Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Гребенюка Владислава Ігоровича
Прізвище, ім'я, по батькові студента(ки)

на здобуття ступеня вищої освіти Бакалавра

Вебзастосунок для обліку та автоматизації процесів військових зборів
Назва теми

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ. 230132.01.05.ПЗ

Виконав студент IV курсу, група ІПЗс-23-1 _____ Владислав ГРЕБЕНЮК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник канд. пед. наук, доцент _____ Оксана ОНИШКО
Науковий ступінь, звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. тех. наук, доцент _____ Юрій ФОРКУН
Посада Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії
програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

3 червня 2026 р.

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
02 01 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Гребенюку Владиславу Ігоровичу
Прізвище, ім'я, по батькові студента

1. Тема роботи Вебдодаток для обліку та автоматизації процесів військових зборів

Керівник роботи Онишко Оксана Григорівна, канд. пед. наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 20.01.2026 р. № 7-КП
2. Строк подання студентом роботи на кафедру 01.06.2026 р.
3. Вихідні дані до роботи Матеріали переддипломної роботи
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області та постановка задач, проєктування програмного забезпечення, програмна реалізація та тестування застосунку.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Форкун Ю. В., канд. тех. наук доцент	25.01.26	25.05.26
Антиплагіат	Форкун Ю. В., канд. тех. наук доцент	25.01.26	25.05.26

7. Дата видачі завдання «_20_» січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Ознайомлення з тематикою дипломного проєктування, визначення та узгодження індивідуальної теми кваліфікаційної роботи (КвР)	01.12– 31.12.2025	
2 Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог.	01.01 – 20.02.2026	
3 Проєктування програмного забезпечення	21.02 – 20.03 2026	
4 Програмна реалізація з використанням відповідних засобів розроблення. Тестування ПЗ	21.03 – 30.04.2026	
5 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 – 25.05.2026	
6 Попередній захист КвР	Травень 2026	Згідно графіка
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.05.2026	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2026	

Студент _____ Владислав ГРЕБЕНЮК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Оксана ОНИШКО
Підпис Ім'я, ПРІЗВИЩЕ

АНОТАЦІЯ

Тема кваліфікаційної роботи: Вебдодаток для обліку та автоматизації процесів військових зборів

Автор роботи: Гребенюк Владислав Ігорович

Керівник роботи: Онишко Оксана Григорівна

Пояснювальна записка: 82 с., 43 рис., 4 табл., 4 дод., 41 джерело.

Графічна частина: 3 креслення.

Метою кваліфікаційної роботи є проєктування та програмна реалізація вебдодатку для автоматизації та обліку військових зборів, що дозволить підвищити ефективність адміністративних процесів, забезпечити оперативне формування аналітичної звітності та мінімізувати часові витрати на опрацювання даних.

У роботі проведено детальний змістовий аналіз предметної області волонтерського краудфандингу та існуючого програмного забезпечення. Визначено функціональні та нефункціональні вимоги до системи, спроєктовано монолітну клієнт-серверну архітектуру з використанням хмарних сервісів.

Для програмної реалізації клієнтської частини застосовано бібліотеку React та утилітарний CSS-фреймворк Tailwind. Серверну інфраструктуру, систему авторизації та управління нереляційною базою даних реалізовано на базі екосистеми Firebase.

У результаті створено готовий до експлуатації програмний продукт, який забезпечує мультиканальну оплату, та багатофункціональну панель адміністрування з аналітикою для волонтерів.

25.04
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.230132.01.05.ПЗ	Пояснювальна записка	82		
2	А4		Завдання на кваліфіка-	1		
			ційну роботу			
3	А4		Анотація	1		
			Графічні документи			
4	А3	КвРІПЗ. 230132.01.05.Е8	Діаграма класів	1		
5	А3	КвРІПЗ.230132.01.05.Е8	Діаграма послідовностей	1		
6	А3	КвРІПЗ.230132.01.05.Е8	Діаграма станів	1		

Змін.	Аркуш	№ докум.	Підпис	Дата	КвРІПЗ. 230132.01.05.ВД		
Виконав		Гребенюк В.І			Вебдодаток для обліку та автоматизації процесів військових зборів	Літ	Аркуш 1 / Аркушів 1
Керівник		Онишко О.Г					
Рецензен						ХНУ, ІПЗс-23-1	
Н. Контр.		Форкун Ю.В.					
Зав. каф.		Бедратюк Л.П					

ЗМІСТ

ВСТУП

Сучасний етап розвитку світової цивілізації характеризується тотальною цифровізацією всіх сфер життєдіяльності, що зумовлює перехід від традиційних методів управління до інтелектуальних автоматизованих систем. В умовах воєнного стану та необхідності постійної підтримки високого рівня обороноздатності країни, трансформація процесів у військовому секторі набуває стратегічного значення. Ефективність функціонування оборонної інфраструктури безпосередньо залежить від швидкості опрацювання інформації, точності обліку ресурсів та оперативності взаємодії між різними ланками управління.

