Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система для організації збору коштів. Клієнтська частина для створення ініціатив та проведення зборів
(тема)

Виконав:
здобувач 4 року навчання
групи ПЗПІ-21-1

Сергій ПАХАРЕНКО
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник ст.викл. кафедри Віталій ЛЯПОТА.
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило Смеляков
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Пахаренку Сергію Олеговичу
(прізвище, ім’я, по батькові)

1. Тема роботи Програмна система для організації збору коштів. Клієнтська частина для створення ініціатив та проведення зборів. Затверджена наказом по університету від 19.05.2025 р. № 397 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 19.06.2025 р.

3. Вихідні дані до роботи Розробити клієнтську частину програмного рішення, що забезпечує управління ініціативами, створення зборів, облік пожертв та інтеграцію з платіжними платформами, з використанням Next.JS та React

4. Перелік питань, що потрібно опрацювати в роботі: Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	08.04.2025	*виконано*
2	Створення специфікації ПЗ	12.04.2025	*виконано*
3	Проектування ПЗ	15.04.2025	*виконано*
4	Розробка ПЗ	16.04.2025	*виконано*
5	Тестування ПЗ	28.04.2025	*виконано*
6	Оформлення пояснювальної записки	30.04.2025	*виконано*
7	Підготовка презентації та доповіді	10.05.2025	*виконано*
8	Попередній захист	17.06.2025	*виконано*
9	Нормоконтроль, рецензування	17.06.2025	*виконано*
10	Здача роботи у електронний архів	18.06.2025	*виконано*
11	Допуск до захисту у зав. кафедри	18.06.2025	*виконано*

Дата видачі завдання «07» «квітня» 2025 р.

Здобувач _____ _____ Сергій ПАХАРЕНКО
(підпис)

Керівник роботи _____ ст.викл. Віталій ЛЯПОТА
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра, 60 стор., 8 рис., 1 табл., 13 джерел, 3 додатків.

АВТЕНТИФІКАЦІЯ, АРХІТЕКТУРА, БЕЗПЕКА, ІНТЕГРАЦІЯ, МІКРОСЕРВІСИ, МОБІЛЬНІСТЬ, ПЛАТЕЖІ, СПОВІЩЕННЯ.

Метою дослідження є створення клієнтської складової програмної системи, яка спрощує запуск та супровід благодійних ініціатив. На підставі аналізу поведінкових моделей сформовано вимоги до низки сервісів: швидке формування кампанії з перевіркою організатора, гнучке налаштування цілі та дедлайну, безпечна обробка внесків із двохфакторною перевіркою, а також персоналізовані сповіщення для підтримання залученості аудиторії.

Запропонований інтерфейс поєднує мінімалістичний дизайн і миттєвий візуальний відгук, що дозволяє донорові перейти від ознайомлення до переказу коштів за декілька дотиків, а ініціатору – керувати збором без зайвих переходів. Мікросервісна архітектура клієнт-сервера дає змогу масштабувати критичні ділянки, тоді як дворівневе шифрування й суворе розмежування прав підтримують цілісність даних.

Результатом роботи став працездатний прототип клієнтської частини, який демонструє високу продуктивність під час навантажувального тесту, відповідає вимогам захисту даних та створює основу для подальшого розширення – зокрема, інтеграції алгоритмів рекомендацій і багатомовної підтримки.

ABSTRACT

AUTHENTICATION, ARCHITECTURE, SECURITY, MOBILITY, MICROSERVICES, PAYMENTS, NOTIFICATIONS, INTEGRATION.

The study aims to create the client component of a fundraising software system that simplifies the launch and maintenance of charitable campaigns. Behavioural analysis defined requirements for a set of services: rapid campaign creation with organiser verification, flexible goal and deadline settings, secure contribution processing with two-factor checks, and personalised notifications that keep the audience engaged.

The proposed interface blends a minimalist layout with instant visual feedback, allowing a donor to move from discovery to donation in just a few taps, while giving an initiator full control over a campaign without redundant navigation. A client-server design based on microservices lets the platform scale critical paths, whereas dual-layer encryption and strict role separation safeguard data integrity.

The outcome is a working prototype of the client side that shows high performance under load testing, meets data-protection requirements, and lays the groundwork for further enhancements such as recommendation algorithms and multilingual support.

ЗМІСТ

ВСТУП

Веб-фандрейзингові платформи вже давно перестали бути статичними «дошками оголошень» і дедалі частіше слугують середовищем, де ініціатива може народитися, пройти модерацію, зібрати кошти й відзвітувати про використані ресурси буквально в один клік. Саме браузер сьогодні диктує правила гри між організаторами та донорами від моменту, коли користувач випадково натрапляє на картку кампанії, до секундного повідомлення про успішну оплату. Конкуренція платформ росте, так само як і очікування щодо прозорості, швидкості та безпеки взаємодії.

Актуальність практичного дослідження зумовлена потребою у гнучкому веб-клієнті, здатному одночасно задовольнити два різних сценарії поведінки. Перший – імпульсний донат, де вирішальними є мінімум кроків і миттєвий відгук. Другий – довготермінове адміністрування ініціативи, що вимагає детального дашборда, систематичних оновлень і живої комунікації з аудиторією. Такий інтерфейс повинен поєднувати адаптивність, багатомовність, верифікацію користувачів і сувору відповідність нормам захисту даних та фінансового моніторингу.

Метою практичнлї роботи є розроблення архітектури та реалізація клієнтської частини системи збору коштів, яка забезпечує інтуїтивну навігацію, швидке створення й редагування кампаній, безпечну обробку платежів, персоналізовану стрічку ініціатив і надійний двосторонній зв’язок між усіма учасниками процесу. Особливий акцент ставиться на компонентну побудову, горизонтальну масштабованість і застосування сучасних практик UI/UX-дизайну.

Об’єктом дослідження виступає веб-взаємодія користувачів із благодійними кампаніями, предметом – моделі реалізації клієнтської підсистеми управління ініціативами та зборами. Практичне значення полягає у створенні технічної бази, придатної як для стартап-проєктів, так і для некомерційних організацій, які прагнуть підвищити ефективність і прозорість фандрейзингу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Особливості клієнтської частини для створення ініціатив та проведення зборів

Новітня клієнтська частина фандрейзингової платформи більше схожа на TikTok-студію, ніж на форму для податкової декларації: вона дозволяє користувачеві відкрити телефон, ковзнути великим пальцем по кількох полях, миттєво побачити прев’ю власної кампанії й уже за хвилину після публікації спостерігати, як прогрес-бар підстрибує від перших внесків друзів. Цей драматичний зсув неможливо пояснити самою лише «любов’ю до мобільних технологій»; він спирається на жорстку статистику поведінки. За підрахунками платформи GoFundMe, у дві тисячі двадцять четвертому році приблизно шістдесят відсотків усіх пожертв на сервісі зроблено зі смартфонів, причому середня швидкість волонтерської «трикадрової історії» у стрічці становила дві транзакції на секунду [1]. У такій швидкісній інфраструктурі фронтенд, що змушує прокручувати форму довше тридцяти секунд, уже вважається технічним анахронізмом.

Пришвидшення мобільного сценарію супроводжується не абстрактним глянцем, а конкретним грошовим ефектом: згідно з галузевою аналітикою AbsRBD, мобільні перекази зросли на двісті п’ять відсотків за останні кілька років, тоді як десктоп - лише на сорок [2]. Публіка, що тримає телефон у долоні, не бажає проходити квест із трьома сторінками полів; вона очікує того самого one-tap-досвіду, який дає Apple Pay за ранкову каву. Тому клієнтська частина, орієнтована на ініціаторів, сьогодні сповідує принцип «single-screen storytelling»: користувач бачить одразу обкладинку, назву, потрібну суму, таймер дедлайну й коротку емоційну підводку, а все вторинне детальний опис проблеми, юридичні деталі, фінансові гарантії - заховано у вкладку «дізнатися більше», що відкривається вже після натискання «створити».

Ініціатор заходить у застосунок не лише як автор ідеї, а й як менеджер майбутнього міні-проєкту. Щойно він ставить першу цифру у полі «ціль», бекенд

ініціює приховану сесію зі службою ризик-скорингу: перевіряється унікальність тексту, подібність зображення до наявних у базі шахрайських кампаній, співставляється гео-тег фотографії з задекларованим місцем закупівлі. Успішна перевірка триває менше секунди, і фронтенд навіть не натякає, що машина перевірила користувача на наявність незакритих «банок» чи підозрілих IP-кластерів. Якщо алгоритм помічає ризиковий збіг, застосунок не лякає червоними поп-апами; він чемно пропонує завантажити додатковий документ або уточнити суму. Ця м’яка корекція важлива: за даними Kickstarter, різке перенаправлення на сторінку додаткової верифікації знижує шанс запуску проєкту на третину [3], тоді як ненав’язливий «підкажіть трішки більше» майже не б’є по конверсії, бо користувач уже відчуває себе хазяїном процесу.

Дизайнери нового покоління зневажають довгі текстові інструкції й використовують мікрокопі на кшталт «залишилося два кроки» чи «це займе не більше хвилини». Практика показує, що такі підказки зменшують відтік під час створення кампанії майже удвічі, особливо на ринку, де половина потенційних ініціаторів заходить через мобільний трафік із нестабільним інтернетом [4]. Сам фронтенд збирається як реактивний потік: усі поля двобічно зв’язані з локальним станом, тому навіть у тунелі метро можна заповнити опис, обрати фотографію з галереї, зберегти чернетку й завершити публікацію, коли повернеться сигнал. Це не косметична фіча, а страховка від найпоширенішого сценарію «зависло і все пропало», який здатен назавжди відбити охоту до волонтерства в цифрі.

Щойно кампанія опублікована, починається справжній танець реального часу. Прогрес-бар не просто малює зелену лінію; він стріляє ендорфінами. Кожне нове надходження вдаряє анімацією «+₴200» біля кнопки Share, підштовхуючи автора перекинути посилання в чат. Саме тому платформи висувають до клієнтської частини вимогу «two-second glory»: між фінальним кроком оплати й візуальним оновленням суми не має минути більше двох секунд; інакше відчуття «я зробив добро» блякне, а на повторний внесок шансів менше. Якщо сервер затримується, фронтенд використовує оптимістичне відображення і позначає суму сірим відтінком, який перетворюється на нормальний колір, щойно транзакція

з’являється в базі. Людина отримує емоцію «встиг!» навіть у мить, коли банк ще не прислав фінальний webhook.

Особливий виклик у клієнтській частині – багатовалютність. Мобільний донор із Варшави хоче залишити злоті через Apple Pay, друг із Сіднея – кине австралійські долари з Google Pay, хтось узагалі віддасть перевагу USDC на мережі Polygon, щоб уникнути конвертації. Застосунок не має права питати «а що таке USDC?», він повинен показати готовий сценарій: вибір мережі, підказку комісії та попереджувальне повідомлення, що транзакція може зависнути, якщо валюта летить через міжланцюговий міст. Під капотом це означає велику оркестровку. Кожен платіжний метод завертається у свій модуль, але фронтенд відображає їх у єдиному стилі, щоб користувач не відчув, що з пластиковою карткою все просто, а стейблкоїн – це гік-квест. Легкість має залишатися тотальною, хоч би скільки нормативних актів додалося на задньому плані, а після ухвалення посиленого пакета AML у Раді ЄС мобільні клієнти апріорі вбудовують лінію перевірки великих переказів та автоматизовано ставлять сумнівні адреси у затримку .

