Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОРГАНІЗАЦІЇ ТА ПІДТРИМКИ БЛАГОДІЙНИХ ЗБОРІВ
DEVELOPMENT OF SOFTWARE FOR CHARITABLE FUNDRAISING MANAGEMENT

Виконав(ла): студент(ка) 4 курсу, групи КНТ-113сп
Спеціальності 121 Інженерія програмного
(код і найменування спеціальності)
забезпечення

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

КЛЕЩОВ А.В.
(ПРІЗВИЩЕ та ініціали)
Керівник СКРУПСЬКИЙ С. Ю.
(ПРІЗВИЩЕ та ініціали)
Рецензент ШИТІКОВА О.В.
(ПРІЗВИЩЕ та ініціали)

2026

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)

Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____Сергій СУББОТІН
“____” _____2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

КЛЕЩОВА Артема Володимировича
(ПРІЗВИЩЕ, ім’я, по батькові)

1. Тема проєкту (роботи) Розроблення програмного забезпечення для організації та підтримки благодійних зборів. Development of Software for Charitable Fundraising Management

керівник проєкту (роботи) к.т.н., доцент, СКРУПСЬКИЙ Степан Юрійович,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім’я, по батькові)
затверджені наказом закладу вищої освіти від “ 07 ” квітня 2026 року № 139
2. Строк подання студентом проєкту (роботи) 10 червня 2026 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Реалізація програми. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з з точним зазначенням обов’язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	СКРУПСЬКИЙ С.Ю., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання « 20 » квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	1.1.1 Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій	2 тиждень	Розділ 2
	розробки.		
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження	5 тиждень	Розділ 4
	програмного забезпечення.		
7	Оформлення пояснювальної записки та документів	6 тиждень	Додатки
	до неї.		
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка) _____ Артем КЛЕЩОВ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Степан СКРУПСЬКИЙ
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 91 с., 10 табл., 35 рис., 3 дод., 16 джерел.

КРАУДФАНДИНГ, БЛАГОДІЙНІСТЬ, КАМПАНІЯ, ТРАНЗАКЦІЯ, ГЕЙМІФІКАЦІЯ, ВЕБЗАСТОСУНОК, ВЕБФРЕЙМВОРК.

Об’єкт роботи - процес інженерії програмного забезпечення для організації, адміністрування та підтримки благодійних зборів.

Предмет роботи - програмні засоби створення краудфандингового вебзастосунку з модулями автоматизованого обліку транзакцій, модерації та гейміфікації.

Мета роботи - скоротити час адміністрування благодійних зборів для організаторів і підвищити залученість користувачів до повторних пожертв через розробку вебзастосунку з автоматизованою обробкою транзакцій та інтегрованою системою мотивації.

Використані матеріали та методи: мова програмування TypeScript, система керування базами даних PostgreSQL (Supabase), ORM Drizzle, фреймворк Vue.js для клієнтської частини, NestJS для серверної частини, REST API та платіжний сервіс Stripe.

Результати. Результатами виконання роботи є створення програмного забезпечення для краудфандингу, що забезпечує автоматизацію створення та модерації благодійних кампаній, облік транзакцій, а також гейміфікацію процесу пожертв. Система дозволяє організаторам зручно керувати зборами, а донорам - відстежувати свій вплив та отримувати бейджі за активність, забезпечуючи централізоване управління фінансовою статистикою.

Висновки. Розроблене програмне забезпечення дозволяє зробити процес фінансування проєктів більш прозорим та інтерактивним, зменшити адміністративне навантаження на авторів кампаній та стимулювати

користувачів до регулярної підтримки ініціатив за рахунок автоматизації ключових процесів краудфандингу.

Галузь використання. Благодійні фонди, волонтерські організації, громадські ініціативи та платформи для фінансування соціальних проєктів.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 91 pages, 10 tables, 35 figures, 3 appendixes, 16 sources.

CROWDFUNDING, CHARITY, CAMPAIGN, TRANSACTION, GAMIFICATION, WEB APPLICATION, WEB FRAMEWORK.

The object of study is the software engineering process for organizing, administering, and supporting charity fundraisers.

The subject of study is the methods and software tools for creating a crowdfunding web application with modules for automated transaction tracking, moderation, and gamification.

The purpose of the work is to reduce the time organizers spend on administering charitable fundraising campaigns and to increase user interest in repeat donations by developing a web application with automated transaction processing and a built-in incentive system.

Materials, methods, and technical tools: TypeScript programming language, PostgreSQL database management system (Supabase), Drizzle ORM, Vue.js client-side framework, NestJS server-side framework, REST API and Stripe payment service.

Results. The results of this work are the creation of crowdfunding software that automates the creation and moderation of charitable campaigns, tracks transactions, and gamifies the donation process. The system allows organizers to conveniently manage fundraising, and donors to track their impact and earn badges for their activity, providing centralized management of financial statistics.

Conclusions. The developed software makes the project funding process more transparent and interactive, reduces the administrative burden on campaign creators, and encourages users to regularly support initiatives by automating key crowdfunding processes.

Areas of application: charitable foundations, volunteer organizations, community initiatives, and platforms for funding social projects.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

LTV – Lifetime Value;

БД – база даних;

ВФ – вебфреймворк;

СКБД – система керування базами даних.

ВСТУП

Благодійність та колективний збір коштів стали невід'ємною частиною суспільного життя. Значущість цього явища важко переоцінити, адже щодня фінансуються тисячі ініціатив. Про вкорінення цієї галузі свідчить швидка адаптація фінансового сектору, який створює зручні платіжні інструменти на кшталт цифрових «банок», якими користуються мільйони людей.

Зараз підтримка краудфандингових ініціатив відбувається переважно на таких рівнях:

– через розповсюдження прямих платіжних посилань у соціальних мережах;

– за допомогою монолітних платформ великих фондів;

– через персональні сторінки волонтерів, які ведуть облік транзакцій вручну.

Незважаючи на розвиток технічної бази для переказу грошей, поза увагою залишається критично важливий аспект - психологія та утримання мотивації донора. Здебільшого процес зводиться до одноразової транзакції: людина переказує кошти і просто закриває сторінку. Через відсутність інтерактивного зворотного зв'язку благодійність часто залишається лише епізодичним імпульсом, а не регулярною звичкою. До того ж, організатори витрачають значну кількість часу на ручне зведення статистики, замість того щоб працювати зі своєю аудиторією.

Сучасні користувачі добре реагують на механіки гейміфікації, які успішно працюють у навчальних чи фітнес-додатках. Створення подібного середовища в сфері благодійних зборів має потенціал кардинально змінити поведінку благодійників. Планується, що інтеграція механіки «стріків», тобто відстеження серій послідовних днів із пожертвами, це стимулюватиме користувача не переривати свій ланцюжок добрих справ, формуючи стійку звичку. А система накопичувальних бейджів, які видаватимуться за підтримку різних кампаній чи регулярність, дозволить візуалізувати індивідуальний вплив

людини та підвищити її статус. Саме на проєктуванні та розробці таких мотиваційних інструментів у поєднанні з автоматизацією звітів планується зробити акцент у даній роботі.

Метою роботи є скорочення витрат часу організаторів на адміністрування благодійних зборів та збільшення залученості користувачів до повторних пожертв шляхом розроблення вебзастосунку з автоматизованою обробкою транзакцій та вбудованою системою мотивації.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз застосунків для підтримки та організації благодійних зборів

Сучасний ландшафт цифрової благодійності перебуває у стані фундаментальної трансформації, зумовленої зміною парадигми взаємодії між ініціаторами соціальних проєктів та донорами. Глобальна цифровізація фінансових інструментів призвела до виникнення сектору «Філантропії 2.0», де ключовими чинниками успіху є швидкість залучення ресурсів, прозорість цільового використання коштів та рівень емоційного залучення аудиторії [1]. Актуальність дослідження даної галузі підсилюється соціально-економічними викликами, які вимагають від волонтерських спільнот високої операційної ефективності та здатності до швидкої мобілізації капіталу в умовах обмежених ресурсів.

Дослідження бізнес-середовища свідчить, що цифрова благодійність еволюціонувала від поодиноких акцій великих фондів до масового явища мікро-донацій [2]. Даний процес характеризується децентралізацією, де кожен учасник мережі може виступати одночасно і донором, і розповсюджувачем інформації, формуючи складні структури соціального капіталу (рис. 1.1).

Рисунок 1.1 – Діаграма зростання поширення благодійних зборів [3]

Проте, зі зростанням кількості публічних зборів, спостерігається феномен «втоми від співчуття», коли з’являється дуже велика потреб для донатів і люди починають реагувати на це «скрізь очі», то це вимагає впровадження нових підходів до утримання уваги благодійників та автоматизації рутинних процесів управління зборами.

Для забезпечення однозначності трактування процесів у межах даного дослідження необхідно визначити базовий термінологічний апарат предметної області:

– публічний збір - процес відкритого залучення добровільних пожертв від необмеженого кола осіб для досягнення конкретної соціально значущої або благодійної мети;

– мікро-донація - фінансова транзакція малого обсягу, яка за умови високої частоти та масовості забезпечує стале наповнення фонду збору;

– цільове фінансування - принцип використання акумульованих коштів виключно за призначенням, задекларованим на початку кампанії;

– верифікація волонтера - процедура підтвердження особистості та репутації організатора збору, що є критично важливою для формування довіри в безперсональному цифровому середовищі;

– значення LTV донора - прогнозований сукупний обсяг внесків, які один благодійник здійснить протягом усього періоду взаємодії з платформою або організатором [4].

На рис. 1.2 зображено життєвий цикл благодійної кампанії в існуючих умовах (процес «як є») зазвичай починається з ініціації збору в соціальних мережах або месенджерах. Організатор створює публікацію з описом потреби, необхідних доказових файлів та реквізитами для переказу коштів. Поточні механізми передбачають використання банківських інструментів для накопичення, проте вони функціонують відокремлено від інформаційних каналів.

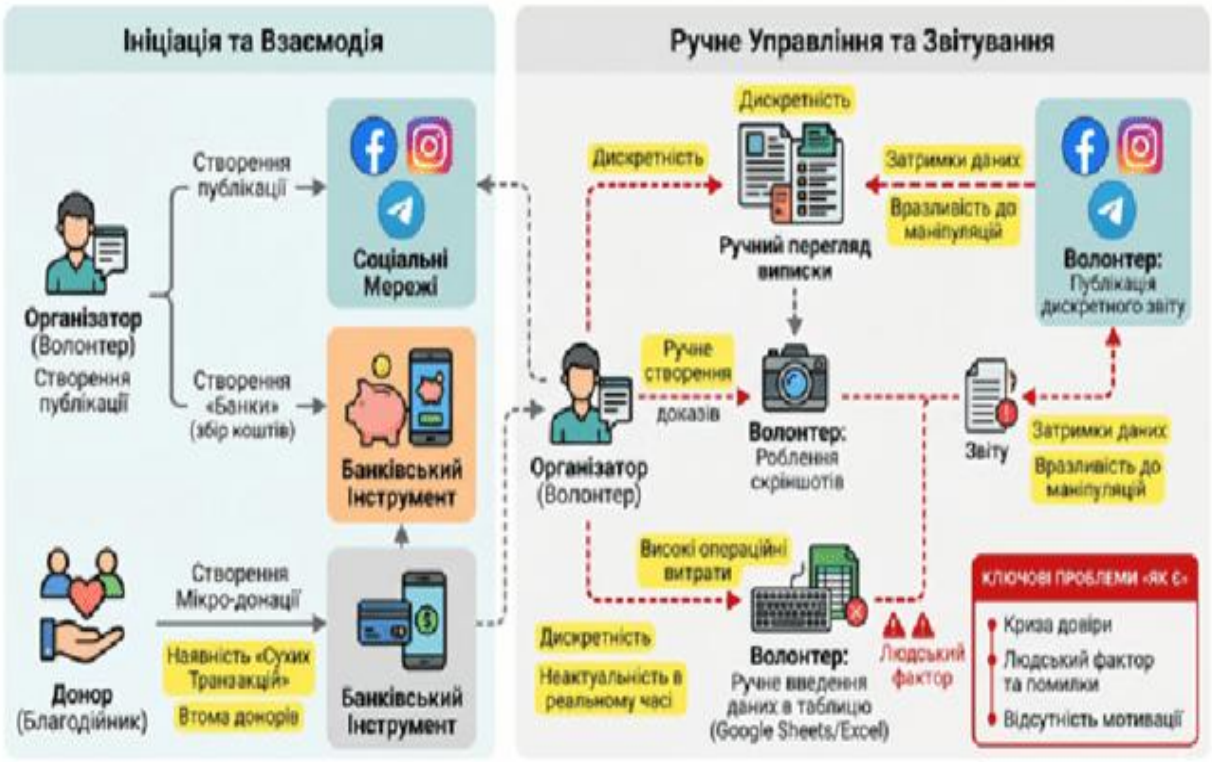

Рисунок 1.2 – Схема бізнес–процесу організації благодійного збору «як є»

На етапі активного збору волонтер змушений вручну відстежувати надходження, робити скріншоти банківських виписок та публікувати їх як доказ прозорості. Такий підхід до звітування є дискретним і не забезпечує актуальності даних у режимі реального часу. Крім того, облік донорів, їхньої лояльності та частоти внесків практично не ведеться, або здійснюється за допомогою табличних процесорів у ручному режимі. Це створює значне операційне навантаження на організаторів, відволікаючи їх від основної діяльності.

Дослідження процесів дозволило ідентифікувати низку критичних проблемних місць, що потребують системного розв'язання на рівні програмної інженерії:

– криза довіри та складність верифікації. Відсутність автоматизованих механізмів підтвердження транзакцій створює можливості для маніпуляцій та шахрайства. Донори змушені покладатися на репутацію особи, а не на технічні гарантії системи;

– високі операційні витрати на звітування. Процес збору та систематизації даних про витрати та надходження виконується вручну, що призводить до затримок у наданні інформації та потенційних помилок у розрахунках [5];

– ефект «сухої транзакції» та проблема мотивації. Більшість існуючих методів збору не передбачають зворотного зв'язку для донора. Після здійснення платежу користувач не отримує підтвердження свого внеску в загальний прогрес, що знижує ймовірність повторної участі;

– низький рівень утримання. Відсутність інструментів для побудови довгострокових відносин з благодійниками призводить до того, що більшість зборів тримаються на одноразових внесках, що є неефективним з точки зору стратегічного планування ресурсів.

Виявлені недоліки свідчать про технологічний розрив між потребами сучасного волонтерського руху та можливостями наявних інструментів. Обґрунтування розробки спеціалізованого програмного забезпечення базується на необхідності ліквідації цього розриву шляхом автоматизації фінансового моніторингу та впровадження інженерних методів управління мотивацією користувачів. Перехід від «ручного» управління до автоматизованої системи дозволить не лише підвищити прозорість благодійності, а й значно збільшити обсяги залучених коштів за рахунок покращення користувацького досвіду та оптимізації процесів звітності.

1.2 Порівняльний аналіз розробки та аналогів

Сучасний ринок програмного забезпечення для благодійності перебуває у фазі активної сегментації, зумовленої диференціацією потреб суб'єктів предметної області. Аналіз існуючих рішень дозволяє класифікувати їх за кількома базовими архітектурно–функціональними моделями, кожна з яких орієнтована на специфічний тип взаємодії з донором. Зокрема, виділяються професійні платформи для некомерційних організацій, інструменти класичної

сфери благодійних внесків, сервіси для регулярних пожертв та спеціалізовані банківські інструменти швидкого накопичення коштів [6]. Дана сегментація відображає еволюцію галузі від простих форм передачі капіталу до складних екосистем управління соціальним капіталом та лояльністю донорів. Проте, незважаючи на технологічну зрілість ринку, спостерігається значний розрив між інструментами фінансового адміністрування та механізмами психологічного утримання користувачів.

Перший значний сегмент ринку охоплює професійні платформи та інструменти інституційного краудфандингу, де найбільш показовими прикладами є вітчизняна платформа «dobro.ua»[7] на рис. 1.3 та глобальний лідер «GoFundMe» [8].

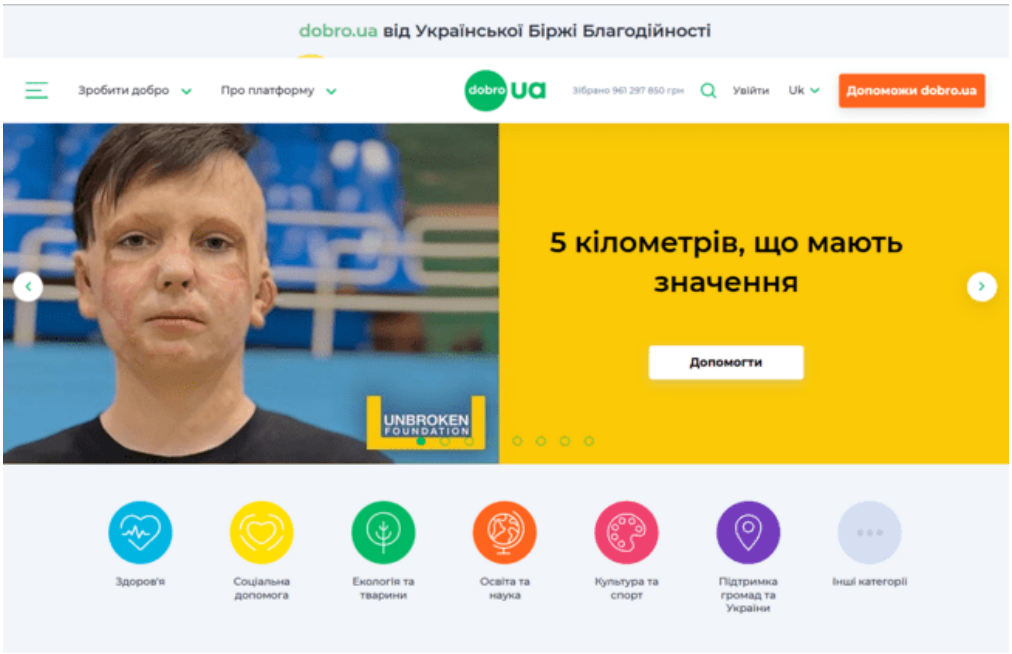

Рисунок 1.3 – Вебзастосунок «dobro.ua» [7]

Застосунки цього типу орієнтовані на забезпечення найвищого рівня довіри та легітимності зборів через складні процедури перевірки. Українська платформа «dobro.ua» базується на моделі суворої верифікації організаторів, де доступ до створення кампаній мають лише офіційно зареєстровані некомерційні організації (НКО), що проходять глибоку юридичну та фінансову перевірку.

Функціонал даної платформи забезпечує повний цикл звітності: кожна витрата має бути підтверджена фіскальними документами, які вручну завантажуються адміністраторами організацій. Це гарантує цільове використання коштів, проте створює значне операційне навантаження та призводить до дискретності звітування, коли дані оновлюються із затримкою. Для індивідуального волонтера така модель є занадто складною через високий поріг входу. Глобальне рішення «GoFundMe», хоч і розширює цю модель на фізичних осіб, покладається на механізми соціального підтвердження, проте зберігає фокус на великих одноразових цільових зборах. Звітування тут часто обмежується текстовими оновленнями від організатора, що знижує прозорість для масового мікро-донора.

Окрему нішу займають сервіси, що базуються на моделі регулярної підтримки та персонального фандрейзингу, серед яких провідні позиції посідають «Patreon» [9] та «4fund». Платформа «Patreon» на рис. 1.4 використовує модель підписки, що є ефективним для побудови сталого фінансування довгострокових проєктів. Організатор збору формує різні рівні підтримки із фіксованими сумами, а донори отримують певні привілеї або емоційну винагороду.

