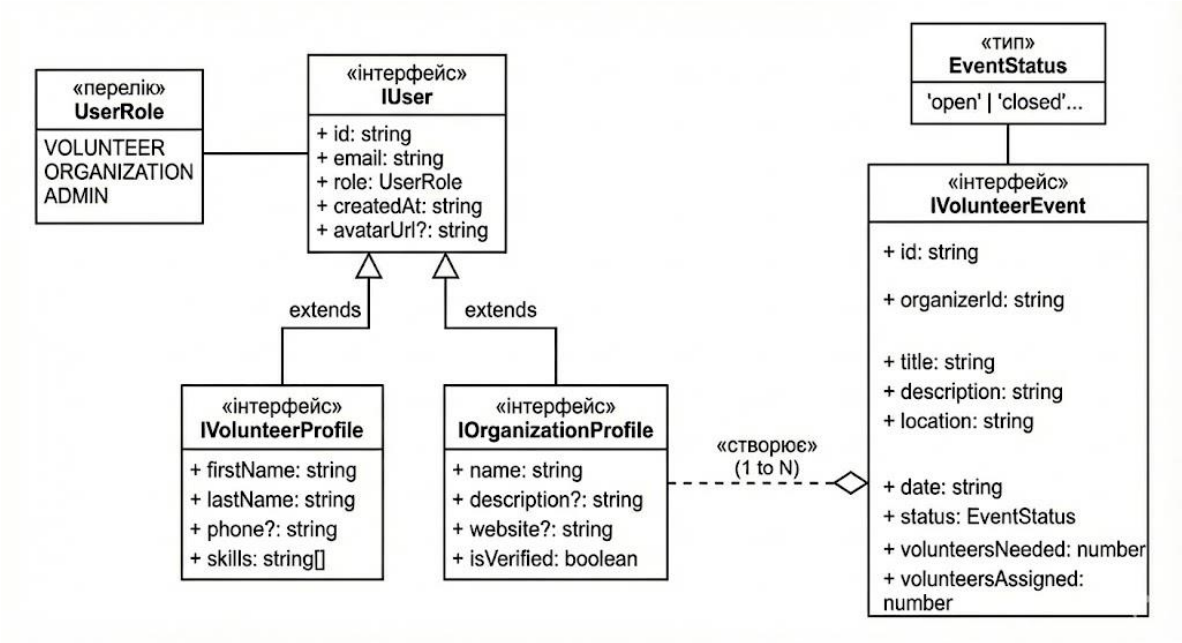

Рисунок 3.3 – Діаграма основних інтерфейсів та типів даних

Як показано на рисунку 3.3, в основі ієрархії користувачів лежить базовий інтерфейс User, який визначає спільні атрибути для всіх учасників системи. Ключовим полем тут є role, яке використовується системою безпеки (RBAC). Програмна реалізація цього інтерфейсу та переліку ролей наведена у лістингу А5 у додатку А.

Для специфічних ролей створено інтерфейси-нащадки, які розширюють базовий User. Наприклад, інтерфейс волонтера (VolunteerProfile) додає поля для особистих даних та масиву навичок, необхідних для подій.[23,24]

Центральною сутністю предметної області є "подія". Її структура даних спроєктована так, щоб містити всю необхідну інформацію для відображення в каталозі та управління з боку організації. Реалізація інтерфейсу події наведена у лістингу А6 у додатку А.

Застосування розробленої системи інтерфейсів на всіх рівнях програмного коду забезпечує сувору перевірку типів даних ще на етапі написання програми. Це дозволяє автоматично виявляти та попереджати потенційні помилки, пов'язані з неправильною структурою даних, до моменту запуску системи, що суттєво підвищує загальну надійність та стабільність роботи платформи.

3.3 Реалізація компонентної моделі та взаємодія модулів інтерфейсу

Практична реалізація користувацького інтерфейсу платформи базується на ієрархічній компонентній моделі Vue.js. Відповідно до архітектури, файлова структура проєкту чітко розділяє компоненти-сторінки та багаторазові UI-компоненти.

Сторінки виступають у ролі компонентів-контейнерів вищого рівня. Вони відповідають за отримання даних (наприклад, зі сховища Pinia або через API-запити) та їх розподіл між вкладеними презентаційними компонентами. Презентаційні компоненти, своєю чергою, фокусуються виключно на відображенні даних та перехопленні дій користувача.[24]

Складна сторінка каталогу декомпозована на менші, логічно завершені модулі. Наприклад, організм EventList.vue відповідає за відображення сітки подій, використовуючи для кожної окремої події молекулу EventCard.vue. Остання, у свою чергу, складається з базових атомів інтерфейсу, таких як кнопки (Button.vue) та теги статусів (Tag.vue).

Взаємодія між рівнями цієї ієрархії реалізована через стандартні механізми Vue:

1. Передача даних вниз - батьківські компоненти передають дані дочірнім через вхідні параметри. Наприклад, EventList передає об'єкт конкретної події у компонент EventCard для відображення.

2. Передача подій вгору - дочірні компоненти повідомляють батьківські про дії користувача через генерацію подій. Наприклад, при натисканні кнопки "Подати заявку" в атомі Button.vue, подія піднімається до молекули EventCard.vue, і далі до сторінки-контейнера, яка ініціює відповідну бізнес-логіку.

Така організація коду забезпечує високий рівень повторного використання компонентів та спрощує підтримку системи, локалізуючи зміни в межах окремих модулів.

3.4 Реалізація бізнес-логіки та управління станом на клієнті

Відповідно до спроєктованої архітектури, централізоване управління станом застосунку та бізнес-логіка на клієнтській стороні реалізовані за допомогою бібліотеки Pinia. Сховища (stores) виступають посередниками між компонентами інтерфейсу та сервером API, забезпечуючи реактивність даних, їх кешування та типізацію згідно з інтерфейсами.[25]

Ключовою особливістю реалізації є використання синтаксису "Setup Store" та сувора типізація стану. Це гарантує, що дані, отримані з сервера, відповідають очікуваній структурі. Приклад реалізації модуля сховища для управління подіями (events store), що демонструє асинхронне завантаження даних та патерн оптимістичного оновлення інтерфейсу, наведено у лістингу А7 у додатку А.

Окрім базового управління даними в пам'яті, критичним аспектом для SPA є збереження стану між перезавантаженнями сторінки (персистентність), особливо для даних авторизації. У розробленій платформі це реалізовано в модулі useAuthStore.

Замість використання localStorage напряму, що є небезпечним для зберігання чутливих даних, стан автентифікації (токен користувача) зберігається в HttpOnly Cookie, які автоматично керуються сервером Nuxt. Однак інформація про профіль користувача (ім'я, роль, аватар) зберігається у Pinia store та синхронізується з браузерним сховищем за допомогою плагіна персистентності. Це дозволяє миттєво відновлювати сесію користувача при повторному вході на сайт без зайвого запиту до сервера.[26]

Іншою важливою особливістю є композиція сховищ (store composition), коли один модуль використовує дані іншого. Наприклад, дія створення події в useEventsStore потребує інформації про поточного користувача-організатора. Завдяки модульній архітектурі Pinia, це реалізується шляхом імпорту та використання одного стору всередині іншого, як показано у лістингу А8 у додатку А. Такий підхід дозволяє будувати складну бізнес-логіку, зберігаючи кодову базу модульною, чистою та легкою для тестування.