Паралельно з фінансовими викликами клієнтська частина вирішує завдання живої комунікації. Донор, який щойно підкинув сотню гривень, не хоче чекати кінця збору, щоб дізнатися, чи вирушив тепловізор на фронт. Тому автору ініціативи пропонується вбудований редактор оновлень: коротке вертикальне відео, фотографія з підписом, аудіонотатка, усе це злітає на сервер одним дотиком і миттєво з’являється у стрічці кампанії. Опціонально кожен апдейт штовхає push на телефони донорів, але не більше трьох разів на добу, щоб не перетворювати благодійність на шкідливий спам. Парадокс в тому, що саме регулярність і медійна якість апдейтів найсильніше впливають на повторні внески; за спостереженнями dobro.ua проміжний звіт із фото удвічі підвищує приріст суми протягом наступних сорока восьми годин [5].

Універсальна мандрівка грошей через мобільний інтерфейс завершується на екрані адміністратора кампейну, де автор уже бачить, що робити далі. У старих веб-версіях організатор мусив чекати закриття всієї суми, сьогодні ж часткове виведення стало стандартом. Клієнт готовий показати автору кнопки

«забронювати», «отримати», «відзвітувати», і кожен тап відкриває мікропроцедуру з підписом у два кроки, щоби не пропустити неточну суму чи випадкове списання. Саме тут проявляється концепція «ініціатива як продукт»: додаток пропонує автору встановити етапи – наприклад, сорок відсотків суми піде на закупівлю перших десяти турнікетів, ще сорок – на логістику, а залишок на страховий резерв. Увесь цей таймлайн візуалізується як roadmap, щоб донор бачив, навіщо потрібні наступні гроші. Таким чином фронтенд фактично перетворює гуманітарну ініціативу на мікростартап із дорожньою картою, але без складних корпоративних графіків.

Звичайно, жоден клієнтський інтерфейс не може обійтися без багатомовності. В українських реаліях мінімум дві локалізації, українська й англійська розширюють охоплення на аудиторію діаспори, а додавання польської або німецької переводить кампанію з локальної у пан-європейську. Тут майбутнє вже стукає у двері: сучасні фреймворки дозволяють завантажувати переклади на льоту, а LLM-модулі дороблюють стилістику заголовків під час публікації, щоби польський користувач отримав не «збір на тепловізор», а коректне «zbiórka na kamerę termowizyjną». Якщо філолог раптом помітить недоладність, він може внести редагування, і система навчиться на цьому, щоб наступного разу пропонувати кращий варіант.

Важко переоцінити роль аналітики in-app. Коли донор натискає «поділитися», фронтенд відразу помічає, чи лінк полетів у Telegram, WhatsApp чи Instagram Stories, і підставляє відповідні параметри UTM. Через добу автор кампанії отримує сповіщення: «Stories принесли сімнадцять нових донорів, середній чек – шістдесят гривень». Цей мікроінсайт мотивує зробити другу серію контенту. А от якщо три доби поспіль немає новин, додаток тактовно штовхає пуш «люди сумують за апдейтом» і пропонує зняти коротке відео прямо в застосунку. Усе це здається дріб’язком, але з практики великих платформ очевидно: кампанії з регулярним мультимедіа-пульсом збирають у середньому на третину більше.

Тим часом архітектура клієнтської частини потроху готується до безпарольного майбутнього. Запуск Passkey-автентифікації в iOS і Android уже

привчив мільйони користувачів заходити без паролів, а платформи на кшталт GitHub показали, наскільки менше friction-кроків потрібно, щоби переконати людину повернутися до застосунку. Тому новенький фронтенд одразу уникає полів «створи пароль», він пропонує Face ID або Touch ID і підштовхує зберегти резервний ключ у менеджері паролів. Єдина складність – прив’язати Passkey до мульти-ролевої моделі, де одна особа може керувати кількома ініціативами під різними юридичними особами. Проте рішення вже на горизонті: self-issued OpenID змушує бекенд зберігати лише анонімний суб’єкт-ідентифікатор, а всі інші атрибути підтягувати через додаткові claims, підписані самим користувачем.

Попри всі тонкощі досвіду, найголовнішим показником залишається час від кліку до внеску. Коли команда розробників тестує черговий реліз, вони не питають «чи красиво виглядає кнопка», вони запускають спринт, у якому щоночі гонять тисячу симульованих платежів і відстежують P99-затримку. Гігієнічна норма сьогодні – не більше двохсот мілісекунд між запитом клієнта та відповіддю шлюзу. Якщо число росте, продукт-менеджер уже наступного ранку отримує червону картку. І це не перфекціонізм заради премії дизайнера; це сувора економіка довіри. Донор, який не бачить, як цифра росте, підсвідомо сумнівається, чи дійшли його кошти, а отже не повернеться повторно.

У найближчому майбутньому клієнтська частина фронтенду для ініціатив та зборів перетвориться на справжню творчу лабораторію. Застосунок підказуватиме оптимальний час публікації оновлення, прогнозуватиме, якою буде середня швидкість надходжень і якої миті варто переключити аудиторію з «вогнепальними турнікетами» на «зимові спальники», тому що модель LLM помітила сезонну втому користувачів від військової тематики. Перші прототипи подібних порад уже з’являються в закордонних додатках, і вже зрозуміло, що питання не в тому, чи прийдуть вони в українські мобільні збірки, а в тому, чи встигне кожна платформа навчитись діагностувати свою аудиторію швидше за конкурентів.

Отже, сучасна клієнтська частина для керування ініціативами – це не просто форма з полями, а насичений екосистемний вузол, який одночасно вирішує драматичне завдання трьох секунд емоції, ніжний баланс між зручністю і

нормативами, медіа-режисуру, фінансову безпеку й багатомовний культурний код. Хто приборкає цей гібридний талмуд, той отримає не лише вищу конверсію, а й право на глобальну експансію, бо мобільний донор із Нью-Йорка, Варшави чи Львова мислить однаково: дайте кнопку «зробити добро», покажіть, що гроші вже в роботі, і зробіть так, щоби все це тривало менше часу, ніж завантажується сторінка ранкового мем-каналу.

1.2 Технічні та регуляторні вимоги до клієнтської логіки й взаємодії з бекендом

Клієнтська частина, яка дозволяє людині вигадати ініціативу і за хвилину показати зростаючий лічильник пожертв, – це не просто «красивий екран», а суміш мережевої фізики, платіжної юриспруденції та психології нетерплячого користувача. Коли він торкається кнопки «Створити збір», додаток запускає каскад асинхронних подій: локальний сторедж приймає чернетку, на фоні відпрацьовує перевірка зображення на схожість із відомою базою шахрайських кампаній, а ще глибше прокручується ризик-скоринг, який звіряє IP з нещодавніми хвилями ботів. Усе це відбувається за частку секунди, бо будь-яка помітна затримка перекладається у цифри відтоку.

Саме тому парадигма offline-first перестала бути модним словом; вона стала гарантією, що ініціатива народиться навіть у київському метро між станціями, де немає жодного стільникового сигналу. Flutter-клієнт кешує все введене у зашифрований box, а інтерфейс поводиться так, ніби зв’язок ідеальний. Щойно з’являється мережа, пакет синхронізації летить до бекенду, який доліплює мікровідбиток часу й ставить запис у Kafka. Рекомендації Flutter-спільноти чіткі: локальний стан ведеться окремо від віддаленого репозиторію, реплеї запитів мають бути ідемпотентні, а конфлікти розв’язуються за правилом «сервер авторитетний, але не грубий» – тобто повертає останню версію і пропонує об’єднати зміни, замість безапеляційно відкидати кеш клієнта [6].

Така архітектура робить нову для індустрії ставку: затримка на рівні мережі компенсується передбачуваністю на рівні інтерфейсу. Користувач бачить, що його

заголовок збережено локально, і в цей момент йому вже психологічно байдуже, скільки мілісекунд знадобиться, щоб бекенд розмістив байти у сховищі. Утім, коли мова переходить до грошей, грають не емоції, а P99-метрики. Бізнес ставить планку: дев’яносто дев’ять відсотків запитів до API «confirmDonation» мусять пройти за дві сотні мілісекунд, інакше користувач не дочекається візуального фідбеку й почне тицяти кнопку вдруге. Цей поріг не взятий зі стелі; у практичних гайдах з оптимізації латентності саме 200 мс виступають межею між «о, вже спрацювало» і «чомусь довго думає» [7].

Фінансова інтеграція у мобільному клієнті поділяється на дві великі гілки. Перша – карткові й мобільні гаманці, де правила PSD2 диктують Strong Customer Authentication. Щоб не перетворювати кожен донат на танець із трьома кодами підтвердження, застосунок позначає дрібні пожертви як «low value» або «merchant initiated» і просить банк звільнити їх від додаткової перевірки. Страйп-документація підтверджує: платежі до тридцяти євро часто проходять за спрощеним маршрутом, а рекурентні списання за збереженими картками взагалі виходять за рамки SCA, якщо їх правильно позначити в API [8]. Друга гілка – криптовалюта і мультивалютні шлюзи. Тут клієнтський код мусить не просто показати QR-адресу, а сходу оцінити комісію мережі, повідомити про можливі затримки бридж-транзакцій і, головне, передати бекенду достатньо метаданих, щоби той міг прогнати платіж через chain-analysis і переконатися, що кошти не надходять із санкційного пулу. Після затвердження нових правил AML у Євросоюзі платформа зобов’язана вмикати ручну модерацію на великі, анонімні поповнення і документувати ланцюжок рішень для ревізорів [9].

Швидкість обробки не означає відсутності шифрування. Усі токени платіжних методів зберігаються окремо від профілю ініціатора, а ключі, що підписують локальний сторедж, містяться в апаратному HSM пристрою. Коли користувач у домені iOS додає карту, додаток одержує лише Device Account Number, ніколи – сам номер; коли ж він активує Passkey-вхід, бекенд бачить одноразовий публічний ключ і анамнестичний hash, але не знає справжнього імені, доки користувач сам його не розкриє. Пароль і, тим паче, секретне запитання про

дівоче прізвище матері випали зі стандарта UX: мобільні гіди Apple і Google прямо радять не гальмувати потік зайвим полем.

Паралельно з платіжною логікою працює система дельт-синхронізації. Кожна зміна стану лічильника «зібрано» видається бекендом через WebSocket-шину разом із контрольним числом, що дозволяє клієнту відсіювати дублікати. Якщо в кадри потрапляють тисячі одночасних донатів, фронтенд не перерисовує бар кожен раз: він агрегує прирости й оновлює інтерфейс тридцять разів на секунду, синхронізуючись із частотою екрана. Так пристрій не витрачає зайвих циклів CPU, а користувач дістає максимально плавну анімацію.

Ретеншн-механіку втілюють push-нотифікації, але не навмання. Аналітика на 700 мільйонах користувачів показує, що перевищення трьох пушів на добу збільшує відпис від повідомлень утричі, тоді як персоналізований контент піднімає рейт відкриття майже на сорок відсотків [10]. Тому клієнт підписується не на «всі апдейти», а на конкретну ініціативу з опцією «показувати лише фото» чи «лише фінальні звіти». Логіка таймінгу живе у бекенді, але саме апка обробляє шари пріоритетів, об’єднуючи декілька дрібних апдейтів у єдиний тихий push, якщо їхня сукупна вага перевищила ліміт, а час доставки ще не критичний.