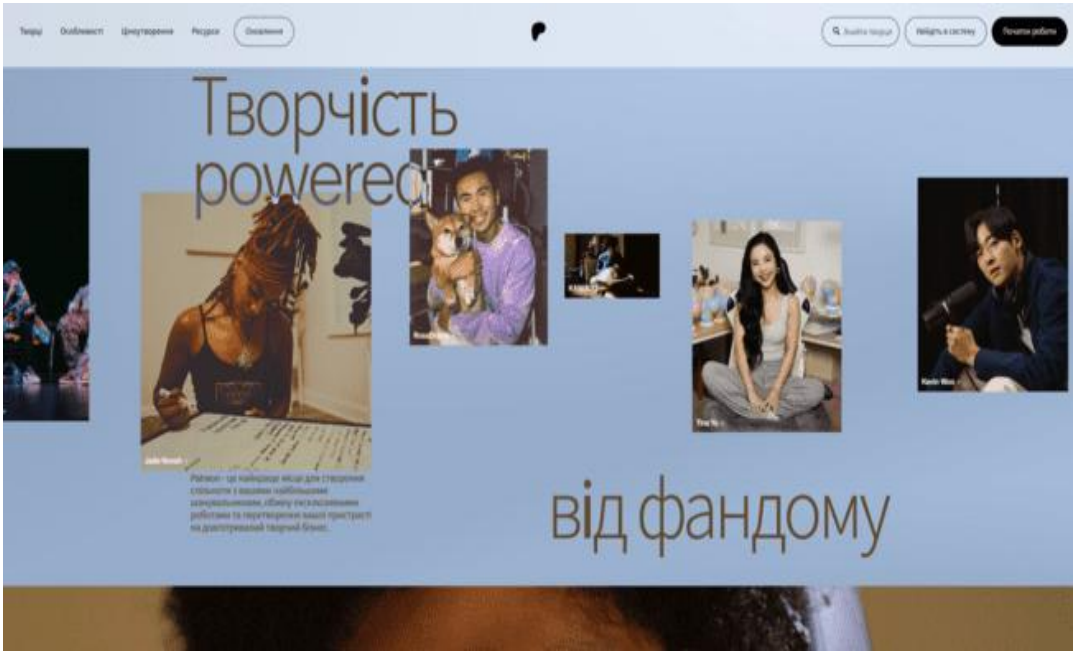

Рисунок 1.4 – Вебплатформа «Patreon» [9]

Ця модель є найбільш наближеною до концепції утримання донорів, оскільки базується на регулярності платежів. Проте, архітектура «Patreon» не оптимізована для термінових благодійних зборів (наприклад, зборів на медикаменти чи обладнання з чітко обмеженим часом), де критичною є візуалізація прогресу в реальному часі. Високі комісійні витрати (від 5% до 12% плюс збори платіжних систем) та складність виводу коштів для українських користувачів є суттєвими бар'єрами. Схожий сервіс «4fund» намагається спростити процес для персонального використання, проте він позбавлений розвинених інструментів мотивації, обмежуючись лише базовими індикаторами збору коштів.

Найбільш динамічним сегментом в Україні наразі є фінтех-інструменти швидкого накопичення, представлені сервісом «Банка» від Monobank на рис. 1.5. Популярність даного інструменту серед волонтерів зумовлена парадигмою «мінімізації тертя» при здійсненні транзакцій. Донор має можливість зробити внесок у кілька кліків через інтегрований банківський застосунок або через швидкі платежі «Apple Pay» та «Google Pay», що критично важливо для спонтанних мікро-донацій.

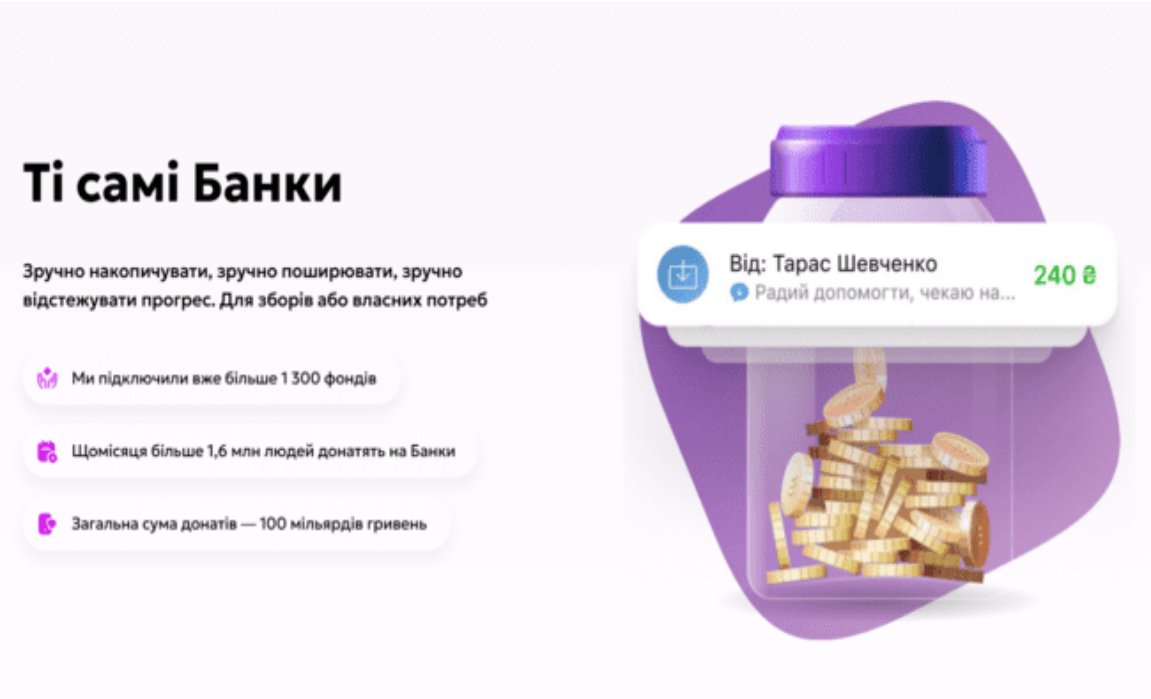

Рисунок 1.5 – Сервіс «Банка» від Monobank [10]

Під час аналізу виявлено фундаментальний розрив: професійні платформи забезпечують прозорість, але мають високий поріг входу, тоді як банківські сервіси забезпечують швидкість, але ігнорують емоційний контекст та утримання аудиторії (табл 1.1).

Таблиця 1.1 – Порівняння функціональних можливостей існуючих систем

Критерій порівняння	GoFundMe	dobro.ua	Patreon	Monobank «Банка»
Автоматизація звітності	Низька	Середня	Середня	Низька
Комісія сервісу	2.9% + $0.30	0%	5–12%	0%
Механіки утримання (Streaks)	Відсутні	Відсутні	Частково	Відсутні
Система нагород (Badges)	Відсутня	Відсутня	Відсутня	Відсутня
Синхронізація в реальному часі	Так	Ні	Так	Так
Доступність для фіз. осіб	Висока	Низька	Висока	Висока

Ключовою проблемою галузі залишається низький показник LTV донора. Існуючі системи не використовують мотиваційні чинники, описані в методологіях типу Octalysis [11]. Брак механік гейміфікації, таких як серії внесків або цифрові нагороди, призводить до того, що донори не відчувають власного прогресу. Це створює «когнітивний вакуум», який організатори змушені заповнювати ручною роботою в соціальних мережах. Обґрунтування розробки спеціалізованого ПЗ базується на необхідності ліквідації цього

розриву шляхом створення системи, що поєднає надійність банківських API із сучасними методами Software Engineering у сфері мотиваційного дизайну.

1.3 Висновки до розділу

Результати проведеного системного аналізу предметної області дозволяють стверджувати, що сфера цифрової благодійності в Україні перебуває на етапі активної трансформації у бік децентралізації та масових мікро–донацій. Попри розвиток загальних платіжних систем, бізнес–середовище досі характеризується значним розривом між банківськими інструментами та інформаційними платформами волонтерських груп, що створює надмірне операційне навантаження на організаторів зборів.

Під час аналізу існуючих аналогів було виявлено низку фундаментальних недоліків, серед яких найважливішими є відсутність дієвих механік утримання донорів та низький рівень автоматизації звітності. Більшість систем розглядають благодійний внесок як разову фінансову операцію, не забезпечуючи емоційного зворотного зв’язку чи візуалізації індивідуального внеску. Це призводить до швидкого виснаження мотивації благодійників та змушує волонтерів вести облік надходжень у ручному режимі, що підвищує ризики помилок і знижує прозорість кампаній.

Виявлена проблематика обґрунтовує необхідність розробки спеціалізованого програмного забезпечення, яке виступатиме інтелектуальною надбудовою над банківськими API. Метою подальшої роботи є проєктування системи, що поєднає автоматизований моніторинг транзакцій у реальному часі із сучасними методами мотиваційного дизайну, такими як серії внесків та цифрові нагороди. Такий підхід дозволить не лише спростити адміністрування зборів, а й сформувати сталу лояльність донорів через прозору модерацію та персоналізовану візуалізацію їхніх досягнень.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вибір засобів проєктування

Проєктування сучасної платформи для організації та підтримки благодійних зборів з високим рівнем інтерактивності та гейміфікації вимагає ретельного підбору технологічного стеку. Основними критеріями вибору є надійність обробки фінансових даних, масштабованість архітектури та можливість реалізації механік реального часу. Нижче наведено інженерне обґрунтування обраних засобів розробки.

2.1.1 Обґрунтування вибору мови програмування

Для реалізації як серверної, так і клієнтської частин системи обрано мову TypeScript. На відміну від класичного JavaScript, TypeScript надає механізми суворої статичної типізації, що є критично важливим для проєктів, які оперують грошовими транзакціями та складними об’єктами лояльності [12].

Порівняно з Python, який часто використовується у веб–розробці завдяки лаконічності, TypeScript у середовищі Node.js демонструє вищу продуктивність при обробці великої кількості одночасних I/O операцій (асинхронна модель Event Loop). Використання TypeScript дозволяє:

– мінімізувати кількість помилок на етапі компіляції (Type–safety), що особливо важливо при інтеграції із зовнішніми банківськими інтерфейсами;

– забезпечити консистентність типів даних між Frontend та Backend частинами застосунку;

– використовувати сучасні патерни проєктування (декоратори, інтерфейси, абстрактні класи), що відповідає стандартам інженерії програмного забезпечення [13].

2.1.2 Вибір та обґрунтування вебфреймворків

Для побудови серверної логіки (Backend) обрано фреймворк NestJS. Це прогресивне рішення для Node.js, яке за замовчуванням використовує TypeScript та пропонує чітку модульну архітектуру. Вибір NestJS обґрунтований наступними факторами:

– впровадження залежностей - вбудований механізм дозволяє створювати слабозв’язані компоненти, що полегшує тестування та подальше розширення функціоналу;

– модульність - розподіл системи на окремі модулі (UserModule, BadgesModule, CampaignModule) дозволяє паралельно розробляти та масштабувати різні частини системи;

– екосистема - наявність готових інструментів для валідації даних, обробки винятків та інтеграції з БД значно прискорює процес розробки без втрати якості архітектури [14].

Для розробки користувацького інтерфейсу (Frontend) обрано ВФ Vue.js 3 із використанням Composition API. У порівнянні з React, Vue.js пропонує більш інтуїтивно зрозумілу систему реактивності та чітке розділення логіки, шаблонів і стилів. Використання Vue.js дозволяє забезпечити швидке відтворення інтерфейсу (Virtual DOM) та легку інтеграцію з бібліотеками UI–компонентів, такими як PrimeVue, що є оптимальним для створення складних адміністративних панелей та дашбордів волонтера.

2.1.3 Архітектурні підходи та взаємодія

Взаємодія між клієнтом та сервером базується на принципах REST API для стандартних CRUD–операцій (створення збору, реєстрація користувача, перегляд профілю), UML-діаграма архітектури зображена на рис. 2.1. Також використання Webhook для відловлювання дійсності операції з оплати.

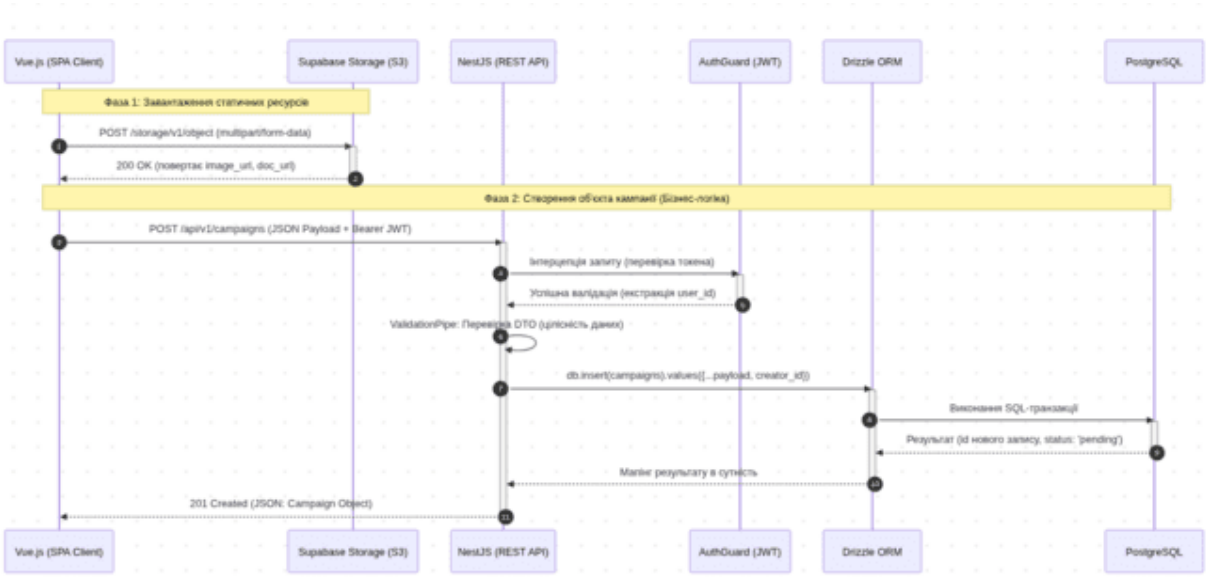

Рисунок 2.1 – UML-діаграма архітектури застосунку

Для маніпуляції даними обрано Drizzle ORM [15]. На відміну від традиційних «важких» ORM, Drizzle є максимально легким та забезпечує повну синхронізацію типів TypeScript зі схемою БД PostgreSQL. Це дозволяє уникнути розбіжностей у типах даних на рівні коду та БД, а також гарантує високу швидкість виконання запитів за рахунок відсутності зайвих абстракцій (табл. 2.1).

Таблиця 2.1 – Порівняльних аналіз технологій

Критерій	TypeScript / NestJS	Python / Django	JavaScript / Express
Типізація	Статична (сувора)	Динамічна	Відсутня
Швидкість розробки	Висока	Середня	Висока
Продуктивність (I/O)	Висока	Середня	Висока
Масштабованість	Висока (Модульна)	Висока	Низька (хаотична)
Підтримка впровадження залежностей	Вбудована	Сторонні бібліотеки	Відсутня

2.1.4 Середовище розробки та допоміжні інструменти

Основним середовищем розробки обрано Visual Studio Code (VS Code) через його глибоку інтеграцію з екосистемою TypeScript та наявність розширень для роботи з Docker та Git.

Для забезпечення ідентичності середовищ розробки, тестування та продакшну використовується технологія контейнеризації Docker. Це дозволяє ізолювати залежності проєкту та БД, мінімізуючи ризики несумісності на різних платформах. Контроль версій здійснюється за допомогою системи Git.

В якості хмарної інфраструктури для БД та авторизації обрано платформу Supabase, яка надає керований екземпляр PostgreSQL та вбудовані механізми Row Level Security (RLS) для захисту даних користувачів на рівні БД [16]. Тестування критичних вузлів системи (логіка нарахування стріків, обробка транзакцій) реалізується за допомогою фреймворку Vitest, який забезпечує високу швидкість виконання Unit–тестів у Vite–середовищі.

Обґрунтований вибір засобів проєктування створює надійний фундамент для реалізації функціональних вимог системи, забезпечуючи високу відмовостійкість та можливість гнучкого розширення ігрових механік у майбутньому.

2.2 Сценарії використання програми

Проєктування функціональної структури системи базується на методології прецедентів, що дозволяє формалізувати взаємодію між зовнішніми акторами та програмним продуктом.

2.2.1 Класифікація акторів системи

У проєктованій системі виділено три основні ролі (актори), які мають ієрархічну структуру доступу до функціональних можливостей:

– анонімний користувач - це базовий актор, який має доступ до загальнодоступної інформації та публічних інструментів благодійності без необхідності ідентифікації в системі;

– авторизований користувач - актор, що пройшов процедуру реєстрації та автентифікації. Ця роль успадковує всі права анонімного користувача та отримує доступ до персоналізованих сервісів, управління власним профілем та гейміфікаційних механік;

– модератор - привілейований актор, відповідальний за цілісність та легітимність контенту. Успадковує функціонал авторизованого користувача, додаючи до нього інструменти адміністративного управління та валідації кампаній.

2.2.2 Опис основних сценаріїв взаємодії

Для кожної групи акторів розроблено набір прецедентів, що покривають повний життєвий цикл благодійного збору та взаємодії з платформою (рис. 2.2).

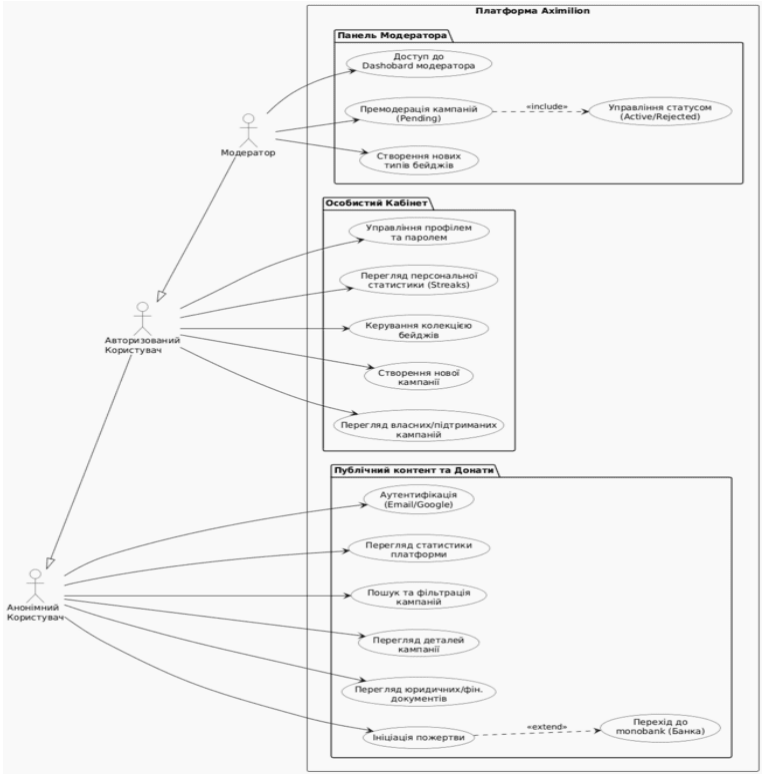

Рисунок 2.2 – Use–case діаграма системи

Основний потік сценаріїв для анонімного користувача:

– авторизація на платформі через email та пароль;

– авторизація на платформі через google аккаунт;

– переглядати статистику платформи;

– переглянути кампанії по певній категорії;

– переглядати верифіковані кампанії;

– переглядати трендові кампанії;

– переглядати останні донати по платформі;

– переглянути список усіх активних кампаній;

– відфільтрувати активні кампанії за категорією, проміжком між необхідної суми кампанії, терміном закінчення кампанії;

– відсортувати за терміном закінчення кампанії;

– переглянути певну кампанію;

– переглянути прикріплені зображення до кампанії;

– переглянути верифікований юридично доказовий документ;

– переглянути верифікований фінансовий аудит;

– скопіювати посилання на кампанію;

– перейти по посиланню «банки» monobank, якщо то прикріплено;

– зробити пожертву за запропонованими варіантами, або вписати певну суму;

– сплатити пожертву через платіжну систему Stripe;

– переглядати етапи кампанії;

– переглядати останні пожертви на обрану кампанію;

– шукати певну кампанію по ключовим словам.

Функціонал авторизованого користувача наслідує потік сценаріїв анонімного актора, але більше зосереджений на персональному прогресі та активному фандрейзингу:

– оновлювати персональну інформацію профіля;

– оновлювати пароль аккаунту;

– переглянути статистику по сумі пожертв;

– переглянути статистику по кількості підтримань унікальних кампаній;

– переглянути серію днів коли робились пожертви;

– переглянути усі отримані бейджі;

– переглянути інформацію по певному бейджу;

– переглянути усі доступні бейджі;

– переглянути нещодавно підтримані кампанії;

– переглянути нещодавні персональні операції;

– переглянути персональні кампанії;

– створити нову кампанію.

Актор «Модератор» наслідує потік сценаріїв авторизованого актора, але більш фокусується на підтримці, модерації та якості платформи:

– відкривати дешбоард модератора;

– переглядати кампанії які є в статусі очікування;

– підтверджувати кампанію та переводити її в статус «активної»;

– відхиляти кампанію та переводити її в статус «відхиленої»;

– створювати нові бейджі.

Сценарій реєстрації забезпечує створення нового облікового запису в системі та ініціалізацію персонального профілю донора. Процес базується на використанні сервісу Supabase Auth для забезпечення безпечного зберігання облікових даних (рис. 2.3).