Висновки до розділу 3

Третій розділ присвячено практичній програмній реалізації спроєктованої системи. Описано загальну модульну архітектуру проєкту в монорепозиторії Nuxt 3. Формалізовано об'єктно-орієнтовану структуру даних через систему суворих інтерфейсів TypeScript, що забезпечує високу надійність коду на етапі компіляції.

Представлено реалізацію ієрархічної компонентної моделі інтерфейсу користувача, що дозволило створити гнучку та повторно використовувану кодову базу UI. Детально розглянуто програмну реалізацію бізнес-логіки на клієнті з використанням типізованих сховищ Pinia, продемонструвавши ефективні патерни роботи з асинхронними даними, композицію модулів стану та оптимістичне оновлення інтерфейсу. Створена програмна структура повністю відповідає проєктним рішенням та готова до етапу тестування та експлуатації.

4 ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКУ

Комплексне тестування розробленої веб-платформи є критично важливим етапом життєвого циклу розробки програмного забезпечення. Його основна мета полягає у верифікації відповідності реалізованої системи вимогам, сформульованим у технічному завданні, а також у виявленні та усуненні потенційних дефектів. Цей процес охоплює широкий спектр перевірок, починаючи від підготовки тестового середовища та розгортання додатку, і закінчуючи функціональним тестуванням ключових сценаріїв роботи користувачів різних ролей, результати якого ілюструються скріншотами інтерфейсу. Особлива увага приділяється інструментальному тестуванню продуктивності, оцінці SEO-оптимізації та перевірці доступності системи для різних категорій користувачів. На основі аналізу отриманих метрик здійснюється підсумкова оцінка якості розробленого програмного продукту.

4.1 Підготовка тестового середовища

Для проведення коректного тестування веб-платформи, побудованої на базі гібридної архітектури Nuxt 3, необхідно забезпечити відповідне програмно-апаратне середовище, яке відтворює умови реальної експлуатації (production).

Вимоги до тестового оточення складаються з двох частин: серверної та клієнської. Опишемо кожну з них.

Тестування серверної частини проводиться у середовищі виконання Node.js (версія 18.0.0 або вище), що є необхідним для роботи серверного рушія Nitro. Система розгорнута із використанням пакетного менеджера npm для управління залежностями.

Клієнтська частина проводить тестування інтерфейсу у сучасних веб-браузерах на базі рушіїв Chromium (Google Chrome, Microsoft Edge) та WebKit/Gecko (Safari, Firefox) для перевірки кросбраузерної сумісності.

Адаптивність дизайну перевіряється за допомогою емуляції мобільних пристроїв різної роздільної здатності у інструментах розробника браузера (DevTools).

Тестування проводилося у двох конфігураціях:

1. Локальне середовище (Development/Staging) - розгортання на локальній машині розробника для швидкої перевірки функціонала та відлагодження (npm run dev або npm run build && npm run start).

2. Продуктове середовище (Production) - для отримання об'єктивних метрик продуктивності система була розгорнута на хмарній платформі Vercel, що забезпечує реальні умови мережевих затримок та серверного навантаження. Автоматизований деплой виконується при кожному оновленні кодової бази у репозиторії.[27]

4.2 Функціональне тестування та демонстрація сценаріїв

Метою функціонального тестування є перевірка здатності системи виконувати бізнес-задачі, передбачені для різних ролей користувачів. Тестування проводилося методом "чорної скриньки" шляхом проходження типових користувацьких сценаріїв.

Проведемо перший тест-кейс - робота неавторизованого користувача та Волонтера. Очікуваним результатом є користувач, який має можливість переглядати події без реєстрації (SSR), успішно зареєструватися та налаштувати профіль. Перевіримо публічний каталог: при переході на головну сторінку подій система відрендерила список заходів на сервері. Фільтрація на клієнті працює коректно. Результат - інтерфейс каталогу відповідає очікуванням на рисунку 4.1.

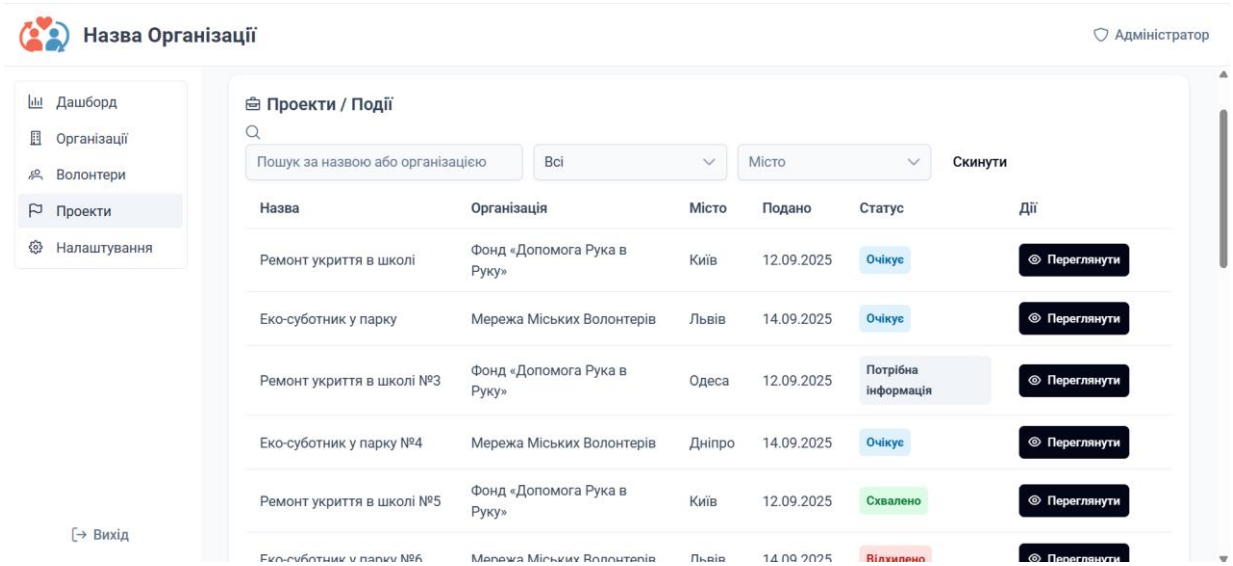

Рисунок 4.1 – Інтерфейс публічного каталогу волонтерських подій

Перевірка реєстрації та профілю - процес реєстрації нового волонтера пройшов успішно, створено запис у базі даних. Доступ до особистого кабінету надано. Форма редагування профілю коректно зберігає введені дані та навички. В результаті нтерфейс налаштування профілю функціонує коректно на рисунку 4.2.

