Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка програмного забезпечення для планування та координації благодійних ініціатив»

Виконала: студентка 4 курсу
групи ЗПІ-21б спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Багнюк О.В.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Романюк О.В.
(прізвище та ініціали)

Рецензент: к.т.н., доц. каф. ЗІ Куперштейн Л.М.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ
д.т.н., проф. Романюк О. Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення

УЗГОДЖЕНО
Голова громадської організації «ЮС ГЕНТІУМ»
Івашківська А.О.
«21» березня 2025

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«21» березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Багнюк Ользі Віталіївні

1. Тема роботи – «Розробка програмного забезпечення для планування та координації благодійних ініціатив».

Керівник роботи: Романюк Оксана Володимирівна, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи: 2 червня 2025.

3. Вихідні дані до роботи: тип роботи – вебзастосунок; модель розробки – ітеративна модель; засоби організації даних програми – фреймворки Node та React; вхідні дані: інформаційні дані про благодійні заходи, збори, користувачів та їхню активність; вихідні дані: підтвердження та інформування користувачів, відображення персоналізованої інформації, відображення списку благодійних ініціатив та зборів; середовище розробки – Visual Studio Code; мова розробки – JavaScript; операційна система – Windows 8/10/11.

4. Зміст текстової частини: вступ; аналіз програмних систем для планування та координації благодійних ініціатив; розробка методів та алгоритмів роботи програми; розробка програмного забезпечення системи; тестування програми; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу: титульний слайд – автор, науковий керівник бакалаврської кваліфікаційної роботи, тема; актуальність розробки; мета, об'єкт та предмет дослідження; постановка задачі; наукова новизна; практична цінність; порівняльний аналіз аналогів; структура застосунку та дизайн; метод реєстрації на благодійний захід; метод побудови соціальної активності на платформі в реальному часі; модель застосунку у вигляді діаграми діяльності; модель застосунку у вигляді діаграми класів; алгоритм авторизації користувача та входу в особистий кабінет; алгоритм користувацької взаємодії з благодійними

зборами; алгоритм реєстрації на благодійний захід; модуль одноразової оплат
оформлення щомісячної підписки; модуль авторизації та реєстрації користува
валідацією даних; модуль для взаємодії з благодійними зборами; використ
технології; функціонал розробленої системи; апробація, публікація
впровадження результатів роботи; висновки; фінальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдан прийня
1-4	Романюк О.В., к.т.н., доцент кафедри ПЗ	24.03.25	01.06.2

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Приміт
1	Аналіз стану розвитку програм для планування та координації благодійних ініціатив	25.03.25 – 27.03.25	виконан
2	Розробка структури та дизайну вебзастосунку	28.03.25 – 30.03.25	виконан
3	Розробка методів, моделі та алгоритмів роботи програми	31.03.25 – 12.04.25	виконан
4	Аналіз та вибір засобів для реалізації програмного застосунку	13.04.25 – 18.04.25	виконан
3	Розробка програмного забезпечення	19.04.25 – 12.05.24	виконан
4	Тестування програмного застосунку	13.05.24 – 17.05.24	виконан
5	Оформлення матеріалів до захисту БКР	20.05.24 – 30.05.24	виконан

Студент _____ **Багнюк**
(підпис) (ініціали та пр

Керівник бакалаврської кваліфікаційної роботи _____ **Романюк**
(підпис) (ініціали та пр

АНОТАЦІЯ

УДК 004.42

Багнюк О. В. Розробка програмного забезпечення для планування та координації благодійних ініціатив: бакалаврська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 160 с.

На укр. мові. Бібліогр.: 45 назв; рис.: 98; табл.: 4.

У бакалаврській кваліфікаційній роботі аргументовано доцільність, проведено аналіз існуючих програмних рішень, спрямованих на підтримку організації благодійних заходів та ініціатив. Було досліджено переваги та недоліки наявних вебзастосунків, що дозволило обґрунтувати необхідність розробки власного інструменту для планування та координації благодійної діяльності.

Розроблено методи та алгоритми реалізації основних функціональних можливостей додатку, зокрема реєстрація на захід, побудова соціальної активності на платформі, користувацька взаємодія зі зборами, авторизація та реєстрація користувача, здійснення оплати та оформлення підписок. Програмне забезпечення реалізоване із використанням мови JavaScript, фреймворків Node та React, а також бази даних MongoDB. Як середовище розробки було використано Visual Studio Code. Здійснено тестування програмного забезпечення та підготовлено інструкцію користувача.

Отримані результати бакалаврської кваліфікаційної роботи можуть бути застосовані організаціями, що займаються благодійною діяльністю, для цифрової підтримки процесів планування та координації заходів та зборів кошт, взаємодії між користувачами, а також для моніторингу динаміки участі та представлення ключових показників активності користувачів.

Ключові слова: вебзастосунок, благодійність, JavaScript, Node, React, координація проєктів, ініціативи, благодійні заходи, збори коштів.

ABSTRACT

UDC 004.42

Bahniuk O. V. Development of Software for Planning and Coordination of Charitable Initiatives: Bachelor's Qualification Thesis in the Specialty 121 – Software Engineering, Educational Program – Software Engineering. Vinnytsia: VNTU, 2025. 160 p.

In Ukrainian. Bibliography: 45 sources; figures: 98; tables: 4.

This bachelor's qualification thesis substantiates the relevance and necessity of developing a software solution for organizing and coordinating charitable events and initiatives. The paper includes an analysis of existing software tools aimed at supporting charitable activities, examining their strengths and limitations, which formed the basis for the development of a custom solution.

The work presents methods and algorithms for implementing the application's key functionalities, including event registration, building social activity on the platform, user interaction with fundraisers, user authorization and registration, payment processing, and subscription management. The software is implemented using the JavaScript programming language, Node and React frameworks, and the MongoDB database. Visual Studio Code was used as the development environment. The software was tested, and a user manual was prepared.

The results of the bachelor's qualification thesis can be applied by organizations involved in charitable activities to digitally support the planning and coordination of events and fundraising campaigns, facilitate user interaction, monitor engagement dynamics, and visualize key user activity metrics.

Keywords: web application, charity, JavaScript, Node, React, project coordination, initiatives, charitable events, fundraising.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сучасні технології відіграють важливу роль у плануванні та координації благодійних ініціатив. Для успішного управління волонтерськими програмами, збором коштів та оптимальним розподілом ресурсів необхідні інноваційні цифрові рішення. Вони дозволяють автоматизувати численні процеси, полегшити взаємодію між усіма учасниками ініціатив та підвищити прозорість діяльності організацій. Розробка програмних модулів для благодійних організацій має значний потенціал для значного покращення їх роботи, збільшення комунікації та залучення нових учасників до соціально важливих проєктів [1].

Актуальність розробки програмного забезпечення для благодійної діяльності значно зросла з початком повномасштабної війни в Україні. Станом на 1 липня 2024 року в Україні зареєстровано понад 31 122 благодійну організацію, що на 52% більше порівняно з довоєнним періодом [2]. Велика кількість гуманітарних ініціатив, потреба в оперативному зборі та розподілі ресурсів, координація волонтерів та швидка реєстрація на благодійні ініціативи стали критично важливими завданнями. Зважаючи на ці виклики, розробка цифрової платформи стає важливим кроком для підвищення оперативності реагування на запити, забезпечення контролю та полегшення комунікації між користувачами [3].

Незважаючи на наявність певної кількості вже існуючих програмних рішень, більшість із них мають низку суттєвих недоліків. Наприклад, у багатьох рішеннях відсутні гнучкі механізми для регулярної фінансової підтримки, як-от можливість оформлення щомісячних внесків. Також бракує зручної системи фільтрації благодійних зборів кошт за напрямками допомоги, чи типами потреб. Окрім цього, платформи рідко надають повноцінний функціонал для взаємодії між користувачами. Наприклад, не передбачена можливість реєстрації на ініціативи, комунікації в реальному часі, перегляду відгуків чи залишення коментарів. В результаті проєкти є недостатньо відкритими до зворотного зв’язку та участі громади. Крім того, багато платформ орієнтовані лише на окремі функціональні

частини роботи, не надаючи повноцінного інтегрованого підходу до управління благодійними процесами.

Таким чином розробка програмного забезпечення для планування та координації благодійних ініціатив, яка враховує наявні недоліки, є на сьогодні надзвичайно актуальною. Особливо актуальним є питання підвищення продуктивності та рівня якості управління благодійними ініціативами за рахунок створення програмного забезпечення, що автоматизує процеси планування, керування та координації діяльності – зокрема шляхом організації подій, моніторингу активностей, якісній комунікації та збору коштів. Така система повинна об’єднувати волонтерів та користувачів на одній платформі, забезпечуючи прозору та зручну взаємодію.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення продуктивності та рівня якості управління благодійними ініціативами за рахунок розробки програмного забезпечення, що автоматизують процеси планування, управління та координації ініціатив, зокрема через організацію подій та аналіз результатів.

Основними **задачами дослідження** є:

– аналіз існуючих методів і засобів планування та координації благодійних ініціатив для визначення напрямків підвищення їх рівня якості та продуктивності;

– розробка структури програмного забезпечення;

– розробка дизайну інтерфейсу та аналіз принципів інклюзивності;

– розробка методу реєстрації на благодійний захід;

– розробка методу побудови соціальної активності на платформі в реальному часі;

– розробка моделі системи;

– розробка алгоритму авторизації користувача та входу в особистий кабінет;

– розробка алгоритму користувацької взаємодії з благодійними зборами;

– розробка алгоритму вибору та участі в благодійних інціативах;

– розробка програмного забезпечення модуля одноразової оплати та оформлення щомісячної підписки для підтримки благодійних зборів;

– розробка програмного забезпечення модуля авторизації та реєстрації користувачів з валідацією даних, модуля для взаємодії з благодійними зборами;

– тестування розробленого рішення з оцінкою його функціональної повноти, стабільності роботи, зручності інтерфейсу та ефективності взаємодії між учасниками платформи (волонтерами та користувачами);

– розробка інструкції користувача та визначення системних вимог.

Об’єкт дослідження – процес управління благодійними ініціативами з використанням сучасних інформаційних технологій.

Предмет дослідження – методи та засоби розробки програмного забезпечення для координації благодійних ініціатив, які забезпечують якісне управління та планування благодійних проєктів.

Методи дослідження. У процесі досліджень використовувались: теорія інформаційних систем та баз даних; теорія алгоритмів для розробки та опису алгоритмів роботи вебзастосунку; теорія програмної інженереї для розробки програмного забезпечення; методи тестування програмного забезпечення; методи проектування користувацького інтерфейсу (UI/UX Design Method); комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів:

1. Удосконалено метод реєстрації на благодійний захід, який на відміну від відомих аналогів включає повний цикл інтегрованої взаємодії між клієнтською та серверною частинами вебзастосунку з багаторівневою валідацією, підтвердженням участі через email із використанням бібліотеки nodemailer та унікального токена, а також механізм інформування адміністратора про нові заявки, що дало можливість підвищити рівень якості та продуктивності шляхом забезпечення безпеки, масштабованості та зручності обробки даних учасників в режимі реального часу.

2. Удосконалено метод побудови соціальної активності на платформі в реальному часі, який на відміну від відомих аналогів поєднує використання

WebSocket-технології (на базі Socket.IO) з функціоналом підписок, коментування та інтерактивної модерації через спеціалізовані компоненти React і хук useSocket, що дало можливість підвищити рівень якості та продуктивності шляхом забезпечення миттєвого обміну даними, гнучкої взаємодії користувачів і оперативної реакції адміністрації.

Практична цінність отриманих результатів полягає в тому, що на основі отриманих в бакалаврській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено програмне забезпечення для планування та координації благодійних ініціатив.

Впровадження. Впровадження результатів досліджень підтверджується відповідним актом та використовується в такій організації:

– громадська організація «ЮС ГЕНТІУМ» для підвищення продуктивності та якості управління благодійними ініціативами.

Особистий внесок здобувача. Усі наукові результати, викладені у БКР, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати: статистичний аналіз благодійних організацій в Україні, аналіз основних факторів зростання кількості благодійних ініціатив, таблиця порівняння функціональних характеристик аналогів [2]; обчислення узагальненого показника релевантності аналогів з урахуванням вагових коефіцієнтів критеріїв [3]; аналіз функціональності розробки методу реєстрації на благодійний захід, алгоритм реєстрації на благодійний захід [4]; аналіз проблем і підходів до інклюзовного дизайну, визначення принципів і практичних рішень [5].

Апробація матеріалів бакалаврської кваліфікаційної роботи. Результати бакалаврської кваліфікаційної роботи доповідалися на:

– Всеукраїнській науково-технічній конференції молодих вчених, аспірантів і студентів «Комп’ютерні ігри та мультимедіа як інноваційний підхід до комунікації» (Одеса, 2024 р.).

– Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (Вінниця, 2025р.).

– Міжнародній науково-практичній конференції «Молодь в науці: дослідження, проблеми, перспективи (Вінниця, 2025 р.).

Публікації. За тематикою дослідження опубліковано 4 наукові праці у збірниках матеріалів конференцій.

Структура та обсяг БКР. БКР Складається із вступу, чотирьох розділів, висновків, списку використаних джерел і додатків, у які винесено технічне завдання, протокол перевірки на наявність текстових запозичень, акт впровадження на підприємстві, лістинг програмного забезпечення й ілюстративний матеріал до захисту роботи.

1 АНАЛІЗ ПРОГРАМНИХ СИСТЕМ ДЛЯ ПЛАНУВАННЯ ТА КООРДИНАЦІЇ БЛАГОДІЙНИХ ІНІЦІАТИВ

1.1 Аналіз предметної області

Сучасний розвиток інформаційних технологій докорінно змінив підходи до управління благодійною діяльністю, зробивши процеси планування, координації та реалізації ініціатив більш ефективними, прозорими та масштабованими. Якщо на початкових етапах взаємодія з донорами та організаторами здійснювалася традиційно – через паперову документацію, особисті зустрічі чи телефонні дзвінки, то впровадження цифрових рішень значно розширило інструментарій благодійних організацій, надаючи нові можливості для автоматизації процесів і підвищення довіри з боку суспільства.

Поява спеціалізованих інформаційних систем стала наступним етапом еволюції, що дозволив автоматизувати облік фінансів, ведення баз даних донорів і волонтерів, формування звітності та контроль за використанням ресурсів [6]. На сьогодні розвиток IT у благодійності охоплює такі передові напрями, як штучний інтелект, блокчейн та хмарні обчислення. Штучний інтелект допомагає персоналізувати взаємодію з донорами, прогнозувати потреби, оцінювати ефективність проєктів, а блокчейн підвищує прозорість і запобігає зловживанням фінансами [7].

Мобільні додатки та платіжні сервіси дали змогу здійснювати пожертвування швидко й зручно, що значно розширило донорську базу. А соціальні мережі відкрили нові канали комунікації та мобілізації спільноти, посилили зворотний зв’язок, сприяли інформуванню про соціально важливі проблеми та залученню нових учасників до доброчинності.

Разом із перевагами цифровізації виникають і виклики. Найгострішим є питання кібербезпеки, адже витік особистих чи фінансових даних може спричинити серйозні наслідки. Малі організації часто мають обмеження в технічних і людських ресурсах, що ускладнює впровадження сучасних систем. Крім того, зростає потреба у зміцненні довіри з боку громадськості через випадки

шахрайства в секторі [8].

Перспективи розвитку інформаційних систем для благодійних ініціатив включають активніше використання машинного навчання, розширення функцій фінансового моніторингу та прозорої звітності. Вже зараз благодійні організації впроваджують цифрові платформи з можливістю відстеження транзакцій у реальному часі, оцінки соціального впливу та підвищення відкритості всіх процесів.

Одним із ключових напрямів удосконалення є розробка програмного забезпечення, що дозволить автоматизувати рутинні процеси, зменшити адміністративне навантаження, підвищити точність і прозорість фінансових операцій. Зокрема, це стосується обліку пожертв, керування базами даних, створення звітів, планування проєктів і моніторингу ефективності. Автоматизація таких процесів дозволяє організаціям зосередити ресурси на стратегічних напрямах діяльності та досягати вагоміших соціальних результатів.

Актуальність ефективного управління благодійністю в Україні зросла особливо після початку повномасштабного вторгнення у 2022 році. Станом на 1 липня 2024 року в Україні зареєстровано понад 31 122 благодійних організації, що на 52% більше, ніж у довоєнний період [9].

Загальна кількість неприбуткових організацій сягає 208 385, з яких благодійні становлять майже 10%. Найбільше таких організацій зареєстровано в Києві (11,8%), на другому місці – Львівська область (7,8%), на третьому – Дніпропетровська (7,3%) [9].

Діаграма розподілу кількості благодійних організацій в Україні за 2021-2024 роки зображена на рисунку 1.1.

Ці статистичні дані свідчать про суттєве зростання інтересу до благодійної діяльності серед населення та бізнесу, що, у свою чергу, створює потребу в більш ефективних інструментах управління й координації діяльності таких організацій. З огляду на географічну концентрацію та стрімке збільшення кількості благодійних структур, стає очевидним, що традиційні методи ведення звітності та взаємодії з донорами та бенефіціарами вже не відповідають сучасним вимогам. Це підсилює

актуальність розробки цифрових рішень, які дозволяють автоматизувати процеси, підвищити прозорість та забезпечити зручну комунікацію, а також створити єдину екосистему для ефективної координації благодійних ініціатив.

Рисунок 1.1 – Діаграма розподілу кількості благодійних організацій в Україні за 2021-2024 роки

Зростання кількості благодійних ініціатив обумовлене не лише війною, а й іншими важливими соціальними проблемами [2]:

1. Допомога хворим людям. Щорічно тисячі українців потребують коштів на складне лікування чи трансплантацію. Незважаючи на війну, галузь трансплантації активно розвивається. Близько 5 000 людей щороку потребують пересадки органів.

2. Захист тварин. Проблема безпритульних тварин загострилася через війну. Благодійні організації створюють притулки, допомагають із лікуванням, стерилізацією та пошуком нових домівок. Онлайн-платформи прискорюють збір коштів і підбір власників для тварин.

3. Підтримка дітей-сиріт та вразливих груп. Благодійники забезпечують гуманітарну, психологічну та освітню підтримку. У 2023 році кількість дітей, позбавлених батьківського піклування, становила 45 936. Станом на кінець 2024 року було усиновлено 1 270 дітей – на 27% більше, ніж у 2023 році [10].

4. Екологічні ініціативи. Благодійні проєкти боряться з незаконною вирубкою лісів, забрудненням водойм, розвивають сортування та переробку відходів. Лише за 4 місяці 2025 року зафіксовано більше 1,5 тис.випадків незаконної вирубки на 10 тис. куб. м лісу. Заподіяна шкода склала 302,2 млн грн, але добровільно відшкодовано лише 2% [11].

З огляду на стрімке зростання кількості благодійних ініціатив і волонтерських проєктів, дедалі актуальнішою стає потреба у гнучких програмних інструментах, здатних забезпечити автоматизацію ключових процесів: планування подій, координації роботи волонтерів, ведення обліку пожертв і формування звітності. Застосування сучасних вебтехнологій сприяє підвищенню рівня якості реалізації благодійних програм. Поєднання таких технологічних рішень у межах єдиної цифрової платформи дозволяє налагодити централізоване управління та забезпечити злагоджену взаємодію між усіма учасниками процесу.

1.2 Аналіз аналогів розроблюваного програмного забезпечення

З розвитком благодійності та зростанням кількості волонтерських проєктів, у суспільстві виникає потреба у використанні ефективних технологічних рішень. Сучасні програми та платформи можуть значно полегшити управління подіями, організацію волонтерської роботи та облік коштів, що пожертвують доброчинці. Використання інноваційних технологій дозволяє організаторам зберігати прозорість та підвищити ефективність кампаній, знижуючи адміністративні витрати і забезпечуючи легкий доступ до інформації для всіх учасників. Важливо, щоб такі рішення були доступними і зрозумілими для широкої аудиторії, незалежно від рівня їхніх технічних навичок.

Це підкреслює необхідність створення програмного забезпечення, яке забезпечить інтеграцію всіх елементів благодійного процесу – від створення кампаній до моніторингу результатів, що дозволяє спростити і покращити взаємодію між волонтерами, донорами та організаторами. У результаті цього процесу благодійні ініціативи можуть досягти нових висот, забезпечуючи більшу ефективність та досягнення поставлених цілей.

Отже, зі збільшенням кількості благодійних проєктів та волонтерських ініціатив стає очевидною потреба у гнучких і ефективних програмних рішеннях, які дозволяють автоматизувати управління подіями, координацію волонтерів, облік пожертв і складання звітів. Використання новітніх вебтехнологій, мобільних додатків і алгоритмів для аналізу даних відкриває нові можливості для підвищення результативності благодійних проєктів. Інтеграція цих інструментів в єдину платформу дозволяє централізовано управляти процесами та забезпечувати взаємодію між усіма учасниками ініціатив [3].

Для оцінки актуальності розробки програмного забезпечення для координації благодійних ініціатив варто звернути увагу на існуючі вебресурси, зокрема: «ДоброДій», «HelpKarma», «Зграя» та «Peremoga».

«ДоброДій» – вебресурс для підтримки благодійних ініціатив, що дозволяє організаторам створювати кампанії, залучати волонтерів та отримувати фінансову допомогу від донорів (див. рисунок 1.2). Сервіс орієнтований на підвищення прозорості збору коштів і спрощення взаємодії між учасниками благодійних заходів [12].

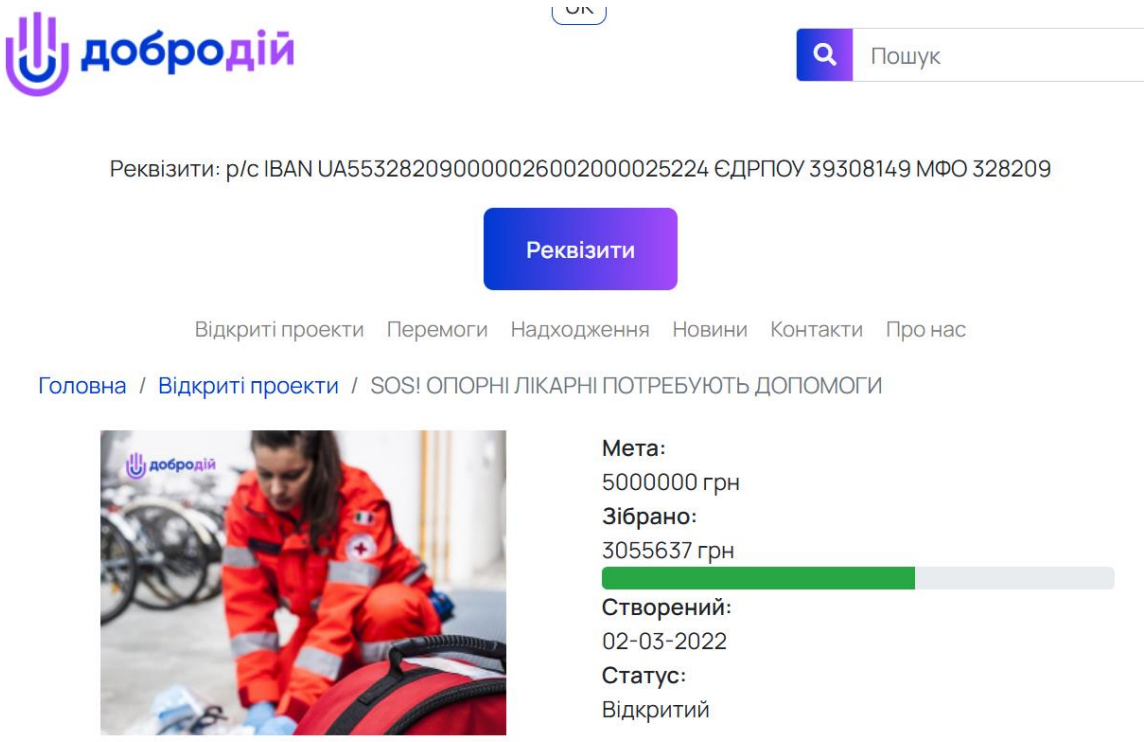


Рисунок 1.2 – Вебсторінка збору коштів на благодійній платформі «ДоброДій»

«HelpKarma» – міжнародна платформа для збору коштів на соціальні та благодійні ініціативи, що дозволяє створювати кампанії, поширювати їх серед потенційних донорів та отримувати підтримку (див. рисунок 1.3). Включає механізм рейтингування організаторів для підвищення довіри до благодійних кампаній [13].

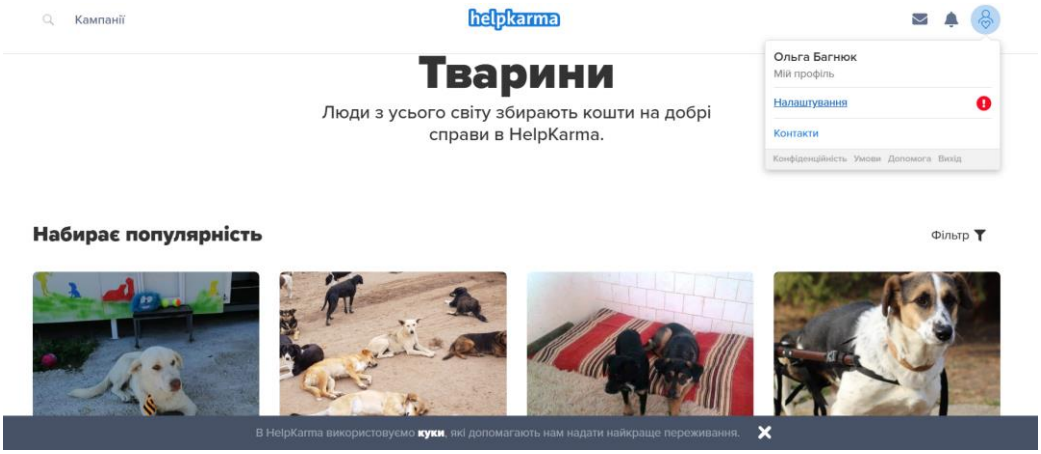


Рисунок 1.3 – Вебсторінка обраної категорії «Тварини» міжнародної платформи «HelpKarma»

«Зграя» – благодійний проєкт, який об'єднує донорів, меценатів та організації, що потребують допомоги (див. рисунок 1.4). Платформа дозволяє фінансувати різні ініціативи, підтримуючи проєкти в сферах охорони здоров'я, освіти, соціального захисту та розвитку громад [14].

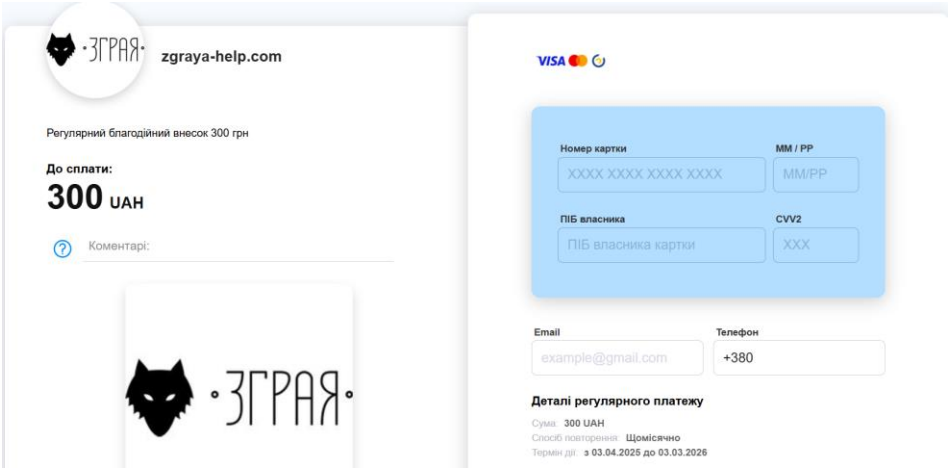


Рисунок 1.4 – Вебсторінка оформлення регулярного внеску проєкту «Зграя»

«Peremoga» – благодійний фонд, що об'єднує людей, лідерів, бізнеси та кожного небайдужого для створення проєктів майбутнього України. Фонд реалізує низку проєктів, спрямованих на підтримку Збройних сил України, внутрішньо переміщених осіб та дітей (див. рисунок 1.5). Також активно співпрацює з іншими організаціями в рамках ініціативи Charity Marketing для реалізації спільних благодійних проєктів [15].

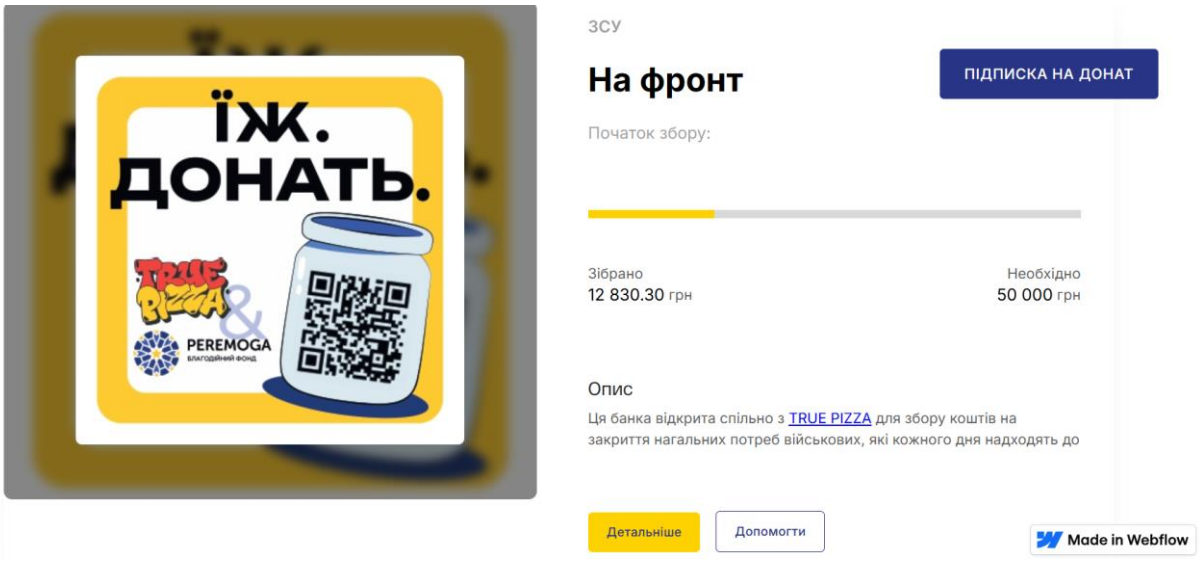


Рисунок 1.5 – Вебсторінка «Термінові збори» благодійного фонду «Peremoga»

Для наочного порівняння розглянутих вебресурсів було зведено їх переваги та недоліки у таблицю порівняння (див. таблицю 1.1), що дозволяє виявити їхні ключові особливості, сильні сторони та можливі недоліки. Аналіз аналогічних ресурсів є важливим етапом, оскільки він допомагає визначити найкращі практики, уникнути помилок, притаманних іншим проєктам, та врахувати можливості для покращення. Це сприяє розробці більш ефективного та конкурентоспроможного рішення, що відповідатиме потребам користувачів і сучасним вимогам ринку [2].

Крім того, порівняльний аналіз дає змогу оцінити рівень інноваційності кожного ресурсу та визначити напрями для подальшого вдосконалення. Такий підхід дозволяє створити більш ефективну стратегію розвитку, орієнтовану на покращення користувацького досвіду та підвищення результативності роботи платформи.

Таблиця 1.1 – Порівняльна характеристика аналогів

Функціональні можливості	ДоброДій	HelpKarma	Зграя	Peremoga	Власна розробка
Фільтрація благодійних зборів за категоріями	0	1	0	1	1
Особистий кабінет	0	1	0	0	1
Функціональність підписок і активності користувачів	0	0	0	0	1
Функціонал підписки на щомісячну підтримку	0	1	1	1	1
Рейтинг організаторів	0	1	0	0	1
Онлайн-пожертвування	1	1	1	1	1
Система взаємодії користувачів з підтримкою чат-бота	0	0	0	0	1
Функціональність реєстрації та супроводу участі в заходах	0	0	0	0	1
Сумарний коефіцієнт	1	5	2	3	8

Аналізуючи таблицю 1.1, можна зазначити, що власна розробка має вищий сумарний коефіцієнт порівняно з міжнародною платформою «HelpKarma» на 16% (100% - 5/8*100% = 38%), благодійним фондом «Peremoga» на 63%, благодійним проєктом «Зграя» на 75% та вебресурсом «ДоброДій» на 88% [3].

Високий сумарний коефіцієнт свідчить про наявність у власній розробці ширшого функціонального наповнення, що охоплює як базові можливості, так і нові інноваційні рішення. Платформа включає не лише підтримку онлайн-пожертвувань, яка є спільною для всіх розглянутих систем, а й інші важливі компоненти, що відсутні в більшості аналогів. Зокрема, це можливість фільтрації зборів за категоріями, наявність особистого кабінету користувача,

функціональність підписок та активностей, а також підтримка щомісячної фінансової допомоги.

Крім того, система містить рейтинг організаторів, що сприяє прозорості процесів і формуванню довіри серед користувачів. Інтеграція чат-бота забезпечує оперативну взаємодію з користувачами, тоді як функція реєстрації та супроводу участі в заходах відкриває можливості для організації волонтерської активності та масових подій. Таким чином, власна розробка не лише відповідає актуальним вимогам у сфері благодійності, а й перевершує існуючі рішення за рахунок комплексного підходу до управління всіма аспектами благодійних ініціатив.

Інтеграція ключових функцій в єдине рішення усуває проблему розпорошеності благодійних ініціатив між різними платформами та каналами зв’язку. Централізована система для створення подій, координації волонтерів і збору пожертв значно спрощує управління ресурсами та покращує комунікацію між усіма учасниками.

Крім того, запропоноване рішення має значний потенціал для подальшого розширення та адаптації відповідно до потреб благодійних організацій. Інтеграція з фінансовими сервісами, аналітичними інструментами та системами автоматизованого управління допоможе оптимізувати процеси та забезпечити стабільну роботу платформи [3].

Таким чином, результати аналізу підтверджують доцільність створення спеціалізованого програмного забезпечення для координації благодійних ініціатив, що дозволить вирішити існуючі проблеми, зробить процеси більш прозорими, а взаємодію учасників – зручнішою та ефективнішою.

1.3 Аналіз методів розв’язання задачі

Розробка вебзастосунку для планування та координації благодійних ініціатив потребує комплексного підходу до вибору методів реалізації, які забезпечать ефективність, масштабованість та зручність використання системи. На сучасному етапі існує низка перевірених рішень, які можуть бути адаптовані до специфіки благодійної діяльності.

Перш за все, розробка таких систем базується на клієнт-серверній архітектурі, що забезпечує поділ між користувацьким інтерфейсом (фронтендом) та логікою обробки даних (бекендом). Це дозволяє забезпечити гнучке масштабування, розділення відповідальності між модулями та спрощення подальшого супроводу [16].

Серед поширених методів реалізації користувацької взаємодії використовуються компонентно-орієнтовані підходи, які дозволяють створювати динамічні інтерфейси, що адаптуються до потреб різних категорій користувачів – організаторів, волонтерів, учасників. Особлива увага приділяється побудові інтуїтивного інтерфейсу, який сприяє залученню навіть тих користувачів, що не мають глибоких технічних знань.

Для реалізації логіки планування та сповіщення актуальними є подієво-орієнтовані підходи, які дозволяють ефективно обробляти події, пов’язані з реєстрацією, оновленням заходів, змінами статусів, нагадуваннями тощо. У системах, орієнтованих на взаємодію, особливе значення має реалізація асинхронної обробки даних, зокрема для забезпечення повідомлень у реальному часі, оновлення списків подій, зборів чи коментарів [17].

У сфері координації користувачів важливу роль відіграють методи, пов’язані з організацією доступу до даних – реалізація ролей, рівнів доступу, фільтрації інформації. Такі методи дозволяють забезпечити як прозорість інформації, так і її захищеність.

Також доцільно звернути увагу на інтеграційні можливості – системи повинні підтримувати підключення до зовнішніх платформ для оплати, поштових сервісів, картографічних сервісів (для візуалізації місць проведення заходів), що досягається шляхом застосування API-орієнтованого підходу.

У контексті сучасних вимог важливим є застосування адаптивних методів проєктування – система повинна залишатися працездатною як на стаціонарних пристроях, так і на мобільних, що забезпечується застосуванням технологій адаптивної верстки та mobile-first дизайну.

Загалом, аналіз методів розв’язання задачі показує, що ефективна система планування та координації благодійних ініціатив повинна поєднувати інструменти організації подій, гнучкого управління учасниками, інтерактивного спілкування та прозорої звітності. Отже, саме тому було вирішено поєднати більшість з описаних методів у єдиній вебплатформі, що дає змогу підвищити рівень якості та продуктивності благодійної діяльності, полегшити організаційні процеси та зміцнити взаємодію між учасниками спільноти.

1.4 Постановка задач дослідження

Якісне планування та координація благодійних заходів вимагають впровадження сучасних і зручних цифрових інструментів, здатних оптимізувати ключові процеси – від організації подій до залучення волонтерів і збору коштів. Дослідження наявних рішень у цій сфері засвідчило низку істотних недоліків. Зокрема, більшість платформ мають обмежену або зовсім відсутню інтеграцію з фінансовими сервісами, не пропонують зручних механізмів для взаємодії між користувачами, а також не забезпечують гнучкого налаштування подій чи зборів відповідно до специфіки кожної ініціативи.

Ще одним суттєвим обмеженням є відсутність чіткого розподілення між благодійними заходами та зборами коштів. У більшості систем користувач може лише надати загальну допомогу, не маючи змоги окремо зареєструватися на конкретний захід з зазначенням дати, часу та місця його проведення, а також отримати автоматичні нагадування. Водночас функціонал для зборів також часто є обмеженим: користувачі не мають можливості залишати відгуки, ставити оцінки, обговорювати важливі питання в коментарях або детально ознайомлюватися з інформацією про організатора збору. Крім того, відсутні функції грошової підтримки у вигляді разового внеску чи підписки, а також персоніфікована історія активності. Усе це суттєво знижує якість користувацького досвіду та ускладнює прозорість благодійної діяльності.

У зв’язку з цим виникла нагальна потреба у створенні нової вебсистеми, яка б ураховувала сучасні вимоги користувачів, сприяла підвищенню якості

організаційної діяльності, забезпечувала зручну комунікацію та розширений функціонал як для волонтерів, так і для тих, хто потребує допомоги.

З урахуванням вищезазначеного, було визначено основні завдання, які необхідно реалізувати для створення продуктивного програмного забезпечення:

– провести аналіз існуючих методів і засобів планування та координації благодійних ініціатив для визначення напрямків підвищення їх рівня якості та продуктивності;

– розробити структуру програмного забезпечення;

– розробити дизайн інтерфейсу та провести аналіз принципів інклюзивності;

– розробити метод реєстрації на благодійний захід;

– розробити метод побудови соціальної активності на платформі вреальному часі;

– розробити модель системи;

– розробити алгоритм авторизації користувача та входу в особистий кабінет;

– розробити алгоритм користувацької взаємодії з благодійними зборами;

– розробити алгоритм вибору та участі в благодійних інціативах;

– розробити програмне забезпечення модуля одноразової оплати та оформлення щомісячної підписки для підтримки благодійних зборів;

– розробити програмне забезпечення модуля авторизації та реєстрації користувачів з валідацією даних, модуля для взаємодії з благодійними зборами;

– провести тестування розробленого рішення з оцінкою його функціональної повноти, стабільності роботи, зручності інтерфейсу та ефективності взаємодії між учасниками платформи (волонтерами та користувачами);

– розробити інструкцію користувача та описати системні вимоги.

Реалізація зазначених завдань створює основу для формування зручної, функціональної та високопродуктивної системи, орієнтованої на ефективне планування та координацію благодійних ініціатив. Якість і продуктивність системи підвищуються завдяки врахуванню актуальних потреб користувачів, оптимізації взаємодії між усіма учасниками процесу та усуненню недоліків, характерних для наявних рішень у цій сфері.

Очікується, що впровадження такої системи не лише полегшить процес організації заходів, а й сприятиме залученню ширшого кола користувачів. Система забезпечить формування активного співтовариства, яке спільно працюватиме над досягненням суспільно важливих цілей, водночас підвищуючи результативність збору коштів та матеріальної допомоги.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У першому розділі було проведено комплексний аналіз сучасного стану технологій, що застосовуються у сфері планування та координації благодійних ініціатив, а також обґрунтовано необхідність розробки нових програмних рішень, які б відповідали актуальним викликам.

Розглянуто етапи еволюції цифрових технологій у благодійному секторі – від традиційних підходів до сучасних цифрових платформ, що інтегрують штучний інтелект, блокчейн, великі дані, хмарні обчислення, мобільні додатки та соціальні мережі. Детально охарактеризовано переваги впровадження таких інструментів – підвищення прозорості, автоматизація процесів, оптимізація ресурсів, побудова довіри та покращення взаємодії з донорами і волонтерами.

Здійснено порівняльний аналіз чотирьох вебресурсів, які виступають аналогами програмного рішення для координації благодійних ініціатив: «ДоброДій», «HelpKarma», «Зграя», «Peremoga». Для кожного аналогу проаналізовано функціональність, переваги, напрямки діяльності та рівень взаємодії з користувачами. Виявлено, що жодна з наявних платформ не забезпечує повну інтеграцію всіх етапів благодійної діяльності – від планування до моніторингу результатів. Це стало підґрунтям для формулювання задач дослідження та розробки нового рішення, яке об’єднає координацію волонтерів, управління ресурсами, контроль фінансів і прозорість взаємодії в єдиному інтерфейсі.

2 РОЗРОБКА МЕТОДІВ ТА АЛГОРИТМІВ РОБОТИ СИСТЕМИ

2.1 Аналіз вхідних та вихідних даних програмного застосунку

Будь-яка програма базується на обробці вхідних і вихідних даних. У вебзастосунку для планування та координації благодійних ініціатив джерелом вхідних даних виступає користувач, який вводить інформацію через графічний інтерфейс.

Система обробляє різноманітні категорії вхідних даних, ключовими з яких є: інформація про користувачів, благодійні збори та заходи, зібрані кошти, локації проведення заходів, а також звіти про виконання. Дані користувачів охоплюють реєстраційні відомості, історію участі у ініціативах та індивідуальні налаштування профілю. Для забезпечення конфіденційності та захисту особистої інформації реалізовано механізм автентифікації на основі токенів, із використанням шифрування та сучасних методів авторизації.