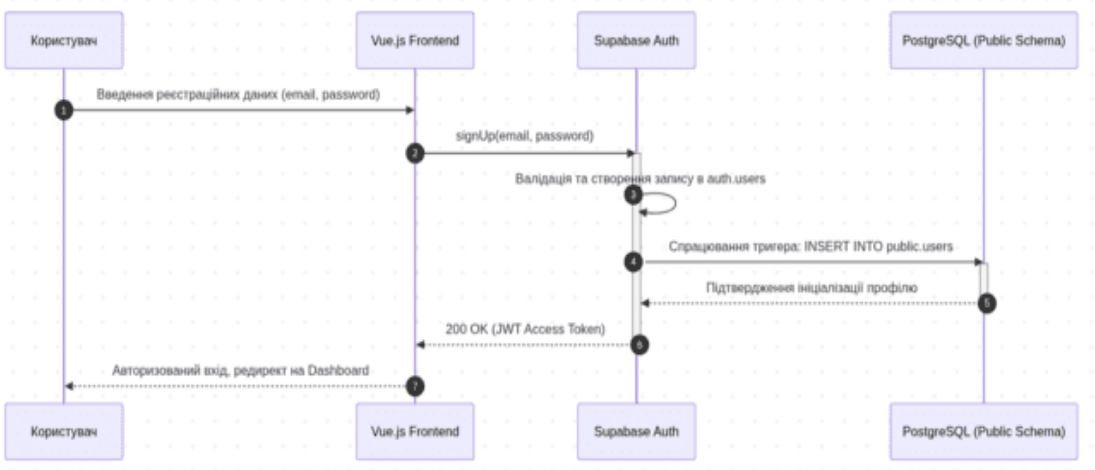

Рисунок 2.3 – UML діаграма послідовності реєстрації користувача

Сценарій створення нової благодійної кампанії реалізує логіку ініціації збору коштів волонтером. Ключовою особливістю є забезпечення прозорості через обов'язкове завантаження доказових документів (рис. 2.4).

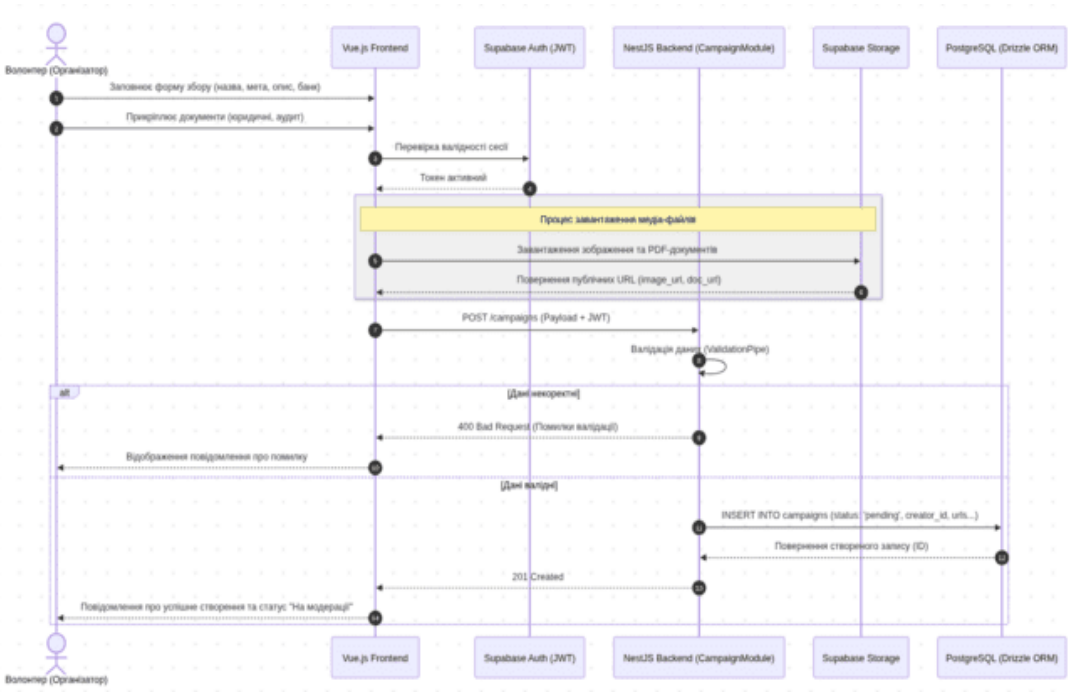

Рисунок 2.4 – UML діаграма послідовності процесу створення нової кампанії

Сценарій підтримки кампанії описує інтеграцію із зовнішнім банківським API та подальшу обробку подій для оновлення стану системи та нарахування ігрових нагород (рис. 2.5).

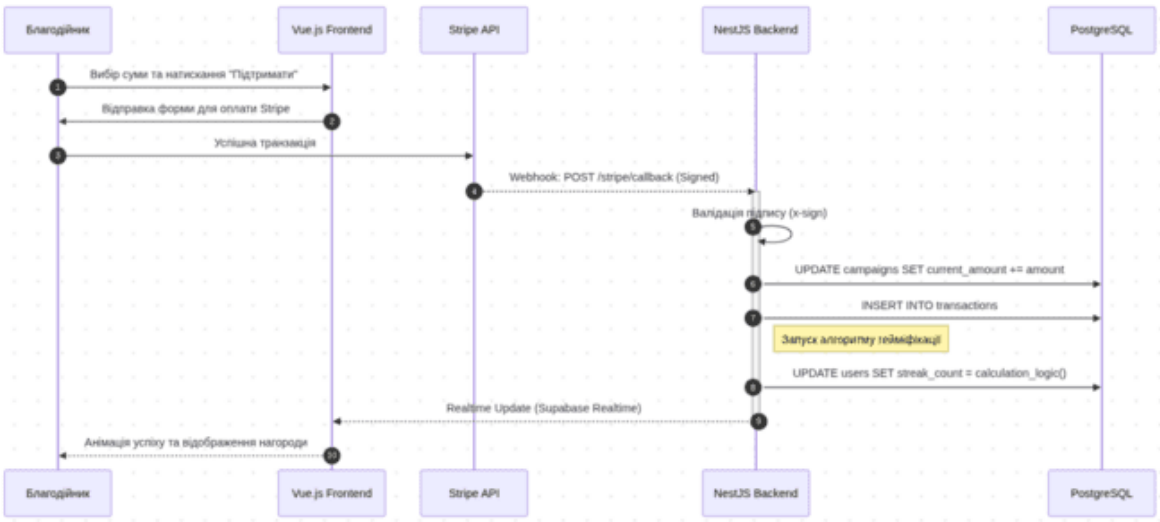

Рисунок 2.5 – UML діаграма послідовності підтримки кампанії

2.2.3 Прототипування

На етапі проєктування інтерфейсу було розроблено прототипи низької деталізації, що дозволило візуалізувати інформаційну архітектуру вебзастосунку та логіку взаємодії користувача з елементами системи без прив’язки до фінального графічного дизайну. Основна увага приділялася ергономіці розміщення елементів управління та логіці навігації.

Головна сторінка (рис. 2.6) спроєктована як лендинг, основною метою якого є швидке залучення користувача та демонстрація прозорості платформи.

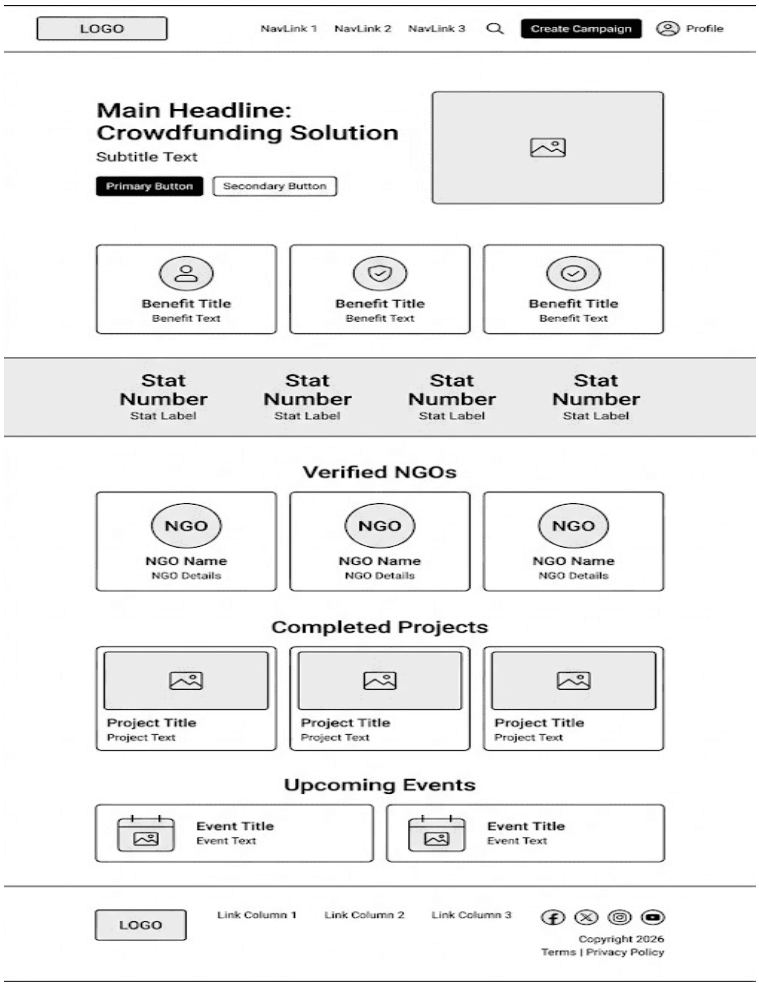

Рисунок 2.6 – Прототип головної сторінки

Прототип головної сторінки має структуру, яка містить наступні блоки:

– Hero-секція містить головний заклик до дії (CTA), кнопки швидкого переходу до створення збору або перегляду активних кампаній;

– блок глобальної статистики має динамічні лічильники, що відображають загальну суму зібраних коштів, кількість активних донорів та успішно завершених проєктів;

– секція верифікованих НГО містить картки перевірених фондів та організацій, що підвищує рівень довіри до платформи;

– трендові та активні кампанії це сітка з прев'ю–картками зборів, де відображається назва, категорія та поточний прогрес у відсотках;

– футер містить посилання на юридичні документи, соціальні мережі та навігаційну мапу сайту.

Сторінка конкретного збору (рис. 2.7) є ключовим вузлом взаємодії донора з організатором.

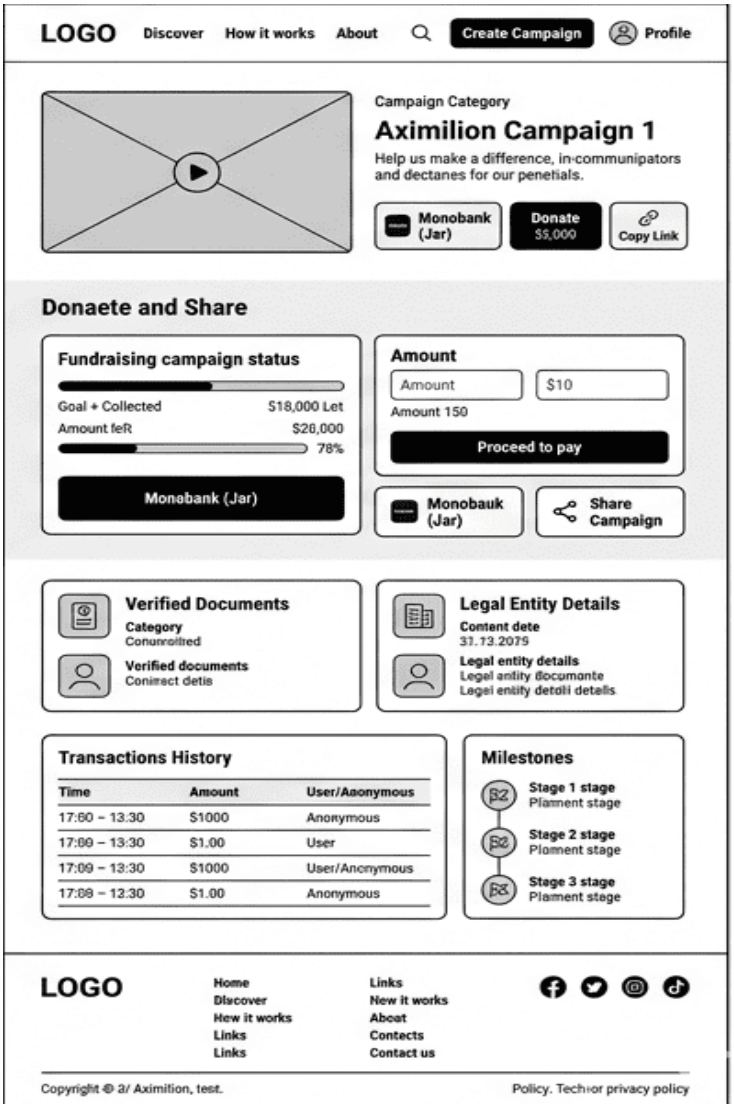

Рисунок 2.7 – Прототип сторінки благодійної кампанії

Її прототип орієнтований на максимальну інформативність та доказовість:

– медіа-зона та опис це місце для візуальних матеріалів (фото/відео) та детальної історії збору;

– віджет донації: інтерактивний блок із можливістю швидкого вибору суми або введення довільного значення, а також пряме посилання на накопичувальний рахунок (Банку) Monobank;

– доказова база: окрема секція з посиланнями на юридичні документи та фінансові аудити, що підтверджують валідність збору.

– стрічка транзакцій та Етапи (Milestones): список останніх пожертв у реальному часі та візуалізація кроків виконання проєкту (наприклад, закупівля обладнання, логістика тощо);

– система мотивації: блоки, що відображають потенційні бейджі, які користувач може отримати за підтримку саме цієї кампанії.

2.2.4 Логіка альтернативних сценаріїв

Сценарії взаємодії передбачають обробку виняткових ситуацій через альтернативні потоки подій. Це забезпечує надійність системи та захист від некоректного використання. Перелік альтернативних сценаріїв:

– помилки валідації, якщо при створенні кампанії або здійсненні донату дані не відповідають формату (наприклад, від’ємна сума або непідтримуваний формат зображення), система перериває основний потік, відображає локалізоване повідомлення про помилку та повертає актора до кроку введення даних;

– помилки авторизації, у разі введення невірного паролю або закінчення терміну дії сесії, система перенаправляє користувача на сторінку входу, зберігаючи стан попереднього запиту, якщо це можливо;

– порушення прав доступу, при спробі несанкціонованого доступу до дешборду модератора за прямим посиланням, система перевіряє роль

користувача в JWT–токені. Якщо роль не відповідає рівню доступу, сервер повертає статус помилки 403 (Forbidden), а клієнтська частина відображає повідомлення про обмеження доступу та логує інцидент безпеки.

2.3 Структура бази даних

Для забезпечення надійного зберігання даних, підтримки цілісності та можливості масштабування системи було обрано реляційну систему керування БД PostgreSQL, розгорнуту на платформі Supabase. Вибір зумовлений підтримкою складних зв’язків, механізмів типізації даних та вбудованими інструментами безпеки на рівні рядків.

На рис. 2.8 представлена ER-модель БД, що складається з п’яти основних таблиць, які забезпечують функціонування облікових записів, управління кампаніями, фінансові транзакції та систему гейміфікації. Проєктування реляційної структури базується на принципах нормалізації, де центральною сутністю є таблиця користувачів, що слугує вузлом для ідентифікації та накопичення ігрових метрик. Взаємодія між волонтерами та ініціативами реалізована через зв’язок типу один-до-багатьох, що дозволяє одному організатору керувати необмеженою кількістю зборів.

Фінансові потоки інтегровані через відношення між кампаніями та транзакціями, де кожна пожертва ідентифікується за унікальним ідентифікатором збору. Це забезпечує точність розрахунку зібраних коштів та оновлення статусів у реальному часі, а необов’язковий зв’язок із користувачами дозволяє проводити анонімні платежі. Система мотивації реалізована через зв’язок багато-до-багатьох між сутностями користувачів та нагород за допомогою проміжної таблиці. Використання типів uuid та jsonb гарантує гнучкість логіки нарахування досягнень, а обмеження зовнішніх ключів забезпечують цілісність даних при масштабуванні платформи.

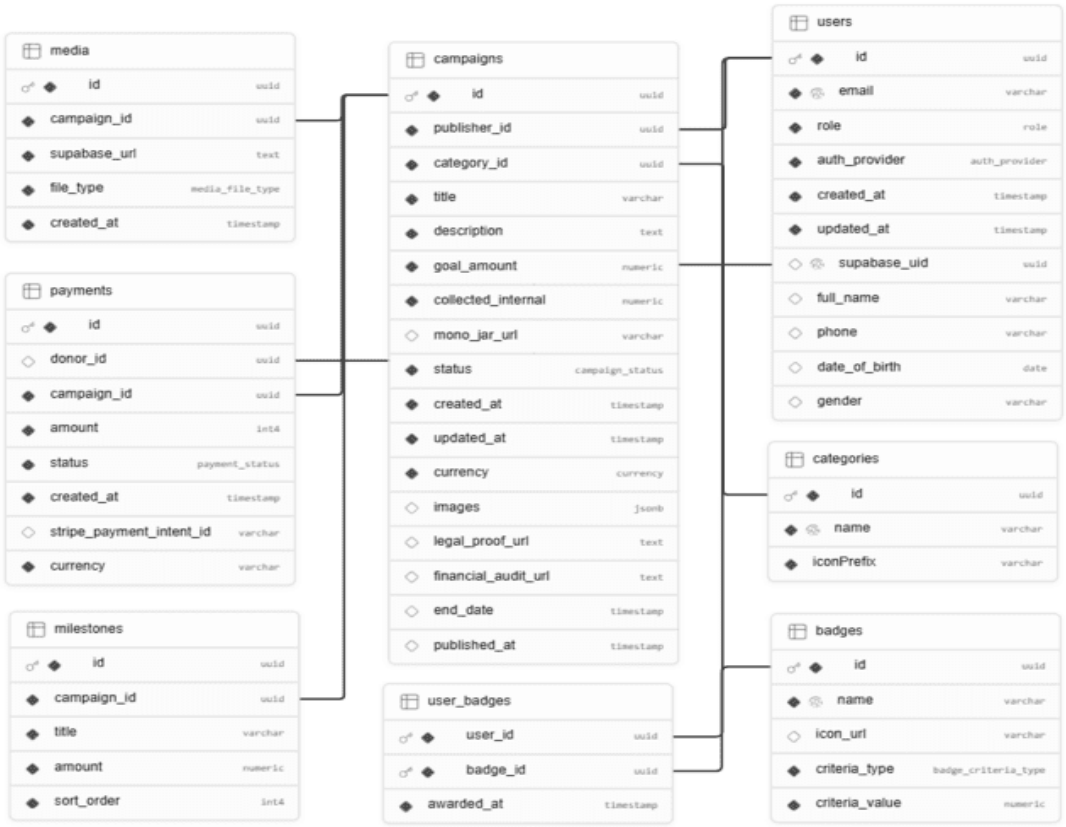

Рисунок 2.8 – ER–модель БД програмного забезпечення для організації та підтримки благодійних зборів

Структура БД представлена набором взаємопов’язаних сутностей, опис яких наведено нижче.

Сутність «Users» визначає облікові записи користувачів та їхні показники залученості, структура таблиці відображена в табл. 2.2.

Таблиця 2.2 – Структура таблиці сутності «Users»

Назва стовпця	Тип даних	Призначення
id	uuid	Унікальний ідентифікатор користувача (Primary Key).
email	text	Електронна пошта для входу та сповіщень.
role	Enum(role)	Роль користувача, може бути «registered»або «moderator».

Кінець таблиці 2.2

Назва стовпця	Тип даних	Призначення
auth_provider	Enum(auth_provider)	Провайдер через який користувач був авторизований. Може бути «local» або «google»
created_at	timestamptz	Дата та час реєстрації облікового запису.
updated_at	timestamptz	Дата та час оновлення облікового запису.
supabase_uid	uuid	Унікальний ідентифікатор користувача для синхронізацією з Supabase
full_name	text	Повне ім'я та прізвище користувача.
phone	varchar	Номер телефона користувача
date_of_birth	date	Дата народження користувача
gender	varchar	Стать користувача

Сутність «Campaigns» визначає повну інформацію про збори коштів, структура таблиці відображена в табл. 2.3.

