ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка платформи для волонтерської взаємодії з використанням Web-фреймворку Next.js»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Владислав ІВАНЧУК
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

Владислав ІВАНЧУК

Керівник: Владислав ЯСКЕВИЧ
к.т.н.

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Іванчуку Владиславу Олександровичу

1. Тема кваліфікаційної роботи: «Розробка платформи для волонтерської взаємодії з використанням Web-фреймворку Next.js»
керівник кваліфікаційної роботи к.т.н., доцент кафедри ІПЗ Владислав ЯСКЕВИЧ, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: офіційна документація фреймворків і бібліотек, опис волонтерської діяльності, навчальні матеріали роботи з базами даних.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз існуючих рішень у сфері волонтерської взаємодії.
2. Огляд та аналіз технологій розроблення веб-платформ.
3. Проектування застосунку для волонтерської взаємодії.
4. Програмна реалізація, опис функціонування та тестування застосунку для волонтерської взаємодії.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до застосунку.
3. Програмні засоби реалізації.
4. ER-діаграма.
5. Діаграма діяльності.
6. Use Case діаграма
7. Структура платформи
8. Екранні форми
9. Апробація результатів дослідження

6. Дата видачі завдання«28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Огляд та аналіз існуючих рішень у сфері волонтерської взаємодії	14.03-17.03.2024	
4	Огляд та аналіз технологій розроблення веб-платформ	17.03-21.03.2024	
5	Проектування застосунку для волонтерської взаємодії	22.03-30.03.2024	
6	Програмна реалізація, опис функціонування та тестування застосунку для волонтерської взаємодії	31.03-28.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Владислав ІВАНЧУК
(підпис)

Керівник
кваліфікаційної роботи _____ Владислав ЯСКЕВИЧ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 55 стор., 2 табл., 32 рис., 25 джерел.

Мета роботи – підвищення ефективності волонтерської діяльності шляхом створення цифрової платформи для зручного спілкування, координації дій та залучення громадян до волонтерської діяльності.

Об'єкт дослідження – волонтерська діяльність як соціальне явище, що охоплює взаємодію волонтерів та громадськості через використання інформаційних технологій.

Предмет дослідження – веб-платформа на базі фреймворку Next.js.

Короткий зміст роботи: В роботі проаналізовано існуючі платформи для волонтерської взаємодії та виявлено ключові недоліки та обмеження їхньої функціональності. Розглянуто і порівняно основні технології розробки, які використовуються у створенні веб-застосунків, зокрема використання Next.js як основної платформи для фронтенду та бекенду. Розроблено архітектуру системи та варіанти використання платформи, програмно реалізовані основні модулі взаємодії користувачів з інтерфейсом та базою даних, зокрема систему реєстрації, пошуку розміщення та перегляду потреб і волонтерських можливостей. Проведено ряд тестувань, які підтвердили функціональність та стабільність роботи системи. В роботі використано сучасні підходи до серверної оптимізації та клієнтської взаємодії через використання SSR і CSR у межах фреймворку Next.js.

Сферою використання застосунку є підтримка і координація волонтерських ініціатив, що вимагають швидкого реагування на змінювані потреби в різних регіонах.

КЛЮЧОВІ СЛОВА: ВОЛОНТЕРСЬКА ВЗАЄМОДІЯ, NEXT.JS, ВЕБ-РОЗРОБКА, СЕРВЕРНИЙ РЕНДЕРИНГ, ІНТЕРФЕЙС КОРИСТУВАЧА.

ЗМІСТ

ВСТУП

У сучасному світі роль волонтерства стрімко зростає, особливо в контекстах соціальних криз і природних катастроф. В Україні, зокрема, активізація волонтерської діяльності відбулася у зв'язку з потребами населення під час воєнних дій та політичних криз. Розробка ефективних платформ для волонтерської взаємодії виступає важливим інструментом для координації зусиль і ресурсів, які можуть бути витрачені більш ефективно.

Розробка таких платформ є вкрай актуальною, оскільки існуючі рішення часто не враховують специфіку волонтерської діяльності в Україні або мають технічні та функціональні обмеження. Значна частина існуючих систем фокусується на зборі коштів або розподілі фізичних ресурсів, недостатньо уваги приділяючи координації дій волонтерів і аналізу потреб.

Ця кваліфікаційна робота спрямована на наступні основні задачі:

1. Аналіз існуючих рішень у сфері волонтерської взаємодії та виявлення їхніх основних недоліків.
2. Визначення вимог до розробки платформи волонтерської взаємодії.
3. Проектування архітектури системи волонтерської взаємодії.
4. Розробка інтерфейсу користувачів.
5. Реалізація функціональних можливостей платформи волонтерської взаємодії.
6. Тестування системи волонтерської взаємодії.

Робота пройшла апробацію: Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях» за темою «Дослідження системи рекомендацій для платформи волонтерської взаємодії»[1]; IV Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті» за темою «Безпека даних у хмарних сервісах» [2].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд волонтерської діяльності в сучасних умовах

Волонтерська діяльність є важливою складовою сучасного суспільства, яка зазнала значних змін та розвитку протягом останніх десятиліть. Значущість волонтерства значно зросла в Україні, особливо після початку військового конфлікту з Росією у 2014 році, та внаслідок повномасштабного вторгнення у 2022 році, коли волонтерська діяльність набула загальносуспільного значення. Волонтери відіграють важливу роль у підтримці Збройних Сил України, допомозі внутрішньо переміщеним особам, а також іншим категоріям людей, що постраждали від війни.

Закон України "Про волонтерську діяльність" визначає волонтерство як добровільну, соціально спрямовану, неприбуткову діяльність, що здійснюється волонтерами шляхом надання волонтерської допомоги [3]. Сучасне волонтерство в Україні відзначається не тільки соціальною спрямованістю, а й значною організованістю і професіоналізмом у різних сферах допомоги.

Волонтерство не має єдиного визначення і може варіюватись в різних культурних та соціальних контекстах. Волонтерська діяльність базується на принципах добровільності, де людина безкорисливо виконує роботу на користь інших. В Україні та Польщі проведено анкетування, що демонструє усвідомлене розуміння суті волонтерства та його основних характеристик [4].

Результати опитувань свідчать про відмінності у видах волонтерської діяльності між цими двома країнами. Українські волонтери активно підтримують військові потреби та допомагають постраждалим, тоді як в Польщі акцент зроблено на екологічну турботу та допомогу знедоленим, включаючи українських біженців. На рисунку 1.1 показано порівняння актуальних видів діяльності волонтерів в Україні та Польщі.

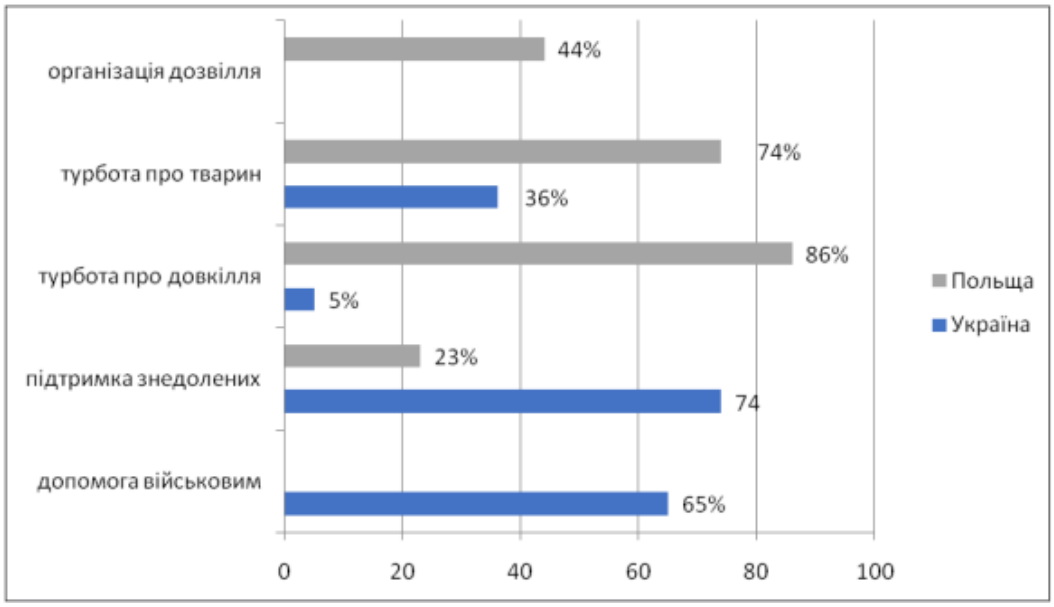


Рис. 1.1 Актуальні види діяльності волонтерів в сучасних умовах

Сприяння розвитку волонтерського руху в Україні має велике значення, особливо у період військових дій та криз. Згідно з даними анкетування, учасники волонтерської діяльності не тільки надають безпосередню допомогу, але й беруть активну участь у формуванні суспільної думки та поширенні демократичних цінностей [5]. Волонтерство сприяє згуртуванню спільноти, підвищує соціальну активність та допомагає людям відчути свою значущість у вирішенні суспільно важливих проблем.

1.2 Аналіз аналогів

1.2.1 Гуманітарна платформа «Є Допомога»

Платформа "Є Допомога" була створена Міністерством соціальної політики України за підтримки Міністерства цифрової трансформації України та Програми розвитку ООН з фінансовою підтримкою Швеції [5]. Ця ініціатива має на меті задовольняти нагальні потреби громадян, які зазнали впливу від військової агресії або були змушені змінити місце проживання.

"Є Допомога" дозволяє користувачам з будь-якої країни світу допомагати

онлайн, завдяки простій реєстрації на платформі. Особи, які потребують допомоги, можуть подавати заявки, в яких вказують, якого саме типу допомога їм необхідна. Це може бути як матеріальна допомога, так і консультації або інші види підтримки.

Переваги:

- Доступність та універсальність: Платформа доступна користувачам з усього світу, що дозволяє масштабувати допомогу та залучати міжнародну підтримку.
- Простота використання: Простий та зрозумілий інтерфейс забезпечує легке подання заявок на отримання допомоги та реагування на потреби інших.
- Стимулювання волонтерів: Наявність системи нагород мотивує волонтерів активніше брати участь та допомагати іншим.

Недоліки:

- Обмежений функціонал для довгострокового планування та відстеження проектів.
- Оцінка ефективності: Вимірювання реального впливу та ефективності допомоги через платформу може бути складним завданням.
- Обмежені способи волонтерської допомоги

Ця платформа має здатність оперативно мобілізувати ресурси та волонтерів, забезпечуючи ефективну допомогу в критичний час. Втім, її потенціал у довгостроковому плануванні та управлінні волонтерськими проектами залишається обмеженим, що може впливати на сталість волонтерських ініціатив. Приклад роботи з платформою наведено на рисунках 1.2, 1.3.

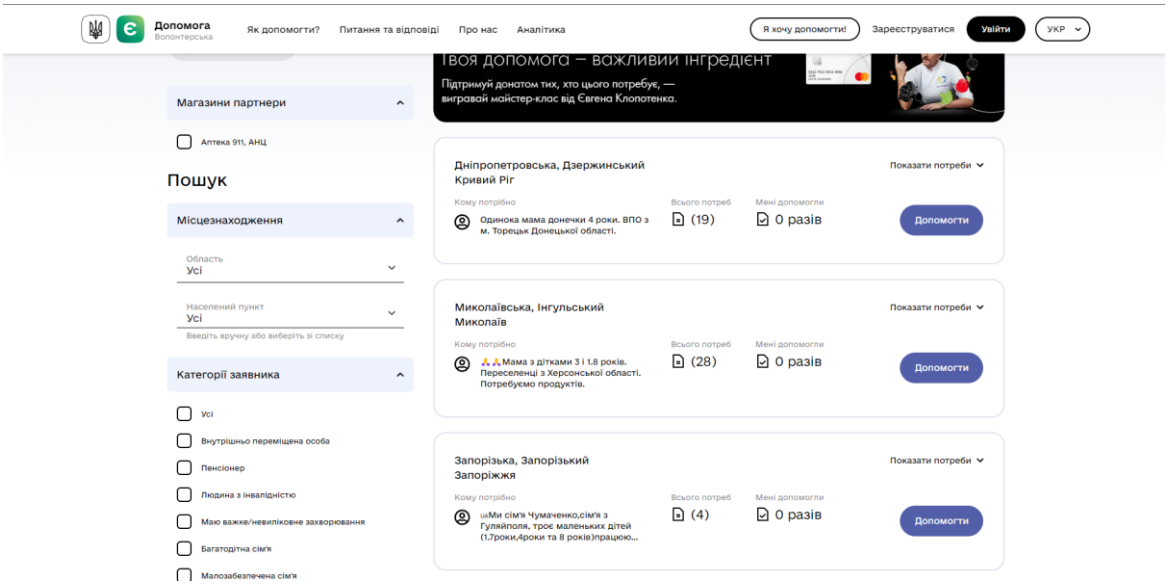


Рис.1.2 Перегляд потреб на головній сторінці платформи Є Допомога

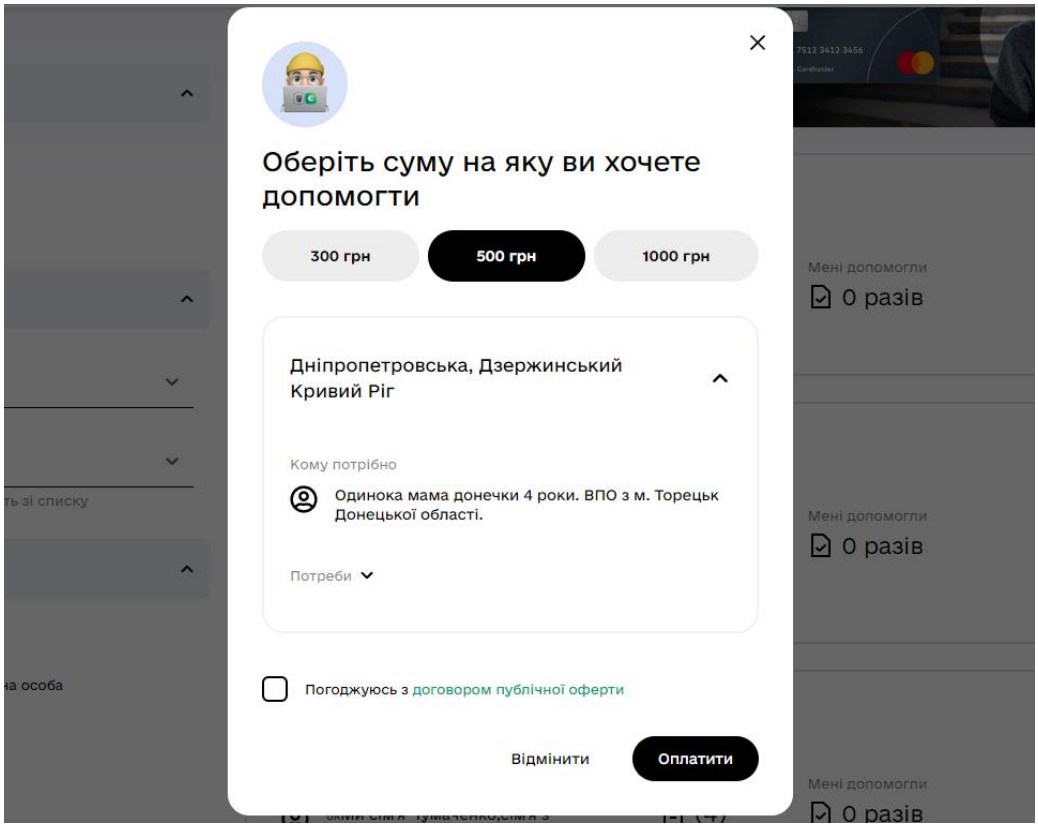


Рис.1.3 Надання допомоги на потребу на платформі Є Допомога

1.2.2 Платформа «dobro.ua»

dobro.ua є однією з найбільших платформою в Україні для збору коштів онлайн на благодійність. Заснована у 2011 році Фондом Віктора Пінчука, платформа виступає як ефективний і сучасний інструмент для залучення фінансування на благодійні та соціальні проекти будь-якого напрямку [6].

Основною місією dobro.ua є створення нової системної моделі благодійності на засадах прозорості, відповідальності та залучення громадян. Платформа зібрала понад 720 мільйонів гривень на благодійність, підтримавши більше 8,000 проектів, і стала лідером збору коштів в Україні.

dobro.ua пропонує допомогу через різноманітні проекти, які включають медицину, соціальну допомогу, освіту, екологію та багато інших сфер. Фінансування платформи здійснюється завдяки грантам від Фонду Віктора Пінчука та додатковій добровільній підтримці філантропів, забезпечуючи, що 100% зібраних коштів йдуть на потрібні проекти.

Платформа також забезпечує простий спосіб підтримки через пожертви одноразові або регулярні, і використовує інноваційні засоби для залучення коштів, такі як SMS-благодійність та криптовалюти.

Переваги:

- Прозорість і відповідальність: dobro.ua встановлює високі стандарти прозорості, демонструючи, де й як використовуються зібрані кошти. Це збільшує довіру донорів.
- Велика кількість проектів: Платформа підтримала понад 8,000 різноманітних проектів, охоплюючи широкий спектр потреб – від медичної допомоги до підтримки освіти і культури.
- Широке використання технологій: Платформа використовує сучасні технологічні засоби для збору коштів, включаючи SMS-благодійність та

криптовалюти, що робить процес пожертвування доступнішим і зручнішим для широкої аудиторії.

- Сильна підтримка з боку великих організацій: Фінансування та підтримка від донорів дозволяють платформі продовжувати свою роботу і розвиватися.

Недоліки:

- Залежність від донорів: Фінансування платформи в значній мірі залежить від грантів та пожертв від фондів і приватних донорів, що може становити ризик стабільності у разі зміни їхніх пріоритетів або фінансових обставин.
- Потенційна перенасиченість проектами: З великою кількістю проектів на платформі, може бути складно забезпечити достатню увагу та ресурси для кожного з них, що може призвести до недофінансування деяких ініціатив.
- Обмеженість географічного охоплення: Хоча платформа і відкрита для міжнародних донорів, її основний фокус та основна аудиторія зосереджені в Україні, що може обмежувати її можливості залучення коштів на глобальному рівні.

Приклад роботи з платформою dobro.ua наведено на рисунках 1.4, 1.5.