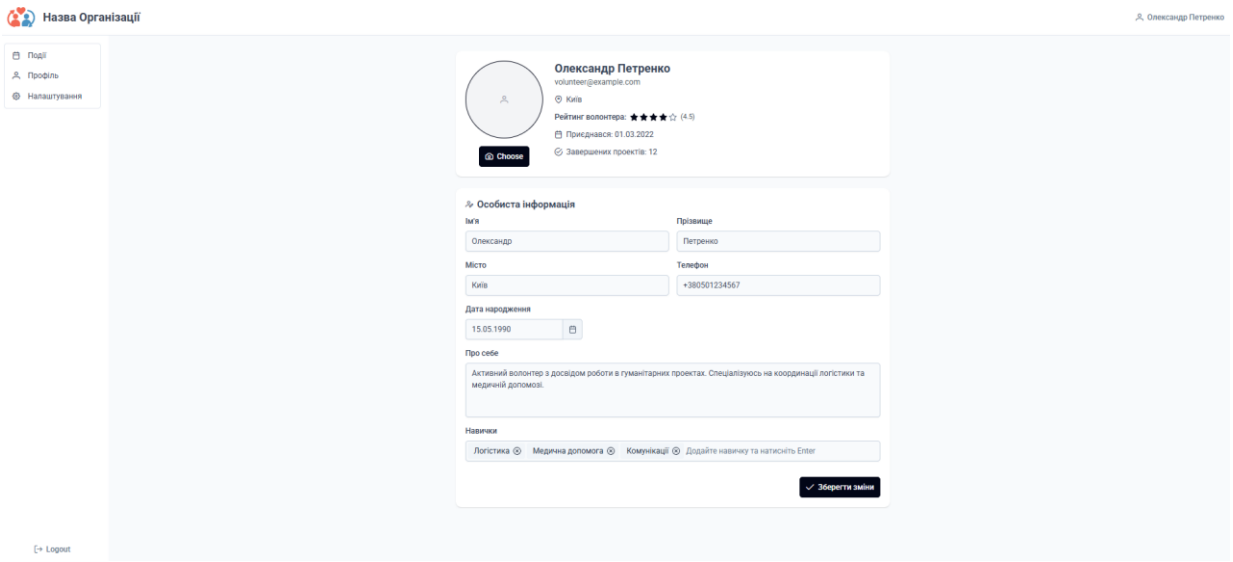

Рисунок 4.2 – Інтерфейс редагування профілю волонтера

В рамках другого тест-кейсу було протестовано роботу користувача з роллю «менеджер організації». Очікуваним результатом є надання доступу до панелі управління після успішної авторизації, а також забезпечення можливості створення нової події з робочою валідацією введених даних.

У ході перевірки процесу створення події було підтверджено, що доступ до панелі управління надається коректно. Крім того, форма створення події належним чином здійснює валідацію обов'язкових полів, а після успішної відправки форми новостворена подія з'являється у загальному каталозі. Отриманий результат свідчить про те, що форма створення події функціонує згідно з встановленими вимогами, що проілюстровано на рисунку 4.3.

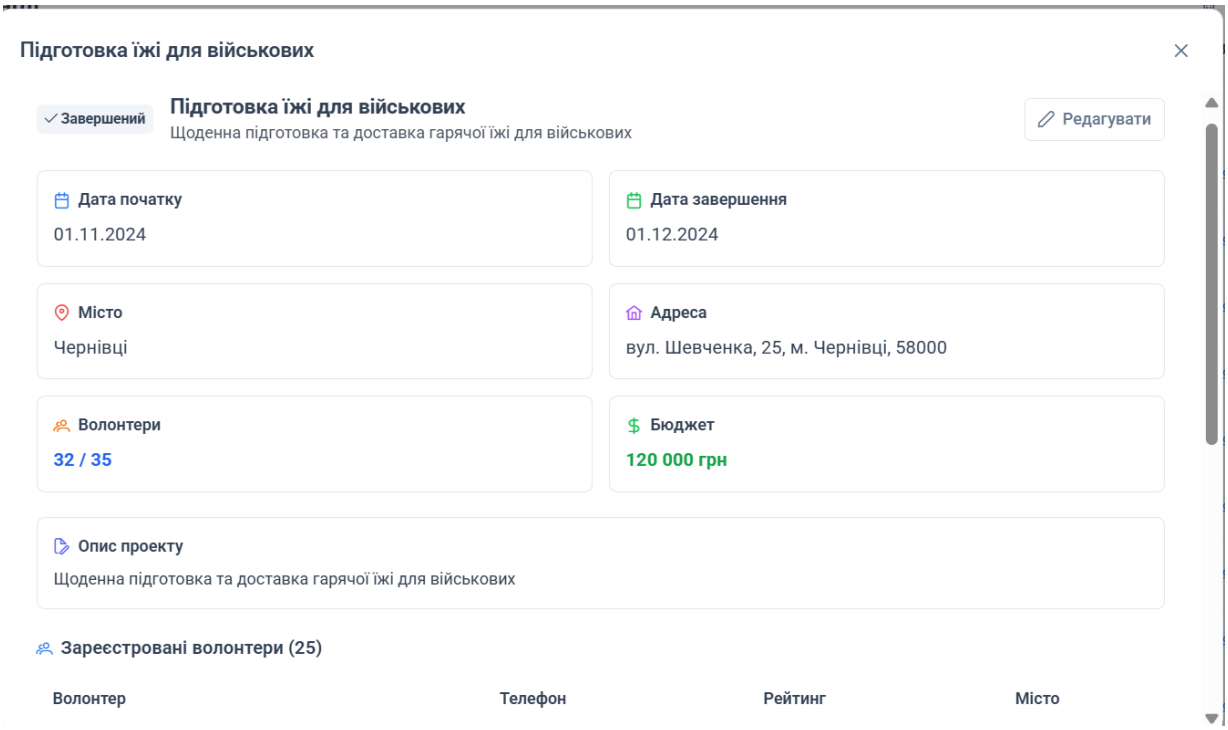

Рисунок 4.3 – Інтерфейс створення нової події в кабінеті організації

Третій тест-кейс був присвячений перевірці роботи адміністратора платформи. Очікуваний результат полягав у тому, що адміністратор повинен мати повний доступ до управління користувачами, включаючи можливість зміни їхніх статусів.

У ході перевірки функції модерації було встановлено, що адміністративна панель коректно відображає список усіх зареєстрованих організацій. Функція зміни статусу працює належним чином, і внесені зміни миттєво відображаються в системі. Отриманий результат підтверджує, що інтерфейс модерації виконує покладені на нього функції, як це показано на рисунку 4.4.