Таблиця 2.3 – Структура таблиці сутності «Campaigns»

Назва стовпця	Тип даних	Опис
id	uuid	Унікальний ідентифікатор кампанії (Primary Key).
publisher_id	uuid	Унікальний ідентифікатор користувача який створив кампанію
category_id	uuid	Зовнішній ключ для зв’язку з таблицею категорій
title	text	Назва благодійного збору.
description	text	Текстовий опис мети та деталей кампанії.
goal_amount	numeric	Цільова сума збору коштів.

Кінець таблиці 2.3

Назва стовпця	Тип даних	Опис
collected_internal	numeric	Фактично зібрана сума (за замовчуванням 0).
mono_jar_url	text	Посилання на накопичувальний рахунок (Банку) Monobank.
status	text	Поточний стан (draft, pending, active, completed, rejected).
created_at	timestamptz	Дата та час створення запису.
updated_at	timestamptz	Дата та час оновлення запису.
currency	Enum(currency)	Поле валюти кампанії. Може бути «UAH», «EUR» або «USD»
images	jsonb	Масив посилань на зображень зі сховища
legal_proof_url	text	Посилання на файл юридичного доказу зі сховища
financial_audit_url	Text	Посилання на файл фінансовий аудит кампанії зі сховища
end_date	timestamp	Дата та час закінчення кампанії
published_at	timestamp	Дата та час публікації кампанії.

Сутність «Payments» визначає інформацію про платіжну операцію, структура таблиці відображена в табл. 2.4.

Таблиця 2.4 – Структура таблиці сутності «Payments»

Назва стовпця	Тип даних	Опис
id	uuid	Унікальний ідентифікатор транзакції (Primary Key).
donor_id	uuid	Зв'язок із користувачем (Foreign Key -> users.id).
campaign_id	uuid	Зв'язок із кампанією (Foreign Key -> campaigns.id).
amount	numeric	Сума грошового переказу.
status	text	Статус успішності операції.
created_at	timestamptz	Точний час здійснення пожертви.
stripe_payment_intent	varchar	Номер операції в Stripe
currency	varchar	В якій валюті робилась операція

Сутність «Badges» визначає інформацію про бейджі, структура таблиці відображена в табл. 2.5.

Таблиця 2.5 – Структура таблиці сутності «Badges»

Назва стовпця	Тип даних	Опис
id	uuid	Унікальний ідентифікатор бейджа (Primary Key).
name	text	Публічна назва нагороди.
icon_url	text	Посилання на графічний файл іконки.
criteria_type	text	Тип умови отримання (напр., 'donation_count', 'streak').
criteria_value	numeric	Порогове значення для виконання умови.

Сутність «User_Badges» визначає зв’язок між користувачем та бейджами які він отримав, структура таблиці відображена на табл. 2.6.

Таблиця 2.6 – Структура таблиці сутності «User_Badges»

Назва стовпця	Тип даних	Опис
id	uuid	Унікальний ідентифікатор запису (Primary Key).
user_id	uuid	Посилання на користувача (Foreign Key –> users.id).
badge_id	uuid	Посилання на бейдж (Foreign Key –> badges.id).
awarded_at	timestamptz	Дата та час моменту отримання нагороди.

Сутність «Categories» визначає категорії кампаній, структура таблиці відображена в табл. 2.7.

Таблиця 2.7 – Структура таблиці сутності «Categories»

Назва стовпця	Тип даних	Опис
id	uuid	Унікальний ідентифікатор запису (Primary Key).
name	varchar	Назва категорії
iconPrefix	varchar	Префікс іконки відповідну до іконок бібліотеки PrimeVue

Сутність «Media» визначає інформацію про медіа файли, структура таблиці відображена в табл. 2.8.

Таблиця 2.8 – Структура таблиці сутності «Media»

Назва стовпця	Тип даних	Опис
id	uuid	Унікальний ідентифікатор запису (Primary Key).
campaign_id	uuid	Зв'язок із кампанією (Foreign Key -> campaigns.id).
supabase	text	Посилання на файл в сховищі Supabase
file_type	Enum(media_file_type)	Тип файлу. Може бути «gallery», «cover», «legal_proof» або «financial_audit»
created_at	timestamptz	Точний час завантаження файлу

Сутність «Milestones» визначає шляхи кампанії, структура таблиці відображена в табл.2.9.

Таблиця 2.9 – Структура таблиці сутності «Milestones»

Назва стовпця	Тип даних	Опис
Id	uuid	Унікальний ідентифікатор запису (Primary Key).
campaign_id	uuid	Зв'язок із кампанією (Foreign Key –> campaigns.id).
name	text	Назва шляху.
amount	numeric	Необхідна сума для передолання шляху
sort_order	int4	Номер порядку шляху

2.4 Висновки за розділом 2

У ході виконання другого розділу було розв’язано комплекс архітектурних та проєктних задач, спрямованих на створення програмного забезпечення для організації благодійних зборів. Результати даного етапу роботи дозволяють зробити наступні підсумкові твердження:

Обґрунтування технологічного стеку. Проведено детальний аналіз та підтверджено вибір засобів реалізації, що базуються на мові програмування TypeScript та екосистемі Node.js. Використання фреймворку Nest.JS для серверної частини та Vue.js для клієнтської сторони дозволяє побудувати масштабовану систему з високим рівнем типізації та швидким відгуком інтерфейсу. Вибір реляційної СКБД PostgreSQL у хмарному середовищі Supabase спільно з Drizzle ORM забезпечує надійність збереження фінансових даних.

Визначено основних акторів системи (анонімний користувач, авторизований благодійник та модератор) та розроблено детальні сценарії їхньої взаємодії з програмою. Описано прецеденти, що охоплюють як базовий функціонал фандрейзингу, так і специфічні гейміфікаційні механіки та адміністративні процедури валідації кампаній.

Спроєктовано логічну схему БД, яка повністю адаптована для управління сутностями благодійних кампаній, фінансових транзакцій та профілів користувачів. Структура БД підтримує складні реляційні зв’язки, необхідні для реалізації систем лояльності (стріки, бейджі) та автоматизованої звітності. Впроваджені рішення щодо індексації та використання специфічних типів даних (UUID, JSONB) гарантують високу продуктивність та гнучкість системи.

Таким чином, сформований набір проєктних рішень створює необхідний теоретичний та технічний фундамент для переходу до етапу безпосередньої реалізації програмних модулів та тестування системи.

3 РЕАЛІЗАЦІЯ ПРОГРАМИ

3.1 Визначення порядку роботи програми

Логіка функціонування програмного забезпечення базується на забезпеченні безперервного користувацького досвіду, починаючи від ознайомлення з публічними даними до отримання ігрових нагород за активність. Порядок роботи застосунку визначається функціональною схемою, представленою на рис. 3.1.

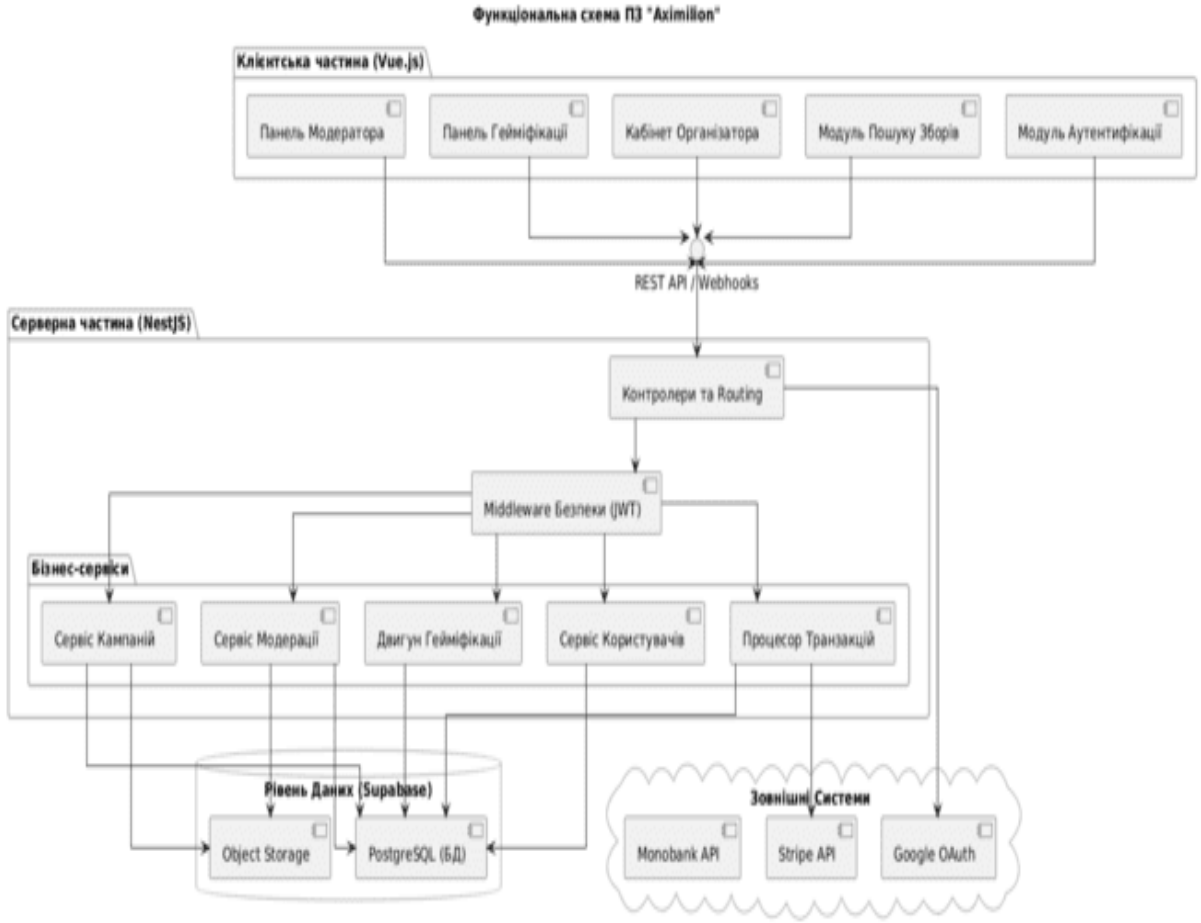

Рисунок 3.1 – Функціональна схема роботи програмного забезпечення для організації та підтримки благодійних зборів

При зверненні до вебплатформи користувач потрапляє на головну сторінку, де реалізовано механізми глобального пошуку та фільтрації активних зборів. Для доступу до персоналізованого функціоналу та гейміфікаційних

механік (нарахування стріків, колекціонування бейджів) користувач проходить автентифікацію через інтегрований модуль авторизації.

Алгоритм взаємодії із цільовою кампанією передбачає перехід до сторінки деталей збору, де система виконує асинхронний запит до БД для отримання актуального фінансового прогресу та верифікаційних документів. Здійснення пожертви ініціює перенаправлення на зовнішній платіжний шлюз Stripe або через посилання на «Банку» Монобанку. Після успішного завершення операції серверна частина через Webhooks отримує сповіщення та автоматично активує тригери нарахування ігрового досвіду в профілі благодійника.

Адміністративний сегмент програми передбачає окремий порядок роботи, де уповноважений актор здійснює перевірку черги нових кампаній та керує словником цифрових нагород.

3.2 Результати реалізації програмних одиниць застосунку

Проєктування сервера застосунків базується на двох основних принципах, а саме: високій модульності та чіткому розподілі функціональних обов'язків різних компонентів. Кореневий модуль застосунку, клас AppModule, функціонує як центральний вузол агрегації всіх функціональних блоків застосунку, що забезпечує їх взаємозв'язок. За допомогою декоратора @Module система NestJS здатна створювати метадані для створення ієрархічної інверсії керування. Використання цього методу дозволяє всім імпортованим модулям бути незалежними пісочницями (що містять їхню бізнес–логіку); проте, використовуючи систему впровадження залежностей, усі модулі можуть надавати свої послуги іншим частинам застосунку.

У структурі кореневого модуля ConfigModule займає особливе місце. Це перший модуль, який завантажується за допомогою методу forRoot(), а isGlobal має значення True. Це було важливе архітектурне рішення, оскільки воно дозволяє отримувати доступ до даних конфігурації (наприклад, змінних середовища, що використовуються для підключення до бази даних, або

секретних ключів, що використовуються для автентифікації) з будь–якої точки програми, а це означає, що не потрібно повторно імпортувати конфігурацію кілька разів у вашій програмі.

Далі в масиві import йдуть три модулі, які забезпечують основну функціональність платформи: DatabaseModule використовується для зв'язку з PostgreSQL та використовує Drizzle ORM для досягнення цієї мети; AuthModule та UsersModule використовуються для обробки інформації про безпеку та профіль користувача для осіб, які відвідують зустрічі. Три модулі, які надають благодійній платформі спеціалізовані можливості: CategoriesModule, MediaModule та PaymentsModule, використовуються для керування категоріями зустрічей, керування медіафайлами та обробки платежів/фінансових розрахунків відповідно. Щоб забезпечити покриття мотиваційного аспекту проєкту, Модуль значків (BadgesModule) та Модуль статистики (StatsModule) пов’язані між собою, де перший займається присудженням значків, винагород та значків, а другий - збором статистичних показників, пов’язаних з діяльністю донорів. Інший функціональний блок називається Модуль модератора (ModeratorModule), який функціонує разом зі своїм контролером, надаючи модератору набір інструментів, необхідних для автентифікації нових волонтерських ініціатив (лістинг 3.1).

Лістинг 3.1 – Реалізація базового модуля

```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { DatabaseModule } from './database/database.module';
import { AuthModule } from './auth/auth.module';
import { UsersModule } from './users/users.module';
import { ConfigModule } from '@nestjs/config';
import { CategoriesModule } from
'./categories/categories.module';
import { CampaignsModule } from
'./campaigns/campaigns.module';
import { MediaModule } from './media/media.module';
import { PaymentsModule } from './payments/payments.module';
import { BadgesModule } from './badges/badges.module';
import { StatsModule } from './stats/stats.module';
```

```
      import { ModeratorController } from
'./moderator/moderator.controller';
      import { ModeratorModule } from
'./moderator/moderator.module';

      @Module({
        imports: [
          ConfigModule.forRoot({
            isGlobal: true,
            envFilePath: '.env'
          }),
          DatabaseModule,
          AuthModule,
          UsersModule,
          CategoriesModule,
          CampaignsModule,
          MediaModule,
          PaymentsModule,
          BadgesModule,
          StatsModule,
          ModeratorModule,
        ],
        controllers: [AppController, ModeratorController],
        providers: [AppService],
      })
      export class AppModule {}
```

Гейміфікаційний підсистемний модуль для застосунку вбудований у загальну архітектуру як незалежний шар сервісів і взаємодіє з ядром системи через підтвердження транзакцій за допомогою асинхронних викликів. Ця логіка фактично реалізована в програмному забезпеченні в BadgeService, де метод checkAndAssignBadges виконує аналіз фінансової історії користувача, щоб нараховувати бейджі. Технічне рішення спирається на агрегатні функції Drizzle ORM (наприклад, count, sum), щоб сформувати лаконічний звіт про активність донатора щодо виконавця одним запитом: кількість успішних платежів, сумарні донатні кошти в межах валютних діапазонів і кількість унікальних підтриманих кампаній. Саме тому цей метод називають «демократичним» і, по суті, високопродуктивним (усі обробки відбуваються в межах бази даних PostgreSQL) (лістинг 3.2).

Лістинг 3.2 – Код перевірки та присвячення бейджу

```
async checkAndAssignBadges(supabaseUid: string) {
    const user = await this.db.query.users.findFirst({
      where: eq(schema.users.supabaseUid, supabaseUid)});
    if (!user) return;
    const internalUserId = user.id;
    const [stats] = await this.db
      .select({
        donationsCount: count(schema.payments.id),
        totalAmountCents: sum(schema.payments.amount),
        uniqueCampaignsCount:
countDistinct(schema.payments.campaignId),

}).from(schema.payments).where(and(eq(schema.payments.donorId,
supabaseUid), eq(schema.payments.status, 'success')));
    const donationsCount = stats?.donationsCount || 0;
    const totalAmount = (parseInt(stats?.totalAmountCents ||
'0', 10)) / 100;
    const uniqueCampaignsCount = stats?.uniqueCampaignsCount
    const badges = await this.db.query.badges.findMany();
    for (const badge of badges) {
    let earned = false;
    const threshold = parseFloat(badge.criteriaValue as any);
      switch (badge.criteriaType) {
        case 'donations_count':
          earned = donationsCount >= threshold; break;
        case 'total_amount':
          earned = totalAmount >= threshold; break;
        case 'unique_campaigns':
          earned = uniqueCampaignsCount >= threshold;break;}
      if (earned) {
        await this.db.insert(schema.userBadges).values({
          userId: internalUserId,
          badgeId: badge.id
        }).onConflictDoNothing();}}}
```

Клієнтська частина виконана в односторінковій формі застосунку. Дизайн складається з реактивних юнітів, побудованих за допомогою фреймворку Vue.js. Використовується метод декларативного програмування з Composition API, що дозволяє створювати складну логіку інтерфейсу користувача як пов'язані функції, які можна легко використовувати повторно. Основні точки користувацького досвіду (UX) (сторінка збору коштів та панелі інструментів користувача) використовують стандарт Single File Component. Таким чином, код TypeScript (логіка), декларативні шаблони та Tailwind CSS (стилі) об'єднані в один файл для кожного компонента. Повна синхронізація

структур даних між фронтендом та бекендом за допомогою TypeScript. Помилки через неправильні типи вхідних параметрів під час обробки результатів запиту API повністю уникаються.

Під час розробки було приділили багато уваги оптимізації відображення компонентів під час їх завантаження, а також стану цих компонентів у цей момент. Щоб забезпечити користувачам безперебійну навігацію та хорошу продуктивність, використовуються перехоплювачі життєвого циклу для контролю порядку всіх асинхронних процесів.

Крім того, функція отримання даних кампанії починає працювати, щойно компонент стає частиною дерева DOM. Таким чином, дані кампанії можна одночасно завантажувати як з локальних, так і з зовнішніх джерел даних. Внутрішня логіка програми включає суворий контроль дозволів доступу на рівні клієнта; якщо колекція перебуває в стані очікування, звичайним користувачам перегляд обмежено, і їх перенаправляють на сторінку помилки. Використання реактивної змінної для керування «станом завантаження» дозволяє динамічно керувати «екранами каркаса» або іншими індикаторами прогресу, щоб забезпечити кращий досвід для клієнтів разом із дуже низькою затримкою візуалізації найбільшого елемента (лістинг 3.3).

Лістинг 3.3 – Структура компонента CampaignView.vue

```
async function fetchCampaign() {
  isLoading.value = true;
  try {
    const id = route.params.id as string;
    const { data } = await api.get(`/campaigns/${id}`);
    if (data.status === 'pending' && authStore.user?.role !== 
'moderator') {
      router.push({ name: 'not-found' }); return;
    }
    campaign.value = data;
    totalCollected.value = parseFloat(data.collectedInternal 
|| '0');
  } catch (err: any) {
    console.error('Failed to fetch campaign:', err);
    if (err.response?.status === 404) {
      router.push({ name: 'not-found' });
    }} finally {
    isLoading.value = false;
```

```
        }}
      async function fetchDonations(loadMore = false) {
        if (!campaign.value) return;
        isLoadingDonations.value = true;
        try {
          const limit = 5;
          const offset = (currentPage.value - 1) * limit;
          const        {        data        }        =        await
api.get(`/campaigns/${campaign.value.id}/donations`, {
            params: { limit, offset }
          });
          if (loadMore) { donations.value = [...donations.value,
...data.donations];} else { donations.value = data.donations; }
          hasMore.value = data.hasMore;
        } catch (err) {
          console.error('Failed to fetch donations:', err);
        } finally {
          isLoadingDonations.value = false;}}
      onMounted(async () => {
        await fetchCampaign();
        await getDataFromMonobankJar();
        if (campaign.value) { await fetchDonations();}});
```

Керування станом на стороні клієнта базується на репозиторії Pinia та Composables (модульна функціональна архітектура), що дозволяє інкапсулювати процедури, пов'язані з обробкою сеансів. Спосіб синхронізації профілів між службою автентифікації та базою даних забезпечує гнучкість системи; тому з протоколом OAuth 2.0 можна використовувати декілька методів входу. Це досягається через спеціалізований контролер, який валідує вхідний токен та гарантує наявність актуального запису в реляційній структурі для кожного ідентифікованого суб'єкта.

Представлений метод синхронізації виконує логіку перевірки існування профілю за ідентифікатором та створює новий запис у разі його відсутності. Після створення нового запису в БД використовується метод returning, який дозволяє системі миттєво ініціалізувати внутрішні параметри користувача, такі як ролі та ігрові метрики, без додаткових запитів. Такий підхід забезпечує відповідність операції входу та стабільну роботу авторизації незалежно від обраного провайдера (лістинг 3.4).