Утримання донорів неможливе без моментальної аналітики для ініціатора. Дашборд, зібраний прямо у мобільному клієнті, показує карту джерел трафіку, коефіцієнт конверсії кожного каналу, середній чек та прогноз часу до досягнення цілі. Усе це обчислюється на бекенді, але передається готовими слайсами, щоб телефон не топив акумулятор. Проте інтерактивність залишається: торкання сектору діаграми відкриває деталізовану розбивку, а зміна часової шкали миттєво підвантажує дані з CDN.

Не менш вибагливою стає і процедура часткового виведення коштів. На екрані автора з’являється кнопка «забронювати суму», яка, по суті, підписує escrow-транзакцію. Коли буде потрібне підтвердження витрати, апка пропонує зняти коротке відео чи фото рахунку, автоматично накладає водяний знак із хешем платежу, а у фоні передає PDF-копію чеку в бекенд для анонімної OCR-перевірки. Якщо документ пристойної якості, бот-рецензент ставить зелений штамп і бекенд

відкриває наступний транш. Користувачі нічого цього не бачать; вони отримують лише пуш із текстом «перша партія турнікетів поїхала». Водночас аудиторія ревізора дістає посилання на захищений канал, де зберігається повний ланцюжок операцій на термін, передбачений податковим законодавством.

Кістяк безпеки – zero-trust. Окремий сервіс відповідає лише за автентифікацію, окремий – за менеджмент ініціатив, ще один – за фінансові токени. Мобільний клієнт отримує вузькі JWT-пропуски, що закінчуються через годину і потребують оновлення за допомогою refresh-токена, зашифрованого у Keychain. Варто деталі автентифікації змінити, і токен миттєво відкликається, тож втрачений смартфон не перетвориться на бомбу уповільненої дії.

Навіть анімація підпорядковується законам продуктивності. Віджети, які малюють прогрес, працюють на CanvasKit і вивантажують GPU-текстури лічильника з альфа-каналом, аби не відтоплювати CPU прокруткою. При зміні теми додатка включається режим адаптивної палітри, який вирізає анімації до трьох кадрів у секунду, коли система вмикає «збереження енергії». Усе це дрібниці у вакуумі, але разом вони зменшують середню витрату батареї настільки, що програми-монітори вже не підсвічують ініціативний клієнт червоним, а значить – менше підстав для видалення.

Розробники не мають розкоші запускати фічі «випадково». Канарейкові релізи йдуть на п’ять відсотків аудиторії з увімкненим детальним логуванням; якщо P99 підстрибує, фічу відмикають буквально одним прапорцем, і користувачі навіть не підозрюють про міні-інфаркт у бекенд-команди. Всі crash-репорти Telegramом тікають у Slack-канал «funds-extreme», де дежурний інженер бачить стек і назву гілки, що його породила.

Приватність на клієнті завершує картину. Додаток зберігає мінімальний профіль: анонімний айді, псевдонім, країну, список ініціатив. Після запиту «забудь мене» фронтенд запускає локальний scrubber, що зануляє Keychain, очищує базу Room, анулює кеш фотографій і посилає бекенду сигнал відкликати refresh-токен. За тридцять секунд користувачеві показують лаконічний екран «ми вас забули», а

бекенд протягом сімдесяти двох годин довідаляє залишкові PII за регламентом GDPR.

Тож сучасна клієнтська логіка, яка обслуговує створення та проведення зборів, – це хронічна гонка між мілісекундами й мегабайтами правил. Вона мусить дати людині відчуття творця, який одним жестом запускає міні-економіку, а водночас задовольнити банкіра, ревізора й GDPR-офіцера. Кожна зайва форма, кожен некомпактний запит, кожна втрата кадру на анімації віддаляє користувача від моменту «я допоміг», а значить забирає шанси на повторний донат. Перемагає той інтерфейс, що прикриває вибухову суміш платіжних SDK, криптографії та серверних черг витонченою ілюзією: все відбулося миттєво, безшовно і надійно, хоча під поверхнею цілодобово трудяться шифровані сховища, алгоритми AML і бездушний P99-хронометр.

1.3 Постановка задачі

У центрі цього дослідження – клієнтська частина веб-платформи, призначеної для формування, публікації й супроводу благодійних ініціатив та пов’язаних із ними зборів коштів. Її архітектуру передбачено збудувати на зв’язці Next.js + React, що дозволяє отримати динамічний, безшовний інтерфейс без повного перезавантаження сторінок, а взаємодію з бекендом реалізувати через REST-інтерфейс HTTP-запитів. Завдання полягає у тому, щоб за допомогою компонентного підходу, атомарного дизайну й адаптивних шаблонів створити одне «вікно» для двох принципово різних сценаріїв: імпульсивного донейту та довготривалого адміністрування кампаній [11].

Необхідно:

– проаналізувати наявні рішення у сфері веб-фандрейзингу й виокремити вимоги до швидкості відгуку, зручності навігації та прозорості даних;

– сформувати функціональні й нефункціональні потреби для модулів сторінок, форм, навігаційних елементів, віджетів відображення та службових утиліт;

– описати UML-модель діаграми компонентів, класів, послідовностей і

діяльності, що фіксує взаємодію користувача з системою від введення до серверної відповіді;

– спроєктувати маршрути та файлову структуру, де кожна сторінка має окремий фізичний представник у дерево-каталозі Next.js, а стан додатка синхронізується через двосторонній канал із бекендом;

– розробити інтерфейс, що поєднує мінімалізм, матеріальну візуальність, коректну типографіку й доступність за WCAG 2.1, забезпечуючи при цьому коректну роботу від мобільного екрана до великого десктопа;

– реалізувати механізми кешування, оптимістичного оновлення та серверних дій, які скорочують кількість мережевих звернень і прискорюють зворотний зв’язок;

– упровадити валидацію введених даних, псевдонімізацію персональної інформації, багаторівневу авторизацію та рольове розмежування доступів відповідно до вимог GDPR, PSD2 і актуальних норм протидії відмиванню коштів;

– протестувати працездатність клієнтської частини під навантаженням, оцінити час рендеру першого екрана й затримки оновлень, а також перевірити коректність відображення на основних браузерах і пристроях.

Підсумковим результатом має стати єдина веб-оболонка, яка забезпечує: для донора – швидкий шлях від перегляду до пожертви з миттєвим візуальним підтвердженням, для організатора – повноцінний інструмент управління кампаніями з живою статистикою й можливістю гнучко комунікувати з аудиторією. Саме така постановка задачі створює рамки подальшого проєктування, розробки й оцінки ефективності клієнтської складової системи збору коштів.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Загальний опис системи

Програмна система для організації збору коштів призначена для створення та управління благодійними ініціативами, а також для проведення зборів коштів на різноманітні потреби. Система надає можливість користувачам створювати власні ініціативи, здійснювати пожертви на існуючі проєкти, відстежувати стан своїх внесків та керувати особистим профілем.

Клієнтська частина системи забезпечує зручний інтерфейс для взаємодії користувачів з функціоналом платформи, дозволяючи ефективно організовувати збір коштів та керувати благодійними ініціативами. Система орієнтована на широке коло користувачів, включаючи організаторів благодійних зборів, благодійників та отримувачів допомоги.

2.2 Функціональні вимоги

Система повинна забезпечувати можливість реєстрації нових користувачів з наданням базової інформації, такої як ім'я користувача, електронна пошта та пароль з підтвердженням. Важливою функцією є підтримка входу зареєстрованих користувачів за допомогою електронної пошти та пароля, а також через облікові записи соціальних мереж. Безпека облікових даних користувачів та надійне зберігання паролів є критичними вимогами. Додатково система має надавати можливість відновлення забутого пароля.

Користувачам має бути доступна можливість перегляду та редагування особистих даних, включаючи повне ім'я, контактний телефон та фотографію профілю. Система повинна забезпечувати зручний доступ до історії власних пожертв, перегляду створених ініціатив та керування підписками на ініціативи. Для безпеки користувачів необхідна функція виходу з облікового запису.

Система має забезпечувати повний цикл створення нових ініціатив зі збору коштів з можливістю вказання назви, детального опису, вибору категорії, встановлення цільової суми та терміну збору коштів. Користувачам має бути

доступна функція редагування створених ініціатив, відстеження прогресу збору коштів та додавання оновлень про хід реалізації. Також необхідна функціональність для завершення або скасування ініціативи у разі потреби.

Користувачам має бути доступний перегляд списку всіх активних ініціатив та зручний пошук за різними критеріями, такими як назва, категорія, статус або сума зібраних коштів. Для кожної ініціативи має бути доступна детальна інформація, включаючи повний опис, поточний стан збору коштів, дані про організатора та історію оновлень. Важливою функцією є можливість фільтрації ініціатив за категоріями.

Система має забезпечувати зручний процес здійснення пожертв на обрані ініціативи з підтримкою різних способів оплати. Користувачам має бути доступна можливість вибору фіксованої суми пожертви або введення довільної суми. Для постійної підтримки важливих ініціатив система повинна забезпечувати можливість налаштування регулярних пожертв. Після кожної транзакції користувач має отримувати підтвердження успішного здійснення пожертви.

Система повинна надавати можливість підписки на оновлення обраних ініціатив та отримання сповіщень про важливі події, такі як оновлення ініціатив, успішне завершення збору коштів або поява нових ініціатив у обраних категоріях. Для зручності використання система має забезпечувати можливість налаштування частоти та типу сповіщень.

2.3 Нефункціональні вимоги

Інтерфейс системи має бути інтуїтивно зрозумілим та легким у використанні для користувачів різного рівня технічної підготовки. Важливою вимогою є адаптивний дизайн, який забезпечує зручне використання на різних пристроях. Система повинна надавати швидкий доступ до основних функцій з головної сторінки, зрозумілі повідомлення про помилки та підказки для користувачів. Уніфікований стиль оформлення всіх сторінок та компонентів забезпечить візуальну цілісність платформи.

Система має забезпечувати швидке завантаження сторінок, не більше 2 секунд для основних сторінок. Важливою вимогою є підтримка одночасної роботи великої кількості користувачів без значного зниження продуктивності. Система повинна оптимізувати завантаження та відображення зображень та ефективно використовувати кешування для зменшення навантаження на сервер.

Система має бути доступною не менше 99,5% часу, що означає допустимий час простою не більше 3,65 днів на рік. Критично важливою є коректна обробка помилок та виключних ситуацій без припинення роботи системи. Система повинна забезпечувати збереження даних користувачів та ініціатив при виникненні збоїв та мати механізми резервного копіювання та відновлення даних.

Система має забезпечувати захист персональних даних користувачів відповідно до законодавства. Усі операції повинні здійснюватися через захищене з'єднання (HTTPS). Надійне зберігання паролів має забезпечуватися з використанням сучасних алгоритмів хешування. Особливо важливою є безпечна обробка платіжних операцій з дотриманням стандартів безпеки платіжних систем. Система має включати механізми виявлення та запобігання шахрайським діям.