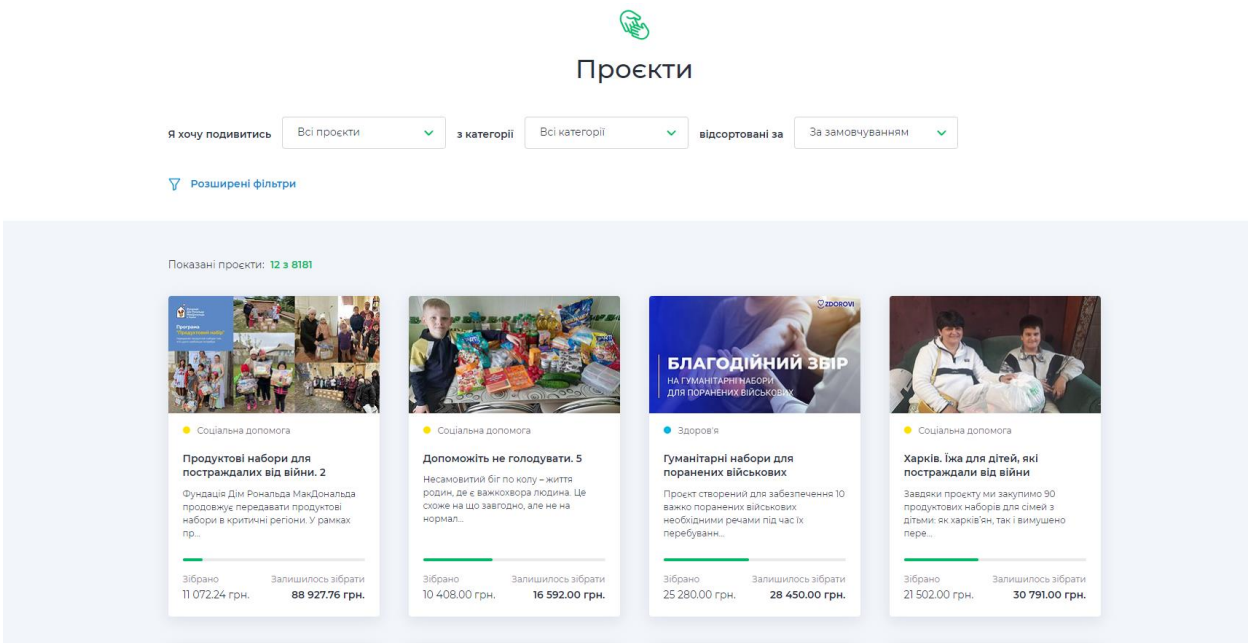


Рис.1.4 Перегляд проектів на платформі dobro.ua

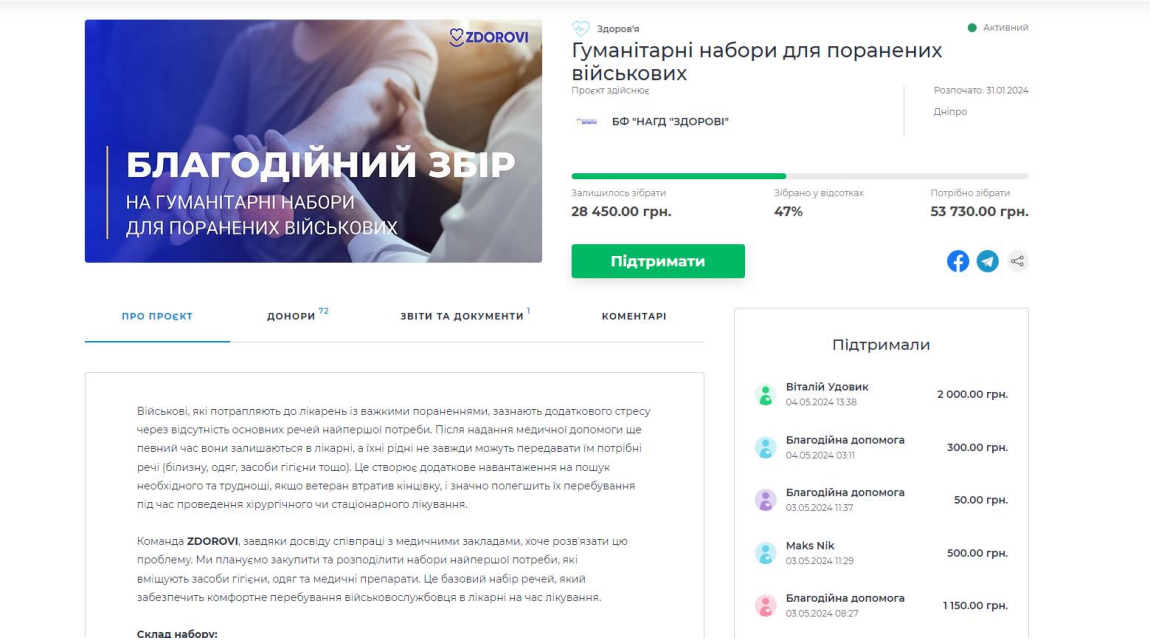


Рис.1.5 Перегляд інформації про проект на платформі dobro.ua

1.2.3 Платформа «GivePulse»

GivePulse — це американська платформа для управління волонтерами, збору коштів та організації заходів, яка спрямована на залучення громади до активної участі у соціальних ініціативах [7]. Основна місія платформи полягає в тому, щоб дозволити кожному знаходити, організовувати і оцінювати соціальний вплив ініціатив у своїй громаді.

Платформа активно використовується вищими навчальними закладами, некомерційними організаціями, муніципалітетами та компаніями для управління корпоративною соціальною відповідальністю і волонтерськими програмами. Платформа забезпечує простий і ефективний доступ до інструментів, необхідних для глибокого залучення волонтерів і реалізації соціальних проектів на місцевому рівні

Переваги:

- Всеохоплююча система: GivePulse об'єднує управління волонтерами, збір коштів, та організацію заходів у єдиній платформі, що спрощує координацію та управління.
- Мобільний застосунок: Наявність мобільного застосунку дозволяє волонтерам легко реєструватися та відстежувати свою участь через мобільні пристрої.
- Підтримка спільноти: GivePulse підтримує широкий спектр спільнот, включаючи університети, некомерційні організації, урядові установи та бізнеси.

Недоліки:

- Комплексність: Через багатофункціональність платформи нові користувачі можуть відчувати складності з орієнтацією та використанням всіх наявних інструментів.
- Обмежена персоналізація: Певні аспекти платформи можуть не повністю відповідати специфічним потребам окремих організацій або індивідуальних проектів.
- Інтеграція з іншими інструментами: Хоча платформа інтегрується з деякими зовнішніми системами, можуть бути труднощі з інтеграцією із специфічними, нестандартними або застарілими системами.

Незважаючи на високий функціонал, GivePulse може виявитись складною для простих волонтерських проектів та вимагає додаткових зусиль для освоєння всіх її можливостей, особливо для не-англомовних користувачів. Приклад роботи з платформою dobro.ua наведено на рисунках 1.6, 1.7.

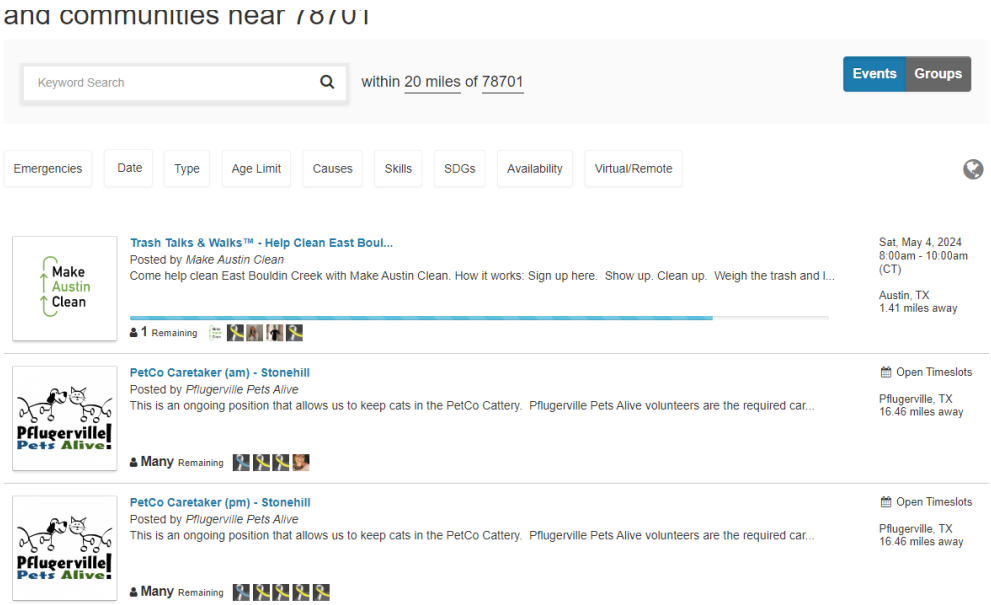


Рис.3.5 Перегляд волонтерських можливостей на платформі GivePulse

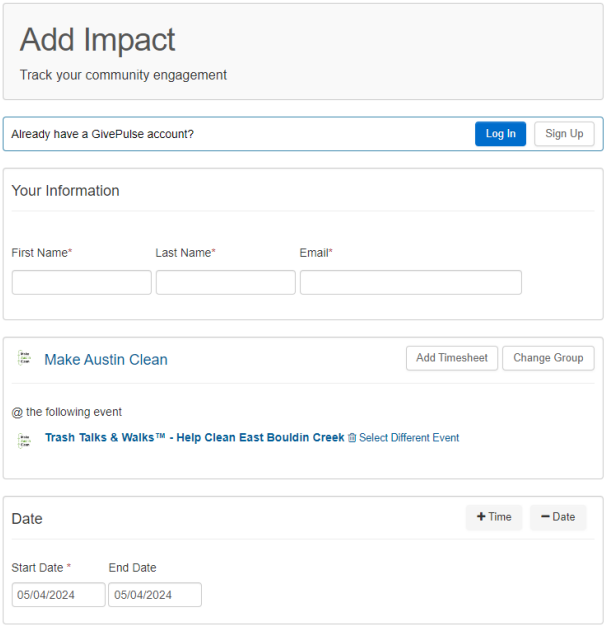


Рис.3.6 Долучення до волонтерських можливостей на платформі GivePulse

Зведені результати аналізу характеристик розглянутих застосунків наведено у таблиці 1.1.

Цей аналіз виявляє, що "GivePulse" має значно ширші можливості щодо технологічних платформ та інтеграції, а також забезпечує більш гнучку підтримку для користувачів, що може бути особливо важливим для масштабних або глобальних ініціатив. "Є Допомога" та "dobro.ua" зосереджені на українському

контексті та мають обмеження у плані технологічної адаптації підтримки порівняно з "GivePulse".

Таблиця 1.1

Зведені результати аналізу характеристик застосунків для волонтерської взаємодії

Показник	Є Допомога	dobro.ua	GivePulse
Платформи	Web	Web	Android, iOS, Web
Функціональні можливості	Онлайн допомога, матеріальна підтримка	Волонтерство, збір коштів, освіта	Управління волонтерами, навчальні курси, збір коштів
Сектори діяльності	Соціальна допомога	Широкий спектр соціальних ініціатив	Екологія, освіта, здоров'я, культура
Технологічні особливості	Проста реєстрація і використання	Низькі транзакційні витрати, ефективний збір коштів	Мобільний застосунок, інтеграція з системами управління навчанням
Географічне покриття	Україна	Україна	Міжнародне покриття
Особливості взаємодії	Підтримка від державних структур	Співпраця з великими компаніями та фондами	Взаємодія з університетами, бізнесами, громадськими організаціями

1.3 Формулювання вимог до платформи

Створення вимог до програмного забезпечення включає повну специфікацію і опис вимог до програмного забезпечення які повинні бути виконані для ефективного створення програмної системи. Ці вимоги охоплюють як функціональні, так і нефункціональні аспекти, що залежить від характеру кожної окремої вимоги [8]. Основна мета полягає у створенні зручного, надійного та функціонального ресурсу для волонтерів та організацій, що потребують допомоги.

Функціональні вимоги:

- Реєстрація і авторизація користувачів: необхідно забезпечити наявність безпечної і зрозумілої системи реєстрації та авторизації користувачів.
- Перегляд та редагування профілю користувача: користувачі повинні мати можливість легко оновлювати свої особисті дані, включаючи контактну інформацію та інтереси.
- Пошук та розміщення волонтерських можливостей і потреб: система має надавати гнучкі інструменти для пошуку та публікації інформації про волонтерські можливості та потреби, з можливістю фільтрації за різними критеріями.
- Система відгуків та рейтингу: важливо впровадити механізм відгуків та рейтингів, що дозволить забезпечити відкритість і прозорість діяльності на платформі.
- Карта потреб: інтеграція інтерактивної карти для візуалізації географічного розташування волонтерських активностей та потреб.
- Створення та перегляд новин та подій, їх обговорення: платформа має дозволяти користувачам публікувати, переглядати новини та події та брати участь в обговореннях.

Нефункціональні вимоги:

- Висока швидкість реакції і обробки запитів: система має бути оптимізована для швидкої відповіді на користувацькі запити та ефективної обробки даних.
- Забезпечення стабільної та безперебійної роботи системи: необхідно впровадити рішення, що забезпечують високу доступнність платформи.
- Адаптація під різні розміри екранів: платформа повинна бути адаптивною, тобто коректно відображатися на різних типах пристроїв, включаючи настільні комп'ютери, планшети та смартфони.

Ці вимоги визначають загальний напрямок для розробки волонтерської платформи, спрямованої на ефективне залучення та підтримку волонтерів і тих, хто потребує допомоги.

2 АНАЛІЗ ТЕХНОЛОГІЙ РОЗРОБЛЕННЯ

2.1 Засоби розроблення front-end частини застосунку

Фронтенд – це та частина веб-застосунків, яка безпосередньо доступна користувачам. Вона включає усі елементи інтерфейсу, які можна бачити і з якими можна взаємодіяти на веб-сайті. В основу фронтенду входить відображення інтерфейсу та обробка користувацьких запитів на стороні клієнта, забезпечуючи функціональність, яка видима при відкритті веб-сторінки.

Загалом, веб-застосунок – це клієнт-серверна архітектура, де браузер виступає як клієнт, а веб-сервер – як сервер, що забезпечує логіку застосунку і зберігання даних. Фронтенд-розробка полягає у створенні та налаштуванні публічної частини застосунку, де кожен елемент, включно з кнопками і текстом, виконує свої задумані функції.

Основні компоненти фронтенд-розробки:

- HTML (HyperText Markup Language): Мова розмітки, яка структурує весь контент на сторінці [9].
- CSS (Cascading Style Sheets): Мова стилізації, що визначає вигляд елементів на веб-сторінці [10].
- JavaScript: Мова програмування, яка додає інтерактивність веб-сторінкам, виконуючи скрипти на клієнтській стороні для обробки подій, таких як кліки мишкою або натискання клавіш [11].

Проте сьогодні фронтенд розробка не обмежується лише цими базовими компонентами. У великих компаніях та на масштабних проектах розробка фронтенду зазвичай значно складніша та комплексна. Тому постає потреба у використанні відповідних технологій які б оптимізували та спростили розробку а також розширили функціонал. Саме тому сучасні веб-розробники вдаються до використання різноманітних фреймворків і бібліотек, які допомагають підвищити

ефективність розробки, забезпечити більшу модульність коду, і покращити користувацький досвід.

JavaScript-бібліотеки та фреймворки стали критично важливим елементом у створенні кожного сучасного веб-застосунку. З огляду на широкий спектр доступних пропозицій, розробники часто стикаються з викликом вибору найбільш підходящих інструментів для своїх проектів. Вибір правильного інструментарію може значно вплинути на ефективність проекту, його масштабування та легкість підтримки в майбутньому.

Ключ до вибору правильної бібліотеки або фреймворку полягає у зрозумінні їх основних відмінностей та переваг. Наприклад, React, розроблений Facebook, забезпечує гнучку розробку компонент-орієнтованих інтерфейсів; Angular від Google пропонує широкий функціонал для розробки більш масштабних застосунків з потужною структурою; Vue.js славиться своєю простотою та легкістю інтеграції, тоді як Next.js, що також базується на React, вирізняється можливостями серверного рендерингу, що забезпечує покращення продуктивності та оптимізацію SEO [12].

Для вибору підходящого фреймворка потрібно провести детальний аналіз з порівнянням ключових характеристик та функцій. Для порівняння використано такі характеристики:

1. Virtual DOM (Віртуальний DOM) - Механізм, що дозволяє фреймворку вносити оптимізовані зміни в DOM (структуру веб-сторінки), що сприяє підвищенню швидкості та ефективності рендерингу інтерфейсу без необхідності перезавантаження повної сторінки.

2. Two-Way Data Binding (Двостороннє зв'язування даних) - Функція, що дозволяє синхронізувати поля форми та інші елементи інтерфейсу з моделлю даних. Це означає, що зміни в DOM автоматично оновлюють дані в моделі і навпаки.

3. Component-Based Architecture (Компонентна архітектура) - Підхід до розробки, у якому інтерфейс користувача будується з використанням відокремлених, повторно використовуваних компонентів. Кожен компонент інкапсулює свою власну логіку та стилі.

4. Розмір - Вказує на загальний розмір бібліотеки або фреймворку, який може впливати на час завантаження сторінки та загальну продуктивність застосунку.

5. Server-Side Rendering (Рендеринг на стороні сервера) - Техніка, що дозволяє генерувати HTML на сервері перед тим, як він буде відправлений до браузера. Це може покращити час завантаження сторінок і підвищити їх доступність для пошукових систем.

6. Routing (Маршрутизація) - Функціональність, яка дозволяє управляти навігацією між різними сторінками у застосунку, реагуючи на зміни в URL браузера.

7. Popular Use Cases (Популярні сценарії використання) - Типові приклади того, як фреймворки можуть бути використані в реальних проектах [13].

У таблиці 2.1 наведена порівняльна таблиця найпопулярніших JavaScript фреймворків.

Таблиця 2.1

Порівняльна таблиця найпопулярніших JavaScript фреймворків

Feature	**React**	**Angular**	**Vue**	**Next.js**
Virtual DOM	+	-	+	+
Two-Way Data Binding	-	+	+	-
Component-Based Architecture	+	+	+	+
Розмір	Малий (43.9 KB)	Великий (2.1 MB)	Малий (80 KB)	Малий (79 KB)
Server-Side Rendering	Не має вбудованої підтримки	Підтримує SSR через Angular Universal	Можливий але вимагає додаткової конфігурації.	Є підтримка SSR "з коробки",
Routing	Можлива за рахунок бібліотеки React Router.	Вбудована підтримка (Angular Router)	Вбудована підтримка (Vue Router)	Вбудований файловий роутинг (File-based Routing)
Popular Use Cases	Facebook, Instagram, Airbnb, Dropbox	Google, Microsoft, IBM, Cisco	Alibaba Group, Xiaomi, Baidu	Uber, Twitch, Netflix, Hulu

З проведеного порівняння можна зробити висновок, що кожен з порівнюваних популярних JavaScript фреймворків має унікальні особливості та оптимальні сценарії використання, які слід враховувати при виборі платформи для розробки веб-застосунків. React і Next.js вирізняються своєю гнучкістю і широкими можливостями для рендерингу на стороні клієнта та сервера, роблячи їх ідеальними для розробки інтерактивних і динамічних інтерфейсів [14-15]. Angular пропонує великий набір вбудованих функцій та сильно структурований підхід, ідеально підходящий для великих корпоративних застосунків [16]. Vue, з іншого боку, є відмінним вибором для проектів, що потребують швидкого розвитку та легкої інтеграції завдяки своїй простоті та гнучкості [17]. Залежно від потреб проекту, технічних вимог та ресурсів команди, кожен фреймворк може бути оптимальним рішенням для певних задач розробки.

Для вибору засобів розробки фронтенд частини застосунку також грає важливу роль його актуальність. Щоб визначити це, можна провести порівняння реакцій спільноти розробників на фреймворки, оскільки це є важливим для розуміння тенденцій веб-розробки та вибору технологій які найкраще відповідають потребам проекту. А також з наявністю великої бази інших розробників які використовують технологію спрощується процес навчання та вирішення комплексних завдань.

1. React:

    - React продемонстрував значний ріст у використанні та популярності з 2016 по 2022 рік. Частка тих, хто вказав, що використовуватиме React знову, стабільно зросла з приблизно 49% у 2016 році до майже 68% у 2022 році.
    - Інтерес до React також збільшувався, а частка респондентів, які ніколи не чули про React, зменшилася з 11.6% у 2016 році до приблизно 9% у 2022 році.

