МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА СЕРВІСУ КОМУНІКАЦІЇ ВОЛОНТЕРІВ ТА БЛАГОДІЙНИХ ОРГАНІЗАЦІЙ З ВИКОРИСТАННЯМ SPRING»**

Виконав: студент 4 курсу, групи 6.1210-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
В.О. Кузьмінов
(ініціали та прізвище)

Керівник завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Кузьмінову Владиславу Олександровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка сервісу комунікації волонтерів та благодійних організацій з використанням Spring

керівник роботи Лісняк Андрій Олександрович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Проєктування архітектури системи.
3. Реалізація програмного забезпечення.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.01.2024	
2.	Збір вихідних даних.	22.01.2024	
3.	Обробка методичних та теоретичних	12.02.2024	
	джерел.		
4.	Розробка першого та другого розділу.	10.04.2024	
5.	Розробка третього розділу.	20.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	19.06.2024	

Студент _____ В.О. Кузьмінов
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.О. Лісняк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка сервісу комунікації волонтерів та благодійних організацій з використанням Spring»: 65 с., 23 рис., 1 табл., 15 джерел, 4 додатки.

БАЗА ДАНИХ, БЛАГОДІЙНІ ОРГАНІЗАЦІЇ, БЛАГОДІЙНІ ЗАХОДИ, ВОЛОНТЕРИ, ВЕБЗАСТОСУНОК, ПРОГРАМНА ІНЖЕНЕРІЯ, РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, СЕРВІС.

Об’єкт дослідження – процес взаємодії між благодійними організаціями та волонтерами у рамках благодійних проєктів.

Предмет дослідження – комплекс теоретичних, методичних та практичних рекомендацій щодо розробки програмного забезпечення для взаємодії двох зацікавлених груп.

Мета роботи – розробити сервіс для зв’язку благодійних організацій та волонтерів: система, яка дозволить організаціям знаходити та координувати волонтерів для різноманітних благодійних заходів та проєктів.

Методи дослідження – методи збору та аналізу вимог до програмного забезпечення, методи моделювання, проєктування, конструювання програмного забезпечення.

При розробці програми було проведено аналіз предметної області. Спроєктовано план дій, використовуючи діаграму Ганта. Вибрано та спроєктовано архітектуру системи за допомогою UML у вигляді наборів діаграм прецедентів, класів, активностей, а також ER-діаграми, та за допомогою eEPC та BPMN нотацій. Реалізовано серверну частину з використанням Spring. Клієнтська частина була розроблена за допомогою HTML, CSS, JS та шаблонного движка Thymeleaf.

В результаті роботи отримано програмне забезпечення для взаємодії волонтерів та благодійних організацій.

SUMMARY

Bachelor’s qualifying paper “The Development of the Communication Service for Volunteers and Charitable Organizations Based on Spring”: 65 pages, 23 figures, 1 table, 15 references, 4 supplements.

DATABASE, CHARITIES, CHARITABLE EVENTS, VOLUNTEERS, WEB APPLICATION, SOFTWARE ENGINEERING, SOFTWARE DEVELOPMENT, SERVICE.

The object of the study is the process of interaction between charitable organizations and volunteers within the framework of charitable projects.

The subject of the study is a set of theoretical, methodological and practical recommendations for the development of software for the interaction of two interested groups.

The goal of the work is to develop a service for connecting charitable organizations and volunteers: a system that will allow organizations to find and coordinate volunteers for various charitable events and projects.

Research methods – methods of collecting and analyzing software requirements, methods of modeling, designing, and designing software.

During the development of the program, an analysis of the subject area was carried out. An action plan was designed using a Gantt chart. Selected and designed system architecture using UML in the form of sets of case diagrams, classes, activities, as well as ER diagrams, and using eEPC and BPMN notations. The server part was implemented using Spring. The client side was developed using HTML, CSS, JS and the Thymeleaf template engine.

As a result of the work, software for the interaction of volunteers and charitable organizations was obtained.

ЗМІСТ

ВСТУП

У сучасному світі технології відіграють ключову роль у покращенні комунікації та сприянні у різних сферах суспільства. Однією з ключових сфер, де застосування технологій може значно підвищити ефективність та результативність, є благодійність. Наразі актуальною проблемою є нестача ефективних інструментів для координації та взаємодії між благодійними організаціями та волонтерами. У цьому контексті розробка сервісу для зв'язку та співробітництва стає важливим напрямом, що сприятиме оптимізації процесів благодійності та розширенню можливостей для залучення волонтерів.

Мета – розробити сервіс для зв'язку благодійних організацій та волонтерів: система, яка дозволить організаціям знаходити та координувати волонтерів для різноманітних благодійних заходів та проєктів.

Досягнення поставленої мети передбачає реалізацію наступних завдань:

- проаналізувати потреби благодійних організацій та волонтерів для виявлення їх потреб у платформі;
- дослідити функціональні потреби для ефективної комунікації у рамках благодійних заходів;
- спроєктувати інтерфейси та функціональні компоненти системи;
- розробити архітектуру сервісу на основі фреймворку Spring, що забезпечує надійність, масштабованість та зручність використання.

Об'єкт дослідження – процес взаємодії між благодійними організаціями та волонтерами у рамках благодійних проєктів.

Предмет дослідження – комплекс теоретичних, методичних та практичних рекомендацій щодо розробки програмного забезпечення для взаємодії двох зацікавлених груп.

Методи дослідження – методи збору та аналізу вимог до програмного забезпечення, методи моделювання, проєктування, конструювання

програмного забезпечення.

Для досягнення результату було обрано фреймворк Spring, оскільки він надає потужні можливості розробки додатків мовою Java. Spring дозволяє легко реалізувати масштабовану та гнучку архітектуру системи, забезпечуючи надійність та безпеку. Для зберігання даних була обрана реляційна система управління базами даних PostgreSQL. Взаємодія між серверною частиною та базою даних здійснюється за допомогою Spring Data JPA, що дозволяє ефективно керувати даними та мінімізувати кількість коду. Інтерфейс користувача буде реалізований з використанням HTML, CSS та JavaScript , а також шаблонизатора Thymeleaf.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку посилань та додатків. Загальний обсяг роботи становить 67 сторінки. Додатки викладено на 4 сторінках. У переліку посилань 15 використаних джерел.

У першому розділі розглядається базовий огляд предметної області, що становить ключову основу для розуміння актуальності та необхідності розробки системи для благодійності.

У другому розділі розглядаються ключові аспекти розробки системи благодійності, починаючи з технічного завдання та аналізу інструментів для розробки, та завершуючи проєктуванням системи та інтерфейсу, а також аналізом процесу реєстрації користувача та механізмів зворотного зв’язку та оцінки.

Третій розділ присвячено опису реалізації платформи. Опис управління подіями та їх історією у рамках благодійної платформи. У цьому розділі детально розглядається процес переміщення завершених подій на сторінку “History of events”, забезпечуючи їх збереження як архівні дані. Використання шаблонного двигуна. Також описуються механізми доступу до цієї інформації для різних типів користувачів, таких як волонтери та організації.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Базовий огляд предметної області

Благодійність існує в різних формах та під різними назвами вже багато тисячоліть. Соціальними проблемами опікувались релігійні організації, а в сучасному вигляді вона з’явилась в кінці XIX сторіччя завдяки Ендрю Карнегі, промисловцю шотландсько-американського походження, одному з перших філантропів, який встановив свого роду стандарти сучасної благодійності.

Сторіччя потому багато чого змінилося, з’явились нові форми благодійності. Тепер благодійністю зацікавились не лише окремі люди, а й цілі корпорації. Така форма благодійності отримала назву соціально відповідального бізнесу.

До того ж тепер волонтером може стати кожен, хто цього забажає, якщо пожертвує навіть невелику суму на одній з краудфандингових платформ задля допомоги у тій сфері, яка відгукується в серці донора, або ж допоможе іншим чином – талантом, не витрачаючи грошей [1].

На сьогоднішній день розробка системи для благодійності є не лише актуальним, а й невідкладним завданням, що має глобальне значення. Це зумовлено низкою факторів, які є ключовими у контексті сучасної соціальної дійсності.

По-перше, зростаюча кількість соціальних проблем та викликів, з якими стикаються різні верстви населення в нашій країні та в цілому по всьому світу, наголошує на необхідності більш ефективного та координованого підходу до благодійної діяльності. Економічні, екологічні, медичні та соціальні кризи вимагають широкомасштабної участі суспільства у підтримці нужденних.

По-друге, цифрова трансформація та розвиток інформаційних технологій надають унікальні можливості для створення інноваційних платформ та інструментів, які сприятимуть спрощенню процесів

благодійності. Впровадження сучасних технологій дозволяє покращити доступність та прозорість благодійних ініціатив, а також підвищити ефективність їх реалізації.

В умовах масштабних криз благодійні організації та волонтери відіграють ключову роль у наданні допомоги нужденним та підтримці вразливих груп населення.

Наразі Україна стикається з низкою соціальних проблем та викликів, пов’язаних із наслідками конфлікту на сході країни, економічними труднощами, медичним забезпеченням та іншими аспектами. Наявність ефективної системи благодійності дозволить забезпечити підтримку нужденним верствам населення, надати допомогу постраждалим від конфлікту та соціально вразливим групам.

Україна є країною, де цінності громадянського суспільства та волонтерства займають важливе місце у суспільному житті. Створення платформи для зв’язку благодійних організацій та волонтерів сприяє об’єднанню зусиль громадянських ініціатив та індивідуальних учасників для вирішення соціальних завдань та підтримки соціальних проєктів.

Використання сучасних інформаційних технологій як інструменту для покращення доступності та ефективності благодійної діяльності. Розвиток інформаційних платформ та онлайн-сервісів сприяє оптимізації процесів пошуку, координації та моніторингу благодійних проєктів, що значно спрощує їх реалізацію та контроль.

У контексті інтеграції України до європейської спільноти та прагнення до зміцнення громадянського суспільства розвиток системи для благодійності набуває стратегічного значення для формування культури соціальної відповідальності, добровольчості та взаємодопомоги.

Таким чином, розробка системи для благодійності в сучасних умовах є необхідним і важливим завданням, спрямованим на підвищення ефективності соціальної допомоги, покращення якості життя та формування гармонійного суспільства.

1.2 Вивчення потреб та очікувань благодійних організацій

Аналіз потреб та очікувань благодійних організацій є фундаментальною частиною дослідницького процесу при створенні та розвитку платформи для благодійності. Розробка та впровадження такої платформи потребує глибокого розуміння потреб та очікувань організацій, щоб забезпечити максимальну відповідність функціональності та можливостей платформи їх потребам.

Першою основною потребою благодійних організацій є ефективне керування волонтерськими ресурсами. Для досягнення цієї мети організаціям необхідні інструменти, які дозволяють швидко знаходити, обирати, координувати та відстежувати волонтерів. Такі інструменти повинні забезпечувати доступ до детальної інформації про волонтерів, їх навички, досвід роботи та доступність для участі в різних благодійних проєктах.

Друга ключова потреба полягає у наявності функціональності платформи для розміщення та управління благодійними проєктами. Благодійні організації повинні мати можливість легко та зручно розміщувати інформацію про свої проєкти, описувати їх цілі, завдання та очікувані результати, а також відстежувати прогрес виконання проєктів та взаємодіяти з волонтерами.

Третя потреба організацій пов’язана із забезпеченням зворотного зв’язку та взаємодії з волонтерами. Організації очікують від платформи можливості ефективного обміну інформацією з волонтерами, у тому числі надсилання повідомлень про нові проєкти, відгуки на заявки на участь та зворотний зв’язок у рамках реалізації проєктів. Для цього потрібні відповідні функції, такі як система повідомлень про нові проєкт и та відповіді на заявки, можливість надсилання повідомлень та коментарів, а також функції для проведення спілкування та координації роботи в рамках проєктів.

Четверта важлива потреба благодійних організацій полягає у забезпеченні високого рівня безпеки та прозорості на платформі. Організації очікують від платформи надійного захисту персональних даних,

конфіденційності інформації та можливості моніторингу діяльності на платформі. Для цього потрібні відповідні механізми та інструменти, такі як система шифрування даних, механізми автентифікації та авторизації, а також система моніторингу та аудиту дій користувачів на платформі.

Нарешті, особливості культурно-специфічних потреб вимагають урахування різноманітності благодійних організацій та їх діяльності при розробці та реалізації функціональності платформи. Це передбачає адаптацію платформи під конкретні потреби та очікування користувачів з огляду на їх культурні та соціальні особливості.

Загалом облік зазначених потреб та очікувань благодійних організацій є невід'ємною частиною процесу проєктування та розробки платформи для благодійності. Їхнє задоволення сприяє підвищенню ефективності діяльності організацій, покращенню взаємодії з волонтерами та забезпеченню якісного виконання благодійних проєктів.

1.3 Дослідження вимог потенційних волонтерів

Вивчення вимог потенційних волонтерів відіграє важливу роль у процесі розробки платформи для взаємодії між благодійними організаціями та волонтерами. Розуміння мотивації та очікувань потенційних волонтерів дозволяє ефективно формувати функціональність та особливості платформи, що сприяє залученню та утриманню активних учасників.

Першою важливою вимогою з боку потенційних волонтерів є можливість участі у благодійних заходах, які відповідають їхнім інтересам та цінностям. Волонтери шукають проєкти, які дозволять їм зробити свій внесок у вирішення соціальних проблем або допомогти тим, хто потребує підтримки. Вони прагнуть знайти проєкти, які матимуть для них особисте значення та зможуть стати джерелом задоволення від участі.

Другою важливою вимогою є можливість особистісного зростання та

розвитку. Багато волонтерів шукають проєкти, які нададуть їм можливість набути нових навичок та досвіду, розширити кругозір та познайомитися з новими людьми. Вони прагнуть використовувати участь у благодійних заходах як можливість для саморозвитку та розкриття свого потенціалу.

Третьою важливою вимогою є можливість соціального спілкування та встановлення нових зв'язків. Волонтери цінують можливість зустрічатися з однодумцями, обмінюватися досвідом та ідеями, а також працювати у команді над спільною метою. Для них важливо мати можливість спілкування та встановлення соціальних зв'язків як у рамках конкретного проєкту, так і загалом на платформі.

Четвертою важливою вимогою є можливість отримання особистого задоволення та благополуччя від участі у благодійних заходах. Волонтери прагнуть випробувати радість та задоволення від допомоги іншим людям та внесення свого внеску у спільне благо. Для них важливо мати можливість відчути особисте задоволення від досягнення конкретних результатів та позитивного впливу на життя інших людей.

Виходячи з вищезазначених вимог, розробка платформи для взаємодії між благодійними організаціями та волонтерами має враховувати різноманітність мотивацій та потреб потенційних волонтерів, забезпечуючи їм можливість вибору та задовольняючи їх очікування від участі у благодійних проєктах.

1.4 Технічне завдання

1.4.1 Найменування і область застосування

Програмний продукт, що розробляється, отримує найменування: «ConnectCharity».

Область застосування сервісу, що розробляється, охоплює широкий

спектр діяльності у сфері благодійності та волонтерства. Як важливий інструмент для координації та управління благодійними проєктами, платформа надає можливість ефективної взаємодії між благодійними організаціями та потенційними волонтерами.

Сервіс призначений для використання благодійними організаціями різного масштабу, включаючи як міжнародні великі фонди, так і місцеві неприбуткові організації.

Основною аудиторією сервісу є волонтери, які готові надати свій час та зусилля для підтримки благодійних та соціальних ініціатив. Сервіс надає їм можливість швидко та зручно знаходити цікаві їм проєкти, подавати заявки на участь, а також відстежувати свою активність та отримувати зворотний зв’язок від організацій.

Сервіс має потенціал для застосування в освітніх та соціальних програмах, спрямованих на підвищення обізнаності та навчання населення у галузі благодійності та соціальної відповідальності.

1.4.2 Технічні вимоги до програмного продукту

На початку роботи над системою були сформульовані наступні вимоги:

- реєстрація та профілі: користувачі можуть зареєструватися на платформі, створюючи свої профілі. При реєстрації можна вибрати два варіанти облікового запису: волонтер або благодійна організація;
- розміщення проєктів: організації можуть розміщувати інформацію про свої благодійні проєкти та заходи, до яких потрібні волонтери;
- заявки: волонтери можуть подати заявку на участь у проєктах, що відповідають їхнім інтересам та навичкам;
- історія проєктів: сторінка історій завершених проєктів;
- відгуки: на сторінці завершених проєктів волонтери та благодійні організації можуть поставити оцінку (зірки від 1 до 5) або залишити

коментар до минулої події;

- сторінка користувача: домашня сторінка користувача – вся інформація про обліковий запис знаходиться там;
- зміна інформації: користувачі можуть редагувати інформацію про себе;
- сторінка пошуку: волонтери можуть шукати заходи за такими критеріями (ключові слова з назви, місце, дата початку, дата закінчення, види волонтерства);
- система прийому заявок: благодійні організації можуть приймати чи відхиляти заявки – на сторінці заходу для благодійних організацій відображається список учасників;
- сторінка «Заходи»: волонтери можуть переглянути список заходів на сторінці «Заходи», який включатиме заплановані або події, що вже відбуваються – волонтери можуть брати участь у заходах від різних благодійних організацій.

Платформа надає три типи користувачів: гість, волонтер, благодійна організація. Гість (користувач, який не пройшов автентифікацію) матиме доступ лише до цільової сторінки платформи. Волонтери та організації мають доступ до повного функціоналу платформи. Платформа надає два різні способи реєстрації залежно від типу користувача: волонтер чи благодійна організація.

Реєстрація волонтерів:

- ім’я: (обов’язково);
- прізвище: (обов’язково);
- електронна пошта: (обов’язково);
- пароль: (обов’язково);
- повторити пароль: (обов’язково).