Процеси організації та проведення військових зборів є складним багатоаспектним завданням, що включає реєстрацію учасників, ведення актуальних баз даних резервістів, облік їхніх військово-облікових спеціальностей, контроль за проходженням медичних оглядів та координацію логістичних процесів. На сьогодні у багатьох підрозділах та центрах комплектування спостерігається значний відсоток використання застарілих методів документообігу та фрагментарного використання електронних таблиць, що не забезпечують цілісності даних. Це призводить до виникнення низки критичних проблем: дублювання інформації, високої ймовірності людських помилок, складності швидкого пошуку потрібних фахівців у критичні моменти та відсутності єдиного захищеного середовища для спільної роботи координаторів.

Актуальність теми кваліфікаційної роботи зумовлена нагальною потребою у створенні спеціалізованого програмного інструментарію, який би дозволив автоматизувати рутинні операції та забезпечити прозорість обліку учасників військових зборів. Використання веборієнтованої архітектури при розробці такого додатку є найбільш обґрунтованим рішенням, оскільки це забезпечує кросплатформеність, можливість одночасної роботи багатьох користувачів з

різними правами доступу та легкість оновлення системи без необхідності встановлення ПЗ на клієнтські машини. Крім того, централізоване зберігання даних у поєднанні з сучасними методами шифрування та автентифікації дозволяє досягти необхідного рівня безпеки, що є першочерговою вимогою для систем військового призначення.

Об’єктом дослідження є процеси автоматизації обліку та координації військових зборів. Предметом дослідження є методи, технології та інструментальні засоби розробки вебдодатку для забезпечення ефективного функціонування системи обліку.

Метою кваліфікаційної роботи є проєктування та програмна реалізація вебдодатку для автоматизації та обліку військових зборів, що дозволить підвищити ефективність адміністративних процесів, забезпечити оперативне формування аналітичної звітності та мінімізувати часові витрати на опрацювання даних.

Для досягнення поставленої мети визначено наступні завдання:

− виконати огляд існуючих аналогів та обґрунтувати доцільність власної розробки;

− сформувати функціональні та нефункціональні вимоги до вебдодатку;

− спроєктувати клієнт-серверну архітектуру системи;

− розробити логічну та фізичну структуру бази даних;

− спроєктувати інтерфейс користувача з урахуванням різних ролей доступу;

− здійснити програмну реалізацію вебдодатку;

− провести тестування розробленого програмного забезпечення.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовий аналіз предметної області

В умовах тривалих бойових дій забезпечення оборонного сектору значною мірою спирається на волонтерський краудфандинг.

Предметною областю даного дослідження є комплекс інформаційних, фінансових та комунікаційних процесів, що супроводжують ініціалізацію, ведення, облік та звітування благодійних зборів на потреби Збройних Сил України.

Сучасний волонтерський рух перетворився на складну екосистему, де ефективність залучення коштів напряму залежить від рівня організації та технологічного забезпечення самого процесу.

Детальний аналіз життєвого циклу типового військового збору дозволяє виділити низку специфічних проблем та «вузьких місць», з якими стикаються ініціатори (волонтери) та благодійники (донори).

Проблематика ініціалізації (старту) збору. Запуск нової фандрейзингової кампанії супроводжується значним адміністративним та когнітивним навантаженням на волонтера.

Для успішного старту необхідно не лише отримати та верифікувати запит від військовослужбовців, але й сформувати чітку фінансову мету, врахувати комісії платіжних систем, коливання валютних курсів та відкрити цільові рахунки на різних платформах (банки, криптовалютні гаманці, міжнародні платіжні системи).

Відсутність єдиного інструменту для структурування цих вхідних даних призводить до того, що на етапі старту волонтери часто припускаються помилок у розрахунках або публікують неповну інформацію, що одразу знижує рівень початкової довіри.

Найскладнішим етапом є просування збору в інформаційному просторі.

Через велику кількість одночасних кампаній спостерігається висока конкуренція за увагу аудиторії. Щоб «розкрутити» збір і підтримувати динаміку надходжень, волонтер змушений постійно генерувати контент, нагадувати про мету та візуалізувати прогрес. При використанні ручного обліку (електронні таблиці, нотатки) синхронізація залишків з різних банків та публікація актуального прогресу займає багато часу.