На рисунку 2.1 показано результати опитування користувачів Developer Survey від Stack Overflow щодо використання React [16].

Рис. 2.1 Результати опитування користувачів щодо використання React

2. Angular:

   - Angular показав деяке зростання відсотка тих, хто вказав, що використовуватиме цей фреймворк знову, з 13.6% у 2016 році до 20.8% у 2022 році. Однак, цей ріст не є таким стрімким, як у React.
   - Проте, велика частина респондентів залишається зацікавленою в Angular, із стабільним зниженням кількості людей, які ніколи не чули про цей фреймворк.

На рисунку 2.2 показано результати опитування користувачів Developer Survey від Stack Overflow щодо використання Angular [16].

Рис. 2.2 Результати опитування користувачів щодо використання Angular

3. Next.js:

- За період з 2018 по 2022 рік, Next.js показав динамічне зростання зацікавленості та використання. Відсоток респондентів, які заявили, що використовуватимуть Next.js знову, зріс з 8.7% у 2018 році до 43.7% у 2022 році.
- Цей фреймворк також зазнав значного зниження відсотка тих, хто ніколи про нього не чув, що свідчить про збільшення його популярності та прийняття в розробницькій спільноті.

На рисунку 2.3 показано результати опитування користувачів Developer Survey від Stack Overflow щодо використання Next.js [17].

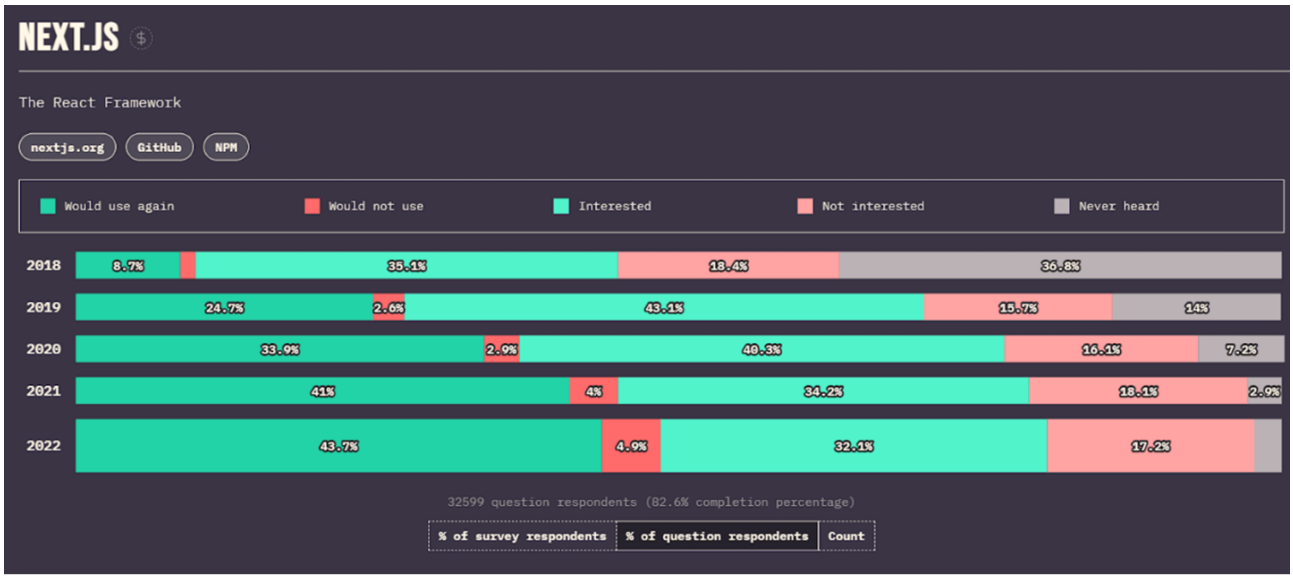


Рис. 2.3 Результати опитування користувачів щодо використання Next.js

Згідно з даними опитувань можна прийти висновку, що React продовжує набирати популярність, що підтверджується стабільним зростанням відсотка розробників, які вказують, що використовуватимуть його знову. Angular має стабільний інтерес серед розробників, але не показує значного зростання популярності. Next.js, що базується на React, показав стрімкий ріст використання та інтересу, особливо у останні роки.

На основі аналізу характеристик і популярності було вирішено використовувати Next.js у роботі. Основними факторами, що вплинули на це рішення, стали:

- Вбудована підтримка Server-Side Rendering (SSR), яка значно покращує індексацію в пошукових системах і загальну швидкість завантаження сторінок.
- Простота розгортання і оптимізація продуктивності, завдяки чому Next.js стає відмінним вибором для проектів, які потребують швидкої відповіді та оптимальної працездатності.
- Ростуча спільнота і екосистема, що забезпечує велику кількість ресурсів для навчання і підтримки, а також постійне оновлення і поліпшення застосунку.

Вибір Next.js для розробки дозволяє не тільки скористатися перевагами React, але й забезпечити додаткові можливості для оптимізації продуктивності та швидкості, важливі для сучасних веб-застосунків. На підставі аналізу популярності та функціональних переваг, Next.js було обрано як оптимальний фреймворк для розробки цього проекту.

2.2 Засоби розроблення back-end частини застосунку

Бекенд - це розробка логіки на стороні сервера, яка забезпечує роботу веб-сайтів і застосунків "за лаштунками". Він включає весь код, необхідний для створення бази даних, сервера та програми. Від міграції баз даних до інтеграції API та налаштування серверних технологій, які забезпечують роботу веб-сайту.

Традиційні back-end рішення

Розроблення back-end традиційними рішеннями (такими як Node.js з Express, Ruby on Rails, Django тощо), можемо виділити наступні аспекти:

1. Архітектура: Традиційні back-end рішення зазвичай вимагають окремого налаштування для сервера і можуть включати більш складні архітектурні конфігурації. Вони забезпечують більшу гнучкість для великих застосунків, які потребують розгалуженої логіки обробки даних, безпеки і інтеграції з різними базами даних і зовнішніми сервісами.
2. Розширеність: Великі back-end системи надають більші можливості для масштабування та оптимізації за допомогою різноманітних інструментів та підходів, таких як мікросервіси, контейнеризація тощо.
3. Спеціалізація: Традиційні back-end платформи часто пропонують спеціалізовані рішення для певних типів застосунків, такі як електронна комерція, CMS, CRM системи, що можуть бути краще оптимізовані під конкретні потреби бізнесу.

Full stack рішення

Next.js традиційно відомий своєю роботою на front-end, проте останнім часом набирає популярності і поширеності види застосунків які використовують один фреймворк як для фронтенду так і бекенду оскільки з’являються нові технології і ресурси для цього. Розглянемо основні аспекти створення бекенду в Next.js:

1. API маршрути: API маршрути в Next.js дозволяють створювати бекенд ендпоінти без необхідності налаштування окремого сервера. Кожен файл у директорії pages/api автоматично стає доступним як API маршрут. Наприклад, файл pages/api/user.js буде доступний за адресою /api/user.

2. Обробка запитів: Next.js API маршрути обробляють HTTP-запити (GET, POST, PUT, DELETE тощо) за допомогою стандартного об’єкта req (request) і res (response). Це забезпечує гнучкість у розробці різноманітних бекенд-функцій, таких як аутентифікація, обробка форм, робота з базами даних тощо.

3. Інтеграція з базами даних: для збереження і обробки даних можна використовувати різні бази даних, такі як SQL, NoSQL, або навіть GraphQL. Найчастіше використовуються ORM (Object-Relational Mapping) бібліотеки, такі як Prisma або TypeORM для роботи з SQL базами даних. Наприклад:

- Prisma: ORM для TypeScript і Node.js, який спрощує роботу з базами даних.
- Mongoose: Бібліотека для моделювання об'єктів MongoDB в середовищі Node.js.

4. Аутентифікація та авторизація

Next.js легко інтегрується з різними рішеннями для аутентифікації та авторизації, такими як:

- NextAuth.js: Потужна бібліотека для аутентифікації в Next.js з підтримкою різних провайдерів (Google, GitHub, Facebook тощо).
- Auth0: Хмарне рішення для управління ідентифікацією і доступом.

В результаті порівняння було зроблено вибір на користь Next.js для написання бекенду.

Розглянемо основні причини:

1. Уніфікація фронтенду і бекенду

Next.js дозволяє розробляти фронтенд і бекенд в одній кодовій базі, що спрощує розробку, підтримку та розгортання застосунку. Це знижує витрати на управління окремими проектами та забезпечує кращу інтеграцію між клієнтською та серверною частинами.

2. Простота і зручність

Next.js має інтуїтивно зрозумілий API для створення API маршрутів. Це дозволяє швидко і легко додавати нові серверні функції, не вимагаючи значних зусиль для налаштування середовища. Така простота є великим плюсом для команд розробників різного рівня досвіду.

4. Інтеграція з базами даних і зовнішніми API

Next.js легко інтегрується з різними базами даних (SQL, NoSQL) та зовнішніми API. Це дозволяє розробникам швидко підключатися до необхідних джерел даних і забезпечувати їхнє ефективне використання у застосунку.

5. Підтримка TypeScript

Next.js має вбудовану підтримку TypeScript, що підвищує безпеку коду і полегшує його підтримку. TypeScript допомагає запобігти багатьом помилкам ще на етапі написання коду, що особливо важливо для великих проектів.

На основі порівняльного аналізу, для розробки бекенду було вирішено вибрати Next.js як повноцінне Full Stack рішення. Основні фактори, що вплинули на це рішення, включають уніфікацію розробки фронтенду і бекенду в одній кодовій базі, що забезпечує зниження витрат на управління проектами та покращує інтеграцію між серверною та клієнтською частинами. Також Next.js пропонує простоту в створенні API маршрутів, інтеграцію з різноманітними базами даних та зовнішніми API, а також вбудовану підтримку TypeScript для забезпечення більшої безпеки та легшої підтримки коду.

2.3 Засоби розроблення бази даних

Вибір системи бази даних є дуже важливим рішенням яке впливає на продуктивність, масштабованість та загальну успішність проекту. Вибір залежить

від багатьох факторів, включаючи тип даних, обсяги даних, вимоги до продуктивності та специфічні функціональні потреби.

Реляційні бази даних (SQL): Підходять для структурованих даних з чіткими зв'язками між таблицями. Використовуються для складних транзакцій і забезпечують дотримання вимог ACID. Приклади: MySQL, PostgreSQL, Oracle, Microsoft SQL Server.

Нереляційні бази даних (NoSQL): Краще підходять для роботи з великими обсягами неструктурованих або слабоструктурованих даних, таких як документи, графи, ключ-значення та колоночні дані. Приклади: MongoDB (документо-орієнтована), Cassandra (стовпцева), Redis (ключ-значення), Neo4j (графова).

З огляду потреб та вимог проекту було зроблено вибір на користь нереляційних баз даних.

Зростання обсягів інформації та вимоги до швидкості її обробки призвели до появи NoSQL баз даних, які використовуються для вирішення проблем, що виникають при роботі з реляційними базами даних у веб-застосунках [19].

Причини появи NoSQL:

- Великі обсяги даних - високонавантажені веб-застосунки обробляють дані, що перевищують можливості реляційних баз даних.
- Масштабованість - реляційні СУБД не завжди забезпечують одночасний доступ мільйонам користувачів.
- Обробка нетабличних даних - сучасні вимоги включають роботу з різними типами даних, які не завжди добре підходять до реляційної моделі.

Типи NoSQL баз даних:

1. Сховища "ключ-значення"

- Особливості - дані зберігаються у вигляді пар "ключ-значення". Ця модель є гнучкою та швидкою.
- Приклади - Redis, MemcacheDB, BerkeleyDB.
- Переваги - висока продуктивність, підтримка транзакцій та реплікацій.
- Недоліки - обмежена функціональність у порівнянні з більш складними моделями.

2. Стовпцеві СУБД

- Особливості - дані зберігаються по стовпцях, що дозволяє ефективно обробляти великі обсяги даних.

- Приклади: Cassandra, HBase.
- Переваги - зменшення навантаження на сервер, можливість компресії даних.
- Недоліки - низька швидкість запису при збільшенні об'єму бази даних.

3. Документо-орієнтовані СУБД

- Особливості - дані зберігаються у вигляді документів (XML, JSON, BSON тощо).
- Приклади: MongoDB, CouchDB, IBM Lotus Notes.
- Переваги - гнучкість структури даних, швидкий доступ до документів через унікальні ключі.
- Недоліки - відсутність підтримки транзакцій.

4. Графо-орієнтовані СУБД

- Особливості - дані представлені у вигляді графів з вузлами та зв'язками.
- Приклади: Neo4j, AllegroGraph.
- Переваги - ефективний пошук шляхів, виділення спільнот, гнучка схема даних.
- Недоліки - можуть бути складними для розуміння та використання.

5. Об'єктно-орієнтовані СУБД

- Особливості - бази даних, які працюють з об'єктами так само, як з об'єктами при об'єктно-орієнтованому програмуванні.
- Приклади: ObjectDB, Caché, Jasmine.
- Переваги - природна інтеграція з об'єктно-орієнтованими мовами програмування.
- Недоліки - можуть вимагати значних ресурсів для підтримки.

На рисунку 2.4 представлене дерево прийняття рішення для вибору NoSQL сховища на якому представлені компромісні рішення потенційно придатних систем бази даних. На ньому можна розглянути рішення від простого кешування до обробки Big Data [20]. Вузли дерева показують вибір між швидкими запитами, що використовують системи як Redis, і складнішими запитами, що вимагають розподілених систем як MongoDB.

Для розробки платформи волонтерської взаємодії було обрано базу даних MongoDB [20], використовуючи дерево рішень з наданої схеми, яке допомагає визначити оптимальну базу даних відповідно до специфічних потреб проекту.

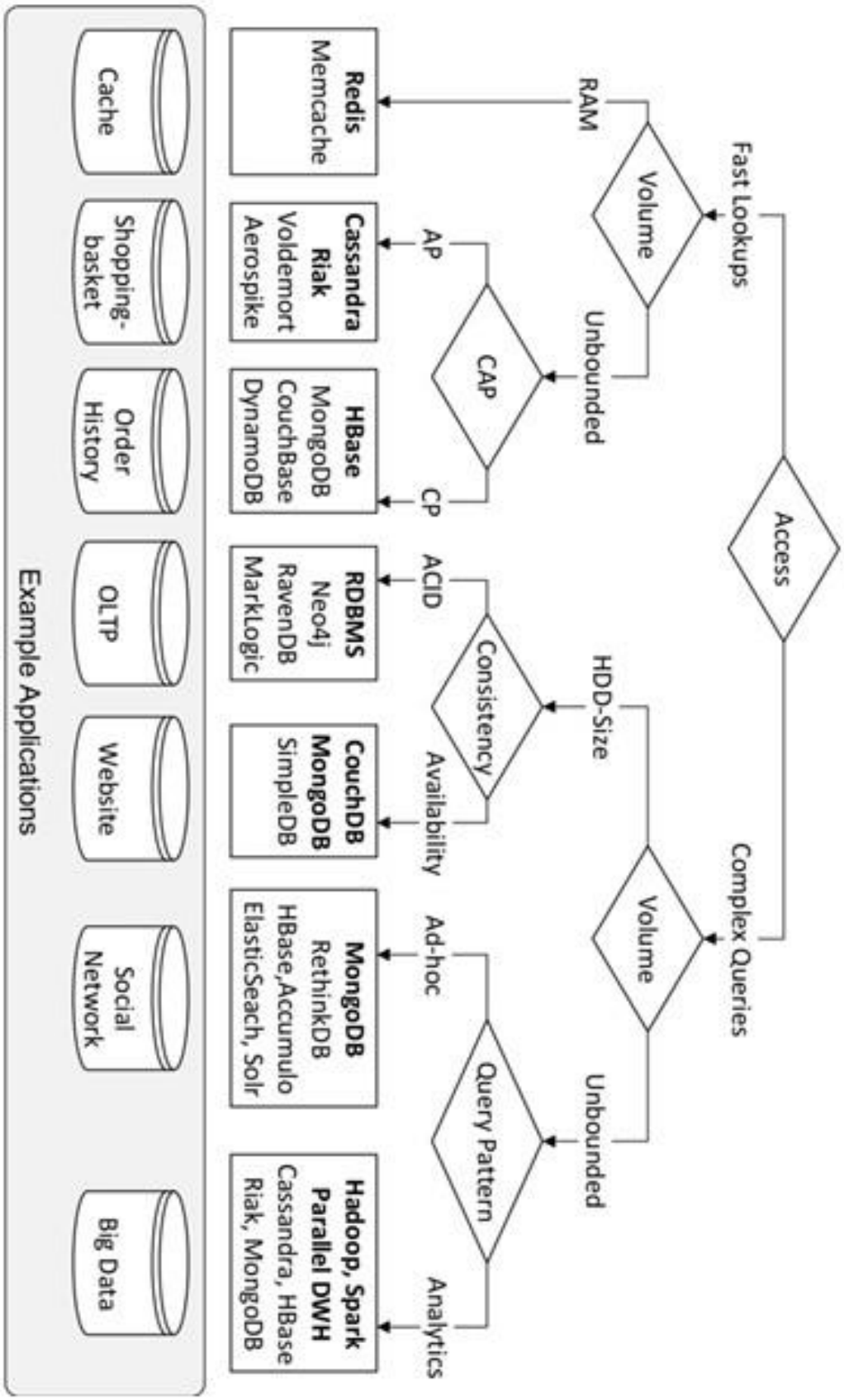


Рис. 2.4 Дерево прийняття рішення для вибору NoSQL сховища

Такий вибір обумовлений кількома перевагами MongoDB, які відповідають вимогам проекту:

Гнучкість схеми. MongoDB не вимагає строгого дотримання схеми даних, що дозволяє легко модифікувати структуру даних без необхідності зупиняти базу даних або вносити зміни в код. Це особливо важливо для платформи волонтерської взаємодії, де типи даних та відносини між ними можуть змінюватися у міру розвитку проекту.

Масштабованість. MongoDB підтримує горизонтальну масштабованість через шардування, що дозволяє ефективно розподіляти дані по кластерах серверів. Це забезпечує здатність обслуговувати великі обсяги даних та високий рівень одночасних запитів, що можуть виникати на платформі з активними волонтерськими проектами. На рисунку 2.5 зображено приклад структури даних в MongoDB.