Реєстрація благодійних організацій:

- назва організації: (обов’язково);
- електронна пошта: (обов’язково);

- пароль: (обов’язково);
- повторити пароль: (обов’язково);
- опис організації: організації надають короткий опис своєї діяльності та місії.

Після заповнення обов’язкових полів та успішної перевірки полів волонтери та організації можуть завершити реєстрацію, після чого вони матимуть доступ до функціонала платформи. Вони будуть перенаправлені на головну сторінку. Автентифікація волонтера та організації здійснюється електронною поштою та паролем.

Після автентифікації (введення логіну та пароля) представник благодійної організації отримує доступ до її функціоналу в системі. Є меню для переміщення різними розділами платформи. Цей функціонал складається з наступних блоків:

- редагування даних профілю;
- публікація подій: реєстрація події шляхом введення необхідних даних;
- видалення вже створених подій;
- прийом чи відхилення заявок від волонтерів;
- переглянути список волонтерів, які взяли участь у заході;
- на сторінці «Історія подій» організації можуть переглядати минулі події;
- натискаючи на минулі події, організації можуть оцінити їх та/або залишити коментар.

Після автентифікації (введення логіну та пароля) волонтер отримує доступ до свого функціонала в системі. Цей функціонал складається з наступних блоків:

- редагування даних профілю;
- пошук заходів за такими критеріями (ключові слова з назви, місце проведення, дата початку, дата закінчення, види волонтерства);

- подати заявку на участь у заходах;
- волонтери можуть переглянути список заходів на сторінці «Заходи», який включатиме заплановані або заходи, що вже відбуваються;
- на сторінці «Історія подій» волонтери можуть переглядати минулі події;
- натискаючи на минулі події, волонтери можуть оцінити їх та/або залишити коментар.

1.4.3 Аналіз інструментів для розробки

Аналіз інструментів для розробки є важливим етапом у створенні платформи для взаємодії благодійних організацій та волонтерів. Цей процес включає вивчення різних технологій, фреймворків, інструментів і платформ, які будуть використані для розробки та реалізації функціональності системи. Мета даного аналізу полягає у виборі оптимального набору інструментів, який дозволить ефективно реалізувати завдання проєкту з огляду на його особливості та вимоги. У ході аналізу проводиться оцінка переваг та недоліків різних інструментів, їх відповідності поставленим цілям та вимогам, а також їх придатності для вирішення конкретних завдань проєкту.

HTML, CSS і JavaScript – це основні технології front-end розробки, що забезпечують створення вебінтерфейсу. Вони мають високу крос-браузерну сумісність і широку підтримку вебстандартів. HTML використовується для структурування контенту сторінки, CSS – для оформлення та стилізації елементів, а JavaScript – для створення інтерактивності та динамічної поведінки. Вартість створення власних вебсторінок з використанням HTML, CSS та JavaScript фактично дорівнює нулю [2].

Java – одна з найбільш поширених мов програмування, що широко застосовується для розробки back-end частини вебдодатків. Java дозволяє розробляти та розгортати програми Java на настільних комп’ютерах та

серверах. Java та компонентні технології пропонують багатий інтерфейс користувача, продуктивність, універсальність, мобільність і безпеку, необхідні сучасним додаткам [3].

Spring Framework – універсальний фреймворк із відкритим вихідним кодом для Java-платформи. Він надає широкий набір інструментів та функціональності, що спрощують розробку додатків та підвищують продуктивність розробників. Spring Framework заснований на принципах інверсії управління та впровадження залежностей, що дозволяє створювати програми, що легко підтримуються і тестуються [4].

Thymeleaf – сучасний серверний механізм шаблонів Java як для веб, так автономних середовищ. Основна мета Thymeleaf – впровадити елегантні природні шаблони у робочий процес розробки. HTML, який може коректно відображатись у браузерах, а також працювати як статичні прототипи, забезпечуючи більш тісне співробітництво у групах розробників. Завдяки модулям для Spring Framework, безлічі інтеграцій з інструментами та можливості підключення власних функцій Thymeleaf ідеально підходить для сучасної веброзробки [5].

PostgreSQL – об’єктно-реляційна система управління базами даних (ORDBMS), заснована на POSTGRES [6]. Вона має багатий набір функціональних можливостей, включаючи підтримку складних запитів, транзакцій, масштабованості та безпеки даних. Аналіз PostgreSQL включає оцінку його продуктивності, надійності, сумісності з іншими інструментами розробки та можливостей інтеграції з вебдодатком.

Maven – інструмент складання номер один, який використовується розробниками, і він існує вже понад десять років. Maven виділяється серед інших інструментів збирання завдяки своїй надзвичайно розширюваній архітектурі, побудованій на основі концепції угоди про конфігурацію. Фактично це зробило Maven де-факто інструментом управління та створення проєктів Java. Він широко використовується в багатьох проєктах Java з відкритим вихідним кодом у рамках Apache Software Foundation [7].

JUnit – популярний фреймворк для написання та запуску автоматизованих тестів на мові Java. Він забезпечує зручні засоби для тестування різних частин програми, включаючи окремі методи, класи та модулі. Незважаючи на привабливо простий API, інструмент має глибокі і добре продумані концепції [8].

Проведений аналіз інструментів для розробки вебдодатку підтверджує, що обраний стек технологій є обґрунтованим та ефективним для досягнення поставленої мети. Загалом, обраний стек технологій забезпечує необхідну функціональність та інструменти для створення сучасного та професійного вебдодатку, здатного ефективно вирішувати поставлені завдання.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Етапи створення сервісу

Як інструмент для планування та управління часом проєкту використовувалася діаграма Ганта. Діаграма Ганта є формою горизонтальної гістограми. Оскільки горизонтальна вісь відображає дати початку та завершення завдань і дій проєкту, вона може надати візуальний і надзвичайно цінний огляд проєкту [9]. Вона буде складена з урахуванням усіх етапів створення сервісу та використовуватиметься для відстеження прогресу та контролю термінів виконання робіт.

Розглянемо основні етапи створення сервісу.

Збір вимог:

- визначення та аналіз вимог до проєкту;
- формалізація вимог у документі вимог до проєкту (ТЗ);
- затвердження теми та вимог із науковим.

Архітектура та дизайн:

- проєктування загальної архітектури програми;
- вибір технологічного стека для реалізації проєкту;
- розробка дизайну інтерфейсу користувача.

Розробка:

- створення базової функціональності згідно з вимогами;
- розробка модулів для основних функціональних блоків;
- реалізація взаємодії між компонентами програми;
- оптимізація продуктивності та обробка помилок.

Тестування:

- планування тестових сценаріїв та створення тест-кейсів;
- проведення модульного та інтеграційного тестування;

– налагодження та виправлення виявлених помилок.

Впровадження:

– підготовка середовища для розгортання програми;

– встановлення та конфігурація програми на цільовому сервері;

– проведення фінального тестування перед запуском у продакшн.

Для реалізації діаграми Ганта використовувалась програма ProjectLibre. ProjectLibre – безкоштовний аналог Microsoft Project, розроблений для управління проєктами. Оскільки ProjectLibre програмується на Java, він також доступний для всіх поширених операційних систем.: Microsoft Windows, Linux, Mac OS X [10].

На рисунку 2.1 зображено заповнену таблицю завдань у програмі у якій зазначені:

– номер завдання;

– назва задачі;

– початок завдання;

– закінчення завдання.

	Название	Продолжитель...	Начало	Окончание
1	**ПРОЕКТ**	87 дней?	**08.01.24 8:00**	**07.05.24 17:00**
2	Початок проекту	1 день?	08.01.24 8:00	08.01.24 17:00
3	**Збір вимог**	8 дней	**08.01.24 8:00**	**17.01.24 17:00**
4	Визначення та аналіз вимог до проекту	3 дней	08.01.24 8:00	10.01.24 17:00
5	Формалізація вимог у документі вимог до проекту (ТЗ)	4 дней	11.01.24 8:00	16.01.24 17:00
6	Затвердження теми та вимог із науковим керівником	1 день	17.01.24 8:00	17.01.24 17:00
7	**Архітектура та дизайн**	10 дней	**11.01.24 8:00**	**24.01.24 17:00**
8	Проектування загальної архітектури програми	6 дней	17.01.24 8:00	24.01.24 17:00
9	Вибір технологічного стека для реалізації проекту	3 дней	17.01.24 8:00	19.01.24 17:00
10	Розробка дизайну інтерфейсу користувача	4 дней	11.01.24 8:00	16.01.24 17:00
11	**Розробка**	68 дней	**22.01.24 8:00**	**24.04.24 17:00**
12	Створення базової функціональності згідно з вимогами	20 дней	22.01.24 8:00	16.02.24 17:00
13	Розробка модулів для основних функціональних блоків	20 дней	19.02.24 8:00	15.03.24 17:00
14	Реалізація взаємодії між компонентами програми	16 дней	18.03.24 8:00	08.04.24 17:00
15	Оптимізація продуктивності та обробка помилок	12 дней	09.04.24 8:00	24.04.24 17:00
16	**Тестування**	8 дней	**19.02.24 8:00**	**28.02.24 17:00**
17	Планування тестових сценаріїв та створення тест-кейсів	5 дней	19.02.24 8:00	23.02.24 17:00
18	Проведення модульного та інтеграційного тестування	5 дней	19.02.24 8:00	23.02.24 17:00
19	Налагодження та виправлення виявлених помилок	3 дней	26.02.24 8:00	28.02.24 17:00
20	**Впровадження (deploy)**	8 дней	**25.04.24 8:00**	**06.05.24 17:00**
21	Підготовка середовища для розгортання програми	3 дней	25.04.24 8:00	29.04.24 17:00
22	Встановлення та конфігурація програми на цільовому сервері	3 дней	30.04.24 8:00	02.05.24 17:00
23	Проведення фінального тестування перед запуском у продакшн	2 дней	03.05.24 8:00	06.05.24 17:00
24	Проект завершено	1 день?	07.05.24 8:00	07.05.24 17:00

Рисунок 2.1 – Таблиця завдань у ProjectLibre

У цьому списку частина задач є звичайними завданнями. Завдання, виділені напівжирним шрифтом, – сумарні: кожна з них є сукупністю кількох підзадач, об’єднаних спільною метою. Сумарні завдання допомагають структурувати проєкт та забезпечують більш чітке уявлення про його прогрес. Вони полегшують керування проєктом, тому що дозволяють фокусуватися на ключових етапах виконання роботи. Під час відстеження виконання завдань сумарні завдання можуть бути розгорнуті для детального перегляду або згорнуті для зручності:

- задача це один із заходів, спрямованих на досягнення мети проєкту – основними параметрами задачі є дати початку та завершення, тривалість, трудомісткість, а також види та кількість ресурсів, необхідних для її виконання; кожна задача в межах проєкту повинна мати унікальне ім’я;
- сумарна задача це задача, що складається із задач нижчого рівня;
- задачі-віхи, контрольні точки, для яких тривалість дорівнює 0 на діаграмі Ганта зображуються чорними крапками.

Тривалість проєкту орієнтовно становить 87 днів.

Для побудови календарного плана-графіка робіт необхідно логічне зв’язування задач. Існує 4 типи логічного зв’язку між задачами:

- «закінчення-початок» (ЗП, FS): задача-послідовник може розпочатися лише після закінчення задачі-попередника;
- «початок-початок» (ПП, SS): задача-послідовник може розпочатися лише після того, як розпочнеться задача-попередник;
- «закінчення-закінчення» (ЗЗ, FF): задача-послідовник може завершитися лише після завершення задачі-попередника;
- «початок-закінчення» (ПЗ, SF): задача-послідовник може завершитися лише після того, як розпочнеться задача-попередник.

На рисунку 2.2 зображено готову діаграму Ганта з усіма зв’язками.

Загальна тривалість проєкту може бути скорочена за рахунок зменшення

часу виконання задач, що лежать на критичному шляху. Критичний шлях – це послідовність задач, що має максимальну тривалість та найменшу величину резерву часу виконання. Задачі, що лежать на критичному шляху, називаються критичними. Затримка у виконанні якоїсь задачі на критичному шляху призведе до порушення планового терміну виконання проєкту. Це дозволяє або зрушувати їх у часі або збільшувати термін виконання.

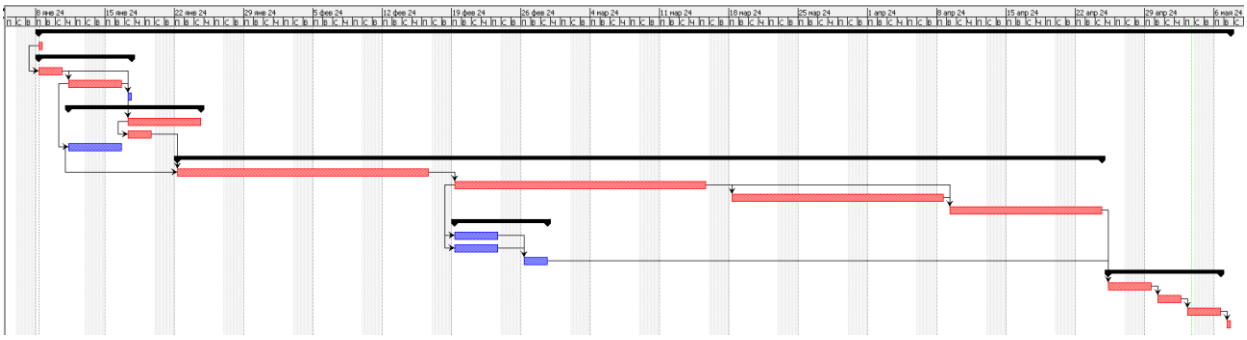
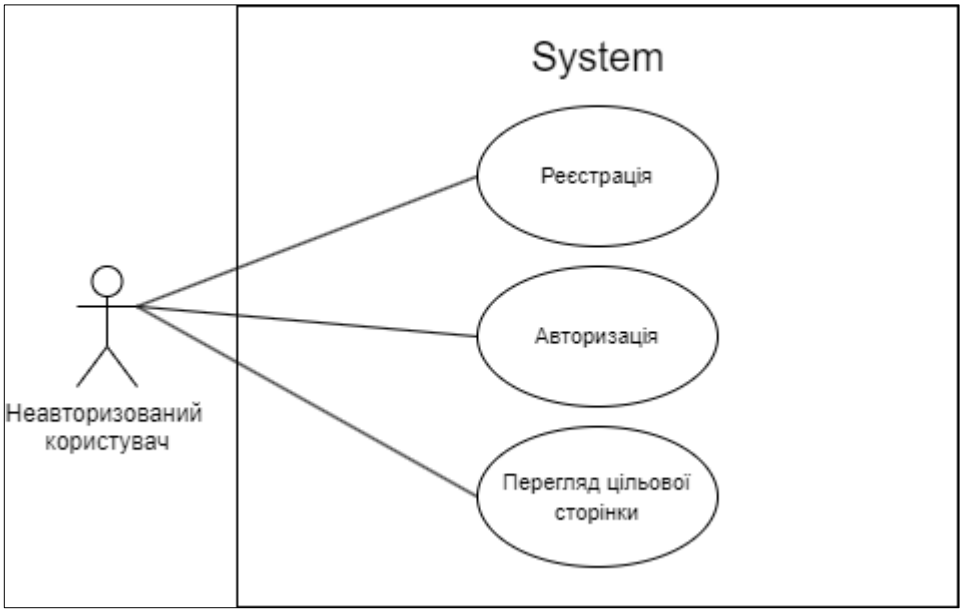


Рисунок 2.2 – Діаграма Ганта у ProjectLibre

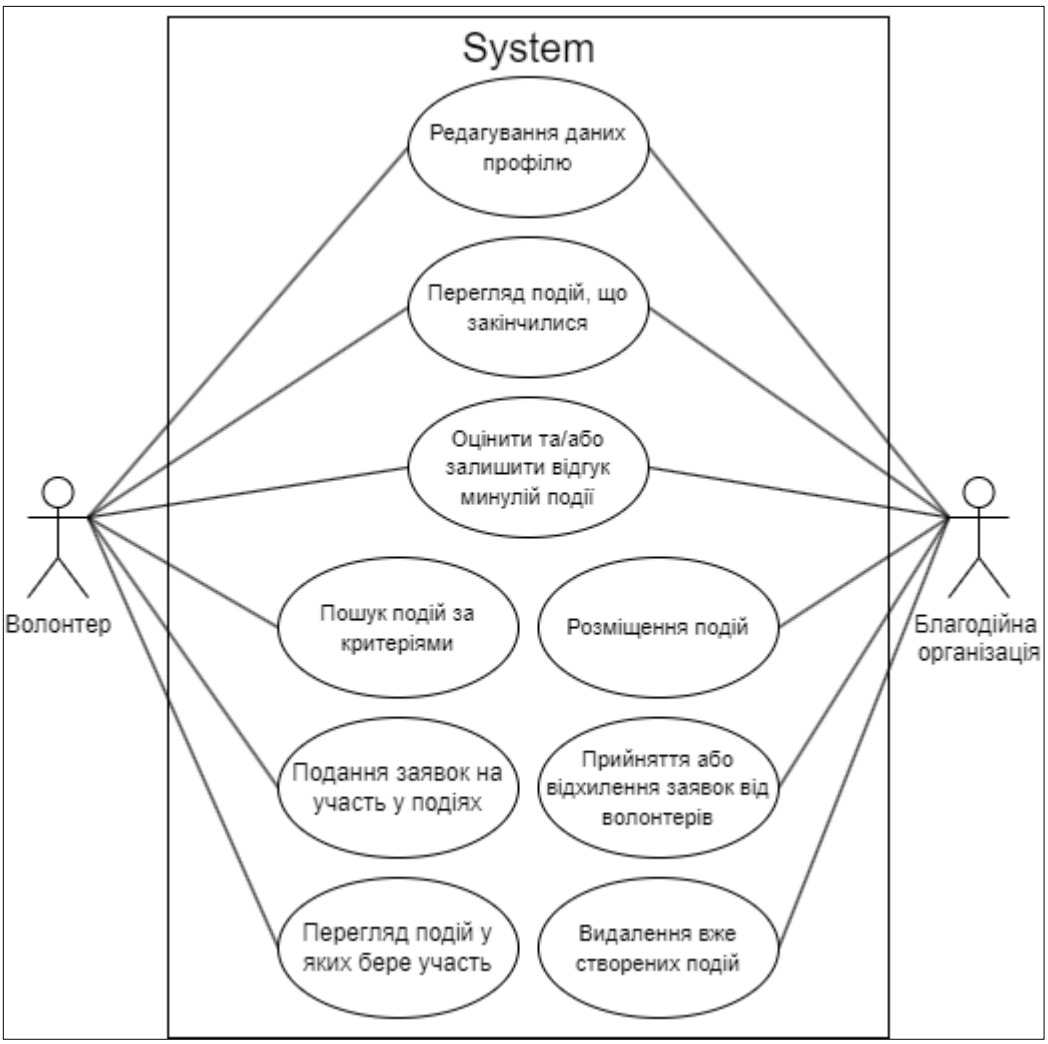
2.2 Проєктування системи

Уніфікована мова моделювання (UML) дозволяє описувати системи словами та зображеннями. Примітними є різні графічні діаграми, особливо така як діаграма варіантів використання [11]. Проєктування системи є ключовим етапом розробки, на якому визначаються функціональні можливості системи, її архітектура та основні компоненти. На рисунку 2.3 зображено діаграма прецедентів, яка ілюструє основні дії, які виконуються різними типами користувачів у системі.