Архітектура системи має забезпечувати можливість горизонтального масштабування для обробки зростаючого навантаження. Важливою вимогою є підтримка можливості додавання нових функціональних модулів без суттєвої переробки існуючих компонентів. Система повинна забезпечувати можливість розширення списку категорій ініціатив та способів оплати, а також підтримувати можливість інтеграції з новими зовнішніми сервісами.

Інтерфейс системи має бути реалізований українською мовою з підтримкою можливості додавання інших мов у майбутньому. Система має забезпечувати коректне відображення специфічних символів української мови та локалізацію дат, чисел та валют відповідно до українських стандартів.

2.4 Вимоги до інтерфейсу користувача

Головна сторінка системи має містити верхню панель навігації з логотипом та основними елементами керування, слайдер з актуальними ініціативами, блок

пошуку та фільтрації, список популярних ініціатив та нижню панель з контактною інформацією. Головна сторінка повинна забезпечувати швидкий доступ до основних функцій системи та можливість швидкого переходу до створення нової ініціативи.

Сторінка профілю користувача має містити інформацію про користувача, можливість редагування особистих даних, список створених ініціатив, історію пожертв та список підписок на ініціативи. Сторінка профілю повинна надавати зручний інтерфейс для управління особистими даними та створеними ініціативами.

Сторінка перегляду ініціативи має містити повну інформацію про ініціативу, зображення, інформацію про організатора, поточний стан збору коштів, кнопку для здійснення пожертви, можливість підписки на оновлення та історію оновлень. Сторінка повинна відображати прогрес збору коштів у вигляді графічного індикатора для наочного представлення успішності збору.

2.5 Вимоги до інтеграції з зовнішніми системами

Система має інтегруватися з популярними платіжними системами для забезпечення можливості здійснення пожертв різними способами. Критично важливим є забезпечення безпечної передачі платіжних даних з дотриманням стандартів безпеки. Система також повинна надавати можливість налаштування регулярних платежів для підтримки ініціатив на постійній основі.

Система має забезпечувати можливість реєстрації та входу через облікові записи популярних соціальних мереж, а також поширення інформації про ініціативи в соціальних мережах для залучення нових благодійників. Система повинна забезпечувати можливість імпорту базової інформації про користувача з соціальних мереж при реєстрації.

Система має інтегруватися з сервісами електронної пошти для надсилання сповіщень та підтверджень, а також підтримувати надсилання сповіщень через мобільні додатки або веб-сповіщення. Важливою функцією є можливість налаштування частоти та типу сповіщень для користувачів.

2.6 Обмеження та припущення

Система розробляється як веб-додаток з використанням сучасних технологій розробки інтерфейсів. Важливою вимогою є підтримка роботи в сучасних веб-браузерах та забезпечення коректного відображення на пристроях з різною роздільною здатністю екрану. Також необхідна оптимізація використання ресурсів клієнтських пристроїв для забезпечення плавної роботи навіть на пристроях з обмеженими можливостями.

Система має відповідати вимогам законодавства України щодо збору та обробки персональних даних, а також дотримуватися вимог законодавства щодо благодійної діяльності та збору коштів. Система повинна надавати користувачам інформацію про умови використання та політику конфіденційності, а також забезпечувати можливість отримання згоди користувачів на обробку їх персональних даних.

Система має забезпечувати прозорість використання зібраних коштів для підвищення довіри користувачів. Важливою вимогою є можливість верифікації організаторів ініціатив для запобігання шахрайству. Система також повинна забезпечувати можливість моніторингу та модерації ініціатив для дотримання етичних норм та законодавства.

Сформовані вимоги до програмної системи для організації збору коштів визначають основні функціональні та нефункціональні характеристики, які повинна забезпечувати клієнтська частина системи. Реалізація цих вимог дозволить створити зручний та ефективний інструмент для організації благодійних ініціатив та проведення зборів коштів.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проектування ПЗ

Архітектура клієнтської частини програмної системи для організації збору коштів побудована на основі сучасного підходу до розробки веб-застосунків з використанням фреймворку Next.js та бібліотеки React. Така архітектура забезпечує створення динамічного та інтерактивного інтерфейсу користувача з швидкою навігацією між різними сторінками без повного перезавантаження сторінки.

Клієнтська частина системи відповідає за відображення інтерфейсу користувача, обробку взаємодії з користувачем, валідацію введених даних та відправлення запитів до серверної частини. Взаємодія з серверною частиною здійснюється через програмний інтерфейс (API) з використанням протоколу HTTP. Основні компоненти клієнтської частини:

– компоненти сторінок – відповідають за відображення окремих сторінок системи (головна сторінка, сторінка ініціативи, профіль користувача тощо);

– компоненти форм – відповідають за відображення та обробку форм введення даних (форма створення ініціативи, форма здійснення пожертви тощо);

– компоненти навігації – відповідають за навігацію між сторінками системи (заголовок, меню, підвал);

– компоненти відображення даних – відповідають за відображення даних (список ініціатив, картка ініціативи, прогрес збору коштів тощо);

– утилітарні компоненти – надають допоміжну функціональність (валідація даних, форматування, обробка помилок тощо).

Діаграма компонентів відображає основні компоненти клієнтської частини системи та їх взаємозв'язки. Клієнтський застосунок складається з компонентів сторінок, форм, навігації, відображення даних та утилітарних компонентів (див. рис. 3.1).

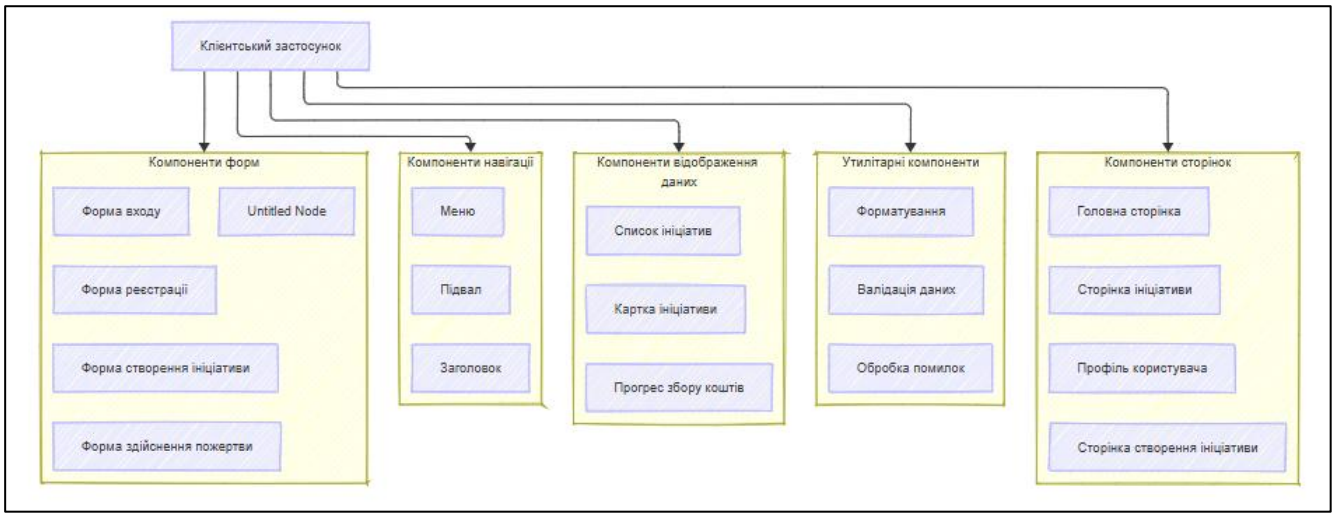

Рисунок 3.1 – Діаграма компонентів клієнтської частини (рисунок зроблено самостійно)

Клієнтська частина системи забезпечує повний цикл створення та управління ініціативами зі збору коштів, від створення нової ініціативи до відстеження прогресу збору та здійснення пожертв. Інтуїтивно зрозумілий інтерфейс та чітка навігація дозволяють користувачам різного рівня технічної підготовки ефективно взаємодіяти з системою. Нижче наведено схему відображає структури навігації (див. рис. 3.2).

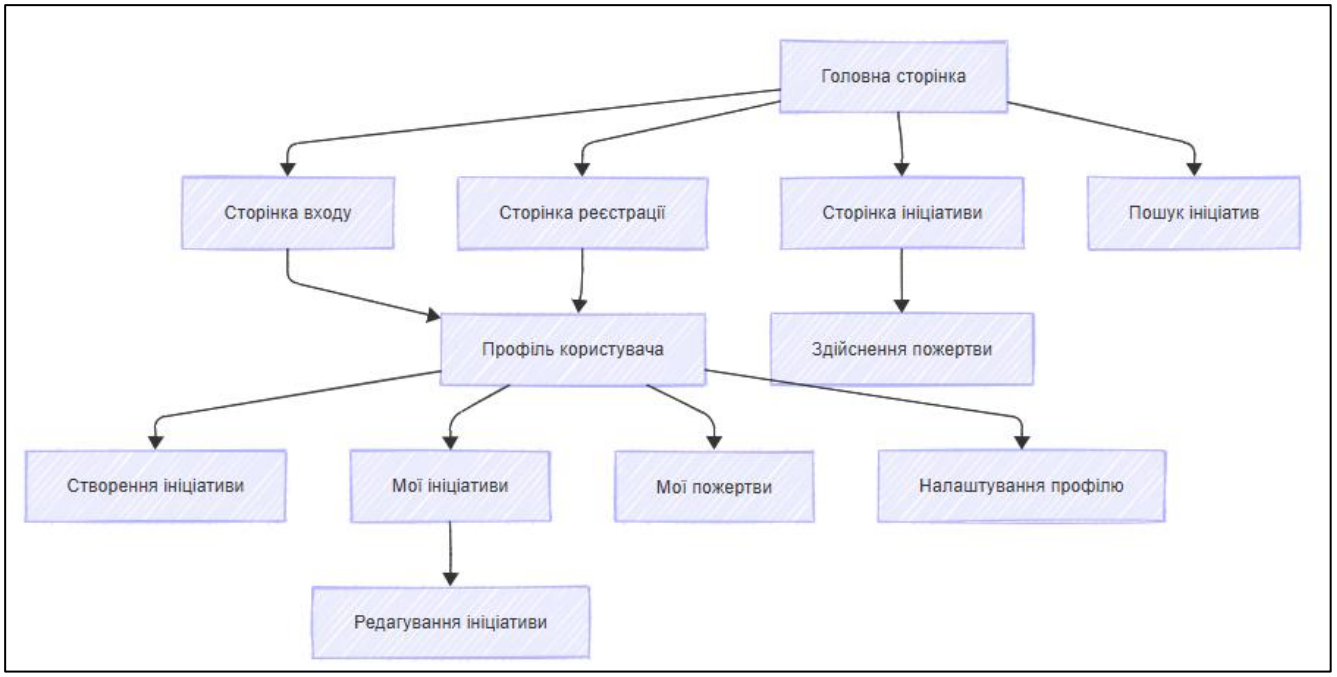

Рисунок 3.2 – Діаграма навігації (рисунок зроблено самостійно)

Схема відображає логічну структуру навігації веб-інтерфейсу платформи збору коштів і демонструє, як користувач послідовно проходить крізь основні функціональні вузли системи. Вона починається з головної сторінки, яка виконує роль центрального хаба й надає негайний доступ до двох взаємодоповнювальних потоків: авторизаційного та інформаційного.