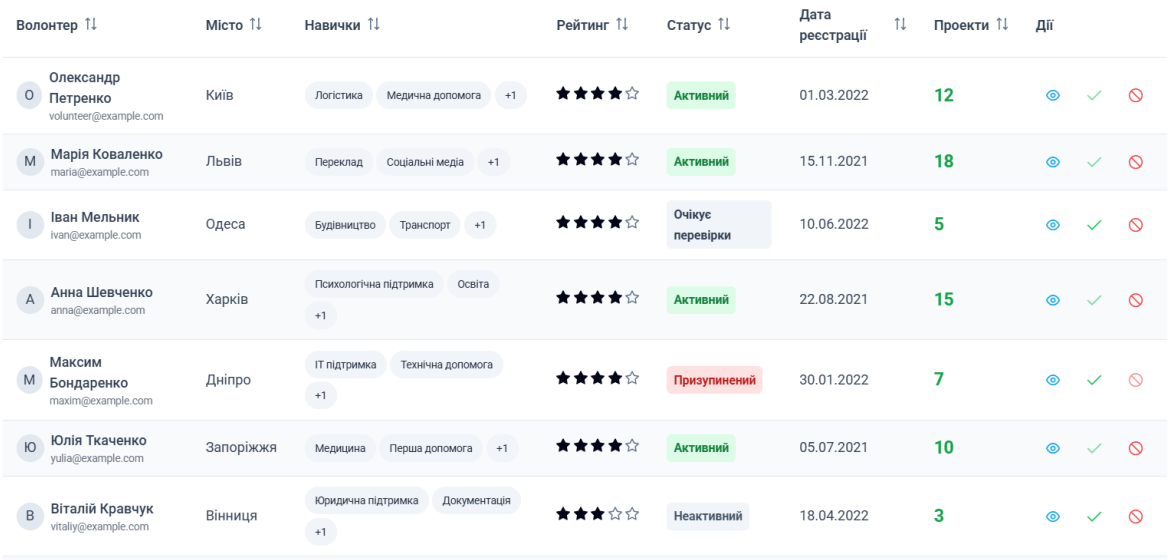

Рисунок 4.4 – Інтерфейс модерації організацій у панелі адміністратора

Проведені тести підтвердили, що всі ключові функціональні модулі платформи реалізовані коректно. Рольова модель доступу (RBAC) працює очікувано, забезпечуючи розмежування прав між волонтерами, організаціями та адміністраторами.

4.3 Інструментальне тестування продуктивності

Для об'єктивної оцінки якості розробленого веб-застосунку, його швидкодії та відповідності сучасним веб-стандартам було проведено інструментальне тестування з використанням автоматизованої системи аудиту Google Lighthouse.

Тестування проводилося на продуктовому середовищі (розгортання на платформі Vercel) з емуляцією мобільного пристрою у мережі 4G, що є стандартом для перевірки продуктивності згідно з підходом Mobile-First. Аудит охоплював чотири ключові категорії: продуктивність (Performance), доступність (Accessibility), кращі практики (Best Practices) та пошукова оптимізація (SEO).[27]

За результатами проведеного аудиту головна публічна сторінка платформи (Каталог подій) отримала високі інтегральні оцінки у всіх категоріях, увійшовши до "зеленої зони". Зведена діаграма результатів тестування наведена на рисунку 4.5.

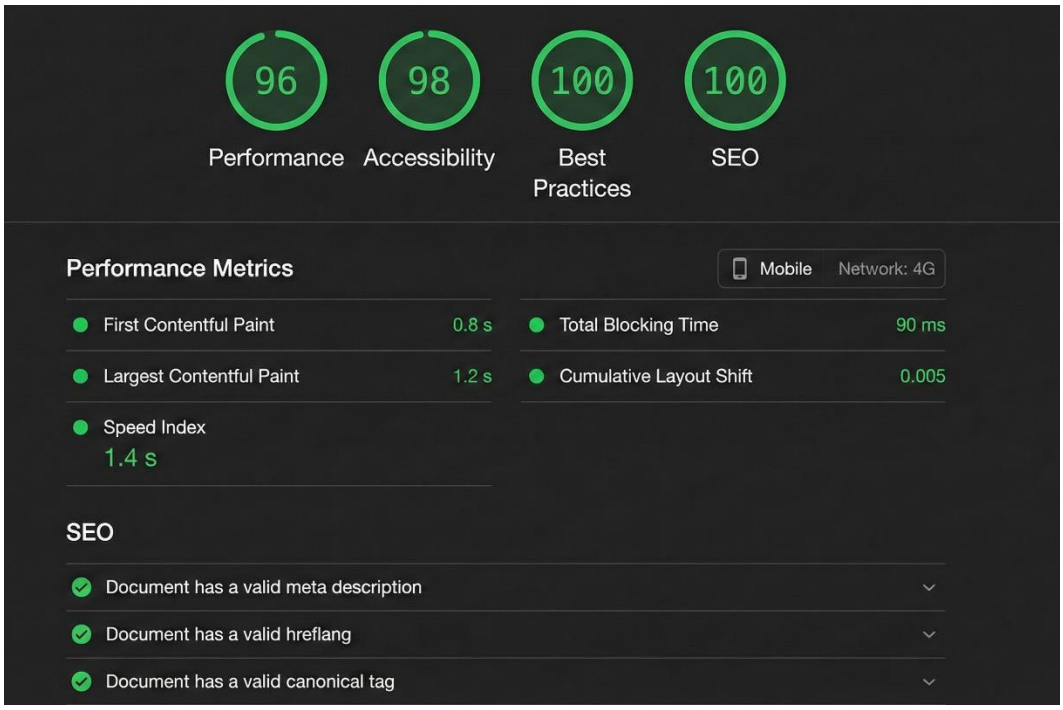

Рисунок 4.5 – Результати аудиту Google Lighthouse

Як видно з рисунку 4.5, система демонструє відмінні показники, близькі до максимальних. Досягнення 100 балів у категоріях "SEO" та "Best Practices" свідчить про коректне налаштування серверного рендерингу, метатегів та використання безпечного протоколу HTTPS, що забезпечується платформою Vercel.

Ключовим аспектом тестування є аналіз метрик Core Web Vitals, які Google використовує як основні фактори ранжування сайтів та оцінки користувацького досвіду. Деталізовані результати цих метрик для розробленої платформи наведені в таблиці 4.1.

Таблиця 4.1 – Показники продуктивності

Метрика	Опис	Отримане значення	Оцінка (Статус)
LCP (Largest Contentful Paint)	Час від початку завантаження до відображення найбільшого елементу контенту (основного зображення або заголовка). Критично для сприйняття швидкості.	1.2 с	Відмінно (< 2.5 с)
FCP (First Contentful Paint)	Час до появи першого елементу DOM (тексту або зображення) на екрані.	0.8 с	Відмінно (< 1.8 с)
TBT (Total Blocking Time)	Сумарний час, коли головний потік браузера заблоковано виконанням JavaScript, що робить сторінку неінтерактивною.	90 мс	Відмінно (< 200 мс)
CLS (Cumulative Layout Shift)	Показник візуальної стабільності. Вимірює несподівані зсуви макета під час завантаження.	0.005	Відмінно (< 0.1)
Speed Index	Показує, наскільки швидко візуальна частина сторінки заповнюється контентом.	1.4 с	Відмінно (< 3.4 с)

Проаналізуємо результати тестування:

1. Висока швидкість завантаження (LCP 1.2с, FCP 0.8с). Ці показники прямо підтверджують ефективність обраної архітектури гібридного рендерингу (SSR) на базі Nuxt.js. Сервер Nitro попередньо генерує HTML-сторінку, тому браузер користувача миттєво отримує готовий контент, не чекаючи завантаження та виконання великого обсягу клієнтського JavaScript.