Діаграма прецедентів (use case diagram) відображає взаємодію трьох типів користувачів із системою: неавторизований користувач, волонтер та благодійна організація. Розглядаються основні функції та взаємодії для кожного з цих учасників.



а) Неавторизований користувач



б) Волонтер та благодійна організація

Рисунок 2.3 – Діаграма прецедентів

У таблиці 2.1 представлені актори та прецеденти діаграми прецедентів.

Діаграма прецедентів є важливим інструментом проєктування, який допомагає визначити основні функції та взаємодії в системі. Аналізуючи цю діаграму, можна чітко уявити ролі та дії різних типів користувачів, що є основою для подальшої розробки системи та її компонентів.

Таблиця 2.1 – Актори та прецеденти

Актор	Прецедент
Неавторизований користувач	– реєстрація: процес створення облікового запису для отримання доступу до системи; – авторизація: процес входу в систему з використанням облікових даних; – перегляд цільової сторінки: можливість ознайомлення із головною сторінкою системи.
Волонтер	– перегляд цільової сторінки: можливість ознайомлення із головною сторінкою системи; – редагування даних профілю: можливість зміни особистої інформації облікового запису; – перегляд подій, що завершилися: доступ до інформації про події, в яких волонтер брав участь; – оцінка та/або залишення відгуку про минулу подію: можливість залишити відгук та оцінити захід, у якому волонтер брав участь; – пошук подій за критеріями: можливість пошуку за критеріями подій за заданими параметрами; – подання заявок на участь у подіях: можливість подання заявки на участь у благодійному заході; – перегляд подій, в яких бере участь волонтер: можливість відстеження поточних та майбутніх заходів, в яких волонтер бере участь.

Продовження таблиці 2.1.

Актор	Прецедент
Благодійна організація	перегляд цільової сторінки: можливість ознайомлення із головною сторінкою системи; редагування даних профілю: аналогічно до волонтерів, організації можуть редагувати свою профільну інформацію; перегляд завершених подій: можливість отримання інформації про завершені заходи; оцінка та/або залишення відгуку про минулу подію: можливість залишати відгуки та оцінки про проведені заходи; розміщення подій: створення та розміщення інформації про нові благодійні заходи; прийняття або відхилення заявок від волонтерів: можливість керування заявками на участь від волонтерів, включаючи їхнє схвалення або відхилення; видалення вже створених подій: можливість видалення застарілих чи скасованих заходів.

Створення UML-діаграми класів є дуже важливим етапом при проєктуванні системи, що забезпечує структуроване уявлення основних сутностей та їх взаємодій. На рисунку 2.4 зображено діаграма класів.

Система включає такі основні компоненти: події, заявки, волонтери, благодійні організації та зворотній зв’язок/оцінки.

Клас “Event” є основною одиницею організації заходів. Він включає такі атрибути, як ідентифікатор події (event_id), назва події (event_name), категорія (category), дата початку (start_date) та закінчення (end_date), а також місцезнаходження (location). Основні методи, пов’язані з класом “Event”, включають додавання нової події (AddEvent) та видалення існуючої події

(DeleteEvent). Ці методи забезпечують управління життєвим циклом подій, починаючи від створення до видалення.

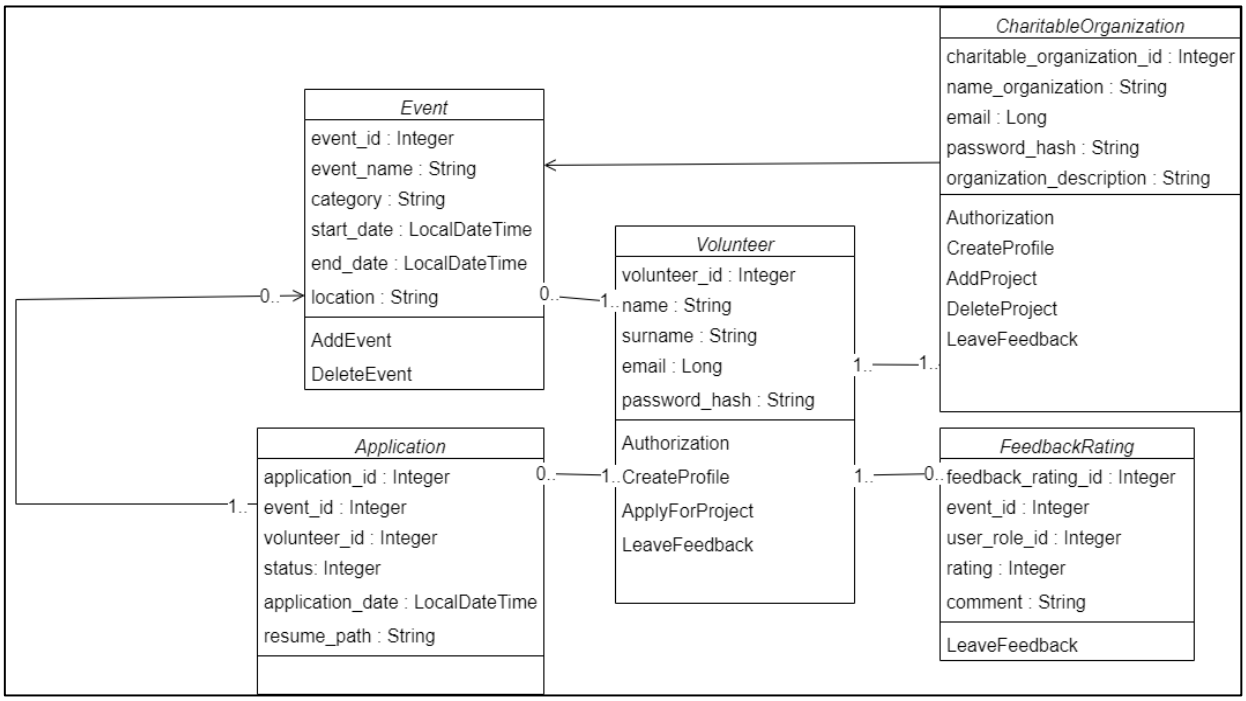


Рисунок 2.4 – Діаграма класів

Клас “Volunteer” описує волонтерів, які беруть участь у заходах. Основні атрибути включають ідентифікатор волонтера (volunteer_id), ім’я (name), прізвище (surname), електронну пошту (email) та хеш пароля (password_hash). Методи, пов’язані з волонтерами, включають авторизацію (Authorization), створення профілю (CreateProfile), подання заявок на участь у проєктах (ApplyForProject) та залишення відгуків (LeaveFeedback). Цей клас відіграє ключову роль у системі, оскільки дозволяє керувати інформацією про волонтерів та їхню активність.

Клас “CharitableOrganization” представляє благодійні організації, які створюють та керують подіями. Атрибути цього класу включають ідентифікатор організації (charitable_organization_id), назву організації (name_organization), електронну пошту (email), хеш пароля (password_hash) та опис організації (organization_description). Методи, пов’язані з цим класом, включають авторизацію (Authorization), створення профілю (CreateProfile),

додавання проєкту (AddProject), видалення проєкту (DeleteProject) та залишення відгуків (LeaveFeedback). Цей клас забезпечує функціональність, необхідну управління організацією та її проєктами.

Клас “Application” є заявками волонтерів на участь у подіях. Він включає такі атрибути, як ідентифікатор заявки (application_id), ідентифікатор події (event_id), ідентифікатор волонтера (volunteer_id), статус заявки (status), дату подання заявки (application_date) та шлях до резюме (resume_path).

Клас “FeedbackRating” використовується для управління відгуками та рейтингами, що залишаються користувачами. Основні атрибути включають ідентифікатор відгуку/рейтингу (feedback_rating_id), ідентифікатор події (event_id), ідентифікатор ролі користувача (user_role_id), рейтинг (rating) та коментар (comment). Метод LeaveFeedback дозволяє користувачам залишати відгуки, що сприяє підвищенню прозорості та якості заходів, що проводяться.

Класи у системі пов’язані множинними асоціаціями, що відбивають їх взаємодії. Наприклад, клас “Event” пов’язаний із класом “Application” асоціацією «один до багатьох», що означає, що одна подія може мати безліч заявок. Аналогічно, клас “Volunteer’ пов’язаний із класом “Application” асоціацією «один до багатьох», що дозволяє одному волонтеру подавати заявки на участь у кількох подіях.

Клас “FeedbackRating” пов’язаний з класами “Event” і “User” (де User представляє як “Volunteer”, так і “CharitableOrganization”) асоціаціями «один до багатьох», що дозволяє одній події та одному користувачеві мати безліч відгуків та рейтингів. Ця взаємодія забезпечує можливість комплексного оцінювання заходів та користувачів.

Діаграма класів дає структурований підхід до проєктування системи благодійної платформи. Вона визначає основні сутності та їх взаємозв’язку, що дозволяє ефективно керувати даними та функціональністю системи. Такий підхід забезпечує високу гнучкість та розширюваність, що особливо важливо для систем, що потребують масштабованості та можливості інтеграції нових функцій. Проєктування на основі UML-діаграм класів сприяє чіткому

розумінню вимог та покращує процес розробки, забезпечуючи якісне та надійне програмне забезпечення.

Крім того, використання діаграми класів дозволяє розробникам візуалізувати архітектуру системи, що сприяє більш точному та ефективному розподілу завдань серед команди. Цей підхід забезпечує єдине бачення проєкту та узгодженість дій на всіх етапах розробки.

2.3 Структура бази даних

Структура бази даних є фундаментальною частиною будь-якої інформаційної системи, що забезпечує зберігання, керування та доступ до даних. У цьому проєкті використовується реляційна база даних, спроєктована за допомогою ER-діаграми (діаграми «сутність-зв’язок»), що ілюструє логічні зв’язки між різними компонентами системи.

У контексті реляційних баз даних зв’язок між двома таблицями має дві цікаві потужності: потужність – кількість елементів – пов’язаних з першою таблицею та потужність – як ви здогадалися, кількість елементів – пов’язаних з другою таблицею [12]. В даному випадку основні сутності включають волонтерів, користувачів, заходи, заявки на участь, організації, ролі, типи волонтерства та місця розташування. Ці сутності є основними об’єктами, з якими система працює, і в яких зберігаються всі необхідні дані. На рисунку 2.5 зображено ER-діаграма.

Кожна сутність характеризується набором атрибутів, що описують її властивості. Наприклад, для сутності «волонтер» атрибутами є ім’я, прізвище та ідентифікатор, що унікально визначає кожен запис. Ці атрибути допомагають ідентифікувати та керувати даними, пов’язаними з конкретними об’єктами.

Взаємозв’язки між сутностями визначають як дані однієї сутності пов’язані з даними іншої. Наприклад, зв’язок між волонтерами та заходами

виражений через таблицю заявок на участь, яка дозволяє відстежувати, які волонтери подали заявки на якісь заходи, а також статус цих заявок. Такий зв’язок забезпечує цілісність даних та можливість ефективного пошуку та фільтрації інформації.

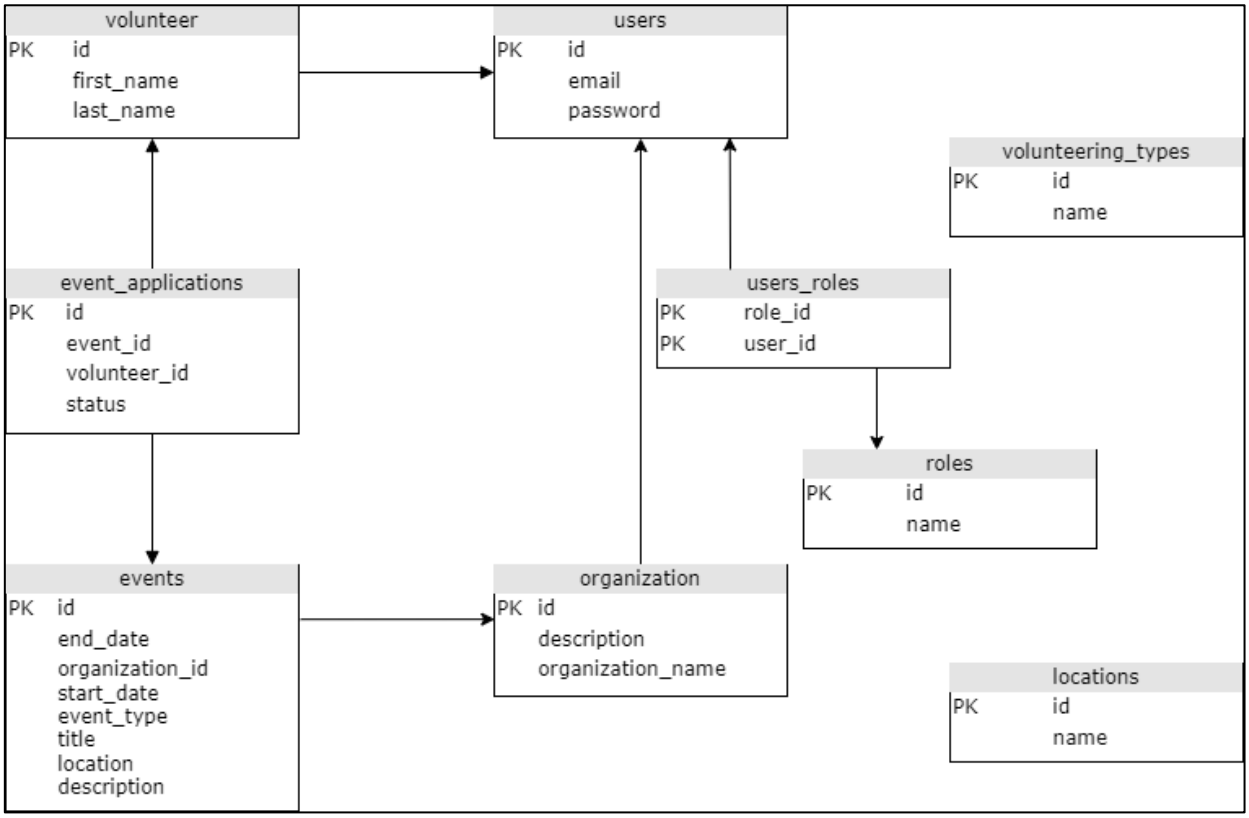


Рисунок 2.5 – ER-діаграма

Для управління користувачами та його ролями використовується асоціативна таблиця, що дозволяє присвоювати користувачам різні ролі. Це забезпечує гнучкість в управлінні доступом та розподіл обов’язків серед користувачів системи. Ролі можуть включати, наприклад, адміністраторів, модераторів та звичайних користувачів, що дозволяє розмежовувати рівні доступу та функціональні можливості. Розташування заходів також організовано в окрему сутність, що спрощує управління та дозволяє легко оновлювати інформацію про місця проведення заходів без необхідності зміни даних про самі заходи.

Структура бази даних у цьому проєкті була ретельно спроєктована для

забезпечення ефективного управління даними та їхньої цілісності. Чітке визначення сутностей та їх взаємозв’язків дозволяє системі легко масштабуватися і адаптуватися до вимог, що змінюються. Використання реляційної бази даних та системи управління PostgreSQL гарантує високу продуктивність та надійність, що є критично важливим для успішної реалізації та експлуатації системи волонтерства. Таким чином, структура бази даних відіграє ключову роль у забезпеченні функціональності та стійкості всієї системи.

2.4 Аналіз процесу реєстрації користувача

Реєстрація користувача у вебдодатках є процесом, який дозволяє новим користувачам створювати облікові записи для доступу до функціональності програми. Цей процес включає заповнення реєстраційної форми, введення необхідних персональних даних, таких як ім’я, адресу електронної пошти і пароль, і підтвердження створення облікового запису.

Нотації – це набір графічних елементів, які використовуються для відображення елементів системи. Простіше кажучи, нотації визначають правила, які ми використовуємо для визначення процесів, операцій і подій у схемі та зв’язуємо їх один з одним.

Абревіатура eEPC означає розширений ланцюг процесів, керований подіями. Подія «створює» операцію (функцію), яка, будучи своїм завданням, «створює» наступну подію [13].

Під час реєстрації користувач має можливість вибрати між створенням профілю «Волонтер» або «Благодійна організація». Цей вибір виключає інший варіант, тому процес наступної реєстрації розгалужується відповідно до вибраного типу облікового запису.