Лістинг 3.4 – Код процесу синхронізації користувача між Supabase Authentication та БД

```
      import { Controller, Post, Body, UseGuards, Request } from
'@nestjs/common';
      import { AuthGuard } from '@nestjs/passport';
      import { AuthService } from './auth.service';
      @Controller('auth')
      export class AuthController {
       constructor(private readonly authService: AuthService) {}
       @UseGuards(AuthGuard('jwt'))
       @Post('sync')
       syncUser(@Request() req: any) {
         const { supabase_uid, email, full_name } = req.user;
         return    this.authService.syncSupabaseUser(supabase_uid,
email, full_name);}}
      async  syncSupabaseUser(supabaseUid:  string,  email:  string,
fullName: string | null) {
          let user = await this.db.query.users.findFirst({
            where: eq(schema.users.supabaseUid, supabaseUid),
          });
          if (!user) {
            const [newUser] = await this.db
              .insert(schema.users)
              .values({
                email: email,
                supabaseUid: supabaseUid,
                fullName: fullName,
                role: 'registered',
                authProvider: 'local',}).returning();
            user = newUser; }
          return user;}
```

Взаємодія з базою даних у проєкті реалізована за допомогою Drizzle ORM. Це дозволило описати структуру PostgreSQL безпосередньо мовою TypeScript, що значно спростило життя. Основна перевага такого підходу полягає в тому, що база даних і код тепер працюють як єдине ціле. Якщо буде потреба в раптовому зміненні назви колонки або типу даних, компілятор відразу "підсвітить" усі місця в коді, де виникла розбіжність. Це майже повністю виключає помилки, які зазвичай знаходять уже на етапі тестування. До того ж, використання типізованих запитів автоматично захищає застосунок від SQL–ін’єкцій, що важливо для фінансового сервісу.

Особливу увагу було приділено структурі таблиці кампаній. Вона спроєктована так, щоб зберігати не лише базову інформацію на кшталт назви

чи опису, а й складні структури даних, як–от масиви зображень у форматі jsonb чи специфічні статуси. Для грошових сум використан тип decimal з фіксованою точністю, адже в благодійності кожен цент має значення, а звичайні типи на кшталт float можуть давати похибки при великій кількості операцій. Зв'язки між сутностями, наприклад між волонтерами та їхніми зборами, прописані на рівні схеми, що дозволяє легко витягувати повні профілі користувачів разом із усією їхньою активністю (лістинг 3.5).

Лістинг 3.5 – Згенерована схема БД через Drizzle ORM

```
// --- Tables --
export const campaigns = pgTable('campaigns', {
  id: uuid('id').primaryKey().defaultRandom(),
  publisherId: uuid('publisher_id')
  references(() => users.id, { onDelete: 'cascade'
}).notNull(),
  categoryId: uuid('category_id')
  .references(() => categories.id, { onDelete: 'restrict'})
  .notNull(),
  title: varchar('title', { length: 255 }).notNull(),
  description: text('description').notNull(),
  goalAmount: decimal('goal_amount', { precision: 12, scale:
2 }).notNull(),
  currency:
currencyEnum('currency').default('USD').notNull(),
  collectedInternal: decimal('collected_internal', {
precision: 12, scale: 2 }).default('0').notNull(),
  monoJarUrl: varchar('mono_jar_url', { length: 255 }),
  images: jsonb('images').$type<{ url: string; type: string
}[]>(),
  legalProofUrl: text('legal_proof_url'),
  financialAuditUrl: text('financial_audit_url'),
  status:
campaignStatusEnum('status').default('pending').notNull(),
  endDate: timestamp('end_date'),
  publishedAt: timestamp('published_at'),
  createdAt: timestamp('created_at').defaultNow().notNull(),
  updatedAt: timestamp('updated_at').defaultNow().notNull(),
});
// --- Relations ---
export const usersRelations = relations(users,({ many })=> ({
  campaigns: many(campaigns),
  payments: many(payments),
  userBadges: many(userBadges),
}));
```

Опрацювання банківських даних потребує виняткової точності та високого рівня безпеки, через що інтеграції з Stripe та Monobank API стали фундаментом фінансового модуля. Головним завданням при розробці було забезпечення повної автономності процесів, аби виключити потребу в ручному перенесенні цифр чи підтвердженні кожної операції волонтерами. Архітектурна концепція передбачає самостійну взаємодію системи з банківським інтерфейсом для отримання актуального стану накопичувальних рахунків. Це створює необхідний рівень прозорості, що є критично важливим для будь–якого благодійного збору, оскільки донори мають можливість бачити відображення своїх внесків у реальному часі.

Одним із складних етапів реалізації цієї взаємодії стала обробка мультивалютних потоків. Оскільки платформа дозволяє встановлювати цілі в іноземних валютах, а внутрішні «банки» Monobank переважно працюють у гривні, впровадження логіки динамічної конвертації стало об'єктивною необхідністю. Алгоритм, наведений у лістингу 3.6, демонструє процес вилучення ідентифікатора з посилання на збір та подальше звернення до сервера для отримання поточного балансу. Програмний код коректно обробляє дрібні грошові одиниці та здійснює перерахунок за актуальним банківським курсом. Такий підхід гарантує точність прогрес–бару на сайті незалежно від валюти фактичного внеску.

Лістинг 3.6 – Отримання даних про «Банку» через Monobank API

```
async function getDataFromMonobankJar() {
 if(!campaign.value?.monoJarUrl) return;
 const jarId = campaign.value.monoJarUrl.split('/').pop();
 if(!jarId) return;
 try {
   const response = await api.get(`/payments/mono-
status/${jarId}`);
   const { jarAmount, currency, rate } = response.data;
   const amount = Number(jarAmount) / 100;
   let mappedCurrency = '';
   if (currency == 980) mappedCurrency = 'UAH';
   else if (currency == 840) mappedCurrency = 'USD';
   else if (currency == 978) mappedCurrency = 'EUR';
   let convertedJar = amount;
```

```
        const campaignCurrency = campaign.value.currency?.toUpperCase() || 'USD';
        if (mappedCurrency && mappedCurrency !== campaignCurrency && rate) { if (mappedCurrency === 'UAH' && campaignCurrency === 'USD') { convertedJar = amount / rate.usd.sale;
          } else if (mappedCurrency === 'USD' && campaignCurrency === 'UAH') {convertedJar = amount * rate.usd.buy;
          } else if (mappedCurrency === 'USD' && campaignCurrency === 'EUR') { convertedJar = (amount * rate.usd.buy) / rate.eur.sale;
          } else if (mappedCurrency === 'EUR' && campaignCurrency === 'USD') { convertedJar = (amount * rate.eur.buy) / rate.usd.sale; }}
        totalCollected.value = Number(campaign.value.collectedInternal || '0') + convertedJar;
      }catch (error) { console.error('Failed to get Monobank jar data:', error);}}
```

Ключовим компонентом можливостей платформи є алгоритмічна гейміфікація, де донат стає інтерактивним процесом взаємодії. Головним чином вона зосереджена на Day Streaks (розрахунок безперервної активності). Це свого роду інструмент «уникнення втрат», адже через необхідність щодня мати показник користувач змушений регулярно повертатися. Технічно це описує той самий принцип, який можна отримати в Octalysis. Логіка програми прив’язана до аналізу часових міток усіх успішних транзакцій благодійності, що дає змогу точно звузити вікно активності та своєчасно оновлювати статус ігрового профілю в показниках.

У рівні сервісу реалізовано алгоритм, який виконує агрегований SQL–запит за допомогою Drizzle ORM. Однією з критично важливих частин для того, щоб перевірки дат працювали коректно, є використання date_trunc на рівні БД, щоб нормалізувати час створення кожної транзакції до однієї конкретної частини повної доби, не враховуючи години й хвилини тощо. Потім унікальні дати впорядковуються у зворотному хронологічному порядку, що згодом дає змогу легко обчислити дельту між двома подіями. Цикл проходить список і перевіряє, чи розрив між суміжними днями дорівнює 1. Порушення цієї умови призводить до зупинки процесу підрахунку, що гарантує отримання фактичної безперервної серії (лістинг 3.7).

Лістинг 3.7 – Логіка розрахунку серії днів

```
        const paymentDates = await this.db
            .select({
              date:                    sql<Date>`date_trunc('day',
${schema.payments.createdAt})`
            }).from(schema.payments).where(and(
                eq(schema.payments.donorId, supabaseUid),
                eq(schema.payments.status,
'success'))).groupBy(sql`date_trunc('day',
${schema.payments.createdAt})`).orderBy(desc(sql`date_trunc('day'
, ${schema.payments.createdAt})`));
          let   streakDays   =   0;   const   today   =   new
Date();today.setHours(0, 0, 0, 0);
            if (paymentDates.length > 0) {
            let   lastDate   =   new   Date(paymentDates[0].date);
lastDate.setHours(0, 0, 0, 0);
      const   diffFromToday   =   Math.floor((today.getTime()   -
lastDate.getTime()) / (1000 * 60 * 60 * 24));
              if (diffFromToday <= 1) { streakDays = 1;
              for (let i = 1; i < paymentDates.length; i++) {
                const current = new Date(paymentDates[i].date);
                current.setHours(0,0,0,0);
                const prev = new Date(paymentDates[i-1].date);
                prev.setHours(0,0,0,0);
                const   gap   =   Math.floor((prev.getTime()   -
current.getTime()) / (1000 * 60 * 60 * 24));
                if (gap === 1) streakDays++; else break;}}}
```

3.3 Висновки за розділом 3

У ході виконання третього розділу було здійснено практичну реалізацію програмної платформи Aximilion. Результати розробки дозволяють сформулювати наступні висновки:

Програмний продукт розроблено у повному обсязі відповідно до функціональних вимог та архітектурних рішень, прийнятих на етапі проєктування. Усі ключові сценарії використання, включаючи реєстрацію, створення благодійних кампаній та обробку вхідних транзакцій, імплементовані та готові до експлуатації.

Використання модульної архітектури NestJS дозволило досягти високого рівня інкапсуляції бізнес–логіки та слабкої зв'язності компонентів системи. Впровадження політик безпеки на рівні рядків у Supabase PostgreSQL

забезпечило надійний захист чутливих фінансових даних та персональної інформації користувачів, гарантуючи цілісність даних на рівні СКБД.

Завдяки використанню Vue.js 3 та інтеграції з механізмами Real–time оновлень, реалізовано динамічний користувацький інтерфейс. Система забезпечує миттєве відображення змін у прогресі зборів та активності донорів без необхідності примусового оновлення сторінок, що значно покращує показники UX та залученість аудиторії.

Успішно реалізовано складні алгоритми розрахунку безперервних серій активності та автоматизовану систему видачі цифрових нагород. Програмна імплементація цих механік дозволяє ефективно керувати мотивацією благодійників, перетворюючи поодинокі внески на стійку соціальну звичку.

Підтверджено працездатність механізму взаємодії з Monobank API та Stripe через систему Webhooks. Реалізований ендпоінт забезпечує автоматизоване звітування та синхронізацію балансу «банок» із внутрішньою БД застосунку, що виключає людський фактор та підвищує прозорість волонтерської діяльності.

Таким чином, успішне завершення етапу програмної реалізації створює необхідні умови для переходу до наступного етапу роботи – комплексного тестування системи та оцінки її надійності в умовах, наближених до реальної експлуатації.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Програмний продукт розроблений для системного стимулювання благодійної активності та формування сталої звички донатити через інструменти гейміфікації. Застосунок дозволяє візуалізувати персональний соціальний вплив користувача, верифікувати прозорість волонтерських ініціатив через юридичну документацію, здійснювати моніторинг динаміки внесків та забезпечувати автоматизоване управління життєвим циклом краудфандингових кампаній із фіксацією транзакцій у реальному часі.

4.2 Умови виконання програми

Безперебійне функціонування платформи Aximilion безпосередньо залежить від активності адміністратора (модератора), який відповідає за валідацію вхідних даних та підтримку довіри до системи. До його ключових обов’язків належить верифікація юридичних документів волонтерів, премодерація нових благодійних кампаній, а також конфігурація параметрів гейміфікації (створення унікальних бейджів та налаштування драйверів стріків). Окрім контент–менеджменту, адміністратор забезпечує технічне розгортання та підтримку серверної частини.

Для стабільної роботи контейнеризованого застосунку та обробки запитів мінімальні параметри сервера мають включати:

– процесор: 4–ядерний vCPU, архітектура x86–64, частота від 2.4 ГГц;

– оперативна пам’ять: 8 ГБ ОЗП;

– накопичувач: 40 ГБ NVMe SSD.

Успішний запуск проєкту можливий лише за умови коректної конфігурації середовища, що включає:

– наявність файлу .env з валідними ключами доступу для клієнтського та серверного додатка;

– налаштований Google OAuth Client ID для авторизації;

– підготовлений Supabase для роботи з БД, зберігання файлів та аутентифікації;

– підлаштований Stripe акаунт для перехоплення Webhook запитів;

– встановлене середовище Docker та Docker Compose для ізоляції компонентів системи.

4.3 Виконання та тестування програми

Метою даного етапу є перевірка цілісності та працездатності інтегрованої системи, що включає клієнтську частину на Vue.js, серверне API на NestJS, реляційну БД PostgreSQL (Supabase) та зовнішні фінансові сервіси. Тестування проводиться шляхом виконання наскрізного (end–to–end) сценарію, який охоплює повний життєвий цикл благодійного збору – від ініціації волонтером до модерації та отримання ігрової нагороди донором.

Процес тестування починається з перевірки механізмів автентифікації. Користувачу пропонується два методи входу: класичний через пару Email/Password та інтегрований за допомогою Google OAuth 2.0 (рис. 4.1).

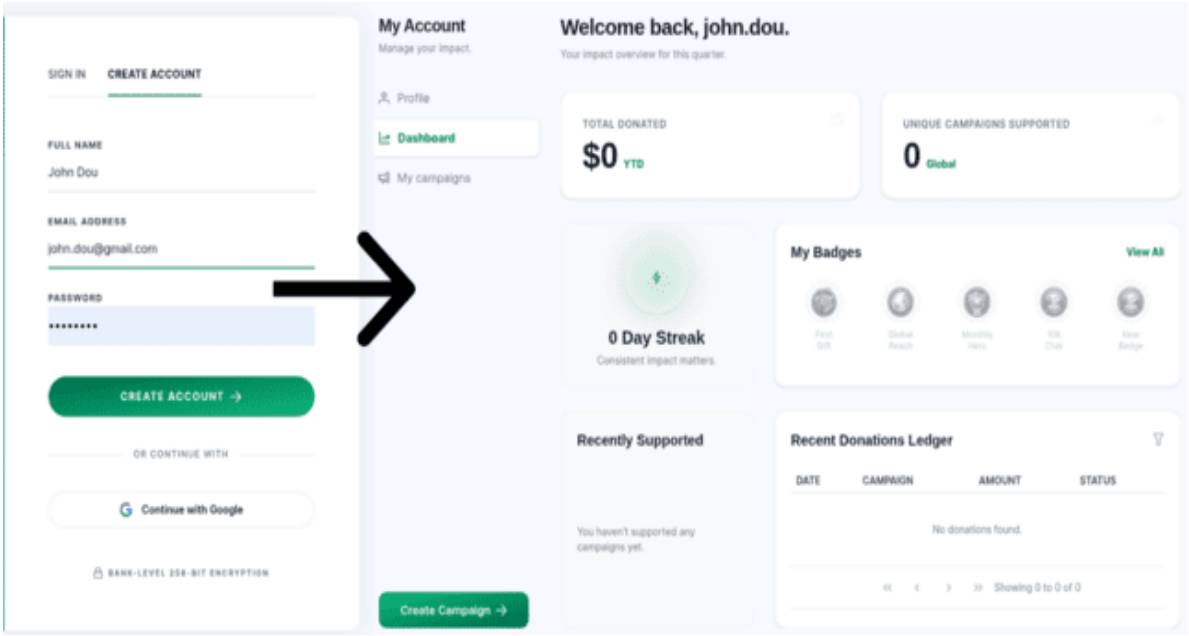

Рисунок 4.1 – Форма авторизації та візуалізація успішного входу в систему

Після успішного входу користувач переходить до публічного каталогу. Тестування цього етапу передбачає перевірку роботи фільтрів та динамічного рендерингу даних. При виборі категорії (наприклад, «Військова допомога») фронтенд надсилає запит GET /campaigns?category=military.

Система повинна миттєво відфільтрувати список, відображаючи картки зборів із актуальними прогрес–барами. При переході на сторінку конкретної кампанії перевіряється доступність верифікаційних документів (PDF–звітів) та юридичних підтверджень, що завантажені у Supabase Storage. Реактивні компоненти Vue.js забезпечують плавне завантаження медіа–контенту без перезавантаження всієї сторінки (рис. 4.2).

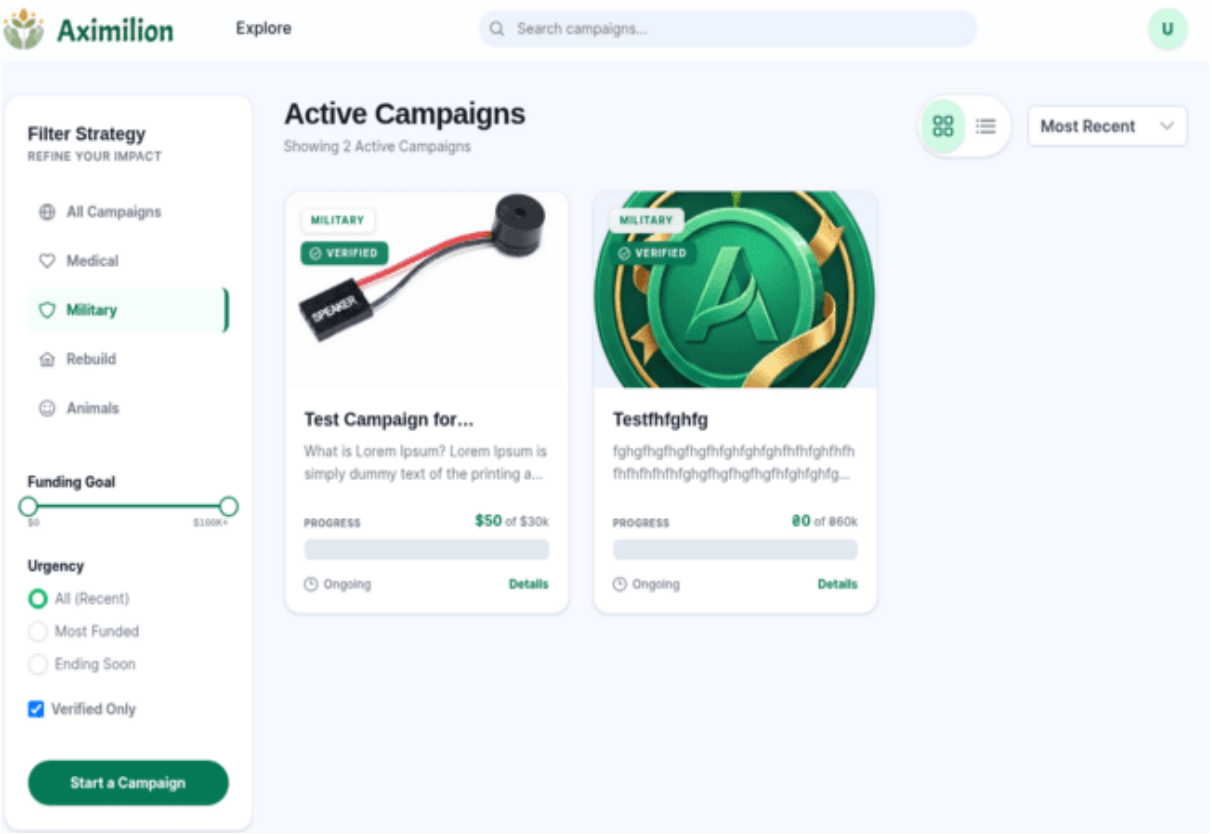

Рисунок 4.2 – Інтерфейс перегляду каталогу кампаній з активними фільтрами

Спроба проведення операції з підтримки кампанії. Користувач вказує суму пожертви та натискає кнопку «Donate Now», після цього з’являється модальне вікно де потрібно увести платіжну інформацію для здійснення операції (рис. 4.3).