Вихідними даними є повідомлення про підтвердження успішної реєстрації та подачі заявки, доступ до персонального кабінету, повідомлення про зміни у профілі тощо.

Інформація про благодійні збори зберігається у структурованому вигляді, що включає назву, опис, категорію допомоги (наприклад, постраждалі від війни, ЗСУ, діти-сироти тощо), відгуки та функцію уподобання. Також включає дані про відповідального волонтера, дату публікації збору, рейтинг волонтера стосовно кожного збору, а також потреби й цілі проєкту. Реалізовано функціонал пошуку та фільтрації ініціатив за різними критеріями, що полегшує навігацію.

Управління участю в заходах забезпечує можливість користувачеві ознайомлюватися з актуальними подіями, реєструватися на обрані заходи, переглядати інформацію про учасників, місце проведення, час, а також зберігати ці дані у власному профілі. Робота підсистеми базується на обробці відповідних вхідних та вихідних даних.

До вхідних даних належать відомості про сам захід (назва, опис, дата та час проведення, місце розташування із зазначенням географічних координат для

візуалізації на карті, категорія заходу, статус активності) та персональні дані користувача, який подає заявку (ПІБ, електронна пошта, номер телефону, додатковий коментар або повідомлення). Ця інформація вводиться під час реєстрації на захід через відповідну форму.

У результаті обробки цих даних система формує вихідну інформацію, яка відображається в інтерфейсі користувача. Зокрема, після реєстрації дані про захід автоматично додаються до особистого кабінету у вкладку «Ініціативи», де вони групуються за категоріями: «Майбутні» та «Минулі». Користувач отримує підтвердження реєстрації, може переглядати деталі події, список учасників (якщо ця опція активна), а також скасувати участь у разі потреби. Захід також може бути синхронізований з календарем, що дозволяє краще планувати участь у доброчинних ініціативах. Крім того, реалізовано можливість поширення інформації про подію серед інших користувачів за допомогою зовнішніх каналів, зокрема соціальних мереж або електронної пошти.

Додатково, система включає модуль реєстрації та авторизації з розподілом прав доступу, який дозволяє створювати облікові записи через email. Процес авторизації включає використання надійних методів шифрування та автентифікації, що гарантує безпеку даних користувачів.

Функціонал соціальної взаємодії між користувачами вебресурсу реалізовано як важливу складову платформи, що сприяє підвищенню активності, персоналізації взаємодії та формуванню спільноти навколо благодійних ініціатив. Основне призначення цього модуля – забезпечити механізми підписки, сповіщень, публічного перегляду активності та комунікації між користувачами у межах системи.

Вхідні дані включають інформацію про ініціатора взаємодії (ідентифікатор користувача, який виконує дію), об’єкт дії (ідентифікатор користувача, на якого відбувається підписка), а також службову інформацію, необхідну для реєстрації події у базі даних (дата, час, тип взаємодії). У разі коментування додатково обробляється текст повідомлення, ідентифікатор збору або заходу, до якого

залишено коментар, та інформація про коментар, на який надається відповідь (у випадку вкладеного обговорення).

У відповідь на виконану дію система формує вихідні дані, що відображаються у вигляді інтерактивних елементів інтерфейсу. Зокрема, при підписці на користувача його ім’я з’являється в розділі «Підписки» ініціатора, а також у розділі «Підписники» профілю того, на кого здійснено підписку. Користувач, на якого підписалися, отримує відповідне сповіщення з можливістю взаємної підписки, що дає змогу стежити за оновленнями і публічною активністю іншого учасника.

У профілі кожного користувача доступна інформація, дозволена для перегляду, зокрема список його опублікованих благодійних зборів коштів, активних і минулих заходів, дата реєстрації на платформі, контактна інформація (якщо користувач надав дозвіл на її публікацію). Це дозволяє іншим користувачам оцінити рівень залученості, підтримати актуальні ініціативи або долучитися до майбутніх заходів.

Окрему роль у забезпеченні соціальної взаємодії відіграє система коментування. Користувачі можуть залишати коментарі під зборами або заходами, а також відповідати на коментарі інших користувачів, що створює динамічну структуру обговорення. Реалізація вкладених відповідей дає змогу підтримувати логічну структуру діалогу, що особливо актуально при великій кількості повідомлень. Усі нові коментарі відображаються у режимі реального часу, що сприяє живій комунікації без необхідності оновлення сторінки.

Окремим елементом платформи є реалізований онлайн-бот для взаємодії з користувачами, що інтегрований безпосередньо в інтерфейс вебресурсу. Основне призначення цього інструменту – забезпечення швидкої, зручної та доступної комунікації між системою та користувачем у режимі реального часу без необхідності встановлення сторонніх додатків або переходу на інші платформи.

Онлайн-бот функціонує у вигляді інтерактивного вікна на сайті, з яким користувач може вести текстове спілкування. Вхідними даними є повідомлення, які користувач надсилає у вікні чату – це можуть бути питання, ключові слова або

прості команди (наприклад, «Хочу зареєструватися на захід», «Допомога» тощо). Бот обробляє запит і формує відповідь залежно від контексту та інтегрованої логіки взаємодії.

У процесі роботи бот виконує кілька важливих функцій: надає загальну довідкову інформацію, допомагає зорієнтуватися в інтерфейсі платформи, перенаправляє користувача до потрібного розділу (наприклад, реєстрації на захід або перегляду зборів), а також може повідомляти про нові можливості або нагадування.

Модуль оплати системи дозволяє користувачам здійснювати одноразові пожертви або оформляти підписку на щомісячну допомогу. Користувачі можуть вибирати конкретну суму для одноразових переказів або встановлювати автоматичну щомісячну пожертву на обрану суму. Система інтегрована з платіжними сервісами, що забезпечує зручний і безпечний процес оплати.

Інтерфейс системи є інтуїтивно зрозумілим, зручним для користувачів та оптимізованим для ефективної роботи на різних пристроях. Він забезпечує швидкий доступ до основного функціоналу, дозволяє керувати подіями, переглядати статистику та історію активності. Також реалізовано систему рейтингу й відгуків, яка надає можливість оцінювати якість заходів та надійність організаторів, формуючи загальний рейтинг довіри в системі.

Отже, описана система для планування та координації благодійних ініціатив є комплексним вебзастосунком, який ефективно обробляє різноманітні вхідні та вихідні дані користувачів, забезпечуючи їх зручний та безпечний доступ до функцій платформи.

2.2 Розробка структури програмного забезпечення

Розробка структури програмного забезпечення є одним із ключових етапів у процесі створення вебзастосунку. Вона передбачає формування логічної, функціональної та навігаційної архітектури, яка дозволяє забезпечити зручність користування, масштабованість системи та спрощення подальшої підтримки коду. Структуроване ПЗ дає змогу розподілити обов’язки між компонентами, запобігти

дублюванню функціоналу, зменшити ризики помилок та забезпечити зрозумілу логіку навігації як для користувача, так і для розробника.

Діаграма варіантів використання (use case diagram) є важливим інструментом при проєктуванні програмного забезпечення, особливо на етапі визначення функціональних вимог. Вона дозволяє візуально представити взаємодію користувача з системою, окреслити основні сценарії поведінки та визначити, які саме функції мають бути реалізовані у застосунку. За допомогою цієї діаграми можна чітко встановити ролі користувачів, описати, які дії вони можуть виконувати, і сформувати базу для подальшої розробки [18].

Діаграма варіантів використання, зображена на рисунку 2.1, відображає ключові функціональні сценарії взаємодії трьох основних акторів у системі благодійного вебзастосунку: користувача (донора), волонтера (організатора) та сервера (системного процесора).

Користувач (донор) має змогу реєструватися в системі, проходити процес автентифікації та входити в особистий кабінет. У межах особистого кабінету він може змінювати свої контактні дані, переглядати ініціативи, які його зацікавили, та налаштовувати профіль тощо. Основна функціональність користувача полягає у взаємодії з благодійними заходами та зборами. Він може переглядати списки актуальних зборів і заходів, обирати конкретні ініціативи, здійснювати пошук за ключовими словами та фільтрами, переглядати детальну інформацію про збори, а також залишати відгуки, коментарі. Окрім одноразових внесків, користувач може оформити щомісячну підписку та зареєструватися на конкретний захід. При заповненні форм користувач проходить процес верифікації введених даних.

Волонтер (організатор) взаємодіє з системою переважно з адміністративною метою. Він має змогу створювати обліковий запис, формувати нові благодійні збори або заходи, вказуючи їх опис, ціль та інші параметри. Також волонтер може переглядати активні заявки користувачів, підтверджувати їх участь, надсилати форми з відповідною інформацією. Деякі функції волонтера включають обробку анкет для участі в заходах, перегляд інформації про інших волонтерів, а також здійснення підписки на них або подію.

Сервер виступає посередником між діями користувача й волонтера та внутрішніми процесами обробки інформації. Його основний узагальнений прецедент – «Обробка запитів користувача й волонтера» – включає в себе верифікацію введених даних, перевірку облікових даних, збереження даних до бази, оновлення або видалення інформації, а також надання відповіді системі після отримання даних з бази. Цей прецедент використовується майже в усіх варіантах використання обох акторів – користувача та волонтера, і реалізується через зв’язок <<include>>.

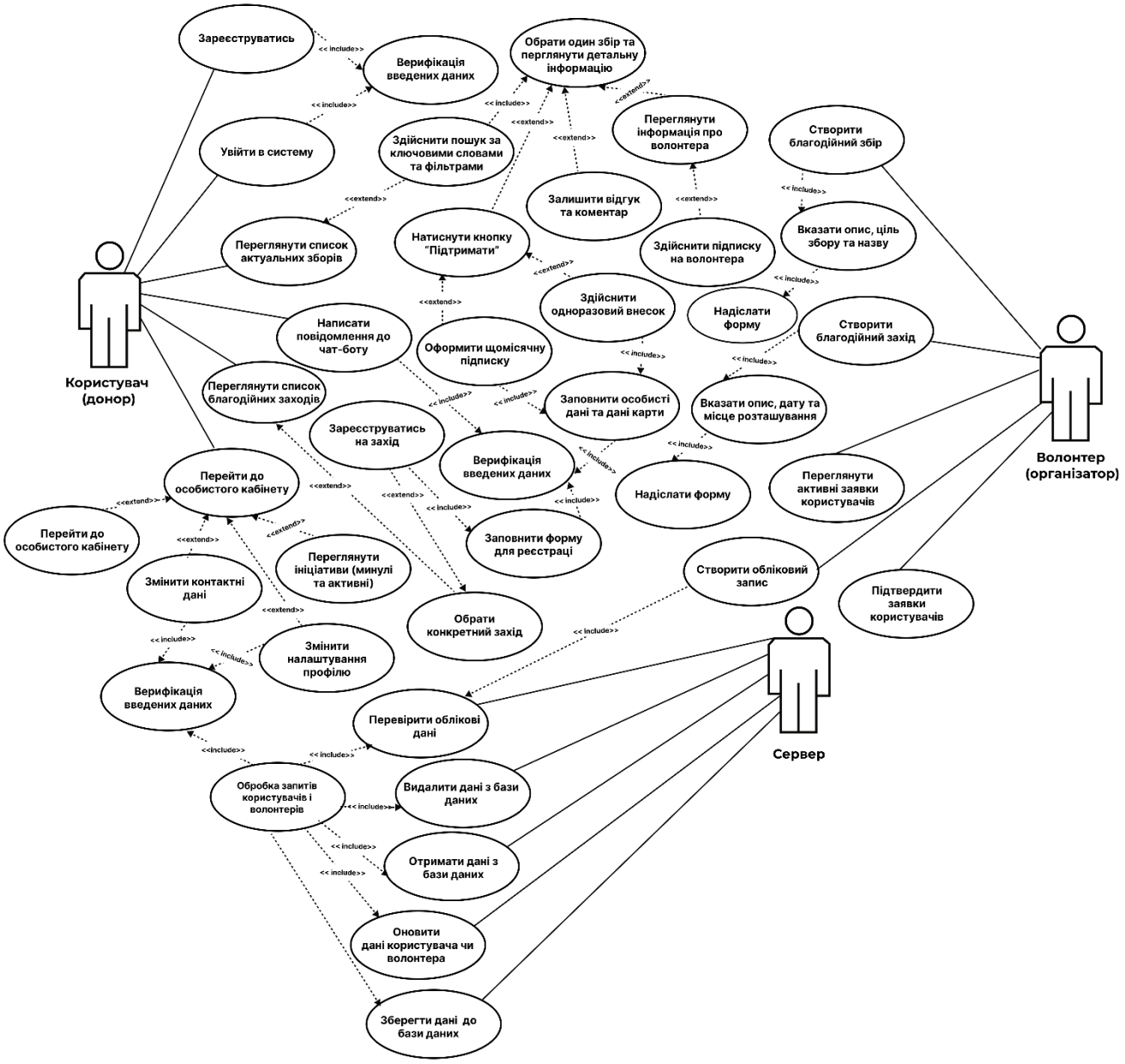


Рисунок 2.1 – Діаграма варіантів використання

Таким чином, діаграма демонструє узгоджену й логічну структуру роботи системи, в якій усі дії користувачів та волонтерів підпорядковуються централізованій серверній обробці. Це забезпечує надійність, контрольованість та прозорість процесів у межах благодійного вебзастосунку.

Проєктування інтерфейсу користувача є невід'ємною частиною розробки структури програмного забезпечення, адже саме воно визначає, як користувачі взаємодіятимуть із системою. Інтерфейс має бути інтуїтивно зрозумілим, доступним і зручним, особливо коли йдеться про соціально важливі сервіси, як-от благодійні вебзастосунки. Правильно спроєктований інтерфейс не лише підвищує задоволеність користувача, а й безпосередньо впливає на ефективність виконання функціональних задач – наприклад, підтримку збору чи реєстрацію на захід.

Особливо важливу роль інтерфейс відіграє у створенні логічної навігаційної архітектури, що є продовженням загальної структури програмного забезпечення. Через інтерфейс реалізується доступ до основних функцій – перегляду зборів, надсилання форм, реєстрації, взаємодії з волонтерами. Чітке візуальне розділення логічних блоків, наявність закликів до дії (наприклад, кнопки «Підтримати» або «Зареєструватися»), мінімізація зайвих кроків для досягнення мети – усе це результат ретельного проєктування інтерфейсу.

Одним із ключових етапів на шляху до створення повноцінного користувацького інтерфейсу є розробка прототипу. Прототип – це попереднє візуальне відтворення вигляду сторінки або системи загалом, яке демонструє розташування елементів інтерфейсу та логіку переходів між ними. Він може бути у вигляді схематичних ескізів (wireframe) або вже графічно опрацьованих макетів. Прототип використовується для візуалізації ідей, обговорення функціональності з командою, тестування логіки взаємодії та подальшого вдосконалення інтерфейсу ще до початку безпосередньої розробки [19].

Для благодійного вебзастосунку створення прототипу є особливо важливим, адже користувачі повинні мати можливість швидко орієнтуватися в системі, знаходити актуальні збори, переглядати події та без труднощів здійснювати внески. Прототип допомагає передбачити потенційні труднощі у взаємодії з платформою,

зменшити ризики непорозумінь між технічною та дизайнерською командами, а також значно прискорити процес узгодження й реалізації інтерфейсу.

У межах цієї роботи було створено прототип головної сторінки, який відображає основні напрями взаємодії з користувачем. Зокрема, прототип передбачає доступ до ключових функцій та слугує фундаментом для подальшої розробки всієї інтерфейсної частини системи (див. рисунок 2.2).

Рисунок 2.2 – Прототип головної сторінки вебзастосунку

Інтерфейс логічно поділений на верхню навігаційну панель, центральний блок мотиваційного повідомлення і заклику, інтерактивні кнопки для вибору напрямів допомоги, та зону соціальних інтеграцій. Структура відповідає принципам UX-дизайну: користувач з першого погляду розуміє мету ресурсу, має зручний доступ до основних дій і додаткової інформації. Такий підхід покращує залучення, орієнтування на сторінці та сприяє прийняттю рішення про підтримку ініціатив [19].

Сторінка «Благодійні збори» є однією з ключових на благодійному ресурсі, адже саме через неї користувачі можуть безпосередньо долучитися до допомоги.

Розробка її прототипу має критичне значення, бо вона повинна забезпечити швидкий доступ до актуальної інформації, викликати довіру й мотивувати до дій.

Для ефективного залучення користувачів важливо, щоб на сторінці була можливість фільтрувати збори за категоріями, тегами та ключовими словами. Це дозволяє людям легко знаходити ті ініціативи, що їм близькі або в яких вони готові взяти участь. Інтеграція пошуку значно покращує навігацію, особливо якщо зборів багато.

Також важливо, щоб інформація про кожен збір відображалася зрозуміло: мета, сума, що залишилася, терміни, короткий опис та кнопка для донату. Візуальна чіткість і прозорість підвищують довіру користувача й імовірність того, що він здійснить пожертву (див. рисунок 2.3).

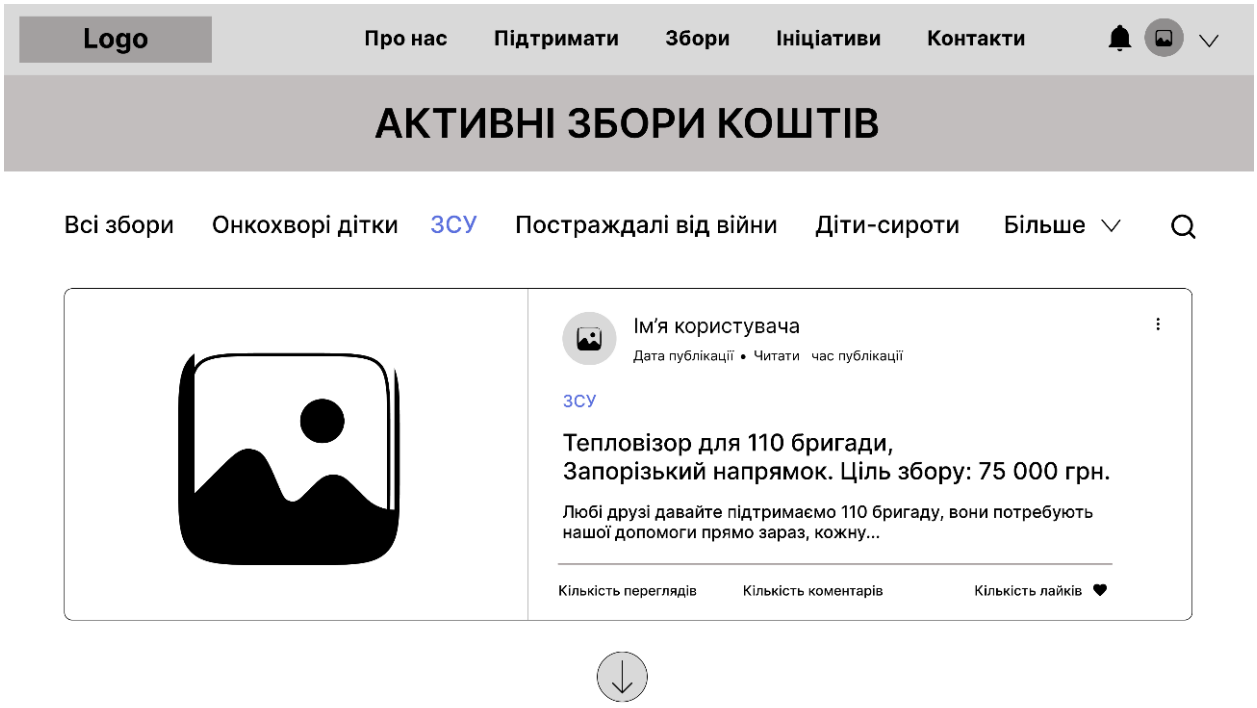


Рисунок 2.3 – Прототип сторінки «Збори коштів»

Прототип реалізований логічно та зручно. Верхня панель містить навігацію, далі – чіткий заголовок «Активні збори коштів», що орієнтує користувача. Збірки згруповані за категоріями (наприклад, «ЗСУ», «Онкохворі дітки», «Діти-сироти»), що полегшує навігацію та швидкий доступ до релевантного контенту. Карточка збору чітка й інформативна: відображено назву, цільову суму, опис, категорію,

активність користувачів (перегляди, лайки, коментарі). Є візуальна структура, яка сприяє легкому сприйняттю та взаємодії. Присутні елементи фільтрації та навігації, які покращують UX.

Отже, чітка структура програмного забезпечення є основою його надійності, масштабованості та зручності в користуванні. Вона дозволяє ефективно організувати функціонал, забезпечити стабільність системи та спростити подальший розвиток.

2.3 Дизайн графічного інтерфейсу та принципи інклюзивності

Дизайн інтерфейсу користувача є важливим етапом у процесі створення сучасного вебзастосунку. Візуальне оформлення не лише формує перше враження про продукт, а й безпосередньо впливає на зручність взаємодії користувача з системою. У межах роботи особливу увагу було приділено структурованості інтерфейсу, візуальній простоті, логічному розміщенню елементів, використанню контрастної кольорової палітри та зрозумілим кнопкам дії. Всі ці чинники мають сприяти швидкому сприйняттю інформації, мінімізації навантаження на користувача та забезпеченню інтуїтивної навігації [20].

Одним із ключових аспектів візуального дизайну стала кольорова палітра інтерфейсу. Основний фон виконаний у світлому, майже білому відтінку, що полегшує сприйняття контенту. Акцентні елементи, як-от кнопки взаємодії, фільтри та посилання, мають насичений синій та оранжеві кольори, які добре контрастують із фоном та привертають увагу користувача. Для тексту використано графітовий колір. Обрана кольорова палітра представлена на рисунку 2.4.

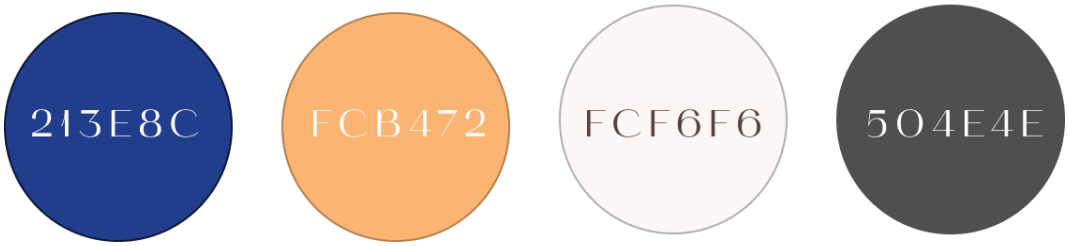


Рисунок 2.4 – Кольорова палітра вебзастосунку

Додатково, важливою складовою візуальної айдентики є логотип, який відіграє роль як навігаційного, так і ідентифікаційного елемента. Логотип вебзастосунку для планування та координації благодійних ініціатив розташовано у верхній частині інтерфейсу, ліворуч від головного меню. Це забезпечує швидке розпізнавання бренду та підтримує послідовність візуального стилю на всіх сторінках вебзастосунку. Логотип розроблено в мінімалістичному стилі з використанням основних кольорів палітри, що підсилює цілісність дизайну. Зображення логотипа наведено на рисунку 2.5.

Рисунок 2.5 – Логотип вебзастосунку

Логотип «Kind Way, зображений на рисунку 2.5, вирізняється витонченим, гармонійним і мінімалістичним оформленням, що візуально передає ідеї доброзичливості, відкритості та сучасності. У логотипі використано український шрифт Mak – це авторський сучасний гарнітур, розроблений з натхненням від кириличної каліграфії та декоративних форм української візуальної культури [21].

Таким чином, використання саме українського шрифту Mak підкреслює національну ідентичність, естетичну унікальність і сучасний підхід до дизайну.

Проте створення сучасного інтерфейсу неможливе без врахування принципів інклюзивного дизайну. Зовнішня естетика та ергономіка – це лише частина завдання. Важливо забезпечити, щоб усі користувачі, незалежно від їхніх фізичних чи когнітивних можливостей, могли повноцінно взаємодіяти із системою. Тут виникає потреба в більш глибокому підході – розробці доступного, інклюзивного інтерфейсу.

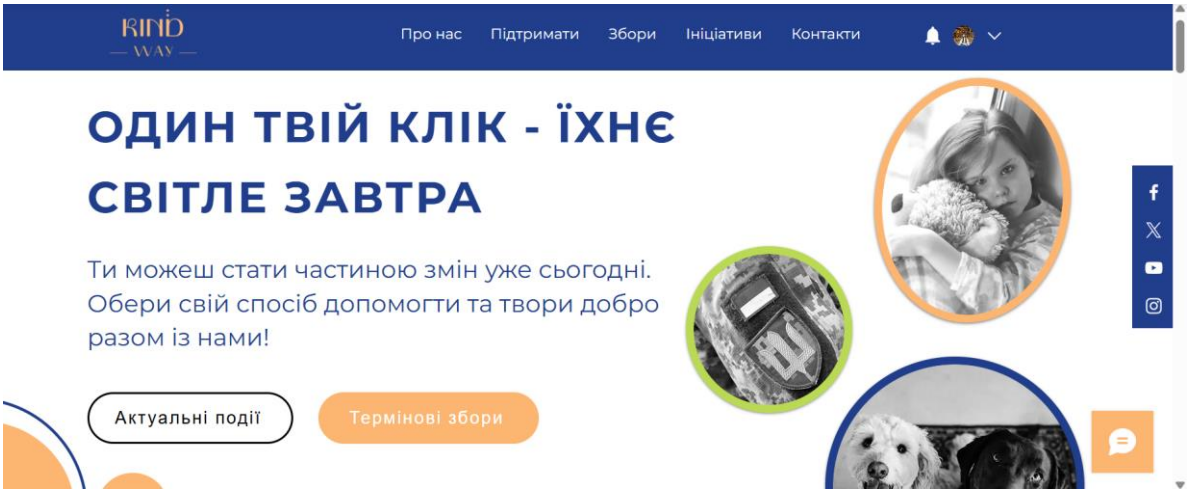
Інклюзивний дизайн та доступність в інтерфейсах користувача є одними з ключових напрямків сучасної розробки цифрових продуктів, які мають на меті забезпечити рівний доступ для всіх користувачів незалежно від їх фізичних, когнітивних чи сенсорних можливостей. Основна проблема полягає в тому, що значна частина цифрових інтерфейсів все ще не відповідає потребам людей з обмеженими можливостями, попри існуючі міжнародні стандарти, такі як Web Content Accessibility Guidelines (WCAG). Це свідчить про недостатнє застосування інклюзивного дизайну на практиці, що негативно впливає на доступність цифрових продуктів для значної частини користувачів [5].

Аналіз літератури з теми інклюзивного дизайну вказує на те, що цей напрямок має глибоке наукове підґрунтя, яке базується на багаторічних дослідженнях у галузі ергономіки, когнітивної психології та соціальної справедливості. Зокрема, Рональд Мейс, засновник концепції універсального дизайну, підкреслював, що продукти та середовища мають бути створені так, щоб максимально враховувати потреби людей з інвалідністю без потреби у спеціалізованих адаптаціях. Його праця «Universal Design: Barrier-Free Environments for Everyone» [5] стала основою для подальших досліджень у цій галузі, демонструючи, що інклюзивний дизайн – це не просто технічна задача, а й соціальна відповідальність.

Отже, врахування принципів інклюзивного дизайну – це не лише етичний, але й стратегічний крок у процесі створення цифрових продуктів. Забезпечення доступності інтерфейсів для людей з різними типами сприйняття, моторики та когнітивних можливостей дозволяє розширити аудиторію користувачів, зменшити цифрову нерівність та підвищити загальний рівень задоволення від користування системою.

У результаті, головна сторінка вебзастосунку для планування та координації благодійних ініціатив демонструє приклад застосування інклюзивного дизайну на практиці – з урахуванням потреб різних категорій користувачів. Вона слугує центральним вузлом навігації, де кожен елемент відповідає критеріям доступності та є інтуїтивно зрозумілим (див. рисунок 2.6). Дизайн цієї головної сторінки було

розроблено у середовищі Figma, що дало змогу ефективно реалізувати принципи інклюзивності, адаптивності та зручності користування ще на етапі проектування інтерфейсу.



Рисунка 2.6 – Дизайн головної сторінки вебзастосунку

Інтерфейс сторінки благодійного сайту Kind Way має інклюзивний, емоційно залучений та зручний дизайн. Контрастна кольорова гама (синій, білий, помаранчевий) забезпечує хорошу видимість і доступність для людей з порушеннями зору. Великі, чіткі заголовки й кнопки полегшують навігацію, а проста мова звернення робить контент зрозумілим для широкої аудиторії. Зображення дітей, військових і тварин створюють емоційний зв’язок і демонструють підтримку різних соціальних груп. Іконки соцмереж та чат підтримки покращують комунікацію. Сайт орієнтований на швидкий доступ до важливої інформації, адаптований до потреб різних користувачів, що робить його прикладом інклюзивного вебдизайну [5].

Отже, дизайн інтерфейсу та принципи інклюзивності є невід’ємною складовою сучасних цифрових продуктів, адже сприяють підвищенню якості та продуктивності взаємодії користувачів із платформою шляхом забезпечення зручної навігації, доступного візуального оформлення, адаптації до потреб різних категорій користувачів, зниження когнітивного навантаження та мінімізації бар’єрів у сприйнятті інформації.

2.4 Розробка методу реєстрації на благодійний захід

Реєстрація на благодійний захід є першим кроком до залучення користувачів до активної участі у заходах, тому метод її реалізації повинен бути максимально доступним, зрозумілим та функціональним. Волонтери (ораганізатори) заходів мають змогу приймати заявки на участь від користувачів у будь-який момент через вебзастосунок. Метод реєстрації забезпечує збір і збереження інформації про учасників, підтвердження їхньої участі, а також інформування адміністрації про нові заявки. Це дозволяє ефективно планувати захід, контролювати кількість учасників та підтримувати зворотній зв’язок із зареєстрованими користувачами.

Реєстрація на захід також важлива для побудови бази даних активних учасників, що може бути корисно для подальших комунікацій, анонсів та статистичного аналізу.

Метод реєстрації на захід включає такий функціонал [4]:

1. Авторизований користувач переходить на сторінку заходу.

2. Користувач заповнює форму реєстрації, у якій вводить ім’я, прізвище та email.

3. Після заповнення форми користувач натискає кнопку «Зареєструватись».

4. Користувач бачить індикатор завантаження (loader).

5. Запускається функція 'handleSubmitRegistration', яка відповідає за обробку даних форми.

6. Функція 'validateRegistrationData' перевіряє коректність введених даних на клієнті.

7. Після валідації викликається 'sendRegistrationRequest', яка відправляє POST-запит на серверний API '/api/registrations'.

8. На сервері функція 'createRegistration' приймає запит, виконує валідацію через 'validateRegistrationSchema' (з бібліотеки Joi).

9. Серверна функція 'checkDuplicateRegistration' перевіряє, чи не існує вже реєстрації користувача на цей захід.

10. Якщо унікальна, серверна функція 'saveRegistrationToDB' зберігає заявку у колекцію MongoDB 'registrations'.

11. Функція 'generateConfirmationToken' створює унікальний токен для підтвердження.

12. Функція 'sendConfirmationEmail' відправляє користувачу підтвердження через email за допомогою 'nodemailer'.

13. Функція 'notifyAdminNewRegistration' надсилає повідомлення адміністратору про нову заявку.

14. Сервер повертає результат у форматі JSON з підтвердженням успішної реєстрації.

15. На клієнті викликається 'handleRegistrationResponse', який знімає індикатор завантаження та виводить повідомлення користувачу.

16. Дані заявки оновлюються в особистому кабінеті через API-метод 'getUserRegistrations'.

Технічна реалізація методу реєстрації на захід передбачає комплексну взаємодію між клієнтською та серверною частинами застосунку, а також базою даних і механізмами безпеки. На клієнтському боці реалізовано React-компонент 'RegistrationForm' у поєднанні з кастомним хуком 'useRegistration', які відповідають за інтерфейс форми та логіку її обробки. Для побудови форми використано бібліотеку 'Formik', а для валідації введених даних – схеми 'Yup', що дозволяють перевіряти коректність полів до надсилання запиту.

На стороні сервера використовується стек 'Node.js' та 'Express'. Створений API-ендпоінт 'POST /api/registrations' обробляє запити на реєстрацію. Дані надсилаються до колекції 'registrations' у базі даних MongoDB, де зберігаються такі поля, як 'userId', 'eventId', 'name', 'surname', 'email', 'status' та 'confirmationToken'. Ці дані використовуються як для підтвердження участі, так і для подальшої аналітики.

Для реалізації підтвердження реєстрації електронною поштою інтегровано бібліотеку 'nodemailer'. Вона використовується у функції 'sendConfirmationEmail', яка відправляє лист із посиланням на підтвердження участі, що містить унікальний токен. Генерація токена здійснюється через функцію 'generateConfirmationToken' з використанням UUID, що гарантує унікальність і безпечність.

Важливим компонентом є валідація даних. На клієнтському рівні валідацію виконує функція 'validateRegistrationData', а на сервері – 'validateRegistrationSchema'. Обидва рівні забезпечують перевірку коректності та повноти введеної інформації. Для обробки помилок використовується механізм 'handleErrors', який формує відповідні HTTP-відповіді (наприклад, 400 при некоректних даних або 409 при конфліктах).

Особливу увагу приділено безпеці. Використовується middleware 'authMiddleware' для перевірки автентичності користувача, а також реалізовано захист від CSRF-атак, що гарантує надійність процесу реєстрації навіть у випадках активної взаємодії з публічними подіями.

Цілісна технічна реалізація забезпечує гнучкий, захищений та масштабований механізм реєстрації, що відповідає потребам сучасного вебзастосунку та гарантує зручність користувачам. Описаний метод забезпечує повний цикл реєстрації з контролем коректності даних, збереженням у базі, підтвердженням для користувача та інформуванням адміністрації [4].

2.5 Розробка методу побудови соціальної активності на платформі в реальному часі

Метод соціальної взаємодії забезпечує безперебійну комунікацію між користувачами, підтримку підписок на ініціативи та оперативне отримання сповіщень. Це підвищує залученість користувачів, сприяє активній участі у житті платформи, а також покращує зворотний зв’язок з адміністрацією. Соціальна взаємодія створює умови для формування спільноти навколо ініціатив, дозволяє користувачам обмінюватися думками, отримувати важливу інформацію у реальному часі та бути в курсі останніх подій.

Опис кроків роботи методу:

1. Користувач заходить на сторінку актуальних зборів і може переглядати благодійні збори інших користувачів (волонтерів).

2. За допомогою функції `subscribeToUser (subscriberId, targetUserId)` виконується підписка на іншого користувача.

3. Запит проходить через API `/api/users/subscribe`, де метод `createUserSubscription` додає запис у колекцію `subscriptions` MongoDB.

4. Після підписки користувач починає отримувати сповіщення у реальному часі через WebSocket, за допомогою події `newSubscriptionNotification`.

5. Користувач може переглядати профілі та заходи підписаних користувачів через API `GET /api/users/: userId/profile` з функцією `getUserProfileData`.

6. Взаємодія відбувається через коментарі під постами, реалізовані компонентом React `CommentsSection`.

7. Для додавання коментаря використовується функція `postComment(postId, userId, commentText)`, яка зберігає коментар у MongoDB через API `/api/comments` і метод `saveComment`.

8. Всі коментарі, повідомлення, підписки та відповіді зберігаються у відповідних колекціях MongoDB: `comments`, `messages`, `subscriptions`.

9. Повідомлення користувачів надходять у панель модерації адміністратора через API `/api/messages` і колекцію `messages`.

10. Адміністратор переглядає повідомлення у реальному часі через WebSocket (`Socket.IO`) і може відповідати через метод `sendAdminResponse (messageId, responseText)`.

11. Відповідь адміністратора миттєво надсилається користувачу через подію WebSocket `adminResponseNotification`.

12. Уся переписка зберігається для подальшого аналізу та звітності.

13. Відстеження активності користувачів (перегляди профілів, лайки, коментарі, підписки) реалізоване функцією `trackUserActivity`, яка записує дані у колекцію `userActivity`.

14. Реалізація WebSocket підключення здійснюється через функцію `initializeSocketConnection(userId)`, що забезпечує доставку сповіщень і повідомлень у реальному часі.

З метою забезпечення повноцінної функціональності системи, було розроблено технічну реалізацію, яка поєднує сучасні вебтехнології для ефективної

взаємодії між користувачами, обробки даних і обміну інформацією в реальному часі.

У клієнтській частині, реалізованій на основі React, використовуються такі основні компоненти: `UserProfile`, `CommentsSection`, `SubscriptionButton`, `NotificationsPanel`, `AdminModerationPanel`, `ChatWindow`. Вони забезпечують функціональність перегляду профілю користувача, додавання коментарів, підписки, перегляду повідомлень і чату з адміністраторами.

Для обробки WebSocket-підключення застосовується спеціальний хук `useSocket`, який відповідає за встановлення з'єднання, отримання та надсилання повідомлень у реальному часі. Серед основних функцій, що використовуються в логіці взаємодії, можна виділити: `subscribeToUser`, `postComment`, `sendMessageToAdmin`, `initializeSocketConnection`, `sendAdminResponse`.

На сервері, що працює на Node.js з Express, реалізовано REST API для обробки запитів. Зокрема, ендпоінт `POST /api/users/subscribe` викликає функцію `createUserSubscription` для створення підписки, а `GET /api/users/:userId/profile` використовується для отримання профілю користувача через `getUserProfileData`. Коментарі додаються за допомогою `POST /api/comments` і функції `saveComment`, а повідомлення адміністраторам надсилаються через `POST /api/messages`, що обробляється функцією `saveUserMessage`. Для модерації передбачено ендпоінт `GET /api/messages/moderation`, який викликає `getMessagesForModeration`, а для відповіді адміністратора – `POST /api/messages/respond` з функцією `saveAdminResponse`.

Обмін повідомленнями у реальному часі реалізовано за допомогою WebSocket-подій через бібліотеку Socket.IO. Основними подіями є: `newSubscriptionNotification`– сигнал про нову підписку, `newCommentNotification` – повідомлення про доданий коментар, `newUserMessage` – надходження нового повідомлення від користувача, та `adminResponseNotification` – відповідь адміністратора у чаті.

Для зберігання даних використовується MongoDB. Основні колекції включають `subscriptions` для обліку підписок, `comments` для збереження

коментарів, `messages` для зберігання діалогів між користувачами та адміністраторами, `userActivity` для відстеження активності користувачів, а також `users` – колекцію з профілями користувачів.

Ініціалізація WebSocket-з'єднання відбувається через функцію `initializeSocketConnection`.

Отже, описаний метод забезпечує комплексний соціальний функціонал із підписками, коментарями, двосторонньою комунікацією між користувачами, реальним часом сповіщень і гнучкою аналітикою.

2.6 Розробка моделі системи

Розробка моделі системи вебзастосунку для управління благодійними ініціативами передбачає створення UML-діаграми діяльності, яка описує послідовність дій користувача під час взаємодії із системою. UML (Unified Modeling Language) – це уніфікована мова моделювання, яка використовується для візуалізації, специфікації, проєктування та документування програмних систем. UML-діаграма активностей є різновидом поведінкових діаграм [22], що дозволяє описати логіку управління потоком дій у системі (див. рисунок 2.7).

Відповідно до розробленої UML-діаграми, користувач розпочинає взаємодію із відкриття вебзастосунку. Система перевіряє, чи має користувач обліковий запис. Якщо обліковий запис наявний, виконується процедура авторизації. У разі відсутності облікового запису користувач проходить процедуру реєстрації, після чого також потрапляє до етапу перевірки введених даних. У разі введення коректних даних система перенаправляє користувача на головну сторінку вебзастосунку. Якщо ж дані виявляються некоректними – користувач повертається на початковий етап для повторного введення або редагування.

Після успішної авторизації користувач отримує доступ до функціональних розділів системи: подання заявки на створення ініціативи, перегляду та підтримки наявних зборів та заходів, а також персонального кабінету.

У секції «Подати заявку» користувач може заповнити форму заявки на проведення заходу, вказавши свої контакні дані, тематику назву та час ініціативи,

після чого вона надсилається організаторам для розгляду. Після підтвердження подія буде опублікована організатором-волонтером у системі та стане доступною для активної участі бажаючих допомогти.

На сторінці благодійних зборів користувач може здійснювати пошук за ключовими словами і тегами та фільтрацію за типом напрямку збору. Після вибору конкретного збору зі списку доступна низка дій: перегляд рейтингу організатора-волонтера стосовно збору; перехід на сторінку організатора-волонтера з можливістю перегляду контактної інформації, опублікованих заходів та здійснення підписки; обрання оцінки від 1 до 5 та можливість написати відгук; підтримка ініціативи: оформлення щомісячної підписки (користувач оформлює підписку на обрану ініціативу, обравши певну суму, яку він готовий виділити на підтримку), зв’язок із волонтером та здійснення одноразового грошового внеску.

Також доступна сторінка благодійних заходів. Тут користувач може обрати подію зі списку, натиснути кнопку «Зареєструватися», заповнити реєстраційну форму та підтвердити участь. Після підтвердження подія додається до календаря користувача.

У межах особистого кабінету користувач може змінити контактну інформацію, переглядати поточні заявки на участь в заходах: минулі та майбутні та переглядати сповіщення (сповіщення про нові підписки, погодження реєстрації на заходи тощо). Також реалізована можливість вийти з облікового запису.

Під час взаємодії з вебзастосунком система обробляє запити користувача, передаючи їх до серверної частини, реалізованої на Node.js. Серверна логіка відповідає за взаємодію з базою даних, забезпечуючи отримання актуальної інформації та швидкий обмін даними в реальному часі.

Користувач має можливість вийти з облікового запису. Це завершує поточний сеанс взаємодії та ініціює очищення тимчасових даних, що забезпечує додатковий рівень безпеки та конфіденційності інформації. Важливо зазначити, що користувач має можливість вийти на будь-якому етапі взаємодії із системою – незалежно від того, на якій сторінці або у якому функціональному блоці він

перебуває. Це підвищує гнучкість користування та забезпечує додаткову зручність і контроль над особистими сесіями.