Затримки в оновленні інформації демотивують потенційних донорів, оскільки вони не бачать реального впливу свого внеску в режимі реального часу.

Український феномен фінансування армії базується на масовості: левова частка необхідних багатомільйонних сум формується за рахунок тисяч дрібних переказів (мікродонатів) від широкого кола осіб. Це створює величезне навантаження на систему обліку.

Обробка тисяч транзакцій вручну для зведення загального балансу є нераціональною тратою людського ресурсу. Система має бути здатною автоматично агрегувати дані про надходження з різних джерел.

Враховуючи високий рівень стресу та інформаційної втоми в суспільстві, критичним фактором успіху збору є показник конверсії (відсоток людей, які побачили збір і здійснили переказ). З точки зору користувацького досвіду (UX), взаємодія донора з інформацією про збір має бути максимально швидкою та інтуїтивно зрозумілою.

Будь-яке перевантаження інтерфейсу зайвою інформацією, складною термінологією, заплутаними таблицями чи незручним копіюванням реквізитів призводить до відмови від здійснення донату.

Інтерфейс як для донора, так і для волонтера має слідувати принципу мінімалізму: користувач повинен за кілька секунд зчитати ключову інформацію (хто збирає, на що, скільки зібрано, скільки залишилося) та зробити внесок в один клік.

Сфера волонтерського краудфандингу та забезпечення військових потреб є надзвичайно динамічною, а створення спеціалізованих вебдодатків для цієї

галузі включає багато нюансів і особливостей, які потрібно врахувати на початку роботи над проєктом. В першу чергу потрібно проаналізувати реалізацію фінансово-облікової системи як окрему предметну підобласть проєкту.

Проєктування та реалізація вебдодатку для обліку зборів є складним та багатоаспектним процесом, що вимагає ретельного підходу. Починається цей процес з визначення базової концепції: розробники аналізують цільову аудиторію (волонтери, донори, військові), визначають ключовий функціонал (бізнес-логіку), вимоги до безпеки та прозорості даних.

Важливим є визначитись, в якому візуальному стилі та з яким користувацьким досвідом (UX) буде виконуватись проєкт, щоб розуміти, які технічні вимоги стоятимуть перед клієнтською (Frontend) та серверною (Backend) частинами. Встановлення чітких цілей та завдань визначає напрямок архітектурної реалізації застосунку.

Наступним етапом розроблення є вибір технологічного стеку та необхідних інструментів, або ж аналіз того, які готові архітектурні патерни можна використати для проєкту, що дозволять реалізувати задуману концепцію в повному обсязі.

Це включає роботу з базами даних, сервісами кешування, API платіжних систем та криптографічним захистом. Сучасні вебдодатки, як і інші складні програмні системи, мають багаторівневу структуру. Вищі рівні (інтерфейс) залежать від нижніх (робота з даними), але не навпаки. Залежність нижнього рівня від верхнього створює циклічну залежність, яку варто уникати.

Такі залежності створюють небажані зв'язки між компонентами, що робить програмне забезпечення нестабільним і утруднює його масштабування. Це особливо важливо для систем, що працюють з фінансовими даними та персональною інформацією.

Після створення базового програмного коду розробники переходять до етапу тестування та відлагодження. Вебдодаток проходить технічне тестування (включаючи нагрузочне тестування та перевірку на вразливості), перевірку

логіки транзакцій та збір фідбеку від тестової групи для виявлення та виправлення помилок. Наступним кроком є оптимізація продуктивності бази даних та адаптація інтерфейсу під різні пристрої. Після цього продукт розгортається на робочому сервері (продакшені).

Весь цей процес допомагає розробникам створити надійний та зручний інструмент, який задовольнить потреби волонтерів, а також забезпечить високий рівень довіри з боку суспільства.

Після розгортання системи розробники продовжують її підтримувати. Це включає в себе підключення нових платіжних методів, виправлення знайдених недоліків та вдосконалення інтерфейсу на основі аналітики поведінки користувачів.

Після визначення проблем, які потрібно буде вирішувати при реалізації технічної частини, потрібно проаналізувати головну властивість таких систем – їхню типологію, як окрему і основну предметну область проєкту.

Як напрямок роботи було обрано вебдодаток, що поєднує в собі риси CRM-системи (Customer Relationship Management) та краудфандингової платформи. Цей клас систем є одним із найбільш затребуваних у некомерційному та волонтерському секторах. Він поєднує в собі строгий фінансовий облік та соціальну взаємодію, створюючи унікальне середовище, яке пропонує користувачам прозорість та зручність. Основна особливість такого додатку полягає в тому, що волонтер може ефективно керувати десятками різних зборів, автоматизувати рутину, в той час як донор отримує максимально спрощений шлях для здійснення благодійного внеску.