При виборі профілю «Волонтер» користувачеві пропонується заповнити реєстраційну форму, що включає поля для вказівки імені, прізвища, логіна,

адреси електронної пошти, пароля та його підтвердження. Після заповнення форми починається процес валідації даних, що включає перевірку коректності інформації, що вводиться. Система перевіряє унікальність логіну та адреси електронної пошти, а також відповідність пароля заданим критеріям безпеки. Після успішної перевірки даних створюється запис у базі даних нового користувача. Надалі волонтер може доповнити свій профіль, вказавши навички, інтереси та переваги, що полегшить пошук відповідних заходів.

При виборі профілю «Благодійна організація» процес реєстрації аналогічний попередньому, проте користувачеві пропонується заповнити форму, що містить поля для вказівки назви організації, логіну, адреси електронної пошти, пароля та його підтвердження. Крім того, можуть бути запитані додаткові відомості, такі як опис діяльності організації та контактна інформація. Після заповнення та валідації даних створюється запис у базі даних для нового користувача. Організація може також додати інформацію про свої проєкти та заходи, щоб залучити волонтерів. Таким чином, система забезпечує індивідуалізований підхід до реєстрації різних типів користувачів, що сприяє ефективній взаємодії волонтерів та благодійних організацій.

Після реєстрації користувачі можуть увійти до своїх облікових записів, щоб керувати профілем і доступними функціями. Волонтери отримують доступ до пошуку та перегляду доступних заходів, а організації можуть створювати та редагувати події. Важливо відзначити, що система передбачає можливість оновлення та зміни профілю у будь-який момент.

На рисунку 2.6 зображено нотація eEPC для графічного опису реєстрації користувача на сайті.

Далі, після завершення процесу реєстрації, проводиться хешування пароля для забезпечення безпеки зберігання даних користувача. Після цього користувач перенаправляється на головну сторінку вебпрограми, де він може розпочати використання всіх доступних функцій.

Таким чином, аналіз процесу реєстрації користувача дозволяє визначити основні кроки та етапи цього процесу, забезпечуючи ефективну та безпечну взаємодію користувача з вебдодатком.

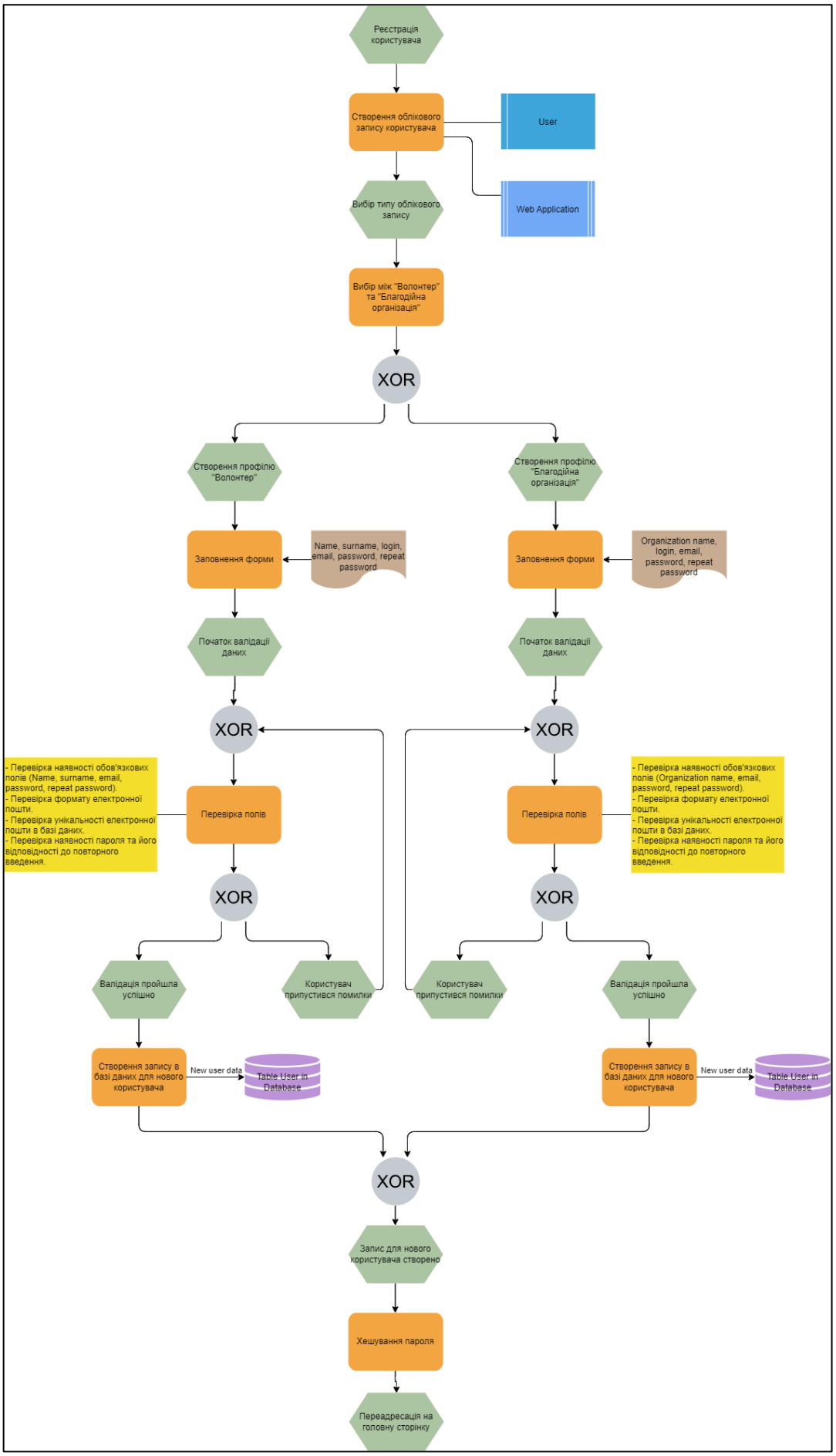


Рисунок 2.6 – Нотація eEPC реєстрації користувача

2.5 Механізми зворотного зв’язку та оцінювання

Механізм зворотного зв’язку та оцінювання в контексті нотації BPMN можна розглядати як послідовність дій, спрямованих на забезпечення ефективного взаємодії між користувачами та сервісами.

Спочатку BPMN була розроблена Ініціативою управління бізнес-процесами (BPMI), консорціумом, що складається в основному з компаній-розробників програмного забезпечення. Спочатку метою було надати графічне позначення описів процесів, виражене на BPML (мові моделювання бізнес-процесів). Діаграма процесу або співробітництва – тип діаграми, що найчастіше використовується. Специфікація BPMN пояснює різні елементи нотації як усно, а й визначає їх у метамоделі. Метамодель документується на діаграми класів UML, які графічно показують особливості різних конструкцій BPMN та їх взаємозв’язку [14]. User Service в свою чергу отримує токен та починає обробку отриманих даних. Після отримання токену, програма розшифровує його та проводить ряд перевірок, включаючи перевірку підпису. Після успішної перевірки, система отримує необхідні дані про користувача з бази даних та переходить до наступних етапів обробки. На рисунку 2.7 зображено нотація BPMN 2.0 для графічного опису зворотного зв’язку користувачів та виставлення рейтингу на сайті.

Механізм зворотного зв’язку та оцінювання, відповідно до нотації BPMN, що включає в себе послідовність дій, спрямованих на забезпечення ефективної взаємодії між користувачами та сервісами. Після завершення події, система переходить до перевірки авторизації користувача та відправлення токену. Користувачеві пропонується авторизуватися на сайті, і після прийняття рішення, система перевіряє авторизацію користувача. В разі позитивного результату, система надсилає токен користувачеві для подальшого використання в сервісі.

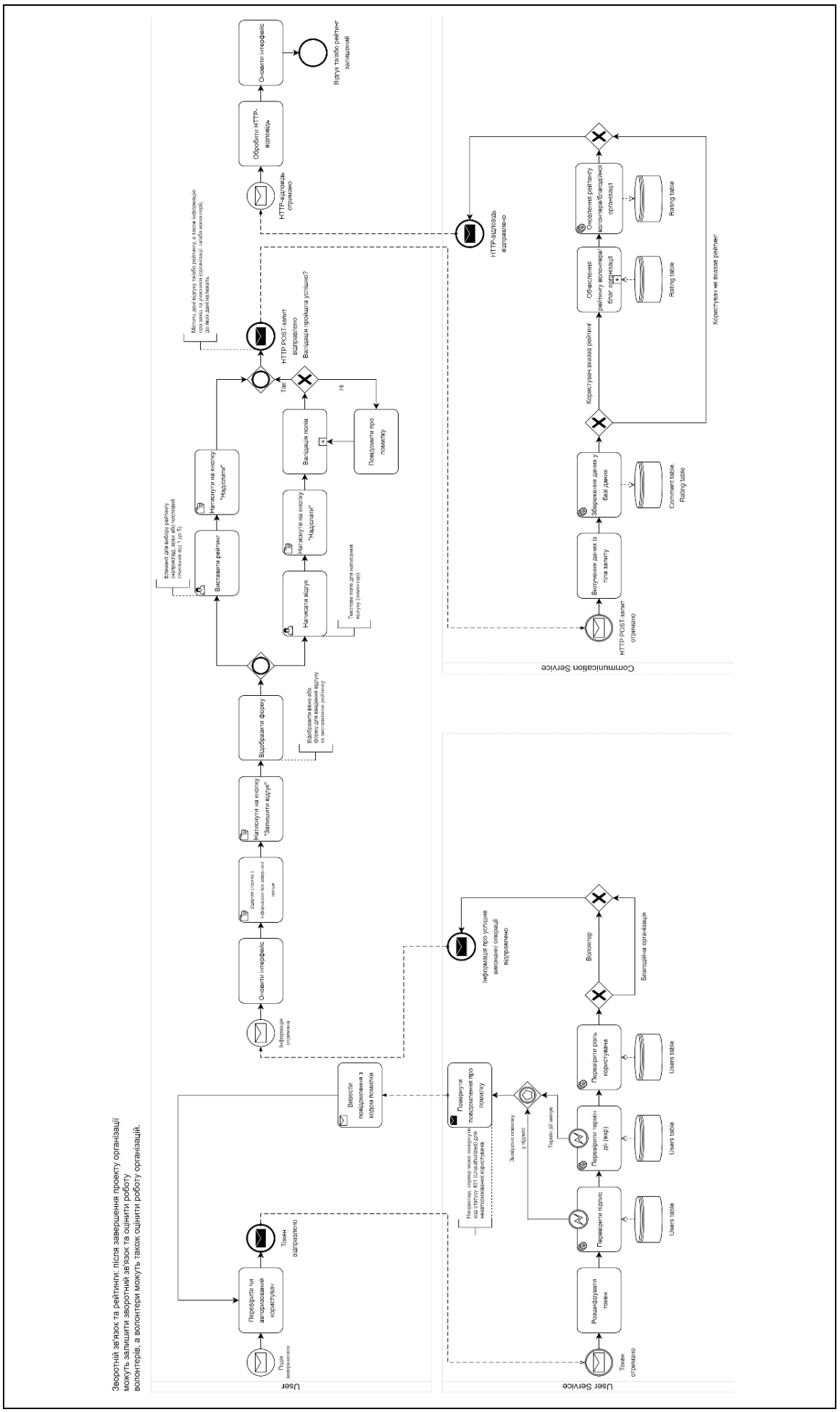


Рисунок 2.7 – Нотація BPMN 2.0 зворотній зв’язок та рейтинги

User Service в свою чергу отримує токен та починає обробку отриманих даних. Після отримання токену, програма розшифровує його та проводить ряд перевірок, включаючи перевірку підпису. У разі виявлення помилок або невідповідностей, служба повідомляє про це користувача та відправляє його на перевірку авторизації знову. Після успішної перевірки, система отримує необхідні дані про користувача з бази даних та переходить до наступних етапів обробки.

На наступному етапі, програма перевіряє токен на наявність в базі даних та проводить аналіз інших параметрів, таких як строк дії токену та роль користувача. В разі невідповідності або помилок, користувачу повідомляється про це, а потім йде переадресація на іншу сторінку для уточнення даних. У разі успішного завершення всіх перевірок, програма отримує необхідну інформацію про користувача з бази даних та переходить до наступних етапів обробки.

Користувач отримує інформацію про результати перевірок та може здійснити подальші дії відповідно до отриманої відповіді. Після отримання інформації, інтерфейс оновлюється та користувач має можливість залишити відгук чи виставити рейтинг минулому благодійному заходу.

Після отримання відповідних відгуків та/або рейтингів від користувача, система переходить до обробки цих даних. Вона зберігає відгуки та рейтинги в базі даних та оновлює відповідні таблиці. Потім виконується ісключаючий вибір, де визначається, чи був вказаний рейтинг користувачем. У випадку, якщо користувач не вказав рейтинг, система продовжує роботу та обробляє відповідні відгуки.

У протилежному випадку, система розраховує та оновлює рейтинг волонтера або благодійної організації.

Кінцевий етап механізму включає обробку HTTP-відповіді та оновлення інтерфейсу користувача, після чого відзначається успішне залишення відгуку та/або рейтингу. Таким чином, механізм зворотного зв’язку та оцінювання

створює повний цикл взаємодії між користувачами та системою, спрямований на покращення якості обслуговування та розвиток спільноти.

У межах аналізу механізму зворотного зв’язку та оцінювання була зроблена діаграма для наочної демонстрації взаємороботи серверної частини програми та інтерфейсу користувача, що відображає послідовність дій та взаємодій у процесі збору відгуків та оцінювання взаємодії між користувачами та системою. Даний тип діаграми широко використовується для моделювання бізнес-процесів та дозволяє структурувати та візуалізувати складні процеси.

У контексті обраної функціональності, діаграма активностей демонструє послідовність операцій та рішень, які виконуються у процесі збору відгуків та оцінювання користувачів. Починаючи з початкової події, такої як завершення взаємодії користувача з системою, діаграма відображає послідовність кроків, необхідних для виконання цього процесу.

Перший крок включає вибір між виставити рейтинг чи написати відгук. У разі написання відгуку на стороні клієнта йде валідація полів. Якщо користувач вибирає рейтинг, то після цього йде відправка HTTP POST-запиту.

Після успішного вилучення даних із тіла запиту та збереження їх у базі даних наприкінці відправляється HTTP-відповідь на клієнтську частину. Де після обробки відповіді оновлюється інтерфейс.

На рисунку 2.8 зображено діаграма активностей для графічного опису зворотного зв’язку користувачів та виставлення рейтингу на сайті.

У контексті діаграми активностей кожен елемент діаграми відповідає певній операції або кроку в процесі, а стрілки вказують на послідовність виконання цих операцій. Діаграма дозволяє візуально уявити процес та взаємозв’язки між його складовими, що полегшує розуміння та аналіз. Цей метод візуалізації сприяє виявленню потенційних покращень та оптимізації процесів. Він також дозволяє ефективно комунікувати складні процедури між членами команди, що сприяє узгодженості та точності під час виконання завдань.

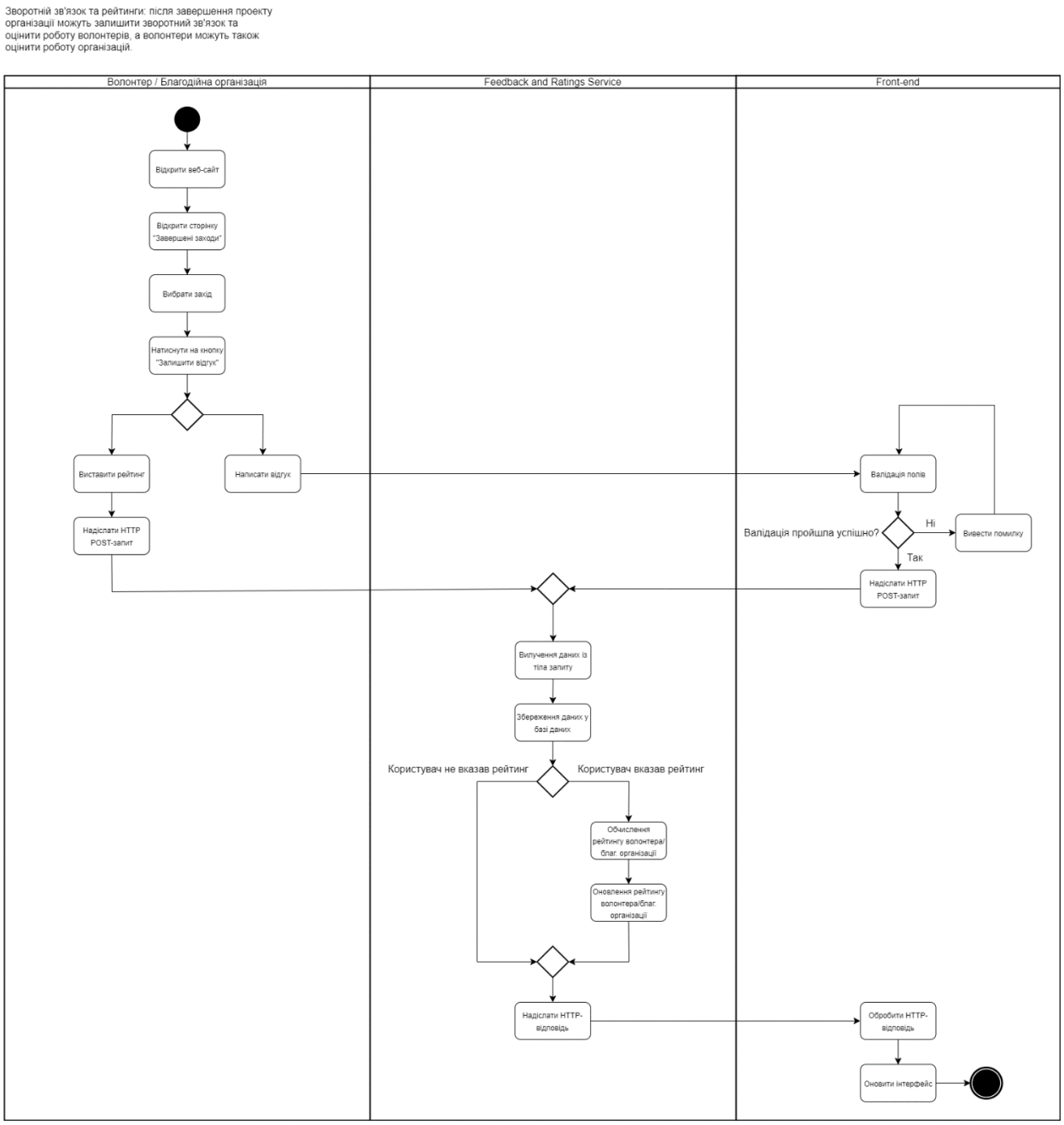


Рисунок 2.8 – Діаграма активностей зворотного зв’язку та рейтинги

2.6 Проєктування інтерфейсу

Проєктування інтерфейсу є критично важливим етапом розробки будь-якої вебпрограми, включаючи платформу для зв’язку благодійних організацій та волонтерів. Цей процес починається зі створення каркасу сторінок, що дозволяє наочно уявити структуру та функціональні елементи інтерфейсу перед початком роботи над його візуальною складовою.