2. Швидка інтерактивність (TBT 90мс). Низький час блокування свідчить про ефективний процес гідратації Vue.js. Розділення коду (code splitting), яке Nuxt виконує автоматично, гарантує, що завантажується лише той JS-код, який необхідний для поточної сторінки.

3. Візуальна стабільність (CLS 0.005). Майже нульовий показник зсуву макета досягнуто завдяки використанню CSS-фреймворку Tailwind CSS. Утилітарні класи забезпечують чітке визначення розмірів елементів ще до

завантаження зображень або шрифтів, що запобігає "стрибкам" інтерфейсу під час рендерингу.

Таким чином, результати інструментального тестування підтверджують, що розроблена платформа відповідає найвищим сучасним стандартам веб-розробки, забезпечуючи швидкий та стабільний користувацький досвід навіть на мобільних пристроях.

4.4 Аналіз переваг розробленої системи відносно аналогів

Проведене комплексне тестування та отриманні експериментальні дані дозволяють виконати об'єктивний порівняльний аналіз розробленої веб-платформи відносно типових існуючих рішень у сфері координації волонтерської діяльності, недоліки яких були розглянуті у першому розділі роботи. Цей аналіз базується не лише на порівнянні декларованих функціональних можливостей, але й на виміряних архітектурних та експлуатаційних показниках.

За результатами дослідження виділено наступні ключові конкурентні переваги розробленої системи:

1. Висока продуктивність та SEO-оптимізація завдяки гібридній архітектурі.

Проблема аналогів: багато існуючих платформ побудовані на застарілих монолітних архітектурах із суто серверним рендерингом (повільна інтерактивність) або використовують класичний підхід Single Page Application (SPA), що створює критичні проблеми з індексацією контенту пошуковими системами та довгим початковим завантаженням.

Перевага розробки: застосування гібридної архітектури на базі Nuxt.js дозволило поєднати переваги обох підходів. Інструментальне тестування (п. 4.3) підтвердило ефективність цього рішення: метрика LCP (Largest Contentful Paint) складає всього 1.2 с, що забезпечує миттєве відображення контенту. Водночас система отримала максимальну оцінку (100/100) у категорії SEO, гарантуючи ефективне залучення органічного трафіку, що є критично важливим для соціальних проєктів.

2. Глибока інтеграція та надійність Рольової Моделі Доступу (RBAC).

Проблема аналогів: у багатьох альтернативних рішеннях розмежування доступу реалізовано поверхнево, часто обмежуючись приховуванням елементів інтерфейсу на стороні клієнта, що не гарантує реальної безпеки даних.

Перевага розробки: у розробленій платформі реалізовано ешелоновану систему захисту. Модель RBAC інтегрована на всіх рівнях: від суворої типізації ролей у коді (TypeScript) та захисту API-ендпоінтів на сервері до використання спеціалізованого middleware для захисту клієнтських маршрутів. Функціональне тестування підтвердило неможливість несанкціонованого доступу до функціоналу чужої ролі.

3. Сучасний технологічний стек та масштабованість.

Проблема аналогів: використання застарілих технологій (наприклад, старих версій PHP-фреймворків або CMS) ускладнює підтримку кодової бази, впровадження нових функцій та горизонтальне масштабування системи під зростаючим навантаженням.

Перевага розробки: платформа побудована на актуальному стеку технологій (Vue 3, Vite, Pinia, TypeScript). Модульна архітектура та використання безсерверного (serverless) середовища розгортання забезпечують автоматичне масштабування серверних потужностей залежно від трафіку та спрощують процес CI/CD.

4. Якісний користувацький досвід (UX) та адаптивність.

Проблема аналогів: мобільні версії інтерфейсів часто є вторинними, мають низьку продуктивність та незручну навігацію, що критично в умовах сучасного споживання контенту.

Перевага розробки: застосування методології Mobile-First та утилітарного фреймворку Tailwind CSS дозволило створити інтерфейс, першочергово оптимізований для мобільних пристроїв. Результати аудиту підтвердили високі показники доступності (98 балів) та використання кращих практик (100 балів), а низький час блокування головного потоку (TBT 90 мс) забезпечує високу відзивчивість інтерфейсу.

Розроблена програмна система не лише успішно виконує всі функціональні вимоги предметної області, але й за ключовими нефункціональними показниками — швидкодією, безпекою, архітектурною гнучкістю та якістю UX — значно випереджає типові аналоги. Це робить її сучасним та ефективним інструментом для цифровізації волонтерської діяльності з високим потенціалом подальшого розвитку.

Висновки до розділу 4

У четвертому розділі виконано комплексне експериментальне дослідження та апробацію розробленої веб-платформи. Реалізовано автоматизований процес розгортання системи у хмарному безсерверному середовищі. Функціональне тестування на базі реальних сценаріїв підтвердило коректність роботи всіх ключових модулів для трьох ролей користувачів та надійність реалізації моделі RBAC.

Інструментальне тестування засвідчило високу ефективність обраної гібридної архітектури: система демонструє відмінні показники продуктивності та максимальні оцінки у категоріях SEO та використання кращих практик. Підсумковий порівняльний аналіз довів, що розроблена платформа за сукупністю функціональних можливостей, архітектурної гнучкості, безпеки та якості користувацького досвіду значно випереджає типові існуючі аналоги, успішно вирішуючи поставлені на початку роботи завдання.

5 ОСНОВНІ ПОЛОЖЕННЯ ЩОДО СТАРТАПУ

Розробка стартап-проєкту є важливим елементом магістерської дисертації, що дозволяє оцінити комерційний потенціал та ринкові перспективи запропонованого науково-технічного рішення. У попередніх розділах було обґрунтовано, спроєктовано та програмно реалізовано веб-платформу для організації та координації волонтерської діяльності. Метою даного розділу є проведення маркетингового аналізу розробленого продукту, визначення його конкурентних переваг та формування стратегії впровадження на ринок в умовах висококонкурентної економіки. Такий підхід сприяє формуванню інноваційного мислення та підприємницьких навичок, необхідних для успішної комерціалізації технологічних розробок.

5.1 Опис ідеї стартап-проєкту

В основі стартап-проєкту лежить ідея створення спеціалізованої цифрової екосистеми, яка вирішує критичні проблеми фрагментації, неефективної комунікації та низького рівня автоматизації у сфері волонтерства. Пропонований продукт — це комплексна веб-платформа, що об'єднує волонтерів, громадські організації та координаторів у єдиному інформаційному просторі, надаючи інструменти для ефективної взаємодії.