Авторизаційний потік спрямовує користувача спершу на сторінку входу або, якщо облікового запису ще немає, на сторінку реєстрації. Обидві точки зливаються у профіль користувача, що стає ядром персоналізованого досвіду. З профілю відкриваються гілки дій, пов’язані з управлінням ініціативами, внесками та персональними даними: можна створити нову ініціативу, переглянути вже наявні та, за потреби, перейти до їхнього редагування; паралельно доступна секція «Мої пожертви», яка агрегує інформацію про участь користувача у фінансуванні проєктів, а також модуль налаштувань, де змінюються параметри облікового запису.

Інформаційний потік, що працює незалежно від того, чи виконано вхід, веде від головної сторінки до каталогу ініціатив або до сторінки пошукового інтерфейсу. Перегляд конкретної ініціативи переводить користувача на сторінку її детального опису, з якої здійснюється пожертва; після завершення транзакції система знову повертає користувача до його профілю, тим самим замкнувши коло та синхронізувавши нові дані про внесок. Така топологія забезпечує інтуїтивну ієрархію переходів: усі публічні дії стартують з головної сторінки, усі приватні – концентруються довкола профілю, а критичні бізнес-процеси, як-от створення або редагування ініціатив, жорстко прив’язані до перевіреного користувача.

Діаграма підкреслює, що система мінімізує кількість кроків між точкою входу, ключовою функцією та зворотним виходом у профіль, одночасно зберігаючи чітку вертикаль доступів і логічну послідовність дій.

3.2 Бібліотеки клієнтської частини

У процесі вибору технологічного стеку для клієнтської частини було виконано багатоетапний порівняльний аналіз сучасних веб-фреймворків і

бібліотек, серед яких React + Next.js, Vue + Nuxt, Angular Universal та Svelte + SvelteKit. Рішення на користь React + Next.js ґрунтується на комплексному наборі критеріїв, що охоплюють архітектурні, експлуатаційні та економічні аспекти створення й подальшого супроводу застосунку.

Передусім React забезпечує компонентно-орієнтовану, декларативну модель розробки інтерфейсу, що дає змогу формувати ієрархію UI-елементів зі строгими контрактами властивостей і станів. Така модель мінімізує когнітивні витрати під час масштабування коду, полегшує модульне тестування та сприяє високому відсоткові повторного використання компонентів. Крім того, React має найрозвиненішу екосистему бібліотек для управління станом, валідації форм, локалізації, візуалізації даних та доступності, що скорочує час до появи виробничого інкременту та знижує ризики самописних реалізацій.

Next.js виконує роль фреймворку верхнього рівня, який стандартизує процеси маршрутизації, рендерінґу та оптимізації ресурсів. Його вбудований гібридний підхід – поєднання серверного рендерінґу (SSR), статичної генерації (SSG/ISR) і клієнтського гідрування – забезпечує показник First Contentful Paint < 1 с і сприяє коректній індексації сторінок пошуковими системами. Підтримка automatic code-splitting та динамічного імпорту знижує розмір початкового бандла, що є критичним для мобільних клієнтів із підвищеною латентністю мережі. Модуль API Routes дозволяє розмістити безсерверні функції в єдиному репозиторії з клієнтським кодом, забезпечуючи централізоване керування логікою проксі-викликів до зовнішніх сервісів і минимізуючи кількість мікросервісних точок відмови.

Іншим вагомим фактором є інтеграція TypeScript «з коробки». Статична типізація зменшує кількість помилок виконання, підвищує надійність рефакторингу та прискорює рев’ю коду завдяки автоматизованому аналізу. Пряма підтримка React-StrictMode, Suspense і React Server Components у Next.js 14 відкриває можливість поступової міграції до нових оптимізацій без глобальних переписувань, що узгоджується з вимогами довгострокової підтримуваності програмного забезпечення.

Окремої уваги заслуговують експлуатаційні переваги. Фреймворк має офіційні адаптери для Vercel, AWS Lambda, Google Cloud Run та інших провайдерів, що спрощує мультихмарне розгортання й автоматично вмикає горизонтальне масштабування serverless-функцій. Вбудовані інструменти моніторингу Core Web Vitals і підтримка Source Maps пришвидшують пошук і усунення дефектів, а широка спільнота React-розробників знижує витрати на рекрутинг і виявляється критичною для коротких ітерацій розвитку продукту.

Таким чином, React + Next.js найбільш повно задовольняє сукупність визначених функціональних та нефункціональних вимог: компонентна ізольованість і повторне використання коду, висока продуктивність рендерінґу, стандартизований SSR/SSG-конвеєр, вбудована оптимізація мережевих ресурсів, безшовна підтримка TypeScript і багатий набір DevOps-інструментів для безперервного розгортання. Сукупність перелічених характеристик робить вибраний стек технологічно обґрунтованим і економічно ефективним для реалізації веб-системи, що потребує стабільної роботи під сплесковими навантаженнями, прогнозованої масштабованості та довгострокової підтримки [12].

3.3 Проектування UI/UX дизайну

Користувацький інтерфейс клієнтської частини системи розроблений з урахуванням принципів зручності використання, доступності та адаптивності. Дизайн системи базується на принципах матеріального дизайну та мінімалізму, що забезпечує сучасний та естетичний вигляд.

Компонентна структура інтерфейсу клієнтської частини організована за принципом атомарного дизайну, де компоненти поділяються на кілька рівнів абстракції:

– атомарні компоненти – найпростіші будівельні блоки інтерфейсу, такі як кнопки, поля введення, іконки;

– молекулярні компоненти – складаються з кількох атомарних компонентів та представляють більш складні елементи інтерфейсу, такі як форми пошуку, картки ініціатив, елементи навігації;

– організмові компоненти – складаються з молекулярних та атомарних компонентів та представляють цілі секції інтерфейсу, такі як заголовок сторінки, форма створення ініціативи, список ініціатив;

– шаблони – визначають структуру сторінки та розташування організмових компонентів;

– сторінки – конкретні реалізації шаблонів з наповненням даними (див. рис. 3.3).

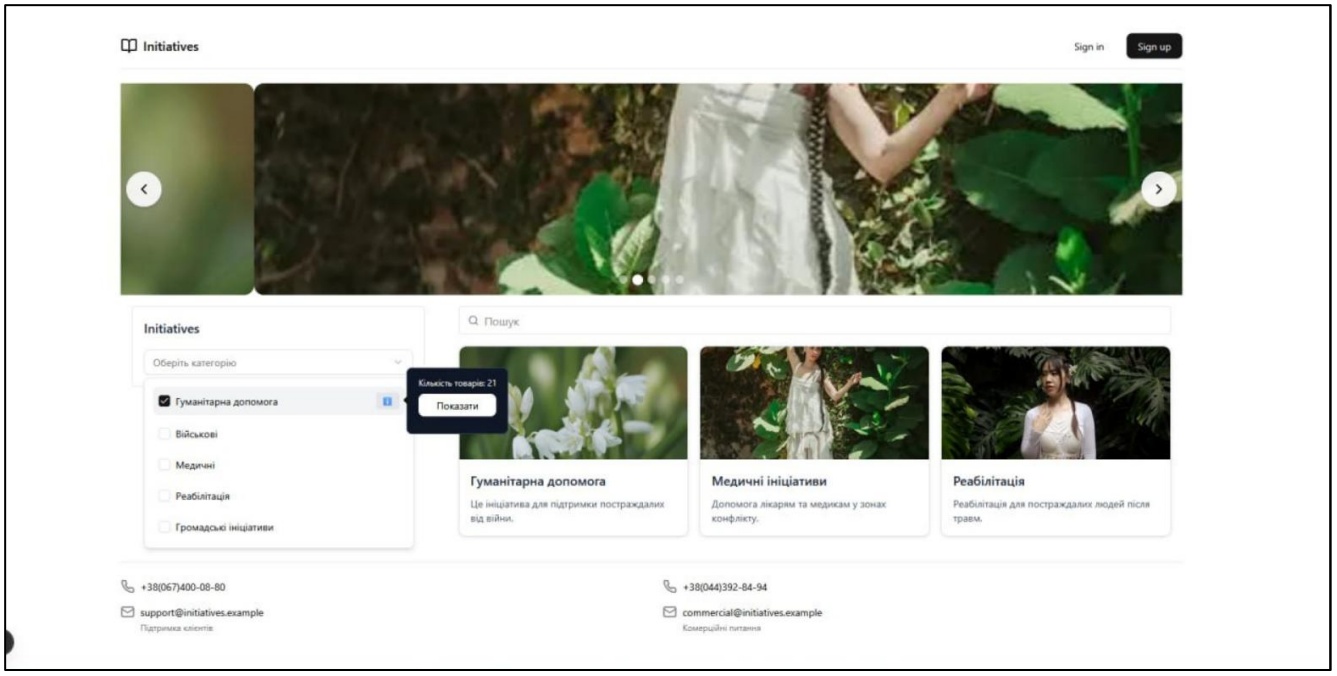

Рисунок 3.3 – Зовнішній вигляд інтерфейсу (рисунок зроблено самостійно)

Під час конструювання клієнтської частини системи ми свідомо відштовхувалися від двох радикально різних типів мотивації. З одного боку, це донор, у якого фактор імпульсу й швидкості переважає над раціональним порівнянням альтернатив: він гортає стрічку між двома зупинками метро, бачить заклик, натискає кнопку й миттєво отримує відчуття виконаного добра. З іншого боку, це організатор, який живе в довгому часовому інтервалі й потребує інструменту, здатного утримувати увагу аудиторії тижнями чи місяцями, регулярно оновлювати контент і в реальному часі ілюструвати, як перетворюються надіслані гроші. Таке подвійне позиціонування вимагало не просто зробити гарний інтерфейс, а спроєктувати адаптивну поведінкову екосистему, де кожна візуальна

деталь працює на залучення, а кожен клік негайно підкріплюється інформаційним фідбеком.

Маршрутизація в клієнтській частині реалізована з використанням вбудованого механізму маршрутизації Next.js, який базується на файловій системі. Кожна сторінка застосунку представлена окремим файлом або директорією в структурі проєкту, що спрощує організацію коду та навігацію між сторінками.

Основні маршрути клієнтської частини включають:

– головна сторінка (`/`) – відображає загальну інформацію про систему, популярні ініціативи та можливості пошуку;

– сторінка реєстрації (`/sign-up`) – дозволяє новим користувачам створити обліковий запис;

– сторінка входу (`/sign-in`) – дозволяє зареєстрованим користувачам увійти в систему;

– профіль користувача (`/user-profile`) – відображає інформацію про користувача, його ініціативи та пожертви;

– сторінка ініціативи (`/initiative/[id]`) – відображає детальну інформацію про конкретну ініціативу;

– створення ініціативи (`/create-initiative`) – дозволяє користувачам створювати нові ініціативи;

– пошук ініціатив (`/search`) – дозволяє користувачам шукати ініціативи за різними критеріями.

На рисунку 3.6 наведено приклад інтерфейсу статистики та загальних деталей збору із можливістю підтримки.