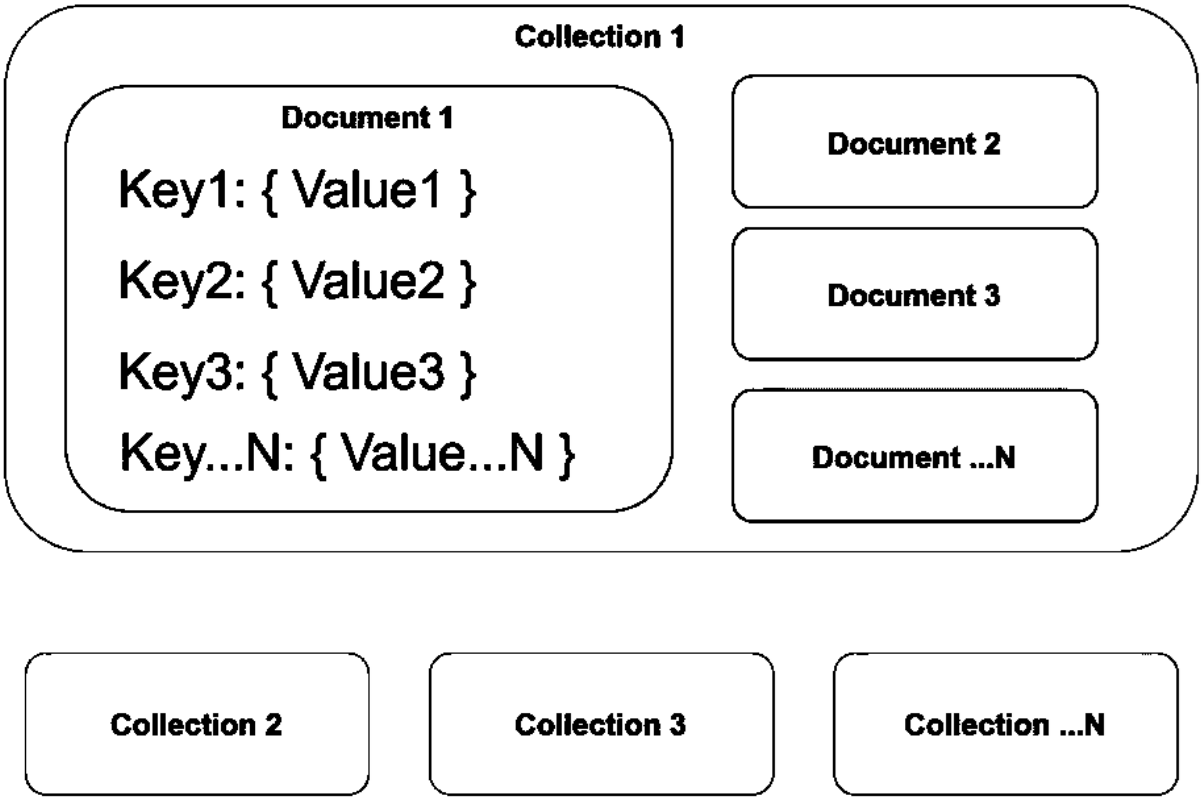


Рис. 2.5 Структура даних MongoDB

Швидкість та продуктивність. MongoDB оптимізована для швидкого зберігання та витягування даних, що забезпечується за рахунок використання документів у форматі BSON (Binary JSON). Це особливо ефективно для оперативної роботи з даними, які мають складну структуру, як-от профілі користувачів, історії дій, звіти про волонтерські заходи та інші.

Робота з багатими запитами. MongoDB підтримує різноманітність запитів включно з текстовим пошуком, геопросторовими запитами та агрегацією, що важливо для платформи, яка може обробляти запити на пошук волонтерських можливостей, фільтрацію за регіонами, аналіз даних волонтерів тощо.

Можливість реплікації. MongoDB має вбудовані можливості реплікації, забезпечуючи високу доступність та стійкість до збоїв. Реплікація даних дозволяє забезпечити постійний доступ до платформи, навіть у випадку відмови одного або кількох серверів.

Простота інтеграції. MongoDB може бути легко інтегрована з більшістю мов програмування та платформ, завдяки широкій підтримці драйверів. Це спрощує розробку та подальшу підтримку платформи.

Вибір MongoDB для розробки платформи волонтерської взаємодії дозволяє не тільки вирішити технічні задачі, але й забезпечити гнучкість, необхідну для адаптації до змінюваних потреб волонтерської громади та організацій.

3 ПРОЕКТУВАННЯ ПЛАТФОРМИ

3.1 Проектування варіантів використання

Варіанти використання описують взаємодію між головним актором, рішенням і вторинними акторами, необхідними для досягнення мети головного актора. Діаграма варіантів використання (Use Case Diagram) є дуже важливим інструментом при проектуванні системи, вона забезпечує візуальне представлення того, як користувачі взаємодіють з системою [21]. Вона слугує планом для розуміння функціональних вимог системи з точки зору користувача, допомагає у спілкуванні між зацікавленими сторонами та керує процесом розробки.

На основі вимог було створено діаграму варіантів використання яку зображено на рисунку 3.1.

Основні ролі користувачів та взаємодії:

1. Незареєстрований користувач - має можливість переглядати доступні можливості та потреби на платформі, але для участі у волонтерських активностях вимагається реєстрація.
2. Бенефіціар (одержувач допомоги) - може створювати заявки на допомогу, переглядати волонтерські можливості, користуватись системою рекомендацій для відбору найбільш підходящих волонтерів.
3. Волонтер - має доступ до створення та реагування на можливості, може вибирати заявки за інтересами або за рекомендаціями системи.

На діаграмі показано наступні ключові випадки використання:

- Реєстрація/Вхід: процес створення облікового запису або входу до системи для отримання повного доступу до функціональності платформи.
- Перегляд запитів на допомогу: можливість для волонтерів переглядати наявні запити на допомогу від бенефіціарів.

- Перегляд волонтерських можливостей: можливість для бенефіціарів переглядати доступні волонтерські можливості.
- Перегляд профілів користувачів: забезпечує можливість перегляду інформації про інших користувачів системи.
- Керування потребами: функціонал для бенефіціарів, який дозволяє створювати, редагувати та видаляти запити на допомогу.
- Керування можливостями: функціонал для волонтерів, що дозволяє створювати, редагувати та видаляти волонтерські можливості.

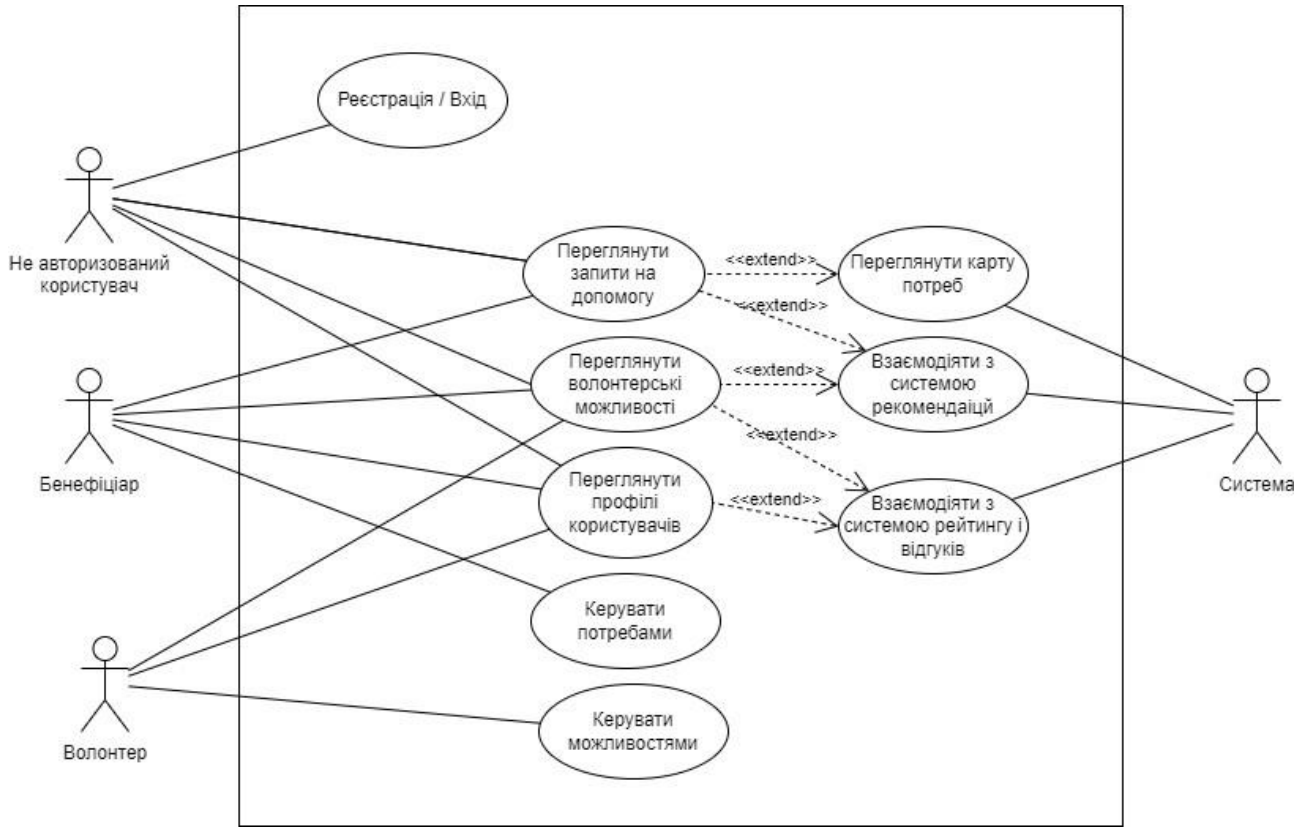


Рис 3.1 Діаграма варіантів використання платформи волонтерської взаємодії

3.2 Моделювання процесів платформи

Процес моделювання діяльності, полягає у створенні діаграми, що представляє динамічні аспекти системи шляхом відображення потоку управління. Це одна з основних технік моделювання UML, яка використовується для детального опису випадків використання або сценаріїв. Основна мета такого моделювання — зосередити увагу на потоках, які керуються внутрішньою обробкою, а не зовнішніми подіями [22].

Основні аспекти, які охоплює моделювання діяльності:

- Фокус на внутрішніх процесах: Моделювання зосереджується на тому, як дії в системі послідовно переходять одна в одну на основі внутрішніх умов або обробок.
- Деталізація випадків використання: Діаграми діяльності можуть використовуватись для ілюстрації конкретного сценарію випадку використання, підкреслюючи як різні дії взаємодіють у межах визначеного процесу.
- Варіативність рівнів моделювання: Можна моделювати діяльності різного рівня складності, від бізнес-рівня до окремих фрагментів програмного коду.

Діаграми діяльності описують, як координуються дії для надання послуги, яка може бути на різних рівнях абстракції. Як правило, подія має бути досягнута за допомогою певних операцій, особливо якщо операція спрямована на досягнення кількох різних цілей, які потребують координації, або як події в одному варіанті використання пов'язані одна з одною, зокрема, у варіантах використання, де дії можуть перетинатися і потребують координації [23].

Діаграми діяльності також допомагають визначити основні кроки в кожному процесі, можливі альтернативні шляхи та умови переходу між різними станами. Це особливо важливо для забезпечення зручності та ефективності використання

платформи, оскільки вони дозволяють врахувати всі можливі сценарії взаємодії користувачів із системою.

На діаграмі діяльності, представленій на рисунку 3.2, зображено детальний процес взаємодії користувачів із платформою для волонтерської діяльності. Діаграма поділена на дві основні гілки, які відповідають двом основним ролям користувачів: волонтер і бенефіціар. Кожна гілка демонструє послідовність дій, починаючи від входу в систему і закінчуючи наданням або отриманням допомоги.

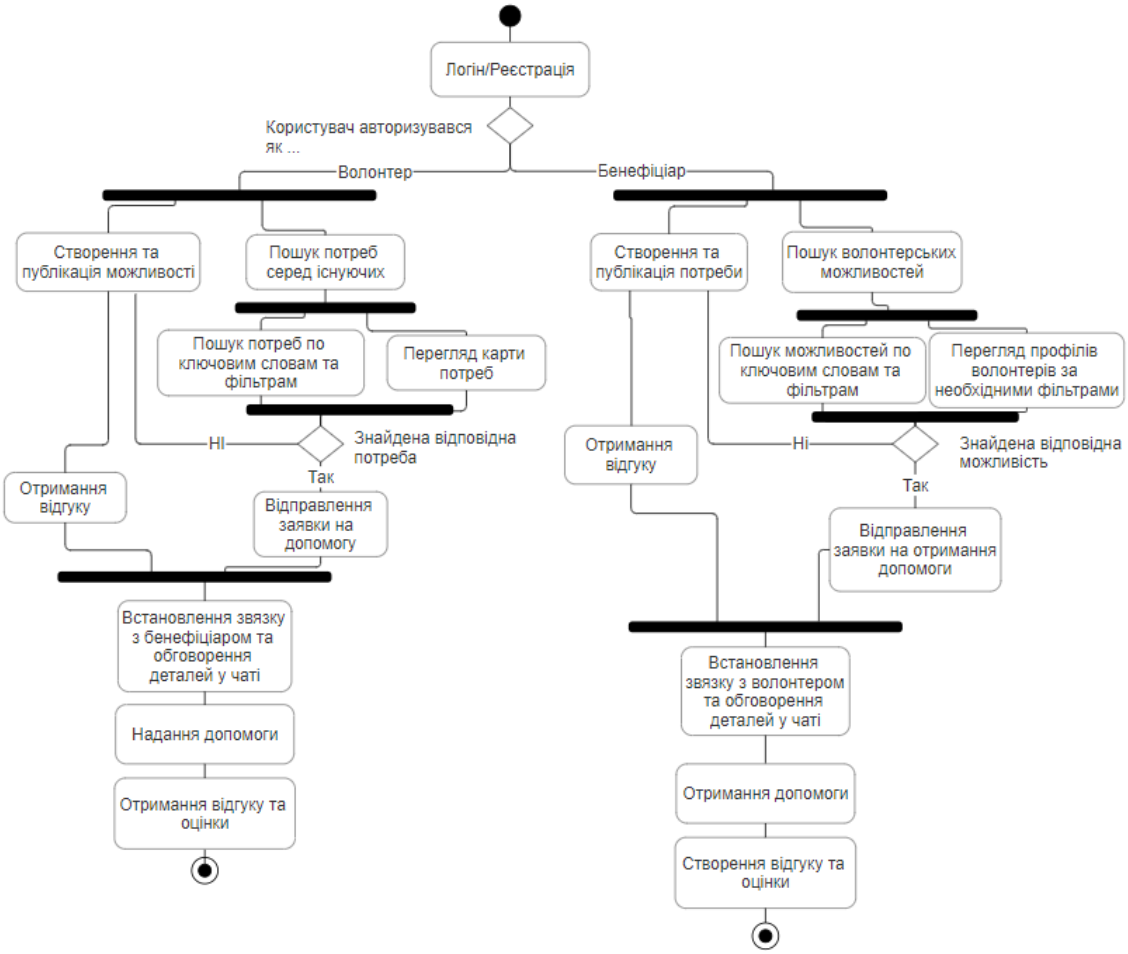


Рис 3.2 Діаграма діяльності платформи волонтерської взаємодії

Процес для волонтерів

1. Логін/Реєстрація: Користувач входить або реєструється в системі.
2. Вибір ролі: Користувач обирає роль волонтера.

3. Створення та публікація можливості: Волонтер створює нову волонтерську можливість і публікує її на платформі.
4. Пошук потреб серед існуючих: Волонтер може шукати потреби за ключовими словами або фільтрами.
5. Перегляд карти потреб: Додаткова можливість переглядати потреби візуально.
6. Відправлення заявки на допомогу: Якщо знайдена відповідна потреба, волонтер відправляє заявку на допомогу.
7. Встановлення зв'язку з бенефіціаром: Волонтер встановлює контакт з бенефіціаром через чат для обговорення деталей.
8. Надання допомоги: Волонтер надає необхідну допомогу.
9. Отримання відгуку та оцінки: Після завершення роботи волонтер отримує відгук від бенефіціара.

Процес для бенефіціарів

1. Логін/Реєстрація: Користувач входить або реєструється в системі.
2. Вибір ролі: Користувач обирає роль бенефіціара.
3. Створення та публікація потреби: Бенефіціар створює новий запит на допомогу і публікує його на платформі.
4. Пошук волонтерських можливостей: Бенефіціар шукає волонтерські можливості за ключовими словами або фільтрами.
5. Перегляд профілів волонтерів: Додаткова можливість переглядати профілі волонтерів для вибору найбільш підходящих.
6. Відправлення заявки на отримання допомоги: Якщо знайдена відповідна можливість, бенефіціар відправляє заявку на отримання допомоги.

7. Встановлення зв'язку з волонтером: Бенефіціар встановлює контакт з волонтером через чат для обговорення деталей.
8. Отримання допомоги: Бенефіціар отримує необхідну допомогу.
9. Створення відгуку та оцінки: Після отримання допомоги бенефіціар залишає відгук і оцінку волонтеру.

3.3 Проектування бази даних

Проектування бази даних для платформи волонтерської взаємодії було виконано з використанням MongoDB, що забезпечує високу гнучкість і швидкість доступу до даних за допомогою своєї нереляційної структури.

Основними одиницями зберігання в MongoDB є документи та колекції. Документи можуть містити різні поля, і не обов'язково всі документи в одній колекції мають однаковий набір полів, що забезпечує гнучкість у зберіганні даних.

Незважаючи на те, що MongoDB є схемо-незалежною базою даних, рекомендується розробити логічну схему даних, щоб забезпечити організацію та консистентність даних. Це може включати визначення необхідних полів, їх типів і взаємозв'язків між документами.

Основні етапи проектування включали:

- Визначення сутностей і їх взаємозв'язків. Було ідентифіковано основні сутності, такі як "Користувач", "Можливості", "Потреби" "Відгуки", "Повідомлення ", та "Чат-кімнати" що відображають ключові аспекти діяльності на платформі.
- Структуризація документів. Кожна сутність була розроблена як окремий документ з визначеними полями, що відповідають атрибутам кожної сутності.

- Візуалізація схеми бази даних. Було створено детальну схему бази даних у форматі ERD (Entity-Relationship Diagram), яка відображає всі сутності, їх поля та зв'язки між ними. Ця схема пердставлена на рисунку 3.3.

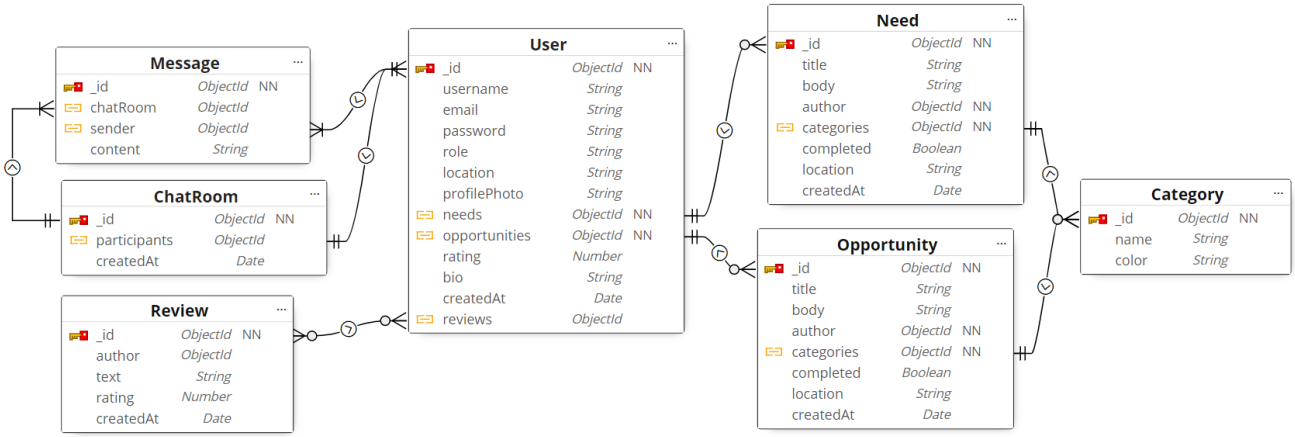


Рис 3.3 Діаграма «сутність — зв'язок»

Цей етап проектування бази даних становить фундамент для наступних стадій розробки системи, забезпечуючи необхідну структуру та організацію даних, яка дозволить ефективно зберігати, обробляти та аналізувати інформацію у межах платформи волонтерської взаємодії.