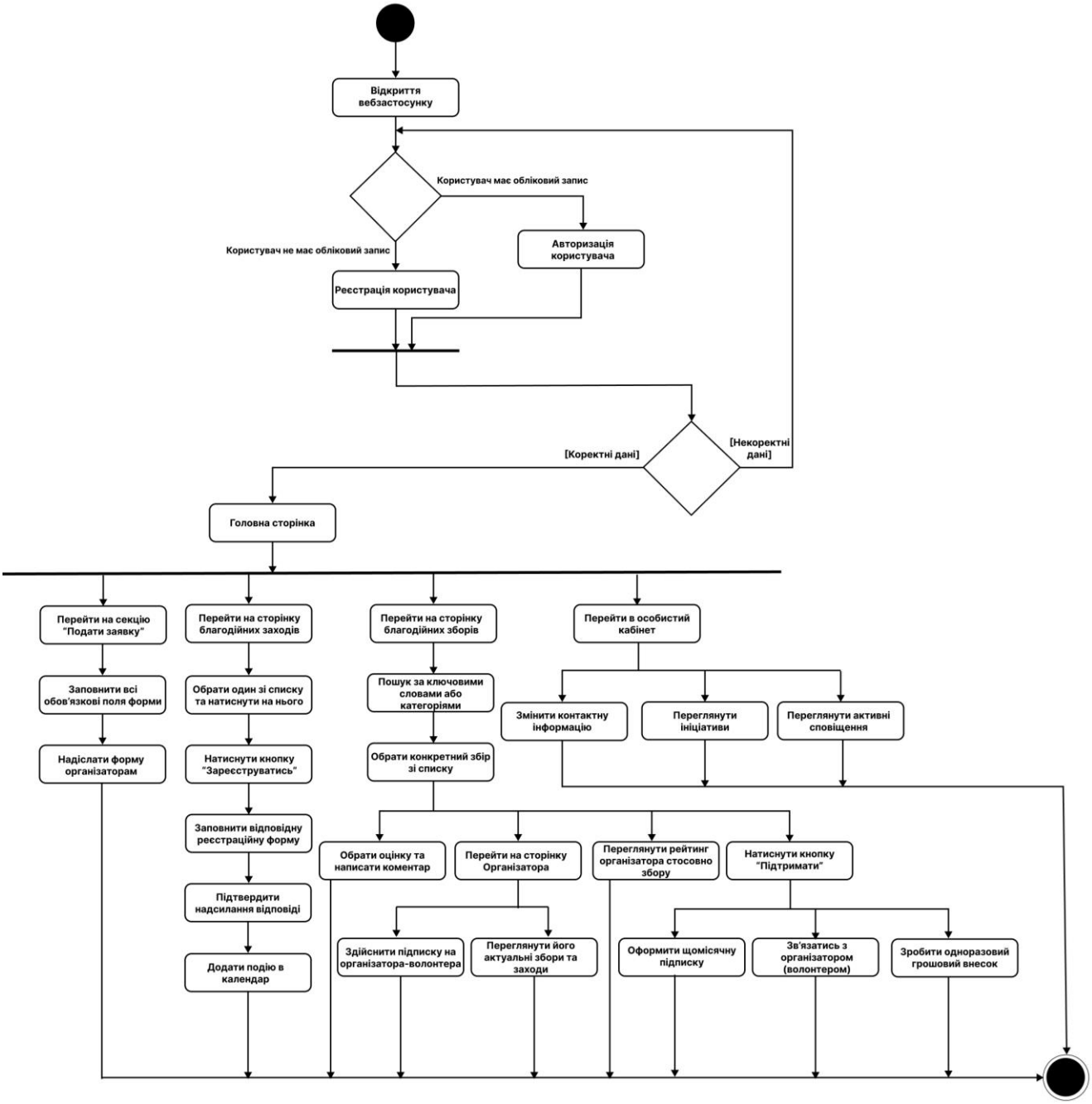


Рисунок 2.7 – Модель у вигляді діаграми діяльності

Отже, розроблена UML-діаграма активностей чітко ілюструє логіку роботи вебзастосунку для управління благодійними ініціативами та послідовність взаємодії користувача із системою. Завдяки використанню цього типу моделювання вдалося структурувати основні сценарії роботи з інтерфейсом – від

реєстрації та авторизації до подання заявок, створення чи підтримки ініціатив, а також керування персональними даними через особистий кабінет.

Модель у вигляді діаграми класів дозволяє формалізувати структуру системи, візуалізувати основні сутності (класи), визначити їх властивості (атрибути), дії (методи) та взаємозв'язки між ними. Завдяки діаграмі класів можна заздалегідь визначити обсяги майбутніх даних, оптимізувати структуру бази даних та мінімізувати помилки (див. рисунок 2.8).

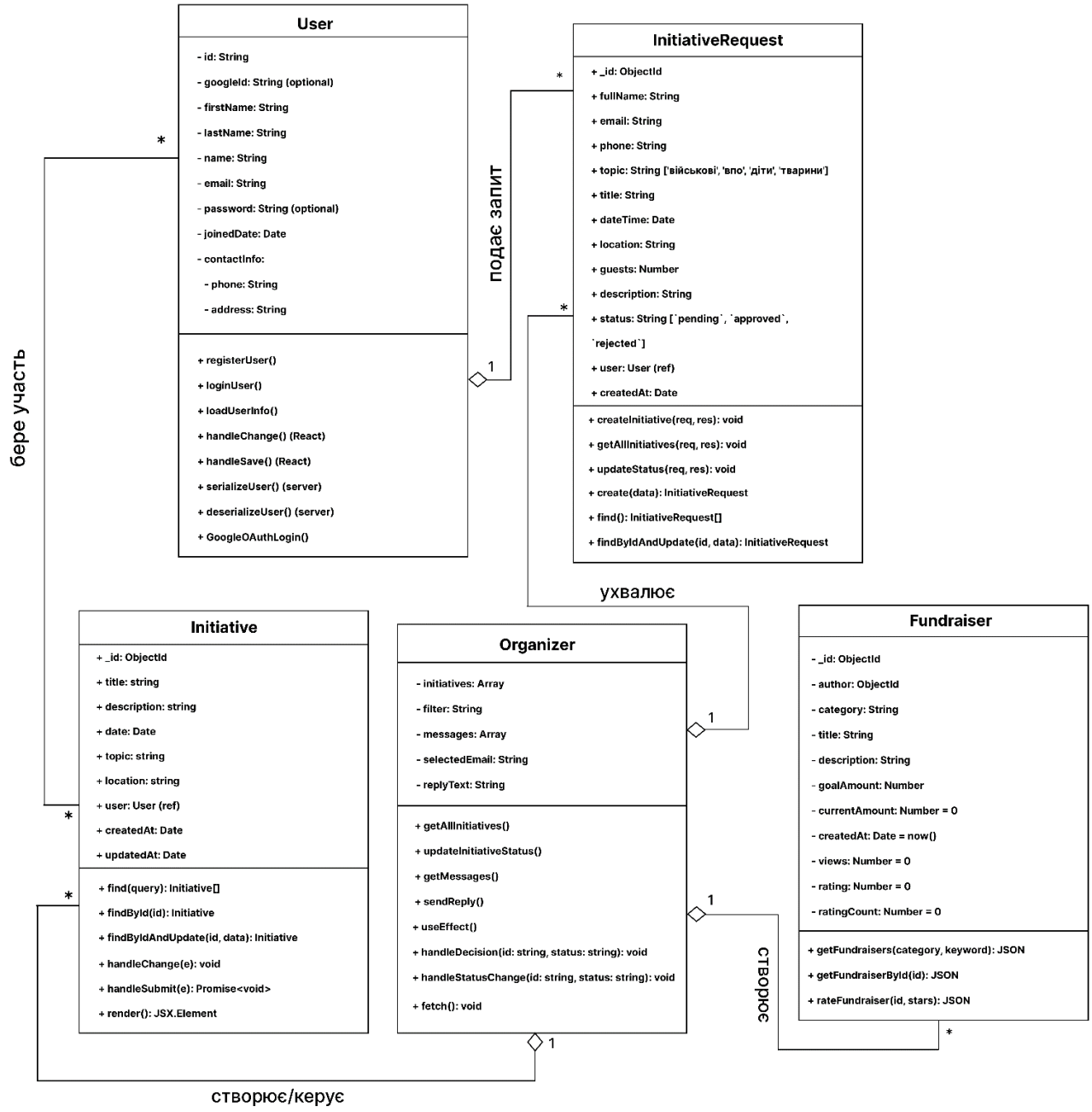


Рисунок 2.8 – Модель у вигляді діаграми класів

Діаграма класів (class diagram) – це тип UML-діаграми, що моделює статичну структуру системи. Вона демонструє класи системи, їх атрибути, методи та зв'язки (асоціації, агрегації, композиції, наслідування) між ними. Завдяки цьому розробник може побачити, які об'єкти існують у системі, що вони зберігають і як взаємодіють між собою. Такі діаграми є основою об'єктно-орієнтованого аналізу та проєктування [22].

На наданій діаграмі змодельовано вебзастосунок для управління благодійними ініціативами. Центральним класом є `User`. Він містить базові атрибути користувача: ім'я (`firstName`, `lastName`, `name`), email, адреса, номер телефону, дата приєднання до системи, а також параметри для входу в систему – `password` (опційний), `googleId` для авторизації через Google. Методи `registerUser()`, `loginUser()` і `loadUserInfo()` відповідають за автентифікацію та завантаження профілю. `handleChange()`, `handleSave()`– це реактивні методи, які використовуються у клієнтській частині, а `serializeUser()` і `deserializeUser()` — на сервері для роботи з сесіями. `GoogleAuthLogin()` дозволяє інтегрувати вхід через Google.

Клас `InitiativeRequest` моделює запит на створення ініціативи. Він містить інформацію про подію – `fullName`, `email`, `phone`, `topic` (допустимі значення задаються масивом), `title`, `dateTime`, `location`, кількість гостей, опис, а також статус заявки (`pending`, `approved`, `rejected`). Методи `createInitiative()`, `getAllInitiatives()`, `updateStatus()`– це логіка, пов'язана зі створенням, переглядом і модерацією запитів. Метод `create(data)` створює об'єкт заявки, `find()` повертає певний запит, `findByIdAndUpdate()`– знаходить та змінює дані заявки.

Клас `Initiative` представляє вже створену ініціативу. Він має назву (`title`), опис, дату проведення, місце, тему, автора (зв'язок з класом `User`) і час створення/оновлення. Методи `find(query)`, `findById(id)`, `findByIdAndUpdate(id, data)` дозволяють здійснювати базові CRUD-операції. Методи `handleChange()`, `handleSubmit()` використовуються у фронтенді для оновлення стану ініціативи, а `render()`– для відображення в інтерфейсі.

Клас `Organizer` призначений для керування та створення ініціатив та благодійних зборів. Він має масив `initiatives`, фільтри, повідомлення та текст відповіді. Метод `getAllInitiatives()` повертає всі ініціативи, `updateInitiativeStatus()` змінює статус, `getMessages()`– отримує повідомлення, а `sendReply()` дозволяє надсилати відповідь. Методи `handleDecision()` і `handleStatusChange()` дають змогу адміністратору приймати рішення щодо ініціатив.

Клас `Fundraiser` відповідає за збір коштів. Він містить інформацію про автора, категорію, назву, опис, цільову суму, поточну зібрану суму, кількість переглядів, рейтинг. Методи `getFundraisers()` та `getFundraiserById()` дозволяють шукати збори, а `rateFundraiser()`– оцінювати їх.

Модель у вигляді діаграми класів відіграє ключову роль у розробці вебзастосунку для управління діяльністю благодійної організації, оскільки дозволяє формалізувати структуру системи, виявити її основні компоненти та взаємозв’язки між ними. Завдяки такому підходу розробники отримують чітке уявлення про логіку предметної області, що знижує ризики помилок під час реалізації функціоналу. Діаграма класів також слугує основою для проектування бази даних, визначення API, а також полегшує взаєморозуміння між членами команди, оскільки візуалізує структуру об'єктів, якими оперує система. Це особливо важливо для складних систем, як-от платформи для координації благодійної діяльності, де багато сутностей мають тісні міжсобою зв’язки.

Клас User представляє користувача системи, що може подавати запити на участь у ініціативах (InitiativeRequest) і брати в них участь, про що свідчить асоціативний зв’язок без ромба між User і Initiative. Зв’язок між User та InitiativeRequest змодельовано як агрегацію з білим ромбом, що означає логічну приналежність: запит існує окремо, але створюється користувачем.

Клас Organizer, у свою чергу, пов’язаний агрегаціями з Initiative, InitiativeRequest і Fundraiser, що логічно відображає, що організатор створює ініціативи, ухвалює запити та запускає збори коштів. Особливо важливо, що Organizer має доступ до масиву ініціатив, повідомлень та рішень, які відображають його функціональні обов’язки.

Таким чином, діаграма класів не лише відображає поточну логіку системи, а й допомагає виявити суперечності або прогалини в моделі, які слід уточнити до початку реалізації. Модель є добре структурованою, охоплює всі ключові аспекти системи – автентифікацію, подання та розгляд заявок, створення ініціатив, керування ними й організацію зборів коштів. Методи чітко визначають функціональність, а зв’язки між класами відображають логічні взаємовідносини між учасниками платформи.

2.7 Розробка алгоритмів роботи системи

Для розробки вебресурсу для координації благодійних ініціатив необхідно визначити основні алгоритми роботи системи, які забезпечать коректну взаємодію користувачів із функціоналом. Загальний алгоритм роботи системи охоплює процеси авторизації, взаємодії з доступними ініціативами та способи допомоги (див. рисунок 2.9).

Першим етапом є введення логіну користувачем. Після цього система перевіряє, чи зареєстрований користувач у базі даних. Якщо користувач ще не зареєстрований, відбувається перенаправлення до процедури реєстрації, де необхідно ввести персональні дані для створення облікового запису.

Процес верифікації користувача реалізовано за допомогою серверної частини на платформі Node.js, яка обробляє запити авторизації, перевіряє відповідність логіну та пароля, а також взаємодіє з базою даних MongoDB, у якій зберігаються облікові записи користувачів. У процесі перевірки використовується хешування паролів (за допомогою бібліотеки «bcrypt»), що забезпечує захист персональних даних.

MongoDB – документоорієнтована нереляційна база даних, яка зберігає дані у форматі BSON (розширення формату JSON) [23]. Такий підхід забезпечує гнучкість структури, що особливо зручно при зберіганні динамічних та різнорідних даних, які можуть змінюватися з часом без необхідності модифікації всієї схеми. MongoDB ідеально підходить для Node.js-додатків, оскільки має офіційний драйвер, який дозволяє швидко реалізувати взаємодію з базою на стороні серверу.

Якщо користувач уже зареєстрований, система запрошує ввести пароль. Після введення паролю Node.js-сервер здійснює перевірку введених даних шляхом порівняння з інформацією у MongoDB [23]. Якщо пароль правильний, здійснюється вхід у особистий кабінет, у якому ініціюється користувацька сесія.

У особистому кабінеті користувач має можливість редагувати свої контактні дані. Якщо така потреба виникає, користувач переходить до відповідного інтерфейсу для зміни інформації. Після редагування система повертає його до основного меню кабінету.

Наступним кроком користувач може перейти до перегляду списку ініціатив, на які користувач був зареєстрований, після реєстрації вони автоматично відображаються в особистому кабінеті. Якщо користувач бажає переглянути список майбутніх заходів на участь – система надає список відповідних запитів, якщо переглянути ті заходи, що вже завершились – система відображає минулі ініціативи. У разі, якщо користувач не обирає жодного з цих варіантів, або завершує перегляд, система повертає його до головного меню.

Останнім етапом взаємодії є вихід із системи. Якщо користувач обирає опцію виходу, сеанс завершується, і алгоритм доходить до завершальної точки – «Кінець».

Взаємодія клієнтської та серверної частин відбувається через API-запити, що гарантує швидкий та надійний обмін даними [24]. Для підвищення ефективності системи реалізовано кешування відповідей, що знижує навантаження на сервер і прискорює завантаження контенту.

Інтерфейс застосунку створено з урахуванням адаптивного дизайну, що забезпечує коректне відображення на різних пристроях – від настільних комп’ютерів до мобільних телефонів. Це дозволяє користувачам зручно користуватись системою незалежно від типу пристрою. Додатково реалізовано можливість персоналізації – збереження історії переглядів, налаштувань профілю та взаємодій з ініціативами, що створює комфортне та індивідуалізоване середовище для кожного користувача.

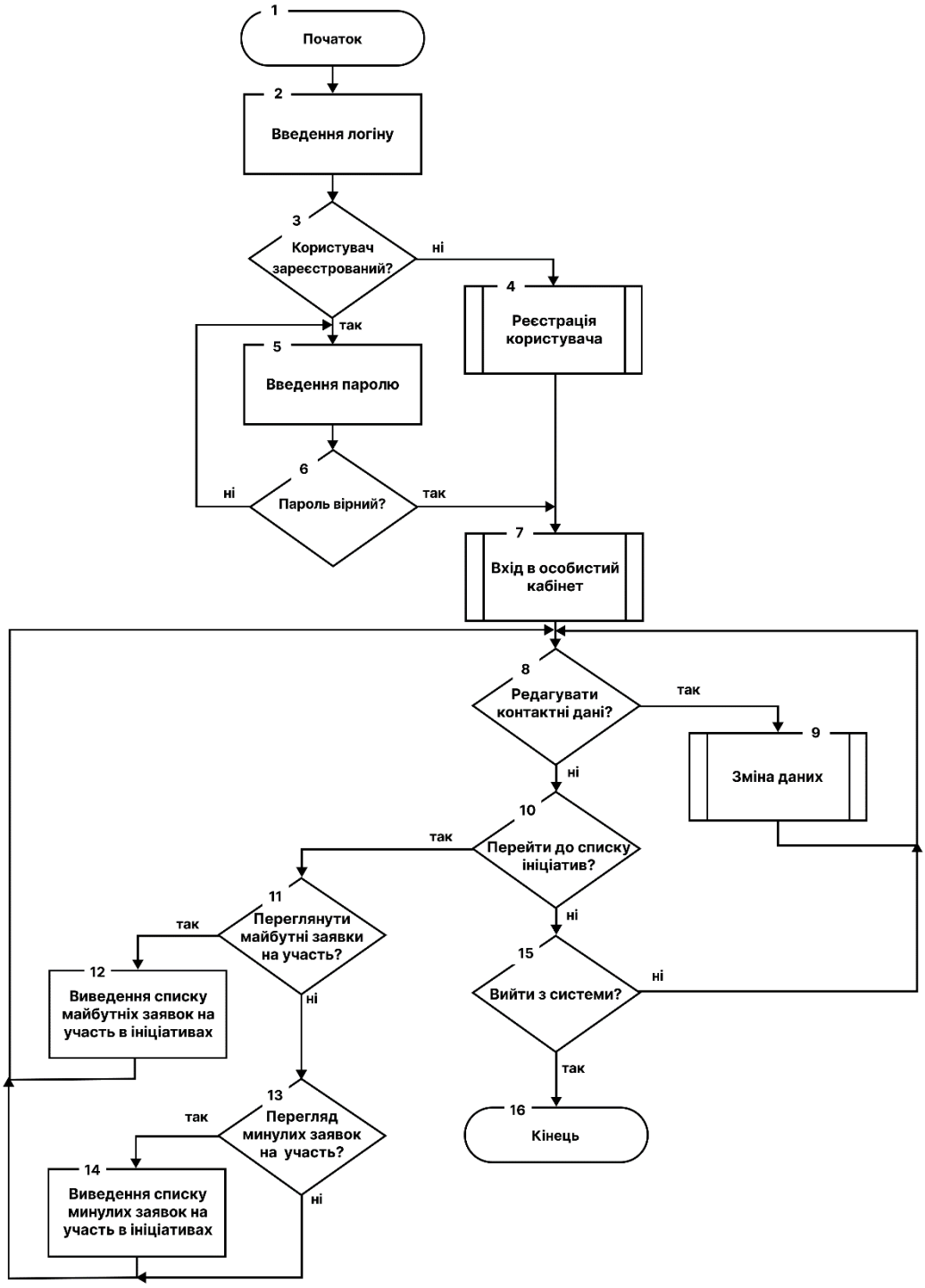


Рисунок 2.9 – Алгоритм авторизації користувача та входу в особистий кабінет

Для забезпечення зручної взаємодії користувача з благодійними зборами на вебресурсі реалізовано чіткий алгоритм, що описує процес пошуку, вибору та підтримки збору, а також можливості взаємодії з іншим користувачами системи, шляхом залишення відгуків, коментарів та здійсненням підписки. На рисунку 2.10 показано відповідну логіку.

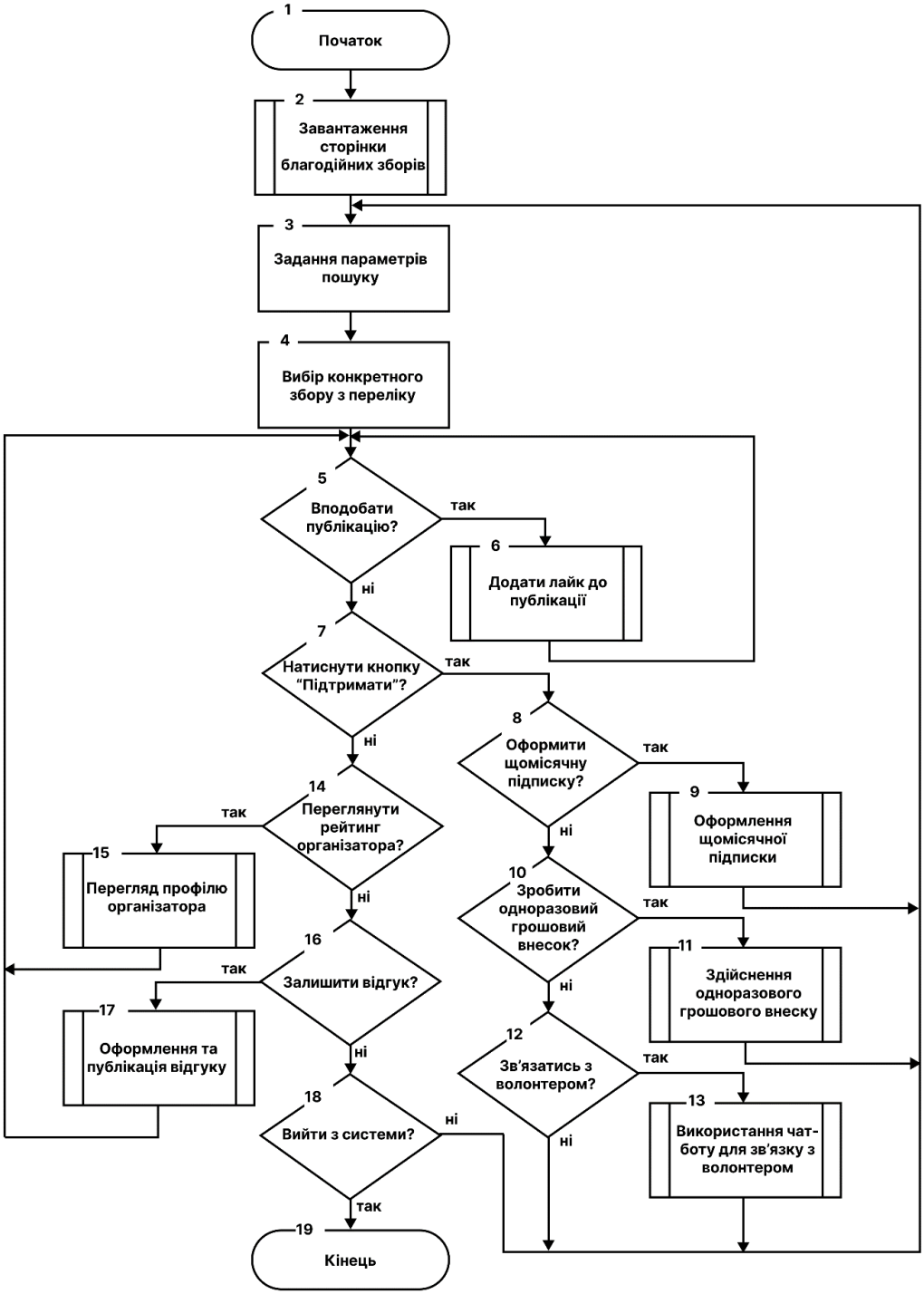


Рисунок 2.10 – Алгоритм користувацької взаємодії з благодійними зборами

Першим кроком є завантаження сторінки зборів, де користувач отримує доступ до переліку актуальних проєктів. Далі він має можливість задати параметри пошуку за ключовими словами або відфільтрувати ініціативи за категоріями. Серед доступних параметрів можуть бути такі типи ініціатив, як «притулки для тварин», «онкохворі діти», «ЗСУ», «постраждалі від війни», «діти-сироти» та «всі збори».

Це дозволяє кожному користувачу швидко знаходити саме ті напрямки, які найбільше відповідають його переконанням або бажанню допомогти.

Після цього здійснюється вибір конкретного збору з оновленого переліку. На цьому етапі користувач приймає рішення, чи бажає вподобати публікацію зі збором, якщо так – виконується функція «лайк». Якщо ж користувач не додає лайк до публікації, система пропонує натиснути кнопку «Підтримати», що відкриває нові варіанти взаємодії.

У разі вибору підтримки, користувачу пропонується кілька шляхів: оформити щомісячну підписку на фінансову підтримку ініціативи, зробити одноразовий грошовий внесок або зв’язатися з волонтером для надання допомоги іншим способом. У випадку оформлення підписки відбувається її налаштування, після чого внески здійснюються автоматично. Одноразовий внесок передбачає швидке та безпосереднє перерахування коштів. Якщо користувач обирає зв’язок із волонтером, він перенаправляється на модальне вікно чат-боту, реалізоване через API [24].

Додатково, користувач має змогу переглянути рейтинг організатора збору, що формується на основі відгуків та оцінок інших користувачів. Якщо збір викликає довіру, користувач може перейти до перегляду профілю організатора і, за потреби, здійснити підписку на нього для відстеження його майбутньої активності. Підписка надсилається через систему і доступна організатору для перегляду у вкладці «Сповіщення». Якщо користувач не бажає переглядати рейтинг організатора-волонтера, він має можливість залишити коментар та оцінку, що проходять валідацію. Після того як перевірка виявиться успішною – відгук публікується та доступний для перегляду іншими користувачами вебсистеми. Коментарі можна уподобувати, відповідати та них, редагувати та видаляти.

Завершальним етапом є рішення про вихід із системи. Якщо користувач обирає вийти, сесія завершується, і він перенаправляється на головну сторінку або форму авторизації.

Таким чином, реалізований алгоритм дозволяє користувачу повноцінно взаємодіяти з ініціативами: підтримувати їх фінансово, брати активну участь, зберігати у вибране та оцінювати організаторів.

Алгоритм реєстрації на благодійний захід описує послідовність дій користувача на платформі, що дає змогу брати участь у благодійних ініціативах, застосовувати фільтри, реєструватися та взаємодіяти з подіями. Цей алгоритм є важливим, оскільки забезпечує логічну, зручну та ефективну модель взаємодії з платформою, підвищує залученість користувачів та сприяє поширенню інформації про ініціативи. Початок процесу визначається завантаженням сторінки ініціатив, що є стартовою точкою взаємодії користувача з системою (див. рисунок 2.11).

Після завантаження користувачу пропонується обрати фільтри, які допоможуть звузити перелік ініціатив. Якщо користувач вирішує скористатися фільтрами, йому пропонується спочатку фільтрувати ініціативи за локацією, що дозволяє відобразити лише події, актуальні для конкретного регіону. У разі підтвердження вибору, застосовується відповідний фільтр. Також користувач має можоивість відфільтрувати за типом допомоги. Якщо користувач обирає цю опцію, він має можливість вибрати конкретний тип благодійної активності, наприклад, притулок для тварин, ЗСУ, діти-сироти тощо. Після цього застосовується відповідний фільтр і відображає перелік заходів обраного типу.

На основі застосованих критеріїв користувач переглядає список ініціатив і обирає одну з них для участі. Після вибору система запитує, чи хоче користувач зареєструватися на подію. У разі згоди він заповнює контактні дані (ім'я, прізвище та електронну адресу) та завершує реєстрацію, підтверджуючи дію кнопкою.

Після реєстрації користувачеві пропонується додати подію у свій календар. Якщо він погоджується, система додає подію у вибраний календар, що дозволяє йому краще організувати час і не пропустити подію, також ініціатива додатково буде відображатись в особистому кабінеті користувача.

Далі користувачу надається можливість поширити інформацію про ініціативу в соціальних мережах. Якщо він погоджується, обирається відповідна мережа користувачем, і система автоматично публікує подію.

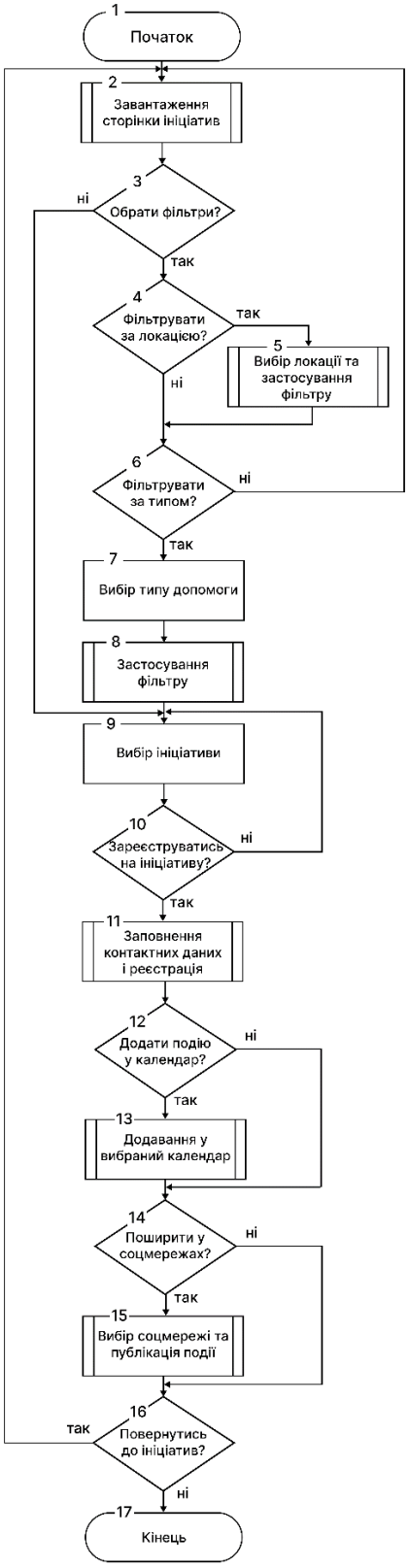


Рисунок 2.11 – Алгоритм реєстрації на благодійний захід

На завершальному етапі система запитує, чи бажає користувач повернутися до списку ініціатив для перегляду інших. Якщо так – процес повторюється знову,

якщо ні – алгоритм завершується. Це дозволяє створити зручну, повторювану і повністю контрольовану логіку взаємодії з ініціативами на платформі.

У результаті розробки вебресурсу було створено три основні алгоритми: авторизації, взаємодії зі зборами та реєстрації на благодійний захід. Вони забезпечують логіку роботи системи, спрощують навігацію та автоматизують ключові процеси. Реалізація таких функцій підвищує рівень якості і продуктивності платформи, завдяки зручній навігації, персоналізації та безпечній авторизації, а також шляхом автоматизації ключових дій та зменшення навантаження на сервер завдяки кешуванню.

2.5 Висновки

У межах розробки благодійного вебзастосунку було створено комплексну систему, яка охоплює основні аспекти взаємодії між ключовими учасниками платформи. На основі діаграми варіантів використання реалізовано чітку функціональну структуру.

Особливу увагу приділено реалізації механізму реєстрації на заходи, який поєднує клієнтську й серверну валідацію, взаємодію з базою даних MongoDB та зворотній зв’язок через email-сповіщення. Окрім цього, впроваджено інструменти для соціальної взаємодії в реальному часі, які активізують участь користувачів у житті платформи – від підписок до коментарів і спілкування з адміністрацією через WebSocket.

Розроблено діаграму діяльності та класів, алгоритми авторизації, взаємодії з благодійними зборами та реєстрації на благодійний захід забезпечують логічну послідовність дій у межах системи, що є основою для подальшої інтеграції та масштабування функціоналу. Прототип головної сторінки, дизайн інтерфейсу, обрана кольорова гама та розробка логотипу відображають цілісність візуальної і технічної концепції вебресурсу.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

У процесі розробки програмного забезпечення для системи планування та координації благодійних ініціатив особливу увагу було приділено вибору технологій, які дозволяють створити зручний, надійний і масштабований застосунок. Головним критерієм під час вибору засобів реалізації була можливість швидкої розробки, простоти у підтримці коду, а також зручності користування для людей, що будуть планувати заходи, координувати волонтерів і відслідковувати прогрес ініціатив.

Фронтенд частина реалізована за допомогою бібліотеки React. Вона надає всі необхідні можливості для створення динамічного інтерфейсу, який швидко реагує на дії користувача. React дозволяє працювати з окремими компонентами, тому інтерфейс легко розділити на логічні частини, такі як події, волонтери, список потреб тощо. Крім того, React має велику спільноту та багато готових рішень, що значно спрощує процес розробки [25].

Для маршрутизації у додатку, тобто перемикання між сторінками без перезавантаження, використовується бібліотека React Router. Вона є простою у використанні й дає змогу швидко реалізувати переходи між різними розділами, наприклад, від головної сторінки до створення події або перегляду волонтерів [25].

На бекенді використовується Node.js у поєднанні з Express.js. Це рішення є логічним продовженням вибору JavaScript як основної мови. Express дозволяє створити зручне API для взаємодії клієнта з сервером. Такий підхід дає змогу легко обробляти запити від користувача – створення подій, реєстрація, збереження інформації про волонтерів тощо [26].

Для збереження даних використовується база даних MongoDB, яка добре підходить для зберігання структурованої, але гнучкої інформації, що може змінюватися з часом. Наприклад, дані про події або учасників можуть мати різну кількість полів, і документна структура MongoDB дозволяє з цим легко працювати.

Для зручності роботи з базою використовується бібліотека Mongoose, яка допомагає описувати схеми та валідувати дані.

Щоб забезпечити автентифікацію користувачів і захист персональних даних, застосовуються JSON Web Tokens (JWT) – токен-базований механізм входу, який зручно реалізовується на Node.js. Також використовується bcrypt для безпечного зберігання паролів [27].

Також було використано Axios для того, щоб реалізувати обмін даними між клієнтською частиною вебзастосунку (на React) і сервером. Axios – це бібліотека для виконання HTTP-запитів, яка дозволяє зручно надсилати запити на сервер, отримувати відповіді, обробляти помилки, а також додавати заголовки, такі як токени авторизації [28].

Оскільки платформа передбачає можливість збору благодійних внесків, для інтеграції платіжних операцій було обрано Stripe. Stripe забезпечує безпечний і гнучкий спосіб обробки платежів, підтримуючи різні платіжні методи та дозволяючи легко інтегрувати платежі у вебзастосунок за допомогою офіційних бібліотек для Node.js і клієнтської частини. Це рішення відповідає вимогам безпеки (наприклад, стандартам PCI DSS) і дозволяє швидко масштабувати систему [29].

Обраний набір технологій дозволяє розробити стабільну, зручну у використанні систему, яка може масштабуватися в майбутньому. Він повністю базується на JavaScript, що спрощує процес розробки та підтримки, оскільки і клієнтська, і серверна частини використовують одну мову.

Саме тому було обрано мову програмування JavaScript. JavaScript – це динамічна, інтерпретована мова програмування, яка є стандартом для створення інтерактивного вмісту у веббраузерах. Вона підтримується всіма сучасними браузерами і дозволяє реалізовувати як фронтенд, так і бекенд за допомогою таких фреймворків і платформ, як React, Node.js та Express [30]. Одномовна архітектура значно спрощує комунікацію в команді, прискорює розробку та зменшує витрати.

Окрім JavaScript, під час вибору мови програмування для вебзастосунку були розглянуті такі популярні альтернативи, як PHP, C# та Python. Кожна з них має свої особливості, переваги й обмеження.

PHP (Hypertext Preprocessor) – це серверна мова програмування, спеціально створена для розробки вебзастосунків. Вона добре інтегрується з HTML і є дуже популярною завдяки таким платформам, як WordPress. PHP проста у вивченні, має велику спільноту, але поступається сучасним рішенням за продуктивністю, масштабованістю та структурованістю коду [31].

C# (C-Sharp) – об'єктно-орієнтована мова програмування, розроблена компанією Microsoft. Вона використовується разом з платформою .NET для створення надійних вебзастосунків. C# добре підходить для корпоративного програмного забезпечення, має розвинену типізацію, високу продуктивність та інструменти для безпеки. Однак її використання часто вимагає Windows-середовища, і поріг входу для новачків вищий [32].

Python – універсальна мова, яка вирізняється простим синтаксисом і читабельністю. Завдяки фреймворкам Django та Flask Python активно використовується у веброзробці, а також у науці, автоматизації та штучному інтелекті. Він має менше інструментів для фронтенду, тому зазвичай поєднується з JavaScript, що ускладнює цілісність стека [33].

Хоч ці мови й мають свої переваги, саме JavaScript виявився найкращим вибором через свою універсальність, підтримку повного стека (frontend і backend), широке ком’юніті та активний розвиток. Щоб підтвердити доцільність вибору JavaScript як основної мови для розробки вебзастосунку, створено порівняльну таблицю функціональності з ключовими аспектами, що реально впливають на ефективність розробки (див. таблицю 3.1).

Таблиця 3.1 – Порівняння функціональних характеристик мов програмування

Критерії функціональних вимог	JavaScript	PHP	C#	Python
Підтримка повного стека (frontend + backend)	1	0	0	0
Гнучкість у створенні SPA (single-page applications)	1	0	0	0
Підтримка розробки мікросервісів	1	0	1	1

Продовження таблиці 3.1

Критерії функціональних вимог	JavaScript	PHP	C#	Python
Наявність екосистеми для real-time застосунків (Socket)	1	0	1	1
Активність спільноти, оновлення, швидкий розвиток	1	1	1	1
Підтримка хостинг-платформ та серверів «із коробки»	1	1	0	1
Масштабованість застосунку без зміни мови	1	0	1	1
Легкий запуск серверної частини без складної конфігурації	1	1	0	1
Кросплатформеність (desktop/web/mobile)	1	0	1	1
Загальний коефіцієнт	10	3	5	7

Мова програмування JavaScript отримала найбільшу кількість балів завдяки своїй універсальності, активній екосистемі, та можливості писати як клієнтську, так і серверну частини однією мовою. Перевага мови особливо відчутна в проєктах, орієнтованих на швидкий розвиток, взаємодію в реальному часі та сучасний користувацький інтерфейс.

Після обгрунтування вибору мови програмування важливим етапом є вибір середовища розробки, яке забезпечить зручну та продуктивну роботу з обраною мовою. Для роботи з JavaScript було обрано Visual Studio Code (VS Code) – популярний безкоштовний редактор коду, розроблений компанією Microsoft. VS Code підтримує широкий спектр мов програмування, має потужну систему розширень, інтегрований термінал, налагоджувач, автодоповнення коду, та інші корисні функції [34], які роблять його одним з найкращих інструментів для розробників.

Було також розглянуто три схожі середовища, які теж широко використовуються для розробки на JavaScript:

WebStorm – це професійне комерційне середовище розробки від компанії JetBrains, яке спеціалізується на розробці веб-додатків, зокрема на JavaScript, TypeScript та сучасних фронтенд-фреймворках (React, Angular, Vue). WebStorm надає розширені інструменти для аналізу коду, інтелектуальне автодоповнення, рефакторинг, інтегрований налагоджувач, підтримку тестування та систему контролю версій, що робить його ідеальним вибором для складних проєктів [35].

Brackets – безкоштовний редактор коду з відкритим вихідним кодом, розроблений компанією Adobe, орієнтований на веб-розробників. Brackets має унікальну функцію «живого попереднього перегляду», що дозволяє миттєво бачити зміни у браузері під час редагування HTML, CSS або JavaScript. Також підтримує численні розширення, які додають нові можливості, і має простий, зрозумілий інтерфейс, зручний для початківців [36].

Sublime Text – це швидкий, легкий та гнучкий редактор коду, відомий своїм мінімалістичним інтерфейсом та високою продуктивністю. Хоча Sublime Text є платним, він має велику базу користувачів завдяки підтримці великої кількості плагінів, які додають функції автодоповнення, підтримку багатьох мов програмування, систему контролю версій і інші інструменти. Цей редактор підходить для тих, хто цінує простоту та швидкість роботи [37].

Точний аналіз і порівняння функціональних можливостей цих середовищ програмування наведено таблицю 3.2.

Таблиця 3.2 – Таблиця порівняння функціональних характеристик середовищ

Функціональна характеристика	VS Code	WebStorm	Brackets	Sublime Text
Вбудований відладчик	1	1	0	0
Підтримка плагінів/розширень	1	1	1	1
Підтримка фреймворків (React, Angular тощо)	1	1	0	0
Підтримка Live Share (спільна розробка)	1	0	0	0

Продовження таблиці 3.2

Функціональна характеристика	VS Code	WebStorm	Brackets	Sublime Text
Інтеграція з базами даних	1	1	0	0
Кросплатформеність	1	1	1	1
Підтримка препроцесорів (LESS, SCSS)	0	0	1	0
Підтримка Jupyter Notebook/інтерактивного коду	1	0	0	0
Теми та адаптивний інтерфейс	1	1	1	1
Сумарний коефіцієнт	8	6	4	3

Як видно з таблиці**,** Visual Studio Code не лише має найширший набір базових функцій, а й підтримує інноваційні можливості, які покращують співпрацю та інтеграцію в сучасних розробницьких процесах.

Отже, вибір цих засобів реалізації базувався на їхній здатності відповідати ключовим вимогам системи, таким як висока продуктивність та якість, зручність у розробці, стабільність роботи та можливість масштабування. Обраний технологічний стек гарантує не лише швидкий і комфортний процес розробки, а й забезпечує надійну інтеграцію окремих модулів, що сприяє легкому підтриманню та подальшому розвитку вебсистеми.

3.2 Розробка модуля одноразової оплати та оформлення щомісячної підписки для підтримки благодійних зборів

Модуль оплати є важливим елементом системи благодійної ініціативи. Його основне завдання – надати користувачам можливість фінансово підтримати проєкти через зручні та безпечні механізми, включно з одноразовими платежами та регулярними щомісячними підписками. Це дозволяє створити стабільний потік донатів та залучити людей до постійної участі в підтримці благодійних проєктів.

Модуль реалізовано з використанням JavaScript як на стороні клієнта (React), так і на стороні сервера (Node.js з Express). Для інтеграції платіжної системи обрано Stripe, який забезпечує високий рівень безпеки, зручний API та підтримку як одноразових, так і регулярних платежів. На фронтенді використовуються бібліотеки @stripe/react-stripe-js та @stripe/stripe-js, які дозволяють інтегрувати платіжні форми та обробляти відповіді від сервера безпосередньо в браузері [29].