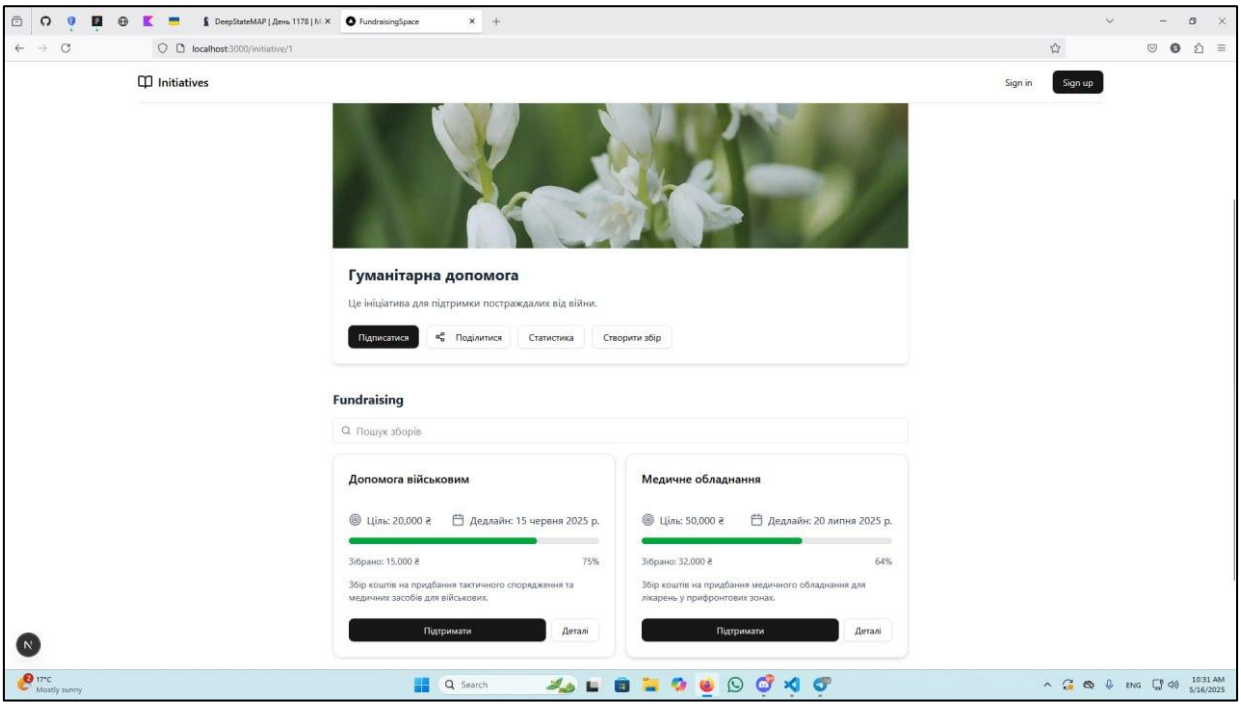

Рисунок 3.6 – Деталі зборів (рисунок зроблено самостійно)

Сторінка відображає список ініціатив у межах платформи FundraisingSpace: у шапці розташовано заголовок «Initiatives» та кнопки входу й реєстрації користувача, далі йде великий банер обраної ініціативи «Гуманітарна допомога» з описом, кнопками «Підписатися», «Поділитися», «Статистика» та «Створити збір», після чого підзаголовок «Fundraising» супроводжується полем пошуку зборів; у головній секції представлені картки двох активних кампаній – «Допомога військовим» і «Медичне обладнання» – кожна з яких містить інформацію про цільову суму й дедлайн, прогрес-бар, фактичний зібраний обсяг у гривнях, короткий опис призначення коштів та кнопки «Підтримати» й «Деталі» для переходу до процесу пожертвування або докладнішого перегляду кампанії.

На рисунку 3.7 наведено приклад реалізації оплати Stripe яка стає можливою тільки після авторизіції користувача.

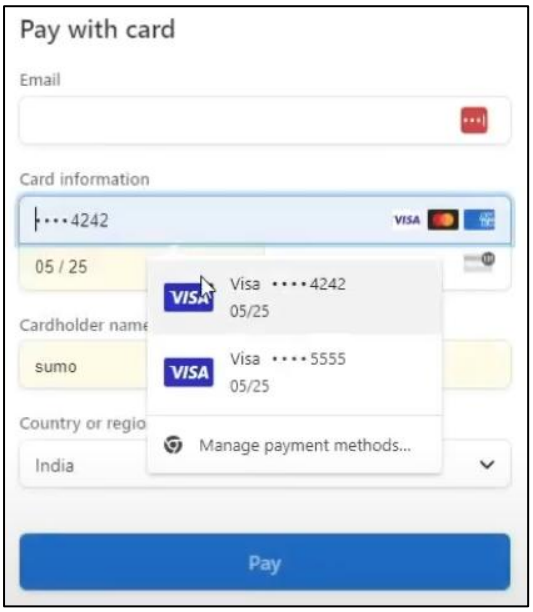

Рисунок 3.7 – Підтримка збору (рисунок зроблено самостійно)

У верхній частині форми користувачеві пропонується ввести адресу електронної пошти, після чого дані банківської картки: номер, термін дії і CVC-код. Під полем введення картки автоматично відображаються збережені платіжні методи (Visa ending in 4242, 5555 тощо) з можливістю керувати способами оплати. Далі заповнюється ім'я власника картки і вибирається країна або регіон зі списку. У самому низу розташована велика синя кнопка «Pay», за натисканням на яку ініціюється платіж.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Файлова структура проекту

Діаграма класів відображає основні клієнтські компоненти системи та їх взаємозв'язки. всі компоненти наслідуються від базового класу компонент. сторінкаініціативи використовує компоненти карткаініціативи, прогрес збору коштів та формаздійсненняпожертви для відображення інформації про ініціативу та надання можливості здійснення пожертви (див. рис. 4.1).

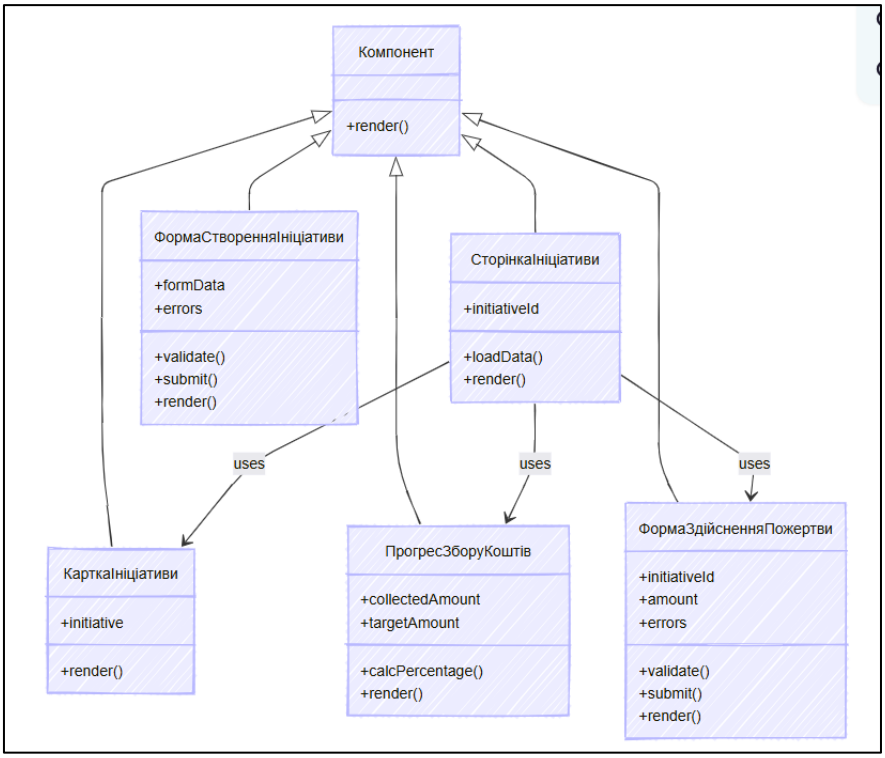

Рисунок 4.1 – Діаграма класів (рисунок зроблено самостійно)

Діаграма послідовності відображає процес створення нової ініціативи користувачем. Користувач заповнює форму створення ініціативи, система валідує введені дані, і якщо дані валідні, відправляє запит на сервер. Після отримання відповіді від сервера система відображає повідомлення про успішне створення ініціативи або помилки, якщо такі виникли (див. рис. 4.2)

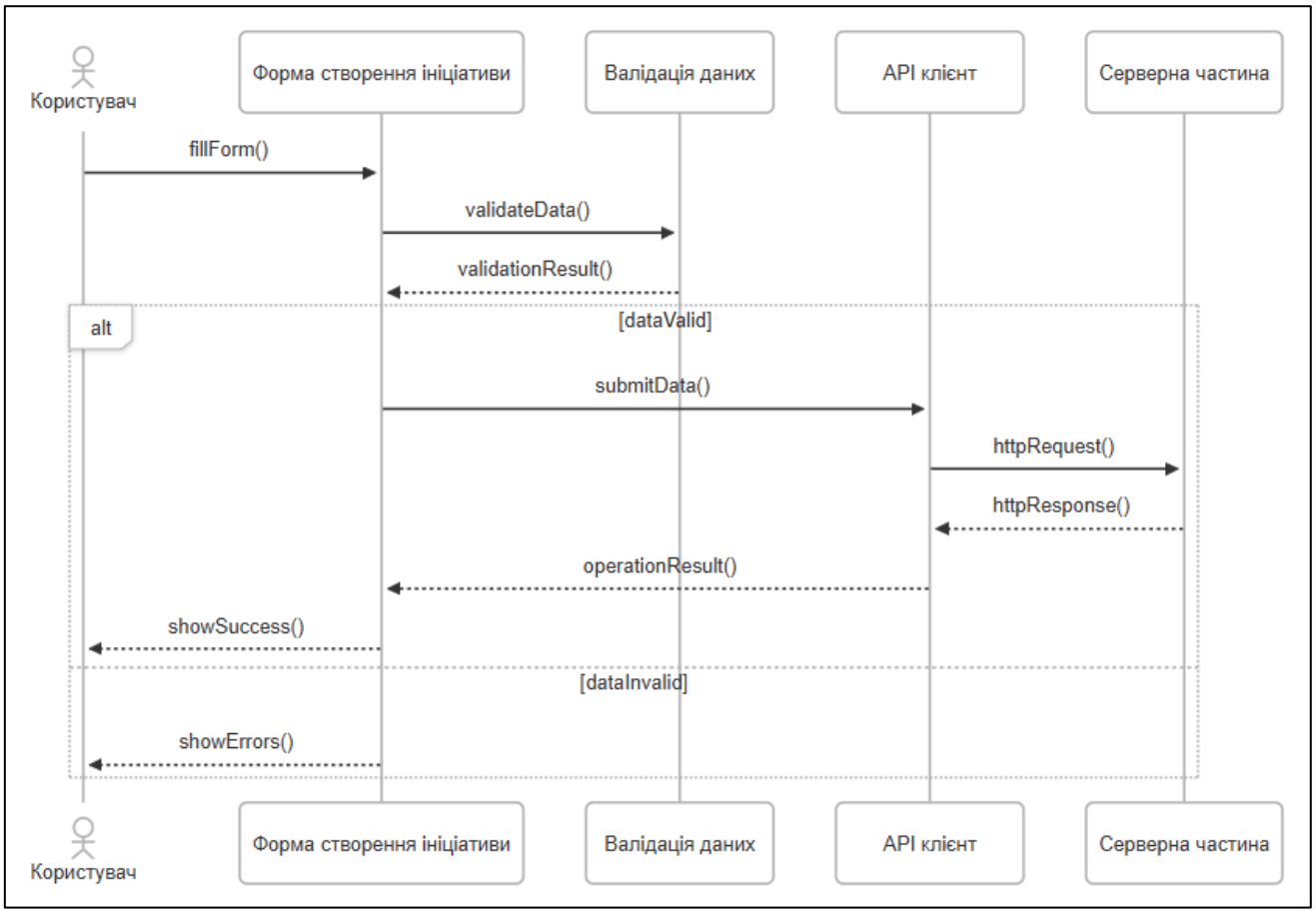

Рисунок 4.2 – Діаграма послідовності (рисунок зроблено самостійно)

Діаграма діяльності відображає процес здійснення пожертви користувачем. Користувач вибирає ініціативу, вводить суму пожертви, система валідує суму, користувач вибирає спосіб оплати, підтверджує пожертву, система відправляє запит на сервер та відображає результат операції (див. рис. 4.3).