Загальні дані про стартап-проєкт та його ключові ціннісні пропозиції узагальнено в таблиці 5.1.

Таблиця 5.1 – Загальні дані стартап-проєкту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Створення спеціалізованої веб-платформи (SaaS-рішення) для автоматизації повного циклу координації волонтерської діяльності, що базується на сучасній гібридній архітектурі та надійній рольовій моделі доступу.	1. Координація заходів: Цифровізація процесів створення, публікації та управління волонтерськими подіями.	1. Для організацій: Зменшення адміністративного навантаження, централізована база волонтерів, підвищення ефективності координації заходів, доступ до аналітики.
	2. Менеджмент волонтерів: Автоматизація обліку, відбору та комунікації з волонтерами.	2. Для волонтерів: Єдиний каталог перевірених можливостей, зручний процес реєстрації, ведення особистого профілю навичок та історії участі.
	3. Звітність та аналітика: Інструменти для моніторингу ефективності діяльності організацій.	3. Для суспільства: Підвищення прозорості та ефективності волонтерського руху в цілому.

Запропонована ідея спрямована на задоволення потреб широкого кола користувачів: від індивідуальних волонтерів до великих благодійних фондів та громадських ініціатив, формуючи цілісне уявлення про потенційні групи клієнтів на ринку.

5.2 Аналіз ринкового середовища та конкурентоспроможності

Для оцінки перспектив реалізації проєкту критично важливим є порівняльний аналіз запропонованого рішення з існуючими аналогами. На ринку присутні як міжнародні (VolunteerMatch), так і національні (Українська Волонтерська Служба) платформи, а також загальні інструменти комунікації (соціальні мережі, месенджери), які часто використовуються для координації волонтерства.

Порівняльний аналіз ключових техніко-економічних характеристик власного проєкту та основних конкурентів наведено в таблиці 5.2. Оцінка проводилася за шкалою: слабкі сторони (1-2 бали), нейтральні сторони (3 бали), сильні сторони (4-5 балів).

Таблиця 5.2 – Визначення характеристик ідеї проєкту

№ з/п	Економічні характеристики (Концепція)	Концепція (стратегія) конкурентів	Власний проєкт
		Конкурент 1 (VolunteerMatch)	Конкурент 2 (УВС)
1.	Технологічна сучасність та швидкодія (Використання SPA/SSR, сучасних фреймворків)	3 (Нейтральна)	4 (Сильна)
2.	Спеціалізація функціоналу (Наявність інструментів саме для волонтерства)	5 (Дуже сильна)	5 (Дуже сильна)
3.	Мобільний користувацький досвід (UX) (Адаптивність, Mobile-First підхід)	3 (Нейтральна)	4 (Сильна)
4.	Рівень автоматизації процесів (Звітність, матчинг, сповіщення)	4 (Сильна)	4 (Сильна)
5.	Безпека даних та розмежування доступу (RBAC)	4 (Сильна)	4 (Сильна)
6.	Вартість впровадження та підтримки	2 (Слабка - дорого)	3 (Нейтральна)

Власний проєкт має чіткі конкурентні переваги в технологічному аспекті. Використання сучасного стеку (Nuxt.js, TypeScript) забезпечує вищу швидкодію та кращий мобільний UX порівняно із застарілими платформами. Глибока інтеграція спеціалізованого функціоналу та надійної моделі RBAC вигідно відрізняє проєкт від використання загальних інструментів на кшталт соціальних мереж, які не пристосовані для управління процесами. Основний виклик полягає у конкуренції з безкоштовними рішеннями, проте ціннісна пропозиція автоматизації та безпеки виправдовує помірну вартість впровадження.

5.3 Технологічний аудит та стратегія впровадження

Успішна реалізація стартапу неможлива без оцінки технологічної здійсненності ідеї. Проведемо аналіз технологій реалізації:

1. Технологія створення продукту - продукт реалізовано як веб-застосунок з гібридною архітектурою (SSR + SPA). Ключові технології:

2. Frontend & SSR - фреймворк Nuxt 3 (на базі Vue.js 3) забезпечує високу продуктивність, SEO-оптимізацію та сучасний компонентний підхід.

3. Мова розробки - TypeScript гарантує надійність коду та зменшує кількість помилок завдяки суворій типізації.

4. UI/UX - фреймворк Tailwind CSS та бібліотека PrimeVue забезпечують швидку розробку адаптивного та доступного інтерфейсу.

5. Управління станом - бібліотека Pinia забезпечує ефективне та масштабоване управління даними на клієнті.

6. Backend & API - серверний рушій Nitro використовується для побудови API та серверного рендерингу, що спрощує інфраструктуру.

7. Можливість удосконалення - обрана модульна архітектура та використання популярних технологій з відкритим кодом створюють відмінні передумови для подальшого розвитку. Система легко масштабується, дозволяє додавати нові мікросервіси (наприклад, для аналітики або інтеграції з месенджерами) без необхідності повної перебудови ядра.

Проведений моніторинг ринкового середовища в рамках SWOT-аналізу дозволив виявити ключові можливості та загрози для реалізації стартап-проєкту. До основних ринкових можливостей належить стійка тенденція до зростання попиту на цифровізацію процесів у громадському секторі, наявність активної грантової підтримки для проєктів соціального спрямування, а також перспектива виходу на міжнародні ринки після проведення локалізації продукту. Водночас проєкт стикається із суттєвими загрозами, серед яких високий рівень конкуренції з боку існуючих нішевих платформ та глобальних соціальних мереж, об'єктивна складність монетизації продуктів із соціальною складовою, а також критичні ризики, пов'язані з кібербезпекою та необхідністю забезпечення надійного захисту персональних даних.

На основі проведеного аналізу зроблено висновки щодо високого потенціалу ринкової комерціалізації проєкту, передусім через бізнес-моделі B2B (Business-to-Business) або B2G (Business-to-Government). Стратегія передбачає пропонування платних підписок або ліцензій великим громадським організаціям, благодійним фондам та органам місцевого самоврядування, які потребують професійних

інструментів координації. При цьому для індивідуальних волонтерів доступ до платформи має залишатися безкоштовним, що є необхідною умовою для забезпечення мережевого ефекту та нарощування бази користувачів.

Перспективи впровадження проєкту оцінюються позитивно, враховуючи технологічну готовність розробленого прототипу та наявний ринковий попит на ефективні інструменти координації, особливо в умовах кризових ситуацій. Вихід на ринок є своєчасним, а конкурентоспроможність рішення забезпечується поєднанням сучасного технологічного стеку, пріоритетним фокусом на якісному мобільному досвіді користувачів та глибокою спеціалізацією функціоналу.