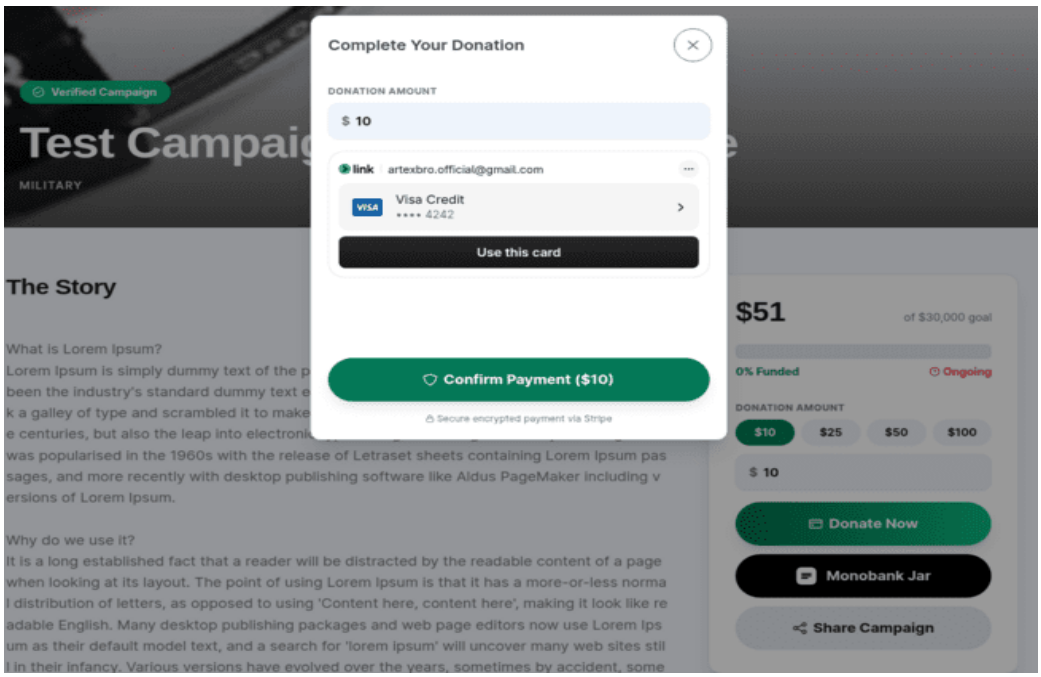

Рисунок 4.3 – Спроба зробити платіжну операцію з підтримки благодійного збора

Після того як операція буде вважатись успішною, рахунок кампанії поповниться на ту суму яка була вказана в транзакції та зможе побачити бейджі які він зміг отримати після проведення операції (рис. 4.4).

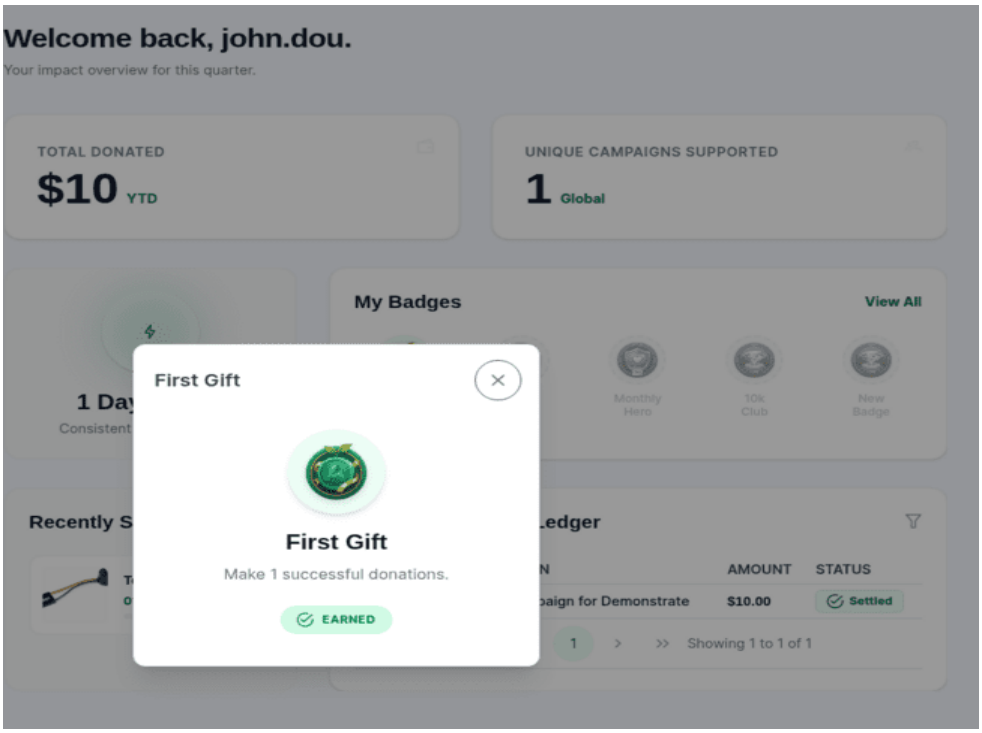

Рисунок 4.4 – Отриманий бейдж за першу успішну підтримку кампанії

Для перевірки функціоналу створення кампанії користувач переходить до форми ініціації збору. Тестується з процесом завантаження файлів типу .png як обкладинка та додаткові фотографії, також PDF–документи юридичного доказу та фінансового аудиту (рис. 4.5).

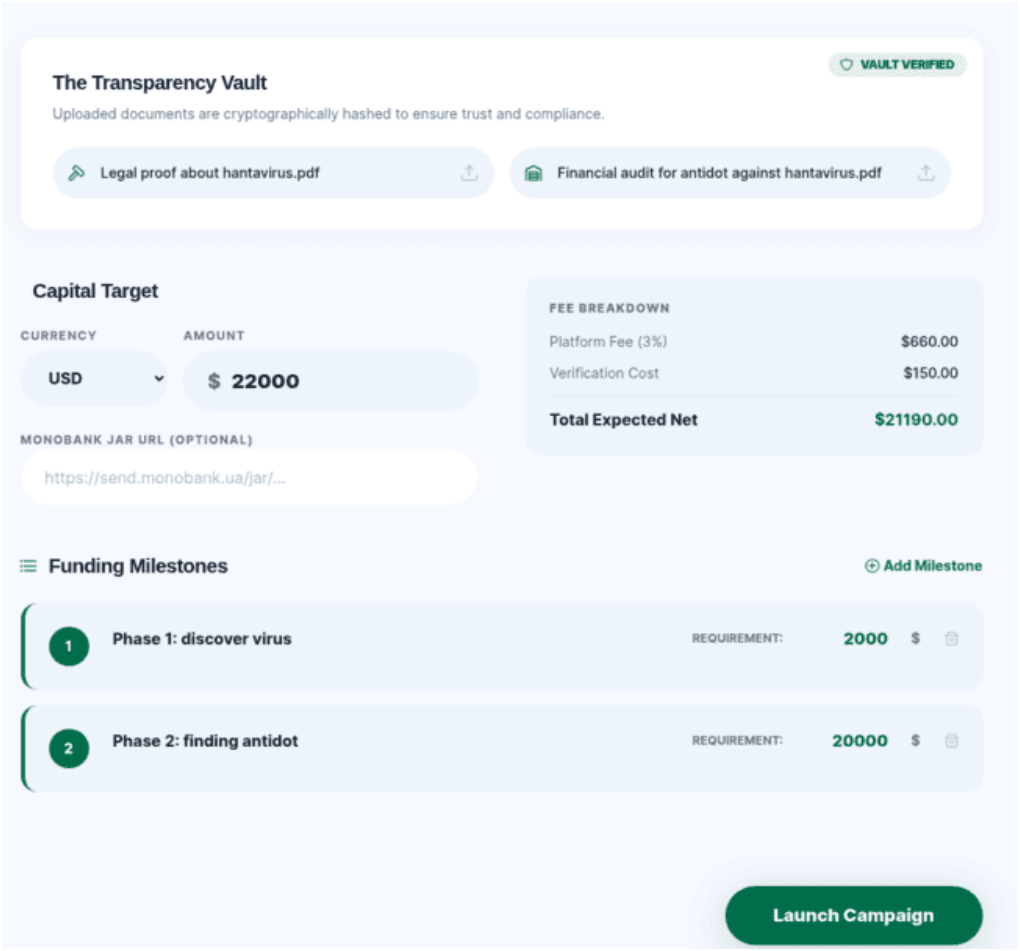

Рисунок 4.5 – Процес заповнення форми для створення збору

Для тестування адміністративного функціоналу виконується вхід під обліковим записом із роллю «Модератор». На рис. 4.6 зображена панель керування модератора, де відображається черга з кампаній зі статусом «pending».

Процес валідації включає перевірку коректності вказаних банківських реквізитів та автентичності завантажених документів. При натисканні кнопки «Approve» система робить запит на сервер з оновлення статусу кампаніїї на

«active». Це призводить до автоматичного спрацювання тригера, що робить кампанії доступною для пошуку та здійснення донатів.

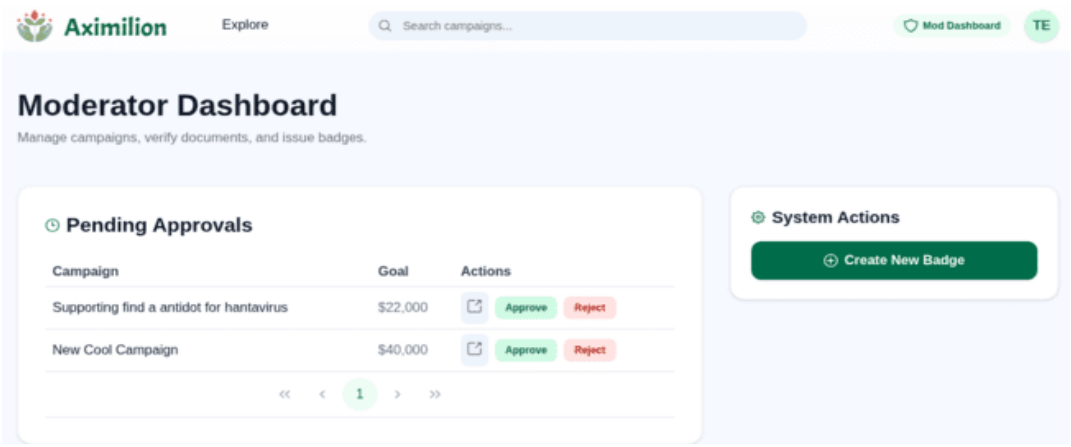

Рисунок 4.6 – Інтерфейс панелі модератора під час верифікації нової кампанії

Фінальний етап тестування, це перевірка дійсності підтведження кампанії у публічному каталозі. Для цього треба перейти до каталогу впевнетись у цьому. При знаходження нової кампанії робиться перевірка на коректні метадані: цільова сума, початковий баланс та верифікаційна помітка (рис. 4.7).

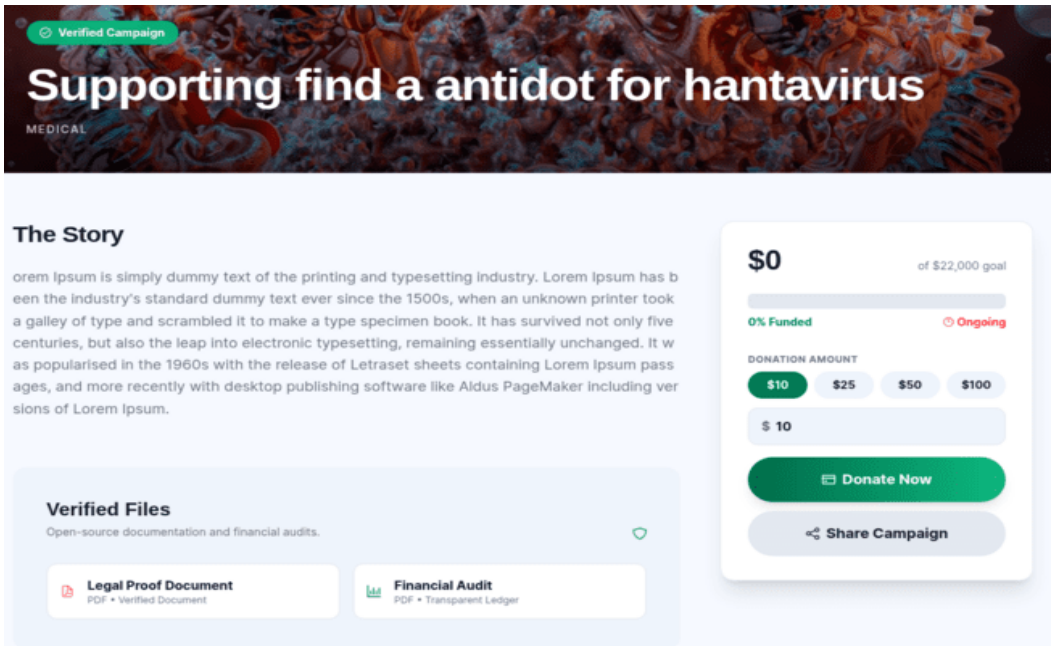

Рисунок 4.7 – Відображення верифікованої кампанії

4.4 Висновки за розділом 4

Під час тестування програмного забезпечення для організації та підтримки благодійних зборів, були виконанні перевірки на працездатність автентифікації користувачів, донату певної кампанії та отримання відповідний бейдж за успішну транзакцію. Був проведений процес зі створенням нової кампанії з проходженням верифікації від модератора та впевненням що після підтвердження кампанії, вона з’являється в публічному каталозі.

Застосунок працює справно, недоліків не знайдено.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи поставлена мета – скорочення витрат часу організаторів на адміністрування благодійних зборів та збільшення залученості користувачів була повністю досягнута шляхом проєктування та реалізації програмної платформи Aximilion.

Аналіз результатів дослідження та практичної розробки дозволяє сформулювати наступні висновки:

Впровадження автоматизованої системи обробки транзакцій через інтеграцію з Stripe та Monobank API дозволило мінімізувати рутинні операції волонтерів. Використання технології Webhooks забезпечило миттєву синхронізацію фінансових показників, що звільнило організаторів від необхідності ручної перевірки виписок та підготовки графічних звітів. Централізоване управління життєвим циклом кампаній через систему статусів дозволило структурувати процес модерації та скоротити час на верифікацію нових ініціатив.

Завдяки використанню хмарної інфраструктури Supabase для надійного зберігання та швидкого доступу до юридичних документів і фінансових аудитів, було створено середовище з високим рівнем верифікації контенту. Це дозволило трансформувати суб’єктивну довіру до організатора у технічно обґрунтовану прозорість, де кожен донат підкріплюється системним звітом у реляційній БД.

Програмна реалізація системи гейміфікації, що базується на алгоритмах розрахунку безперервних серій та автоматизованій видачі цифрових нагород, виступила ефективним технічним інструментом підвищення показника утримання користувачів. Аналіз логіки взаємодії підтверджує, що візуалізація прогресу та персональна система досягнень створюють додаткову мотивацію до повторних внесків, перетворюючи благодійність на стійку соціальну звичку.

Технологічна валідність та стабільність. Обраний технологічний стек, що включає NestJS для серверної логіки, Vue.js для реактивного інтерфейсу та

Drizzle ORM для типізованої взаємодії з БД, забезпечив високу швидкість розробки та стабільність програмного продукту. Результати експериментального дослідження та наскрізного тестування дозволяють стверджувати, що всі алгоритми фінансового моніторингу та мотиваційного дизайну функціонують коректно, забезпечуючи відповідність операцій та цілісності даних.

Розроблена платформа є функціонально завершеним продуктом, готовим до експлуатації в реальних умовах благодійного та волонтерського секторів. Впровадження системи дозволяє не лише масштабувати волонтерські ініціативи, а й значно підвищити якість взаємодії між ініціаторами зборів та благодійниками завдяки сучасним підходам інженерії програмного забезпечення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Kalil T., Kanwar R. Philanthropy 2.0: What the Evolution of For-Profit Investment Tells Us About the Future of Giving. *Sci Phi*. URL: https://renaissancephilanthropy.substack.com/p/philanthropy-20-what-the-evolution (date of access: 10.04.2026).

2. Boyd C. Micro-Donations: Why Small, Frequent Gifts Are Changing Fundraising. *Txt2give*. URL: https://txt2give.co/information-center/micro-donations-are-changing-fundraising/ (date of access: 10.04.2026).

3. Дослідження сфери благодійності. *Zagoriy Foundation.* URL: https://zagoriy.foundation/wp-content/uploads/2025/12/doslidzhennya_sektoru_blagodijnosti_2025.pdf (дата звернення: 10.04.2026).

4. Шевкопляс І. Пожиттєва цінність клієнта (LTV). *SendPulse*. URL: https://sendpulse.ua/support/glossary/customer-lifetime-value (дата звернення: 11.04.2026).

5. Дослідження сфери благодійності в Україні 2024 у цифрах, інсайтах та висновках. *Zagoriy Foundation.* URL: https://zagoriy.foundation/researches/doslidzhennya-sferi-blagodijnosti-v-ukrayini-2024-u-cifrax-insajtax–ta-visnovkax/ (дата звернення: 11.04.2026).

6. Das S. 10 Best Personal Fundraising Websites for Individuals (Tested & Compared 2026). URL: https://whydonate.com/uk/blog/platformy-dlya-zboru-koshtiv-dzboru-hroshey/ (date of access: 11.04.2026).

7. Про платформу. *dobro ua*. URL: https://dobro.ua/ (дата звернення 11.04.2026).

8. About Us. *GoFundMe.* URL: https://www.gofundme.com/c/about-us (date of access: 13.04.2026).

9. About Patreon. *Patreon.* URL: https://www.patreon.com/uk-UA/about (date of access: 13.04.2026).

10. Про цифровий банкінг Monobank. *Monobank.* URL: https://monobank.ua/ (дата звернення 13.04.2026).

11. The Octalysis Framework for Gamification & Behavioral Design. *Yu–kai Chou.* URL: https://yukaichou.com/gamification-examples/octalysis-gamification-framework/ (date of access: 14.04.2026).

12. TypeScript: JavaScript With Syntax For Types. *Microsoft.* URL: https://www.typescriptlang.org/ (date of access: 19.04.2026).

13. Design Patterns in TypeScript. *Refactoring Guru*. URL: https://refactoring.guru/design-patterns/typescript (date of access: 19.04.2026).

14. Jones A. NestJS: The Perfect Javascript Backend Framework for Structure, Clean Code, and Modularity. *Medium.* URL: https://medium.com/@ajonesb/nestjs-the–perfect-javascript-backend-framework-for-structure-clean-code-and-modularity-ae1b3a6e1418 (date of access: 19.04.2026).

15. Drizzle ORM. *Drizzle*. URL: https://orm.drizzle.team/ (date of access: 20.04.2026).

16. Supabase: The Open Source Firebase Alternative. *Supabase*. URL: https://supabase.com/ (date of access: 20.04.2026).

ДОДАТОК А

Технічне завдання

Вступ

Благодійність та колективний збір коштів стали невід'ємною частиною суспільного життя. Значущість цього явища важко переоцінити, адже щодня фінансуються тисячі ініціатив. Про вкорінення цієї галузі свідчить швидка адаптація фінансового сектору, який створює зручні платіжні інструменти на кшталт цифрових «банок», якими користуються мільйони людей. Зараз підтримка краудфандингових ініціатив відбувається переважно на таких рівнях: через розповсюдження прямих платіжних посилань у соціальних мережах, за допомогою монолітних платформ великих фондів, через персональні сторінки волонтерів, які ведуть облік транзакцій вручну. Незважаючи на розвиток технічної бази для переказу грошей, поза увагою залишається критично важливий аспект – психологія та утримання мотивації донора. Здебільшого процес зводиться до одноразової транзакції: людина переказує кошти і просто закриває сторінку. Через відсутність інтерактивного зворотного зв'язку благодійність часто залишається лише епізодичним імпульсом, а не регулярною звичкою. До того ж, організатори витрачають значну кількість часу на ручне зведення статистики, замість того щоб працювати зі своєю аудиторією. Сучасні користувачі добре реагують на механіки гейміфікації, які успішно працюють у навчальних чи фітнес–додатках. Створення подібного середовища в сфері благодійних зборів має потенціал кардинально змінити поведінку благодійників. Планується, що інтеграція механіки «стріків», тобто відстеження серій послідовних днів із пожертвами, це стимулюватиме користувача не переривати свій ланцюжок добрих справ, формуючи стійку звичку. А система накопичувальних бейджів, які видаватимуться за підтримку різних регіонів чи регулярність, дозволить візуалізувати індивідуальний вплив людини та підвищити її статус. Саме на проєктуванні та розробці таких мотиваційних інструментів у поєднанні з автоматизацією звітів планується зробити акцент у даній роботі.

А.1 Підстави для розробки

Дипломна кваліфікаційна робота виконувалась у відповідності до наказу №139 від 07 квітня 2026 р. за Національним університетом «Запорізька політехніка», яким зазначено тему роботи – «Розробка програмного забезпечення для організації та підтримки благодійних зборів».