У цьому класі систем великий акцент приділяється обробці даних у реальному часі.

Система повинна миттєво реагувати на нові транзакції, оновлюючи прогрес-бари зборів.

Волонтери можуть використовувати різноманітні інструменти аналітики для визначення найбільш ефективних каналів залучення коштів.

При цьому, взаємодія користувача з системою відбувається через інтуїтивно зрозумілі панелі управління (дашборди), що дозволяє швидко адаптуватися до функціоналу без додаткового навчання.

Перед тим, як визначитись, хто є кінцевим користувачем і які в нього вимоги, важливо визначитись із «контекстом використання» (UX-сетингом), адже це впливає на проєктування інтерфейсу. Контекст використання – це визначальний фон, у якому користувач взаємодіє з додатком. Він включає в себе емоційний стан, наявність вільного часу, пристрій доступу та загальну атмосферу терміновості, що супроводжує військові збори. Контекст використання безпосередньо впливає на патерни поведінки користувача. Військові збори часто відбуваються в умовах високого стресу та інформаційного перевантаження. Ця атмосфера вимагає від інтерфейсу максимального мінімалізму, відсутності нав'язливої реклами та складних багатокрокових форм. Якщо для розважальних платформ припустима складна навігація, то для системи донатів ключовим правилом є «правило трьох кліків».

Звернувшись до статистики щодо того, які пристрої найчастіше використовуються для здійснення благодійних внесків (рисунок 1.1), у проєкті було вирішено обрати підхід Mobile-First (орієнтація в першу чергу на мобільні пристрої) (Рисунок 1.1).

Рисунок 1.1 - Статистика пристроїв що використовуються для здійснення благодійних внесків

Проведений змістовий аналіз предметної області дозволив комплексно оцінити специфіку організації, ведення та фінансового обліку благодійних військових зборів. Встановлено, що незважаючи на критичну важливість волонтерського краудфандингу для швидкого забезпечення потреб оборонного сектору, існуючі підходи до адміністрування цих процесів є переважно ручними та неоптимізованими. Використання розрізнених неспеціалізованих засобів призводить до фрагментації даних, надмірного адміністративного навантаження на ініціаторів зборів та суттєво ускладнює своєчасне формування прозорої публічної звітності. Окремою проблемою є перевантаженість інформаційного простору, що вимагає від інструментів збору коштів максимальної когнітивної простоти та адаптованості під мобільні пристрої для швидкого здійснення мікродонатів.

Отже, об'єктивно існує нагальна потреба у реінжинірингу поточних процесів (переходу від моделі «Як є» до «Як має бути») шляхом розробки спеціалізованого програмного рішення. Створення єдиного веборієнтованого додатку дозволить автоматизувати агрегацію фінансових потоків, спростити публікацію звітних документів та забезпечити інтуїтивно зрозумілий користувацький досвід для благодійників. Це повністю підтверджує актуальність обраної теми кваліфікаційної роботи та формує базис для подальшого аналізу існуючих програмних аналогів і визначення конкретних системних вимог до розроблюваного продукту.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Для якісного аналізу існуючого програмного забезпечення у сфері краудфандингу та волонтерського обліку, необхідно врахувати специфічні особливості цього класу систем. Платформи для збору коштів відомі своєю необхідністю балансувати між глибокою фінансовою аналітикою для

організатора та максимальною простотою і швидкістю здійснення переказу для кінцевого донора.

При аналізі наявного програмного забезпечення для обліку військових зборів, в першу чергу потрібно зосередитись на оцінці його функціональності, механізмів інтеграції з банківськими сервісами (платіжними шлюзами), зручності створення нових кампаній, а також інструментів ведення публічної звітності. Потрібно оцінити, наскільки добре платформа агрегує фінансові потоки з різних джерел, чи належним чином реалізована система візуалізації прогресу, і які можливості щодо налаштування та адміністрування пропонуються волонтерам.

Аналізуючи існуючі програмні рішення (як спеціалізовані банківські сервіси, так і універсальні інструменти електронного документообігу), можна простежити за останніми тенденціями у сфері фінансових технологій (FinTech) та спробувати ідентифікувати ключові переваги й недоліки. Це дозволить сформувати чіткий перелік вимог до функціоналу, архітектури та користувацького досвіду для розроблюваного вебдодатку, щоб він ефективно вирішував виявлені проблеми предметної області.

До розгляду було взято популярні сервіси та вебзастосунки, що дозволяють здійснювати донати , та узагальнено автоматизовувати процеси зборів.