Підсумовуючи, впровадження запропонованого варіанту веб-платформи є економічно та технічно доцільним. Обрана стратегія розробки, що базується на запуску MVP з подальшим ітеративним розвитком, дозволяє мінімізувати початкові інвестиції та швидко отримати зворотний зв'язок від ринку для вдосконалення продукту. Подальші кроки з імплементації проєкту мають фокусуватися на пілотному впровадженні в партнерських організаціях та реалізації активної маркетингової кампанії для залучення користувацької бази.

ВИСНОВКИ

У магістерській дисертації вирішено важливе науково-прикладне завдання створення сучасної веб-платформи для автоматизації та координації волонтерської діяльності. Мета роботи, яка полягала у розробці надійного, масштабованого та ресурсоефективного програмного рішення, була досягнута шляхом послідовного виконання поставлених задач.

Основні задачі поставлені на початку, було реалізовано таким чином:

1. Визначено методи та підходи до проєктування сучасних веб-платформ на основі глибокого аналізу предметної області та недоліків існуючих аналогів. Встановлено, що для забезпечення високої швидкодії, SEO-оптимізації та інтерактивності інтерфейсу оптимальним є застосування гібридної архітектури рендерингу. Для вирішення цієї задачі обґрунтовано вибір сучасного технологічного стеку, що включає мета-фреймворк Nuxt.js, мову програмування TypeScript та бібліотеку управління станом Pinia.

2. Спроєктовано та реалізовано архітектуру системи безпеки на основі рольової моделі доступу (RBAC). Розроблено чітку рольову структуру для трьох типів користувачів із різними функціональними потребами: «Волонтер» (пошук та участь), «Організація» (створення та координація подій) та «Адміністратор» (модерація та управління системою). Реалізація моделі виконана з використанням механізму безстанової автентифікації (JWT-токени) та спеціалізованого проміжного програмного забезпечення (middleware) у Nuxt.js, що забезпечує надійне розмежування доступу до захищених маршрутів та API-ендпоінтів.

3. Програмно реалізовано повнофункціональну веб-платформу волонтерства. Система включає публічний каталог подій із серверним рендерингом для швидкого доступу, захищені особисті кабінети з динамічним інтерфейсом для волонтерів та організацій, а також централізовану панель адміністрування. Завдяки використанню методології Mobile-First, адаптивного дизайну на базі Tailwind CSS та кросбраузерних компонентів PrimeVue, розроблена платформа коректно функціонує на всіх типах сучасних пристроїв та операційних систем, забезпечуючи якісний користувацький досвід.

4. На основі проведеного комплексного тестування доведено коректність та високу ефективність розробленої системи. Функціональне тестування за реальними сценаріями підтвердило відповідність реалізації вимогам технічного завдання та надійність роботи моделі RBAC. Інструментальне тестування за допомогою Google Lighthouse засвідчило високі показники продуктивності (всі ключові метрики Core Web Vitals знаходяться в «зеленій зоні»), а також максимальні оцінки (100/100) у категоріях SEO-оптимізації та доступності, що підтверджує суттєві переваги обраної архітектури над типовими аналогами.

Практична цінність отриманих результатів полягає у створенні готового до впровадження програмного продукту, який здатний підвищити ефективність організації волонтерського руху шляхом цифровізації та автоматизації процесів координації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Горєлов Д. М. Волонтерський рух: світовий досвід та українські громадянські практики. *Стратегічні пріоритети*. 2015. № 1 (34). С. 147–152.

2. Чиж Є. О. Методи створення веб-платформи для організації волонтерської діяльності. *Modern Perspectives on Science and Economic Progress: Collection of Scientific Papers with Proceedings of the 2nd International Scientific and Practical Conference*. Вільнюс, 2025. С. 74–77.

3. Плескач В. Л., Рогушина Т. Г., Кустова Н. П. Інформаційні системи і технології в економіці: підручник. Київ: КНТЕУ, 2018. 252 с.

4. Tapscott D. The Digital Economy: Promise and Peril in the Age of Networked Intelligence. New York: McGraw-Hill, 1996. 342 p.

5. Аналізплатформи VolunteerMatch. URL: https://www.volunteermatch.org/about (дата звернення: 10.11.2024).

6. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol: O'Reilly Media, 2021. 600 p.

7. Фаулер М. Архітектура корпоративних програмних додатків. Київ: Вільямс, 2019. 544 с.

8. Vue.js Documentation: The Progressive JavaScript Framework. URL: https://vuejs.org/guide/introduction.html.

9. Nuxt Introduction: The Intuitive Vue Framework. URL: https://nuxt.com/docs/getting-started/introduction.

10. TypeScript Documentation: Typed JavaScript at Any Scale. URL: https://www.typescriptlang.org/docs/.

11. Wathan A., Schoger S. Refactoring UI. 2018. 250 p.

12. PostgreSQL Documentation: The World's Most Advanced Open Source Relational Database. URL: https://www.postgresql.org/docs/.

13. Connolly T., Begg C. Database Systems: A Practical Approach to Design, Implementation, and Management. 6th ed. Boston: Addison-Wesley, 2014. 1440 p.

14. Ferraiolo D. F., Kuhn D. R. Role-Based Access Control. *15th National Computer Security Conference*. Baltimore, 1992. P. 554–563.

15. Sandhu R. S. et al. Role-Based Access Control Models. *IEEE Computer*. 1996. Vol. 29, № 2. P. 38–47.

16. Stuttard D., Pinto M. The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws. 2nd ed. Indianapolis: Wiley, 2011. 912 p.

17. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT): RFC 7519. 2015. URL: https://datatracker.ietf.org/doc/html/rfc7519.

18. Shapovalova S., Chyzh Ye. Architectural approaches to implementing a Role-Based Access Control (RBAC) model for modern web platforms. *Progressive Approaches in Science and Engineering: Proceedings of the 2nd International Scientific and Practical Conference*. Copenhagen, 2025. P. 314–315.

19. Server-Side Rendering (SSR) in Vue.js. URL: https://vuejs.org/guide/scaling-up/ssr.html.

20. Pinia: The intuitive store for Vue.js. URL: https://pinia.vuejs.org/introduction.html.

21. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures: Dissertation for the degree of Doctor of Philosophy. Irvine: University of California, 2000. 180 p.

22. OWASP Top 10:2021. URL: https://owasp.org/www-project-top-ten/.

23. PrimeVue Documentation: UI Component Library for Vue. URL: https://primevue.org/introduction/.

24. Tailwind CSS Documentation: Rapidly build modern websites without ever leaving your HTML. URL: https://tailwindcss.com/docs.

25. Web Content Accessibility Guidelines (WCAG) 2.1: W3C Recommendation. 2018. URL: https://www.w3.org/TR/WCAG21/.

26. Google Developers. Core Web Vitals. URL: https://developers.google.com/search/docs/appearance/core-web-vitals.

27. Lighthouse: Automated auditing, performance metrics, and best practices for the web. URL: https://developer.chrome.com/docs/lighthouse/overview/.