А.2 Призначення розробки

Програмний продукт призначений для системного стимулювання благодійної активності та формування сталої звички донатити через інструменти гейміфікації. Застосунок дозволяє візуалізувати персональний соціальний вплив користувача, верифікувати прозорість волонтерських ініціатив через юридичну документацію, здійснювати моніторинг динаміки внесків та забезпечувати автоматизоване управління життєвим циклом краудфандингових кампаній із фіксацією транзакцій у реальному часі.

А.3 Основні вимоги до програми

А.3.1 Вимоги до функціональних характеристик

Функціональні характеристики програмного забезпечення для організації та підтримки благодійних зборів повинні включати:

- авторизація на платформі через email та пароль;
- авторизація на платформі через google аккаунт;
- переглядати статистику платформи;
- переглянути кампанії по певній категорії;
- переглядати верифіковані кампанії;
- переглядати трендові кампанії;
- переглядати останні донати по платформі;
- переглянути список усіх активних кампаній;

– відфільтрувати активні кампанії за категорією, проміжком між необхідної суми кампанії, терміном закінчення кампанії;

– відсортувати за терміном закінчення кампанії;

– переглянути певну кампанію;

– переглянути прикріплені зображення до кампанії;

– переглянути верифікований юридично доказовий документ;

– переглянути верифікований фінансовий аудит;

– скопіювати посилання на кампанію;

– перейти по посиланню «банки» monobank, якщо то прикріплено;

– зробити пожертву за запропонованими варіантами, або вписати певну суму;

– сплатити пожертву через платіжну систему stripe;

– переглядати етапи кампанії;

– переглядати останні пожертви на обрану кампанію;

– шукати певну кампанію по ключовим словам;

– оновлювати персональну інформацію профіля;

– оновлювати пароль аккаунту;

– переглянути статистику по сумі пожертв;

– переглянути статистику по кількості підтримань унікальних кампаній;

– переглянути серію днів коли робились пожертви;

– переглянути усі отримані бейджі;

– переглянути інформацію по певному бейджу;

– переглянути усі доступні бейджі;

– переглянути нещодавно підтримані кампанії;

– переглянути нещодавні персональні операції;

– переглянути персональні кампанії;

– створити нову кампанію;

– відкривати дешбоард модератора;

– переглядати кампанії які є в статусі очікування;

– підтверджувати кампанію та переводити її в статус «активної»;
– відхиляти кампанію та переводити її в статус «відхиленої»;
– створювати нові бейджі.

А.3.2 Умови експлуатації

Обов’язковою умовою експлуатацією програмного забезпечення для організації та підтримки є наявність модератора, який відповідає за валідацію вхідних даних та підтримку довіри до системи. До його ключових обов’язків належить верифікація юридичних документів волонтерів, премодерація нових краудфандингових кампаній, а також конфігурація параметрів гейміфікації, в точності, створення унікальних бейджів та налаштування драйверів стріків. Окрім контент–менеджменту, адміністратор забезпечує технічне розгортання та підтримку серверної частини.

А.3.3 Вимоги до складу та параметрів технічних засобів

Для стабільної роботи контейнеризованого застосунку та обробки запитів мінімальні параметри сервера мають включати:
– процесор: 4–ядерний vCPU, архітектура x86–64, частота від 2.4 ГГц;
– оперативна пам’ять: 8 ГБ ОЗП;
– накопичувач: 40 ГБ NVMe SSD.

А.4 Порядок контролю та приймання

Починаючи з дати видачі завдання – 7 квітня, протягом наступних 5 тижнів має бути забезпечено виконання роботи за календарним планом, що містить в завданні на роботу. Календарний план визначає послідовність етапів виконання роботи та за кожним таким етапом – строк виконання і основний очікуваний результат. Після забезпечення цих очікуваних результатів

формується пояснювальна записка до дипломної кваліфікаційної роботи. Остаточним кроком приймання роботи є її публічний захист.

ДОДАТОК Б

Текст програми

```
import { Module } from '@nestjs/common';
import { CampaignsService } from './campaigns.service';
import { CampaignsController } from './campaigns.controller';
import { DatabaseModule } from '../database/database.module';
import { UsersModule } from '../users/users.module';

@Module({
  imports: [DatabaseModule, UsersModule],
  controllers: [CampaignsController],
  providers: [CampaignsService],
  exports: [CampaignsService],
})
export class CampaignsModule {}
@Controller('campaigns')
export class CampaignsController {
  constructor(
    private readonly campaignsService: CampaignsService,
    private readonly usersService: UsersService,
  ) {}

  @Get()
  findAll(@Query() filters: FilterCampaignsDto) {
    return this.campaignsService.findFiltered(filters);
  }



  @Get('search')
  search(@Query('q') q: string) {
    return this.campaignsService.search(q);
  }

  /**
   * GET /campaigns/:id
   * Returns a single campaign with milestones and category.
   */
  @Get(':id')
  async findOne(@Req() req: any, @Param('id') id: string) {
    let requestingUser = null;
    const authHeader = req.headers?.authorization;
    if (authHeader && authHeader.startsWith('Bearer ')) {
      const token = authHeader.split(' ')[1];
      const { createClient } = require('@supabase/supabase-js');
      const supabase = createClient(process.env.SUPABASE_URL || '',
process.env.SUPABASE_SECRET_KEY || '');
      const { data: { user } } = await supabase.auth.getUser(token);
      if (user) {
        requestingUser                    =                    await
this.usersService.findBySupabaseUid(user.id);
      }
    }
    return this.campaignsService.findById(id, requestingUser);
  }
```

```
 async create(publisherId: string, data: any) {

    return this.db.transaction(async (tx) => {
      // 1. Insert the campaign
      const [newCampaign] = await tx
        .insert(schema.campaigns)
        .values({
          publisherId,
          categoryId: data.categoryId,
          title: data.title,
          description: data.description,
          goalAmount: String(data.goalAmount),
          currency: data.currency || 'USD',
          monoJarUrl: data.monoJarUrl,
          images: data.images || null,
          legalProofUrl: data.legalProofUrl || null,
          financialAuditUrl: data.financialAuditUrl || null,
          status: 'pending',
        })
        .returning();

      // 2. Insert milestones if provided
      let                createdMilestones:                (typeof
schema.milestones.$inferSelect)[] = [];
      if (data.milestones && data.milestones.length > 0) {
        createdMilestones = await tx
          .insert(schema.milestones)
          .values(
            data.milestones.map((m: any, index: number) => ({
              campaignId: newCampaign.id,
              title: m.title,
              amount: String(m.amount),
              sortOrder: index,
            })),
          )
          .returning();
      }

      return { ...newCampaign, milestones: createdMilestones };
    });
  }
export class CreateCampaignDto {
  @IsString()
  @IsNotEmpty({ message: 'Title must not be empty' })
  title: string;

  @IsString()
  @IsNotEmpty()
  categoryId: string;

  @IsString()
  @IsNotEmpty()
  description: string;
```

```
  @IsNumber()
  @IsPositive({ message: 'Goal amount must be a positive number' })
  goalAmount: number;

  @IsOptional()
  @IsString()
  @IsIn(['USD', 'EUR', 'UAH'])
  currency?: string;

  @IsOptional()
  @IsString()
  monoJarUrl?: string;

  @IsOptional()
  @IsArray()
  @ValidateNested({ each: true })
  @Type(() => ImageDto)
  images?: ImageDto[];

  @IsOptional()
  @IsString()
  legalProofUrl?: string;

  @IsOptional()
  @IsString()
  financialAuditUrl?: string;

  @IsOptional()
  @IsArray()
  @ValidateNested({ each: true })
  @Type(() => MilestoneDto)
  milestones?: MilestoneDto[];
}
```

```
 async handleWebhook(signature: string, payload: Buffer) {
    let event: Event;
    try {
      event = this.stripe.webhooks.constructEvent(
        payload,
        signature,
        process.env.STRIPE_WEBHOOK_SECRET as string
      );
    } catch (err: any) {
      throw      new      BadRequestException(`Webhook      Error:
${err.message}`);
    }

    if (event.type === 'payment_intent.succeeded') {
      const paymentIntent = event.data.object as PaymentIntent;
      await this.processSuccessfulPayment(paymentIntent);
    }
```

```
    return { received: true };
  }

  private async processSuccessfulPayment(intent: PaymentIntent) {

    const [payment] = await this.db
      .select()
      .from(schema.payments)
      .where(eq(schema.payments.stripePaymentIntentId, intent.id))
      .limit(1);

    if (!payment || payment.status === 'success') return;

    await this.db
      .update(schema.payments)
      .set({ status: 'success' })
      .where(eq(schema.payments.id, payment.id));

    const campaign = await this.db.query.campaigns.findFirst({
      where: eq(schema.campaigns.id, payment.campaignId),
    });

    if (campaign) {
      const amountInCurrency = payment.amount / 100;

      const newTotal = (parseFloat(campaign.collectedInternal as
string) + amountInCurrency).toFixed(2);

      await this.db.update(schema.campaigns)
        .set({ collectedInternal: newTotal })
        .where(eq(schema.campaigns.id, payment.campaignId));
    }

    if (payment.donorId) {
      await
this.badgesService.checkAndAssignBadges(payment.donorId);
    }
  }
 async getMonobankJarStatus(jarId: string) {
  try {
    const              response              =              await
this.httpService.axiosRef.post('https://send.monobank.ua/api/hand
ler', {
      "Pc":
"BAg2rqDuXLHdmGMSozSifeYs62LNrlauyqYnchg2r3ms+W6zCKLh/mr/vGqm+/Um
8LIGq6Ylcqb+VMquLb5u1pM=",
      "c": "hello",
      "clientId": jarId
    }, {
      headers: {
        "Content-Type": "application/json",
        "Accept": "application/json",
        "User-Agent": "PostmanRuntime/7.53.0",
```

```
      }
    });

    return response.data;
  } catch (error) {
    throw new BadRequestException('Failed to fetch data from
Monobank');
  }
 async create(data: any) {
    const [newUser] = await this.db
      .insert(schema.users)
      .values({
        email: data.email,
        role: 'registered',
        authProvider: 'local',
      })
      .returning();
    return newUser;
  }
  async syncSupabaseUser(supabaseUid: string, email: string,
fullName: string | null) {
    let user = await this.db.query.users.findFirst({
      where: eq(schema.users.supabaseUid, supabaseUid),
    });

    if (!user) {
      const [newUser] = await this.db
        .insert(schema.users)
        .values({
          email: email,
          supabaseUid: supabaseUid,
          fullName: fullName,
          role: 'registered',
          authProvider: 'local',
        })
        .returning();
      user = newUser;
    }

    return user;
  }
@Injectable()
export class StatsService {
  constructor(
    @Inject(DATABASE_CONNECTION)
    private readonly db: NodePgDatabase<typeof schema>,
  ) {}

  async getPlatformStats() {

    const [donorsResult, campaignsResult, fundsResult] = await
Promise.all([
      this.db
```

```
        .select({ count: count() })
        .from(schema.users)
        .where(eq(schema.users.role, 'registered')),
      this.db
        .select({ count: count() })
        .from(schema.campaigns)
        .where(or(eq(schema.campaigns.status,           'active'),
eq(schema.campaigns.status, 'closed'))),
      this.db
        .select({ totalAmount: sum(schema.payments.amount) })
        .from(schema.payments)
        .where(eq(schema.payments.status, 'success'))
    ]);

    const totalDonors = donorsResult[0]?.count || 0;
    const activeCampaigns = campaignsResult[0]?.count || 0;

    const totalFundsRaised = (Number(fundsResult[0]?.totalAmount)
|| 0) / 100;

    return {
      totalDonors,
      activeCampaigns,
      totalFundsRaised,
    };
  }
}
@Injectable()
export class StatsService {
  constructor(
    @Inject(DATABASE_CONNECTION)
    private readonly db: NodePgDatabase<typeof schema>,
  ) {}

  async getPlatformStats() {

    const [donorsResult, campaignsResult, fundsResult] = await
Promise.all([
      this.db
        .select({ count: count() })
        .from(schema.users)
        .where(eq(schema.users.role, 'registered')),
      this.db
        .select({ count: count() })
        .from(schema.campaigns)
        .where(or(eq(schema.campaigns.status,           'active'),
eq(schema.campaigns.status, 'closed'))),
      this.db
        .select({ totalAmount: sum(schema.payments.amount) })
        .from(schema.payments)
        .where(eq(schema.payments.status, 'success'))
    ]);
```

```
    const totalDonors = donorsResult[0]?.count || 0;
    const activeCampaigns = campaignsResult[0]?.count || 0;
    const totalFundsRaised = (Number(fundsResult[0]?.totalAmount)
|| 0) / 100;

    return {
      totalDonors,
      activeCampaigns,
      totalFundsRaised,
    };
  }
}
import { Injectable, CanActivate, ExecutionContext } from
'@nestjs/common';
import { Reflector } from '@nestjs/core';
import { UsersService } from '../users/users.service';
import { ROLES_KEY } from './roles.decorator';

@Injectable()
export class RolesGuard implements CanActivate {
  constructor(private reflector: Reflector, private usersService:
UsersService) {}

  async canActivate(context: ExecutionContext): Promise<boolean> {
    const requiredRoles =
this.reflector.getAllAndOverride<string[]>(ROLES_KEY, [
      context.getHandler(),
      context.getClass(),
    ]);

    if (!requiredRoles) {
      return true;
    }

    const { user } = context.switchToHttp().getRequest();
    if (!user || !user.supabase_uid) {
      return false;
    }

    const dbUser = await
this.usersService.findBySupabaseUid(user.supabase_uid);
    if (!dbUser) return false;

    return requiredRoles.includes(dbUser.role);
  }
}
async createPaymentIntent(dto: { campaignId: string; amount:
number; currency?: string, userId: string | null }) {
    const amountInCents = Math.round(dto.amount * 100);

    const paymentIntent = await
this.stripe.paymentIntents.create({
      amount: amountInCents,
```

```
      currency: dto.currency ?? 'usd',
      metadata: {
        campaignId: dto.campaignId,
        userId: dto.userId ?? 'anonymous',
      },
    });

    const [payment] = await this.db
      .insert(schema.payments)
      .values({
        donorId: dto.userId ?? null,
        campaignId: dto.campaignId,
        amount: amountInCents, // Save in cents
        currency: dto.currency ?? 'usd',
        status: 'pending',
        stripePaymentIntentId: paymentIntent.id,
      })
      .returning();

    // Return clientSecret and paymentId
    return {
      clientSecret: paymentIntent.client_secret,
      paymentId: payment.id,
    };
  }
import { createClient } from '@supabase/supabase-js'

const supabaseUrl = import.meta.env.VITE_SUPABASE_URL ||
'https://dummy.supabase.co'
const supabasePublishableKey =
import.meta.env.VITE_SUPABASE_PUBLISHABLE_KEY || 'dummy-
publishable-key'

export const supabase = createClient(supabaseUrl,
supabasePublishableKey)

import axios from 'axios';

import { supabase } from '../lib/supabase';

const api = axios.create({
  baseURL: import.meta.env.VITE_API_URL,
});

api.interceptors.request.use(
  async (config) => {
    const { data } = await supabase.auth.getSession();
    if (data.session && config.headers) {
      config.headers.Authorization = `Bearer
${data.session.access_token}`;
    }
    return config;
  },
```

```
  (error) => {
    return Promise.reject(error);
  }
);

export default api;

// --- State ---
const campaigns = ref<Campaign[]>([]);
const isLoading = ref(true);
const selectedCategory = ref(DEFAULT_CATEGORY_ID);

const categories = ref<Category[]>([
  { id: DEFAULT_CATEGORY_ID, name: 'All', iconPrefix: 'pi pi-globe'
}
]);

const latestDonations = ref<Donation[]>([]);
const isLoadingDonations = ref(true);

const platformStats = ref<PlatformStats>({ totalDonors: 0,
activeCampaigns: 0, totalFundsRaised: 0 });
const isLoadingStats = ref(true);

function getCoverImage(campaign: any): string {
  if (campaign.image) return campaign.image; // fallback data
  const imgs = campaign.images as CampaignImage[] | null;
  const cover = imgs?.find((i) => i.type === 'cover');
  return cover?.url || '';
}

function currencySymbol(c: any): string {
  return currencyMap[c?.currency] || '$';
}

function formatAmount(val: string | number): string {
  const n = typeof val === 'string' ? parseFloat(val) : val;
  if (!n || isNaN(n)) return '0';
  return n.toLocaleString('en-US', { maximumFractionDigits: 0 });
}

function formatCompactNumber(num: number): string {
  if (num >= 1000000) {
    return (num / 1000000).toFixed(1).replace(/\.0$/, '') + 'M';
  }
  if (num >= 1000) {
    return (num / 1000).toFixed(1).replace(/\.0$/, '') + 'K';
  }
  return num.toString();
}

function progressPercent(c: any): number {
  const collected = parseFloat(c.collectedInternal || '0');
```

```
  const goal = parseFloat(c.goalAmount || '1');
  return Math.min(100, Math.round((collected / goal) * 100));
}

function getDaysLeft(c: any): string {
  if (c.daysLeft) return `${c.daysLeft} Days Left`;
  if (!c.endDate) return 'Ongoing';
  const end = new Date(c.endDate);
  const now = new Date();
  const diff = Math.ceil((end.getTime() - now.getTime()) / (1000 *
60 * 60 * 24));
  return diff > 0 ? `${diff} Days Left` : 'Ended';
}

function getCategoryName(categoryId: string): string {
  const cat = categories.value.find(c => c.id === categoryId);
  return cat ? cat.name : 'General';
}

function timeAgo(dateString: string): string {
  const date = new Date(dateString);
  const now = new Date();
  const seconds = Math.floor((now.getTime() - date.getTime()) /
1000);

  if (seconds < 60) return 'just now';
  const minutes = Math.floor(seconds / 60);
  if (minutes < 60) return `${minutes} minute${minutes !== 1 ? 's'
: ''} ago`;
  const hours = Math.floor(minutes / 60);
  if (hours < 24) return `${hours} hour${hours !== 1 ? 's' : ''}
ago`;
  const days = Math.floor(hours / 24);
  return `${days} day${days !== 1 ? 's' : ''} ago`;
}


    allBadges.value = data.map((b: any) => ({
      id: b.id,
      name: b.name,
      icon_url: b.icon_url ?? b.iconUrl,
      criteria_type: b.criteria_type ?? b.criteriaType,
      criteria_value: b.criteria_value ?? b.criteriaValue,
    }));
  } catch (err) {
    console.error('Failed to load all badges', err);
  } finally {
    loadingAllBadges.value = false;
  }
};

const openAllBadges = () => {
  isBadgesModalVisible.value = true;
```

```
  fetchAllBadges();
};

const getBadgeDescription = (badge: any) => {
  if (!badge) return '';
  // Support both snake_case (dashboard-stats) and camelCase (GET
/badges)
  const val = Number(badge.criteria_value ?? badge.criteriaValue);
  const type = badge.criteria_type ?? badge.criteriaType;
  switch(type) {
    case 'donations_count': return `Make ${val} successful
donations.`;
    case 'total_amount': return `Donate a cumulative total of
$${val.toLocaleString()}.`;
    case 'unique_campaigns': return `Support ${val} distinct
campaigns.`;
    default: return 'Complete the hidden requirement to earn this
badge.';
  }
};

const donations = ref<any[]>([]);
const totalDonations = ref(0);

const fetchStats = async () => {
  loadingStats.value = true;
  try {
    const { data } = await api.get('/users/me/dashboard-stats');
    totalDonated.value = data.totalDonated || 0;
    uniqueCampaigns.value = data.uniqueCampaignsCount || 0;
    streakDays.value = data.streakDays || 0;
    recentlySupported.value = data.recentlySupported || [];

    badges.value = data.badges?.map((b: any) => ({
      ...b,
      name: b.name.replace(' ', '<br/>'),
    })) || [];
  } catch (err) {
    console.error('Failed to load stats', err);
  } finally {
    loadingStats.value = false;
  }
};

const fetchDonations = async (limit = 5, offset = 0) => {
  loadingDonations.value = true;
  try {
    const { data } = await api.get('/users/me/donations', {
      params: { limit, offset }
    });
    donations.value = data.data;
    totalDonations.value = data.total;
  } catch (err) {
```

```
    console.error('Failed to load donations', err);
  } finally {
    loadingDonations.value = false;
  }
};

const onPage = (event: any) => {
  fetchDonations(event.rows, event.first);
};

onMounted(() => {
  fetchStats();
  fetchDonations();
});
```