На першому етапі розроблено каркас (Wireframe) сторінок із використанням онлайн-сервісу Moqups. Каркас є спрощеною схемою інтерфейсу, в якій показані основні елементи та їх розташування на сторінках. Цей етап є важливим для визначення логіки взаємодії користувача з системою, а також для виявлення можливих проблем на ранній стадії розробки. Каркас включає розміщення кнопок, полів введення, меню навігації та інших функціональних елементів, необхідних для виконання завдань користувачами.

Є три найбільш поширені методи, які матеріалізуються в таких результатах, як каркаси, карти сайту та персони. Знання цих методів та правильний час для використання кожного з них – одна з найбільш затребуваних навичок серед UX-дизайнерів [15].

Створення каркасів дозволяє команді розробників та зацікавленим сторонам краще зрозуміти, як користувачі взаємодіятимуть із платформою. Цей метод також допомагає ідентифікувати потенційні поліпшення в досвіді користувача і забезпечити, що кінцевий продукт буде інтуїтивно зрозумілим і зручним. Візуалізація структури інтерфейсу на ранніх стадіях дозволяє заощадити час та ресурси, забезпечуючи гладкіший перехід до етапу деталізованого дизайну та реалізації.

Цей етап закладає міцну основу для подальших етапів розробки, гарантуючи, що кінцевий продукт відповідатиме очікуванням користувачів та вимогам проєкту. На рисунку 2.9 зображено Wireframe цільової сторінки.

Після розробленого каркасу розпочалася робота над детальним дизайном інтерфейсу у програмі Figma. Figma була обрана через велику кількість безкоштовного функціоналу та можливість створення інтерактивних прототипів. На рисунку 2.10 зображено готовий дизайн цільової сторінки.

У процесі проєктування використовувалася сітка (columns system), що забезпечувало гармонійне та впорядковане розташування елементів на сторінках. Сітка дозволяє підтримувати однаковість у розмірах та відступах між елементами, що важливо для створення візуально привабливого та інтуїтивно зрозумілого інтерфейсу. На рисунку 2.11 зображено демонстрацією роботи з сіткою в програмі Figma.

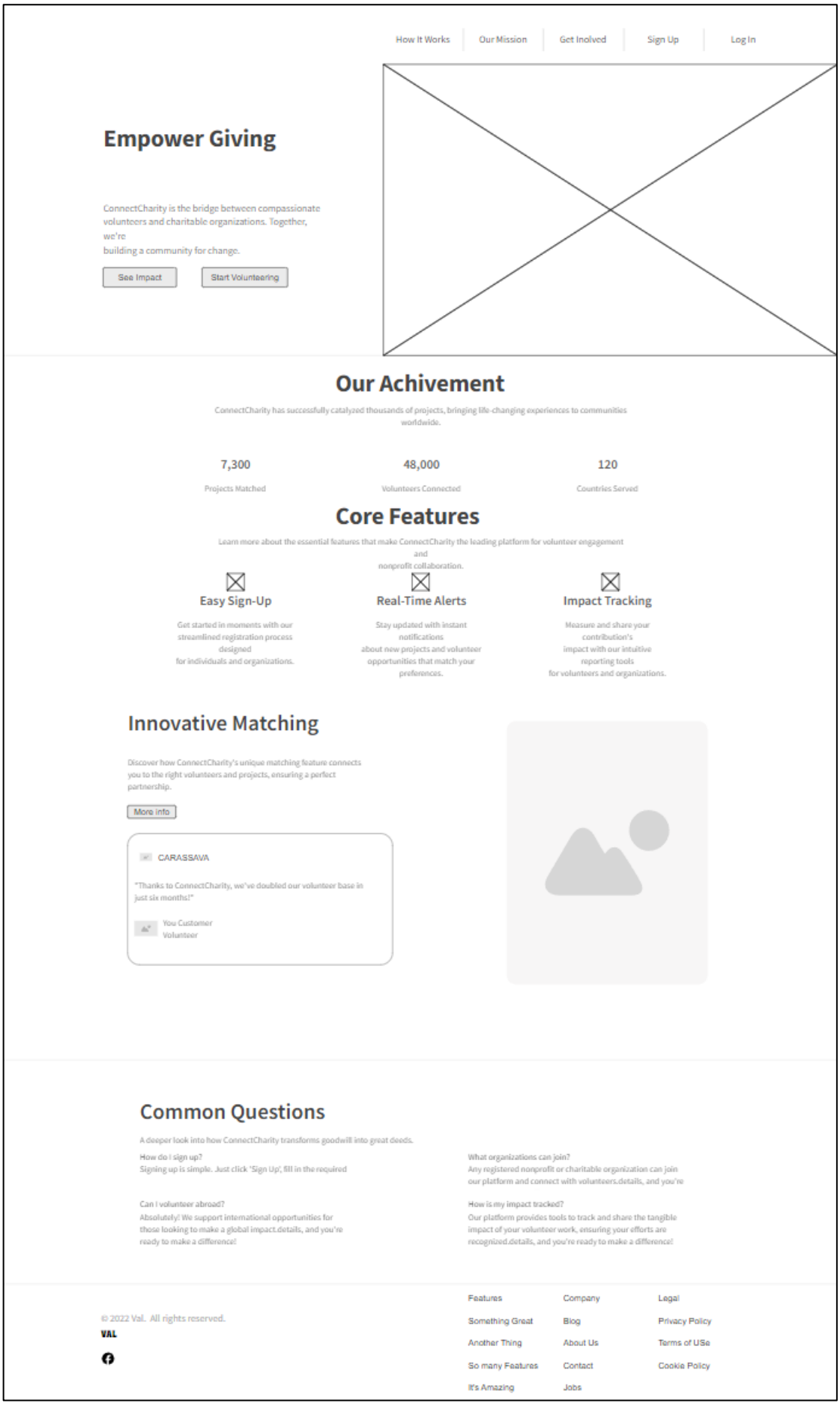


Рисунок 2.9 – Wireframe цільової сторінки

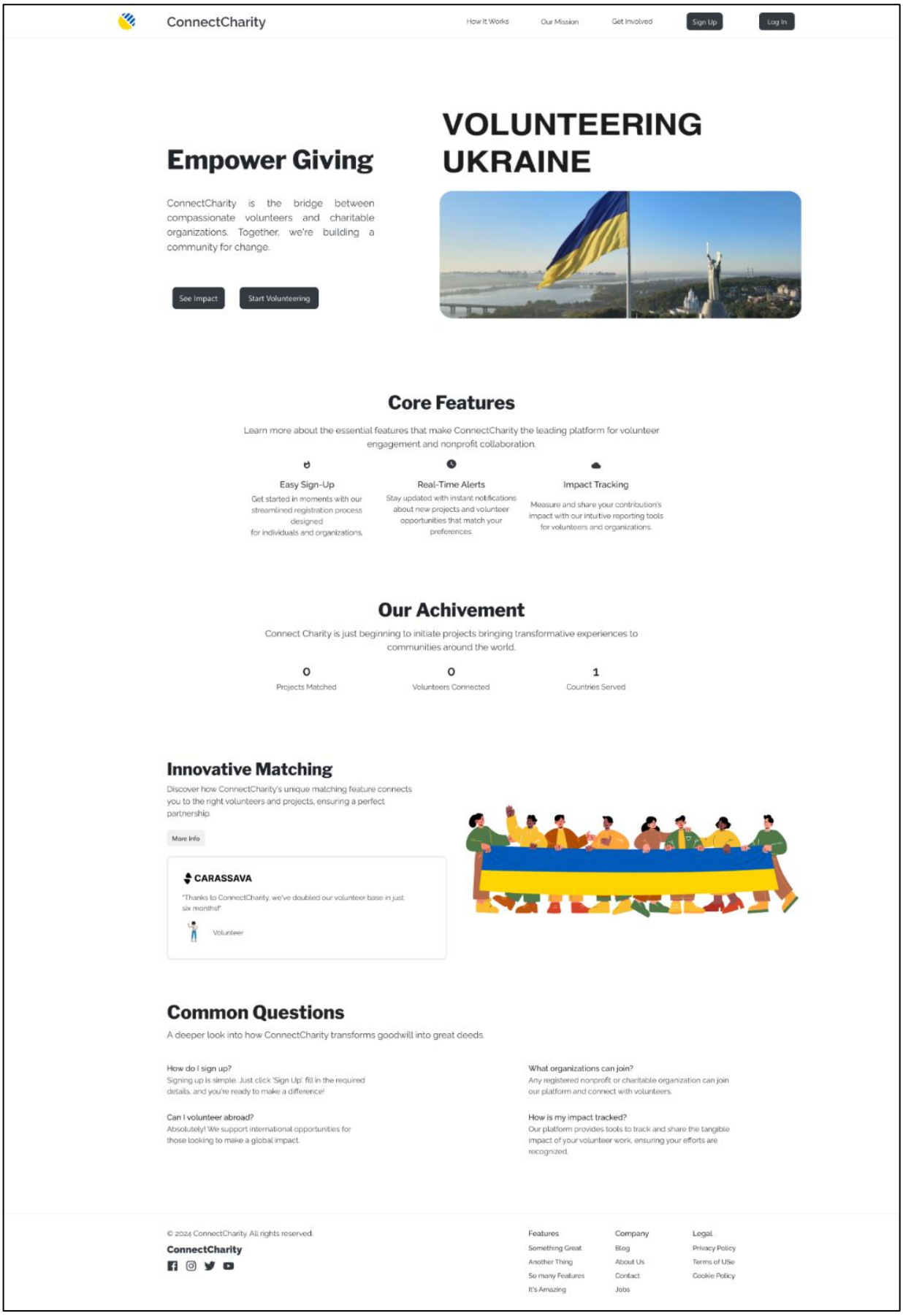


Рисунок 2.10 – Готовий дизайн цільової сторінки

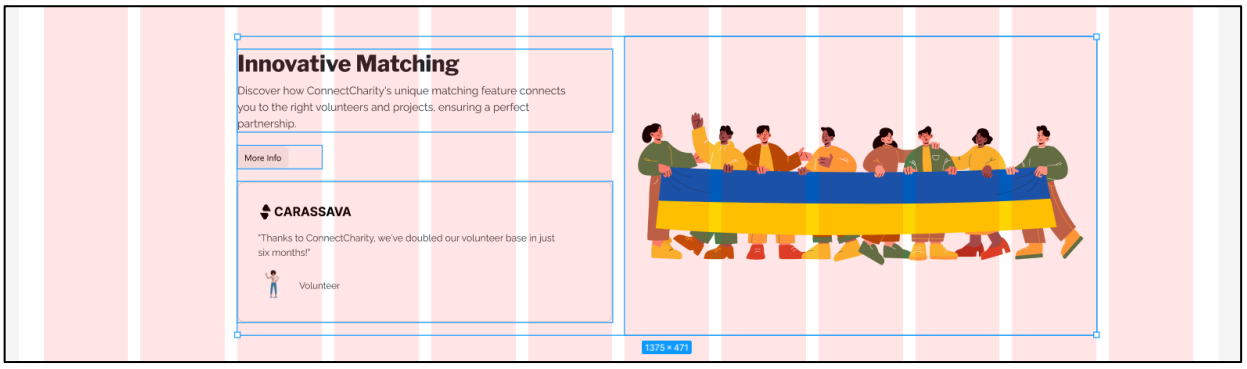


Рисунок 2.11 – Демонстрація роботи з сіткою

Організація шарів та об’єктів у Figma здійснювалася з особливою увагою до структурованості проєкту. Всі елементи інтерфейсу були згруповані за логічними категоріями, що полегшувало навігацію за проєктом та внесення змін на будь-якому етапі розробки.

Одним із ключових аспектів проєктування інтерфейсу стало створення та використання Design Style. Design Style включає набір візуальних рішень і правил, що застосовуються до всіх елементів інтерфейсу. До його складу входять палітра кольорів, типографіка, стилі кнопок, полів введення та інших компонентів. Визначення єдиного стилю дозволяє забезпечити візуальну узгодженість інтерфейсу, що значно покращує користувальницький досвід.

Використання Design Style не тільки сприяє створенню естетично приємного інтерфейсу, а й спрощує процес підтримки та розширення системи. При додаванні нових елементів або зміні існуючих розробники можуть легко застосовувати існуючі стилі, що забезпечує узгодженість і знижує ймовірність помилок. Крім того, це дозволяє прискорити процес розробки, оскільки всі учасники проєкту працюють у рамках єдиного набору правил та стандартів.

На рисунку 2.12 зображено Design style проєкту.

Палітра кольорів була обрана з урахуванням психологічних аспектів сприйняття кольорів, а також вимог доступності. Були вибрані контрастні поєднання кольорів для найважливіших елементів інтерфейсу, що забезпечувало їх помітність та зручність використання. Друкарня була підібрана таким чином, щоб текст залишався читабельним на будь-яких

пристроях та за будь-яких роздільних здатностей екрана. Кожен елемент має кілька станів (наприклад, звичайне, при наведенні та натисканні), що забезпечує зворотний зв’язок і допомагає користувачу краще орієнтуватися в інтерфейсі.

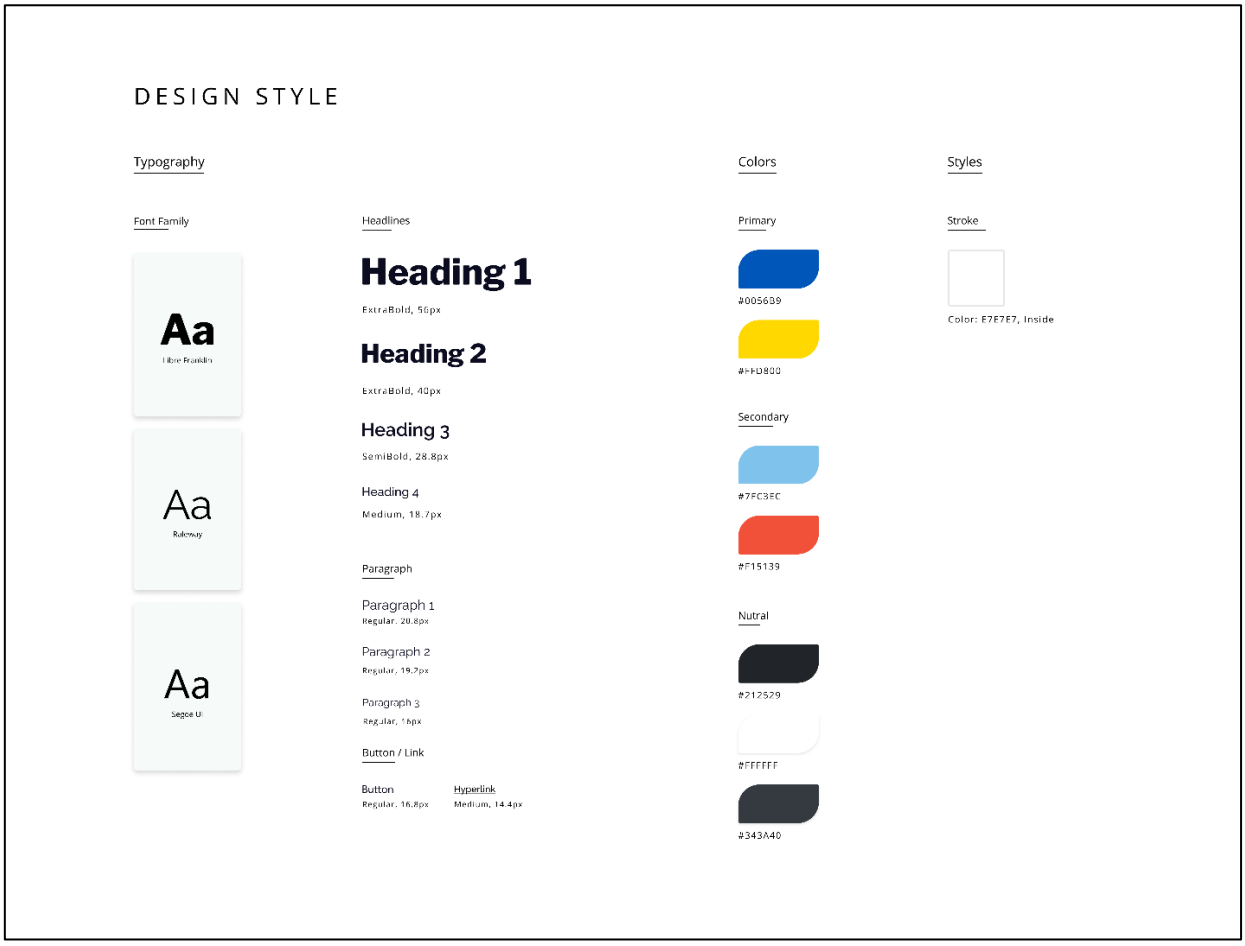


Рисунок 2.12 – Design style проєкту

Таким чином, проєктування інтерфейсу пройшло через кілька ключових етапів: створення каркасу сторінок, детальна розробка дизайну у Figma з використанням сітки та організації шарів, створення єдиного стилю для всіх елементів інтерфейсу та тестування прототипів. Такий підхід дозволив створити зручний, естетично привабливий та функціональний інтерфейс, який задовольняє потреби користувачів та сприяє ефективній взаємодії благодійних організацій та волонтерів.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Створення та конфігурація Spring Boot додатку

Створення проєкту в Spring Initializr це зручний інструмент в розробці програмного забезпечення з використанням Spring Framework, оскільки цей інструмент надає можливість швидко та ефективно створити початкову структуру програмного проєкту на основі Spring Framework. На рисунку 3.1 зображено конфігурація проєкту.