4 РОЗРОБКА ПЛАТФОРМИ ВОЛОНТЕРСЬКОЇ ВЗАЄМОДІЇ

4.1 Архітектура платформи

Web-застосунки, розроблені з використанням Next.js як повноцінного full-stack рішення, інтегрують фронтенд і бекенд у єдиному застосунку. Next.js дозволяє використовувати React для розробки інтерфейсу користувача та обслуговувати серверні вимоги за допомогою Node.js, що створює суцільне середовище розробки.

Особливості архітектури Next.js у якості full-stack рішення:

1. Інтеграція фронтенду та бекенду: Next.js дозволяє розробляти фронтенд і бекенд у межах одного застосунку, використовуючи одну і ту ж мову програмування (JavaScript), що спрощує взаємодію між різними частинами застосунку.
2. SSR і SSG: Next.js підтримує Server-Side Rendering (SSR) і Static Site Generation (SSG), що дозволяє підвищити швидкість завантаження сторінок і покращити SEO, завдяки попередньому рендерингу сторінок на сервері.
3. API Routes: Next.js має вбудовану підтримку API маршрутів, що дозволяє легко створювати API ендпойнти у тому ж застосунку. Це сприяє легкій інтеграції з базами даних та іншими сервісами.
4. Гнучка файлова система: Структура файлової системи у Next.js автоматично маршрутизує файли на основі їх імен у директорії pages, що спрощує створення нових сторінок і керування маршрутами.
5. Спрощення розробки та масштабування: Завдяки об'єднанню фронтенду і бекенду в одному місці, Next.js знижує складність розробки та деплойменту, роблячи масштабування застосунку більш простим.

Цей підхід уніфікує процес розробки, забезпечуючи високу продуктивність та оптимізацію застосунків.

За допомогою маршрутизатора Next.js було створено структуру сторінок представлену на рисунку 4.1. Замість використання конфігураційних файлів або складних програмних налаштувань, App Router [24] дозволяє розробникам визначати маршрути за допомогою структури файлової системи. Це забезпечує більшу прозорість і легкість в управлінні маршрутами. Цей маршрутизатор підтримує серверні компоненти, що дозволяє серверам обробляти більш складні завдання без додаткового навантаження на клієнта. Це сприяє оптимізації продуктивності та швидкості завантаження сторінок.

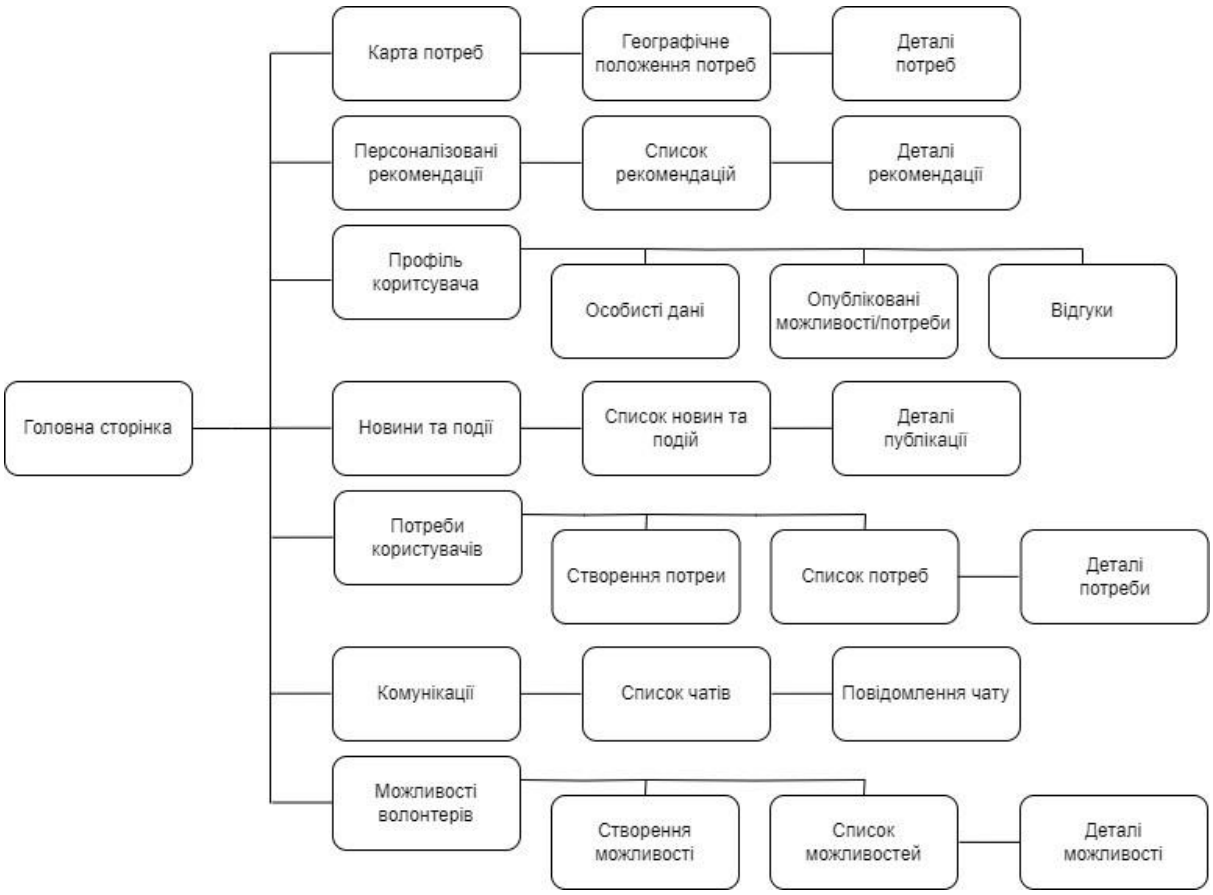


Рис 4.1 Структура сторінок застосунку

App Router легко інтегрується з API маршрутами Next.js, дозволяючи створювати повноцінні RESTful API всередині того ж проекту. За допомогою цього було створено API маршрути включаючи GET, PUT, POST DELETE запити для

сутностей. Для модульності та уникнення повторюваності коду було створено обробники для кожного типу запиту. Вони повторно використовуються при створенні запитів. Приклад створених маршрутів наведено на рисунку 4.2, на рисунку 4.3 наведено приклади обробників.

```
export async function POST(req: NextRequest) {
  return createHandler(req, Need);
}

export async function GET() {
  return getAllHandler(Need);
}
```

Рис 4.2 Маршрути POST і GET запитів потреб.

```
export async function createHandler(req: NextRequest, Model: Model<any>) {
  await dbConnect();
  try {
    const data = await req.json();
    await Model.create(data);
    return NextResponse.json({ message: "Created" }, { status: 200 });
  } catch (err: any) {
    return NextResponse.json({ error: err.message }, { status: 500 });
  }
}

export async function getAllHandler(Model: Model<any>) {
  await dbConnect();
  try {
    const data = await Model.find({});
    return NextResponse.json({ data });
  } catch (err: any) {
    return NextResponse.json({ error: err.message }, { status: 500 });
  }
}
```

Рис 4.3 Обробники маршрутів POST і GET запитів.

4.2 Програмна реалізація

Разом з Next.js для створення інтерфейсу використовувались бібліотеки Styled Components та NextUI.

Styled Components є бібліотекою для React (та Next.js), яка дозволяє використовувати стилізовані компоненти в JavaScript. Це означає, що стилі пишуться безпосередньо у JavaScript-файлах за допомогою шаблонних рядків, що дозволяє легко маніпулювати стилями на основі пропсів або глобальних тем.

Особливості використання Styled Components в розробці:

- Ізоляція стилів: Кожен компонент має унікальні стилі, що запобігає конфліктам і сприяє легшій підтримці коду.
- Динамічне стилізування: Можливість легко змінювати стилі на основі пропсів компонентів, що робить компоненти надзвичайно гнучкими.
- Тематизація: Підтримка тем дозволяє визначати кольори, шрифти та інші стилістики на рівні всього застосунку, забезпечуючи консистентність дизайну.

На рисунку 4.4 представлено приклад використання Styled Components для стилізації компонентів з використанням властивостей та глобальних змінних.

```
export const LinkContainer = styled.div`
  color: var(--base-accent);
  font-size: 1rem;
  font-weight: 600;
  text-align: center;
`;
export const LoaderContainer = styled.div<{ $isFullscreen: boolean }>`
  width: 100%;
  height: ${(props) => (props.$isFullscreen ? "50vh" : "100%")};
  display: grid;
  align-content: center;
`;
```

Рис 4.4 Приклад використання бібліотеки Styled Components.

NextUI — це бібліотека компонентів UI для React, яка забезпечує сучасний, красивий дизайн з набором готових до використання компонентів. Це ідеальний вибір для розробників, які хочуть швидко створити візуально привабливі інтерфейси без необхідності дизайнувати кожен елемент вручну.

Особливості використання NextUI в розробці:

- Готові компоненти: Набір включає кнопки, перемикачі, модальні вікна, карти та інші елементи, які легко інтегруються у проекти.
- Інтеграція з Next.js: NextUI спеціально оптимізована для роботи з Next.js, що забезпечує швидку інтеграцію та високу продуктивність.
- Адаптивність та реагування: Компоненти автоматично адаптуються до різних розмірів екранів, роблячи застосунок доступним на всіх пристроях.

Таким чином, Styled Components використовується для більш детальної настройки окремих компонентів, дозволяючи динамічно змінювати стилі залежно від стану або пропсів. NextUI надає швидкість і ефективність через готові до вживання компоненти з високою продуктивністю, що робить їх ідеальними для створення відповідних секцій, які вимагають більш стандартного підходу до дизайну. Використання обох цих інструментів у поєднанні дозволяє забезпечити не тільки візуальну привабливість, але й функціональну адаптивність та доступність веб-застосунку.

На рисунку 4.5 відображено як використовуються готові компоненти NextUI разом з Styled Components.

```
import {
  NewsItemContainer,
  NewsItemDate,
  NewsItemDescription,
  NewsItemTitle,
} from "@/styles/NewsStyles";
import { Card, Image, Link } from "@nextui-org/react";

const NewsItem = ({
  date,
  title,
  description,
  imageUrl,
}: NewsItemProps): React.ReactElement => {
  return (
    <Card shadow="sm" className="flex-1">
      <NewsItemContainer>
        <Image
          alt="Card background"
          className="object-cover rounded-xl"
          isZoomed
          src={imageUrl}
        />
        <NewsItemDate>{date}</NewsItemDate>
        <NewsItemTitle>{title}</NewsItemTitle>
        <NewsItemDescription>{description}</NewsItemDescription>
        <Link isExternal showAnchorIcon href="">
          Дізнатись більше
        </Link>
      </NewsItemContainer>
    </Card>
  );
};
```

Рис 4.5 Приклад використання бібліотеки NextUI разом з Styled Components.

Для реалізації баз даних у проекті використовувалась бібліотека Mongoose, яка є популярним ODM (Object Data Modeling) інструментом для MongoDB і Node.js. Mongoose надає зручний високорівневий інтерфейс для створення та управління даними, які зберігаються у MongoDB.

Основні переваги використання Mongoose:

1. Схеми даних: Mongoose дозволяє визначати структуру даних через схеми, де можна вказувати типи даних, валідатори, та за замовчуванням значення. Це сприяє кращій організації даних і забезпечує консистентність на рівні вашої бази даних. На рисунку 4.6 відображено приклад схеми.

2. Валідація даних: Mongoose вбудовує потужні можливості для валідації даних, що забезпечує правильність даних перед їх збереженням у базі. Це може включати перевірку діапазонів чисел, довжини рядків, наявності обов'язкових полів, та інше.

3. Запити та зміна даних: Mongoose спрощує створення запитів до MongoDB з використанням своїх методів та функцій, таких як find, findById, update, і delete. Ці методи надають інтуїтивно зрозумілий інтерфейс для взаємодії з базою даних.

4. Мідлвари (Middleware): Mongoose має систему мідлварів, яка дозволяє розробникам виконувати код під час певних дій у життєвому циклі моделі, наприклад, перед збереженням документа. Це може бути корисно для логування, модифікації даних або виконання комплексних перевірок.

5. Підтримка обробки помилок: Робота з помилками в Mongoose чітко структурована, що дозволяє легко обробляти та відповідати на помилки, що виникають під час взаємодії з базою даних.

```
const opportunitySchema = new Schema<IOpportunity>({
  title: { type: String, required: true },
  author: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true },
  body: { type: String, required: true },
  location: { type: String, required: true },
  categories: [{ type: mongoose.Schema.Types.ObjectId, ref: "Category" }],
  createdAt: { type: Date, default: Date.now, required: true },
});

const Opportunity: Model<IOpportunity> =
  mongoose.models.Opportunity ||
  mongoose.model<IOpportunity>("Opportunity", opportunitySchema);

export default Opportunity;
```

Рис 4.6 Приклад схеми бази даних з використанням бібліотеки Mongoose.

4.3 Тестування та публікація

Тестування програмного забезпечення поділяється на два основні типи: статичне та динамічне [25]:

- Статичне тестування не передбачає виконання коду. Воно може включати перевірку коду на відповідність стандартам, рецензування коду, а також використання інструментів для статичного аналізу.
- Динамічне тестування виконується на працюючому програмному забезпеченні і спрямоване на виявлення вад у реальних умовах його експлуатації.

У проект було включене статичне тестування, щоб забезпечити високу якість коду та виявити можливі проблеми на ранніх етапах розробки. Статичне тестування не вимагало виконання коду, що дозволило ефективно аналізувати код на предмет помилок, стилістичних невідповідностей і потенційних вразливостей без необхідності запуску програми.

Було використано інструменти статичного аналізу, які допомогли автоматично перевіряти код на наявність типових помилок і забезпечити дотримання встановлених стандартів кодування.

Інструменти які використовувались:

1. Prettier — це інструмент форматування коду, який допомагає підтримувати однорідний стиль написання коду в проекті. Хоча це не інструмент статичного аналізу у класичному розумінні, він працює в парі з ESLint для забезпечення чистоти і читабельності коду, що також є важливим аспектом якості програмного продукту.

2. ESLint є одним з найпопулярніших інструментів статичного аналізу для JavaScript і TypeScript, що допомагає знаходити і виправляти проблеми в коді на основі конфігурованих правил. Він ідеально підходить для проектів на базі Next.js, дозволяючи автоматизувати перевірку коду на відповідність стандартам кодування, виявлення синтаксичних помилок, а також потенційних проблем з продуктивністю і безпекою. Приклад використання ESLint в коді наведено на рисунку 4.7

3. TypeScript – хоча він сам по собі не є інструментом статичного аналізу, використання TypeScript у проектах Next.js значно підвищує якість коду завдяки суворій типізації і компіляційній перевірці, що дозволяє виявляти помилки ще до виконання коду.

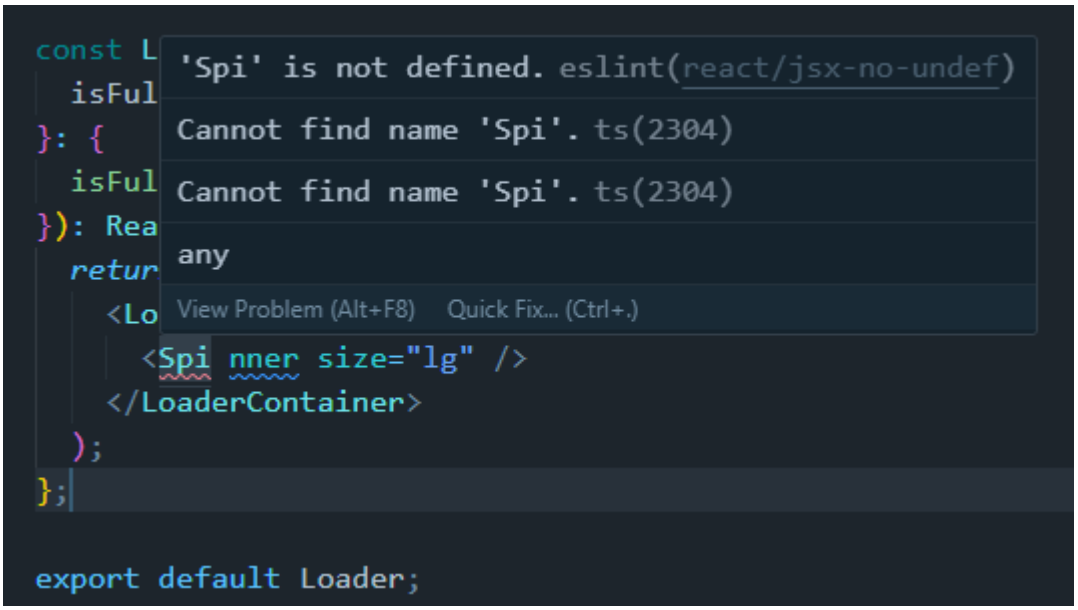


Рис 4.7 Приклад використання ESLint в коді

Ці інструменти допомогли забезпечити високий стандарт якості коду, зменшити кількість помилок на ранніх етапах розробки і спростити процес майбутнього тестування та обслуговування програмного продукту.

Також у проекті було застосоване динамічне тестування яке дозволяє перевіряти функціональність програми в реальному часі, виконуючи код та аналізуючи поведінку програми під час її виконання.

Спочатку було розроблено набір тестових сценаріїв, що відображали різні користувацькі сценарії використання платформи. Це включало сценарії для реєстрації користувачів, пошуку та розміщення потреб, комунікацій між учасниками та відгуків.

Для динамічного тестування були застосовані такі підходи:

- Функціональне тестування: Перевірка кожної функції програми шляхом подачі вхідних даних та оцінювання вихідних результатів.
- Інтеграційне тестування: Тестування взаємодій між компонентами системи для забезпечення їх коректної роботи разом.

Тестування проводилося в різних середовищах, включаючи локальні машини та розгортання в хмарі, для оцінки поведінки програми в різних умовах. Це допомогло ідентифікувати проблеми, які могли б виникнути лише в певних конфігураціях.

Одним з елементів динамічного тестування стала перевірка ефективності застосунку сторонніми сервісами. Тестування ефективності та швидкості веб-сторінок є важливим елементом забезпечення кращого користувацького досвіду та оптимізації роботи веб-сайту. Інструмент, який використовувався для цієї мети, — це PageSpeed Insights від Google. PageSpeed Insights надає оцінку продуктивності сторінки, яка дозволяє зрозуміти, наскільки швидко сторінка завантажується для кінцевого користувача.

PageSpeed Insights аналізує вміст веб-сторінки, а потім генерує пропозиції для підвищення швидкості її завантаження. Інструмент використовує дані з Chrome User Experience Report для надання інформації про реальну швидкість сторінок та взаємодії користувачів.

Процес тестування з PageSpeed Insights:

1. Введення URL-адреси веб-сторінки, яку потрібно тестувати.

2. Аналіз сторінки: PageSpeed Insights проаналізував сторінку, враховуючи фактори, такі як час завантаження, виконання JavaScript, стилів CSS та інших ресурсів.

3. Отримання результатів: Інструмент надав детальний звіт з оцінкою швидкості та рекомендаціями щодо її підвищення, такими як оптимізація зображень, видалення блокувального JavaScript, поліпшення відгуків сервера тощо.

Google PageSpeed Insights вимірює ефективність сайту, надаючи йому бал від 0 до 100 на основі ряду ключових метрик. Цей бал визначається за допомогою інструменту Lighthouse, який аналізує веб-сторінку в лабораторних умовах для моделювання завантаження сторінки. Важливі метрики включають Перше контентне зображення (FCP), Індекс швидкості, Час блокування (TBT), Найбільше контентне зображення (LCP) та Кумулятивний зсув макета (CLS). Кожна метрика має свою вагу у загальному рейтингу продуктивності.