Одним із найбільш розповсюджених та затребуваних інструментів для збору коштів в Україні є сервіс «Банка» від необанку Monobank (Рисунок 1.2). Цей інструмент став де-факто стандартом для волонтерських зборів завдяки глибокій інтеграції в банківський застосунок та максимальному спрощенню процесу здійснення донату.

Сервіс «Банка» характеризується високим рівнем довіри з боку користувачів та пропонує наступні технічні та функціональні переваги:

− максимальна конверсія: можливість здійснення переказу в один клік через Apple Pay, Google Pay або безпосередньо в мобільному застосунку Monobank. Це критично важливо для збору мікродонатів, оскільки будь-які

додаткові кроки (введення номера картки, ПІБ тощо) значно знижують кількість успішних транзакцій;

– візуалізація прогресу: наявність інтерактивного прогрес-бару, який відображає накопичену суму у реальному часі відносно встановленої мети. Це створює елемент гейміфікації та мотивує донорів «закрити» збір;

– прозорість надходжень: публічний доступ до перегляду всіх вхідних транзакцій (із прихованими персональними даними), що забезпечує базовий рівень підзвітності волонтера;

– наявність API: сервіс надає програмний інтерфейс для отримання виписки за «Банкою», що теоретично дозволяє інтегрувати дані про збір у сторонні системи.

Рисунок 1.2 – Сервіс «Банка» від Monobank

Проте, незважаючи на значні переваги для донора, сервіс «Банка» має низку суттєвих недоліків з точки зору професійного адміністрування та автоматизації волонтерської діяльності:

– обмежений інструментарій звітності: сервіс дозволяє бачити надходження, але абсолютно не передбачає функціоналу для фіксації витрат. Волонтер не може прикріпити чеки, інвойси або акти прийому-передачі безпосередньо до «Банки», що змушує його створювати додаткові звіти на інших ресурсах;

− відсутність CRM-можливостей: волонтери, які ведуть одночасно десятки зборів на різні потреби (дрони, авто, медицина), не мають інструментів для їх групування, категоризації та аналізу ефективності. Кожна «Банка» існує як окрема сутність без зв'язку з конкретними запитами військових підрозділів;

− мінімальна інформативність сторінки: функціонал опису збору обмежений лише невеликим текстовим полем та однією ілюстрацією. Це не дозволяє детально розписати складні технічні потреби або надати посилання на документальне підтвердження запиту;

− залежність від однієї екосистеми: повноцінне управління збором та детальний перегляд виписок доступні лише клієнтам одного банку, що створює певні обмеження при масштабуванні діяльності.

Таким чином, сервіс «Банка» є ідеальним шлюзом для прийому платежів, проте він не є повноцінною системою автоматизації волонтерської діяльності.

Це підтверджує доцільність розробки спеціалізованого вебдодатку, який би використовував переваги зручних платежів, але водночас забезпечував професійний рівень обліку, звітності та взаємодії з військовими підрозділами.

Наступним розглянемо вебсайт Сила Сердець (рисунок 1.3).

Сайт належить українській громадській організації (некомерційному благодійному фонду) «Сила Сердець».

Основна інформація про платформу: Організація займається наданням гуманітарної допомоги найуразливішим верствам населення (людям з інвалідністю, сиротам, біженцям, багатодітним сім'ям, літнім людям), а також цілеспрямовано допомагає бійцям Територіальної оборони та Збройних Сил України.

Багато волонтерських ініціатив та фондів змушені створювати власні вебсайти-візитки для консолідації інформації про залучення коштів та публікації звітності. На прикладі платформи української громадської організації «Сила Сердець» можна виділити базовий функціонал, необхідний для роботи легального фонду: наявність інформаційних блоків про поточні гуманітарні та

військові потреби, інтеграція платіжних реквізитів та окремий розділ публічної звітності.

Переваги: Такі індивідуальні рішення забезпечують високий рівень довіри благодійників, оскільки містять офіційну інформацію про організацію, публічну оферту та консолідовані звіти про виконану роботу у структурованому вигляді.

Недоліки: Розробка та технічна підтримка подібних платформ вимагає значних фінансових витрат та залучення профільних IT-спеціалістів. Більшість таких сайтів функціонують переважно як статичні інформаційні ресурси і не мають вбудованої автоматизованої системи для трекінгу мікродонатів у реальному часі. Оновлення прогресу зборів та публікація звітів вимагає ручного втручання адміністратора (контент-менеджера), що значно уповільнює операційну діяльність фонду.