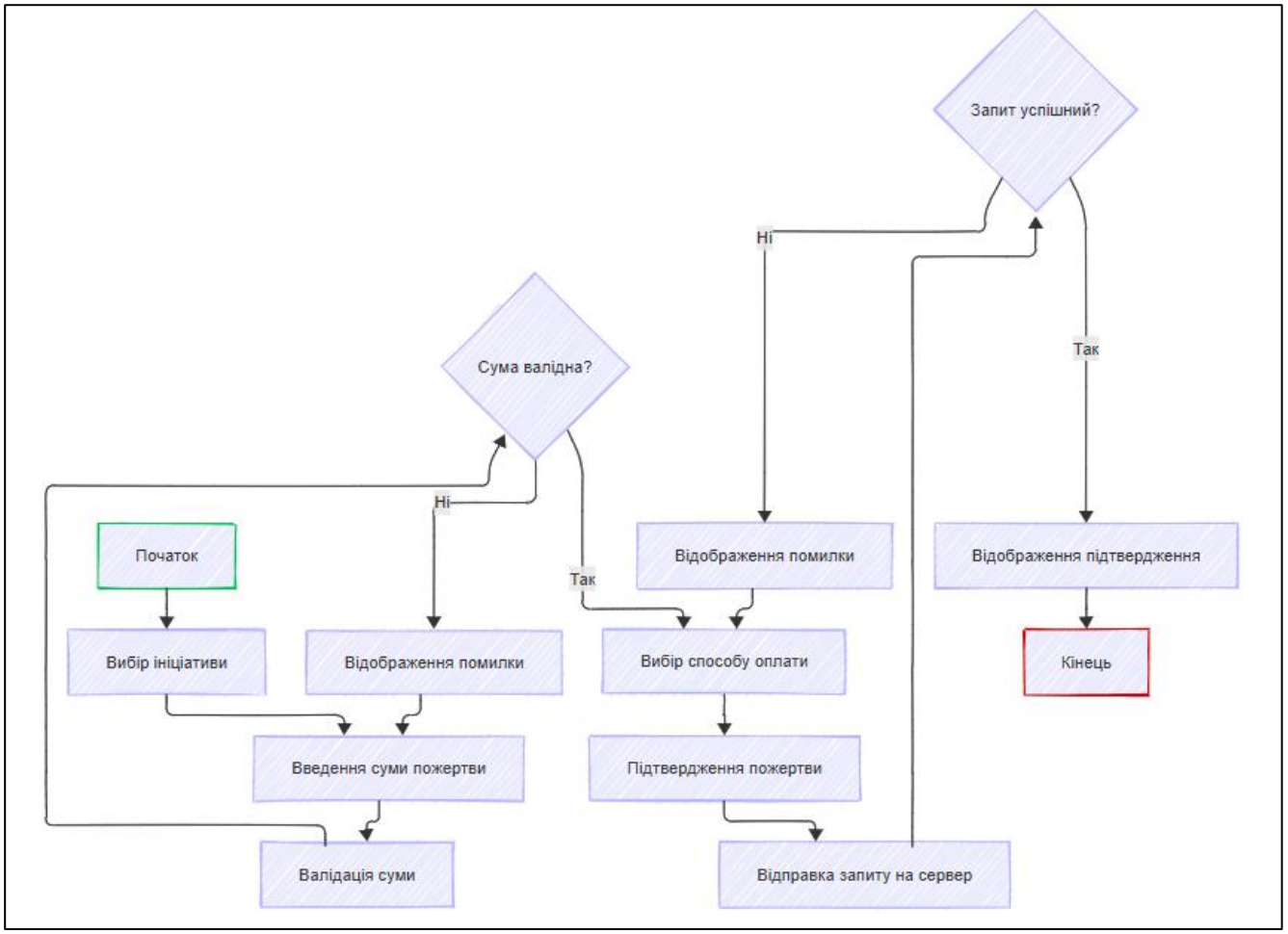

Рисунок 4.3 – Діаграма діяльності (рисунок зроблено самостійно)

Клієнтська частина взаємодіє з серверною частиною через RESTful API з використанням HTTP-запитів. Для виконання запитів використовується вбудована функція `fetch` або бібліотека `axios`, яка надає додаткові можливості для роботи з HTTP-запитами.

Основні типи запитів, які використовуються в клієнтській частині:

– get – для отримання даних (наприклад, список ініціатив, інформація про конкретну ініціативу);

– post – для створення нових ресурсів (наприклад, створення нової ініціативи, реєстрація користувача);

– put/patch – для оновлення існуючих ресурсів (наприклад, редагування ініціативи, оновлення профілю користувача);

– delete – для видалення ресурсів (наприклад, видалення ініціативи).

– для оптимізації взаємодії з сервером використовуються наступні підходи:

– серверні дії (server actions) – дозволяють виконувати мутації даних на сервері безпосередньо з клієнтських компонентів;

– кешування даних – дозволяє зменшити кількість запитів до сервера шляхом кешування результатів попередніх запитів;

– оптимістичні оновлення – дозволяють оновлювати інтерфейс користувача до отримання відповіді від сервера, що покращує користувацький досвід.

4.2 Цікаві методи та алгоритми

У системі статистичні модулі вирішують кілька ключових завдань, які є критично важливими для моніторингу ефективності та прозорості фандрейзингових кампаній і самих ініціатив. По-перше, агрегована статистика FundraisingStat забезпечує швидкий доступ до сукупного обсягу зібраних коштів (TotalCollected) та часу останнього оновлення (UpdatedAt), що дозволяє адміністраторам і власникам ініціатив оперативно оцінювати загальну результативність кампаній. По-друге, щоденна статистика FundraisingDailyIncome фіксує динаміку надходжень у розрізі кожної дати, даючи змогу аналізувати пікові дні, виявляти сезонні тренди і своєчасно коригувати маркетингові стратегії або план комунікацій. Третій рівень статистики – InitiativeStat – агрегує показники всіх пов’язаних зборів для кожної ініціативи, що дає змогу порівнювати окремі кампанії усередині одного проекту та приймати рішення про пріоритетність ресурсів.

Інтерфейс цих розділів повинен задовольняти і бізнес-аналітиків, і операційний персонал: на головній панелі – показники KPI у вигляді карток з ключовими метриками (загальна сума, середній щоденний дохід, кількість донатів), інтерактивні лінійні графіки з розбивкою по днях, гістограми або стовпчикові діаграми задля порівняння декількох кампаній, таблиці з можливістю сортування й фільтрації за датами, категоріями, сумами або кількістю донорів, а також інструменти експорту даних (CSV/PDF). Крім того, для керівників і адміністраторів потрібні віджети прогнозної аналітики – наприклад, трендовий індикатор очікуваного збору за наступний тиждень чи місяць – і сигнали про відхилення від планових показників (наприклад, якщо за перші 3 дні тижня зібрано

менше X% від цільової суми). Така структура статистичних сторінок гарантує глибоке розуміння поточних результатів, оперативне реагування на зміни й можливість стратегічного планування розвитку ініціатив.

Нижче наведено код реалізації відображення статистики по ініціативам та сборам:

```
export default function CollectionStatisticPage() {
  const params = useParams()
  const [collection, setCollection] = useState<(typeof
collectionsMockData)[0] | null>(null)
  const [cumulativeData, setCumulativeData] = useState<{ date: string;
amount: number; rawDate: string }[]>([])
  const [sortedDonors, setSortedDonors] = useState<any[]>([])
  useEffect(() => {
       const id = params.id as string
    const foundCollection = collectionsMockData.find((c) => c.id ===
id)
    if (foundCollection) {
      setCollection(foundCollection)
      let cumulativeAmount = 0
      const cumulativeStats = foundCollection.dailyStats.map((stat) =>
{
        cumulativeAmount += stat.amount
        return {
          date: new Date(stat.date).toLocaleDateString("uk-UA", { day:
"numeric", month: "short" }),
          amount: cumulativeAmount,
          rawDate: stat.date,
        }
      })
      setCumulativeData(cumulativeStats)

          const sorted = [...foundCollection.topDonors].sort((a, b)
=> b.amount - a.amount)
      setSortedDonors(sorted)
    }
```

Компонента CollectionStatisticPage демонструє типову практику реактивного відображення статистики для заданої колекції, спочатку отримуючи параметр id з URL через хук useParams, а потім у useEffect фільтруючи масив collectionsMockData за цим ідентифікатором; у разі успіху знайдена колекція зберігається у стані collection, після чого виконується дві обробки даних: по-перше, кумулятивний обрахунок суми щоденних надходжень із формуванням масиву об’єктів { date, amount, rawDate }, де date – відформатований рядок дати українською, amount – наростаюча сума, а rawDate зберігає оригінальне значення, і цей масив передається в стан cumulativeData; по-друге, список донорів з topDonors сортується у спадному порядку за полем amount і результат зберігається в

sortedDonors, що забезпечує готовність компоненту до візуалізації як графічного представлення динаміки зборів, так і рейтингу найбільших меценатів, причому автоматичне оновлення залежно від зміни параметра id гарантує реактивність і коректність відображених даних.