В результаті тестування було отримано конкретні поради щодо оптимізації, які було застосовано для поліпшення загальної продуктивності сайту. Зокрема, було оптимізовано зображення та стилі CSS, що значно зменшило час завантаження сторінки. Після Результати звіту тестування після оптимізації представлені на рисунку 4.8.

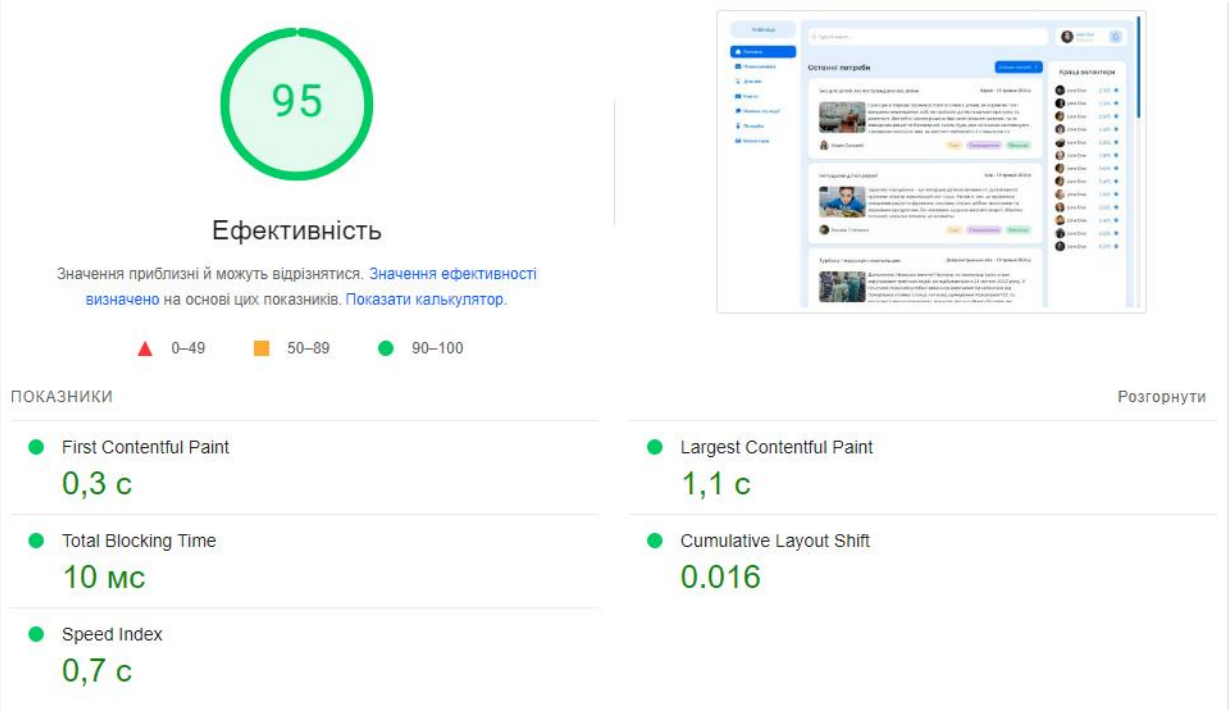


Рис 4.8 Результати тестування ефективності платформи за допомогою сервісу PageSpeed Insights

Для публікації платформи для волонтерської взаємодії було вирішено використовувати сервіс Vercel, який є одним із найкращих рішень для розгортання веб-застосунків на базі Next.js.

У процесі розгортання були виконані такі кроки:

1. Спершу було створено новий проект, вибравши можливість імпортувати проект з репозиторію GitHub, де зберігався код сайту.

2. Після імпортування проекту, Vercel автоматично визначив, що використовувався Next.js, і запропонував оптимальні налаштування для

розгортання. Було додано необхідні змінні середовища через інтерфейс Vercel, які були потрібні для доступу до бази даних MongoDb і інших зовнішніх сервісів.

3. Після натискання кнопки "Deploy" Vercel автоматично обробив код, встановив залежності та створив збірку проекту. Розгортання було здійснено за кілька хвилин.

4. Після розгортання було перевірино роботу сайту за наданою Vercel посиланням. Також були використані інструменти Vercel для аналізу продуктивності сайту.

5. Однією з великих переваг Vercel є те, що будь-які майбутні оновлення коду в репозиторії GitHub автоматично ініціюють новий процес розгортання, що робить процес підтримки та оновлення сайту дуже зручним.

Використання Vercel для розгортання проекту значно спростило управління інфраструктурою та дало можливість розширеного тестування застосунку. На рисунку 4.9 представлена сторінка з деталями опублікованого проекту на Vercel.

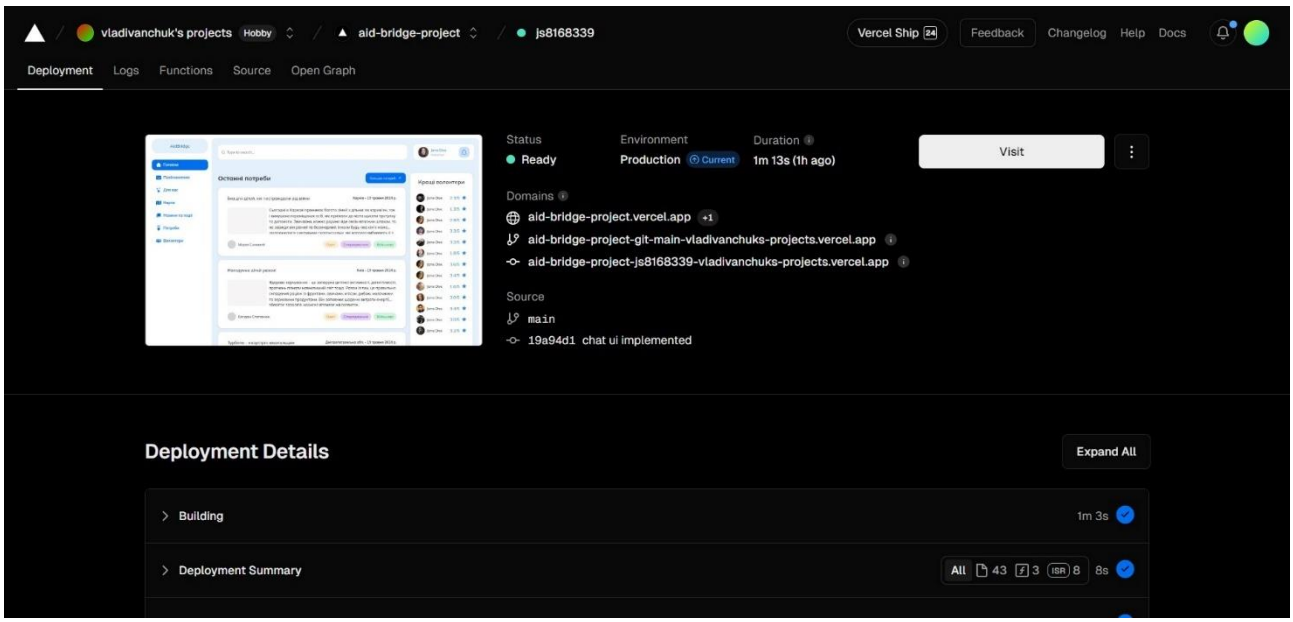


Рис 4.9 Сторінка з деталями опублікованого проекту на Vercel

4.4 Використання платформи

Для використання платформи користувачу потрібно перейти на головну сторінку застосунку (рисунок 4.10) на якій будуть відображені наступні елементи:

Останні потреби: Ця секція відображає список останніх потреб з детальною інформацією про кожну потребу, включаючи назву потреби, короткий опис, інформацію про того, хто опублікував потребу, та кнопки для перегляду деталей.

Нові можливості: Поруч з останніми потребами, секція нових можливостей волонтерів відображає можливості для участі, з основною інформацією та кнопками для отримання більше деталей.

Перелік наявних потреб та волонтерських можливостей на карті: Банер посилання нас сторінку з інтерактивною карта, відображає географічне розташування активних потреб та можливостей.

Навігаційне меню: У лівій частині сторінки розташоване навігаційне меню з доступом до різних розділів сайту, таких як головна сторінка, профіль користувача, потреби, можливості, новини та події, та інші важливі посилання.

Верхня частина: У хедері сайту розміщене поле для пошуку а також кнопки входу та реєстрації.

Панель кращих волонтерів: На правому боці сторінки є панель, яка відображає список кращих волонтерів з їхнім рейтингом, надаючи користувачам швидкий доступ до профілів топ волонтерів.

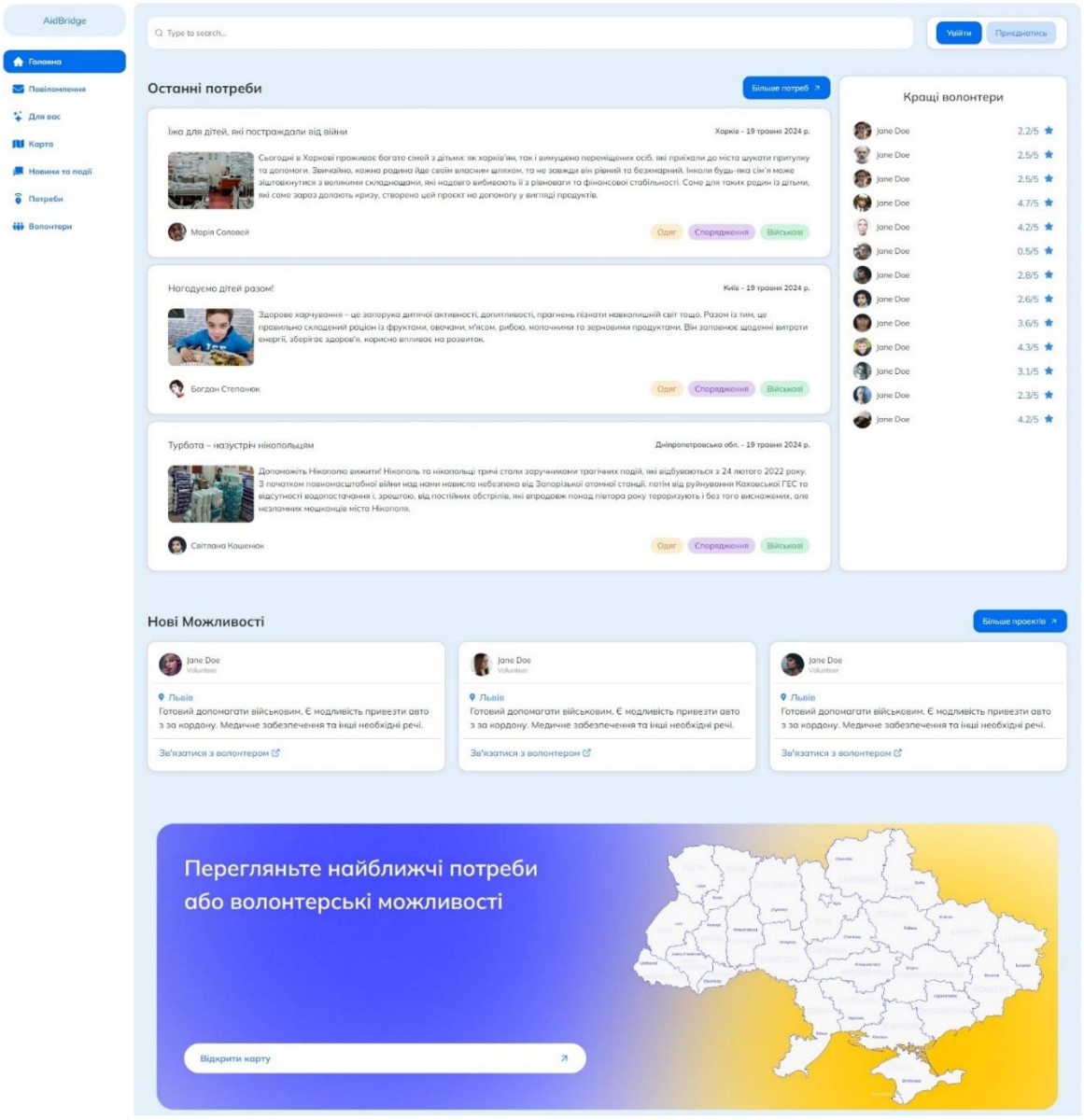


Рис 4.10 Головна сторінка платформи AidBridge

Якщо користувач хоче авторизуватись він може натиснути кнопку «Увійти» або «Приєднатись» в верхній частині сайту. Після цього він побачить вікно авторизації яке зображено на рисунку 4.11. У цьому вікні користувач може ввести пароль та логін для того щоб увійти або перемкнути режим на реєстрацію і ввести необхідні дані для створення акаунту. Після успішної авторизації користувач побачить на місці кнопок відповідний компонент який буде відображати його профіль (рисунок 4.12). При кліку на цей компонент користувач зможе перейти на сторінку свого профілю.

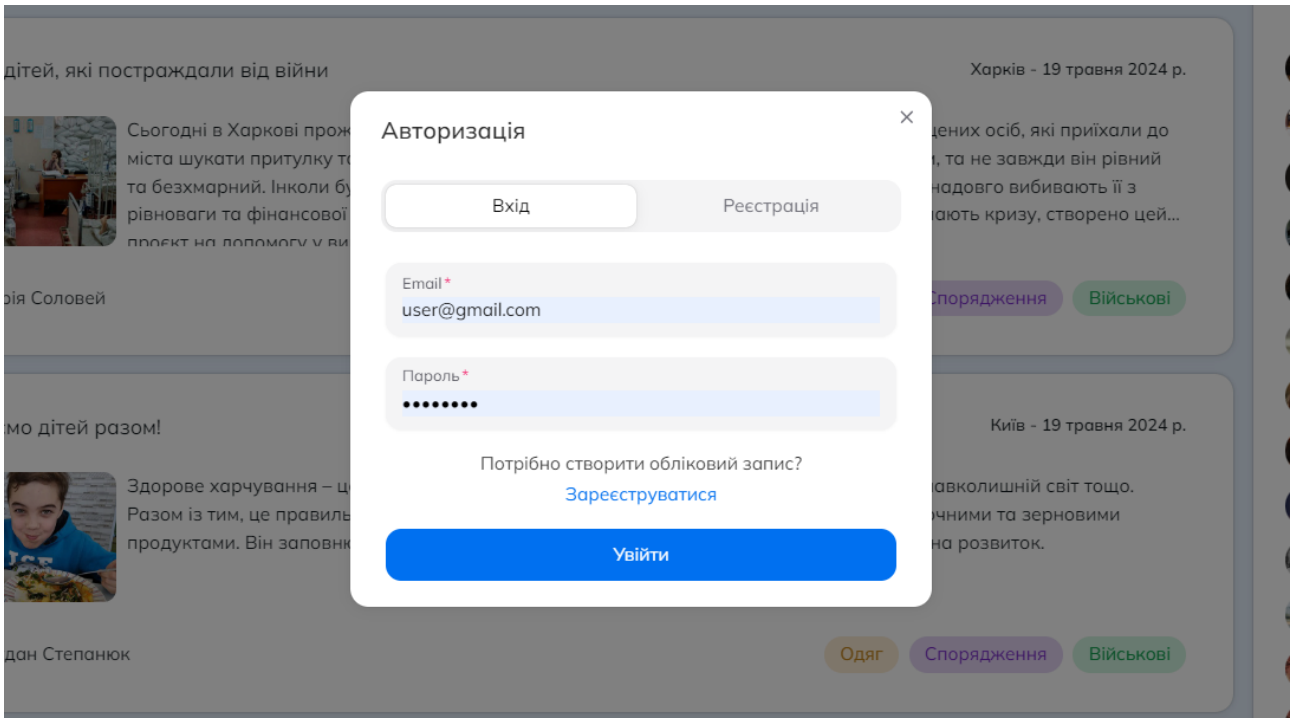


Рис 4.11 Вікно авторизації на платформі AidBridge

Рис 4.12 Компонент який відображає профіль після входу на платформі AidBridge

Якщо користувач перейде на сторінку свого профілю зображену на рисунку 4.13, він побачить у верхній частині компонент який відображає його дані. А саме фото, ім’я, роль на платформі, рейтинг а також кнопку редагування профілю. У центральній частині користувач побачить компонент який відображає опубліковані відгуки інших користувачів. Також є можливість створення нового відгуку при перегляді профілю іншого користувача. Є можливість перемкнути режим відображення з відгуків на опубліковані можливості якщо роль користувача волонтер або потреби якщо користувач бенефіціар. Також у правій частині сторінки можна побачити компонент які відображають іншу інформацію

користувача. А саме вказану локацію, дату з якої користувач зареєстрований на платформі, коли була остання активність користувача а також опис профілю.

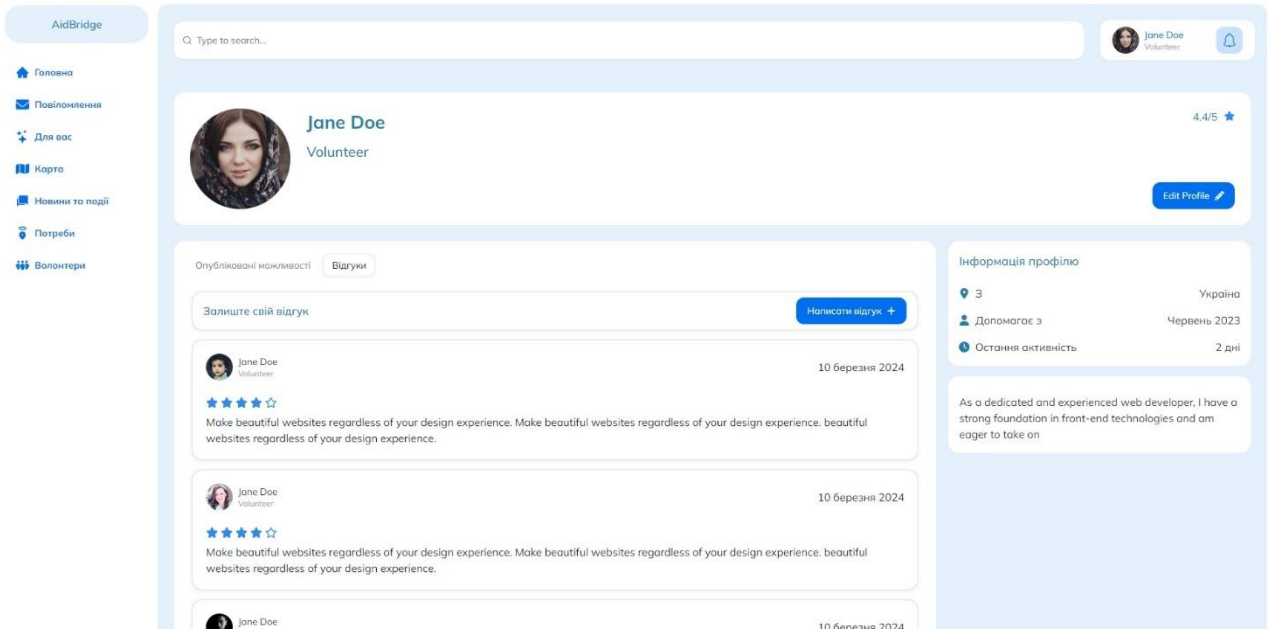


Рис 4.13 Сторінка профілю користувача на платформі AidBridge

Якщо користувач як волонтер захоче почати допомагати він може перейти на сторінку потреб де відображений список останніх потреб інших користувачів. Ця сторінка зображена на рисунку 4.14. У верхній частині знаходяться кнопки для фільтрації за категоріями а також кнопка для створення нової потреби. У центральній частині знаходиться список потреб з картинкою, заголовком та коротким описом потреби а також її категоріями, автором, датою публікації та місцезнаходженням.