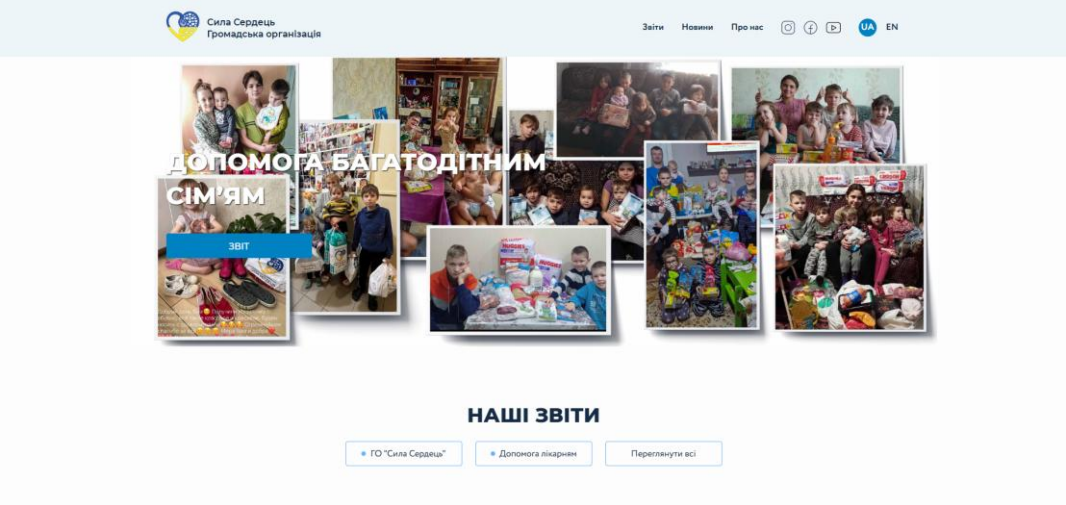


Рисунок 1.3 – Вебсайт «Сила Сердець»

Найвищий рівень автоматизації та користувацького досвіду у сфері волонтерського краудфандингу демонструють пропрієтарні вебплатформи великих інституційних фондів, яскравим представником яких є українська інформаційна благодійна система система Благодійного фонду Сергія Притули (рисунок 1.4).

З технічної точки зору, це складний, високонавантажений вебдодаток, який тісно інтегрований із внутрішніми ERP (Enterprise Resource Planning) системами фонду для управління логістикою та фінансами.

Платформа фонду реалізує передові підходи до організації благодійних зборів (особливо так званих «мегазборів»):

Багатоканальний еквайринг: система підтримує безшовну інтеграцію з десятками платіжних методів, включаючи LiqPay, PayPal, Patreon, криптовалютні шлюзи та прямі банківські перекази за реквізитами (IBAN), що забезпечує глобальне охоплення донорів.

Високорівневий UX/UI дизайн: інтерфейс платформи еталонно використовує елементи гейміфікації та візуалізації. Прогрес-бари, динамічні лічильники зібраних коштів та чітка категоризація потреб (мілітарний напрямок, гуманітарний, медицина) максимально спрощують навігацію та мотивують благодійників.

Структурована звітність: платформа має окремі модулі для публікації масштабних фінансових звітів та візуальних підтверджень закупівель, що підтримує абсолютний рівень суспільної довіри.

Переваги: Ця система є ідеальним прикладом того, як повинна працювати платформа для масового збору коштів. Вона забезпечує максимальну конверсію, абсолютну прозорість, витримує величезні навантаження під час вірусних зборів та автоматизує левову частку рутинної роботи фінансового відділу фонду.

Головним недоліком такого програмного забезпечення є його закрита, пропрієтарна архітектура. Це продукт формату Enterprise (корпоративне рішення), який розроблений виключно для обслуговування внутрішніх бізнес-процесів одного конкретного фонду. Ця система не надається за моделлю SaaS (Software as a Service).

Тобто, незалежний волонтер, невеликий локальний фонд чи підрозділ не можуть зареєструватися на цій платформі, щоб використовувати її потужний інструментарій для адміністрування та просування власних локальних зборів.