Інтерфейс розробленого модуля одноразової оплати та оформлення щомісячної підписки для підтримки благодійних ініціатив зображено на рисунку 3.2.

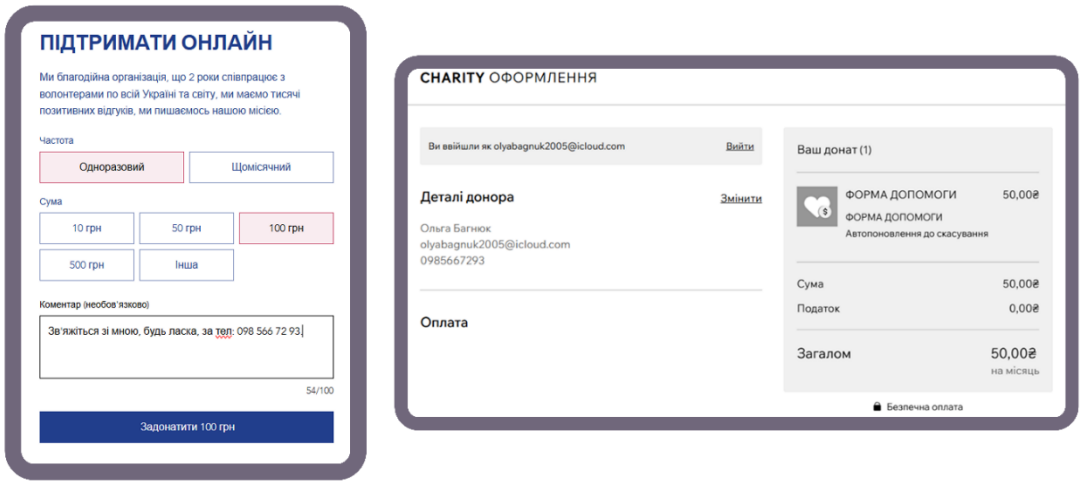


Рисунок 3.2 – Інтерфейс розробленого модуля одноразової оплати та оформлення щомісячної підписки для підтримки благодійних ініціатив

На серверній частині використовується бібліотека `stripe`, яка імпортується у файлі `/server/config/stripe.js`. Тут відбувається ініціалізація Stripe-об'єкта за допомогою секретного ключа, що зберігається в змінній оточення `STRIPE_SECRET_KEY`. Це забезпечує безпечне підключення до сервісу Stripe API, дозволяючи створювати транзакції та обробляти платежі.

Файл `/server/controllers/donationController.js` містить основну серверну логіку обробки платежу. Експортується асинхронна функція `createPaymentIntent`, яка приймає HTTP-запит від клієнта з тілом, що містить суму пожертви (`amount`) і позначку регулярності платежу (`recurring`). Ця функція створює платіжний

намір, викликаючи метод `stripe.paymentIntents.create()`, і вказує суму в копійках (тобто множить на 100), валюту (`uah`) та метадані. Алгоритм обробки платежу через Stripe з використанням клієнт-серверної архітектури зображено на рисунку 3.3.

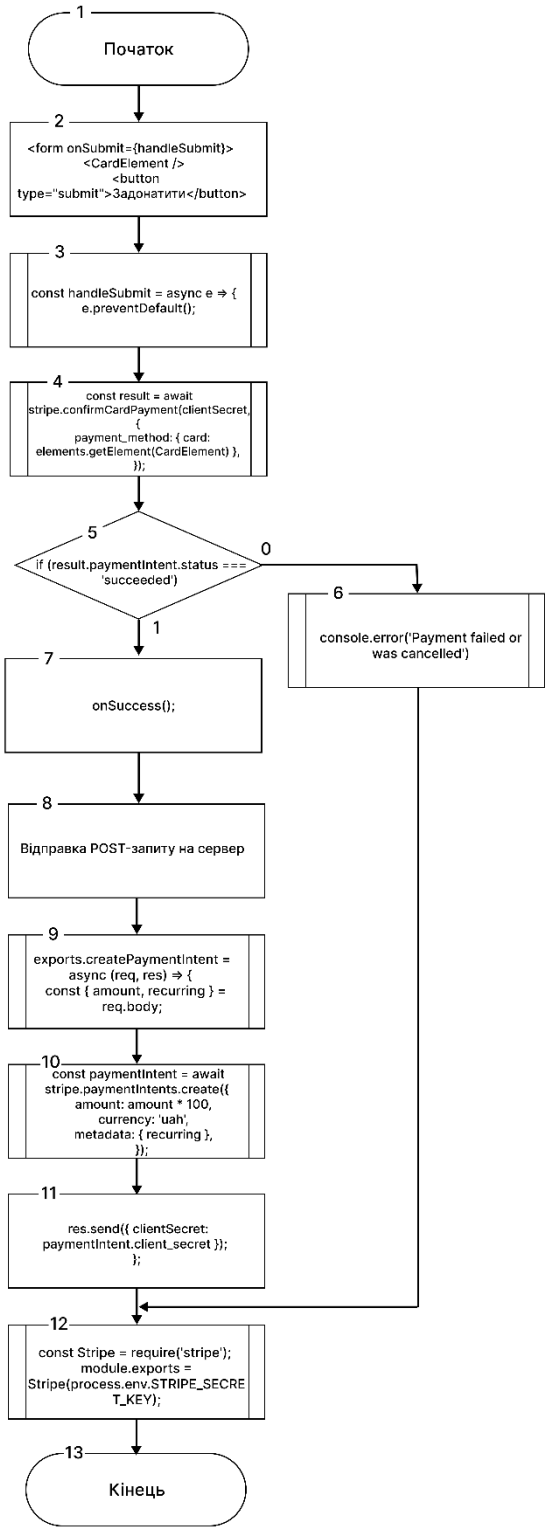


Рисунок 3.3 – Алгоритм обробки платежу через Stripe з використанням клієнт-серверної архітектури

Згідно алгоритму 3.3, на стороні клієнта, у файлі `/client/src/components/Payment/StripeCheckoutForm.jsx`, реалізовано React-компонент `StripeCheckoutForm`, що відповідає за введення платіжних реквізитів користувачем та ініціацію оплати. Для інтеграції з Stripe використовується бібліотека `@stripe/react-stripe-js`, з якої імпортуються хуки `useStripe`, `useElements` та компонент `CardElement`. `CardElement` рендерить захищене поле введення картки, яке обробляється безпечно через Stripe [29].

При натисканні на кнопку «Задонатити» викликається функція `handleSubmit`, що запобігає стандартній подачі форми та виконує метод `stripe.confirmCardPayment`, передаючи `clientSecret` та інформацію про платіжний метод – у цьому випадку, картку, введену користувачем. Умова перевіряє статус платіжного наміру: якщо статус `succeeded`, викликається колбек `onSuccess`, який сигналізує користувачу про успішну оплату.

У веб-застосунку доступно два основні маршрути: для роботи з одноразовими платежами, та для роботи з підписками. Алгоритм для обробки одноразового платежу зображений на рисунку 3.4.

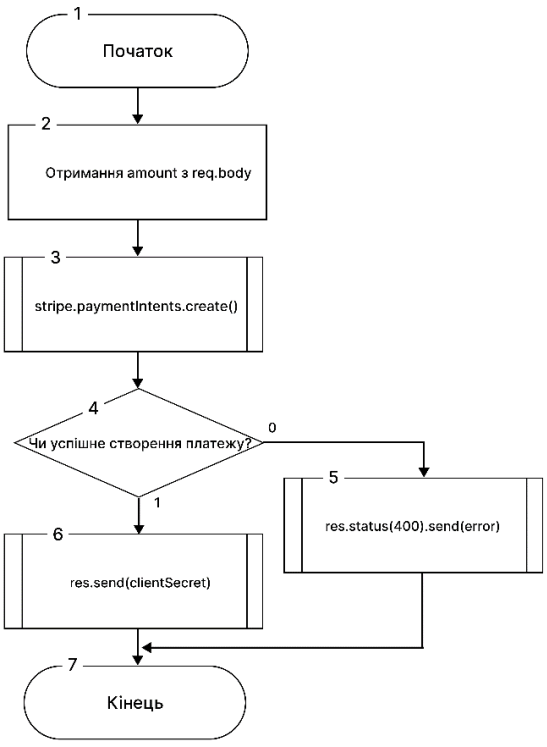


Рисунок 3.4 – Алгоритм обробки одноразового платежу Stripe через маршрут /create-payment-intent

Перший маршрут відповідає за створення одноразового платежу. Користувач надсилає на сервер суму транзакції у тілі запиту (`req.body.amount`). Сервер викликає метод, вказуючи суму, валюту та тип методу оплати – картка. У відповідь клієнт отримує `clientSecret`, який необхідний для подальшого підтвердження платежу на фронтенді. Якщо виникає помилка, сервер повертає повідомлення з кодом 400 і текстом помилки.

Другий маршрут `/create-subscription` реалізує логіку створення підписки (рекурентного платежу). Клієнт надсилає електронну адресу користувача, ID платіжного методу та ID суми (`priceId`, який задається в Stripe Dashboard). Спочатку створюється об'єкт клієнта через `stripe.customers.create()`, де зберігається інформація про платіжну карту. Далі створюється підписка через `stripe.subscriptions.create()`, яка прив’язує клієнта до вибраного тарифу. Параметр `expand: ['latest_invoice.payment_intent']` дозволяє одразу отримати дані про перший платіж. У відповідь сервер надсилає `subscriptionId` та `clientSecret` для підтвердження транзакції (див. рисунок 3.5).

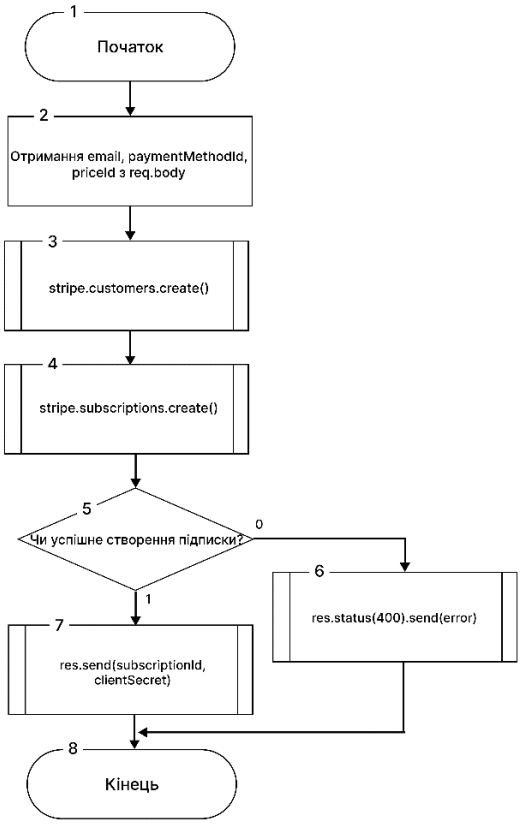


Рисунок 3.5 – Серверний алгоритм обробки підписки з використанням Stripe

У клієнтській частині застосунку було реалізовано інтерактивну форму збору платіжних даних за допомогою бібліотеки @stripe/react-stripe-js. Вона дозволяє безпечно взаємодіяти з платіжною системою Stripe без зберігання даних картки на сервері. Форма створена як React-компонент PaymentForm і є частиною інтерфейсу користувача (фронтенду).

Основною метою цього компонента є створення зручного інтерфейсу, через який користувач може здійснити одноразову фінансову підтримку. Інтерфейс містить інтегрований елемент CardElement, який рендерить поля для введення даних платіжної картки відповідно до стандартів безпеки PCI DSS. Після введення даних та підтвердження користувачем, форма надсилає запит до бекенду для створення платіжного наміру (PaymentIntent) на основі введеної суми.

Після отримання clientSecret – спеціального токена безпеки – відбувається підтвердження транзакції через метод stripe.confirmCardPayment(). У разі успішної оплати користувач отримує повідомлення про успіх, у протилежному випадку обробляється помилка. Таким чином, дана форма реалізує повний цикл взаємодії з платіжною системою з боку клієнта, зберігаючи при цьому високий рівень безпеки й зручності.

Також було розроблено компонент SubscriptionForm, що реалізує клієнтську частину оформлення підписки у вебзастосунку. Форма дозволяє користувачу ввести адресу електронної пошти та дані банківської картки, після чого ініціюється створення методу оплати та надсилається запит на створення підписки на сервер. Отримавши clientSecret, виконується підтвердження транзакції за допомогою Stripe API. У випадку успіху користувачу виводиться повідомлення про успішну підписку.

Однією з важливих функцій для зручності користувачів, є можливість скасування активної підписки у будь-який момент. Користувач має бачити, що може самостійно припинити регулярні платежі без залучення адміністратора чи звернення до служби підтримки. На боці сервера ця функціональність реалізується через окремий маршрут /cancel-subscription, який приймає subscriptionId – унікальний ідентифікатор підписки, що генерується Stripe при її створенні. Запит

викликає метод stripe.subscriptions.del(subscriptionId), який скасовує активну підписку. У разі успішної операції сервер повертає підтвердження, і фронтенд може оновити інтерфейс, наприклад, прибрати статус «активна підписка» або вивести повідомлення «підписка скасована» (див. рисунок 3.6).

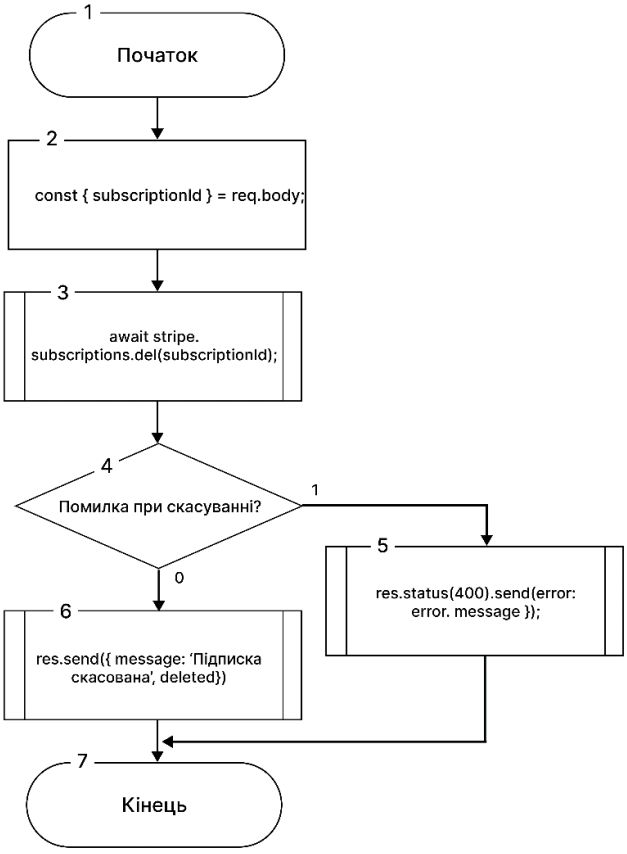


Рисунок 3.6 – Алгоритм скасування підписки користувачем на сервері

Ще одним важливим елементом функціоналу платіжного модуля є збереження інформації про транзакції у базі даних. Збереження виконується за допомогою MongoDB. Створюється окрема колекція, яка містить документи з інформацією про кожну транзакцію: email користувача, суму платежу, тип (одноразовий платіж або підписка), статус транзакції, а також дату. Ці дані записуються після успішного завершення платежу.

Отже, використання Stripe спрощує реалізацію модуля оплати завдяки наявності готових компонентів, зручному API та підтримці валют. У модулі оплати реалізовано повну безпеку платіжних даних користувача. Усі важливі дані картки

не обробляються на сервері – вони передаються напряму на Stripe через зашифровану форму, створену за допомогою Stripe Elements. Це відповідає стандарту PCI DSS і виключає зберігання карткових даних у базі.

Сервер взаємодіє лише з ідентифікатором платіжного методу (paymentMethod.id), який надходить після успішного створення платіжного методу на клієнті. Крім того, для перевірки запитів від Stripe використовується підпис webhook-події, що виключає підроблені запити.

3.3 Розробка модуля авторизації та реєстрації користувачів з валідацією даних

Розробка модуля авторизації та реєстрації є важливим етапом побудови вебзастосунку, що передбачає захищений доступ користувачів до персоналізованих функцій системи. У даному проєкті для реалізації даного модуля використано стек JavaScript-технологій: Node.js + Express на серверній частині, MongoDB для зберігання даних користувачів, bcrypt для безпечного хешування паролів. Крім того, для валідації даних застосовується бібліотека express-validator, що дозволяє запобігти введенню некоректних або потенційно небезпечних даних.

Інтерфейс розробленого модуля авторизації та реєстрації користувачів з валідацією даних зображено на рисунку 3.7.

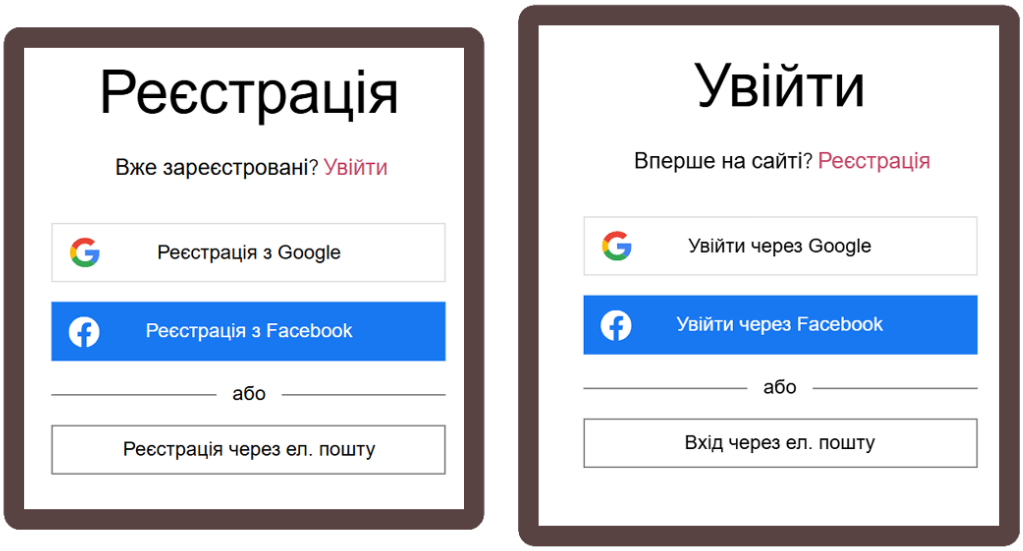


Рисунок 3.7 – Інтерфейс розробленого модуля авторизації та реєстрації користувачів

Також для організації процесу аутентифікації та авторизації використовується Passport – потужний middleware для Node.js, який забезпечує підтримку різноманітних стратегій входу, зокрема локальної авторизації за допомогою пароля [38], а також OAuth 2.0 аутентифікації через сторонні сервіси, такі як Google чи Facebook [39]. Це дозволяє забезпечити гнучкий та безпечний механізм входу користувачів у систему.

На серверній стороні використовується MongoDB і бібліотека Mongoose для збереження даних. Для цього створюється модель користувача (див. рисунок 3.8).

```
const mongoose = require('mongoose');

const UserSchema = new mongoose.Schema({
  name: { type: String },
  email: { type: String, required: true, unique: true, lowercase: true },

  // Локальна авторизація
  passwordHash: { type: String },

  // OAuth дані
  googleId: { type: String, unique: true, sparse: true },
  facebookId: { type: String, unique: true, sparse: true },

  joinedDate: { type: Date, default: Date.now },
});

module.exports = mongoose.model('User', UserSchema);
```

Рисунок 3.8 – Модель користувача (User model) для MongoDB

На даному етапі було розглянуто взаємодію користувача (фронтенд) із інтерфейсом – збір даних, їхню перевірку на стороні клієнта та відправку на сервер для подальшої обробки. Спочатку визначено дві допоміжні функції для валідації введених користувачем даних: isValidEmail перевіряє коректність формату електронної пошти за допомогою регулярного виразу, а isStrongPassword контролює, щоб пароль відповідав мінімальним вимогам безпеки – мав не менше восьми символів і містив принаймні одну літеру та одну цифру.

Основна функція registerUser виконується при спробі відправити форму реєстрації. Вона зупиняє стандартну поведінку браузера, щоб запобігти перезавантаженню сторінки, і збирає значення полів форми: ім’я, прізвище, електронну пошту та пароль. Після цього здійснюється послідовна перевірка: усі

поля мають бути заповнені, email – відповідати правильному формату, а пароль – бути достатньо надійним. У разі невідповідності будь-якої умови користувачу виводиться відповідне повідомлення про помилку.

Якщо перевірки пройдені успішно, функція виконує асинхронний HTTP POST-запит на серверний маршрут /register, передаючи у тілі запиту зібрані дані у форматі JSON [40]. Відповідь сервера опрацьовується: у випадку успішної реєстрації користувач отримує повідомлення, його дані зберігаються у локальному сховищі браузера (localStorage), і відбувається перенаправлення на сторінку особистого кабінету. Якщо ж реєстрація не вдалася або виникла помилка, відповідне повідомлення виводиться на екран. Також реалізована обробка можливих помилок під час спроби з’єднання із сервером. (див. рисунок 3.9).

Рисунок 3.9 – Алгоритм реєстрація з перевіркою унікальності email та валідацією

Якщо валідація даних пройдена успішно і email є унікальним, наступним кроком є хешування пароля. Це важливий механізм безпеки, що дозволяє зберігати не сам пароль, а його зашифровану версію. На першому етапі використовується бібліотека bcrypt для хешування пароля. Функція bcrypt.hash(password, 10) виконує хешування на основі солі з 10 раундами, що дозволяє зберігати пароль у захищеному вигляді, навіть якщо база даних буде скомпрометована.

Далі створюється новий об’єкт користувача newUser, у який передаються ім’я, прізвище, електронна пошта та хешований пароль. Це об’єкт моделі User, що визначений через Mongoose як схема для MongoDB.Після цього новий користувач зберігається в базі даних за допомогою await newUser.save(), що є асинхронною операцією збереження документа у колекції MongoDB.

На завершення сервер повертає відповідь з HTTP-статусом 201 (створено) у форматі JSON [40]. У тілі відповіді міститься повідомлення про успішну реєстрацію та часткова інформація про користувача (без пароля), яка може використовуватись на фронтенді (див. рисунок 3.10).

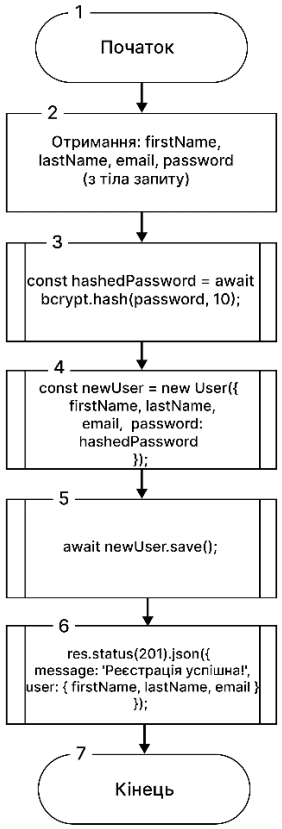


Рисунок 3.10 – Алгоритм хешування пароля перед збереженням користувача

Після реєстрації реалізовано маршрут /login, який забезпечує вхід користувача. Код реалізує клієнтську логіку для форми входу користувача на сайті. Він починається з двох функцій, які перевіряють правильність введених даних: одна переконується, що email відповідає стандартному формату, інша перевіряє, що пароль має мінімальну довжину. Основна функція `loginUser` реагує на подію відправлення форми, зупиняє стандартне оновлення сторінки, зчитує введені email і пароль, перевіряє їх коректність та відправляє асинхронний POST-запит на сервер за адресою `/login` з цими даними у форматі JSON [40]. Відповідь сервера обробляється: якщо вхід успішний, інформація про користувача зберігається у localStorage і відбувається перенаправлення на сторінку користувача, інакше відображається повідомлення про помилку (див. рисунок 3.11).

Рисунок 3.11 – Алгоритм авторизації з валідацією та перевіркою пароля

Крім стандартної форми входу, що на рисунку 3.11 у вигляді алгоритму, код також додає обробники подій для кнопок входу через Google і Facebook. При кліку на ці кнопки користувач перенаправляється на відповідні серверні маршрути `/auth/google` або `/auth/facebook`, де має відбуватися аутентифікація.

Для реалізації цієї авторизації через сторонні сервіси у додатку був розроблений файл passport.js, у якому налаштовані стратегії входу через Google та Facebook з використанням бібліотеки passport. Він забезпечує перевірку, створення користувача у базі даних, а також серіалізацію й десеріалізацію користувачів для збереження сесій.

На початку імпортуються необхідні модулі. Використовуються стратегії Google та Facebook з пакету passport-oauth, а також модель користувача User.

Далі експортується функція, яка приймає об’єкт passport та налаштовує дві стратегії. Спочатку налаштовується Google Strategy. Для цього передається clientID, clientSecret та callbackURL, які зберігаються в змінних оточення. Коли користувач проходить автентифікацію через Google, з об'єкта profile береться email, після чого система перевіряє, чи є такий користувач у базі. Якщо немає – створюється новий користувач з інформацією з профілю (див. рисунок 3.12).

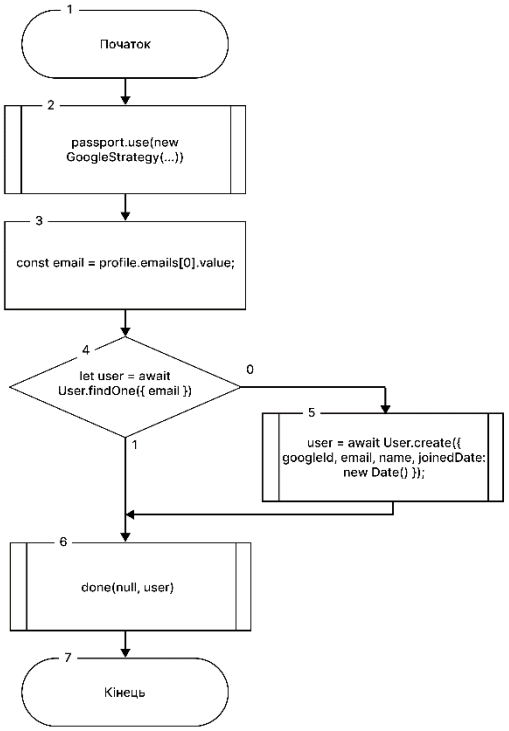


Рисунок 3.12 – Алгоритм входу за допомогою Google

Потім підключається Facebook Strategy. Так само передаються відповідні дані з .env, і вказується profileFields, щоб отримати email та ім’я користувача. Facebook не завжди надає email, тому код перевіряє, чи доступна така інформація, і тільки тоді проводить пошук або створення користувача в базі.

Після визначення стратегій обов’язково потрібно реалізувати серіалізацію та десеріалізацію користувача. Серіалізація означає збереження ідентифікатора користувача у сесії, а десеріалізація – витягування повного об’єкта користувача з бази даних за його ID.

Для виходу з системи реалізована проста функція очищення локального сховища.

Таким чином, реалізований модуль авторизації та реєстрації дозволяє створювати безпечну взаємодію з користувачем. Вбудована валідація значно зменшує навантаження на сервер, а хешування паролів забезпечує захист особистих даних. Зберігання облікових записів у MongoDB дозволяє масштабувати систему та легко управляти користувачами.

3.4 Розробка модуля для взаємодії з благодійними зборами

Модуль благодійних зборів є ключовим елементом платформи, яка об’єднує людей, готових допомагати іншим. Він забезпечує зручний інтерфейс для створення та пошуку зборів, дозволяє фільтрувати їх за різними категоріями допомоги, підтримувати комунікацію через коментарі та оцінки, а також робити пожертви. Це сприяє прозорості, довірі між користувачами і стимулює активну участь громади, що у свою чергу підвищує ефективність збору коштів і допомоги тим, хто її потребує.

Інтерфейс модуля складається зі списку благодійних зборів, де кожен збір представлений у вигляді картки або рядка з основною інформацією: назва, короткий опис, категорія допомоги, прогрес збору коштів (порівняння цільової суми та поточної), рейтинг волонтера, а також кнопки для взаємодії – донат, коментарі, лайки. Користувачі можуть фільтрувати збори за категоріями. На детальній сторінці збору доступна розгорнута інформація, список коментарів,

можливість залишити відгук та подивитись статистику. Інтерфейс розробленого модуля для взаємодії з благодійними зборами зображено на рисунку 3.13.

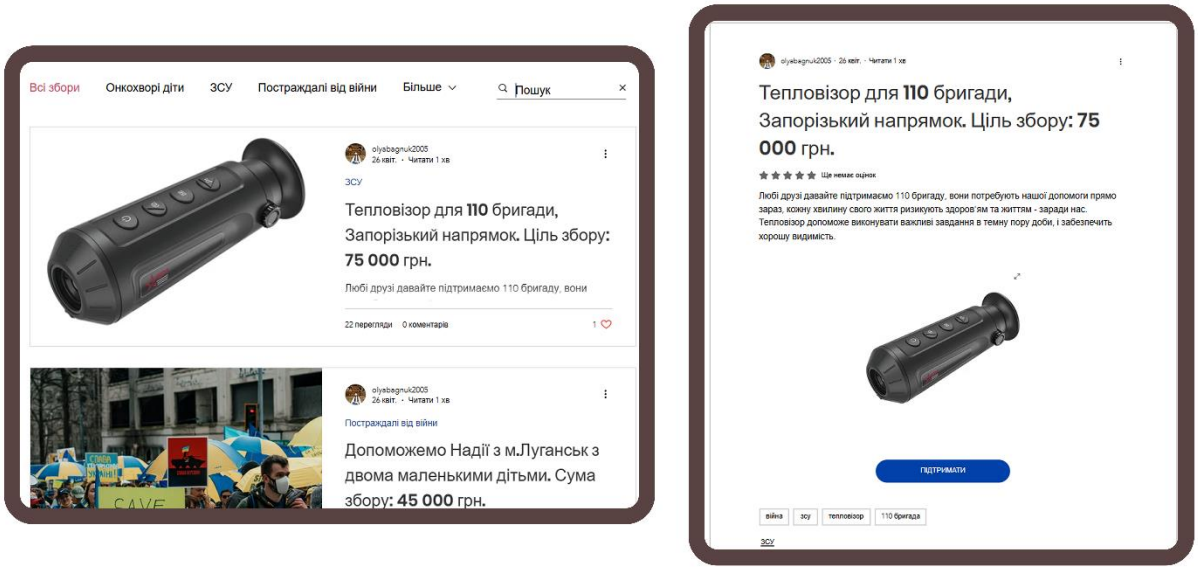


Рисунок 3.13 – Інтерфейс модуля для взаємодії з благодійними зборами

Для збереження інформації про благодійні збори, використано базу даних MongoDB, а саме бібліотеку Mongoose. Кожен збір має автора, який посилається на користувача, категорію, що обмежена конкретним переліком типів допомоги, назву та опис збору.

Для відображення інформації про благодійний збір реалізовано React-компонент FundraiserCard [41] із додатковим функціоналом: лайки, коментарі, теги та кнопки для поширення у соцмережах. Використано бібліотеки React (включно з хуком useState для керування локальним станом лайків) і react-router-dom (useNavigate для переходу на сторінку збору). Компонент приймає data – об’єкт із деталями збору, показує заголовок, короткий опис, прогрес зібраних коштів, рейтинг, теги та кілька коментарів. Для лайків використовується локальний стан (useState), а функція handleLike оновлює його при кліку, запобігаючи одночасному переходу за посиланням. Кнопки соцмереж викликають функцію shareToSocial, яка відкриває вікна з URL для публікації у Facebook або Twitter тощо.

Щоб відображати список зборів коштів (фандрайзерів) із можливістю фільтрації та пошуку використано React-компонент FundraiserList**,** який працює на стороні клієнта в браузері. Використовує useState для станів fundraisers і filters та useEffect для виклику API через api.get('/api/fundraisers', { params: filters }) при зміні фільтрів. Компоненти FundraiserFilters та FundraiserSearch оновлюють фільтри через onChange і onSearch. Якщо fundraisers не пустий, відображає список через fundraisers.map(f => <FundraiserCard key={f._id} data={f} />), інакше виводить повідомлення про відсутність результатів (див. рисунок 3.14).

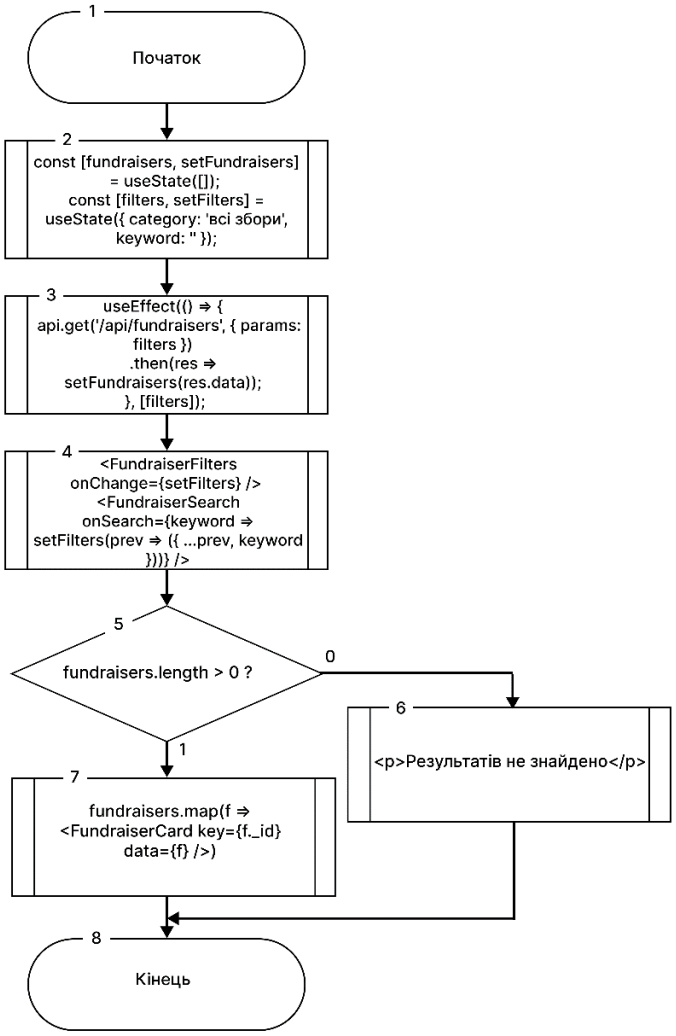


Рисунок 3.14 – Алгоритм фільтрації та відображення зборів з використанням React-компонентів і API-запиту

Для серверної логіки було реалізовано модуль fundraiserController.js, який відповідає за обробку запитів, пов’язаних зі зборами коштів (fundraisers). У цьому

файлі визначено три основні функції, які взаємодіють із базою даних через модель Fundraiser:

1. getFundraisers – отримує список зборів на основі фільтрів, які приходять з клієнта (категорія та ключове слово). Якщо задано категорію, що не дорівнює «всі збори», або ключове слово – формується відповідний запит до бази даних. Результатом є список зборів із приєднаними авторами, який повертається клієнту у форматі JSON (див. рисунок 3.15).

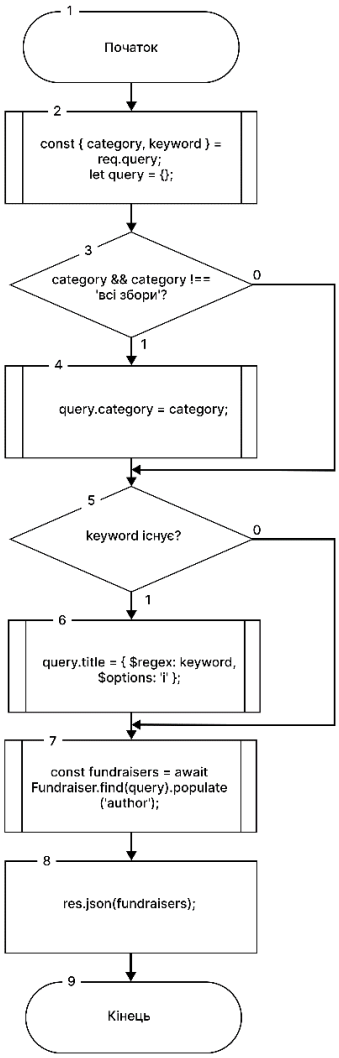


Рисунок 3.15 – Алгоритм фільтрації та отримання списку зборів з бази даних

2. getFundraiserById – дозволяє отримати детальну інформацію про конкретний збір коштів за його унікальним ідентифікатором. При кожному запиті кількість переглядів цього збору автоматично збільшується (див. рисунок 3.16).

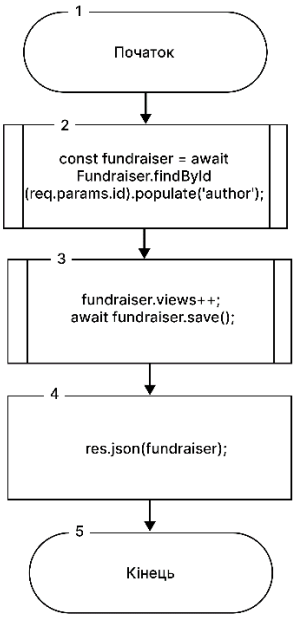


Рисунок 3.16 – Алгоритм отримання одного збору за ID та збільшення переглядів

3. rateFundraiser – реалізує механізм оцінювання зборів. На основі зіркової оцінки, яку надсилає користувач, перераховується середній рейтинг збору та зберігається оновлена інформація в базі (див. рисунок 3.17).

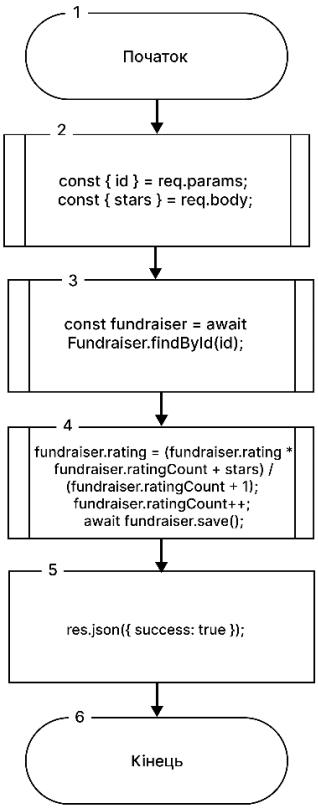


Рисунок 3.17 – Алгоритм оновлення рейтингу збору на основі нової оцінки

Також реалізовано клієнтську частину (frontend) додатку, що відповідає за відображення повної інформації про конкретний благодійний збір. Компонент FundraiserDetailsPage призначений для сторінки деталей збору коштів і використовується у маршрутизації React.

На початку компонент отримує id збору з URL через useParams. Далі, за допомогою хука useEffect, відправляється GET-запит до серверу (/api/fundraisers/:id) з метою отримання повної інформації про відповідний збір. Отримані дані зберігаються в стані fundraiser через useState.

Якщо дані ще не завантажені, на екрані показується повідомлення «Завантаження...». Після завантаження компонент виводить заголовок збору, ім’я автора, опис, суму зібраних коштів, мету збору та кількість переглядів тощо. Також підключаються три додаткові компоненти: FundraiserRating для оцінювання збору, DonationModal для внесення пожертв, і FundraiserComments для перегляду/додавання коментарів.

Отже, цей компонент виконує функцію відображення детальної сторінки збору для користувачів на клієнтському інтерфейсі.

3.5 Висновки

У результаті розробки модуля авторизації та реєстрації, платіжної системи та модуля для взаємодії з благодійними зборами було реалізовано повноцінну платформу, що відповідає вимогам безпеки, зручності користування та масштабованості. Використання сучасного стеку JavaScript-технологій, зокрема React на фронтенді та Node.js з Express на бекенді, дозволило досягти високої продуктивності та забезпечити ефективну взаємодію між клієнтською і серверною частинами системи.

Інтеграція платіжної системи Stripe дала змогу реалізувати як одноразові перекази, так і щомісячні підписки, що суттєво розширює можливості залучення фінансової підтримки до благодійних проєктів. Використання токенізованих платіжних даних гарантує дотримання стандартів безпеки та захист конфіденційної інформації.

Особливу увагу було приділено захисту користувацьких даних – реалізовано механізми реєстрації з валідацією введених даних, перевіркою унікальності, безпечним хешуванням паролів за допомогою bcrypt та автентифікацією з використанням JWT. Це дозволяє забезпечити контрольований доступ до функціоналу системи та персоналізовані можливості для користувачів.

Таким чином, створене рішення повністю відповідає поставленій меті – розробити гнучкий, сучасний і надійний застосунок для організації благодійних ініціатив.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Аналіз методів тестування програмного забезпечення

Тестування є критично важливою складовою розробки програмних модулів будь-якої системи, особливо коли мова йде про сервіси, що підтримують благодійні ініціативи. Система для планування та координації благодійних заходів має забезпечувати безперебійну роботу, коректне збереження даних, надійний обмін інформацією та високу доступність для користувачів. Помилки або нестабільність у такій системі можуть призвести до втрати даних, зниження довіри з боку благодійників та організацій, або навіть до зірваних зборів та ініціатив.

Тестування – це процес перевірки програмного забезпечення з метою виявлення дефектів та забезпечення відповідності розробленої системи встановленим вимогам і очікуванням користувачів. Тестування дозволяє виявити помилки на ранніх етапах розробки, мінімізуючи ризики та вартість виправлення недоліків у майбутньому [42].

При розробці систем варто звернути увагу на основні види тестування [42]:

Функціональне тестування перевіряє, чи працюють усі функції системи відповідно до вимог. Наприклад, у системі для координації благодійних ініціатив важливо, щоб коректно працювали реєстрація користувачів, створення подій та обробка заявок. Без функціонального тестування неможливо гарантувати, що користувачі отримають передбачуваний результат при взаємодії із системою.

Тестування безпеки. Безпека – ключовий аспект для будь-якої платформи, що обробляє особисті дані, особливо фінансові транзакції. Тестування безпеки допомагає виявити вразливості, які можуть призвести до витоку інформації або несанкціонованого доступу. Для благодійної системи важливо забезпечити, щоб усі дані користувачів були захищені, а внески надходили без ризику шахрайства.

Тестування зручності користування оцінює, наскільки інтуїтивно зрозумілою та доступною є система для кінцевого користувача. Якщо інтерфейс складний або незрозумілий, люди можуть відмовитись від участі в благодійних

акціях. Важливо, щоб навіть користувач без технічної підготовки міг легко зареєструватися, зробити внесок або організувати захід.