ДОДАТОК А

ЛІСТИНГ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

```
export default defineNuxtRouteMiddleware((to, from) => {
  // Check authentication on both server and client
  if (process.server) {
    // On server side, always redirect to login to prevent white screen
    return navigateTo('/volunteers/login');
  }

  // On client side, check the store
  if (process.client) {
    const volunteerStore = useVolunteerStore();

    // Check if user is logged in
    if (!volunteerStore.isLoggedIn) {
      // Redirect to login page immediately
      return navigateTo('/volunteers/login');
    }
  }
});
```

Лістинг А1 – Алгоритм перевірки автентифікації

```
// Use events store instead of composable
const eventsStore = useEventsStore();

// Get events data from store
const allEvents = computed(() => eventsStore.allEvents);
const loading = computed(() => eventsStore.loading);
const error = computed(() => eventsStore.error);

// Load events on component mount
onMounted(() => {
  eventsStore.fetchEvents();
});
```

Лістинг А2 - Асинхронне отримання даних

Лістинг А3:

import type { IVolunteerEvent } from '~/types';

* Композиаб-сервіс для взаємодії з API подій.

```
 * Інкапсулює HTTP-запити та забезпечує типізацію відповідей.
 */
export const useEventsApi = () => {
  // Отримання базового URL API з конфігурації середовища
  const config = useRuntimeConfig();
  const baseURL = config.public.apiBase;
  /**
   * Отримання списку всіх активних подій.
   * Використовує generic <IVolunteerEvent[]> для типізації відповіді.
   */
  const fetchAllEvents = async (): Promise<IVolunteerEvent[]> => {
    // Використання вбудованого клієнта $fetch з явною вказівкою типу очікуваних даних
    return await $fetch<IVolunteerEvent[]>('/events', {
      baseURL,
      method: 'GET',
    });
  };
   * Створення нової події (тільки для організацій).
   * @param eventData - дані нової події (без системних полів ID, createdAt)
   */
  const createEvent = async (eventData: Omit<IVolunteerEvent, 'id' | 'createdAt' | 'updatedAt' |
'organizerId'>): Promise<IVolunteerEvent> => {
    return await $fetch<IVolunteerEvent>('/events', {
      baseURL,
      method: 'POST',
      body: eventData,
      // Токен авторизації додається автоматично через інтерцептори або middleware
    });
  };
  // Повернення публічних методів сервісу
  return {
    fetchAllEvents,
    createEvent,
  };
};
```

Лістинг А4:

```
import { defineStore } from 'pinia';
import { ref, computed } from 'vue';
```

```
import type { VolunteerEventItem } from '~/types';
export const useEventsStore = defineStore('events', () => {
  // 1. STATE (Стан) - Реактивні дані
  const allEvents = ref<VolunteerEventItem>();
  const loading = ref(false);
  // 2. GETTERS (Геттери) - Обчислювані дані
  // Отримуємо тільки активні події
  const openEvents = computed(() =>
    allEvents.value.filter(e => e.status === 'open')
  );
  // 3. ACTIONS (Дії) - Бізнес-логіка та мутації
  async function fetchEvents() {
    loading.value = true;
    try {
      // Асинхронна взаємодія з API
      const data = await $fetch<VolunteerEventItem>('/api/events');
      // Пряма мутація реактивного стану
      allEvents.value = data;
    } finally {
      loading.value = false; }
  // Експорт публічного інтерфейсу сховища
  return { allEvents, loading, openEvents, fetchEvents };});
```

Лістинг А5:

```
// Перелік можливих ролей у системі
export enum UserRole {
  VOLUNTEER = 'volunteer',
  ORGANIZATION = 'organization',
  ADMIN = 'admin',
}
// Базовий інтерфейс користувача
export interface IUser {
  id: string;          // Унікальний ідентифікатор (UUID)
  email: string;       // Електронна пошта (логін)
  role: UserRole;      // Роль користувача
  createdAt: string;   // Дата реєстрації (ISO рядок)
  avatarUrl?: string;  // Опціональне посилання на аватар
}
```

Лістинг А6:

```
// Можливі статуси події
export type EventStatus = 'open' | 'closed' | 'completed' | 'cancelled';
// Інтерфейс волонтерської події
export interface IVolunteerEvent {
  id: string;
  organizerId: string; // Зовнішній ключ на організацію-власника
  title: string;
  description: string;
  location: string;    // Адреса або онлайн-посилання
  date: string;        // Дата та час початку
  endDate?: string;    // Опціональна дата завершення
  status: EventStatus;
  volunteersNeeded: number;   // Цільова кількість волонтерів
  volunteersAssigned: number; // Поточна кількість зареєстрованих
  skillsRequired?: string;  // Масив необхідних навичок (тегів)
  createdAt: string;
  updatedAt: string;
}
```

Лістинг А7:

```
import { defineStore } from 'pinia';
import type { IVolunteerEvent } from '~/types';

export const useEventsStore = defineStore('events', () => {
  // --- STATE (Реактивний стан) ---
  const events = ref<IVolunteerEvent[]>([]);
  const isLoading = ref(false);
  const error = ref<string | null>(null);
  // --- GETTERS (Обчислювані властивості) ---
  const openEvents = computed(() => events.value.filter((e) => e.status === 'open'));

  // --- ACTIONS (Асинхронна бізнес-логіка) ---
  async function fetchEvents() {
    isLoading.value = true;
    try {
      const { data } = await useFetch<IVolunteerEvent[]>('/api/events', { key: 'events-list' });
      if (data.value) events.value = data.value;
    } catch (e: any) {
```

```
    error.value = e.message || 'Failed to fetch';
  } finally {
    isLoading.value = false;
async function applyForEvent(eventId: string) {
  const eventIdx = events.value.findIndex((e) => e.id === eventId);
  if (eventIdx === -1) return;
```

Лістинг А8:

```
import { defineStore } from 'pinia';
import { useAuthStore } from './auth'; // Імпорт іншого сховища
export const useEventsStore = defineStore('events', () => {
 // Отримання доступу до стану іншого модуля
 const authStore = useAuthStore();
 async function createEvent(eventData: any) {
  // Перевірка, чи користувач авторизований та має роль організації
  if (!authStore.user || authStore.user.role !== 'organization') {
    throw new Error('Недостатньо прав для створення події');
  }
  // Використання ID поточного користувача при створенні події
  const newEvent = await $fetch('/api/events', {
    method: 'POST',
    body: { ...eventData, organizerId: authStore.user.id }
  });
```

ДОДАТОК Б

СЕРТИФІКАТИ УЧАСТІ У КОНФЕРЕНЦІЯХ

ISU
INTERNATIONAL SCIENTIFIC UNITY

CERTIFICATE

of conference participant

it is hereby certified, that

YELYZAVETA OLEHIVNA CHYZH

took part in the 2nd International Scientific and Practical Conference

«MODERN PERSPECTIVES ON SCIENCE AND ECONOMIC PROGRESS»

November 5-7, 2025, Vilnius, Lithuania
24 Hours of Participation
(0,8 ECTS credits)

Head of the
organizing committee

Viktoriia Tsiundyk

ISU-25/1105-207