Користувач має можливість створити нову потребу натиснувши у верхній частині на кнопку з надписом «Додати потребу». Після цього на екрані з’явиться нове вікно яке включає поля для створення нової платформи. Це вікно зображене на рисунку 4.15. Після успішного заповнення форми валідними даними користувач має натиснути кнопку з надписом «Опублікувати», після чого нова потреба одразу з’явиться у списку інших потреб.

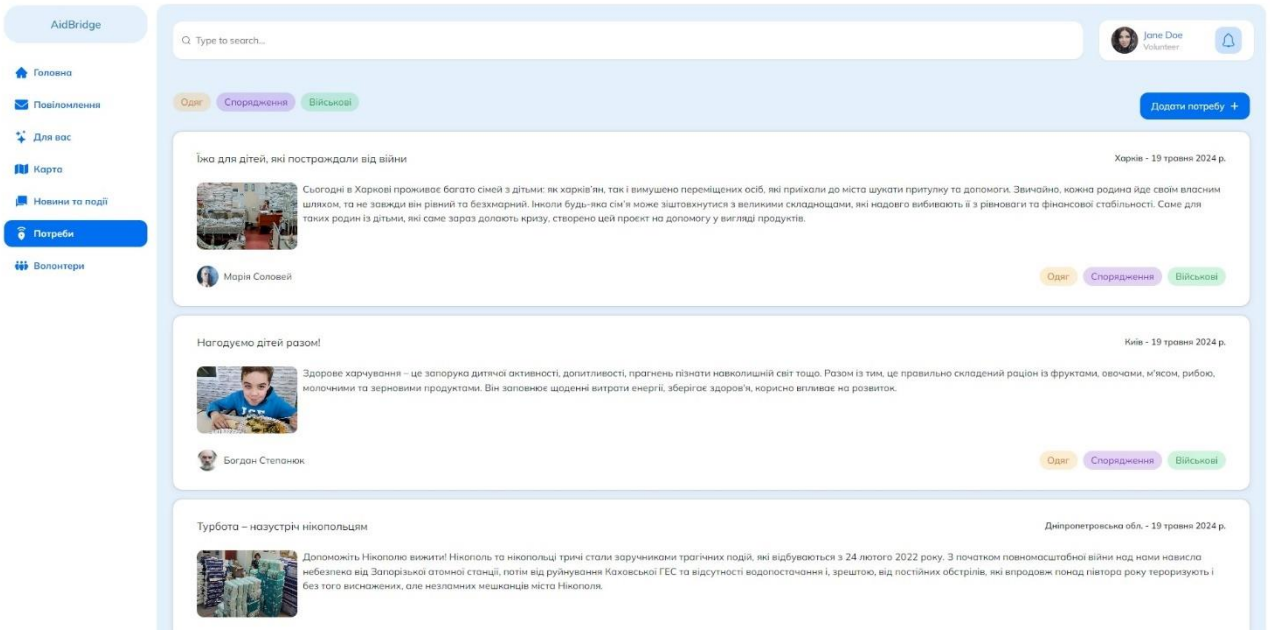


Рис 4.14 Сторінка з списком потреб

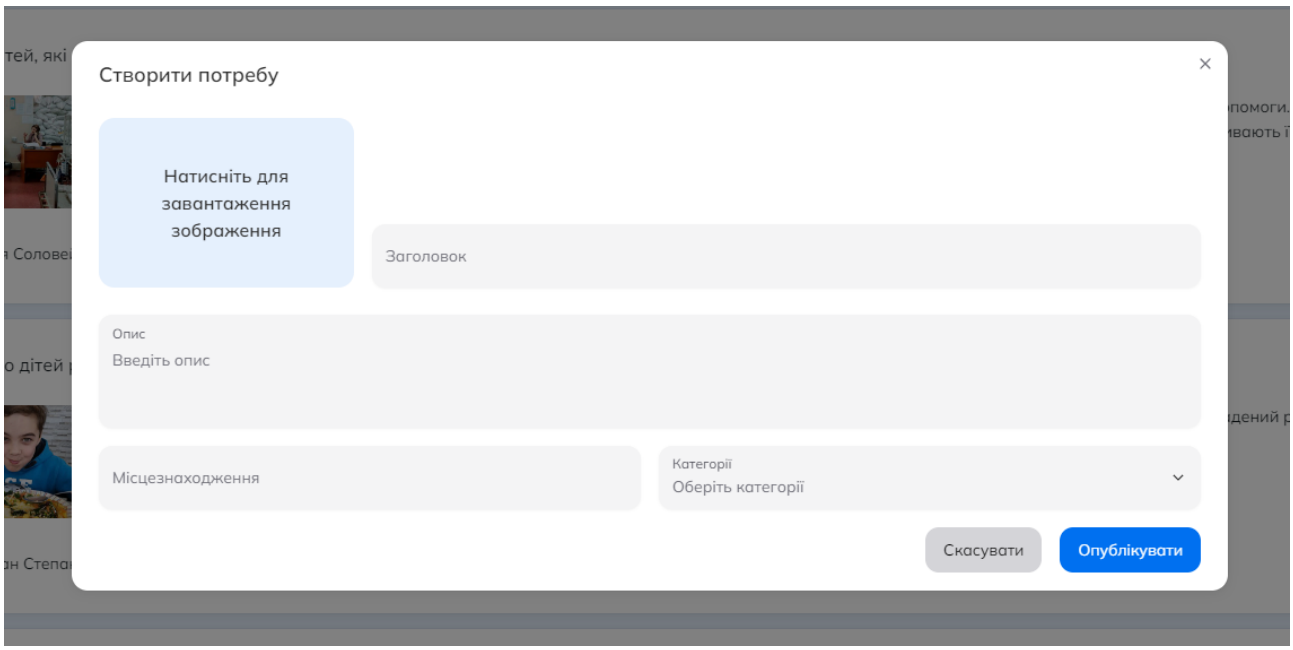


Рис 4.15 Вікно створення потреби

Якщо користувача зацікавила якась потреба він може натиснути на відповідну потребу і потрепити на сторінку зображену на рисунку 4.16. На цій сторінці він може детально ознайомитись з деталями потреби а також натиснути

кнопку «Відгукнутись» якщо він готовий допомогти. При натисканні на цю кнопку користувач попаде в центр комунікації де з правої сторони відображені всі чати користувача, а по центру він побачить відкритий чат з користувачем на потребу якого він може відгукнутись. Сторінка яка відображає центр комунікації відоьражена на рисунку 4.17.

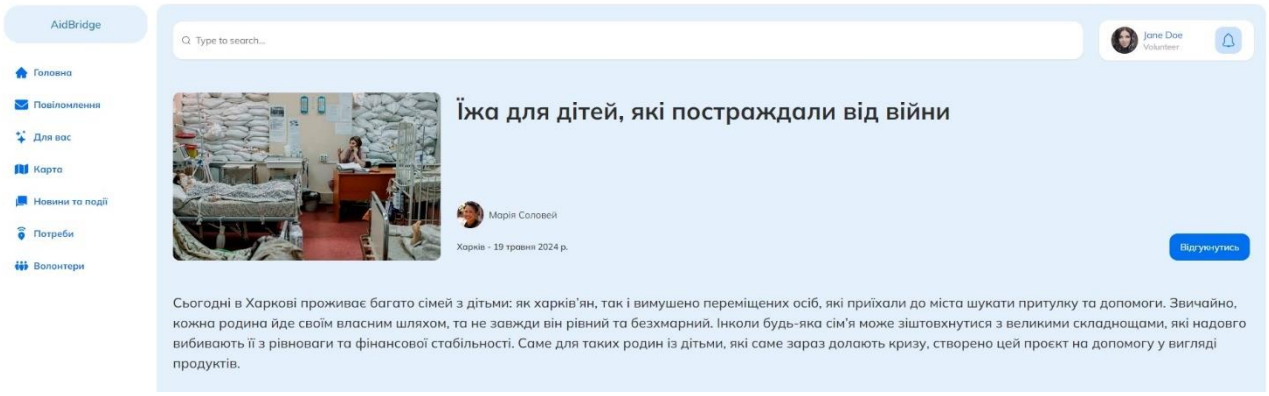


Рис 4.16 Сторінка деталей потреби

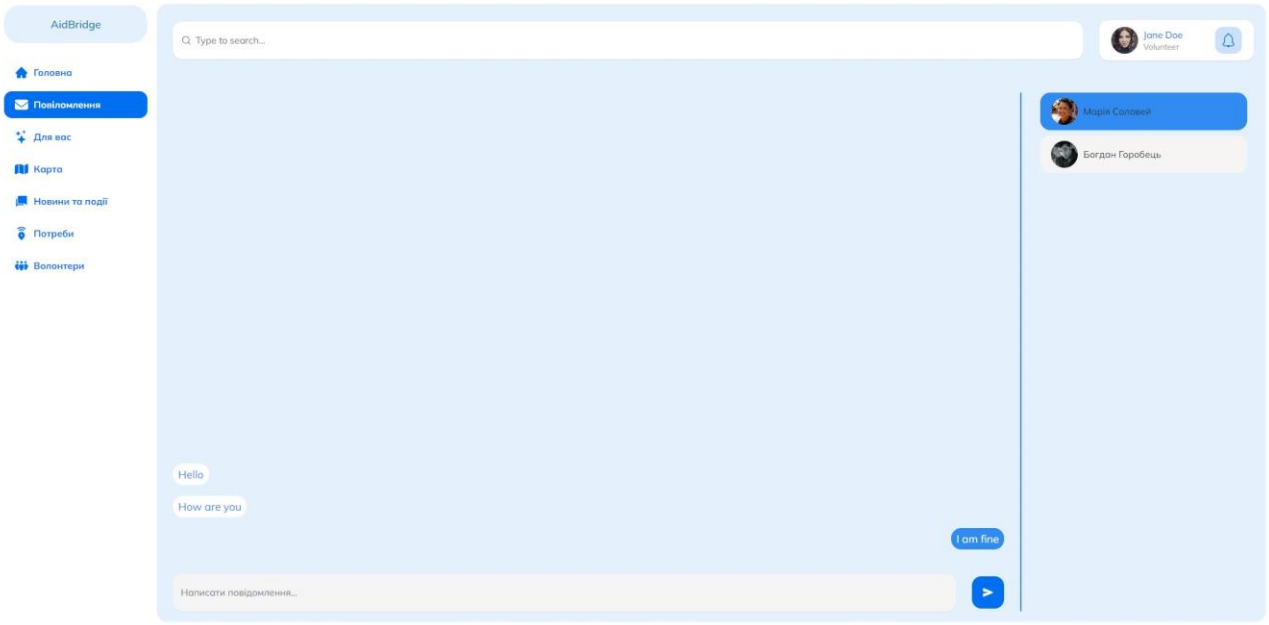


Рис 4.17 Сторінка чату з користувачем

ВИСНОВКИ

1. Аналіз існуючих рішень дозволив ідентифікувати основні недоліки та потенціал для покращення у сфері волонтерської діяльності. Було встановлено, що багато платформ не забезпечують достатньої гнучкості або не охоплюють усі потреби користувачів.

2. Вивчення вимог до розробки платформи виявило ключові функціональні і технічні аспекти, які потрібно було врахувати при проектуванні. Розроблено детальні специфікації, що стали основою для подальшої реалізації платформи.

3. Проектування архітектури системи було виконано з урахуванням сучасних практик та технологій, забезпечивши продуктивність та масштабованість системи. Використання Next.js як основи забезпечило ефективну роботу платформи в реальному часі.

4. Розробка інтерфейсу користувачів була зосереджена на впровадженні функціональних елементів, для взаємодію користувачів з платформою.

5. Були успішно впроваджені ключові функції, які включають реєстрацію та авторизацію користувачів, систему управління профілем з відгуками та рейтингами, механізмам пошуку та фільтрації для ефективної навігації між потребами та волонтерськими можливостями.

6. Тестування системи підтвердило її надійність і відповідність поставленим вимогам, а також допомогло виявити та виправити помилки та покращити продуктивність.

Таким чином, завдяки використанню Next.js як Full Stack рішення, створено платформу для волонтерської взаємодії, що відповідає потребам сучасних волонтерських ініціатив.

ПЕРЕЛІК ПОСИЛАНЬ

1. Яскевич В.О., Іванчук В.О. Дослідження системи рекомендацій для платформи волонтерської взаємодії. *Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях»*, 24 квітня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2024. С.110-112.
2. Яскевич В.О., Іванчук В.О. Безпека даних у хмарних сервісах. *IV Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті»*, 15 травня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2024. С.120-122.
3. Про волонтерську діяльність : Закон України від 19.04.2011 № 3236-VI. https://zakon.rada.gov.ua/laws/show/3236-17#Text (дата звернення: 11.04.2024)
4. Корнят В. С., Боринець Х. М. (2024). Волонтерська діяльність в умовах воєнного стану у порівняльній перспективі. *Гуманітарний форум*, *2*(1), С. 24-29. https://doi.org/10.60022/2(1)-4GF
5. Web – застосунок Є Допомога. URL: https://social.edopomoga.gov.ua/uk/ (дата звернення: 15.04.2024)
6. Web – застосунок dobro.ua. URL: https://dobro.ua/ (дата звернення: 15.04.2024)
7. Web – застосунок GivePulse. URL: https://www.givepulse.com (дата звернення: 15.04.2024)
8. Software Requirements Specification. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/software-requirement-specification-srs-format/. – (дата звернення 18.04.2024)
9. HTML Довідник. W3schoolsUA. URL: https://w3schoolsua.github.io/index.html (дата звернення: 20.04.2024)
10. Український веб-довідник. css.in.ua. URL: https://css.in.ua/ (дата звернення: 20.04.2024)

11. Сучасний підручник з JavaScript. javascript.info. URL:https://uk.javascript.info/ (дата звернення: 20.04.2024)
12. Malokhvii E., Buhai V., Molchanov H., Chernykh O. Аналітичний огляд та порівняння сучасних javascript рішень для розробки веб-застосунків. *Системи управління, навігації та зв'язку*. Збірник наукових праць. 2021, 4(66), 55-58. URL: https://doi.org/10.26906/SUNZ.2021.4.055.
13. Leo Rodriguez. Best JavaScript Frameworks: Pros, Cons, and Statistics. 17.05.2023. URL: https://prerender.io/blog/best-javascript-frameworks-pros-cons-and-statistics/. (дата звернення 22.04.2024).
14. React - The library for web and native user interfaces. react.dev. URL: https://uk.react.dev/ (дата звернення: 25.04.2024)
15. The React Framework for the Web. Документація Next.js. URL: https://nextjs.org/ (дата звернення: 25.04.2024)
16. The web development framework for building the future. Документація Angular. URL: https://angular.io/https://angular.io/ (дата звернення: 25.04.2024)
17. The Progressive JavaScript Framework. Документація Vue.js. URL: https://vuejs.org/ (дата звернення: 25.04.2024)
18. Developer Survey. Stack Overflow. URL: https://survey.stackoverflow.co/2022
19. Шаров, С. В. та Петровський , В. В. (2015) Огляд нереляційних баз даних. Всеукраїнська Internet-конференція «Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення» (14). с. 16-18. http://eprints.mdpu.org.ua/id/eprint/602
20. Кицмен Д. Р. (2023) Система вибору NoSQL сховища даних на основі бажаних характеристик інформаційної системи. Дев'ятнадцята всеукраїнська практично-пізнавальна інтернет-конференція. https://naukam.triada.in.ua/index.php/konferentsiji/49-dev-yatnadtsyata-vseukrajinska-praktichno-piznavalna-internet-konferentsiya/438-sistema-viboru-nosql-skhovishcha-danikh-na-osnovi-bazhanikh-kharakteristik-informatsijnoji-sistemi

21. MongoDB: The Developer Data Platform. MongoDB. https://www.mongodb.com/
22. Kettenis J. (2007) Getting Started With Activity Modeling. Oracle Corporation. URL:https://www.oracle.com/technetwork/developer-tools/jdev/gettingstartedwithactivitymodeling-134487.pdf (дата звернення 02.05.2024).
23. Юлія Каграманова. Як будувати UML-діаграми. Розбираємо три найпопулярніші варіанти. 27.10.2022. URL: https://dou.ua/forums/topic/40575/. (дата звернення 08.05.2024).
24. App Router. Документація Next.js. URL: https://nextjs.org/docs/app. (дата звернення 10.05.2024).
25. Об`єдкова, Д. Д., Родіонов, П. Ю. (2023). Класифікація процесів тестування програмного забезпечення. *Комп'ютерні науки та програмна інженерія: Теоретичні та практичні аспекти сучасних наукових досліджень*. https://archive.logos-science.com/index.php/conference-proceedings/article/view/1314

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО -КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО -НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

РОЗРОБКА ПЛАТФОРМИ ДЛЯ ВОЛОНТЕРСЬКОЇ ВЗАЄМОДІЇ З ВИКОРИСТАННЯМ WEB-ФРЕЙМВОРКУ NEXT.JS

Виконав студент 4 курсу
групи ПД-41
Іванчук Владислав Олександрович
Керівник роботи
К.т.н, доцент кафедри ІПЗ Яскевич Владислав Олександрович

Київ – 2024

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи –** підвищення ефективності волонтерської діяльності шляхом створення цифрової платформи для зручного спілкування, координації дій та залучення громадян до волонтерської діяльності.
- **Об’єкт дослідження –** волонтерська діяльність як соціальне явище, що охоплює взаємодію волонтерів та громадськості через використання інформаційних технологій.
- **Предмет дослідження –** веб-платформа на базі фреймворку Next.js.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Аналіз існуючих рішень у сфері волонтерської взаємодії та виявлення їхніх основних недоліків.
2. Визначення вимог до розробки платформи волонтерської взаємодії.
3. Проектування архітектури системи волонтерської взаємодії.
4. Розробка інтерфейсу користувачів.
5. Реалізація функціональних можливостей платформи волонтерської взаємодії.
6. Тестування системи волонтерської взаємодії.