Тестування продуктивності. Продуктивність системи визначає її здатність ефективно працювати під навантаженням. Під час масштабних благодійних кампаній на платформу може заходити велика кількість людей одночасно. Тестування продуктивності дозволяє переконатися, що система швидко обробляє запити і не сповільнюється в пікові моменти, що є критично важливим для успіху заходів.

Тестування сумісності. Цей вид тестування перевіряє, чи працює система однаково добре на різних пристроях, операційних системах та браузерах. У світі, де користувачі заходять із мобільних телефонів, планшетів і комп'ютерів, дуже важливо забезпечити стабільний досвід незалежно від платформи. Інакше частина аудиторії може просто не отримати доступу до важливих функцій.

Інтеграційне тестування оцінює, наскільки правильно взаємодіють між собою різні частини системи. У системі благодійних ініціатив це може бути перевірка того, як працюють разом модуль реєстрації, база даних користувачів та модуль надсилання повідомлень. Без інтеграційного тестування навіть правильно працюючі окремі модулі можуть викликати збої при взаємодії.

Регресійне тестування виконується після внесення змін у код, щоб упевнитися, що нові зміни не порушили вже існуючу функціональність. У процесі постійного розвитку благодійної системи – наприклад, додавання нових функцій або оптимізацій – важливо гарантувати, що вжеі реалізовані можливості продовжують працювати без помилок.

Після аналізу основних видів тестування було визначено, що для забезпечення надійності та ефективності роботи системи необхідно ретельно перевірити її відповідність щодо тестування. Це дозволяє виявити потенційні недоліки на ранніх етапах розробки та забезпечити стабільну роботу системи при реальному використанні, мінімізуючи ризики збоїв, підвищуючи рівень користувацького досвіду та сприяючи загальній якості програмного продукту.

4.2 Тестування системи

Тестування вебзастосунку включає перевірку його функціональності, безпеки та правильності виконання на пристроях, що дозволяє переконатися у стабільній роботі інтерфейсу, коректному відображенні контенту, відповідності навігації очікуваній логіці, а також у стійкості до потенційних загроз з боку користувача або зовнішнього середовища.

Першим кроком тестування стало функціональне тестування модулів авторизації та реєстрації користувачів [43]. Було перевірено правильність обробки даних та відповідність інтерфейсу очікуваній поведінці системи. Вікно реєстрації та авторизації зображено на рисунку 4.1.

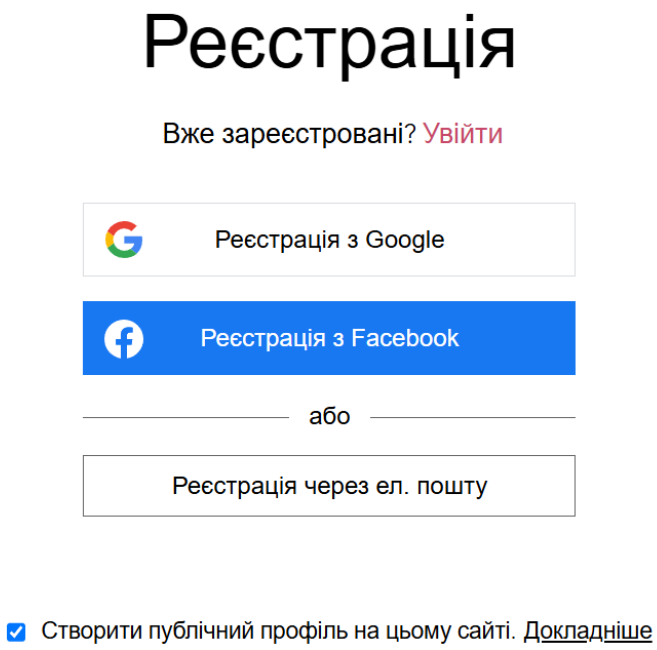


Рисунок 4.1 – Вікно реєстрації та авторизації

Під час тестування модуля авторизації було перевірено обробку помилок введення даних. Зокрема, тестування ситуації, коли користувач вводить неправильну електронну адресу чи невірний пароль. Система коректно розпізнає цю помилку та виводить відповідне повідомлення про те, що пароль або адреса є невірними. Це підтвердило правильність реалізації механізму валідації облікових даних (див. рисунок 4.2).

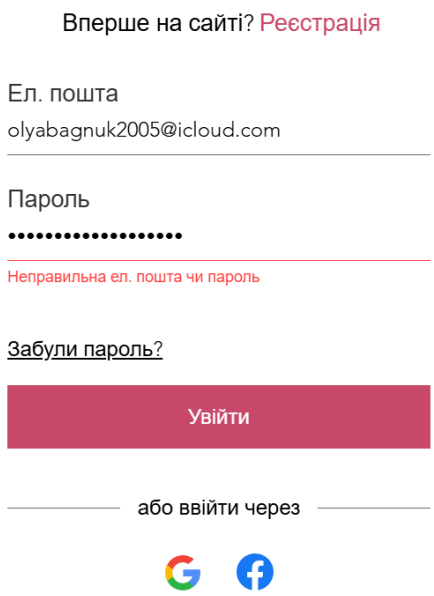


Рисунок 4.2 – Повідомлення про неправильно введені дані

У процесі тестування реєстрації та авторизації було перевірено наявність механізму, що верифікує користувача на предмет того, чи не є він автоматичним ботом. Це тестування пройшло успішно, оскільки система коректно верифікувала користувачів і не допустила спроб автоматичної реєстрації (див. рисунок 4.3).

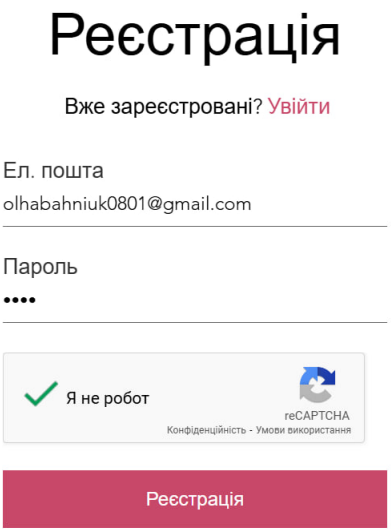


Рисунок 4.3 – Успішна верифікація користувача

Під час тестування було перевірено ситуацію, коли користувач вводить літери українського алфавіту у поле паролю під час реєстрації або авторизації. У разі введення нелатинських символів система коректно виводить повідомлення про

помилку, вказуючи, що можна використовувати лише латинські символи. Це забезпечує правильність введених даних і дозволяє уникнути проблем з обробкою символів, які можуть бути непідтримувані системою. Тестування цього аспекту пройшло успішно, підтвердивши, що система правильно реагує на невідповідність вимогам до вводу (див. рисунок 4.4).

Такі перевірки важливі для забезпечення сумісності даних із системою та підвищення безпеки платформи, оскільки виключають можливість введення недопустимих символів, що можуть порушити працездатність системи або викликати помилки під час обробки інформації.

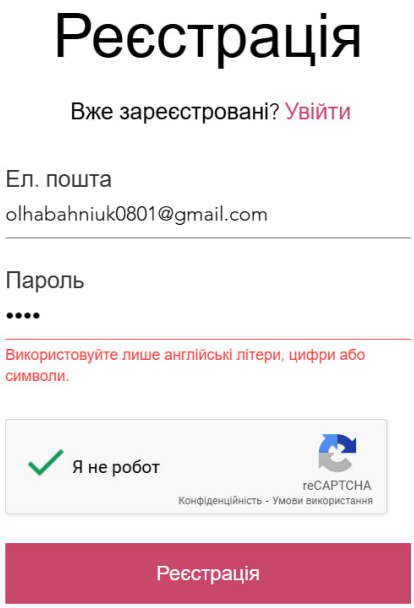


Рисунок 4.4 – Повідомлення про помилку при введенні українських літер

Під час тестування також було перевірено, чи після успішного входу або реєстрації відкривається особистий кабінет користувача. Після введення правильних облікових даних система коректно перенаправляє користувача на його персональну сторінку, що підтверджує правильність функціонування механізмів авторизації та перенаправлення. Це тестування успішно пройшло, що свідчить про належну інтеграцію цих процесів і правильне виконання функціональних вимог (див. рисунок 4.5).

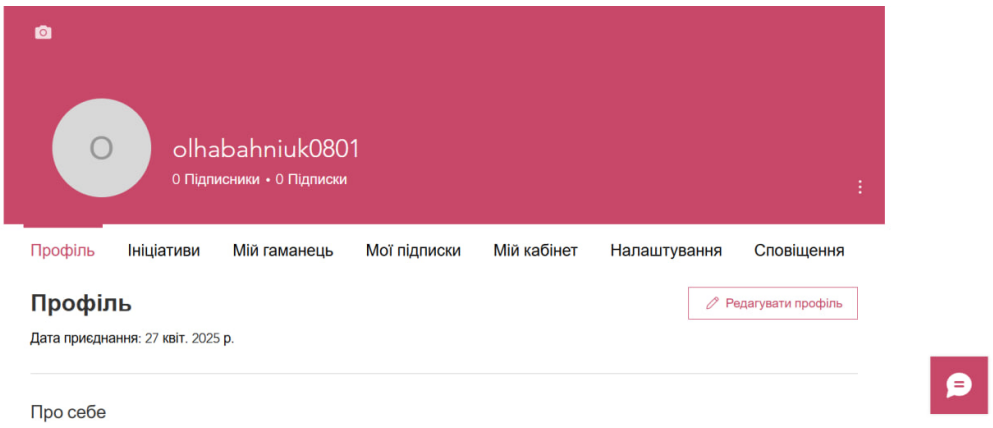


Рисунок 4.5 – Результат успішного входу у вебзастосунок

На наступному етапі тестування перевірялося функціонування анкети для створення нової благодійної ініціативи. Особливу увагу було приділено обов’язковим полям форми. У разі, якщо користувач намагається надіслати анкету без заповнення всіх обов’язкових полів, система коректно виводить повідомлення про помилку з вказівкою на необхідність заповнення відповідних даних. Надіслання анкети без обов'язкової інформації стає неможливим, що гарантує повноту та коректність переданої інформації організаторам для подальшого підтвердження або відхилення ініціативи (див. рисунок 4.6).

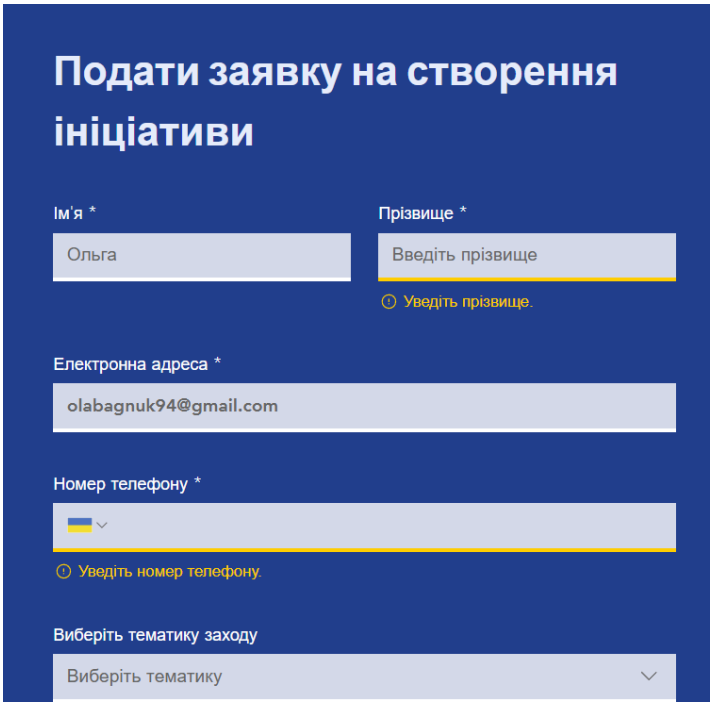


Рисунок 4.6 – Повідомлення про потребу заповнення обов’язкових полів

У випадку, коли всі обов'язкові поля анкети були заповнені коректно, система успішно виводить повідомлення про те, що відповідь надіслано. Це підтверджує правильне функціонування процесу надсилання форми та валідації даних. Результат успішного надсилання зображено на рисунку 4.7.

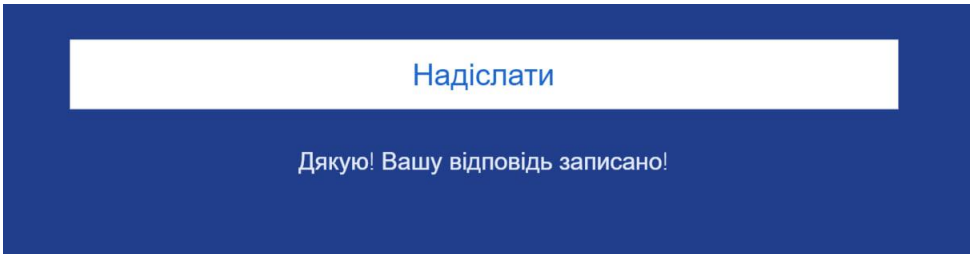


Рисунок 4.7 – Успішне надсилання форми створення ініціативи

Крім того, для остаточного підтвердження коректності роботи системи було перевірено надсилання сповіщення адміністратору на електронну адресу про створення нової ініціативи. Повідомлення надійшло успішно, що підтверджує правильну реалізацію механізму оповіщення та взаємодії між користувачем і адміністрацією платформи (див. рисунок 4.8).

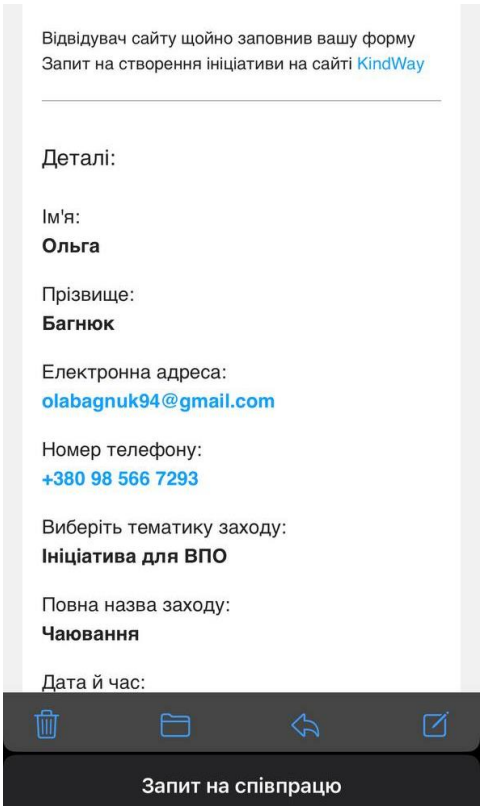


Рисунок 4.8 – Відображення повідомлення на електронній пошті

На наступному етапі тестування перевірялася взаємодія користувачів із благодійними зборами коштів. Було протестовано можливість залишати оцінку та відгук про волонтера та збір. Після додавання коментаря система миттєво, протягом приблизно 0,3 секунди, публікувала його на сторінці збору, що свідчить про високу швидкість обробки даних і коректність роботи механізму відгуків. Результат додавання коментаря під конкретним благодійним збором зображено на рисунку 4.9.

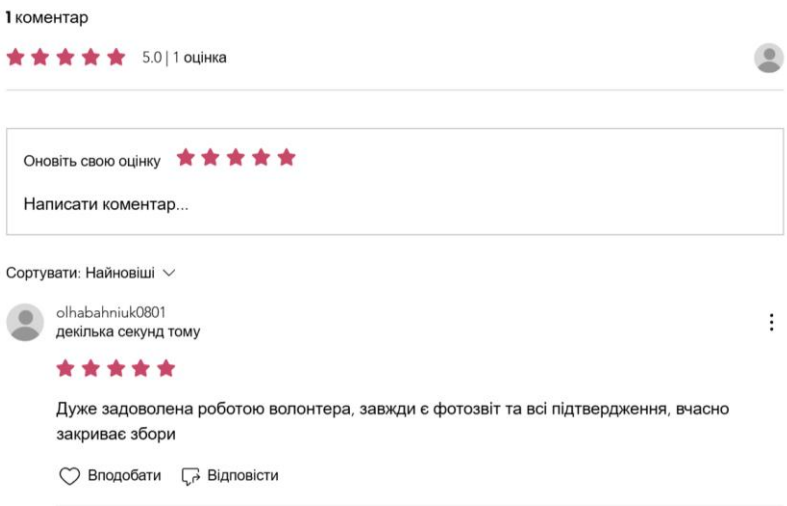


Рисунок 4.9 – Публікація коментаря під активним благодійним збором

Також було протестовано можливість видалення залишеного відгуку. Після запиту на видалення коментар видалявся швидко і без помилок, що підтверджує правильну роботу функціоналу керування відгуками. Усі операції виконувалися стабільно та відповідали вимогам до швидкодії та зручності користувача. Результат видалення зображено на рисунку 4.10.

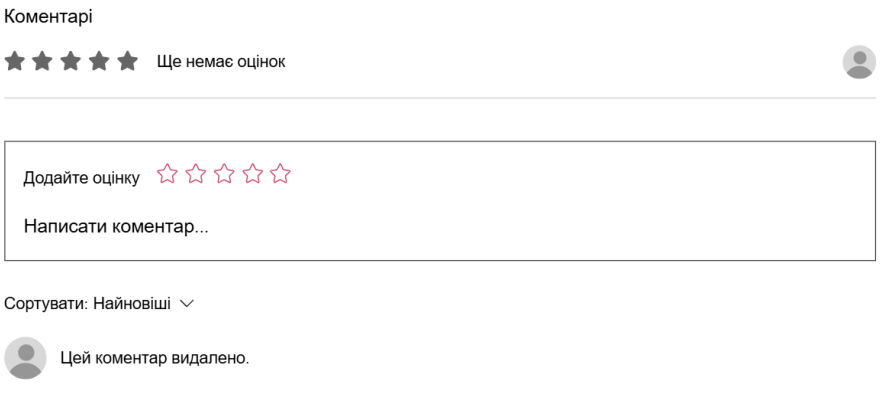


Рисунок 4.10 – Результат видалення коментаря

Далі тестувалася правильність роботи фільтрації постів із благодійними зборами за категоріями. Для перевірки було обрано категорію «Онкохворі діти». Після застосування фільтра система коректно відобразила лише ті пости, які належать до вибраної категорії, без змішування з іншими зборами. Це підтвердило правильність реалізації механізму фільтрації контенту та забезпечення зручності для користувачів у пошуку необхідних ініціатив (див. рисунок 4.11).

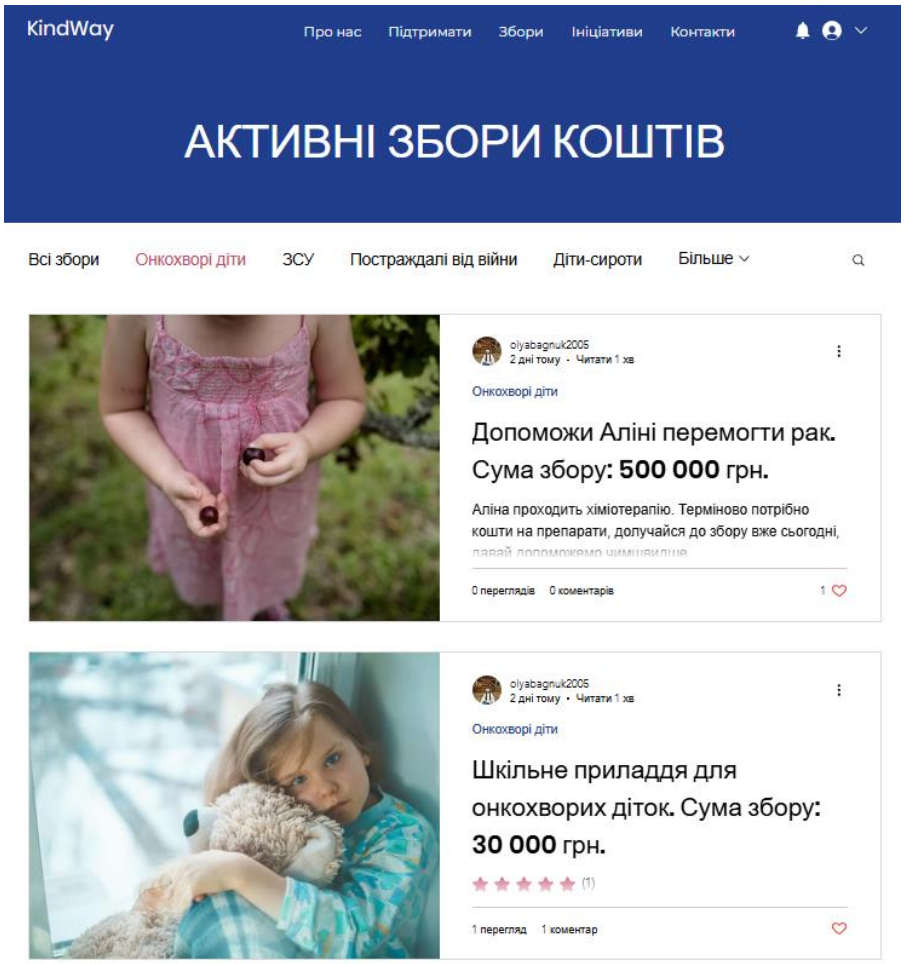


Рисунок 4.11 – Результат фільтрації по категорії «Онкохворі діти»

Наступним етапом було протестовано функціональність пошуку за ключовим словом. Для перевірки було введено слово «тварини». Система успішно знайшла та відобразила всі релевантні пости, пов'язані з допомогою тваринам, що підтверджує правильну роботу пошукового механізму та його здатність швидко знаходити потрібну інформацію за заданими запитами (див. рисунок 4.12).

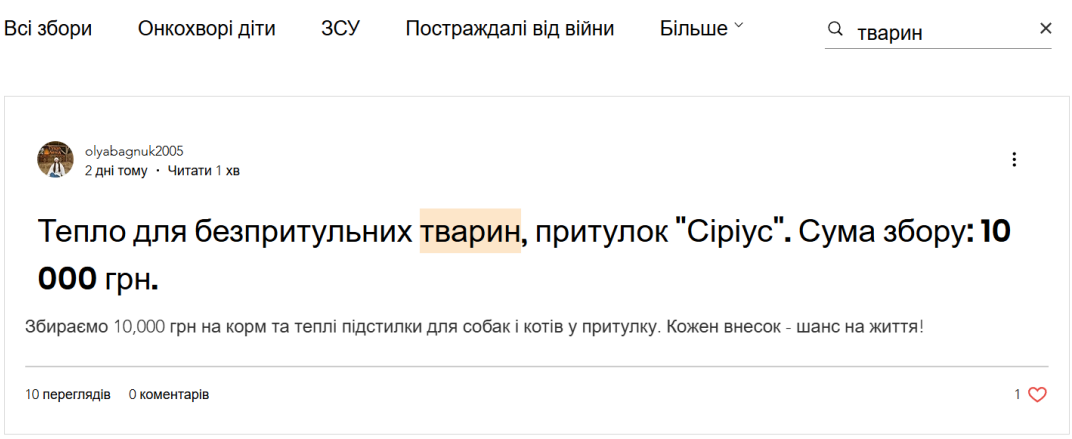


Рисунок 4.12 – Результати пошуку за ключовим словом «тварин»

Також було перевірено поведінку системи у випадку, коли користувач вводить у пошук слово, за яким немає відповідних результатів. Після введення вигаданого слова пошук не знайшов жодного поста, і система коректно вивела повідомлення про відсутність результатів із порадою спробувати інший запит. Це підтвердило правильність роботи механізму обробки порожніх результатів та забезпечення зручності для користувача (див. рисунок 4.13).

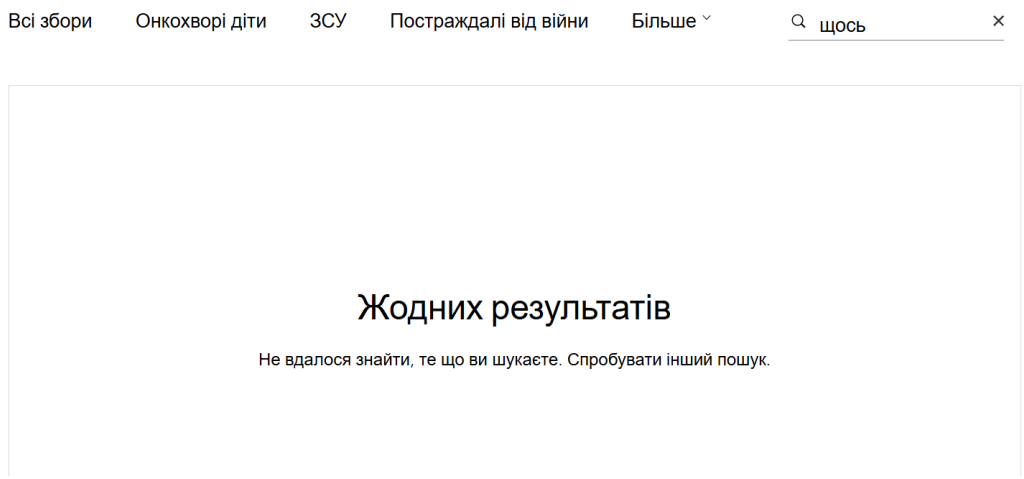


Рисунок 4.13 – Повідомлення про те, що пошук не дав жодних результатів

Також було протестовано ключовий етап роботи вебдодатку, а саме: реєстрації на ініціативу «Чистий дім для тварин з притулку». Спочатку користувачем заповнено необхідні поля анкети та надіслано запит на участь, процес реєстрації відображено на рисунку 4.14.

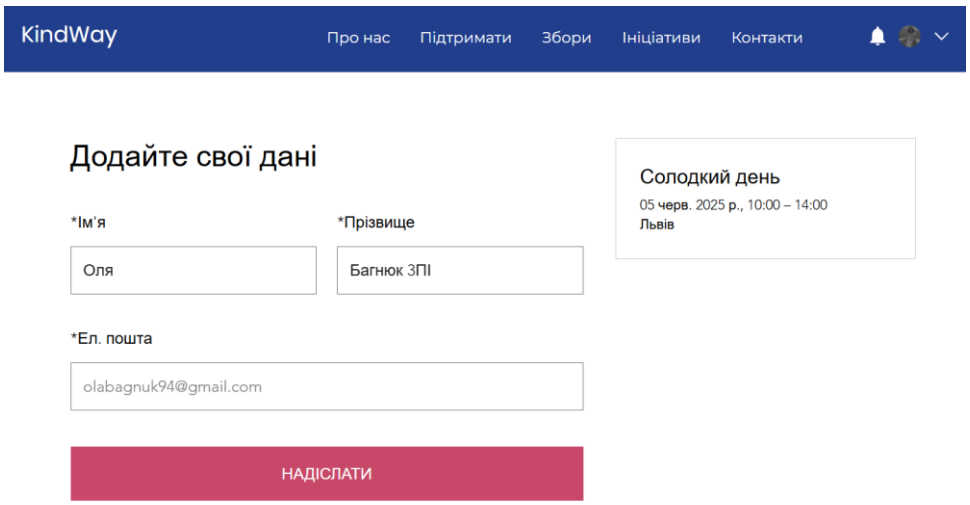


Рисунок 4.14 – Процес реєстрації на ініціативу

Після надсилання система коректно вивела повідомлення про успішне надсилання заявки та повідомила, що додаткова інформація буде надіслана на електронну адресу користувача (див. рисунок 4.15).

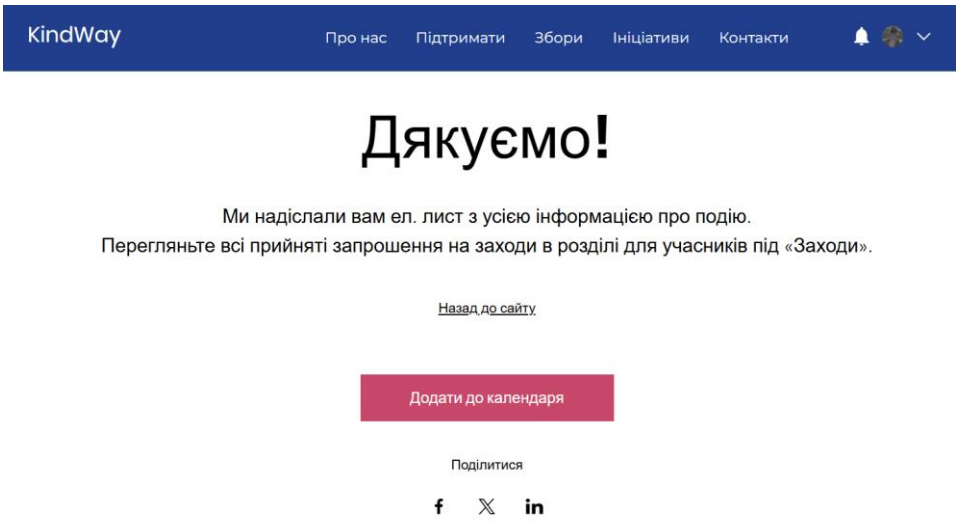


Рисунок 4.15 – Підтвердження успішної реєстрації

Додатково було перевірено, що в особистому кабінеті в розділі «Мої ініціативи» ініціатива «Чистий дім для тварин з притулку» відобразилася у списку майбутніх заходів. Це підтвердило правильність функціонування процесу реєстрації на ініціативи та оновлення інформації в особистому кабінеті. Результат відображено на рисунку 4.16.

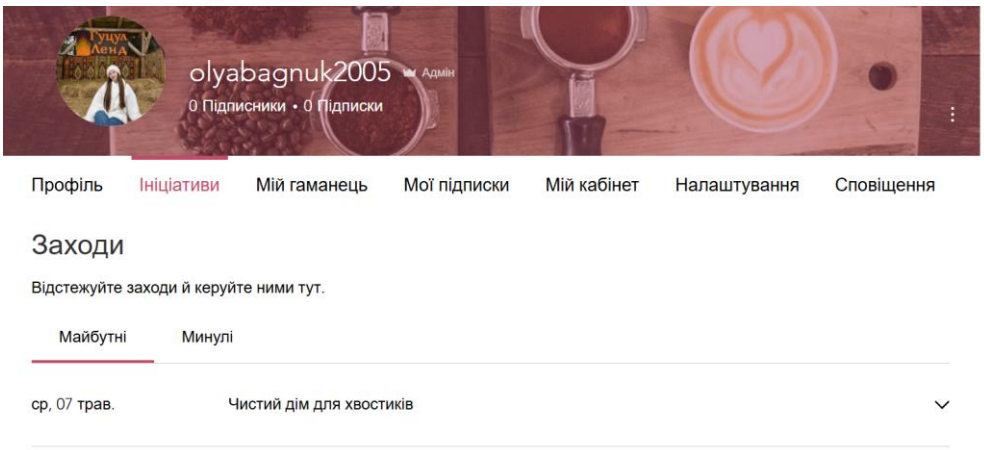


Рисунок 4.16 – Успішне додавання майбутньої ініціативи в особистий кабінет

Також важливо було протестувати розроблювальний програмний продукт у порівнянні з аналогами, щоб оцінити його якість, продуктивність та зручність у реальних умовах. Тестування відбувалося за допомогою Lighthouse – автоматизованого інструменту від Google, який аналізує вебзастосунки за кількома ключовими параметрами, такими як продуктивність, доступність, SEO та найкращі практики [44]. Тестування розроблювального вебдодатку зображено на рисунку 4.17.

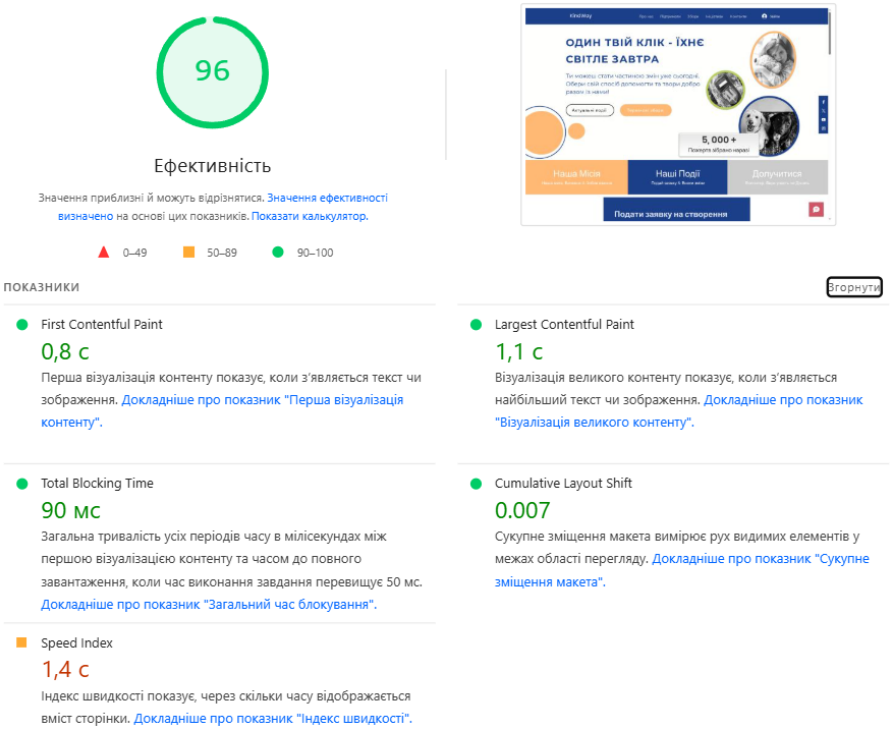


Рисунок 4.17 – Тестування за допомогою Lighthouse власного застосунка

Результати тестування аналога «HelpKarma» зображено на рисунку 4.18.

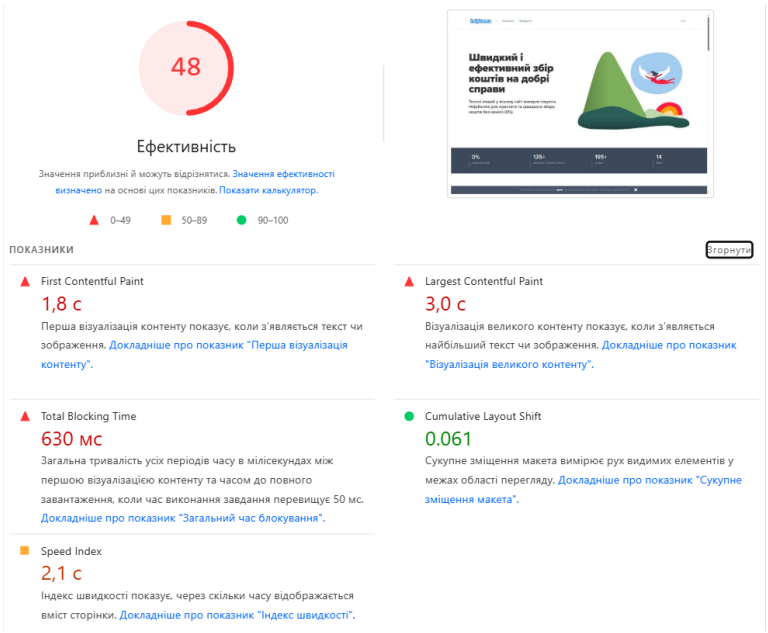


Рисунок 4.18 – Тестування за допомогою Lighthouse аналога «HelpKarma»

Розроблювальний вебзастосунок демонструє високу ефективність – 96%, що майже вдвічі перевищує показник аналога (48%). Це свідчить про більш оптимізовану роботу, кращу продуктивність та відповідність сучасним стандартам, що робить систему більш надійною і комфортною для користувачів.

Результати тестування аналога «ДоброДій» зображено на рисунку 4.19.

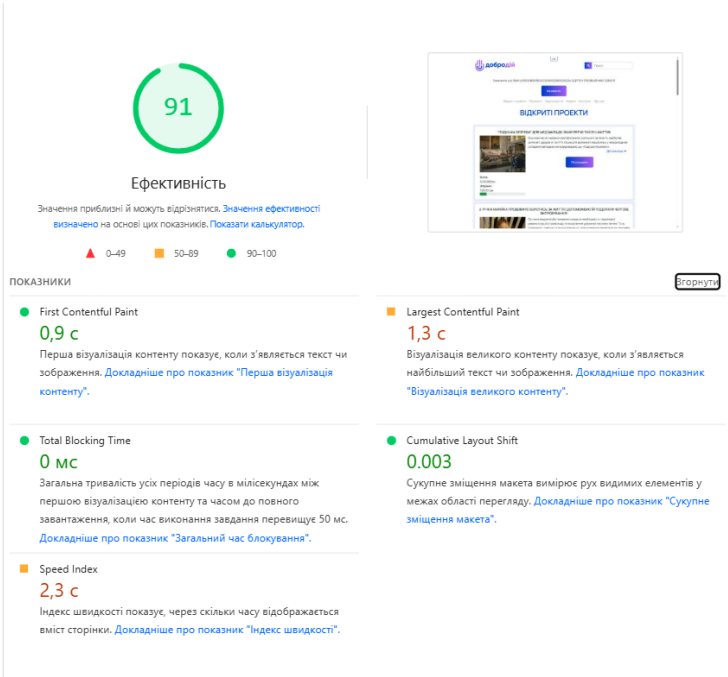


Рисунок 4.19 – Тестування за допомогою Lighthouse аналога «ДоброДій»

Результати тестування аналога «Зграя» зображено на рисунку 4.20.

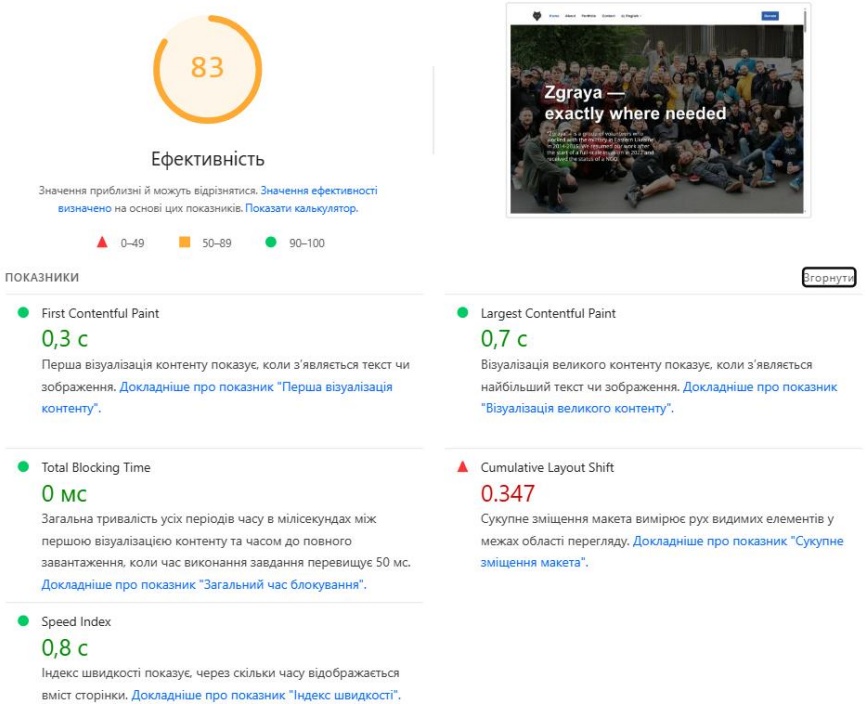


Рисунок 4.20 – Тестування за допомогою Lighthouse аналога «Зграя»

Результати тестування аналога «Peremoga» зображено на рисунку 4.21.

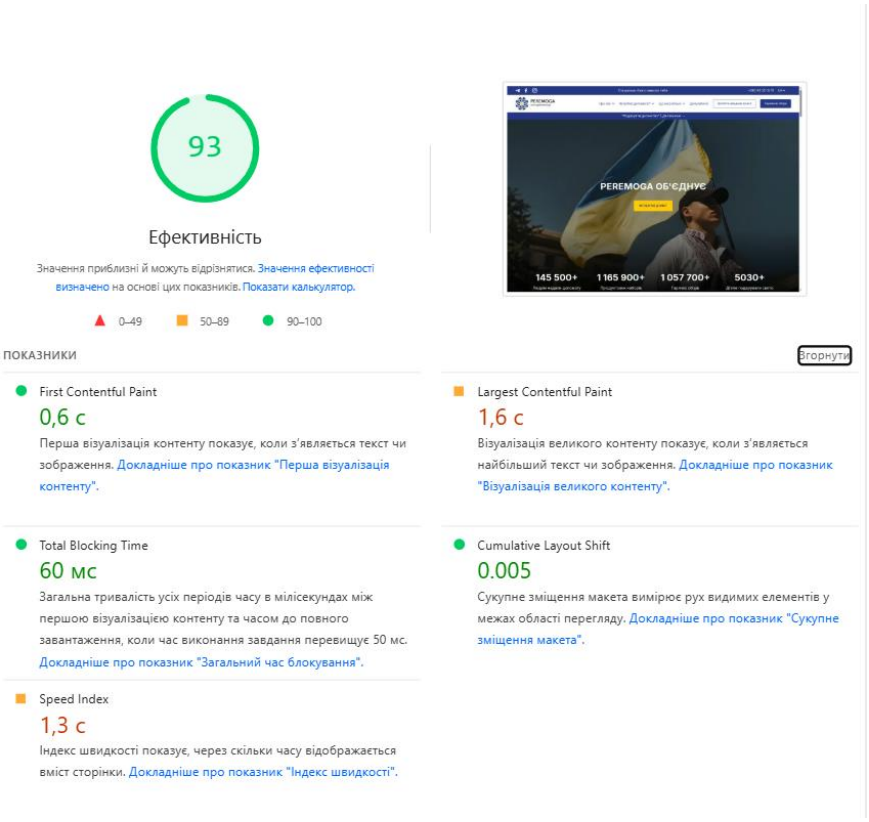


Рисунок 4.21 – Тестування за допомогою Lighthouse аналога «Peremoga»

Отже, розроблювальний програмний продукт показав відмінний результат – 96% ефективності, що перевищує показники аналогів, які становлять 93%, 91%, 84% та 48%. Це свідчить про високу якість, оптимізацію та надійність власної системи, навіть враховуючи більшу функціональність у порівнянні з деякими аналогами. Отже, вебдодаток для планування та координації благодійних ініціатив є конкурентоспроможним і відповідає сучасним стандартам веброзробки.

Також перевірено коректність відображення вебінтерфейсу у всіх популярних браузерах (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari), що підтверджує стабільність та кросбраузерність розробленої системи.