ДОДАТОК В

Слайди презентації

Рисунок В.1 – Слайд 1

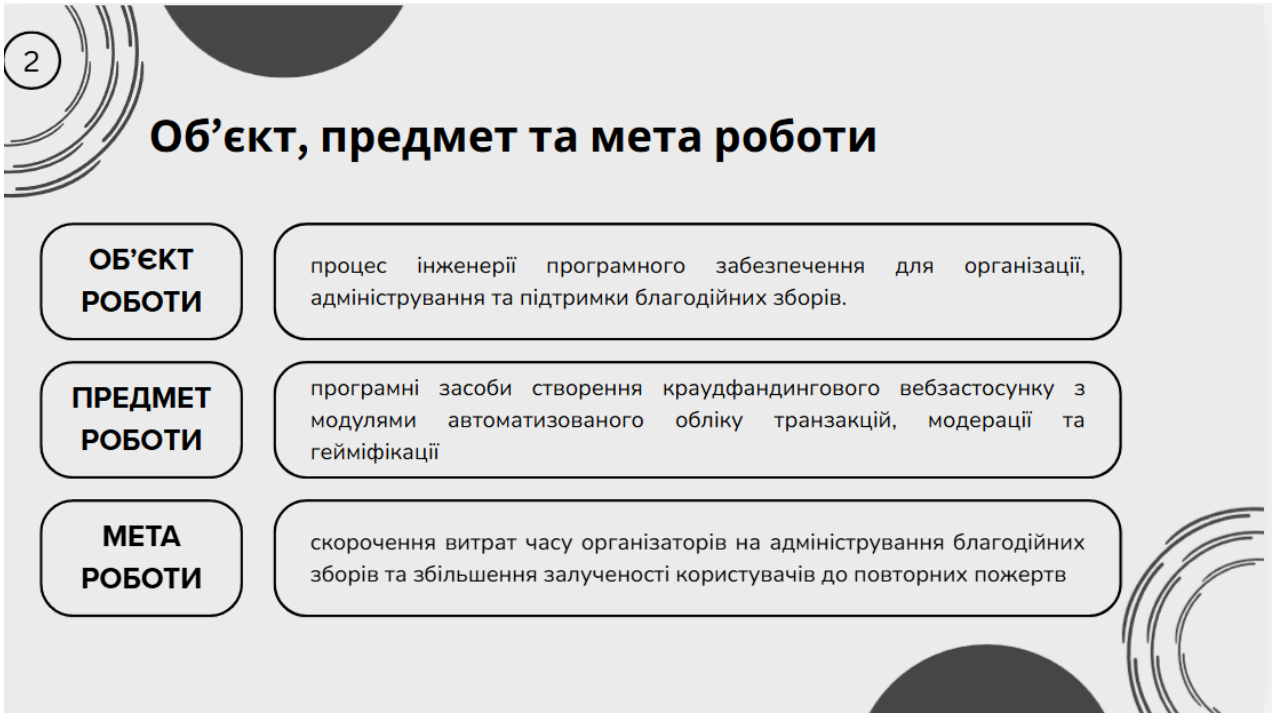

Рисунок В.2 – Слайд 2

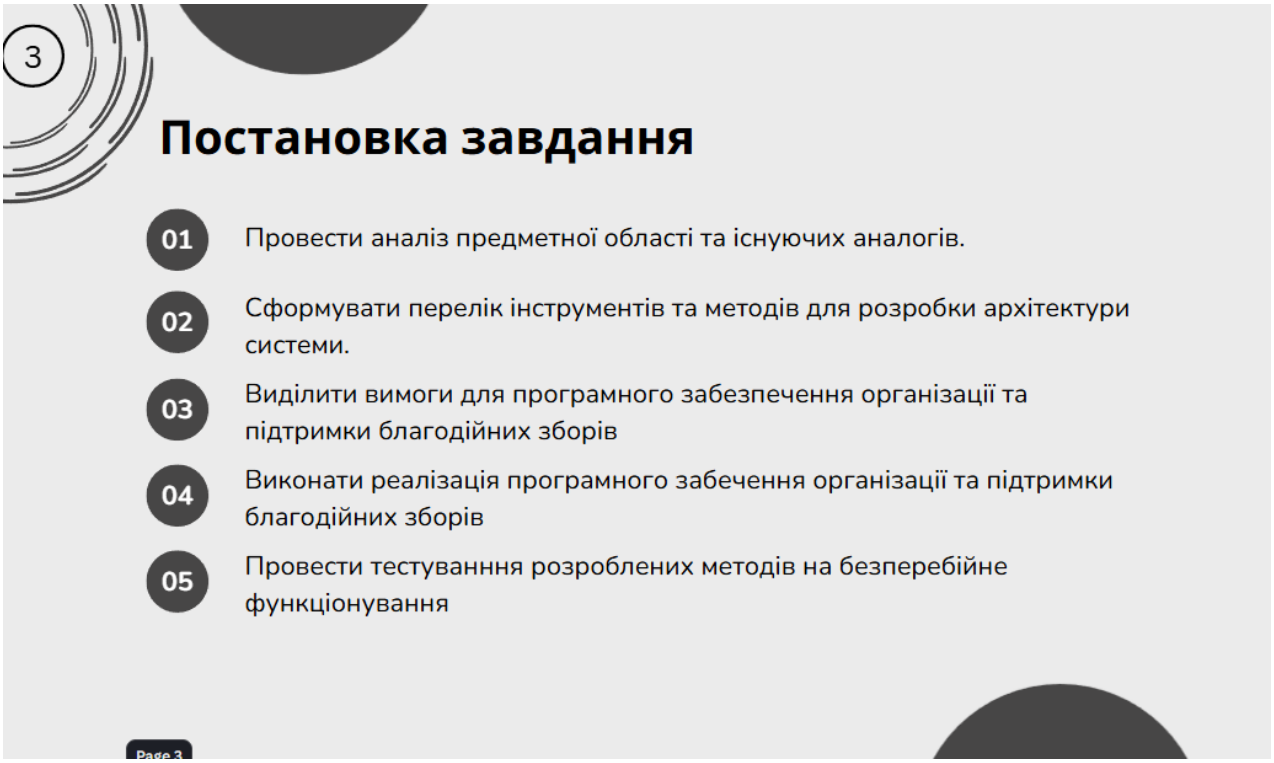

Рисунок В.3 – Слайд 3

Рисунок В.4 – Слайд 4

5

Порівння засобів розробки для програми

Критерій	TypeScript / NestJS	Python / Django	JavaScript / Express
Типізація	Статична (сувора)	Динамічна	Відсутня
Швидкість розробки	Висока	Середня	Висока
Продуктивність (I/O)	Висока	Середня	Висока
Масштабованість	Висока (Модульна)	Висока	Низька (хаотична)
Підтримка впровадження залежностей	Вбудована	Сторонні бібліотеки	Відсутня

Рисунок В.5 – Слайд 5

6

Use-case діаграма системи

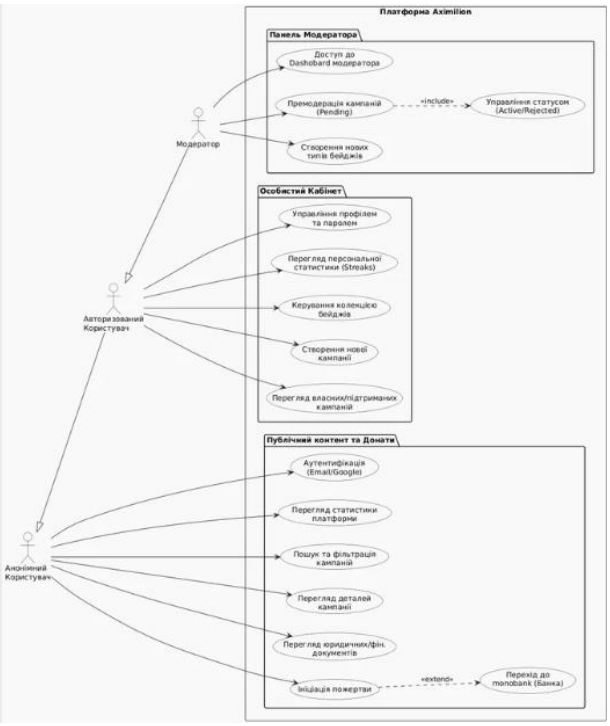

Рисунок В.6 – Слайд 6

Рисунок В.7 – Слайд 7

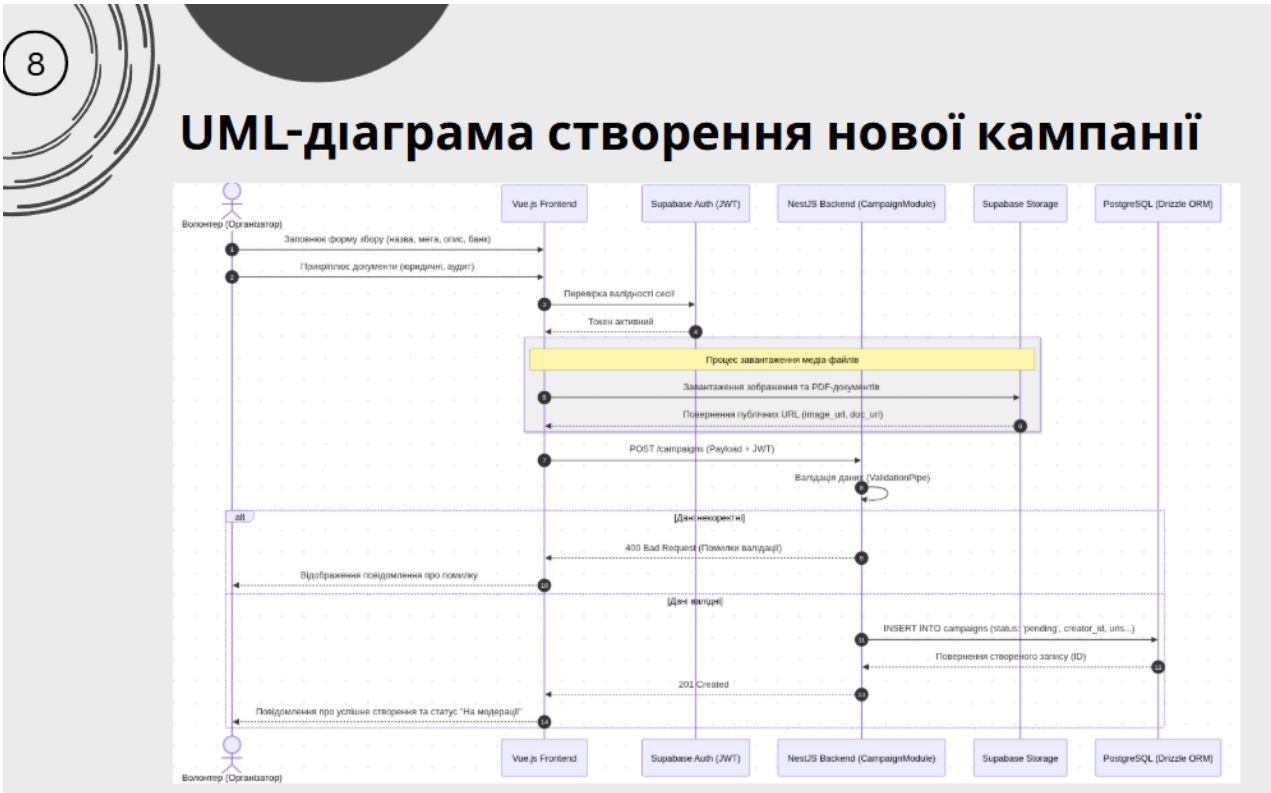

Рисунок В.8 – Слайд 8

Рисунок В.9 – Слайд 9

Рисунок В.10 – Слайд 10

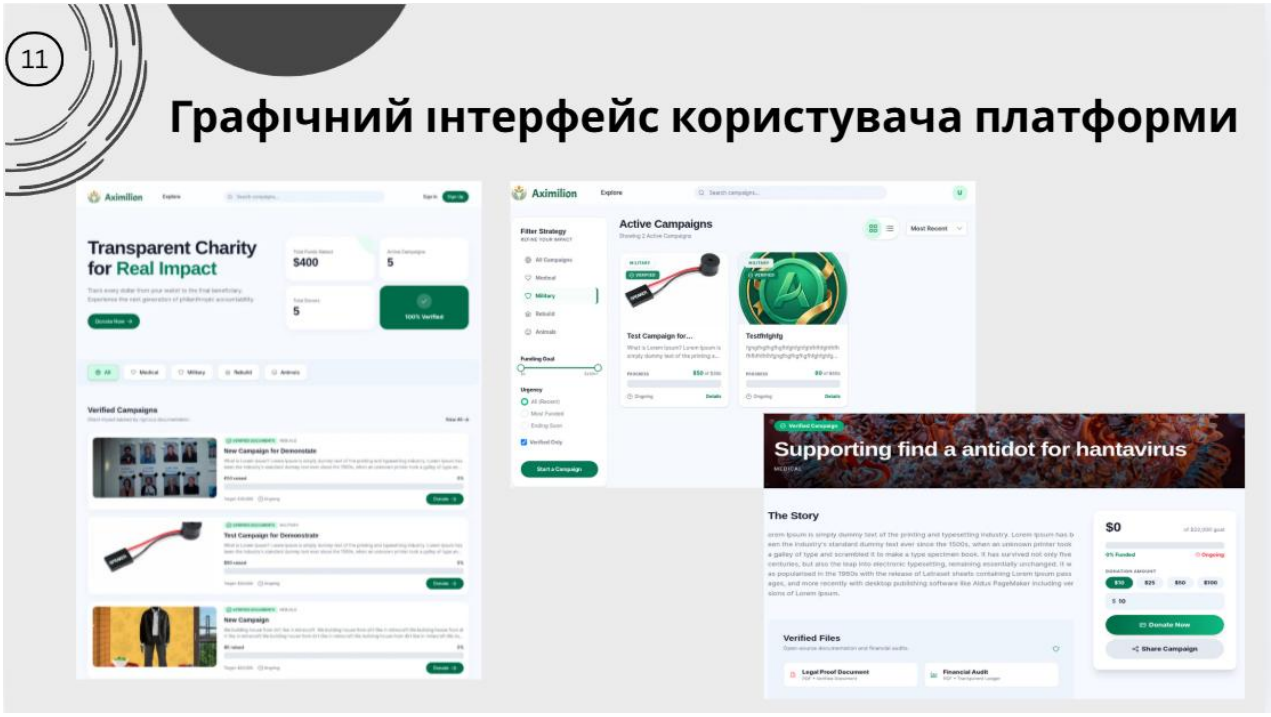

Рисунок В.11 – Слайд 11

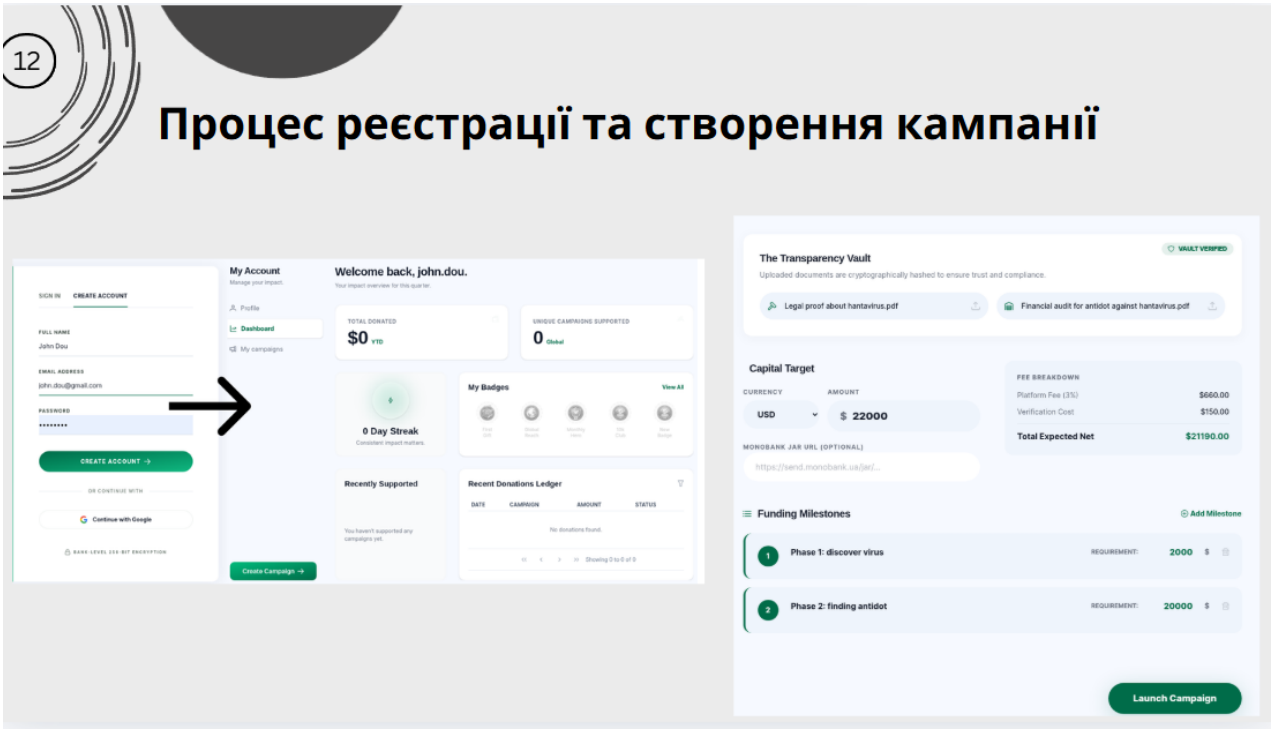

Рисунок В.12 – Слайд 12

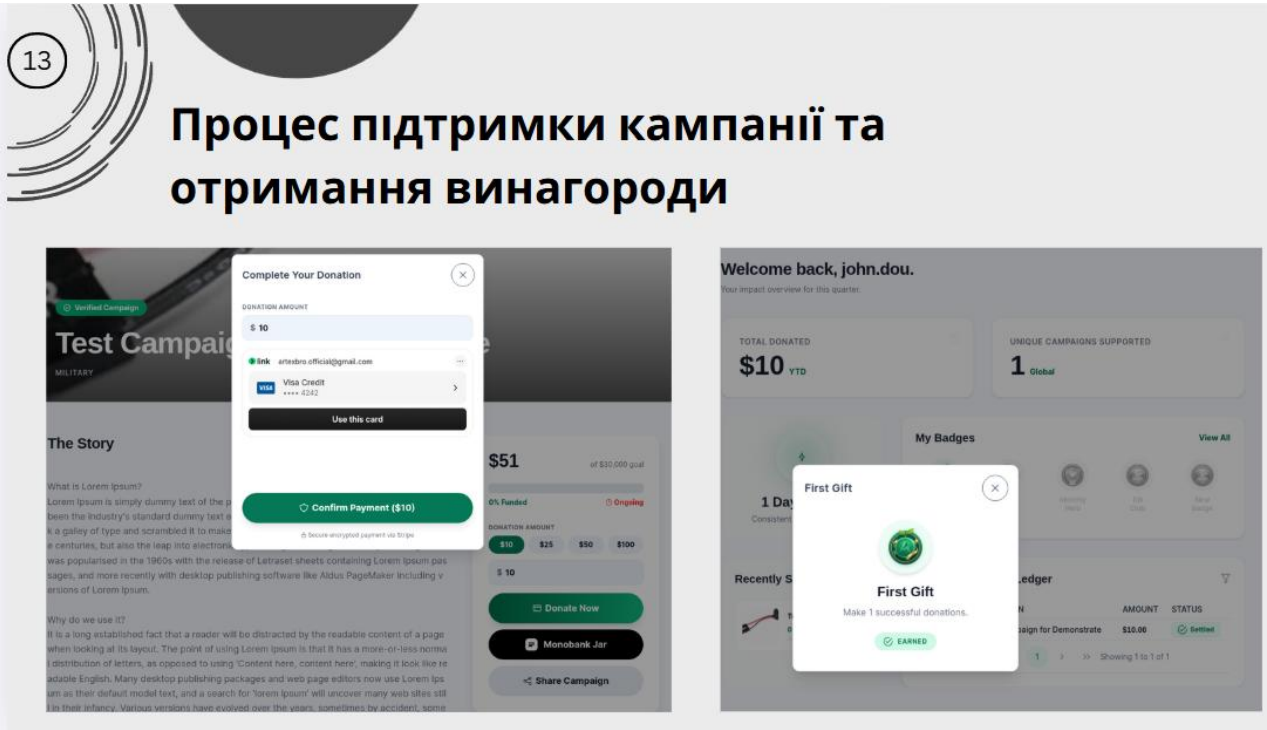

Рисунок В.13 – Слайд 13

Рисунок В.14 – Слайд 14