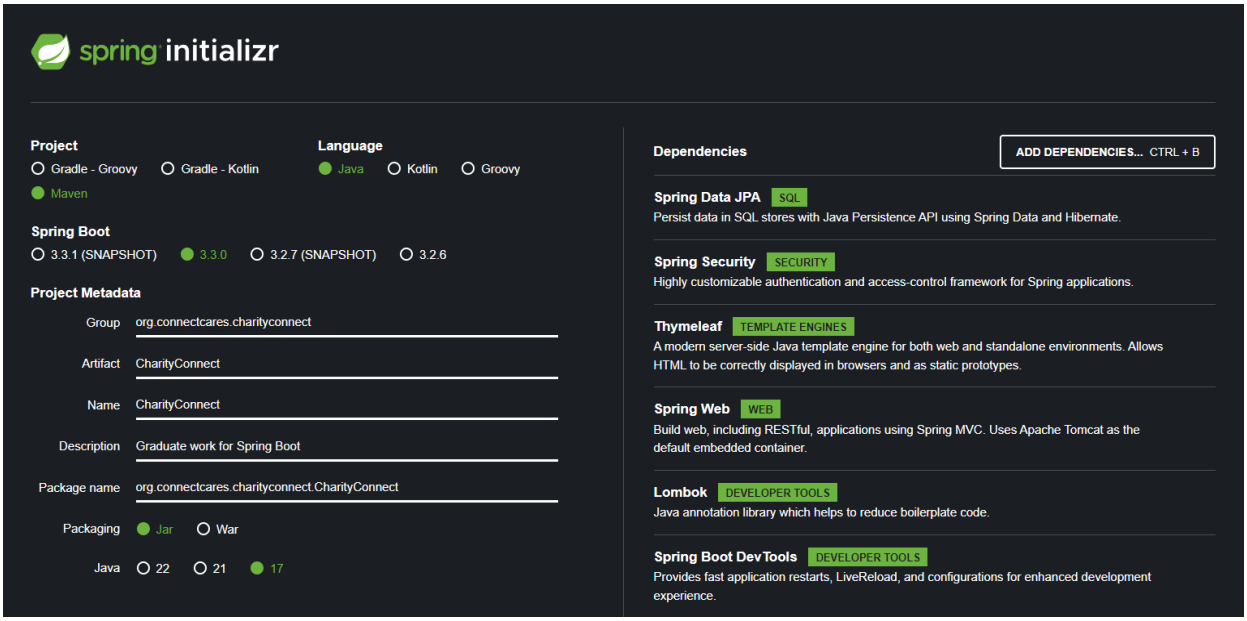


Рисунок 3.1 – Spring Initializr для створення проєкту

Процес створення проєкту в Spring Initializr розпочинається з вибору необхідних залежностей та налаштувань. Вибираємо необхідні компоненти Spring Framework, такі як Spring Data JPA, Spring Security, Thymeleaf, Spring Web, Lombok, Spring Boot DevTools. Після вибору залежностей та налаштувань було згенеровано вихідний код проєкту. Після створення проєкту інструментом Spring Initializr проєкт завантажується в Jar пакеті.

Першим кроком після створення проєкту є ініціалізація середовища

розробки. Це включає налаштування інтегрованого середовища розробки (IDE), такого як IntelliJ IDEA. Далі встановлюються необхідні залежності та бібліотеки. Це може включати в себе встановлення додаткових модулів Spring Framework, інших сторонніх бібліотек та компонентів, які потрібні для реалізації функціональності проєкту. На рисунку 3.2 зображено структура проєкту.

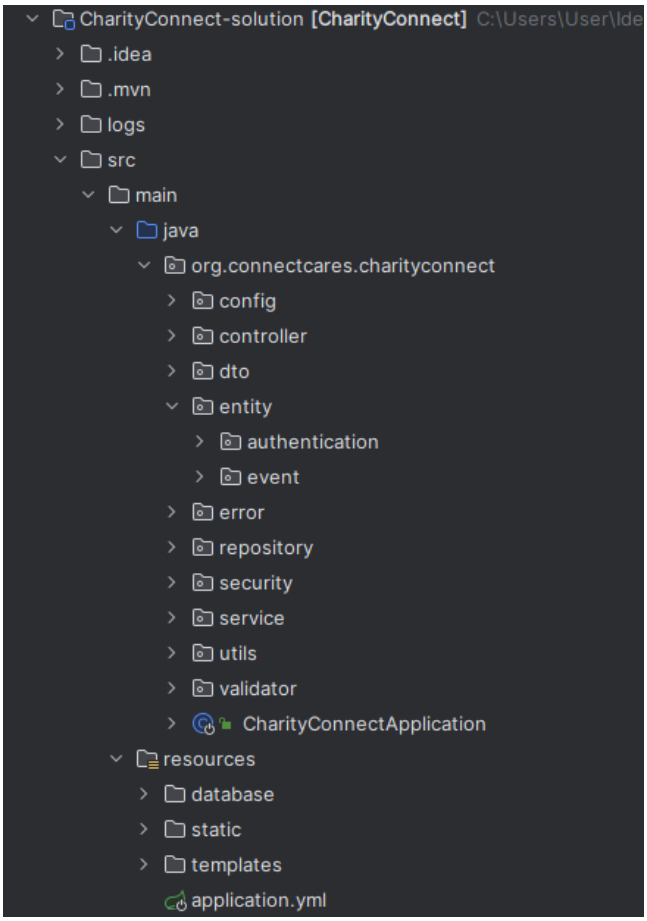


Рисунок 3.2 – Структура проєкту

Частина підмодулів була прихована, щоб скоротити розмір зображення. Структуру проєкту можна умовно поділити на такі пакети:

– org.connectcares.charityconnect – основний пакет програми, що містить кілька підкаталогів та класів;
– config – містить класи конфігурації, які використовуються для налаштування аспектів програми;

- controller – у цьому каталозі знаходяться контролери, що відповідають за обробку HTTP-запитів та повернення відповідних відповідей (контролери реалізують логіку взаємодії з користувачами та клієнтами);
- dto – містить класи Data Transfer Object (DTO), які використовуються для передачі даних між різними шарами програми;
- entity – у цьому каталозі зберігаються класи сутностей, які представляють об’єкти бази даних;
- authentication – сутності, пов’язані з автентифікацією та авторизацією користувачів;
- event – сутності, пов’язані з благодійними заходами;
- error – містить класи для обробки помилок та винятків у програмі;
- repository – у цьому каталозі знаходяться інтерфейси репозиторіїв, які взаємодіють із базою даних;
- security – містить класи та конфігурації, пов’язані з безпекою програми;
- service – у цьому каталозі розташовані сервісні класи, що реалізують бізнес-логіку програми;
- utils – допоміжні класи та утиліти, що використовуються у додатку;
- validator – містить класи для валідації даних;
- database – директорія для файлів, пов’язаних із базою даних, таких як скрипти ініціалізації;
- static – містить статичні ресурси, доступні безпосередньо через вебсервер (CSS, JavaScript, зображення);
- templates – цей каталог містить шаблони сторінок (HTML), які використовуються для рендерингу вебінтерфейсів.

Центральною точкою запуску програми (аналог Main) у Spring Boot і в даному випадку є клас CharityConnectApplication, який викликає SpringApplication.run з необхідними аргументами. Чітка структура проєкту

Spring Boot програми забезпечує поділ коду та ресурсів, що спрощує управління та розвиток проєкту. Логічне поділ на контролери, послуги, репозиторії та конфігурації сприяє підтримуваності та масштабованості програми. Наявність шаблонів і статичних ресурсів в окремих директоріях покращує організацію фронтенд-частини і полегшує роботу з інтерфейсом користувача.

3.2 Розробка фронтенду використовуючи шаблонний двигун

У процесі розробки вебдодатків на базі Spring Boot, одним із ключових аспектів є реалізація фронтенду, який взаємодіє з користувачами та відображає дані по бекенду. Одним із популярних підходів до створення динамічних вебсторінок у Spring Boot є використання шаблонного двигуна Thymeleaf.

Thymeleaf – це сучасний шаблонний двигун, розроблений для Java-додатків. Він надає безліч можливостей, які роблять його найкращим вибором для розробки фронтенду в Spring Boot додатках. Основні переваги Thymeleaf включають: простота інтеграції з Spring MVC, підтримка стандартного HTML, динамічне оновлення даних та розширюваність.

Thymeleaf працює шляхом заміни спеціальних атрибутів та виразів у HTML-шаблонах на реальні дані, що передаються з контролерів. Контролери Spring Boot обробляють HTTP-запити, витягують необхідні дані та передають їх у шаблони для рендерингу. Це забезпечує гнучкість і потужність при створенні інтерактивних і чуйних інтерфейсів користувача. Thymeleaf також підтримує роботу з різними форматами, такими як XML і JavaScript, що розширює його можливості розробки складних вебдодатків. Завдяки своїй адаптованості та простоті використання, Thymeleaf є незамінним інструментом в арсеналі розробників Spring Boot.

На рисунку 3.3 зображено LandingController, який використовує Thymeleaf.

```java
package org.connectcares.charityconnect.controller;

import ...

@Controller
@Slf4j
public class LandingController {

  @GetMapping("/")
  public String landing() {
    log.debug("Initializing landing page");
    return "landing";
  }
}
```

Рисунок 3.3 – LandingController

Даний контролер LandingController обробляє GET-запити за адресою / і повертає ім’я шаблону landing, який повинен бути використаний для відображення стартової сторінки. На рисунку 3.4 зображено шаблон Thymeleaf – landing.html.

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.w3.org/1999/xhtml" lang="en"
      layout:decorate="~{layouts/basic_layout}">

<head>
  <title>Charity Connect</title>
  <link rel="stylesheet" type="text/css" th:href="@{/css/pages/landing.css}">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css">
</head>