4.3 Розробка інструкції користувача

Розробка інструкції користувача є ключовим етапом створення системи для планування та координації благодійних ініціатив. Наявність доступної інструкції значно підвищує комфорт користування вебсистемою, знижує ризик помилок під час роботи та допомагає швидше опанувати її функціонал. Це особливо важливо для благодійних проєктів, де користувачі можуть мати різний вік і рівень цифрової грамотності.

Докладна інструкція сприяє:

– швидкому старту роботи нових користувачів без потреби у додатковій підтримці;

– мінімізації помилок при реєстрації, створенні ініціатив та взаємодії зі зборами;

– підвищенню загального рівня довіри до системи;

– зниженню навантаження на службу підтримки.

Робота з вебдодатком розпочинається з підготовки його до запуску у робочому середовищі. Для цього виконується команда збірки (npm run build), яка обробляє вихідний код і створює оптимізовану версію застосунку. У результаті утворюється папка (build), що містить готові статичні файли – HTML, CSS, JavaScript та інші ресурси, необхідні для коректної роботи додатку у браузері. Наступним кроком є розміщення цих файлів на хостингу. Після успішного

розгортання застосунок стає доступним за визначеною URL-адресою і готовий до використання кінцевими користувачами.

Перед встановленням важливо переконатися, що пристрій відповідає мінімальним технічним вимогам, щоб уникнути помилок або некоректної роботи програми. Всі параметри сумісності наведені в таблиці 4.1, що допомагає користувачам легко перевірити відповідність їхнього обладнання.

Таблиця 4.1 – Вимоги до апаратного забезпечення

Вимоги до конфігурації	Рекомендовано	Мінімально
Оперативна пам'ять	4 ГБ RAM	2 ГБ RAM
Вільний простір на диску	1ГБ	200МБ
Операційна система	Windows 10 / 11, Linux, macOS	Windows 7
Дисплей	Роздільна здатність не менше 1080x1920 пікселів (Full HD)	Роздільна здатність не менше 720x1280 пікселів (HD+)

Щоб забезпечити оптимальну продуктивність і стабільну роботу системи, бажано використовувати обладнання, що відповідає рекомендованим технічним вимогам. Водночас для базового користування, де не потрібна висока швидкість обробки або великі навантаження, буде достатньо й мінімальних характеристик.

Реєстрація та вхід до системи є першим кроком для користувачів, що бажають скористатися всіма функціями вебплатформи. Для цього необхідно перейти на головну сторінку вебсистеми і натиснути на кнопку «Реєстрація», якщо це новий користувач, або «Увійти», якщо обліковий запис вже існує. Після цього користувачу буде запропоновано ввести електронну адресу та пароль або увійти за допомогою Google чи Facebook.

Після успішного входу відображається головна сторінка з активним особистим кабінетом користувача, що відображається зліва у верхньому кутку (див. рисунок 4.22).

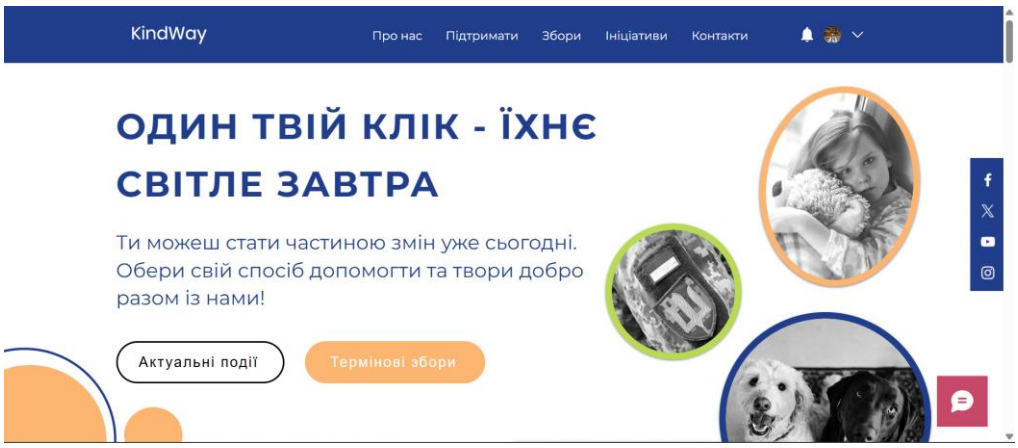


Рисунок 4.22 – Головна сторінка вебзастосунку

У особистому кабінеті користувач може легко переглядати та редагувати свої персональні дані, такі як ім’я, адреса електронної пошти та контактні номери. Це дозволяє підтримувати актуальність інформації та забезпечує зручність при взаємодії з платформою (див. рисунок 4.23). Крім того, є можливість додавати або змінювати фото профілю, що дозволяє персоналізувати акаунт.

У розділі «Сповіщення» зберігаються всі важливі повідомлення про нові події, акції, ініціативи або зміни статусу ініціатив користувача. Ця інформація завжди доступна для перегляду, що дозволяє не пропустити важливі оновлення та тримати руку на пульсі всіх подій.

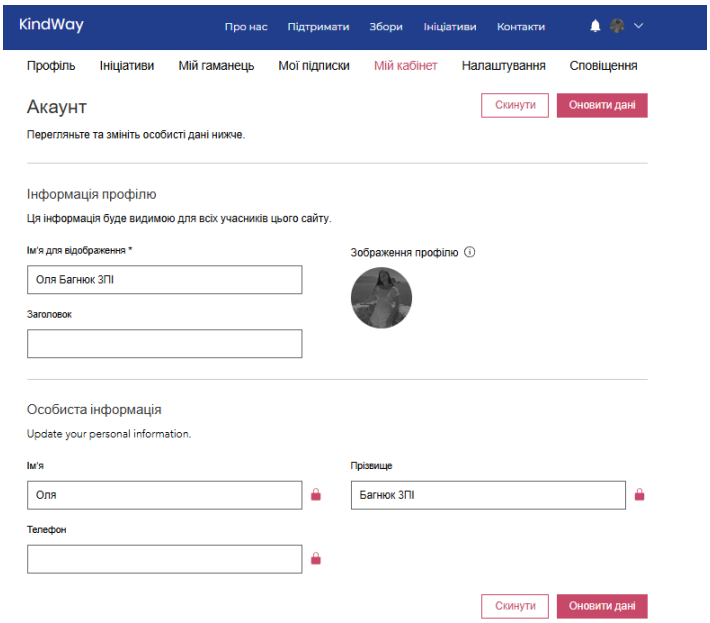


Рисунок 4.23 – Сторінка «Мій кабінет»

Крім того, в особистому кабінеті є можливість переглядати як майбутні, так і завершені ініціативи. Для майбутніх ініціатив коритсувач побачить інформацію про дату, час та місце проведення заходу. Завершені ініціативи містять звітність та результати діяльності, що дозволяє оцінити ефективність участі чи організації.

Інтерфейс особистого кабінету зображено на рисунку 4.24.

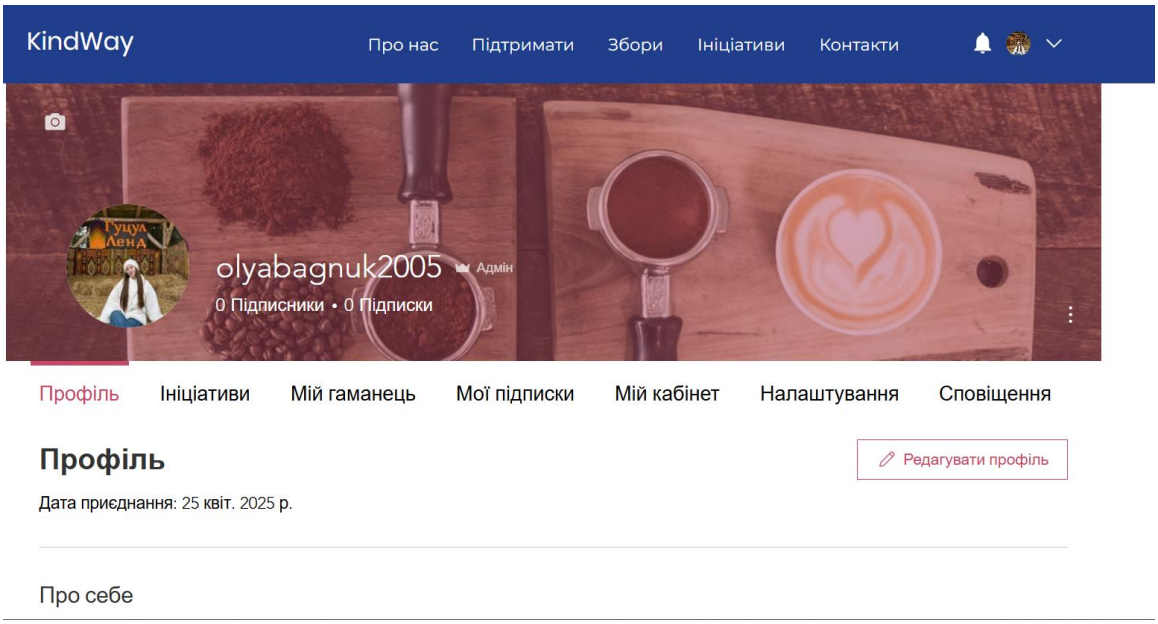


Рисунок 4.24 – Інтерфейс особистого кабінету користувача

Однією з ключових функцій системи є можливість створення ініціатив. Для цього необхідно перейти до головної сторінки та заповнити форму «Створити нову ініціативу». Користувач заповнює форму, вказуючи назву ініціативи, опис, категорію та дату, а також може додати інші важливі дані. Після заповнення форми користувач натискає на кнопку «Надіслати». У разі успішної реєстрації, система надає підтвердження, а ваша ініціатива з’являється у вашому особистому кабінеті. Також на вашу електронну пошту буде надіслано сповіщення, що підтверджує створення ініціативи.

На сторінці благодійних зборів користувачі можуть знайти детальну інформацію про кожен збір, включаючи опис, його мету, цілі та інші важливі деталі. Кожен збір має спеціальні кнопки, що дозволяють взаємодіяти з ним, наприклад, залишити лайк або коментар. Крім того, для кожного збору доступна

кнопка «Підтримати», за допомогою якої користувач має можливість надіслати кошти на підтримку. Це дозволяє користувачам швидко та зручно зробити благодійний внесок (див. рисунок 4.25).

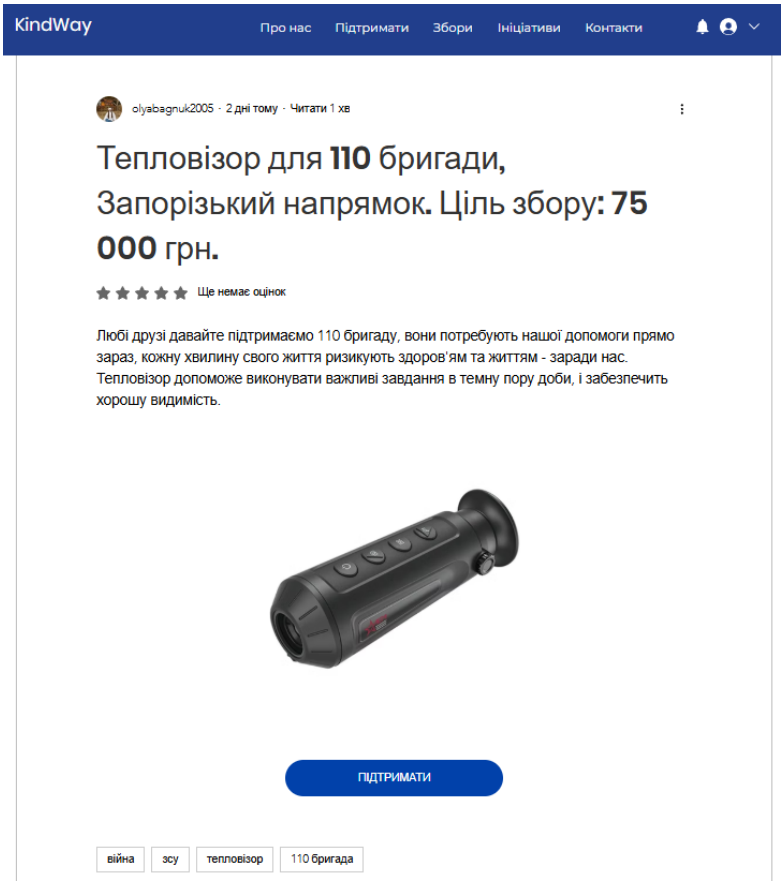


Рисунок 4.25 – Сторінка з описом обраного благодійного збору

Через меню дій (три крапки з правого боку) – користувач також може відстежувати пост або поділитися ним (див. рисунок 4.26).

Рисунок 4.26 – Меню дій в описі благодійного збору

На сторінці благодійного збору користувач має можливість залишати коментарі – це зручно і дозволяє кожному долучитися до обговорення або висловити підтримку. Блок коментарів розташований у нижній частині сторінки, безпосередньо під фото збору, що робить його легкодоступним та інтуїтивно зрозумілим (див. рисунок 4.27).

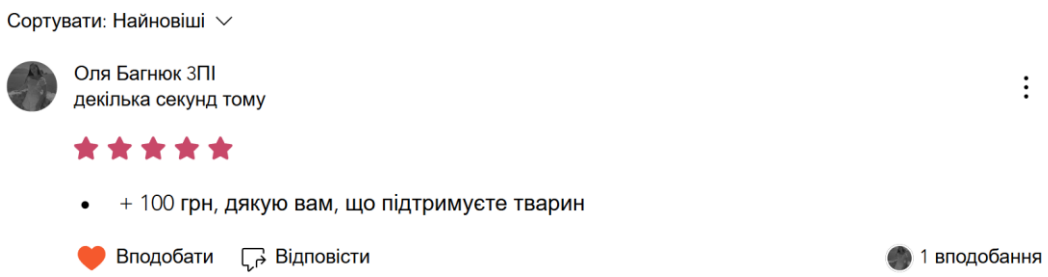


Рисунок 4.27 – Функція вподобання на сторінці «Збори»

Користувач також може зручно сортувати коментарі за типами – наприклад, за новизною чи популярністю. Функція сортування розміщена у верхній частині блоку коментарів, що дозволяє швидко знайти найцікавіші або найактуальніші відгуки (див. рисунок 4.28).

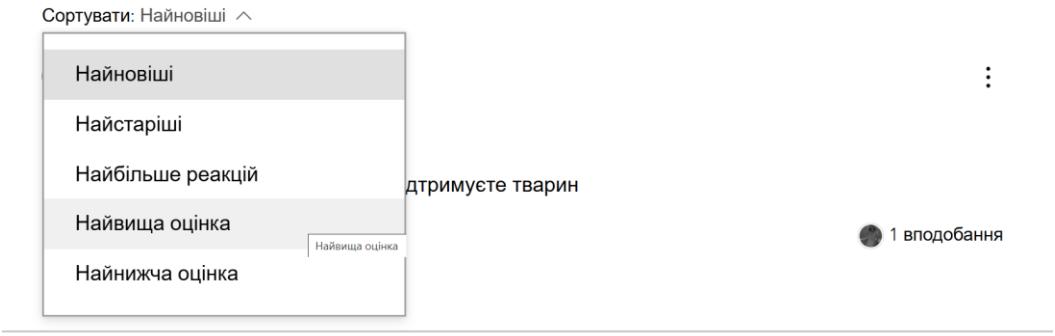


Рисунок 4.28 – Сортування коментарів за критеріями

Користувач також має можливість редагувати свої коментарі, видаляти їх або копіювати посилання – усе це доступно через контекстне меню, яке відкривається при натисканні на три крапки з лівого боку коментаря.

Завдяки коментарям користувачі можуть спілкуватися та обговорювати благодійність, підтримуючи один одного (див. рисунок 4.29). Вони мають змогу

відповідати на коментарі, що створює живий діалог і формує активну спільноту навколо збору.

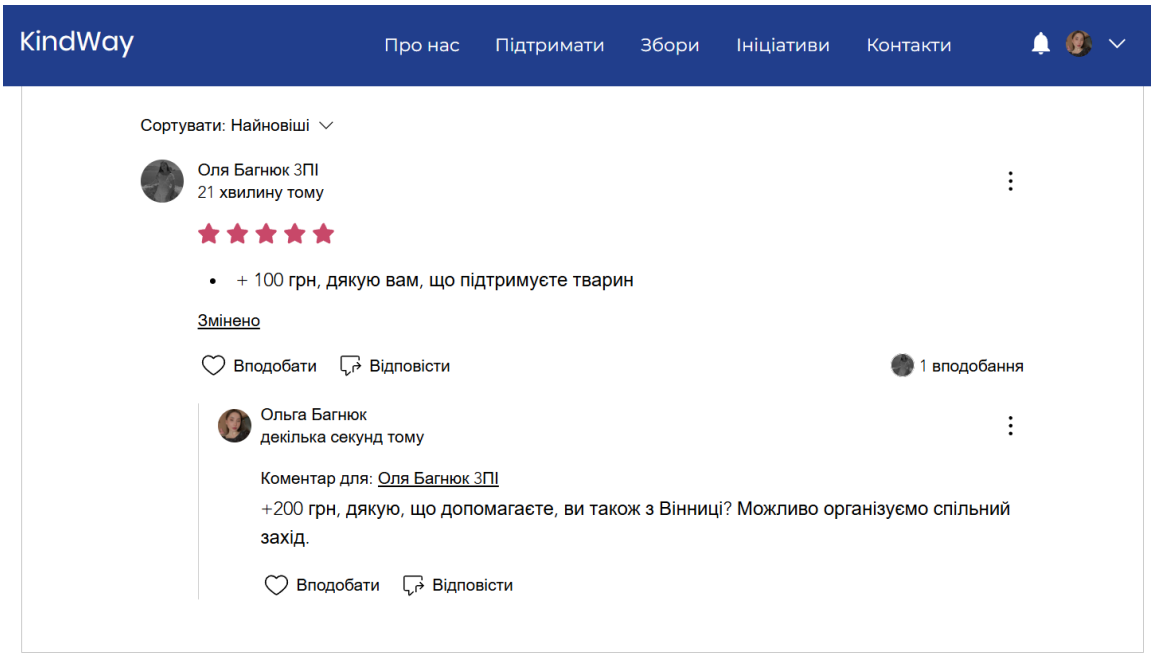


Рисунок 4.29 – Процес діалогу у коментарях до збору

Якщо користувач хоче переглянути профіль іншого, достатньо натиснути на його аватар – після цього відкриється сторінка профілю. За бажанням можна підписатися на цього користувача. Після успішної підписки йому надійде сповіщення в особистий кабінет про нового підписника. Сповіщення відображаться зверху з правого боку, поруч з аватаром особистого кабінету (див. рисунок 4.30).

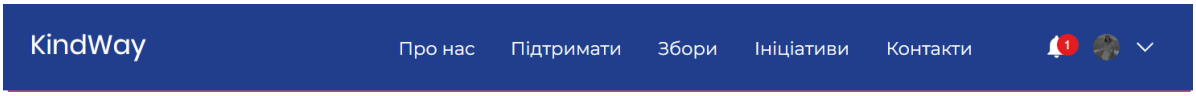


Рисунок 4.30 – Сповіщення про нову підписку

Після успішного виконання у верхній частині сторінки профілю, під аватаром користувача, відображається оновлена інформація – з’являється нова підписка, що обула успішно додана, і тепер, відображається «1 підписник» (див. рисунок 4.31).

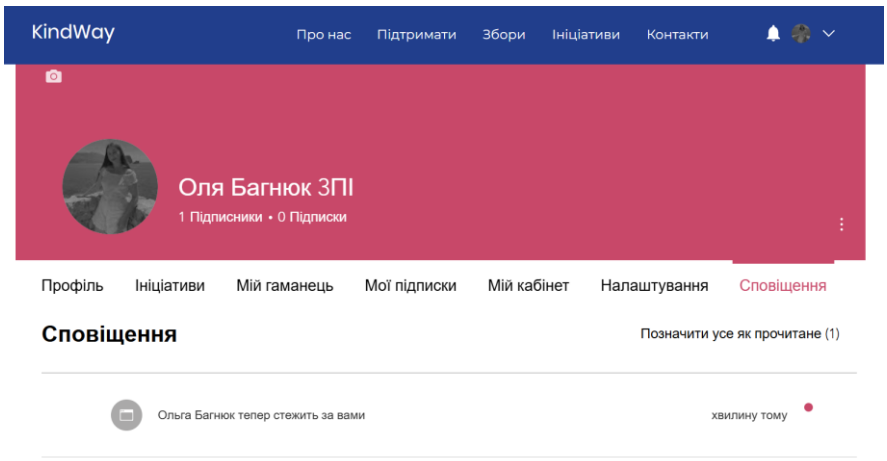


Рисунок 4.31 – Успішне додавання нового підписника

Щодо ініціатив, то на сторінці «Ініціативи» користувач має можливість переглянути точний час, дату та місце проведення події. Для зручності участі в заході на сторінці передбачена можливість зареєструватися на подію. Після реєстрації буде надано доступ до додаткової інформації, що допоможе краще підготуватися до заходу. Сторінка ініціатив зображена на рисунку 4.32.

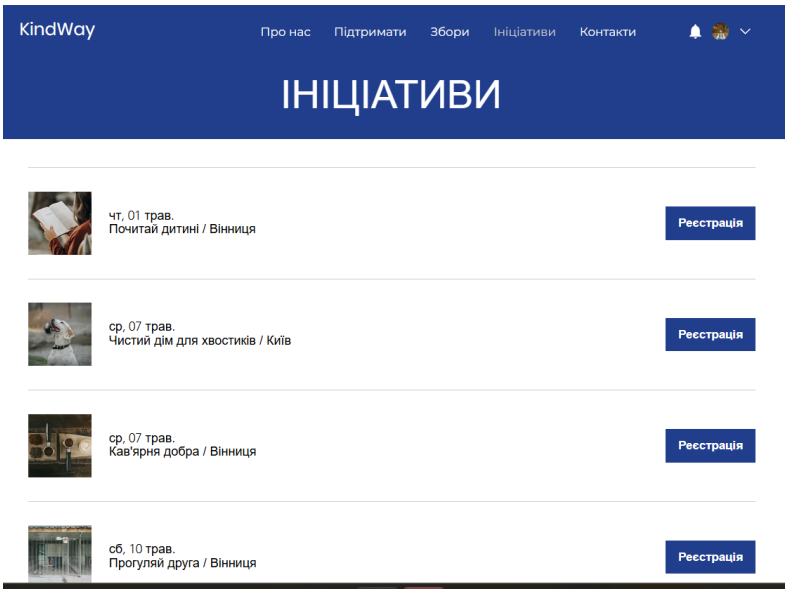


Рисунок 4.32 – Сторінка усіх актуальних ініціатив

Після реєстрації користувач може натиснути кнопку «Додати в календар», щоб зберегти подію у свій календар і не забути про неї. Це зручно для нагадування про важливі дати чи заплановану участь у зборі (див. рисунок 4.33).

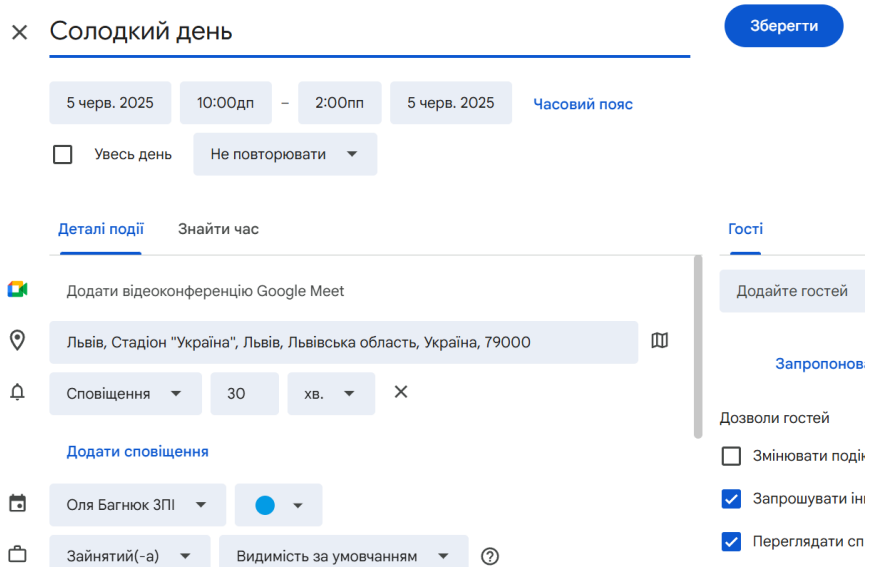


Рисунок 4.33 – Додавання події в календар

Після успішної реєстрації користувачу варто перевірити електронну пошту, яку він вказав під час реєстрації, щоб переконатися, що надійшов лист із підтвердженням (див. рисунок 4.34).

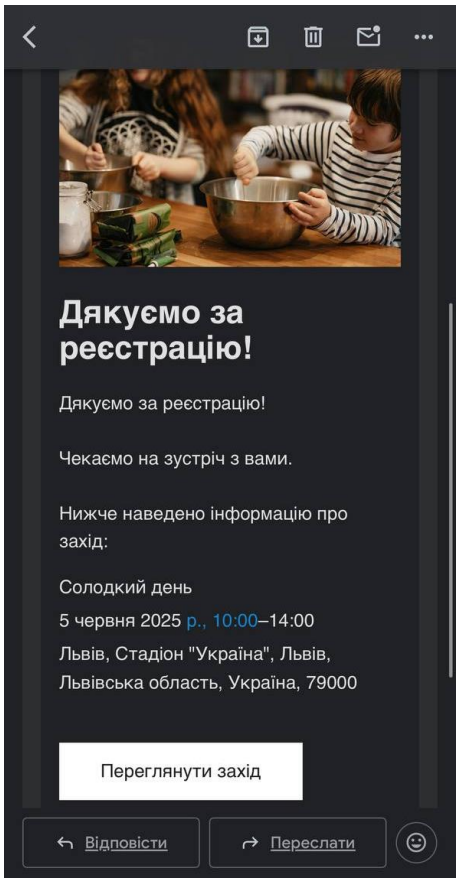


Рисунок 4.34 – Підтвердження про реєстрацію на захід

Окрім основної інформації, користувач також може побачити список людей, які вже зареєструвалися для участі в цій ініціативі. Це дає змогу краще орієнтуватися у складі учасників, побачити активність спільноти та відчути підтримку перед участю в заході (див. рисунок 4.35).

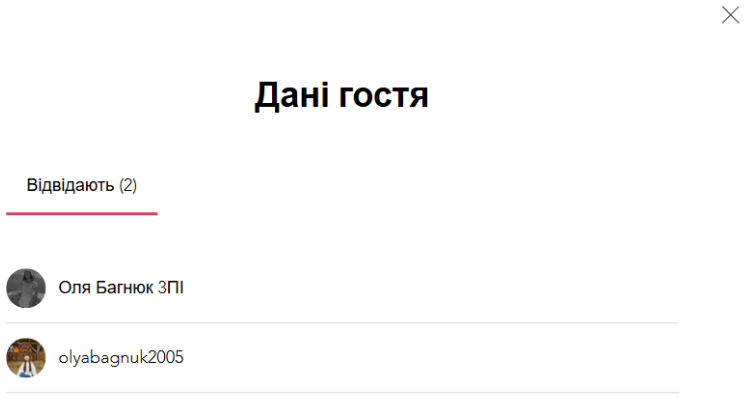


Рисунок 4.35 – Дані про гостей, що відвідають благодійний захід

Важливо, що після реєстрації користувач має можливість повернутися до ініціативи, на яку він відгукнувся, і за потреби скасувати свою відповідь. Це забезпечує гнучкість і дає змогу змінити рішення (див. рисунок 4.36).

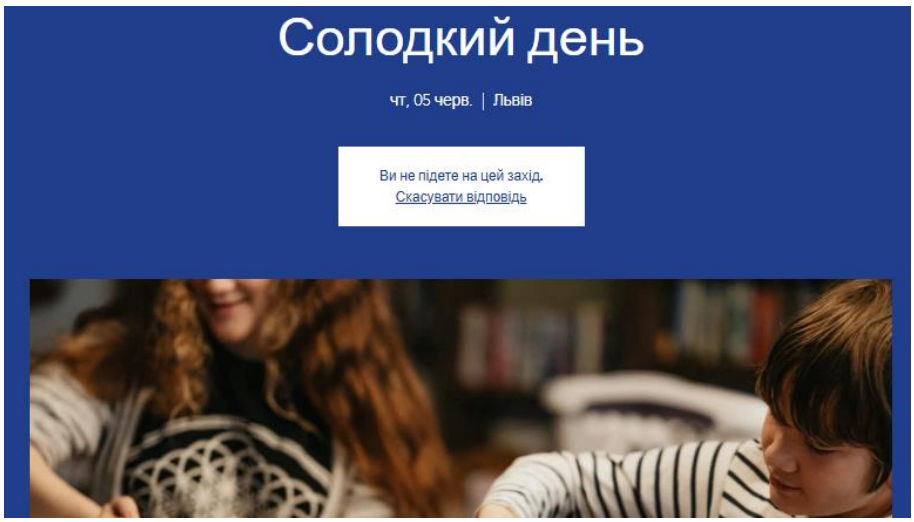


Рисунок 4.36 – Можливість скасування відповіді щодо реєстрації на захід

На сторінці «Підтримати» користувачі можуть оформити підписку для регулярної підтримки ініціатив, здійснити одноразову благодійну пожертву та

зв'язатись з волонтером-організатором за допомогою чат-боту. Для цього необхідно перейти до розділу «Підтримати» на сайті.

Після переходу на сторінку, користувач має можливість вибрати один з двох варіантів підтримки: щомісячну підписку або одноразову пожертву. Якщо користувач обирає щомісячну підписку, йому буде запропоновано ввести суму, яку він готовий жертвувати кожного місяця. Якщо коритсувач бажає зробити одноразову пожертву, достатньо вказати суму, яку він хоче передати на підтримку збору та підтвердити свою дію (див. рисунок 4.37).

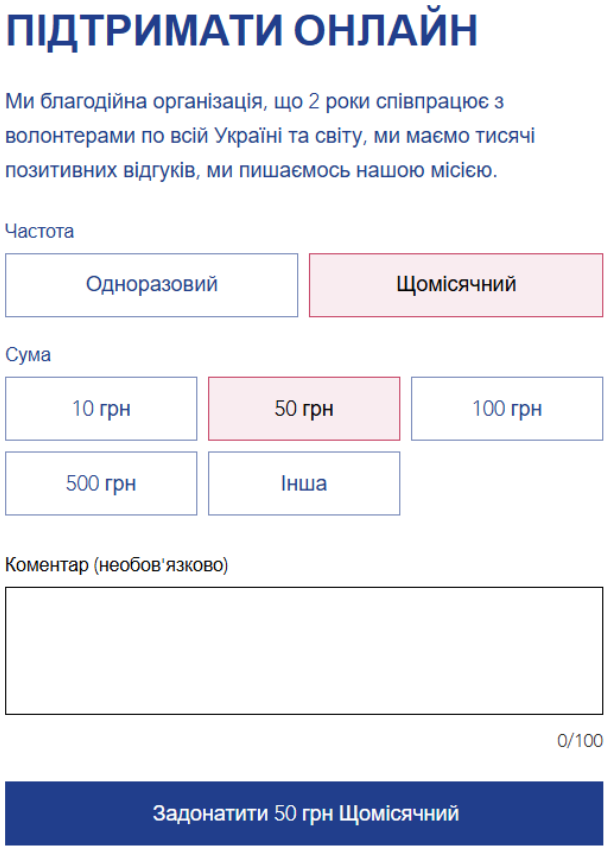


Рисунок 4.37 – Форма для підтримки благодійної діяльності

Після підтвердження дії система автоматично перенаправить користувача на сторінку оплати. На цій сторінці буде відображена вся необхідна інформація про донат, включаючи суму внеску, обрану ініціативу та її мету. Також відображена інформація про донора, з можливістю змінити контактні дані. Користувачам буде запропоновано ввести платіжну адресу для завершення транзакції. Сторінку оформлення платежу продемонстровано на рисунку 4.38.

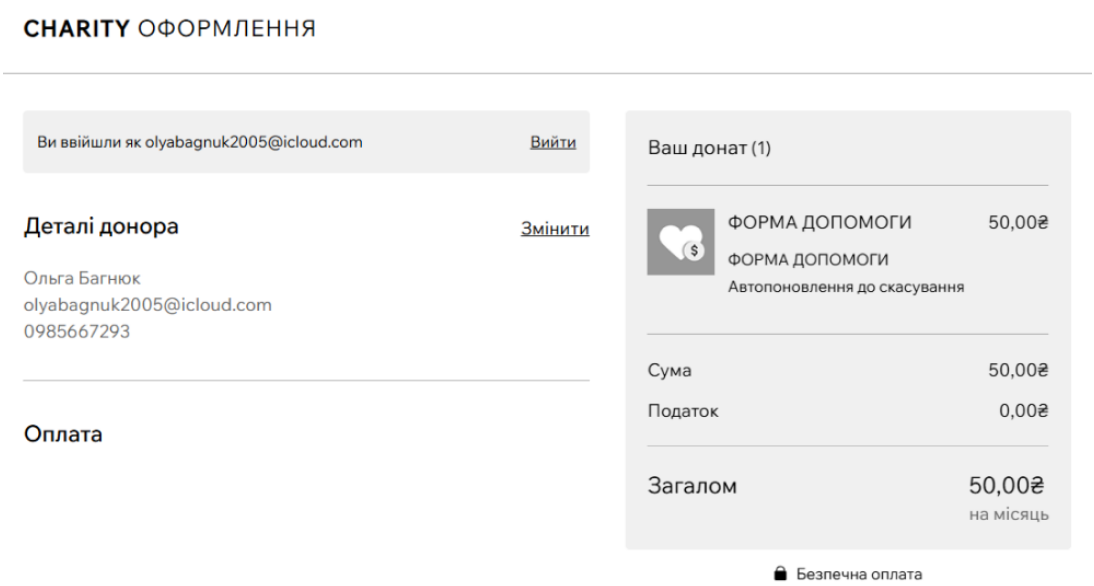


Рисунок 4.38 – Форма оформлення платежу

Також користувач має можливість звернутися до чат-бота, який відображений кнопкою в правому нижньому кутку сторінки. Натиснувши на неї, користувач може почати вводити повідомлення та отримати швидку відповідь на свої питання (див. рисунок 4.39).

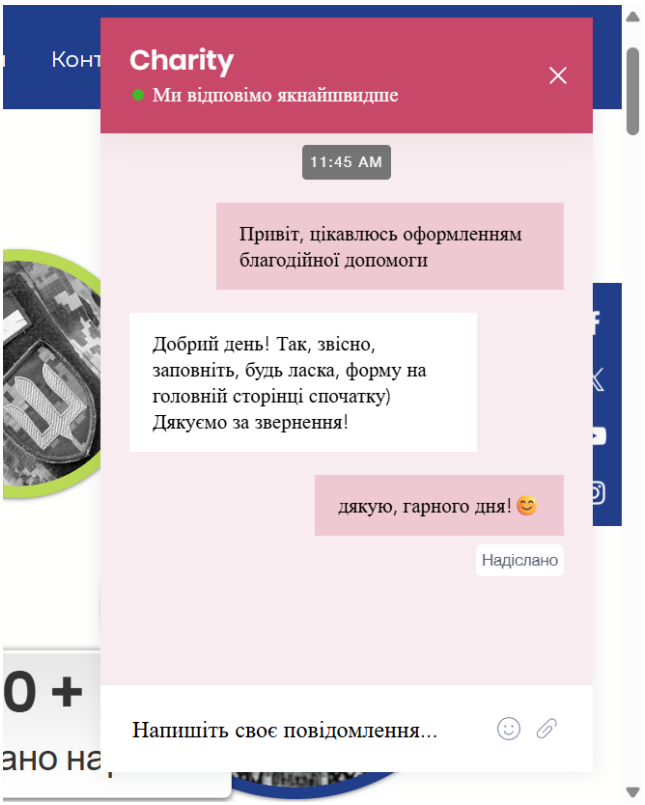


Рисунок 4.39 – Комунікація за допомогою чат-боту

Організатору-волонтеру на електронну пошту надходять відповіді та повідомлення від користувачів, що дозволяє оперативно реагувати на їхні запити та підтримувати зв’язок (див. рисунок 4.40).

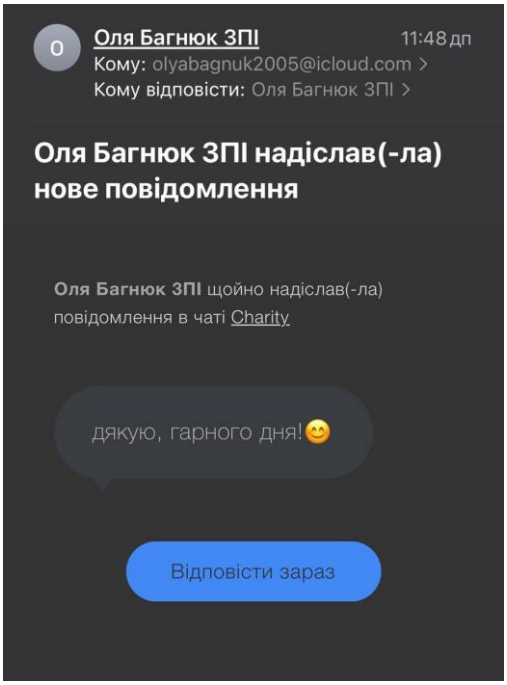


Рисунок 4.40 – Сповіщення про нове повідомлення у чат-боті

Для завершення роботи з вебзастосунком користувач може у будь-який момент натиснути кнопку «Вийти». Це дозволяє безпечно завершити сеанс, захистити особисті дані та запобігти несанкціонованому доступу до облікового запису (див. рисунок 4.41).

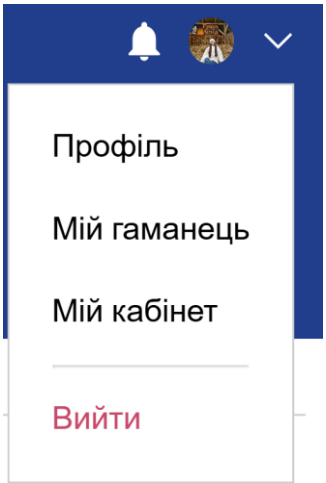


Рисунок 4.41 – Вихід з вебзастосунку

Рекомендується завжди виходити з системи після завершення роботи з вебзастосунком, особливо якщо використовується загальнодоступний або спільний пристрій. Це необхідно для забезпечення безпеки особистих даних та запобігання несанкціонованому доступу до облікового запису.

4.3 Висновки

У результаті тестування було підтверджено, що система для планування та координації благодійних заходів відповідає встановленим функціональним, безпековим та експлуатаційним вимогам. Були успішно перевірені ключові компоненти, такі як модулі реєстрації та авторизації користувачів, механізми обробки помилок введення даних, перевірка автентичності користувачів, правильність перенаправлення до особистого кабінету після входу, а також робота анкети для створення нової ініціативи, здійснення реєстрації на благодійні ініціативи та взаємодії зі зборами.

Особливу увагу було приділено тестуванню правильності заповнення обов'язкових полів форми, відправлення даних організаторам та перевірці надсилання відповідних сповіщень організаторам. Система показала високу стабільність, правильну обробку користувацьких даних та надійну взаємодію між модулями.

Для підвищення зручності користування та мінімізації можливих помилок при роботі з платформою також була розроблена й протестована інструкція користувача. Вона пояснює, як правильно працювати зі сторінками ініціатив, зборів, оплат та чат-ботом. Інструкція допомагає користувачам швидко зорієнтуватися у функціоналі системи: створити нову ініціативу, зареєструватись на існуючу, зробити благодійний внесок або переглянути статус поточних зборів тощо. Наявність чітких підказок і пояснень сприяє підвищенню доступності сервісу для користувачів з різним рівнем технічної підготовки.

Таким чином, проведене тестування та впровадження супровідних інструкцій забезпечують високу якість роботи системи.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було розроблено програмне забезпечення для планування та координації благодійних ініціатив. Реалізація проєкту відповідає методичним вказівкам [45] і базується на використанні сучасних вебтехнологій, зокрема JavaScript та React для фронтенд-частини, а також Node.js для серверної логіки.

На початковому етапі було проведено аналіз сучасних методів і засобів планування та координації благодійних ініціатив. Це дозволило виявити потреби у підвищенні ефективності взаємодії між користувачами, зручності інтерфейсу та автоматизації ключових процесів. Було проаналізовано популярні аналоги, зокрема «ДоброДій», «HelpKarma», «Зграя» та «Peremoga». Для глибшого розуміння функціональності було створено порівняльну характеристику аналогів. Аналіз показав, що власна розробка має вищий сумарний коефіцієнт функціональної повноти та зручності використання порівняно з існуючими рішеннями.

Було розроблено структуру програмного забезпечення, що включає діаграму варіантів використання, прототип головної сторінки вебзастосунку та сторінки «Збори коштів». Це дозволило забезпечити логічну побудову інтерфейсу та ефективний розподіл функціональних модулів.

Особлива увага приділялася дизайну інтерфейсу та принципам інклюзивності. Створено кольорову палітру, логотип платформи та повний дизайн у Figma. Проведено аналіз важливості інклюзивного підходу, і підтверджено, що сторінки розробленого вебзастосунку відповідають вимогам доступності.

Розроблено метод реєстрації на благодійний захід, що підвищує якість та продуктивність за рахунок автоматизації обробки даних, перевірки унікальності заявок, верифікації через email, оновлення особистого кабінету та інформування адміністратора у режимі реального часу.

Також реалізовано метод побудови соціальної активності на платформі в режимі реального часу. Його впровадження дозволяє користувачам підписуватись

на інші профілі, коментувати, отримувати сповіщення через WebSocket, що сприяє підвищенню залученості та продуктивній взаємодії між учасниками платформи.

Було створено модель системи у вигляді діаграми діяльності та діаграми класів, що відображає логіку взаємодії елементів програмного забезпечення та структуру даних.

Розроблено алгоритм авторизації користувача та входу в особистий кабінет, а також алгоритми користувацької взаємодії з благодійними зборами та алгоритм участі в благодійних ініціативах.

Реалізовано модуль одноразової оплати та оформлення щомісячної підписки для підтримки зборів, модуль авторизації та реєстрації користувачів із валідацією даних, а також модуль для взаємодії з благодійними зборами. До кожного модуля були розроблені алгоритми роботи, щоб краще зрозуміти внутрішню логіку їхньої реалізації, візуалізувати послідовність дій та оптимізувати структуру коду.