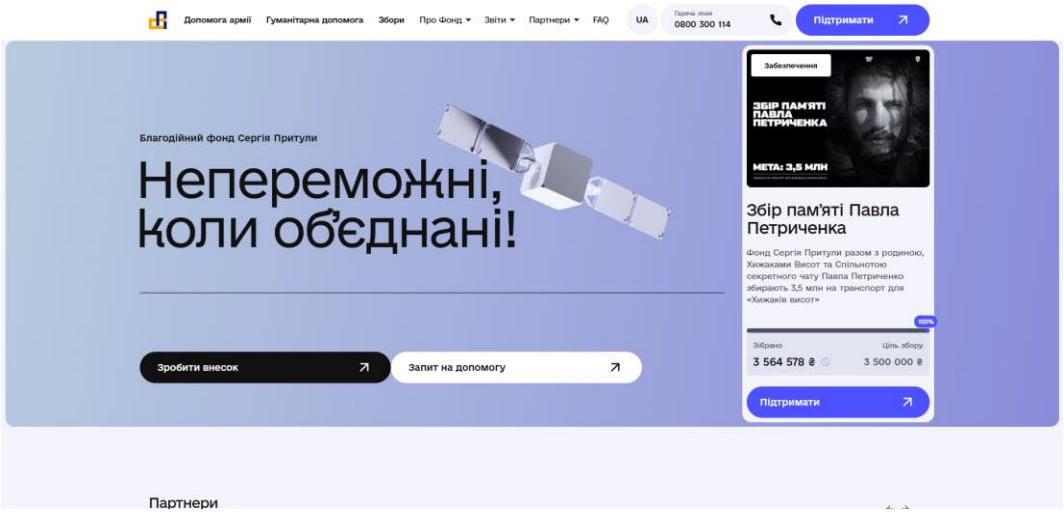


Рисунок 1.4 - система Благодійного фонду Сергія Притули

Підсумовуючи розгляд наявного програмно-технічного забезпечення у сфері волонтерського краудфандингу, можна зробити обґрунтований висновок про критичну необхідність розвитку та вдосконалення подібних інформаційних систем.

Аналіз поточної ситуації яскраво демонструє, що на сьогоднішній день значна частина оперативного матеріально-технічного забезпечення армії тримається саме на успішності та регулярності громадських благодійних зборів.

Відповідно, програмні платформи, які обслуговують ці процеси, перестали бути просто фінансовими інструментами і перетворилися на стратегічно важливий елемент підтримки обороноздатності.

Враховуючи колосальне щоденне навантаження на волонтерський сектор та зростаючу потребу суспільства у зручних і прозорих механізмах взаємодії, попит на оптимізовані вебдодатки для обліку зборів лише зростатиме.

Тому розробка власної спеціалізованої системи, яка автоматизує рутину, забезпечить надійність даних та надасть максимально простий інтерфейс для кінцевого користувача, є не лише актуальним інженерним завданням, а й гострою практичною потребою сьогодення яку потрібно вирішувати та впроваджувати подібні рішення в повсякденне життя задля збереження можливості людей виживати.

1.3 Визначення вимог до вебдодатку

За підсумками детального дослідження предметної області та існуючих схожих програмних систем, було розроблено технічне завдання та список функціональних і нефункціональних вимог до створюваного рішення.

Створюваний вебдодаток для автоматизації та обліку військових зборів мусить мати декілька особливостей порівняно з його аналогами. Головною відмінністю від інших подібних рішень є те, що платформа повинна бути комплексною та складатися з двох ізольованих частин: клієнтської (для донорів) та адміністраторської (для волонтерів). Також має бути передбачена система авторизації та реєстрації для ідентифікації користувачів.

Важливою вимогою є можливість здійснення благодійного внеску як авторизованим, так і неавторизованим (гостьовим) користувачам, щоб максимально спростити процес підтримки зборів. При цьому зареєстровані користувачі отримують доступ до розширеного функціоналу: особистого кабінету, де зберігається їхня статистика, історія донатів та остання активність. Впроваджується система гейміфікації – за певну активність користувачам нараховуються віртуальні відзнаки (ачівки). Також користувачі повинні мати змогу залишати коментарі безпосередньо до зборів та до своїх транзакцій.

В каталозі зборів повинен бути реалізований зручний пошук, а також сортування та фільтрація кампаній за різними критеріями (наприклад, за статусом, категорією потреби чи терміновістю).

Користувачам необхідно надати можливість переглядати загальні новини платформи, а також специфічні звіти та оновлення по кожному окремому збору.

Для адміністратора буде передбачено окрему панель управління (дашборд) з розгорнутою статистикою у вигляді інтерактивних графіків.

Адміністратор повинен мати повний та досить простий контроль (створення, редагування, видалення) над зборами, документами, новинами та загальним контентом сайту.

Беручи до уваги всі зазначені вище особливості, можна скласти перелік функціональних можливостей, які має надавати програма:

Для звичайного користувача (донора):

− реєстрація та авторизація;

− здійснення донату без обов'язкової реєстрації;

− перегляд списку поточних та завершених зборів;

− фільтрація, сортування та пошук зборів;

− детальний перегляд інформації про вибраний збір (опис, прогрес, документи); − перегляд загальних новин сайту та звітів по конкретному збору;

− залишення коментарів до зборів та власних донатів; − доступ до особистого кабінету (для авторизованих); − перегляд власної статистики донатів та історії активності; − отримання та перегляд здобутих відзнак (ачівок) за активність.