<body>
<div layout:fragment="content">
<div class="main-content">
  <div class="left-content">
    <h1>Empower Giving</h1>
    <p>ConnectCharity is the bridge between compassionate volunteers and charitable organizations. Together, we're build
    <div class="buttons">
      <a href="#" class="button button-dark button-action">See Impact</a>
      <a href="#" class="button button-dark button-action">Start Volunteering</a>
    </div>
  </div>
  <div class="right-content">
    <img th:src="@{/images/ukraine-flag.png}" alt="Ukraine Flag" />
```

Рисунок 3.4 – Шаблон Thymeleaf

Цей шаблон використовує атрибути Thymeleaf, які дозволяють динамічно задавати посилання на стилі та підключати загальний макет для

сторінки. Одним із ключових аспектів використання Thymeleaf є можливість інтеграції динамічного контенту. Наприклад, можна передавати дані про користувача або події до шаблонів, щоб відображати їх на сторінці.

3.3 Реєстрація та авторизація

Реєстрація та авторизація користувачів є ключовими аспектами будь-якої вебпрограми, забезпечуючи безпеку даних та функціональність, доступну лише авторизованим користувачам. Тому було вирішено використовувати модуль Spring Security та шаблонний двигун Thymeleaf.

Процес реєстрації користувачів здійснюється через контролер RegisterController, який обробляє HTTP-запити, пов’язані з відображенням та обробкою реєстраційних форм для волонтерів та благодійних організацій. Розглянемо метод showRegistrationChoice з Додатка А, який відображає сторінку, на якій користувачі можуть вибрати, чи хочуть вони зареєструватися як волонтери чи благодійні організації. Ця сторінка є початковою точкою для реєстрації та надає користувачам вибір залежно від їхньої ролі в системі.

Після вибору типу користувача система перенаправляє його на відповідну форму реєстрації, де необхідно вказати персональні дані та інформацію, специфічну для обраної ролі. Контролер також забезпечує валідацію введених даних, запобігаючи реєстрації з некоректними або неповними даними. Після успішної реєстрації користувачі можуть увійти до системи через сторінку авторизації, де їх дані перевірятимуться з використанням Spring Security. Цей підхід гарантує, що доступ до функціональності платформи отримують лише зареєстровані та перевірені користувачі.

На рисунку 3.5 зображено сторінка вибору типу облікового запису для реєстрації.

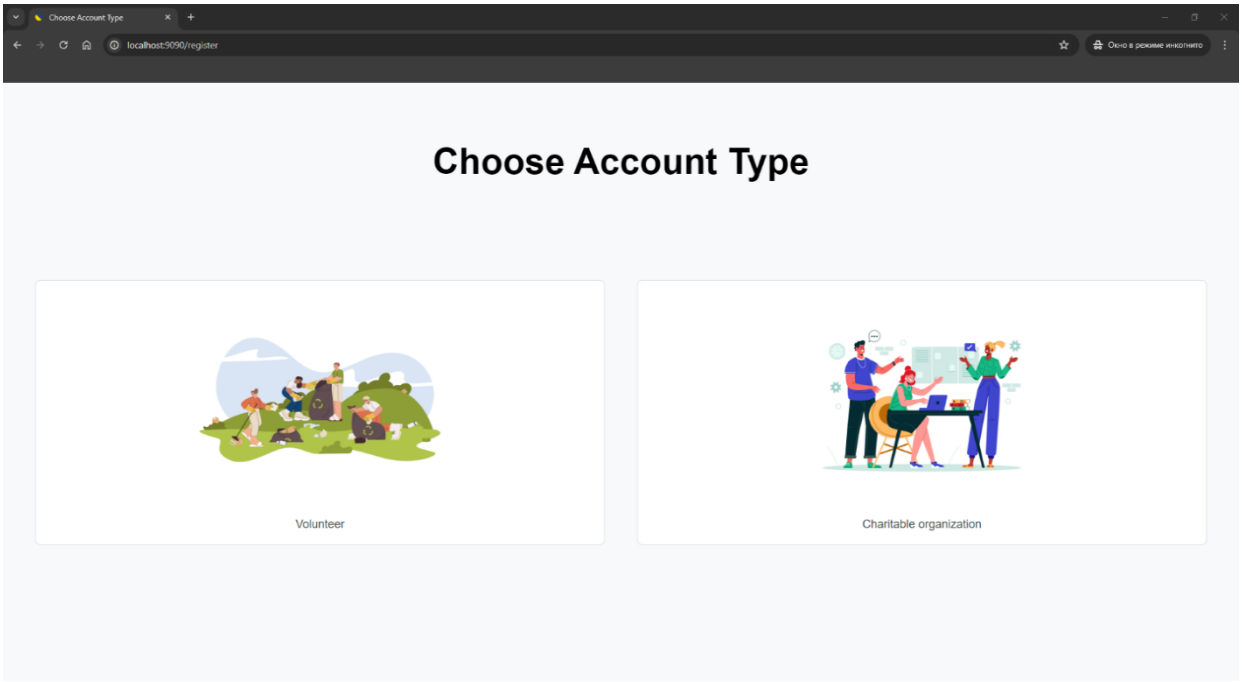


Рисунок 3.5 – Вибір типу облікового запису

Два окремі методи showVolunteerRegistrationForm та showOrganizationRegistrationForm відображають форми реєстрації для волонтерів та благодійних організацій відповідно. В обох методах створюється об’єкт UserRegistrationDto, який використовується передачі даних з форми в контролер. Поле організації цього об’єкта встановлюється false для волонтерів і true для організацій, що дозволяє розрізняти тип користувача при обробці даних.

При надсиланні форми контролер приймає дані, проводить їх валідацію та зберігає нового користувача у базі даних. Цей процес забезпечує коректне рознесення користувачів за категоріями, що важливо для подальшого керування доступом та надання відповідних функціональних можливостей у системі.

Додатково, після успішної реєстрації, користувачі можуть отримувати повідомлення електронною поштою, що підтверджують успішне створення облікового запису. Такий підхід сприяє поліпшенню досвіду користувача і підвищенню безпеки при доступі до платформи. На рисунку 3.6 зображено форма реєстрації волонтера.

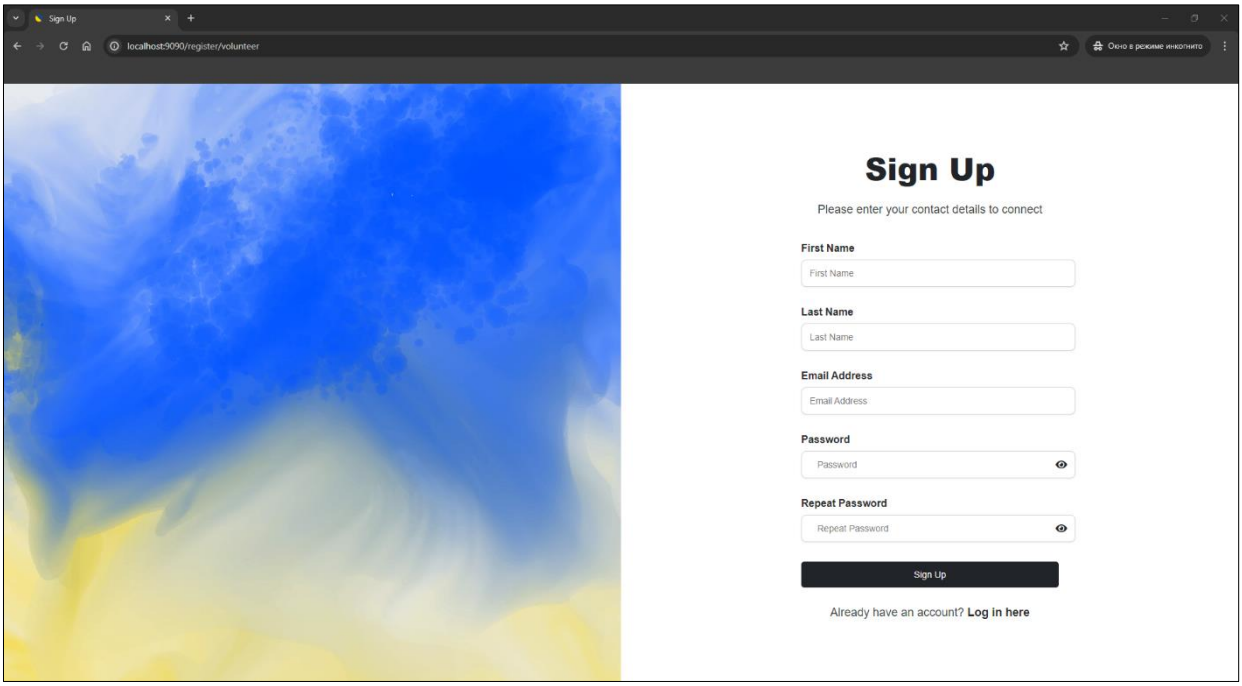


Рисунок 3.6 – Форма реєстрації волонтера

Метод реєструоблікового запису обробляє відправку реєстраційної форми. У ньому перевіряються помилки валідації, і якщо їх немає, обліковий запис зберігається за допомогою userService.save(registrationDto). Після успішної безпеки відбувається перенаправлення на відповідну сторінку успіху реєстрації залежно від типу користувача.

Методи showVolunteerRegistrationSuccess та showOrganizationRegistrationSuccess відображають сторінки успіху для волонтерів та організацій відповідно.

Форми для реєстрації користувача відрізняються наявністю різних полів, але форма авторизації однакова для волонтерів та благодійних організацій. Це дозволяє уніфікувати процес входу в систему та спростити керування обліковими записами.

Такий підхід забезпечує високу гнучкість та масштабованість системи, дозволяючи легко адаптувати її до нових вимог та умов.

Крім того, це спрощує подальшу підтримку та розвиток платформи. На рисунку 3.7 зображено форма авторизації користувача.

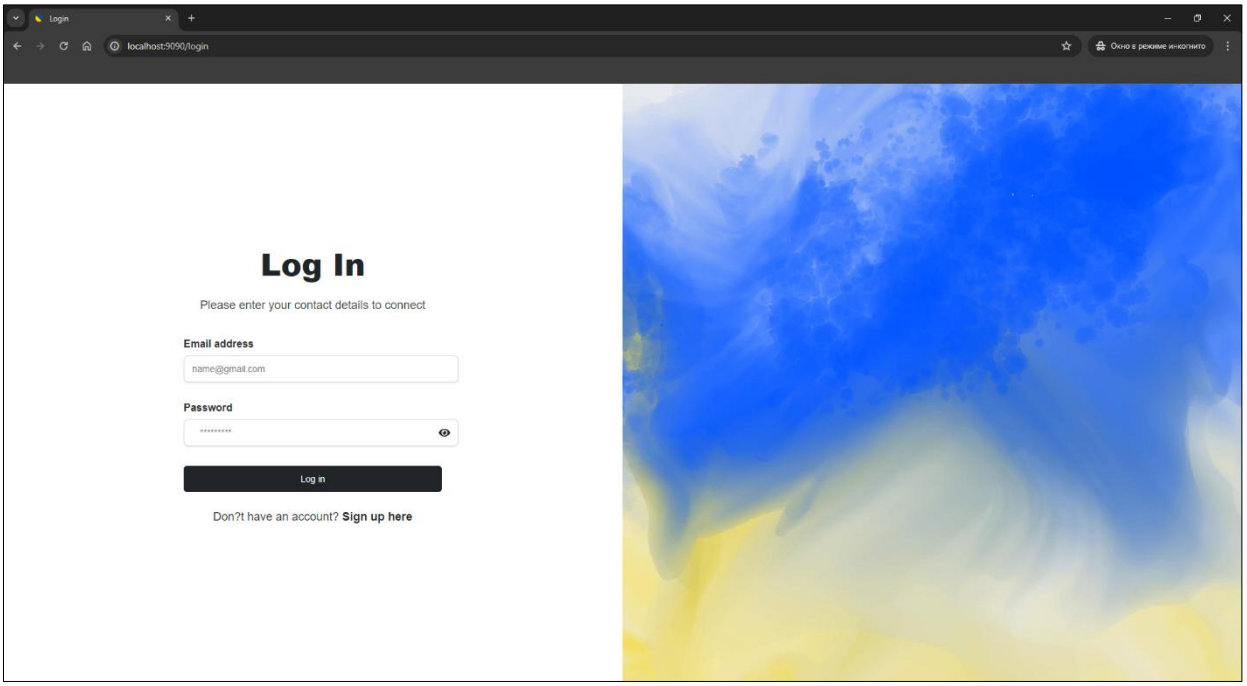


Рисунок 3.7 – Форма авторизації користувача

Таким чином, реєстрація та авторизація користувачів реалізовані з використанням Spring Security та Thymeleaf. Контролер RegisterController обробляє відображення та надсилання реєстраційних форм для двох типів користувачів: волонтерів та благодійних організацій. Конфігурація безпеки в класі SecurityConfig забезпечує необхідні правила авторизації, захищаючи ресурси програми та надаючи доступ до них лише авторизованим користувачам з відповідними ролями.

3.4 Життєвий цикл події

Події, які організовують благодійні організації, відіграють ключову роль у роботі з платформою. Життєвий цикл події включає кілька етапів: від його створення, проведення, завершення та зберігання в історії подій. Розгляд кожного етапу дозволяє краще зрозуміти, як організовуються, керуються та завершуються такі заходи. Ініціація події починається з ідеї, що оформляється благодійною організацією. На цьому етапі визначається мета заходу, його

формат, цільова аудиторія та очікувані результати. Організація переходить на сторінку створення заходу та заповнює форму з такими даними: назва події, короткий опис, дата початку події, дата закінчення події, локація та тип заходу. На рисунку 3.8 зображено форма створення події.

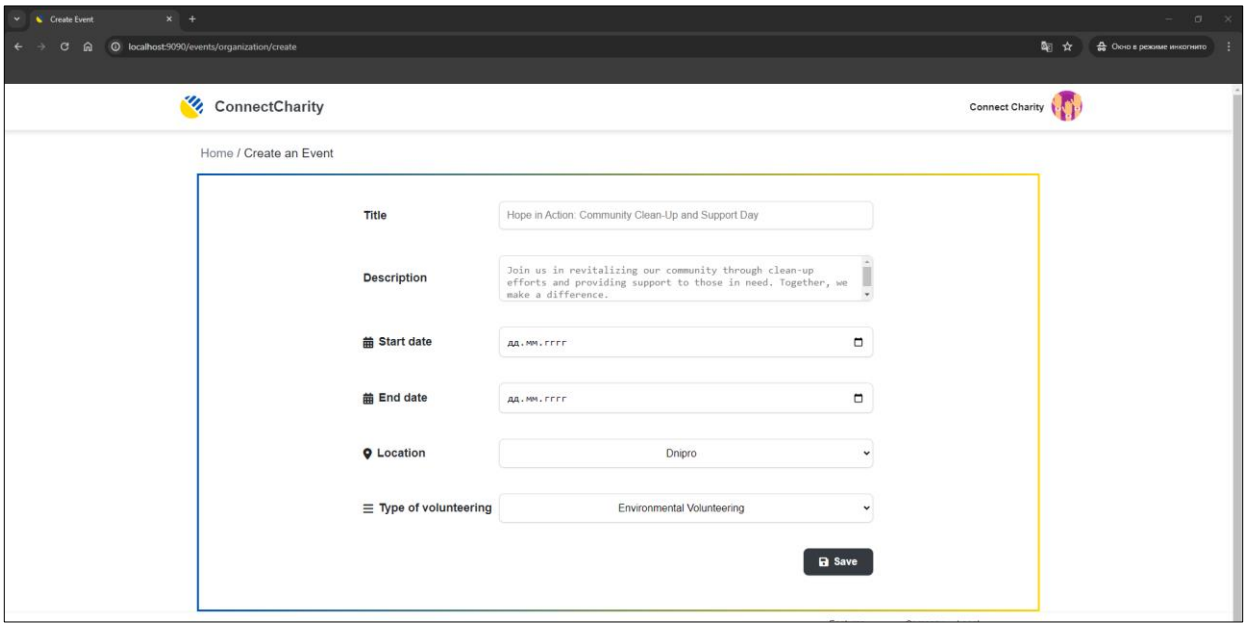


Рисунок 3.8 – Форма створення події

Розглянемо інтерфейс OrganizationEventsController з **Додаток Б** через який організація може створювати нову подію. Метод showCreateEventForm відображає форму для створення нової події, яка запитує інформацію. Після заповнення форми та надсилання, дані надходять до методу createEvent, який обробляє валідацію даних. Він перевіряє, що всі поля заповнені коректно і дати знаходяться в майбутньому. Якщо дані валідні, викликається метод createEvent з EventServiceIF для збереження події у базі даних.

Після створення події організацією, волонтер може знайти її та надіслати свою заявку для участі. Волонтери можуть переглядати доступні події та свої заявки через інтерфейс VolunteerEventsController. Метод showVolunteerEventsPage відображає сторінку з подіями. Залежно від параметра toggle, волонтер може бачити свої заявки (myEvents) або доступні для пошуку події (eventSearch). На рисунку 3.9 зображено сторінка “Events” з подіями.

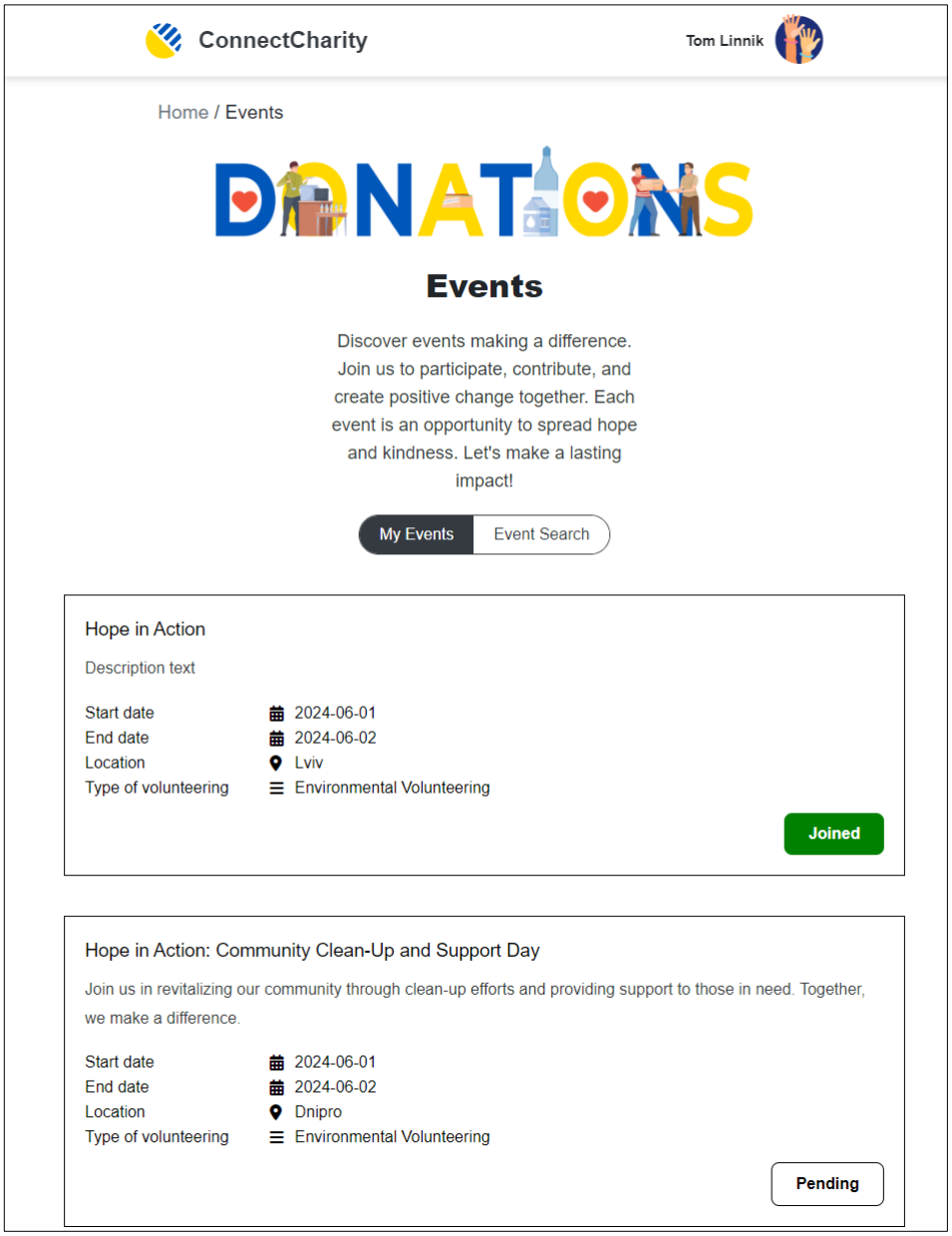


Рисунок 3.9 – Сторінка з подіями волонтера

Розглянемо пошук подій з **Додаток В**. Волонтер може шукати події за критеріями, такими як назва, дата початку та закінчення, місце розташування та тип заходу, а також за ключовими словами. Метод searchEvents обробляє запит і повертає результати пошуку.

Результати пошуку відображаються у вигляді списку подій, які відповідають заданим критеріям. Такий підхід дозволяє волонтерам швидко знаходити заходи, що їх цікавлять, і планувати свою участь. На рисунку 3.10 зображено сторінка “Events” з відкритою вкладкою “Event Search”.

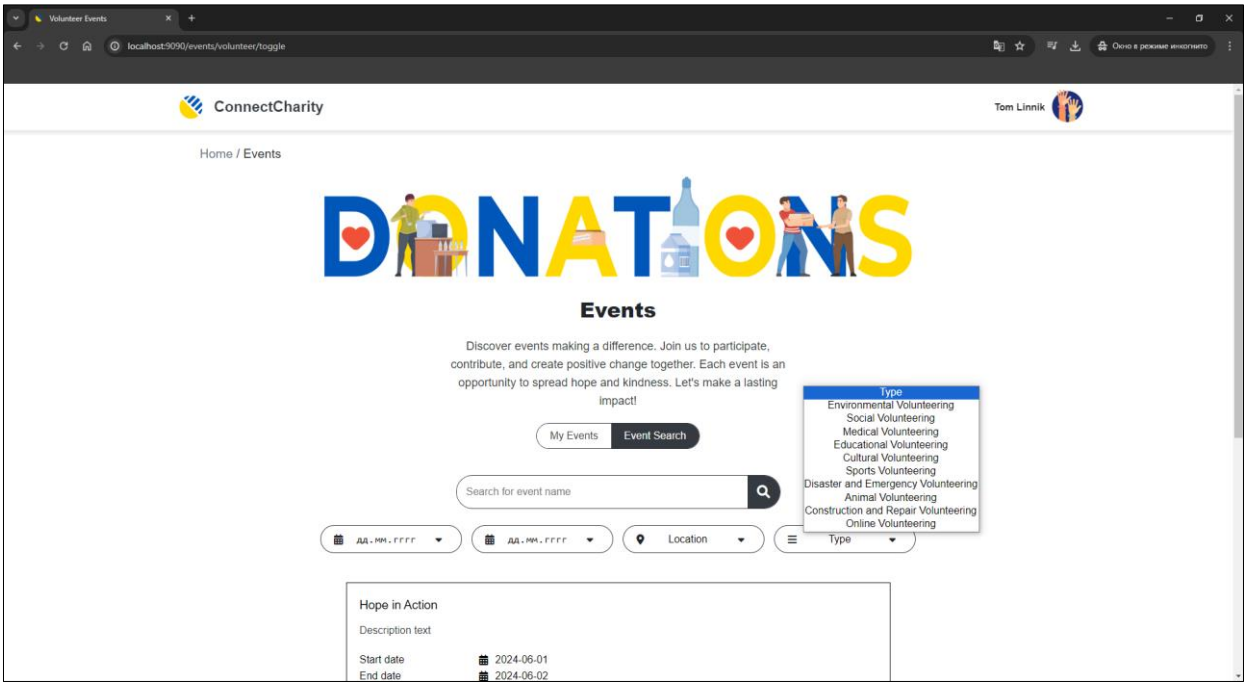


Рисунок 3.10 – Пошук подій

Волонтер може подати заявку на участь у події через метод applyforEvent. Цей метод створює новий запис заявки у базі даних та зберігає її. Організація може оновити статус заявки (наприклад, схвалити або відхилити) через метод updateApplicationStatus. На рисунку 3.11 зображено управління заявками від волонтерів, які хочуть взяти участь у заході.

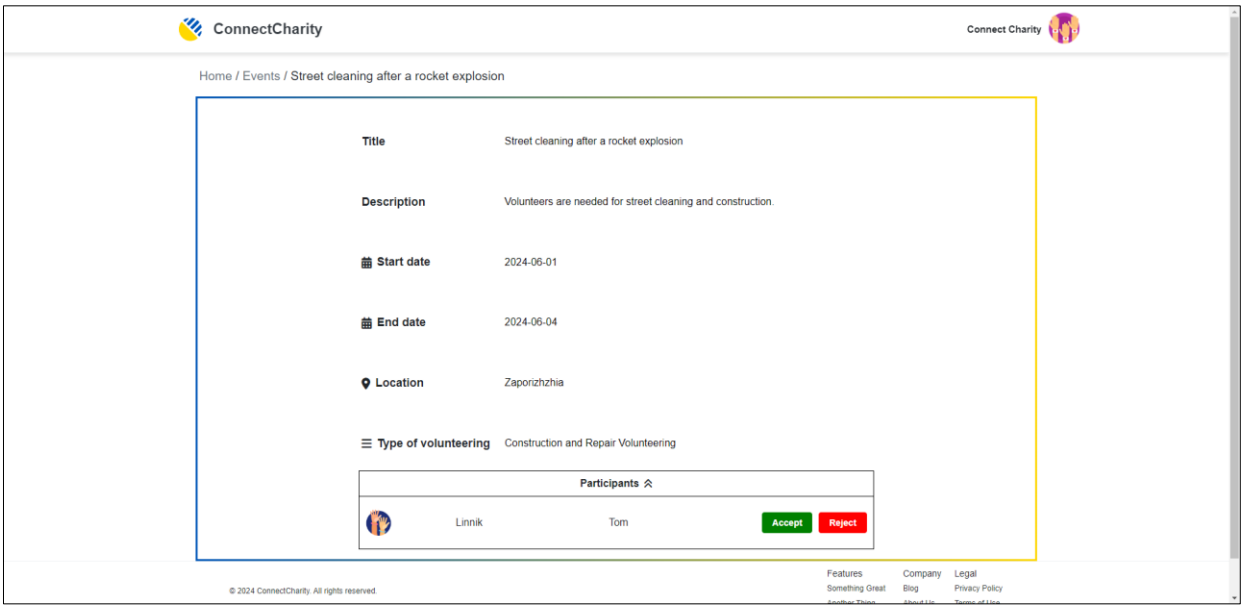


Рисунок 3.11 – Управління заявками

Розглянемо закінчення подій з **Додаток Г**. Було передбачено функціональність автоматичного переміщення подій на сторінку “History of events” після завершення їхнього часу закінчення. Це дозволяє користувачам бачити лише актуальні події на основній сторінці “Events”, а завершені події зберігаються в історії для подальшого перегляду. Для цього систему можна розділити на 4 головні компоненти:

- визначення часу закінчення події – кожна подія має атрибут, який вказує дату та час її закінчення;
- моніторинг поточного часу – система постійно перевіряє активні події та порівнює поточний час із часом закінчення кожної події;
- переміщення в історію – коли система виявляє, що час закінчення події минув, подія автоматично переміщується на сторінку “History of events” (ця дія виконується без втручання користувача);
- зберігання завершених подій – усі завершені події зберігаються на сторінці “History of events” (тут користувачі можуть отримати доступ до детальної інформації про кожну завершену подію, а також залишити відгуки та оцінки).

У контролері HistoryController використовується метод showVolunteerHistory для відображення історії подій волонтера і метод showOrganizationHistory для відображення історії подій організації. Ці методи отримують завершені події з бази даних і додають їх у модель для відображення на сторінці “History of events”. Ці методи використовують EventServiceIF для отримання списку завершених подій, які потім відображаються на сторінці історії.

Таким чином, після закінчення часу події автоматично переміщуються на сторінку “History of events” і зберігаються там як історія. Це забезпечує зручний доступ до інформації про завершені події та підтримує актуальність подій, що відображаються на основній сторінці.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено сервіс комунікації волонтерів та благодійних організацій. Проведена робота охопила кілька ключових етапів. Ці етапи включали дослідження та аналіз предметної області, технічне проєктування, створення інтерфейсу користувача, а також розробку системи.

Початкове дослідження та аналіз предметної області були критично важливими для розуміння вимог та очікувань користувачів платформи. У наш час критично важлива наявність подібної платформи, тому було проведено роботу з вивчення основних потреб волонтерів та благодійних організацій. Це дослідження заклало основу для подальшого проєктування системи, дозволивши сфокусуватися на створенні корисних функцій.

Технічне проєктування системи включало розробку технічного завдання, визначення функціональних вимог і вибір технологій для реалізації проєкту. У процесі проєктування було створено низку діаграм та нотацій, які допомогли структурувати інформацію та ясно уявити взаємодію між різними компонентами системи. Цей етап забезпечив надійну архітектурну основу для подальшої розробки та гарантував, що всі функціональні вимоги будуть враховані та реалізовані.

Особливу увагу було приділено створенню інтерфейсу користувача, який є важливим аспектом будь-якого програмного продукту. Були розроблені докладні каркаси та дизайн макети, які забезпечили інтуїтивно зрозумілу та зручну навігацію по платформі. Це забезпечило позитивний досвід користувача і збільшило задоволеність користувачів від взаємодії з платформою.

Розробка системи була проведена за допомогою сучасних технологій та інструментів. Розробка системи була проведена із застосуванням сучасних методик та інструментів, що дозволило створити стійку та ефективну систему

для обробки даних. Основну увагу було приділено забезпеченню стабільності та продуктивності системи. У процесі розробки були використані кращі практики програмування, що сприяло створенню системи, що легко підтримується і розширюється.

У процесі розробки проєкту було досягнуто всі поставлені цілі та завдання. Створено багатофункціональну платформу, яка об’єднує волонтерів та благодійні організації, полегшує процес організації та проведення заходів, а також забезпечує зручні інструменти для управління та оцінки цих заходів. Проєкт показав значущість комплексного підходу до розробки програмних продуктів, що включає глибоке дослідження предметної галузі, ретельне проєктування, використання сучасних технологій та інструментів. Такий підхід дозволив створити якісний та затребуваний продукт, який задовольняє потреби всіх користувачів та сприяє розвитку благодійності.

Насамкінець розроблений проєкт благодійної платформи має значний потенціал для подальшого розвитку та масштабування. У майбутньому можливе додавання нових функцій та сервісів, інтеграція з іншими платформами та системами, а також розширення географії використання. Все це дозволить залучати більше волонтерів та благодійних організацій, підвищувати якість та ефективність заходів, що проводяться, та вносити значний внесок у розвиток благодійності та волонтерського руху.

ПЕРЕЛІК ПОСИЛАНЬ

1. Все про благодійність в Україні. URL: https://rubryka.com/article/charity-ukraine/ (дата звернення: 15.03.2024).
2. Paul McFedries. HTML, CSS, & JavaScript All-in-One For Dummies. P. 55. URL: https://www.everand.com/book/659828617/HTML-CSS-JavaScript-All-in-One-For-Dummies?language_settings_changed=English (дата звернення: 05.04.2024).
3. Java Documentation. URL: https://docs.oracle.com/en/java/ (дата звернення: 05.04.2024).
4. Spring Framework Documentation. URL: https://docs.spring.io/spring-framework/reference/overview.html (дата звернення: 05.04.2024).
5. Thymeleaf. URL: https://www.thymeleaf.org/ (дата звернення: 03.04.2024).
6. PostgreSQL 15.7 Documentation. URL: https://www.postgresql.org/docs/15/intro-whatis.html (дата звернення: 05.04.2024).
7. Siriwardena P. Maven Essentials. P. 21. URL: https://www.everand.com/book/342444109/Maven-Essentials (дата звернення: 05.04.2024).
8. Tudose C. JUnit in Action. P. 21. URL: https://www.everand.com/book/511817159/JUnit-in-Action (дата звернення: 05.04.2024).
9. Belling R. Research Project Management: 25 Free Tools. P. 19. URL: https://www.everand.com/book/303791889/Research-Project-Management-25-Free-Tools (дата звернення: 05.04.2024).
10. Münninghoff G., Tönnies A. Kurzanleitung Project Libre. P. 11. URL: https://www.everand.com/book/492198152/Kurzanleitung-Project-Libre (дата звернення: 05.04.2024).
11. Grassle P., Baumann H., Baumann P. UML 2.0 in Action: A project-based

tutorial. P. 30. URL: https://www.everand.com/book/272083393/UML-2-0-in-Action-A-project-based-tutorial (дата звернення: 05.04.2024).

12. Allen G. Taylor. SQL All-in-One For Dummies. P. 98. URL: https://www.everand.com/book/220195238/SQL-All-in-One-For-Dummies (дата звернення: 05.04.2024).
13. Notations: How to Explain Complicated Processes With the Help of Logical Diagrams. URL: https://medium.com/geekculture/notations-how-to-explain-complicated-processes-with-the-help-of-logical-diagrams-902b97f8c3f6 (дата звернення: 15.04.2024).
14. Allweyer T. BPMN 2.0: Introduction to the Standard for Business Process Modeling. P. 13. URL: https://www.everand.com/book/311329411/BPMN-2-0-Introduction-to-the-Standard-for-Business-Process-Modeling (дата звернення: 25.04.2024).
15. Teixeira F. Introdução e boas práticas em UX Design. P. 38. URL: https://www.everand.com/book/405708544/Introducao-e-boas-praticas-em-UX-Design (дата звернення: 15.04.2024).

ДОДАТОК А

Програмна реалізація showRegistrationChoice

RegisterController.java