Проведено функціональне тестування системи, під час якого перевірено ключові сценарії взаємодії: реєстрація, авторизація, перегляд ініціатив, подання заявок, підписка, оплати, фільтрація, пошук, коментарі, лайки. Всі дії були протестовані на валідність та коректність роботи. Проведено порівняльне тестування власного вебзастосунку з аналогами за допомогою інструменту Lighthouse, і результати свідчать про вищий показник, у таких пунктах, як час до першої візуалізації, загальна швидкість завантаження, індекс швидкості та стабільність відображення контенту.

Наприкінці було підготовлено інструкцію користувача та описані системні вимоги. Інструкція містить опис кожного етапу взаємодії з платформою, що дозволяє легко орієнтуватися в системі як новачкам, так і досвідченим користувачам.

У результаті реалізації всіх етапів розробки було підтверджено підвищення якості та продуктивності розробленого програмного забезпечення. Це досягнуто завдяки впровадженню чітко структурованої архітектури, автоматизації ключових процесів, а також візуалізації алгоритмів, що сприяло кращому розумінню логіки роботи модулів і забезпечило їх узгодженість.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Волонтерство в Україні: океан можливостей чи море викликів? Інсайти круглого столу Української Волонтерської Служби [Електронний ресурс] – Режим доступу до ресурсу: https://volunteer.country/library/volonterstvo-v-ukrayini (дата звернення: 02.05.2025).

2. Багнюк О.В., Романюк О.В. «Розробка програмного забезпечення для планування та координації благодійних ініціатив» / Матеріали LIV Всеукраїнської науково-технічної конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2025): збірник доповідей [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/24024/19897.

3. Багнюк О.В. «Software development for planning and coordinating charitable initiatives» / Матеріали LIV Всеукраїнська науково-технічна конференція факультету будівництва, цивільної та екологічної інженерії (2025): збірник доповідей [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fbtegp/all-fbtegp 2025/paper/view/24067/19891.

4. Багнюк О.В., Романюк О.В. «Розробка методу реєстрації на благодійний захід» / Матеріали Міжнародної науково-практичної конференції «Молодь в науці: дослідження, проблеми, перспективи» (Вінниця, 2025 р.): збірник доповідей [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25467.

5. Багнюк О.В., Романюк О.В. «Інклюзивний дизайн та доступність в інтерфейсах користувача» / Матеріали IV Всеукраїнської науково-технічної конференції молодих вчених, аспірантів і студентів, Одеса, 26-27 вересня 2024 р. – Одеса, Видавництво ОНТУ, 2024 р. – 369 – 371с.

6. Автоматизація процесу обліку внесків та аналізу звітності з наданої допомоги у благодійних фондах [Електронний ресурс] – Режим доступу до ресурсу:

https://openarchive.nure.ua/entities/publication/3d226352-e65f-4ad6-8bda-8399300d2413 (дата звернення 02.05.2025).

7. Штучний інтелект та хмарні обчислення: симбіоз технологій майбутнього [Електронний ресурс] – Режим доступу до ресурсу: https://blog.colobridge.net/uk/2024/02/artificial-intelligence-and-cloud-computing-ua/ (дата звернення: 03.05.2025).

8. Тренди з кібербезпеки в благодійності у 2025 році [Електронний ресурс] – Режим доступу до ресурсу: https://gurt.org.ua/articles/105442/ (дата звернення: 03.05.2025).

9. З початку повномасштабної війни кількість благодійних організацій зросла майже вдвічі [Електронний ресурс] – Режим доступу до ресурсу: https://zmina.info/news/z-pochatku-povnomasshtabnoyi-vijny-kilkist-blagodijnyh-organizaczij-zrosla-majzhe-vdvichi/ (дата звернення: 03.05.2025).

10. В Україні майже 46 тисяч дітей, позбавлених батьківського піклування – Мінсоцполітики [Електронний ресурс] – Режим доступу до ресурсу: https://salo.li/0d03C84 (дата звернення: 04.05.2025).

11. Державне агентство лісових ресурсів України [Електронний ресурс] – Режим доступу до ресурсу: https://forest.gov.ua/napryamki-diyalnosti/lisove-gospodarstvo/ohorona-i-zahist-lisiv/ohorona-lisiv-vid-nezakonnih-rubok (дата звернення: 05.05.2025).

12. ДоброДій [Електронний ресурс] – Режим доступу до ресурсу: https://248.dp.ua/projects (дата звернення: 07.05.2025).

13. HelpKarma [Електронний ресурс] – Режим доступу до ресурсу: https://uk.helpkarma.com/ (дата звернення: 07.05.2025).

14. Зграя [Електронний ресурс] – Режим доступу до ресурсу: https://zgraya-help.com/ (дата звернення: 07.05.2025).

15. Peremoga [Електронний ресурс] – Режим доступу до ресурсу: https://www.peremoga-ukraine.org/ua/fundraising (дата звернення: 07.05.2025).

16. Клієнт-серверна архітектура [Електронний ресурс] – Режим доступу до ресурсу: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/ (дата звернення: 08.05.2025).

17. Подієво-орієнтоване програмування [Електронний ресурс] – Режим доступу до ресурсу: https://studfile.net/preview/9917360/page:4/ (дата звернення: 09.05.2025).

18. Варіанти використання та сценарії (Use Cases and Scenarios) [Електронний ресурс] – Режим доступу до ресурсу: https://www.maxzosim.com/use-cases-and-scenarios/ (дата звернення: 10.05.2025).

19. Мокап, Каркас і Прототип: що обрати вам? [Електронний ресурс] – Режим доступу до ресурсу: https://luxnet.io/uk/blog/mockups-vs-wireframes-vs-prototypes (дата звернення: 11.05.2025).

20. Mark Wells. User Experience Design: An Introduction to Creating Interactive Digital Spaces. London: Laurence King Publishing, 2023. – 160 p.

21. MAK. Cyrillic & Latin free font [Електронний ресурс] – Режим доступу до ресурсу: https://www.behance.net/gallery/89569065/MAK-Cyrillic-Latin-free-font (дата звернення: 12.05.2025).

22. UML для бізнес-моделювання: для чого потрібні діаграми процесів [Електронний ресурс] – Режим доступу до ресурсу: https://evergreens.com.ua/ua/articles/uml-diagrams.html (дата звернення: 13.05.2025).

23. MongoDB – нереляційна база даних [Електронний ресурс] – Режим доступу до ресурсу: http://energyfirefox.blogspot.com/2012/07/mongodb.html (дата звернення: 13.05.2025).

24. Коротко про API та його тестування [Електронний ресурс] – Режим доступу до ресурсу: https://qagroup.com.ua/publications/korotko-pro-ari-ta-jogo-testuvannia/ (дата звернення: 14.05.2025).

25. React. JavaScript-бібліотека для створення користувацьких інтерфейсів [Електронний ресурс] – Режим доступу до ресурсу: https://uk.legacy.reactjs.org/ (дата звернення: 15.05.2025).

26. Початок роботи з Node.js та Express.js [Електронний ресурс] – Режим доступу до ресурсу: https://javascript.org.ua/pochatok-roboti-z-node-js-ta-express-js-vash-pershij-server/ (дата звернення: 16.05.2025).

27. Introduction to JSON Web Tokens [Електронний ресурс] – Режим доступу до ресурсу: https://jwt.io/introduction (дата звернення: 17.05.2025).

28. Обробка помилок в Axios: детальний гайд для розробників [Електронний ресурс] – Режим доступу до ресурсу: https://markup-ua.com/obrobka-pomilok-v-axios-detalnij-gajd-dlya-rozrobnikiv/ (дата звернення: 17.05.2025).

29. React Native library for Stripe [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/stripe/stripe-react-native (дата звернення: 17.05.2025).

30. Що таке JavaScript та де він використовується [Електронний ресурс] – Режим доступу до ресурсу: https://dzudzylo.com/javascript/shho-take-javascript-ta-de-vin-vykorystovuyetsya.html (дата звернення: 18.05.2025).

31. Як стати PHP-розробником. План дій для початківців [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/lenta/articles/how-to-learn-php/ (дата звернення: 19.05.2025).

32. Що таке C#? Кому підходить програмування на Сі Шарп? [Електронний ресурс] – Режим доступу до ресурсу: https://beetroot.academy/blog/shcho-take-c-chi-pidhodit-meni-cya-mova-programuvannya-chomu-vona-kruta (дата звернення: 19.05.2025).

33. Що таке мова програмування Python? [Електронний ресурс] – Режим доступу до ресурсу: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-jazik-programmirovanija-python/ (дата звернення: 20.05.2025).

34. Що таке Visual Studio Code 2025 [Електронний ресурс] – Режим доступу до ресурсу: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-jazik-programmirovanija-python/ https://top-keis.com.ua/shho-take-visual-studio-code/ (дата звернення: 21.05.2025).

35. Для чого потрібен WebStorm [Електронний ресурс] – Режим доступу до ресурсу: https://alternativescience.net/programming/204-dlya-chogo-potriben-webstorm/ (дата звернення: 22.05.2025).

36. Brackets. Modern, Powerful & Open source [Електронний ресурс] – Режим доступу до ресурсу: https://brackets.io/ (дата звернення: 23.05.2025).

37. Sublime Text, ефективні шорткати та трюки [Електронний ресурс] – Режим доступу до ресурсу: https://hackyourmom.com/kibervijna/sublime-text-efektyvni-shortkaty-ta-tryuky/ (дата звернення: 23.05.2025).

38. Аутентифікація у Node.js використовуючи Passport.js [Електронний ресурс] – Режим доступу до ресурсу: https://devzone.org.ua/post/node-hero-chastyna-8-autentyfikatsiia-u-nodejs-vykorystovuiuchy-passportjs (дата звернення: 24.05.2025).

39. Базове розуміння Oauth 2.0 [Електронний ресурс] – Режим доступу до ресурсу: https://stfalcon.com/uk/blog/post/oauth-2.0 (дата звернення: 25.05.2025).

40. JSON – Introduction [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/js/js_json_intro.asp (дата звернення: 26.05.2025).

41. Building a React/Express Stripe Checkout Form [Електронний ресурс] – Режим доступу до ресурсу: https://dev.to/wra-sol/building-a-reactexpress-stripe-donation-form-2pek (дата звернення: 27.05.2025).

42. Що таке тестування ПЗ, його етапи, види, інструменти [Електронний ресурс] – Режим доступу до ресурсу: https://university.sigma.software/what-is-software-testing/ (дата звернення: 28.04.2025).

43. Функціональне тестування [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/funktsionalne-testuvannya/ (дата звернення: 30.05.2025).

44. Lighthouse [Електронний ресурс] – Режим доступу до ресурсу: https://developer.chrome.com/docs/lighthouse/overview (дата звернення: 31.05.2025).

45. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення» / Уклад. : О. Н. Романюк, Г. О. Черноволик. – Вінниця : ВНТУ, 2022. – 51 с.

ДОДАТКИ

115

Додаток А. Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

УЗГОДЖЕНО
Голова громадської організації «ЮС ГЕНТІУМ»
Вишківська А.О.
«25» березня 2025.

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«25» березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу
«Розробка програмного забезпечення для планування та координації благодійних ініціатив»
за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. Романюк О. В.
« 24 » березня 2025 р.

Виконала:
студент гр. ЗПІ-21б Багнюк О.В.
« 24 » березня 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка програмного забезпечення для планування та координації благодійних ініціатив».

Галузь застосування – інформаційні технології у сфері благодійних організацій та соціальних проєктів.

2. Підстава для розробки

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від 20 березня 2025 року про закріплення тем БКР.

3. Мета та призначення розробки

Метою роботи є підвищення продуктивності та рівня якості управління благодійними ініціативами за рахунок розробки програмного забезпечення, що автоматизують процеси планування, управління та координації ініціатив, зокрема через організацію подій та аналіз результатів.

Призначення роботи – методи та інструменти розробки програмного забезпечення для координації благодійних ініціатив, які забезпечують якісне управління та планування благодійних проєктів.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР:

1. Волонтерство в Україні: океан можливостей чи море викликів? Інсайти круглого столу Української Волонтерської Служби [Електронний ресурс] – Режим доступу до ресурсу: https://volunteer.country/library/volonterstvo-v-ukrayini (дата звернення: 02.05.2025).

2. Mark Wells. User Experience Design: An Introduction to Creating Interactive Digital Spaces. London: Laurence King Publishing, 2023. – 160 p.

3. Building a React/Express Stripe Checkout Form [Електронний ресурс] – Режим доступу до ресурсу: https://dev.to/wra-sol/building-a-reactexpress-stripe-donation-form-2pek (дата звернення: 27.05.2025).

4. Функціональне тестування [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/funktsionalne-testuvannya/ (дата звернення: 30.05.2025).

5. Технічні вимоги

Вихідні дані до роботи: тип роботи – вебзастосунок; модель розробки – ітеративна модель; засоби організації даних програми – фреймворки Node та React; вхідні дані: інформаційні дані про благодійні заходи, збори, користувачів та їхню активність; вихідні дані: підтвердження та інформування користувачів, відображення персоналізованої інформації, відображення списку благодійних ініціатив та зборів; середовище розробки – Visual Studio Code; мова розробки – JavaScript; операційна система – Windows 8/10/11.

6. Конструктивні вимоги

Вебзастосунок та вебклієнт повинні відповідати ергономічним та технічним вимогам. Текстова та графічна документація повинна відповідати стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

- пояснювальна записка до бакалаврської кваліфікаційної роботи;
- технічне завдання;
- лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської Кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану розвитку програм для планування та координації благодійних ініціатив	25.03.25 – 27.03.25
2	Розробка структури та дизайну вебзастосунку	28.03.25 – 30.03.25
3	Розробка методів, моделі та алгоритмів роботи програми	31.03.25 – 12.04.25
4	Аналіз та вибір засобів для реалізації програмного застосунку	13.04.25 – 18.04.25
3	Розробка програмного забезпечення	19.04.25 – 12.05.24
4	Тестування програмного застосунку	13.05.24 – 17.05.24
5	Оформлення матеріалів до захисту БКР	20.05.24 – 30.05.24

10. Порядок контролю та прийняття.

Всі етапи бакалаврської кваліфікаційної роботи контролюються науковим керівником згідно плану по виконанню роботи.

Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

119

Додаток Б. Протокол перевірки бакалаврської кваліфікаційної роботи на наявність текстових запозичень

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка програмного забезпечення для планування та координації благодійних ініціатив

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ: кафедра програмного забезпечення, ФІТКІ, група ЗПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 6,4 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне).

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту.

□ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

□ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Романюк О.В.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Багнюк О.В.
(підпис) (прізвище, ініціали)

120

Додаток В. Акт впровадження на підприємстві

ЗГОДЖЕНО
Голова громадської організації «ЮС ГЕНТІУМ»
А.О. Івашківська
«02» 06 2025 року

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор О.Н. Романюк
«02» 06 2025

АКТ ВПРОВАДЖЕННЯ № 97
результатів науково-дослідних робіт

Замовник громадська організація «ЮС ГЕНТІУМ»
(найменування організації)

Цим актом підтверджується, що результати роботи – «Розробка програмного забезпечення для планування та координації благодійних ініціатив»,
(найменування теми)

що виконала студентка гр. ЗПІ-21б ВНТУ Багнюк О.В.,
(виконавець)

за договором про творчу співдружність без взаємних грошових розрахунків, на громадських засадах відповідно до теми затвердженої наказом № 97 від «20» березня 2025 р.

виконана з 25.03.25 по 2.06.25,
(терміни виконання)

Впроваджено у громадській організації «ЮС ГЕНТІУМ»
(найменування організації, де здійснювалось впровадження)

1. Вид впроваджених результатів: експлуатація програмного забезпечення
(експлуатація вибору, роботи, технології)
2. Характеристика масштабу впровадження: одиничне
(унікальне, одиничне, партія, масове, серійне)
3. Форма впровадження: дослідний зразок
4. Новизна результатів науково-дослідної роботи: модернізація старих розробок
(піонерські, принципово нові, якісно нові, модифікації, модернізація старих розробок)
5. Впроваджені: в системі для планування та координації благодійних ініціатив
6. Соціальний та науково-технічний ефект: спеціальне призначення
(охорона навколишнього середовища, поліпшення й оздоровлення умов праці, удосконалення структури керування, науково-технічних напрямків, спеціальне призначення)

Від виконавця:
студентка групи ЗПІ-21б
О.В. Багнюк

Від громадської організації «ЮС ГЕНТІУМ»
голова організації
А.О. Івашківська

Керівник: к.т.н., доцент кафедри ПЗ
О.В. Романюк

Додаток Г. Лістинг програми

Main.js

```
import PaymentForm from './components/PaymentForm';
import SubscriptionForm from './components/SubscriptionForm';

function myMenuFunction() {
    const menu = document.getElementById("navMenu");
    menu.classList.toggle("responsive");
}

// Перемикання форм входу/реєстрації
const loginBtn = document.getElementById("loginBtn");
const registerBtn = document.getElementById("registerBtn");
const loginForm = document.getElementById("login");
const registerForm = document.getElementById("register");

function login() {
    loginForm.style.left = "4px";
    registerForm.style.right = "-520px";
    loginBtn.classList.add("white-btn");
    registerBtn.classList.remove("white-btn");
    loginForm.style.opacity = 1;
    registerForm.style.opacity = 0;
}

function register() {
    loginForm.style.left = "-510px";
    registerForm.style.right = "5px";
    loginBtn.classList.remove("white-btn");
    registerBtn.classList.add("white-btn");
    loginForm.style.opacity = 0;
    registerForm.style.opacity = 1;
}
```

```
// DOM Ready
document.addEventListener("DOMContentLoaded", function () {
    const registerSubmit = document.querySelector(".register-container .submit");
    const loginSubmit = document.querySelector(".login-container .submit");
    const logoutBtn = document.getElementById("logoutBtn");

    if (registerSubmit) registerSubmit.addEventListener("click", registerUser);
    if (loginSubmit) loginSubmit.addEventListener("click", loginUser);
    if (logoutBtn) logoutBtn.addEventListener("click", logoutUser);

    loadUserInfo();
    highlightActiveLink();
});

// реєстрація користувача

function isValidEmail(email) {
    // Перевірка формату email через регулярний вираз
    return /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email);
}

function isStrongPassword(password) {
    // Мінімум 8 символів, хоча б одна літера та одна цифра
    return /^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d]{8,}$/.test(password);
}

async function registerUser(event) {
    event.preventDefault();

    const firstName = document.querySelector(".register-container [placeholder='Firstname']").value.trim();
    const lastName = document.querySelector(".register-container [placeholder='Lastname']").value.trim();
    const email = document.querySelector(".register-container [placeholder='Email']").value.trim();
```

```
        const password = document.querySelector(".register-container [placeholder='Password']").value;

        if (!firstName || !lastName || !email || !password) {
            return alert("Будь ласка, заповніть всі поля!");
        }

        if (!isValidEmail(email)) {
            return alert("Некоректний формат електронної пошти.");
        }

        if (!isStrongPassword(password)) {
            return alert("Пароль має містити щонайменше 8 символів, включаючи літери та цифри.");
        }

        try {
            const response = await fetch('/register', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({ firstName, lastName, email, password })
            });

            const result = await response.json();

            if (response.ok && result.user) {
                alert(result.message || "Реєстрація успішна!");
                localStorage.setItem('user', JSON.stringify(result.user));
                window.location.href = 'account1.html';
            } else {
                alert(result.message || "Сталася помилка під час реєстрації");
            }
        } catch (error) {
            alert("Помилка з'єднання з сервером: " + error.message);
        }
```

```
}
// функція входу користувача

function isValidEmail(email) {
   return /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email);
}

function isStrongEnough(password) {
   return password.length >= 6; // мінімум 6 символів, можна посилити
}

async function loginUser(event) {
   event.preventDefault();

   const email = document.querySelector(".login-container [placeholder='Username or Email']").value.trim();
   const password = document.querySelector(".login-container [placeholder='Password']").value;

   if (!email || !password) {
      return alert("Будь ласка, введіть email і пароль");
   }

   if (!isValidEmail(email)) {
      return alert("Некоректний формат електронної пошти");
   }

   if (!isStrongEnough(password)) {
      return alert("Пароль має містити щонайменше 6 символів");
   }

   try {
      const response = await fetch('/login', {
         method: 'POST',
         headers: { 'Content-Type': 'application/json' },
```

```
            body: JSON.stringify({ email, password })
        });
        const result = await response.json();

        if (response.ok && result.user) {
            alert(result.message || "Вхід успішний!");
            localStorage.setItem('user', JSON.stringify(result.user));
            window.location.href = 'account1.html';
        } else {
            alert(result.message || "Невірні дані для входу");
        }
    } catch (error) {
        alert("Помилка з'єднання з сервером: " + error.message);
    }
}

// обробник кнопки Google входу
document.querySelector('.google-login-btn')?.addEventListener('click', function () {
    window.location.href = '/auth/google';
});

// обробник кнопки Facebook входу
document.querySelector('.facebook-login-btn')?.addEventListener('click', function () {
    window.location.href = '/auth/facebook';
});

//  Завантаження даних користувача на account1.html
function loadUserInfo() {
    const user = JSON.parse(localStorage.getItem('user'));
    const userInfoBlock = document.getElementById("userInfo");

    if (user && userInfoBlock) {
        userInfoBlock.innerHTML = `
            <p><strong>Ім’я:</strong> ${user.firstName}</p>
            <p><strong>Прізвище:</strong> ${user.lastName}</p>
```

```
            <p><strong>Email:</strong> ${user.email}</p>
        `;
    }
}

function highlightActiveLink() {
    const currentPath = window.location.pathname;
    document.querySelectorAll('.link').forEach(link => link.classList.remove('active'));

    if (currentPath.includes('index.html')) {
        document.getElementById('homeLink')?.classList.add('active');
    } else if (currentPath.includes('services.html')) {
        document.getElementById('servicesLink')?.classList.add('active');
    } else if (currentPath.includes('about.html')) {
        document.getElementById('aboutLink')?.classList.add('active');
    } else if (currentPath.includes('account1.html')) {
        document.getElementById('accountLink')?.classList.add('active');
    }
}

//  Вихід з акаунта
function logoutUser() {
    localStorage.removeItem('user');
    alert("Ви вийшли з системи!");
    window.location.href = 'index.html';
}

const items = [
    "Про нас",
    "Підтримати",
    "Збори",
    "Ініціативи",
    "Контакти",
    "Вхід",
    "Особистий кабінет"
```

```
];

function searchContent() {
    const input = document.getElementById("searchInput").value.toLowerCase();
    const links = document.querySelectorAll(".nav-menu ul li");

    links.forEach(link => {
        const text = link.textContent.toLowerCase();
        link.style.display = text.includes(input) ? "list-item" : "none";
    });
}

const pages = [
    { title: "Онкохворі дітия", url: "children-cancer.html" },
    { title: "ЗСУ", url: "army.html" },
    { title: "Постраждалі від війни", url: "war-victims.html" },
    { title: "Діти-сироти", url: "orphan-children.html" },
   { title: "Притулки для тварин", url: "animalshelters.html" },
    { title: "Контакти", url: "contacts.html" },
    { title: "Підтримати", url: "support.html" }
];

// Функція для автоматичного пошуку схожих слів без необхідності прописувати вручну
function showSuggestions() {
    const input = document.getElementById("searchInput").value.toLowerCase();
    const suggestionBox = document.getElementById("suggestions");
    suggestionBox.innerHTML = "";

    if (input === "") {
        return;
    }

    const filtered = pages.filter(page => {
        const words = page.title.toLowerCase().split(' ');
        return words.some(word => word.includes(input));
```

```
    });

    // Якщо є підказки
    if (filtered.length > 0) {
        filtered.forEach(page => {
            const suggestion = document.createElement("div");
            suggestion.textContent = page.title;
            suggestion.onclick = () => {
                window.location.href = page.url;
            };
            suggestionBox.appendChild(suggestion);
        });
    } else {
        // Якщо не знайдено, виводимо повідомлення про відсутність результатів
        const noResults = document.createElement("div");
        noResults.textContent = "Нічого не знайдено";
        suggestionBox.appendChild(noResults);
    }
}
```

PaymentForm.js

```
import React, { useState } from 'react';
import { CardElement, useStripe, useElements } from '@stripe/react-stripe-js';

const PaymentForm = () => {
    const stripe = useStripe();
    const elements = useElements();
    const [amount, setAmount] = useState(100);

    const handleSubmit = async (e) => {
        e.preventDefault();

        const res = await fetch('http://localhost:5000/create-payment-intent', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
```

```
            body: JSON.stringify({ amount }),
        });

        const { clientSecret } = await res.json();

        const result = await stripe.confirmCardPayment(clientSecret, {
            payment_method: {
                card: elements.getElement(CardElement),
            },
        });

        if (result.error) {
            console.error(result.error.message);
        } else {
            alert('Оплату здійснено успішно!');
        }
    };

    return (
        <form onSubmit={handleSubmit}>
            <CardElement />
            <button type="submit" disabled={!stripe}>Підтримати {amount} грн</button>
        </form>
    );
};

export default PaymentForm;
```

SubscriptionForm.js

```
import React, { useState } from 'react';
import { CardElement, useStripe, useElements } from '@stripe/react-stripe-js';

const SubscriptionForm = () => {
    const stripe = useStripe();
    const elements = useElements();
```

```
    const [email, setEmail] = useState('');

    const handleSubscribe = async (e) => {
      e.preventDefault();

      const paymentMethod = await stripe.createPaymentMethod({
        type: 'card',
        card: elements.getElement(CardElement),
      });

      const res = await fetch('http://localhost:5000/create-subscription', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          email,
          paymentMethodId: paymentMethod.paymentMethod.id,
          priceId: 'price_1Np4hrJMzKqsXXXXXXX',
        }),
      });

      const data = await res.json();

      const confirm = await stripe.confirmCardPayment(data.clientSecret);

      if (confirm.error) {
        console.error(confirm.error.message);
      } else {
        alert('Підписка оформлена успішно!');
      }
    };
    return (
      <form onSubmit={handleSubscribe}>
        <input type="email" placeholder="Ваша пошта" onChange={(e) =>
setEmail(e.target.value)} required />
        <CardElement />
```

```
        <button type="submit" disabled={!stripe}>Оформити підписку</button>
      </form>
    );
};
export default SubscriptionForm;
```

Server.js

```
const express = require('express');
const fs = require('fs');
const path = require('path');
const bodyParser = require('body-parser');
const cors = require('cors');
const Stripe = require('stripe');
require('dotenv').config();

const app = express();
const PORT = 3000;
const stripe = Stripe(process.env.STRIPE_SECRET_KEY);
const USERS_FILE = path.join(__dirname, 'users.json');

// Middleware
app.use(cors());
app.use(bodyParser.json());
app.use(express.json());

app.use(express.static(path.join(__dirname, 'public')));
const readUsers = () => {
  if (!fs.existsSync(USERS_FILE)) {
    return [];
  }
  return JSON.parse(fs.readFileSync(USERS_FILE, 'utf8'));
};

const writeUsers = (users) => {
  fs.writeFileSync(USERS_FILE, JSON.stringify(users, null, 2), 'utf8');
```

```
};

app.post('/register', (req, res) => {
  const { firstName, lastName, email, password } = req.body;

  if (!firstName || !lastName || !email || !password) {
    return res.status(400).json({ message: 'Будь ласка, заповніть всі поля!' });
  }

  let users = readUsers();
  let existingUser = users.find(user => user.email === email);

  if (existingUser) {
    return res.status(400).json({ message: 'Користувач вже існує!' });
  }

  users.push({ firstName, lastName, email, password });
  writeUsers(users);

  res.status(201).json({ message: 'Реєстрація успішна!' });
});

app.post('/login', (req, res) => {
  const { email, password } = req.body;

  let users = readUsers();
  let foundUser = users.find(user => user.email === email && user.password === password);

  if (foundUser) {
    const { firstName, lastName, email } = foundUser;
    res.status(200).json({
      message: `Вітаємо, ${firstName}! Вхід успішний.`,
      user: { firstName, lastName, email }
    });
  } else {
```

```
    res.status(400).json({ message: 'Невірний email або пароль!' });
  }
});

// Створення одноразового платежу
app.post('/create-payment-intent', async (req, res) => {
  const { amount } = req.body;

  try {
    const paymentIntent = await stripe.paymentIntents.create({
      amount: amount * 100,
      currency: 'uah',
      payment_method_types: ['card'],
    });

    res.send({ clientSecret: paymentIntent.client_secret });
  } catch (error) {
    res.status(400).send({ error: error.message });
  }
});
app.post('/create-subscription', async (req, res) => {
  const { email, paymentMethodId, priceId } = req.body;

  try {
    const customer = await stripe.customers.create({
      email,
      payment_method: paymentMethodId,
      invoice_settings: {
        default_payment_method: paymentMethodId,
      },
    });

    const subscription = await stripe.subscriptions.create({
      customer: customer.id,
      items: [{ price: priceId }],
```

```
      expand: ['latest_invoice.payment_intent'],
    });

    res.send({
      subscriptionId: subscription.id,
      clientSecret: subscription.latest_invoice.payment_intent.client_secret,
    });
  } catch (error) {
    res.status(400).send({ error: error.message });
  }
});

// Запуск сервера
app.listen(PORT, () => {
  console.log(`Сервер працює на http://localhost:${PORT}`);
});

const bcrypt = require('bcrypt');
const hashedPassword = await bcrypt.hash(password, 10); // хешування

const newUser = new User({
  firstName,
  lastName,
  email,
  password: hashedPassword
});

await newUser.save();

res.status(201).json({
  message: 'Реєстрація успішна!',
  user: { firstName, lastName, email }
});
```

AdminChatAPI.js

```
export const getMessages = async () => {
   const res = await fetch('/api/chat/messages');
   return await res.json();
};

export const sendReply = async (email, text) => {
   await fetch('/api/chat/reply', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ email, text })
   });
};
```

AdminChatPage.jsx

```
import React, { useState, useEffect } from 'react';
import { getMessages, sendReply } from './adminChatAPI';

export default function AdminChatPage() {
  const [messages, setMessages] = useState([]);
  const [replyText, setReplyText] = useState('');
  const [selectedEmail, setSelectedEmail] = useState('');

  const fetch = async () => {
    const msgs = await getMessages();
    setMessages(msgs);
  };

  useEffect(() => {
    fetch();
  }, []);
  const handleReply = async () => {
    await sendReply(selectedEmail, replyText);
    setReplyText('');
    fetch();
  };
```

```
  const users = [...new Set(messages.map(m => m.email))];

  return (
    <div>
      <h2>Чат з користувачами</h2>
      <select onChange={e => setSelectedEmail(e.target.value)}>
        <option>Оберіть користувача</option>
        {users.map(email => <option key={email}>{email}</option>)}
      </select>
      <div className="chat-thread">
        {messages
          .filter(msg => msg.email === selectedEmail)
          .map((msg, i) => (
            <div key={i} style={{ textAlign: msg.sender === 'admin' ? 'right' : 'left' }}>
              <p><b>{msg.sender}:</b> {msg.text}</p>
            </div>
          ))}
      </div>
      <input value={replyText} onChange={e => setReplyText(e.target.value)}
placeholder="Ваша відповідь..." />
      <button onClick={handleReply}>Відповісти</button>
    </div>
  );
}
```

chatRoutes.js

```
const express = require('express');
const router = express.Router();
const Message = require('../models/message.js');
const nodemailer = require('nodemailer');

// Налаштування поштового транспорту
const transporter = nodemailer.createTransport({
  service: 'gmail',
```

```
  auth: {
    user: 'olabagnuk94@gmail.com,
    pass: 'olya0801mqweryp',
  }
});

router.post('/reply', async (req, res) => {
  const { email, text } = req.body;

  const reply = new Message({ sender: 'admin', email, text, answered: true });
  await reply.save();

  try {
    await transporter.sendMail({
      from: 'youradminemail@gmail.com',
      to: email,
      subject: 'Відповідь від адміністрації',
      text: text
    });
    res.status(201).json({ message: 'Відповідь надіслана й відправлена на пошту' });
  } catch (err) {
    res.status(500).json({ message: 'Повідомлення збережено, але не вдалося надіслати лист',
error: err });
  }
});

module.exports = router;
```

AuthContext.jsx

```
import React, { createContext, useState, useEffect } from 'react';
import axios from 'axios';

export const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
```

```
  const [token, setToken] = useState(localStorage.getItem('token'));

  useEffect(() => {
    if (token) {
      axios.defaults.headers.common['Authorization'] = `Bearer ${token}`;
    }
  }, [token]);

  const login = (data) => {
    localStorage.setItem('token', data.token);
    setToken(data.token);
    setUser(data.user);
  };

  const logout = () => {
    localStorage.removeItem('token');
    setToken(null);
    setUser(null);
  };

  return (
    <AuthContext.Provider value={{ user, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
};
```

InitiativeFormModal.jsx

```
import { useState } from 'react'
import axios from 'axios'
import CalendarIntegration from './CalendarIntegration'

export default function InitiativeFormModal({ onClose, initiative }) {
    const [formData, setFormData] = useState({ name: '', surname: '', email: '' })
    const [success, setSuccess] = useState(false)
```

```
    const handleChange = e => setFormData({ ...formData, [e.target.name]: e.target.value })

    const handleSubmit = async () => {
      await axios.post('/api/initiatives/register', {
        ...formData,
        initiativeId: initiative._id
      })
      setSuccess(true)
    }

    return (
      <div className="fixed inset-0 bg-black bg-opacity-50 flex justify-center items-center">
        <div className="bg-white p-6 rounded shadow w-[400px]">
          {success ? (
            <>
              <p>Дякуємо! Ми надіслали лист з усією інформацією </p>
              <CalendarIntegration event={initiative} />
              <button onClick={onClose} className="mt-4 btn">Закрити</button>
            </>
          ) : (
            <>
              <h3 className="text-lg font-bold">Реєстрація на ініціативу</h3>
              <input name="name" placeholder="Ім’я" onChange={handleChange}
className="input" />
              <input name="surname" placeholder="Прізвище" onChange={handleChange}
className="input" />
              <input name="email" placeholder="Email" onChange={handleChange}
className="input" />
              <button onClick={handleSubmit} className="btn mt-2">Надіслати</button>
            </>
          )}
        </div>
      </div>
    )
  }
```

FundraisersDatailsPage.jsx

```
import React, { useEffect, useState } from 'react';
import { useParams } from 'react-router-dom';
import api from '../utils/api';
import FundraiserRating from '../components/Fundraisers/FundraiserRating';
import FundraiserComments from '../components/Fundraisers/FundraiserComments';
import DonationModal from '../components/Fundraisers/DonationModal';

export default function FundraiserDetailsPage() {
  const { id } = useParams();
  const [fundraiser, setFundraiser] = useState(null);
  const [liked, setLiked] = useState(false);
  const [likesCount, setLikesCount] = useState(0);

  useEffect(() => {
    api.get(`/api/fundraisers/${id}`).then(res => {
      setFundraiser(res.data);
      setLikesCount(res.data.likes || 0);
    });
  }, [id]);

  const handleLike = () => {
    if (!liked) {
      setLikesCount(likesCount + 1);
      setLiked(true);
      // api.post(`/api/fundraisers/${id}/like`);
    } else {
      setLikesCount(likesCount - 1);
      setLiked(false);
      // api.post(`/api/fundraisers/${id}/unlike`);
    }
  };

  if (!fundraiser) return <p>Завантаження...</p>;
```

```
  return (
    <div>
      <h1>{fundraiser.title}</h1>
      <p>Автор: {fundraiser.author.name}</p>
      <p>{fundraiser.description}</p>
      <p>Зібрано: {fundraiser.currentAmount} / {fundraiser.goalAmount}</p>
      <p>Перегляди: {fundraiser.views}</p>

      <button onClick={handleLike} style={{ marginTop: '10px' }}>
        {liked ? ' Лайкнуто' : ' Лайк'} ({likesCount})
      </button>

      <FundraiserRating fundraiserId={fundraiser._id} />
      <DonationModal fundraiser={fundraiser} />
      <FundraiserComments fundraiserId={fundraiser._id} />
    </div>
  );
}
```

StripeCheckoutForm.jsx

```
import React from 'react';
import { useStripe, useElements, CardElement } from '@stripe/react-stripe-js';

export default function StripeCheckoutForm({ clientSecret, onSuccess }) {
  const stripe = useStripe();
  const elements = useElements();

  const handleSubmit = async e => {
    e.preventDefault();
    const result = await stripe.confirmCardPayment(clientSecret, {
      payment_method: { card: elements.getElement(CardElement) },
    });
    if (result.paymentIntent.status === 'succeeded') onSuccess();
  };
```

```
  return (
    <form onSubmit={handleSubmit}>
      <CardElement />
      <button type="submit">Задонатити</button>
    </form>
  );
}
```

Passport.js

```
const GoogleStrategy = require('passport-google-oauth20').Strategy;
const FacebookStrategy = require('passport-facebook').Strategy;
const User = require('../models/User');

module.exports = function(passport) {
  passport.use(new GoogleStrategy({
    clientID: process.env.GOOGLE_CLIENT_ID,
    clientSecret: process.env.GOOGLE_CLIENT_SECRET,
    callbackURL: '/auth/google/callback',
  },
  async (accessToken, refreshToken, profile, done) => {
    const email = profile.emails[0].value;
    try {
      let user = await User.findOne({ email });
      if (!user) {
        user = await User.create({
          googleId: profile.id,
          email,
          name: profile.displayName,
          joinedDate: new Date(),
        });
      }
      done(null, user);
    } catch (err) {
      done(err, null);
```

```
  }
}));

passport.use(new FacebookStrategy({
  clientID: process.env.FACEBOOK_APP_ID,
  clientSecret: process.env.FACEBOOK_APP_SECRET,
  callbackURL: '/auth/facebook/callback',
  profileFields: ['id', 'emails', 'name', 'displayName']
},
async (accessToken, refreshToken, profile, done) => {
   const email = profile.emails && profile.emails[0] ? profile.emails[0].value : null;
  try {
    let user;
    if (email) {
      user = await User.findOne({ email });
    }
    if (!user) {
      user = await User.create({
        facebookId: profile.id,
        email,
        name: profile.displayName || `${profile.name.givenName} ${profile.name.familyName}`,
        joinedDate: new Date(),
      });
    }
    done(null, user);
  } catch (err) {
    done(err, null);
  }
}));

passport.serializeUser((user, done) => {
  done(null, user.id);
});
```

```
 passport.deserializeUser(async (id, done) => {
  try {
   const user = await User.findById(id);
   done(null, user);
  } catch (err) {
   done(err, null);
  }
 });
};
```

RegistrationForm.jsx

```
import React from 'react';
import { useFormik } from 'formik';
import * as Yup from 'yup';
import { useRegistration } from './useRegistration';

export default function RegistrationForm({ eventId }) {
 const { handleSubmitRegistration, loading, error } = useRegistration();

 const formik = useFormik({
  initialValues: { name: '', surname: '', email: '', eventId },
  validationSchema: Yup.object({
   name: Yup.string().required(),
   surname: Yup.string().required(),
   email: Yup.string().email().required()
  }),
  onSubmit: (values) => {
   handleSubmitRegistration(values, () => alert('Успішно зареєстровано!'));
  }
 });

 return (
  <form onSubmit={formik.handleSubmit}>
   <input name="name" onChange={formik.handleChange} value={formik.values.name} />
```

```
        <input name="surname" onChange={formik.handleChange} value={formik.values.surname} />
        <input name="email" onChange={formik.handleChange} value={formik.values.email} />
        <button type="submit" disabled={loading}>Зареєструватись</button>
        {loading && <div>Завантаження...</div>}
        {error && <div>{error}</div>}
      </form>
    );
  }
```

useSocket.js

```
import { useEffect, useRef, useState } from 'react';
import { io } from 'socket.io-client';

export function useSocket(userId, isAdmin = false) {
  const socketRef = useRef(null);
  const [notifications, setNotifications] = useState([]);

  useEffect(() => {
    socketRef.current = io('http://localhost:4000');

    socketRef.current.emit('initializeSocketConnection', { userId, isAdmin });

    socketRef.current.on('newSubscriptionNotification', (data) => {
      setNotifications((prev) => [...prev, { type: 'subscription', data }]);
    });

    socketRef.current.on('newCommentNotification', (data) => {
      setNotifications((prev) => [...prev, { type: 'comment', data }]);
    });

    socketRef.current.on('adminResponseNotification', (data) => {
      setNotifications((prev) => [...prev, { type: 'adminResponse', data }]);
    });
```

```
    return () => {
      socketRef.current.disconnect();
    };
  }, [userId, isAdmin]);

  const sendMessageToAdmin = (message) => {
    socketRef.current.emit('sendMessageToAdmin', message);
  };

  return { notifications, sendMessageToAdmin };
}
```

Додаток Д. Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПЛАНУВАННЯ ТА КООРДИНАЦІЇ БЛАГОДІЙНИХ ІНІЦІАТИВ

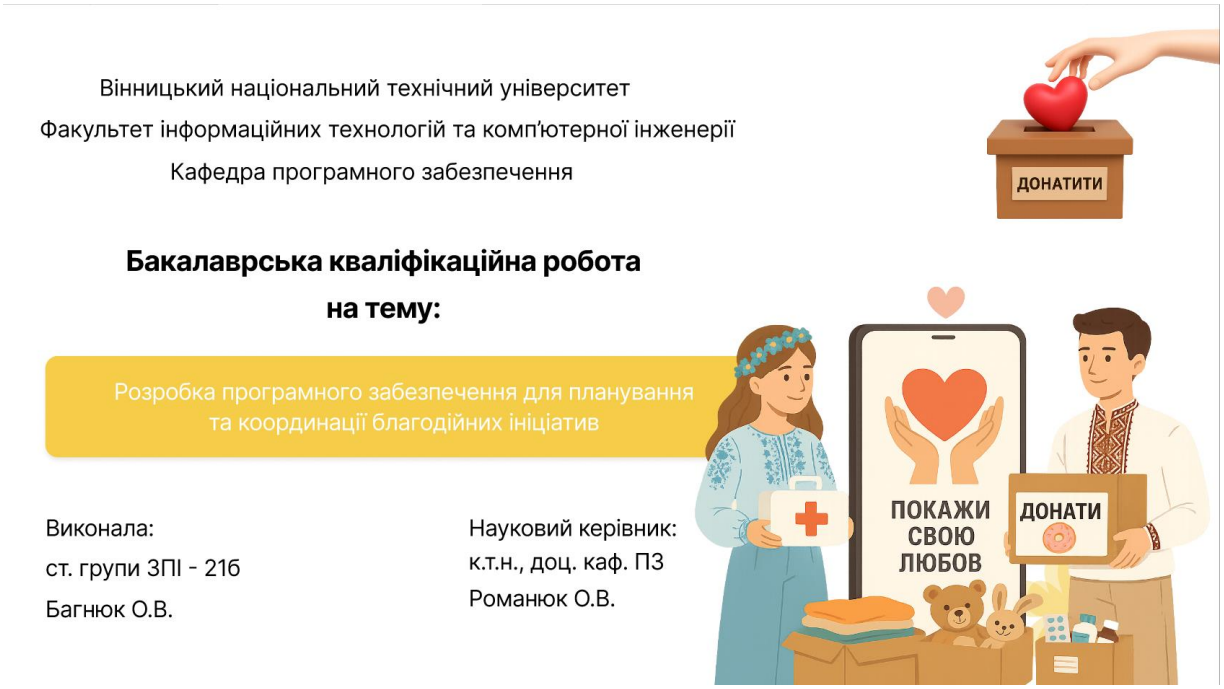


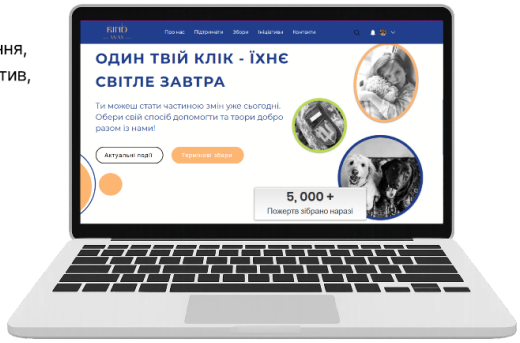
Рисунок Д.1 – Титульний слайд

Рисунок Д.2 – Актуальність теми

02

Мета та завдання дослідження

Метою роботи є підвищення продуктивності та рівня якості управління благодійними ініціативами за рахунок розробки програмного забезпечення, що автоматизують процеси планування, управління та координації ініціатив, зокрема через організацію подій та аналіз результатів.