Далі варто розглянути адаптацію Stripe до запиту на сервер для проведення механізму оплати, нижче наведено код реалізації:

```
"use client";
import { useState, FormEvent } from "react";
import { loadStripe } from "@stripe/stripe-js";
import { Elements, CardElement, useStripe, useElements } from "@stripe/react-stripe-js";
const stripePromise = loadStripe(process.env.NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY!);
export default function DonatePage({ params: { id } }: { params: { id: string } }) {
  return (
    <Elements stripe={stripePromise}>
      <DonateForm fundraisingId={id} />
    </Elements>
  );
}
function DonateForm({ fundraisingId }: { fundraisingId: string }) {
  const stripe = useStripe();
  const elements = useElements();
  const [msg, setMsg] = useState<string>();
  const onSubmit = async (e: FormEvent<HTMLFormElement>) => {
    e.preventDefault();
    const amount = parseFloat((e.currentTarget.amount as HTMLInputElement).value);
    if (amount < 0.01) return setMsg("Мінімум 0.01");

    // 1) створити PaymentIntent
    const res = await fetch(`/api/Fundraising/${fundraisingId}/donate`, {
      method: "POST",
      credentials: "include",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ amount }),
    });
    if (!res.ok) return setMsg("Помилка серверу");
    const { clientSecret } = await res.json();

    // 2) підтвердити оплату
    const card = elements?.getElement(CardElement);
    if (!stripe || !card) return setMsg("Stripe не готовий");
    const { error } = await stripe.confirmCardPayment(clientSecret, {
      payment_method: { card },
    });
    setMsg(error ? error.message! : "Успішно!");
  };
```

Цей компонент для сторінки пожертв Next.js спочатку завантажує Stripe через loadStripe з публічним ключем із змінних середовища й обгортає форму в <Elements>, далі в DonateForm хук useStripe і useElements підключають контекст Stripe, а у формі обробник onSubmit блокує дефолтну поведінку, зчитує значення поля amount, перевіряє, щоб воно було не меншим за 0.01, потім робить fetch POST до контролера Donate на бекенді з передачею { amount } і cookie-авторизацією, із відповіді отримує clientSecret, після чого бере елемент картки через elements.getElement(CardElement) і викликає stripe.confirmCardPayment з clientSecret та payment_method: { card }; якщо в процесі виникає помилка, вона відображається в стані msg, у разі успіху повідомлення змінюється на «Успішно!», а користувач побачить це повідомлення під формою.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі розглянуто декілька сценаріїв тест-кейсів для візуального розуміння процесу розробки та прогресу виконання кожного етапу відповідно технічному завданню. Далі у таблиці 5.1 наведено тестові сценарії.

Таблиця 5.1 – Тест-кейс №1 (таблиця виконана самостійно)

Інформація про тест-кейс			
Ідентифікатор тесту:	Тест-кейс №1		
Власник тесту:	Пахаренко Сергій Олегович		
Дата створення:	23.05.2025		
Мета тесту:	перевірити коректність відображення списку ініціатив (пошук / фільтри / підвантаження), роботу сторінки деталей та повний клієнтський сценарій донату з обробкою успіху / помилки.		
Каталог ініціатив: від списку до прев'ю			
№	Опис випадку	Очікуваний результат	Висновок
1	Відкрити головний маршрут /	На першому екрані відображено 12 карток ініціатив, лічильник «∞» біля скрол-індикатора	Пройдено
2	Прокрутити до кінця видимої області	Тригер «Intersection Observer» підвантажує наступну сторінку; кількість карток +12 без миготіння	Пройдено
3	У верхньому фільтрі вибрати категорію "social"	Список миттєво перебудовується, картки лише з категорією «соціальні», бейдж фільтра активний	Пройдено
4	У полі пошуку ввести «вода»	Лишаються ініціативи, у назві або описі яких є «вода» (без урахування регістру)	Пройдено

Продовження таблиці 5.1.

№	Опис випадку	Очікуваний результат	Висновок

5	Очистити поле пошуку → клік «Reset filters»	Повертається повний список; активний фільтр категорії скидається	Пройдено
6	Відкрити DevTools → Throttle «Slow 3G» → оновити сторінку	Скелетон-прев'ю відображається ≤ 1 с, далі підвантажуються зображення lazy-режимом	Пройдено
7	Вимкнути мережу → оновити	Показується сторінка офлайн-кешу з останніми 20 ініціативами, banner «You're offline»	Пройдено
8	Увімкнути мережу → натиснути «Reconnect»	Дані синхронізуються, banner зникає; час сервіс-синхронізації ≤ 3 с	Пройдено
9	Глибока лінка /initiative/100 з неавторизованим користувачем	Відкривається детальна сторінка ініціативи #100 (тільки читання), кнопка «Support» доступна	Пройдено
10	Відкрити картку ініціативи, де <5 % до цілі	Прогрес-бар має червоний маркер «Urgent», таймер показує залишок днів	Пройдено
Донат і обробка результату через Stripe			
№	Опис випадку	Очікуваний результат	Висновок
1	На сторінці ініціативи натиснути «Support»	Відкривається модальний DonateModal з полем суми та логотипами Stripe/Google Pay	Пройдено
2	Ввести 0 та натиснути Pay	Кнопка неактивна, під полем підказка «Min 0.01 USD»	Пройдено
3	Ввести 25.5 → «Pay»	Stripe PaymentSheet відкривається у режимі тестової картки	Пройдено

Кінець таблиці 5.1.

№	Опис випадку	Очікуваний результат	Висновок
4	Завершити платіж успішно тест-карткою 4242…	PaymentSheet → Success; модалка закривається, зверху toast «Thanks!»	Пройдено
5	Прогрес-бар та сума на сторінці збільшені без	Різниця суми = 25.5, анімація bar + «+25.5»	Пройдено

	перезавантаження (WebSocket)		
6	Перейти у «My Donations» (якщо залогінений)	Перший рядок таблиці — сьогоднішній донат, статус «Succeeded»	Пройдено
7	Повторити крок 3, але натиснути Cancel у PaymentSheet	Модалка закриється, toast «Payment canceled», сума не змінюється	Пройдено
8	Запустити donate, але вимкнути Wi-Fi перед підтвердженням	PaymentSheet повертає Network Error; UI показує червоний toast, модалка лишається відкрита	Пройдено
9	Отримати webhook payment_intent.succeeded із затримкою (backend симуляція)	Через ≤ 5 с після webhook UI синхронізується (навіть у фоні вкладки) й додає суму	Пройдено
10	Донатити суму, що перевищує limit goal → webhook	На UI кампанія змінює статус «Completed», кнопка «Support» прихована, banner «Goal reached!»	Пройдено

ВИСНОВОК

У ході проходження передатестаційної практики було опрацьовано концепцію клієнтського інтерфейсу системи збору коштів, який має об’єднувати два принципово різні сценарії взаємодії – швидку одноразову підтримку проєктів з боку доброчинця та повноцінне керування кампаніями з боку ініціатора. Дослідження показало, що успішність пожертв істотно зростає, коли візуальне сприйняття одразу занурює користувача в контекст збору, а всі наступні дії зведені до декількох інтуїтивних кроків без затримок або фрикцій. Запроєктований інтерфейс враховує ці особливості: тематичний банер створює емоційне занурення, а полегшена форма внеску з мінімальним навантаженням на користувача сприяє швидкому прийняттю рішень і формуванню відчуття причетності завдяки миттєвому оновленню статистики кампанії.

На рівні технічної концепції клієнтський інтерфейс спроєктовано на основі компонентного веб-фреймворку з чіткою ієрархією багаторазових елементів. Передбачено двобічну синхронізацію даних через веб-канали, що забезпечує оновлення суми збору та статусів кампаній у реальному часі. Також закладено використання серверних дій, поступового завантаження зображень та часткової гідратації сторінок, що має забезпечити швидке відображення інтерфейсу навіть на малопотужних мобільних пристроях.

Загальна архітектура клієнтської частини базується на модульному підході з чітким розподілом логіки по сервісах. Передбачено автоматизований CI/CD-конвеєр, що дозволяє оновлювати застосунок без простоїв і з мінімальним ручним втручанням. Для ініціаторів кампаній розроблено концепцію аналітичного екрана з динамічними графіками надходжень, рівнем залучення аудиторії та показниками конверсії.

Уся система безпеки клієнтської частини спроєктована відповідно до принципів «нульової довіри»: кожна критична дія вимагає багатоетапного підтвердження, особисті дані підлягають псевдонімізації, а доступ до функцій жорстко регламентується за ролями користувачів. Передбачено відповідність

основним регуляторним вимогам – GDPR, PSD2 та актуальним нормам AML/KYC, що у перспективі забезпечить правову сумісність у різних юрисдикціях.

Таким чином, під час практики сформовано концепцію клієнтського інтерфейсу, що базується на принципі «єдиного вікна» для доброчинців і організаторів, поєднуючи емоційний дизайн, передбачувану логіку взаємодії та високий рівень інформаційної безпеки. Створена архітектурна основа відкриває широкі перспективи для подальшої реалізації – включно з інтеграцією рекомендаційних систем, підключенням додаткових платіжних каналів, підтримкою багатомовності та впровадженням технологій блокчейн-прозорості для забезпечення повної довіри користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. GoFundMe’s «2024 Year in Help»: основні тенденції глобальної щедрості [Електронний ресурс]. – Режим доступу: https://www.nonprofitpro.com/article/gofundmes-2024-year-in-help-report-highlights-global-generosity/? – Дата звернення: 13.05.2025.

2. Статистика GoFundMe: динаміка кампаній і пожертв [Електронний ресурс]. – Режим доступу: https://www.sci-tech-today.com/stats/gofundme-statistics/? – Дата звернення: 13.05.2025.

3. Kickstarter у цифрах: ключові факти та показники [Електронний ресурс]. – Режим доступу: https://www.searchlogistics.com/learn/statistics/kickstarter-stats-facts/? – Дата звернення: 13.05.2025.

4. Глобальна аналітика краудфандингу за версією Coolest Gadgets [Електронний ресурс]. – Режим доступу: https://www.coolest-gadgets.com/crowdfunding-statistics/? – Дата звернення: 13.05.2025.

5. Про платформу Dobro.ua: місія та показники [Електронний ресурс]. – Режим доступу: https://dobro.ua/en/about_us/? – Дата звернення: 13.05.2025.

6. Offline-first-підхід у мобільних додатках: пояснення та реалізація на Flutter [Електронний ресурс]. – Режим доступу: https://medium.com/nerd-for-tech/offline-first-application-what-does-it-mean-and-how-to-implement-this-concept-in-your-next-flutter-ff40f4b62c6c. – Дата звернення: 13.05.2025.

7. P99-затримка: чому важлива та як її виміряти [Електронний ресурс]. – Режим доступу: https://snowdreamstudios.ch/blog/p99-latency/? – Дата звернення: 13.05.2025.

8. Вимоги до силової автентифікації клієнта: підручник платіжної платформи [Електронний ресурс]. – Режим доступу: https://stripe.com/en-pl/unsupported-browser?location=%2Fen-pl%2Fguides%2Fstrong-customer-authentication%3Futm_source%3D – Дата звернення: 13.05.2025.

9. Створення офлайн-додатків на Flutter: синхронізація даних і робота без мережі [Електронний ресурс]. – Режим доступу: https://code.zeba.academy/offline-apps-flutter-syncing-data-handling-connectivity-issues/ – Дата звернення: 13.05.2025.

10. Push-сповіщення та CRM-стратегії: бенчмарк мобільних повідомлень [Електронний ресурс]. – Режим доступу: https://batch.com/ressources/etudes/benchmark-notifications-push-crm-mobile? – Дата звернення: 13.05.2025.

11. Riva, Michele. Real-World Next.js: Build Scalable, High-Performance, and Modern Web Applications Using Next.js, the React Framework for Production. Packt Publishing, 2022. 384 с. ISBN 978-1-80107-349-3.

12. Bierman, Gavin; Abadi, Martín; Torgersen, Mads. Programming TypeScript: Making Your JavaScript Applications Scale. O'Reilly Media, 2019. 272 с. ISBN 978-1-4920-7936-3.

13. Посилання на код програми на GitHub. GitHub. URL: https://github.com/NurePakharenkoSerhii/2025_B_PI_PZPI-21-1_Pakharenko_S_O (дата звернення: 01.06.2025).