```java
package org.connectcares.charityconnect.controller;

import ...

@Controller
@AllArgsConstructor
@Slf4j
public class RegisterController {

  private final UserServiceIF userService;

  @GetMapping("/register")
  public String showRegistrationChoice() {
    log.debug("Initializing registration choice page");
    return "choose_account_type";
  }

  @GetMapping("/register/volunteer")
  public String showVolunteerRegistrationForm(Model model) {
    log.info("Initializing volunteer registration form");
    UserRegistrationDto userRegistrationDto = new UserRegistrationDto();
    userRegistrationDto.setOrganization(false);
    model.addAttribute( attributeName: "user", userRegistrationDto);
    return "register";
  }

  @GetMapping("/register/organization")
  public String showOrganizationRegistrationForm(Model model) {
    log.info("Initializing organization registration form");
    UserRegistrationDto userRegistrationDto = new UserRegistrationDto();
    userRegistrationDto.setOrganization(true);
    model.addAttribute( attributeName: "user", userRegistrationDto);
    return "register";
  }

  @PostMapping("/register")
  public String registerAccount(@ModelAttribute("user") @Valid UserRegistrationDto registrationDto,
      BindingResult result, Model model, RedirectAttributes redirectAttributes) {
    if (result.hasErrors()) {
      log.debug("Validation errors occurred: {}", result.getAllErrors());
      return "register";
    }

    log.info("Registering new account for email: [{}]", registrationDto.getEmail());
    userService.save(registrationDto);
    redirectAttributes.addFlashAttribute( attributeName: "successMessage", attributeValue: "Registration successful!");

    String redirectUrl =
        registrationDto.isOrganization() ? "redirect:/register/organization?success"
            : "redirect:/register/volunteer?success";
    log.info("Registration successful, redirecting to: [{}]", redirectUrl);
    return redirectUrl;
  }

  @GetMapping("/register/volunteer?success")
  public String showVolunteerRegistrationSuccess(Model model) {
    log.info("Displaying volunteer registration success page");
    return "register";
  }

  @GetMapping("/register/organization?success")
  public String showOrganizationRegistrationSuccess(Model model) {
    log.info("Displaying organization registration success page");
    return "register";
  }
}
```

ДОДАТОК Б

Програмна реалізація OrganizationEventsController

```
66      @GetMapping("/create")
67      public String showCreateEventForm(Model model) {
68        log.debug("Displaying the form for creating a new event");
69
70        // Add an empty Event object to the model to be filled by the form
71        model.addAttribute( attributeName: "event", new Event());
72
73        // Add locations and volunteering types to the model
74        model.addAttribute( attributeName: "locations", eventService.getAllLocations());
75        model.addAttribute( attributeName: "volunteeringTypes", eventService.getAllVolunteeringTypes());
76
77        return "events/create-event";
78      }
```

```
89      @PostMapping("/create")
90      public String createEvent(@Valid @ModelAttribute Event event,
91          BindingResult bindingResult,
92          Authentication authentication,
93          Model model) {
94
95        // If there are validation errors, return the form view
96        if (bindingResult.hasErrors()) {
97          log.debug("Validation errors encountered while creating a new event");
98
99          // Add locations and volunteering types to the model
100         model.addAttribute( attributeName: "locations", eventService.getAllLocations());
101         model.addAttribute( attributeName: "volunteeringTypes", eventService.getAllVolunteeringTypes());
102         return "events/create-event";
103       }
```

HistoryController.java × | VolunteerEventsController.java | EventApplicationServiceImpl.java | DataInitializer.java

```
1   package org.connectcares.charityconnect.controller;
2
3   import ...
25
26  @Controller
27  @RequestMapping("/history")
28  @RequiredArgsConstructor
29  @Slf4j
30  public class HistoryController {
31
32    private final EventServiceIF eventService;
33    private final ReviewServiceIF reviewService;
34    private final UserServiceIF userService;
35
36    @GetMapping("/volunteer")
37    @PreAuthorize("hasRole('ROLE_VOLUNTEER')")
38    public String showVolunteerHistory(Model model, Authentication authentication) {
39      // Get the email of the authenticated user
40      String userEmail = authentication.getName();
41      // Retrieve the volunteer ID using the email
42      Long volunteerId = eventService.getVolunteerId(userEmail);
43      // Get the list of past events for the volunteer
44      List<Event> pastEvents = eventService.getPastEventsByVolunteerId(volunteerId);
45      // Add the list of past events to the model
46      model.addAttribute( attributeName: "events", pastEvents);
47
48      log.info("Loaded history of events for volunteer with email [{}]", userEmail);
49      return "history/history-events";
50    }
```

ДОДАТОК В

Програмна реалізація Пошук подій

```
@PostMapping("/search")
public String searchEvents(
    @ModelAttribute("searchCriteria") SearchCriteria searchCriteria,
    Model model,
    Authentication authentication) {

  log.info("Searching events with criteria: [{}]", searchCriteria);
  // Search events based on the provided search criteria
  List<Event> events = eventService.searchEvents(
      searchCriteria.getTitle(),
      searchCriteria.getStartDate(),
      searchCriteria.getEndDate(),
      searchCriteria.getLocation(),
      searchCriteria.getType());

  // Determine if a search was performed
  boolean isSearchPerformed = !searchCriteria.isEmpty();
  // Populate the model with the search results
  populateModel( toggle: "eventSearch", model, authentication, searchCriteria, events, isSearchPerformed);
  return "events/volunteer-events";
}
```

```
@GetMapping("/apply/{eventId}/{fromCard}")
public String applyForEvent(@PathVariable("eventId") Long eventId,
    @PathVariable("fromCard") Boolean fromCard,
    Authentication authentication) {
  // Retrieve the volunteer ID from the authentication object
  Long volunteerId = eventService.getVolunteerId(authentication.getName());
  // Create a new event application for the volunteer
  eventApplicationService.createEventApplication(eventId, volunteerId);

  log.info("Volunteer [{}] applied for event [{}]", volunteerId, eventId);

  // Redirect to the appropriate page based on the fromCard parameter
  if (fromCard) {
    return "redirect:/events/volunteer";
  }
  return "redirect:/events/volunteer/detail/" + eventId;
 }
}
```

```
@GetMapping("/update-application-status")
public String updateApplicationStatus(@RequestParam Long applicationId, @RequestParam String status) {
  EventApplication.ApplicationStatus newStatus = EventApplication.ApplicationStatus.valueOf(status);
  eventApplicationService.updateApplicationStatus(applicationId, newStatus);

  log.info("Updated application status for application ID: [{}] to status: [{}]", applicationId, newStatus);

  // Find the event ID to redirect back to the event detail page
  EventApplication application = eventApplicationService.findById(applicationId);

  // Redirect back to the event detail page and keep the participants view open
  return "redirect:/events/organization/detail/" + application.getEvent().getId() + "?showParticipants=true";
 }
}
```

ДОДАТОК Г

Програмна реалізація Переміщення подій на сторінку “History of events”

HistoryController.java | VolunteerEventsController.java | EventApplicationServiceImpl.java | DataInitializer.java

```java
package org.connectcares.charityconnect.controller;

import ...

@Controller
@RequestMapping("/history")
@RequiredArgsConstructor
@Slf4j
public class HistoryController {

  private final EventServiceIF eventService;
  private final ReviewServiceIF reviewService;
  private final UserServiceIF userService;

  @GetMapping("/volunteer")
  @PreAuthorize("hasRole('ROLE_VOLUNTEER')")
  public String showVolunteerHistory(Model model, Authentication authentication) {
    // Get the email of the authenticated user
    String userEmail = authentication.getName();
    // Retrieve the volunteer ID using the email
    Long volunteerId = eventService.getVolunteerId(userEmail);
    // Get the list of past events for the volunteer
    List<Event> pastEvents = eventService.getPastEventsByVolunteerId(volunteerId);
    // Add the list of past events to the model
    model.addAttribute( attributeName: "events", pastEvents);

    log.info("Loaded history of events for volunteer with email [{}]", userEmail);
    return "history/history-events";
  }
```

```java
  @GetMapping("/organization")
  @PreAuthorize("hasRole('ROLE_ORGANIZATION')")
  public String showOrganizationHistory(Model model, Authentication authentication) {
    // Get the email of the authenticated user
    String userEmail = authentication.getName();
    // Retrieve the organization ID using the email
    Long organizationId = eventService.getOrganizationId(userEmail);
    // Get the list of past events for the organization
    List<Event> pastEvents = eventService.getPastEventsByOrganizationId(organizationId);
    // Add the list of past events to the model
    model.addAttribute( attributeName: "events", pastEvents);

    log.info("Loaded history of events for organization with email [{}]", userEmail);
    return "history/history-events";
  }
}
```