Об'єкт дослідження

Це процес управління благодійними ініціативами з використанням сучасних інформаційних технологій.

Предмет дослідження

Це методи та засоби розробки програмного забезпечення для координації благодійних ініціатив, які забезпечують якісне управління та планування благодійних проєктів.

Рисунок Д.3 – Мета та завдання, об’єкт, предмет дослідження

03 ОСНОВНІ ЗАДАЧІ ДОСЛІДЖЕННЯ

Основними задачами дослідження є:
– аналіз існуючих методів і засобів планування та координації благодійних ініціатив для визначення напрямків підвищення їх рівня якості та продуктивності;
– розробка структури програмного забезпечення;
– розробка дизайну інтерфейсу та аналіз принципів інклюзивності;
– розробка методу реєстрації на благодійний захід;
– розробка методу побудови соціальної активності на платформі в реальному часі;
– розробка моделі системи;
– розробка алгоритму авторизації користувача та входу в особистий кабінет;
– розробка алгоритму користувацької взаємодії з благодійними зборами;
– розробка алгоритму вибору та участі в благодійних інціативах;
– розробка програмного забезпечення модуля одноразової оплати та оформлення щомісячної підписки для підтримки благодійних зборів;
– розробка програмного забезпечення модуля авторизації та реєстрації користувачів з валідацією даних, модуля для взаємодії з благодійними зборами;
– тестування розробленого рішення з оцінкою його функціональної повноти, стабільності роботи, зручності інтерфейсу та ефективності взаємодії між учасниками платформи (волонтерами та користувачами);
– розробка інструкції користувача та визначення системних вимог.

Рисунок Д.4 – Основні задачі дослідження

04 НАУКОВА НОВИЗНА ОТРИМАНИХ РЕЗУЛЬТАТІВ

01

1. Удосконалено метод реєстрації на благодійний захід, який на відміну від відомих аналогів включає повний цикл інтегрованої взаємодії між клієнтською та серверною частинами вебзастосунку з багаторівневою валідацією, підтвердженням участі через email із використанням бібліотеки nodemailer та унікального токена, а також механізм інформування адміністратора про нові заявки, що дало можливість підвищити рівень якості та продуктивності шляхом забезпечення безпеки, масштабованості та зручності обробки даних учасників в режимі реального часу.

02

2. Удосконалено метод побудови соціальної активності на платформі в реальному часі, який на відміну від відомих аналогів поєднує використання WebSocket-технології (на базі Socket.IO) з функціоналом підписок, коментування та інтерактивної модерації через спеціалізовані компоненти React і хук useSocket, що дало можливість підвищити рівень якості та продуктивності шляхом забезпечення миттєвого обміну даними, гнучкої взаємодії користувачів і оперативної реакції адміністрації.

Рисунок Д.5 – Наукова новизна отриманих результатів

05 ПОРІВНЯЛЬНИЙ АНАЛІЗ АНАЛОГІВ

Для оцінки актуальності розробки програмного забезпечення для координації благодійних ініціатив варто звернути увагу на існуючі вебресурси, зокрема: «ДоброДій», «HelpKarma», «Зграя» та «Peremoga».

Функціональні можливості	ДоброДій	HelpKarma	Зграя	Peremoga	Власна розробка
Фільтрація благодійних зборів за категоріями	0	1	0	1	1
Особистий кабінет	0	1	0	0	1
Функціональність підписок і активності користувачів	0	0	0	0	1
Функціонал підписки на щомісячну підтримку	0	1	1	1	1
Рейтинг організаторів	0	1	0	0	1
Онлайн-пожертвування	1	1	1	1	1
Система взаємодії користувачів з підтримкою чат-бота	0	0	0	0	1
Функціональність реєстрації та супроводу участі в заходах	0	0	0	0	1
Сумарний коефіцієнт	1	5	2	3	8

Власна розробка має вищий сумарний коефіцієнт порівняно з міжнародною платформою «HelpKarma» на 16% (100% - 5/8*100% = 38%), благодійним фондом «Peremoga» на 63%, благодійним проєктом «Зграя» на 75% та вебресурсом «ДоброДій» на 88%

Рисунок Д.6 – Порівняльний аналіз аналогів

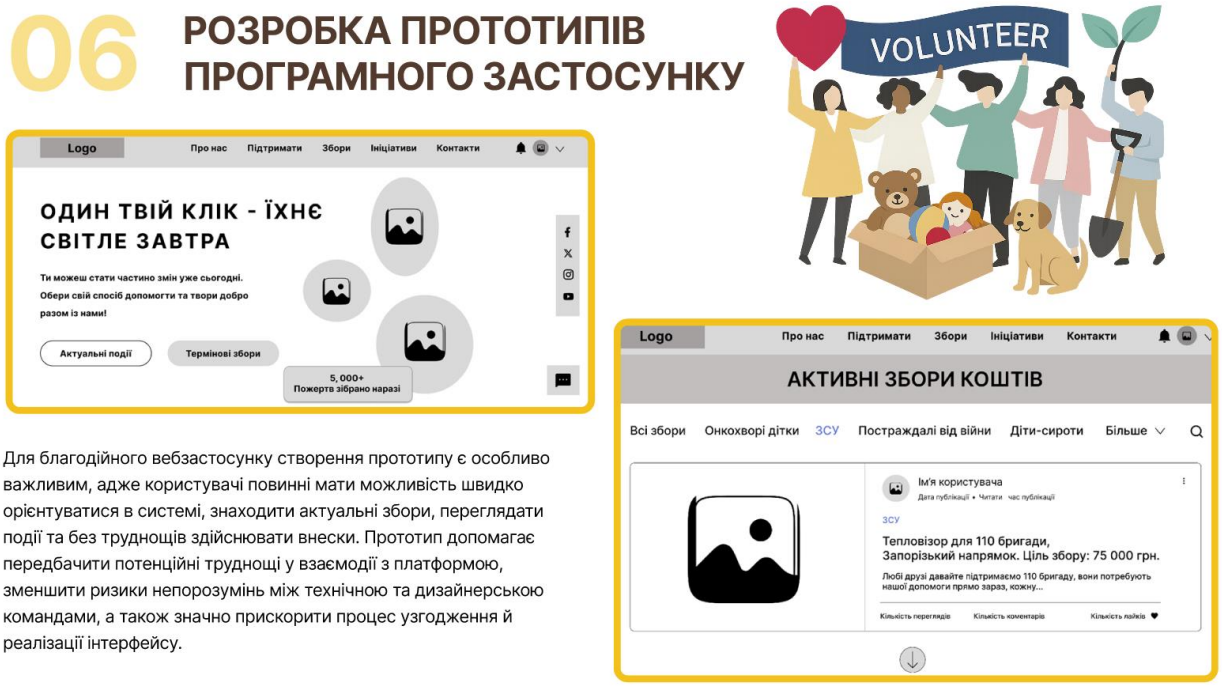


Рисунок Д.7 – Розробка прототипів програмного застосунку

Рисунок Д.8 – Дизайн графічного інтерфейсу

08 РОЗРОБКА МЕТОДУ РЕЄСТРАЦІЇ НА БЛАГОДІЙНИЙ ЗАХІД

Метод реєстрації реалізує повний цикл: від заповнення форми до підтвердження участі й інформування адміністрації. Це забезпечує зручний, захищений та масштабований процес залучення користувачів до благодійних заходів.

Основні функції:

- Користувач заповнює форму (React + Formik), валідація виконується через Yup.
- Дані обробляються функцією `handleSubmitRegistration`, проходять клієнтську й серверну перевірку (Joi).
- Запит надсилається через API (`POST /api/registrations`) до бази MongoDB.
- Використовується `nodemailer` для відправки email з унікальним токеном підтвердження (UUID).
- Застосовано перевірку на дублікати, авторизацію, обробку помилок, захист від CSRF.

Метод поєднує сучасні інструменти (Formik, Yup, Joi, nodemailer, MongoDB) в єдиний механізм реєстрації з двосторонньою валідацією, автоматичним підтвердженням участі та інформуванням адміністраторів. Це актуально для організаторів заходів, які прагнуть забезпечити безпечну, зручну і ефективну взаємодію з учасниками онлайн.

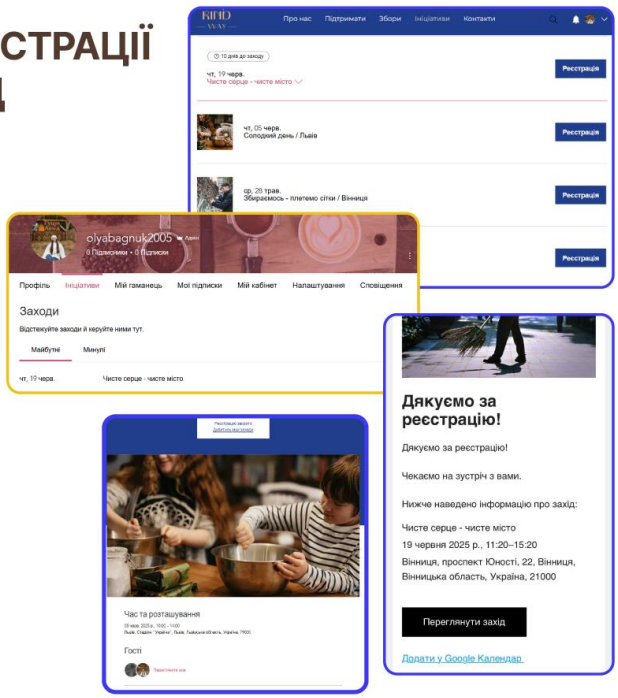


Рисунок Д.9 – Розробка методу реєстрації на благодійний збір

09 РОЗРОБКА МЕТОДУ ПОБУДОВИ СОЦІАЛЬНОЇ АКТИВНОСТІ НА ПЛАТФОРМІ В РЕАЛЬНОМУ ЧАСІ

Метод поєднує реальний час, WebSocket-зв'язок, аналітику активності та гнучку систему взаємодії між користувачами та адміністраторами. Завдяки інтеграції сучасних вебтехнологій (React, Node.js, MongoDB, Socket.IO), метод дозволяє створити живу соціальну інфраструктуру навколо благодійних ініціатив — з миттєвими сповіщеннями, обговореннями та залученням.

Опис кроків роботи методу:

1. Користувач переглядає актуальні благодійні збори інших користувачів.
2. Для підписки викликається функція subscribeToUser(subscriberId, targetUserId) → API POST /api/users/subscribe → createUserSubscription → запис у колекцію subscriptions.
3. Після підписки надходять сповіщення в реальному часі через WebSocket-подію newSubscriptionNotification.
4. Перегляд профілю здійснюється через API GET /api/users/:userId/profile та функцію getUserProfileData.
5. Коментарі реалізовано в компоненті CommentsSection. Додавання відбувається через postComment, API /api/comments та saveComment, що зберігає дані у comments.
6. Усі дані взаємодій (коментарі, підписки, повідомлення) зберігаються в MongoDB: comments, messages, subscriptions.
7. Повідомлення до адміністратора надсилаються через sendMessageToAdmin → API /api/messages.
8. Адміністратор відповідає через sendAdminResponse, а відповідь доставляється користувачу через WebSocket-подію adminResponseNotification.
9. Вся переписка зберігається для аналізу.
10. Активність користувача (лайки, перегляди, підписки тощо) фіксується через trackUserActivity у колекції userActivity.
11. WebSocket-з'єднання підтримується через initializeSocketConnection(userId) для миттєвої доставки повідомлень.

Цей опис стислий, але зберігає всю логіку і послідовність роботи методу.

Метод поєднує реальний час, WebSocket-зв'язок, аналітику активності та гнучку систему взаємодії між користувачами та адміністраторами. Завдяки інтеграції сучасних вебтехнологій (React, Node.js, MongoDB, Socket.IO), метод дозволяє створити живу соціальну інфраструктуру навколо благодійних ініціатив — з миттєвими сповіщеннями, обговореннями та залученням.

Рисунок Д.10 – Розробка методу побудови соціальної активності на платформі в реальному часі

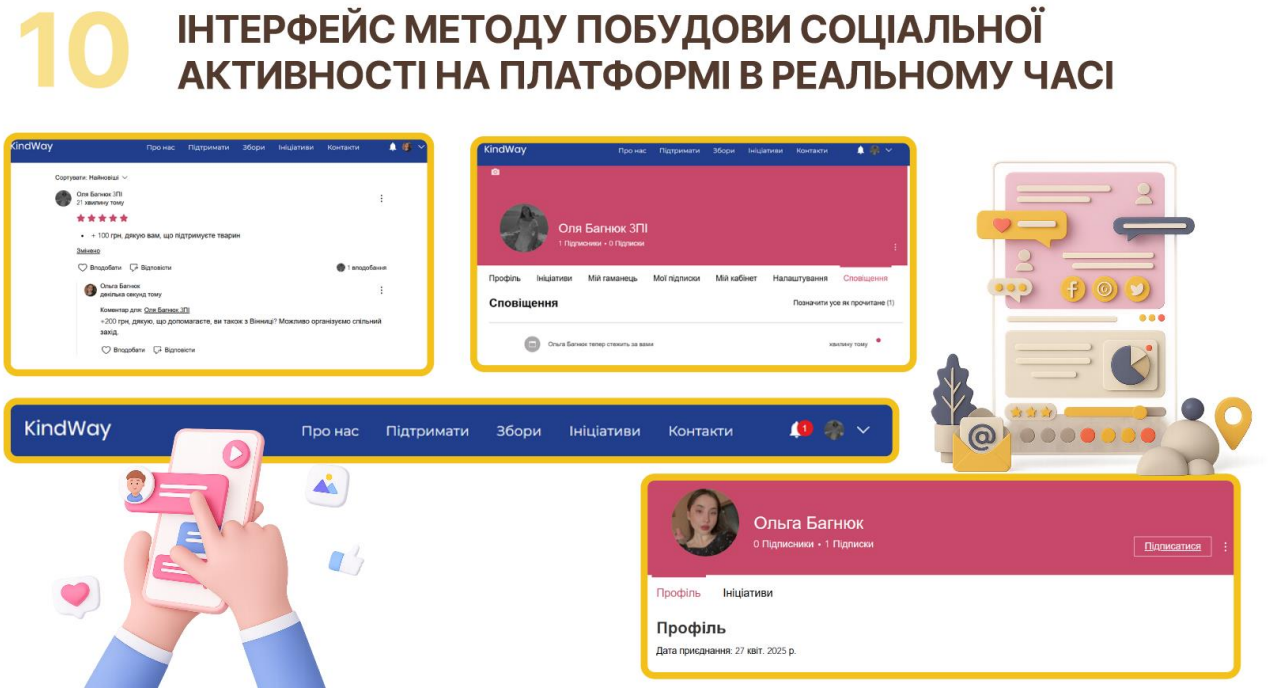


Рисунок Д.11 – Інтерфейс методу побудови соціальної активності на платформі в реальному часі

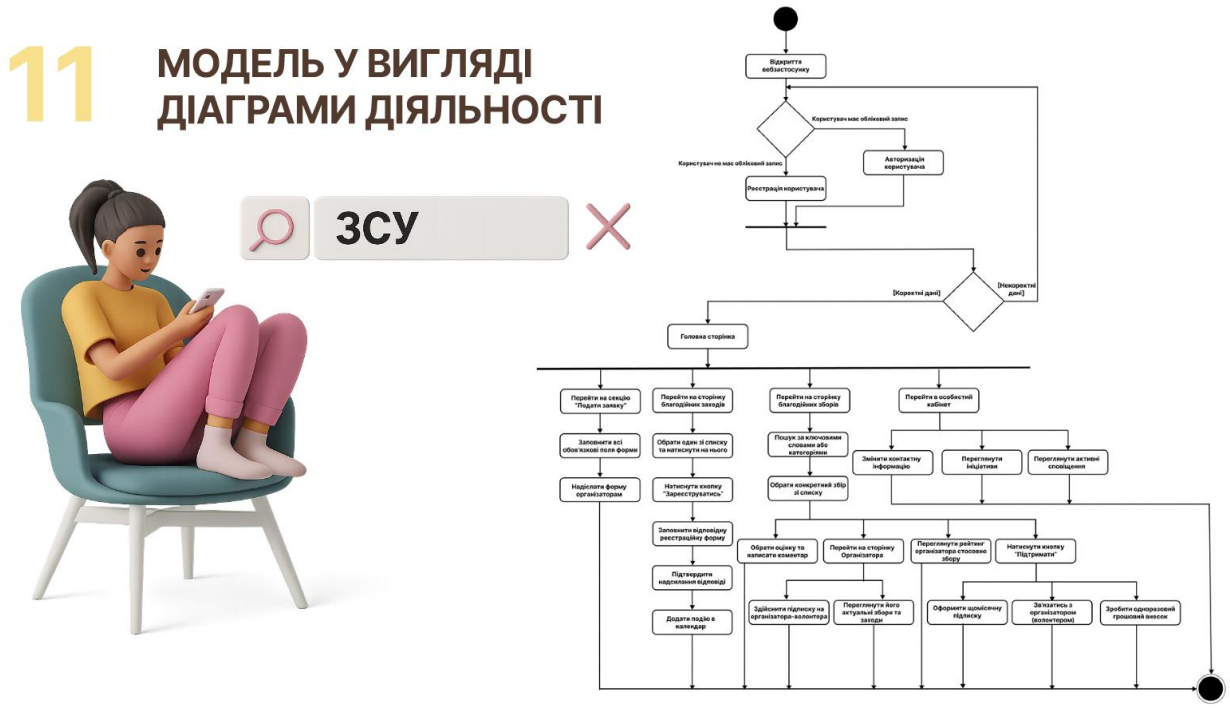


Рисунок Д.12 – Модель у вигляді діаграми діяльності

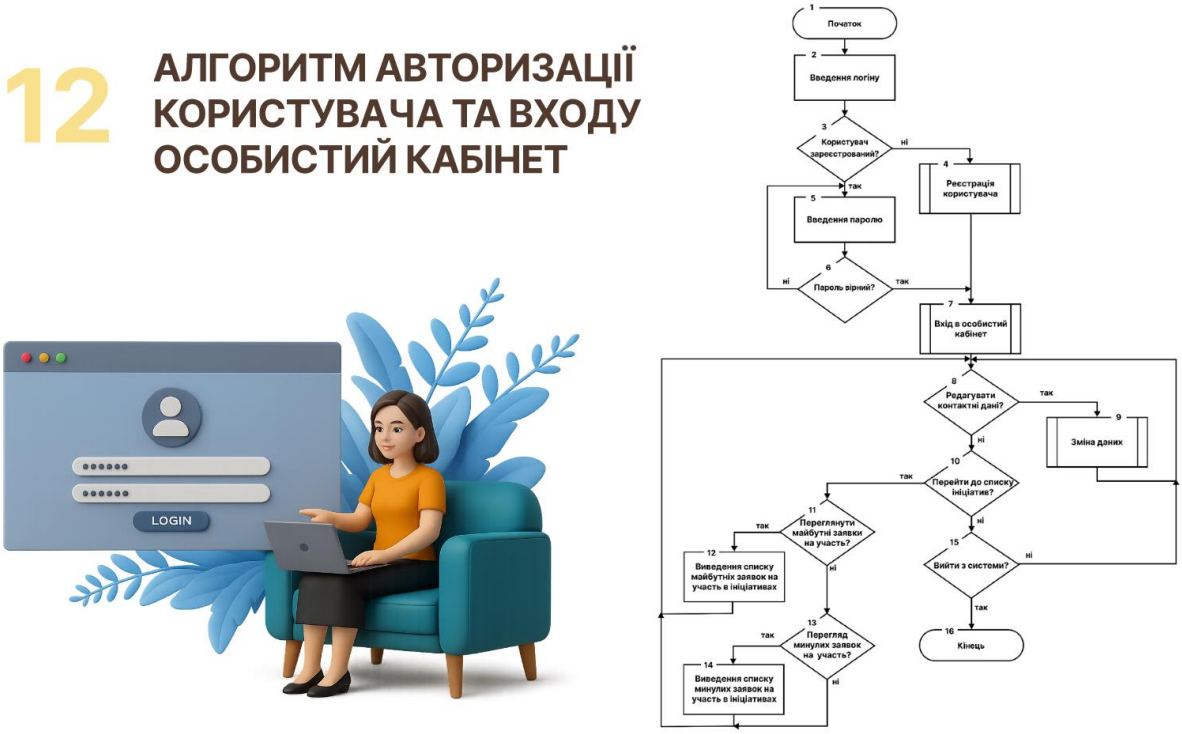


Рисунок Д.13 – Алгоритм авторизації користувача та входу в особистий кабінет

Рисунок Д.14 – Алгоритм користувацької взаємодії з благодійними зборами

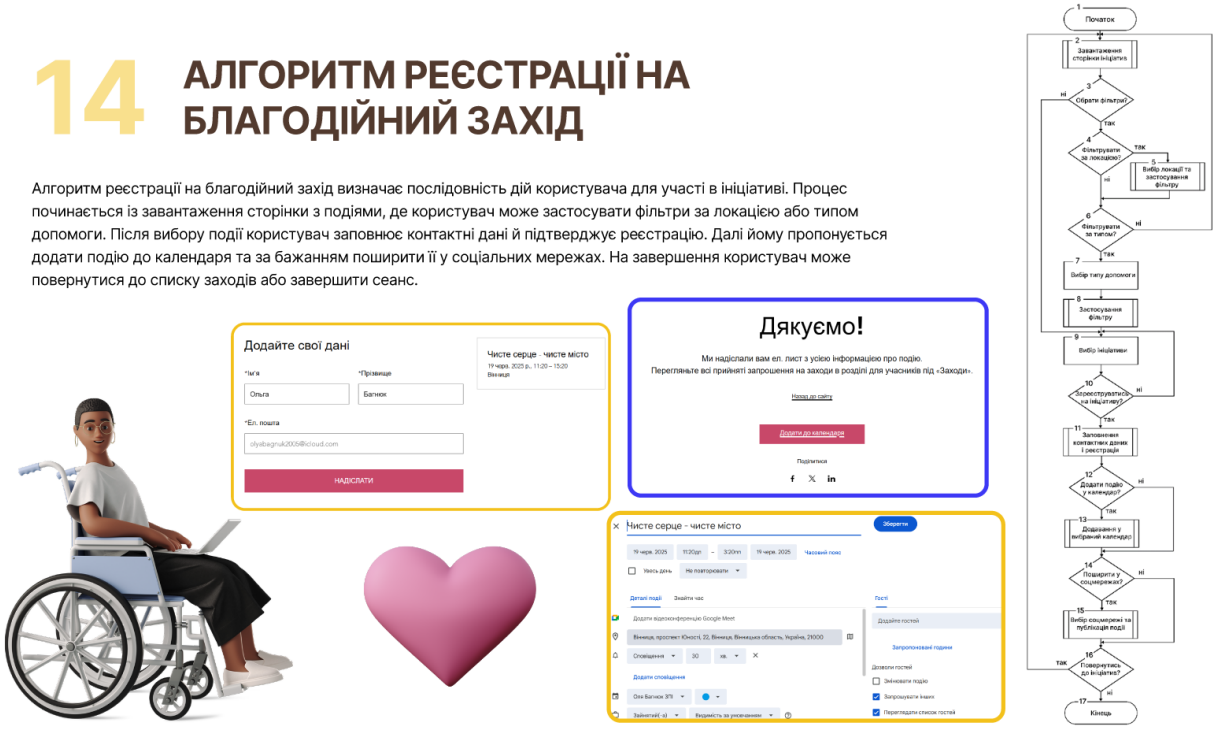


Рисунок Д.15 – Алгоритм реєстрації на благодійний захід

15 ПОРІВНЯЛЬНИЙ АНАЛІЗ МОВ ПРОГРАМУВАННЯ

Критерії функціональних вимог	JavaScript	PHP	C#	Python
Підтримка повного стека (frontend + backend)	1	0	0	0
Гнучкість у створенні SPA (single-page applications)	1	0	0	0
Підтримка розробки мікросервісів	1	0	1	1
Наявність екосистеми для real-time застосунків (Socket)	1	0	1	1
Активність спільноти, оновлення, швидкий розвиток	1	1	1	1
Підтримка хостинг-платформ та серверів «із коробки»	1	1	0	1
Масштабованість застосунку без зміни мови	1	0	1	1
Легкий запуск серверної частини без складної конфігурації	1	1	0	1
Кросплатформеність (desktop/web/mobile)	1	0	1	1
Загальний коефіцієнт	10	3	5	7

Рисунок Д.16 – Порівняльний аналіз мов програмування

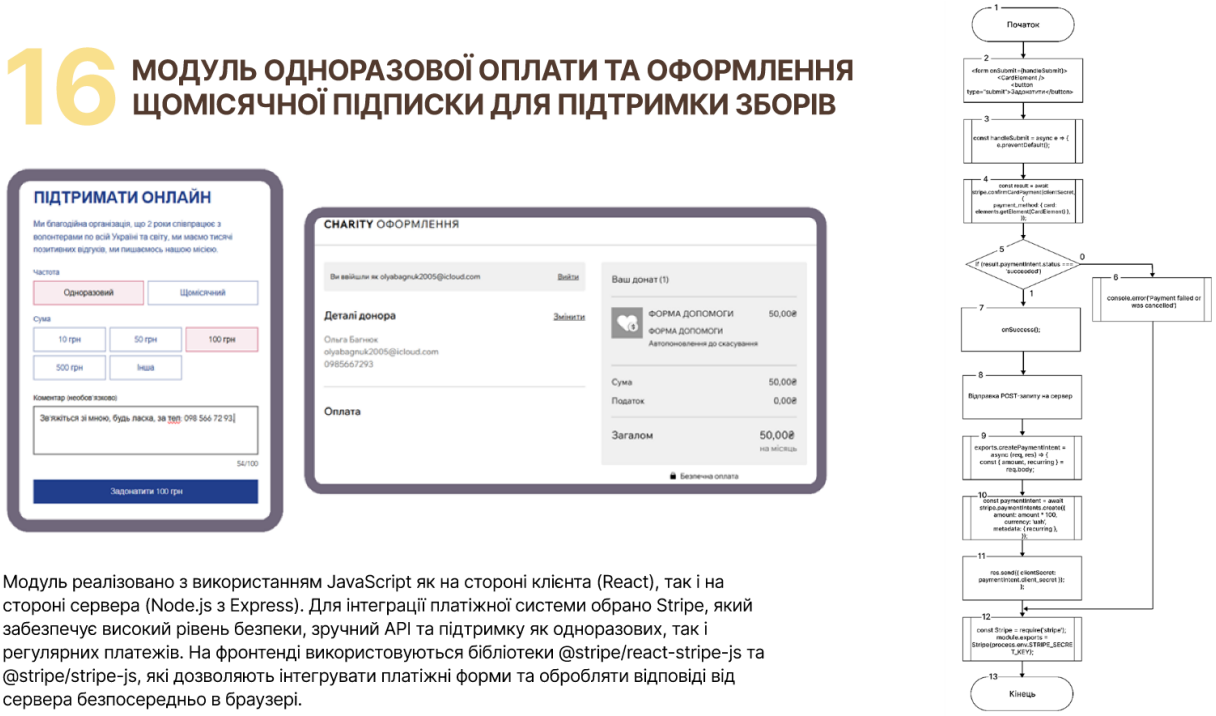


Рисунок Д.17 – Модуль одноразової оплати та оформлення щомісячної підписки для підтримки зборів

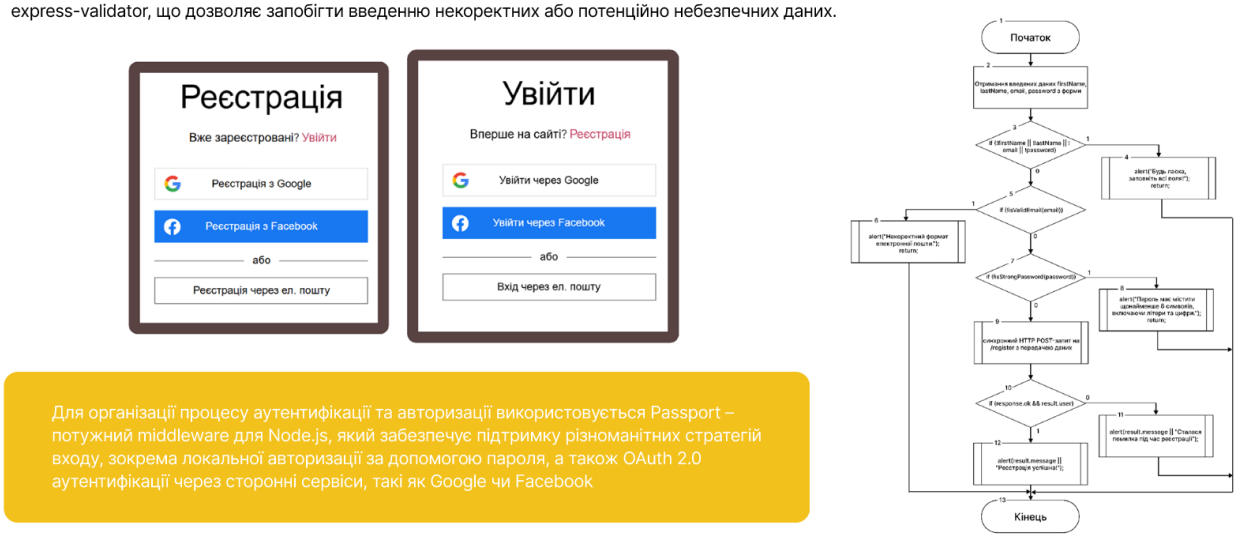
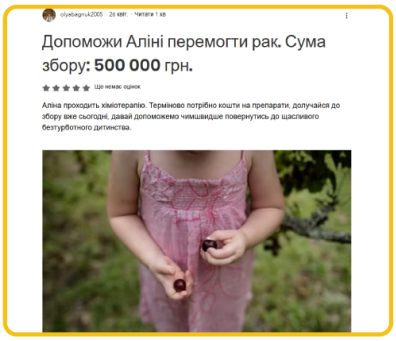


Рисунок Д.18 – Модуль авторизації та реєстрації користувачів з валідацією даних

18 МОДУЛЬ ДЛЯ ВЗАЄМОДІЇ З БЛАГОДІЙНИМИ ЗБОРАМИ

Для відображення інформації про благодійний збір реалізовано React-компонент FundraiserCard із додатковим функціоналом: лайки, коментарі, теги та кнопки для поширення у соцмережах. Використано бібліотеки React (включно з хуком useState для керування локальним станом лайків) і react-router-dom (useNavigate для переходу на сторінку збору). Компонент приймає data – об'єкт із деталями збору, показує заголовок, короткий опис, прогрес зібраних коштів, рейтинг, теги та кілька коментарів. Для лайків використовується локальний стан (useState), а функція handleLike оновлює його при кліку, запобігаючи одночасному переходу за посиланням. Кнопки соцмереж викликають функцію shareToSocial, яка відкриває вікна з URL для публікації у Facebook або Twitter тощо.



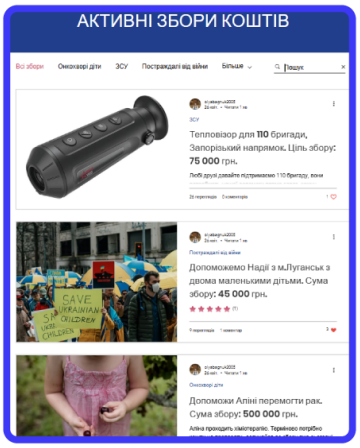


Рисунок Д.19 – Модуль для взаємодії з благодійними зборами

19 ТЕСТУВАННЯ РЕЄСТРАЦІЇ ТА АВТОРИЗАЦІЇ КОРИСТУВАЧА

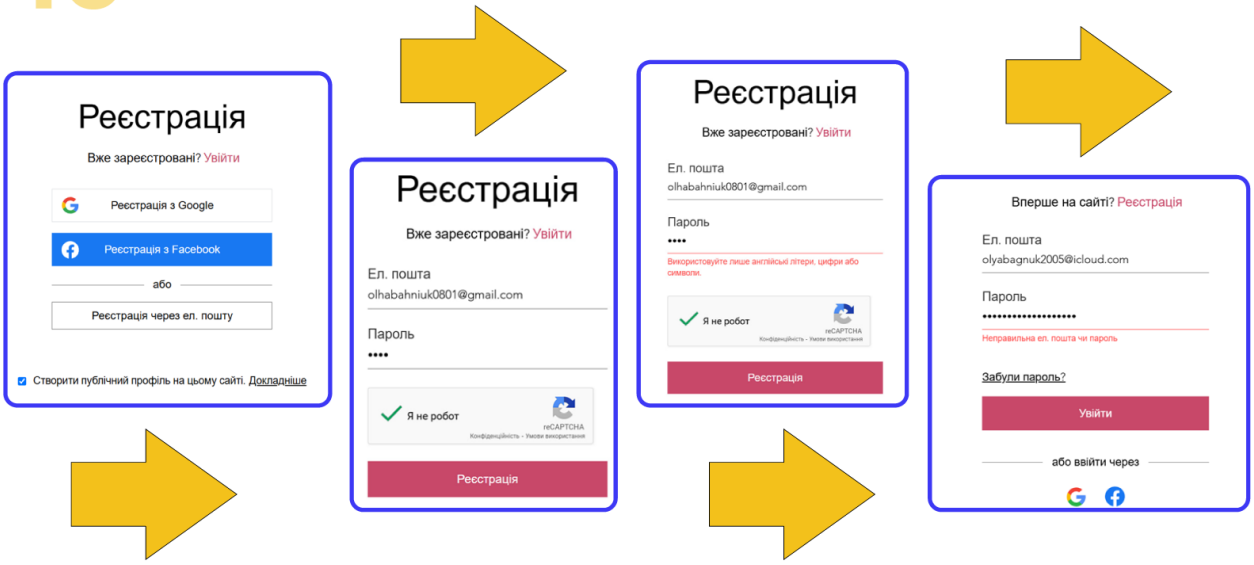


Рисунок Д.20 – Тестування реєстрації та авторизації користувача

20 ТЕСТУВАННЯ ФУНКЦІОНУВАННЯ АНКЕТИ ДЛЯ СТВОРЕННЯ НОВОЇ БЛАГОДІЙНОЇ ІНІЦІАТИВИ

Рисунок Д.21 – Тестування функціонування анкети для створення нової благодійної ініціативи

21 ТЕСТУВАННЯ ВЗАЄМОДІЇ КОРИСТУВАЧА З БЛАГОДІЙНИМИ ІНІЦІАТИВАМИ

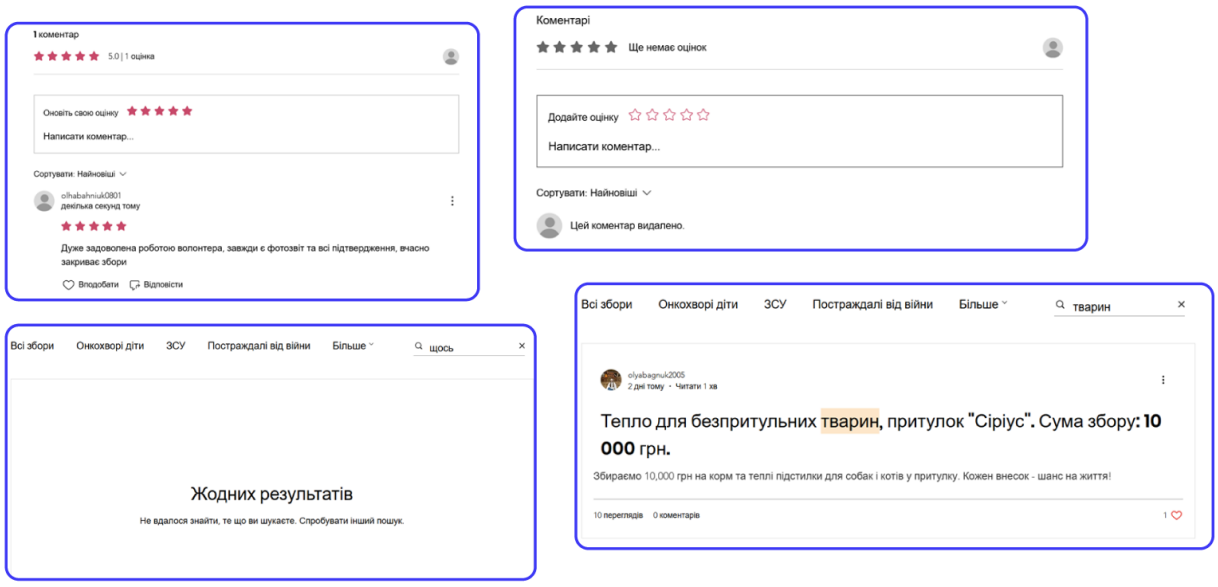


Рисунок Д.22 – Тестування взаємодії користувача з благодійними ініціативами

22 ТЕСТУВАННЯ ВИКОРИСТОВУЮЧИ LIGHTHOUSE

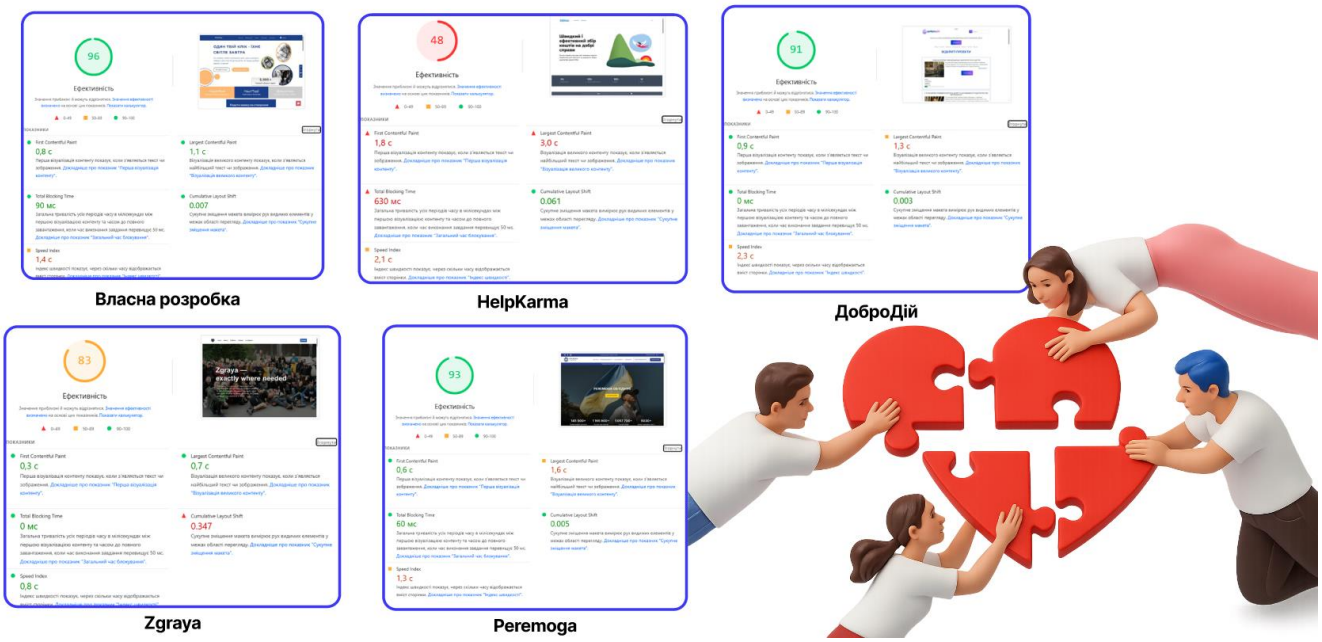


Рисунок Д.23 – Тестування використовуючи Lighthouse

23 АПРОБАЦІЯ, ПУБЛІКАЦІЯ ТА ВПРОВАДЖЕННЯ РЕЗУЛЬТАТІВ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Апробація матеріалів бакалаврської кваліфікаційної роботи.
Результати бакалаврської кваліфікаційної роботи доповідалися на:
– Всеукраїнській науково-технічній конференції молодих вчених, аспірантів і студентів «Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації» (Одеса, 2024 р.).
– Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (Вінниця, 2025р.).
– Міжнародній науково-практичній конференції «Молодь в науці: дослідження, проблеми, перспективи (Вінниця, 2025 р.).

Публікації.
За тематикою дослідження опубліковано 4 наукові праці у збірниках матеріалів конференцій.

Впровадження.
Впровадження результатів досліджень підтверджується відповідним актом та використовується в такій організації:
– громадська організація «ЮС ГЕНТІУМ» для підвищення продуктивності та якості управління благодійними ініціативами.

Рисунок Д.24 – Апробація, публікація та впровадженя результатів бакалаврської кваліфікаційної роботи

Рисунок Д.25 – Фінальний слайд «Дякую за увагу»