3

АНАЛІЗ АНАЛОГІВ

Показник	dobro.ua	Є Допомога	GivePulse	AidBridge
Платформи	Web	Web	Android , iOS, Web	Web
Оцінка ефективності у PageSpeed Insights	Мобільна: 36, Десктоп: 70	Мобільна: 36, Десктоп: 49	Мобільна: 49, Десктоп: 61	Мобільна: 74, Десктоп: 95
Індекс швидкості	Мобільна: 7.8 с, Десктоп: 2.7 с	Мобільна: 4.0 с, Десктоп: 2.0 с	Мобільна: 8,9 с, Десктоп: 2,8 с	Мобільна: 2.3 с, Десктоп: 1,7 с
Отриманння допомоги	Пошук благодійних організації	Розміщення заявки про потребу	-	Розміщення потреб; перегляд волонтерських можливостей; персоналізовані рекомендації
Волонтерство	Пошук або створення кампаній, проектів та зборів	Пошук потреб; фінансові донати; допомага із рук в руки	Пошук проектів; реєстрація на участь в них	Розміщення можливостей; Участь в зборах; перегляд потреб; Пошук потреб; Пошук потреб на інтерактивній карті; персоналізовані рекомендації
Комунікація	-	Обмін контактами для зв'язку	-	Система рейтингу і відгуків; можливість зв'язатись з волонтером або тими хто потребує допомоги

4

ВИМОГИ ДО ЗАСТОСУНКУ

Функціональні вимоги:

1. Реєстрація і авторизація користувачів
2. Перегляд та редагування профілю користувача
3. Пошук та розміщення волонтерських можливостей (для волонтерів)
4. Пошук та розміщення потреб (для бенефіціарів)
5. Система відгуків та рейтингу
6. Карта потреб
7. Створення та перегляд новин та подій, їх обговорення

Нефункціональні вимоги:

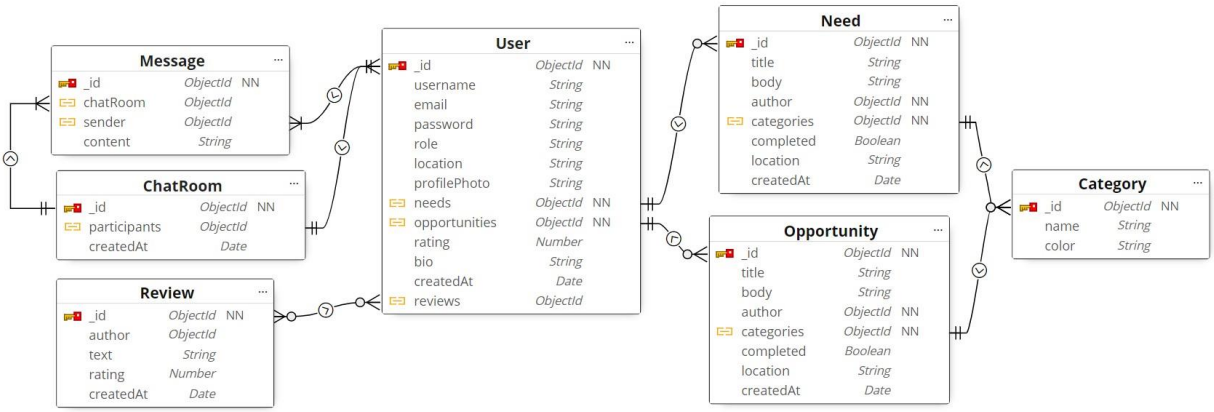
1. Висока швидкість реакції і обробки запитів
2. Забезпечення стабільної та безперебійної роботи системи
3. Адаптація під різні розміри екранів

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

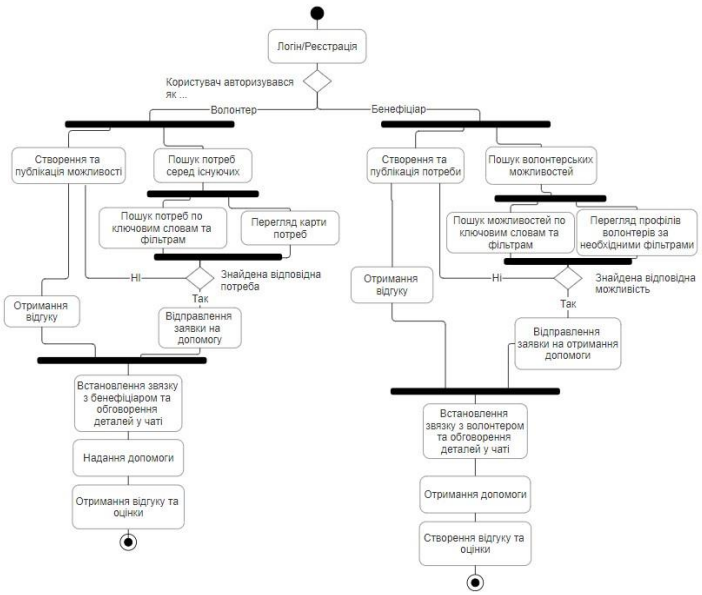
6

ER-ДІАГРАМА



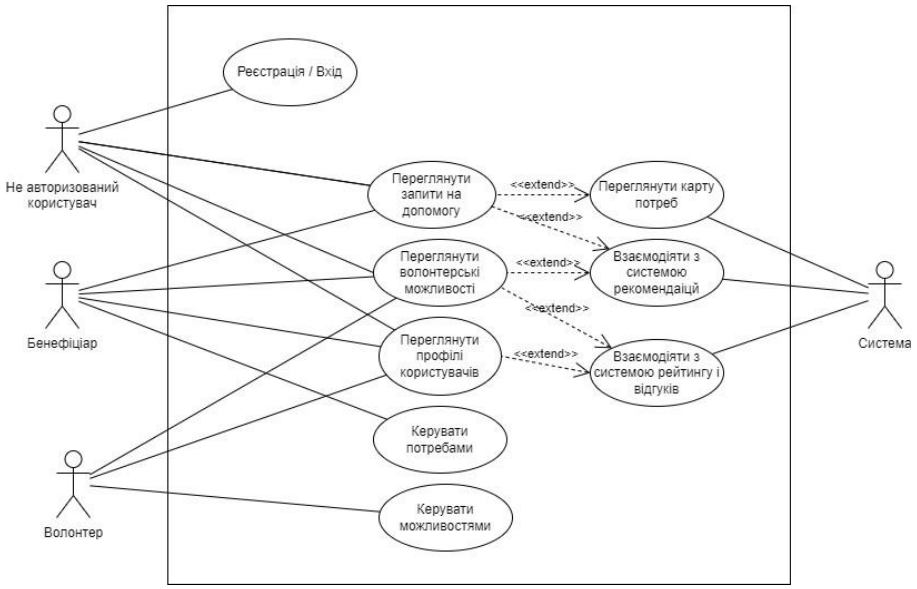
7

ДІАГРАМА ДІЯЛЬНОСТІ



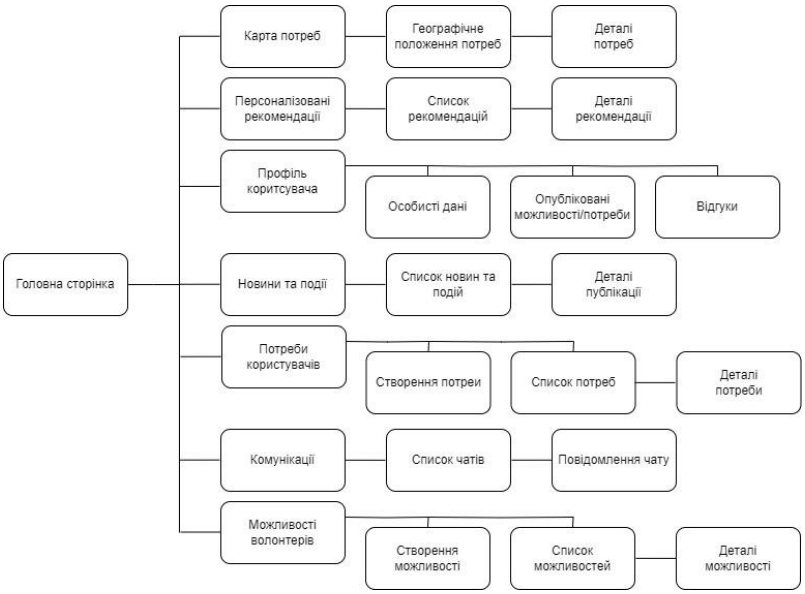
8

USE CASE ДІАГРАМА



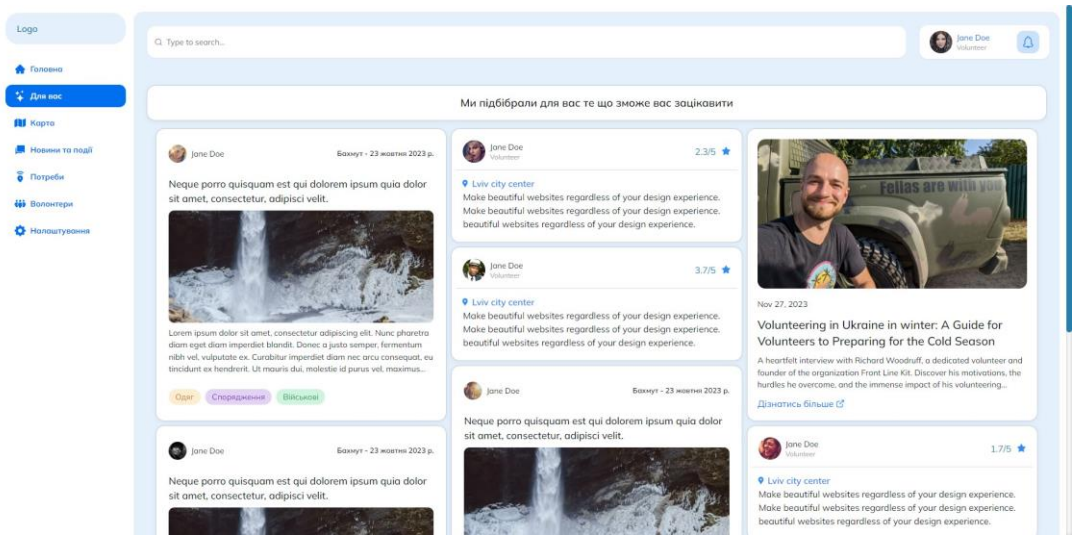
9

СТРУКТУРА ПЛАТФОРМИ



10

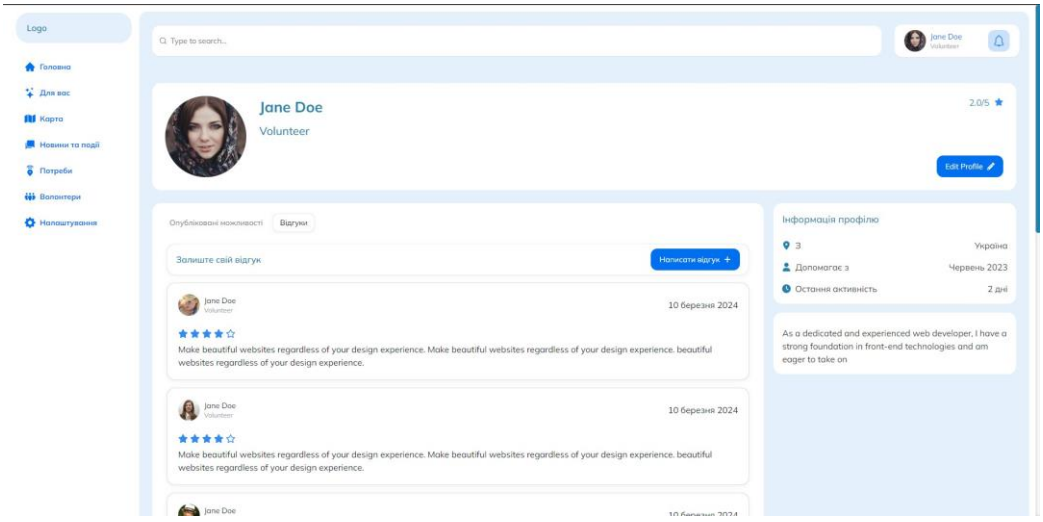
ЕКРАННІ ФОРМИ



Персоналізовані рекомендації

11

ЕКРАННІ ФОРМИ



Профіль користувача

12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Іванчук В.О., Дослідження системи рекомендацій для платформи волонтерської взаємодії / Іванчук В.О, Яскевич В.О. // Застосування програмного забезпечення в інформаційно-комунікаційних технологіях: Матеріали всеукраїнської науково-технічної конференції. 24.04.2024 Збірник тез. – К.: ДУІКТ, 2024, 110 с.
2. Іванчук В.О., Безпека даних у хмарних сервісах / Іванчук В.О, Яскевич В.О. // Сучасні інтелектуальні інформаційні технології в науці та освіті: Матеріали четвертої всеукраїнської науково-технічної конференції. Подано до друку.

ВИСНОВКИ

1. Аналіз існуючих рішень дозволив ідентифікувати основні недоліки та потенціал для покращення у сфері волонтерської діяльності. Було встановлено, що багато платформ не забезпечують достатньої гнучкості або не охоплюють усі потреби користувачів.
2. Вивчення вимог до розробки платформи виявило ключові функціональні і технічні аспекти, які потрібно було врахувати при проектуванні. Розроблено детальні специфікації, що стали основою для подальшої реалізації платформи.
3. Проектування архітектури системи було виконано з урахуванням сучасних практик та технологій, забезпечивши продуктивність та масштабованість системи. Використання Next.js як основи забезпечило ефективну роботу платформи в реальному часі.
4. Розробка інтерфейсу користувачів була зосереджена на впровадженні функціональних елементів, для взаємодію користувачів з платформою.
5. Були успішно впроваджені ключові функції, які включають реєстрацію та авторизацію користувачів, систему управління профілем з відгуками та рейтингами, механізмам пошуку та фільтрації для ефективної навігації між потребами та волонтерськими можливостями.
6. Тестування системи підтвердило її надійність і відповідність поставленим вимогам, а також допомогло виявити та виправити помилки та покращити продуктивність.

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

```
import {
  BestVolunteers,
  MapBanner,
  NewOpportunities,
  News,
  Questions,
  RecentNeeds,
} from "@/components";
import { HomeContainer, HomeTopContainer } from "@/styles/HomeStyles";

const Home = (): React.ReactElement => {
  return (
    <HomeContainer>
      <HomeTopContainer>
        <RecentNeeds />
        <BestVolunteers />
      </HomeTopContainer>
      <NewOpportunities />
      <MapBanner />
      <News />
      <Questions />
    </HomeContainer>
  );
};

export default Home;

"use client";
import {
  RecentNeedsContainer,
  HomeTitle,
  HomeSectionHeader,
```

```
} from "@/styles/HomeStyles";
import { Loader, NeedsItem } from "..";
import { LinkContainer } from "@/styles/UiStyles";
import Link from "next/link";
import { Button, useDisclosure } from "@nextui-org/react";
import { MdArrowOutward } from "react-icons/md";
import { getNeeds } from "@/lib/api";
import { INeed } from "@/models/need";
import { NeedsContainer } from "@/styles/NeedsStyles";
import { useState, useEffect } from "react";

const RecentNeeds = (): React.ReactElement => {
  const [needs, setData] = useState<INeed[]>([]);
  const [isLoading, setLoading] = useState(true);

  useEffect(() => {
    getNeeds().then((data) => {
      setData(data.data);
      setLoading(false);
    });
  }, []);

  if (!needs) return <p>No needs data</p>;
  return (
    <RecentNeedsContainer>
      <HomeSectionHeader>
        <HomeTitle>Останні потреби</HomeTitle>
        <LinkContainer>
          <Button
            as={Link}
            href="needs"
            color="primary"
            endContent={<MdArrowOutward />}
          >
            Більше потреб
```

```
          </Button>
        </LinkContainer>
      </HomeSectionHeader>
      {isLoading ? (
        <Loader isFullscreen={true} />
      ) : (
        <NeedsContainer>
          {needs.map((need) => (
            <NeedsItem key={need.id} {...need} />
          ))}
        </NeedsContainer>
      )}
    </RecentNeedsContainer>
  );
};

export default RecentNeeds;

import mongoose from "mongoose";
import { Schema, Document, Model } from "mongoose";

export interface INeed extends Document {
  title: string;
  author: string;
  body: string;
  location: string;
  categories: mongoose.Types.ObjectId[];
  completed: boolean;
  ImageURL: string;
  createdAt: Date;
}

const needSchema = new Schema<INeed>({
  title: { type: String, required: true },
  author: { type: String, required: true },
```

```
  body: { type: String, required: true },
  location: { type: String, required: true },
  categories: [{ type: mongoose.Schema.Types.ObjectId, ref: "Category" }],
  completed: { type: Boolean },
  ImageURL: { type: String, required: true },
  createdAt: { type: Date, default: Date.now, required: true },
});

const Need: Model<INeed> =
  mongoose.models.Need || mongoose.model<INeed>("Need", needSchema);

export default Need;

import dbConnect from "@/lib/mongodb";
import { NextRequest, NextResponse } from "next/server";
import { Model } from "mongoose";

export async function createHandler(req: NextRequest, Model: Model<any>) {
  await dbConnect();
  try {
    const data = await req.json();
    await Model.create(data);
    return NextResponse.json({ message: "Created" }, { status: 200 });
  } catch (err: any) {
    return NextResponse.json({ error: err.message }, { status: 500 });
  }
}

export async function getAllHandler(Model: Model<any>) {
  await dbConnect();
  try {
    const data = await Model.find({});
    return NextResponse.json({ data });
  } catch (err: any) {
    return NextResponse.json({ error: err.message }, { status: 500 });
```

```
  }
}

export async function getByIdHandler(
  req: NextRequest,
  { params }: { params: { id: string } },
  Model: Model<any>,
) {
  await dbConnect();
  try {
    const { id } = params;
    const data = await Model.findOne({ _id: id });

    if (!data) {
      return NextResponse.json({ error: "Not found" }, { status: 404 });
    }

    return NextResponse.json({ data }, { status: 200 });
  } catch (err: any) {
    return NextResponse.json({ error: err.message }, { status: 500 });
  }
}

export async function updateHandler(
  req: NextRequest,
  { params }: { params: { id: string } },
  Model: Model<any>,
) {
  await dbConnect();
  try {
    const { id } = params;
    const updatedData = await req.json();
    const data = await Model.findByIdAndUpdate(id, updatedData, { new: true });

    if (!data) {
```

```
      return NextResponse.json({ error: "Not found" }, { status: 404 });
    }

    return NextResponse.json({ message: "Updated" }, { status: 200 });
  } catch (err: any) {
    return NextResponse.json({ error: err.message }, { status: 500 });
  }
}

export async function deleteHandler(
  req: NextRequest,
  { params }: { params: { id: string } },
  Model: Model<any>,
) {
  await dbConnect();
  try {
    const { id } = params;
    const data = await Model.findByIdAndDelete(id);

    if (!data) {
      return NextResponse.json({ error: "Not found" }, { status: 404 });
    }

    return NextResponse.json({ message: "Deleted" }, { status: 200 });
  } catch (err: any) {
    return NextResponse.json({ error: err.message }, { status: 500 });
  }
}

export async function createNeed(data: any) {
  const response = await fetch("/api/needs", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
```

```
    body: JSON.stringify(data),
  });
  return response.json();
}

export async function getNeeds() {
  const response = await fetch("/api/needs");
  return response.json();
}

export async function getNeedById(id: string) {
  const response = await fetch(`/api/needs/${id}`);
  return response.json();
}

export async function updateNeed(id: string, data: any) {
  const response = await fetch(`/api/needs/${id}`, {
    method: "PUT",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify(data),
  });
  return response.json();
}

export async function deleteNeed(id: string) {
  const response = await fetch(`/api/needs/${id}`, {
    method: "DELETE",
  });
  return response.json();
}

import mongoose from "mongoose";
```

```
const MONGODB_URI = process.env.MONGODB_URI || "";
if (!MONGODB_URI) {
  throw new Error(
    "Please define the MONGODB_URI environment variable inside .env.local",
  );
}

let cached = global.mongoose || null;

if (!cached) {
  cached = global.mongoose = { conn: null, promise: null };
}

async function dbConnect() {
  if (cached.conn) {
    return cached.conn;
  }
  if (!cached.promise) {
    cached.promise = mongoose
      .connect(MONGODB_URI, { bufferCommands: false })
      .then((mongoose) => {
        return mongoose;
      });
  }
  cached.conn = await cached.promise;
  return cached.conn;
}

export default dbConnect;
```