Для адміністратора (волонтера):

− авторизація із захищеними правами доступу;

− перегляд загальної аналітики та статистики платформи за допомогою графіків;

− створення, редагування та видалення карток зборів;

− завантаження та управління документацією (чеки, акти, фотозвіти) для кожного збору;

− управління модулем новин (створення, редагування, видалення публікацій);

− модерація коментарів користувачів; − редагування загальної інформації на сайті;

− перегляд детальної історії всіх транзакцій.

Також сформовано наступні вимоги до зовнішнього інтерфейсу користувача та інші нефункціональні вимоги:

− повна адаптивність до мобільних пристроїв (Mobile-First підхід), оскільки більшість донатів здійснюється зі смартфонів;

− сучасний, стриманий дизайн із контрастними кольорами для зручного зчитування інформації та фокусування уваги на прогресі зборів;

− на головній сторінці повинні відразу знаходитися найактуальніші збори зі зручними фільтрами та пошуком;

− плавна анімація елементів інтерфейсу (зокрема прогрес-барів зібраних коштів) для покращення візуального досвіду;

− процес здійснення внеску має бути інтуїтивно зрозумілим і займати мінімальну кількість кліків;

− висока швидкодія клієнтської частини та відсутність необхідності повного перезавантаження сторінок при навігації.

Для формалізації та наочного подання функціональних вимог до розроблюваної системи доцільно використати інструментарій об'єктно-орієнтованого аналізу, зокрема уніфіковану мову моделювання (UML).

Для опису поведінки системи з точки зору кінцевих користувачів будується діаграма прецедентів (Use Case diagram). Що ідеально відображає всі зв’язки між користувачами , адміністраторами та їх можливості при роботі з системою та її окремими функціями.

Виділимо основні ролі користувачів, які взаємодіятимуть із вебдодатком. Їх перелік та характеристика наведені у таблиці 1.1.

Таблиця 1.1 - Характеристика акторів системи

Анонімний користувач (Гість)	Відвідувач платформи без створеного облікового запису. Йому доступний публічний функціонал: перегляд загального переліку зборів, використання інструментів пошуку, ознайомлення з новинами та звітністю, а також виконання транзакцій (донатів) в анонімному режимі.

Кінець таблиці 1.1

Авторизований користувач	Зареєстрований учасник системи, який окрім базових можливостей гостя отримує доступ до персонального профілю. Це дозволяє йому відстежувати власну статистику внесків, здобувати віртуальні нагороди за активність та взаємодіяти з контентом через систему коментарів.
Адміністратор платформи	Користувач із найвищим рівнем привілеїв. Керує життєвим циклом зборів (створення, модерація, закриття), відповідає за публікацію фінансових звітів, модерує комунікацію користувачів та аналізує статистичні дані через графічні дашборди.

Визначивши акторів системи та варіанти використання, було побудовано діаграму варіантів використання (рисунок 1.5).

На цій діаграмі відображено функціональні вимоги до системи, де основними акторами виступають Анонімний користувач, Авторизований користувач та Адміністратор.

Діаграма демонструє взаємозв'язки між дійовими особами та ключовими процесами, такими як реєстрація, перегляд каталогу, здійснення транзакцій та керування контентом, що дозволяє візуалізувати межі системи та основні сценарії взаємодії користувача з програмним продуктом.

Варто зазначити, що побудова цієї моделі є критично важливим етапом архітектурного проектування, оскільки вона дозволяє формалізувати очікування від системи ще до початку етапу безпосереднього написання програмного коду. Завдяки візуалізації зв'язків між акторами та прецедентами, розробник отримує цілісне бачення функціонального простору застосунку. Це дає можливість на ранніх стадіях життєвого циклу проекту виявити потенційні прогалини у бізнес-логіці та усунути суперечності, що можуть виникнути при взаємодії різних типів користувачів.

Рисунок 1.5 - Діаграма варіантів використання

1.4 Висновки. Постановка задачі

У цьому розділі було детально розглянуто предметну область волонтерського краудфандингу та забезпечення військових потреб. Ми проаналізували, як саме зараз організовуються благодійні збори, які проблеми виникають при ручному обліку фінансів і чому розробка нового спеціалізованого вебдодатку є дійсно нагальною потребою. Щоб виокремити найбільш вдалі рішення та зрозуміти очікування користувачів, було досліджено існуючі програмні аналоги — від популярних банківських сервісів («Банка» Monobank) до закритих платформ великих фондів.

Цей аналіз дав змогу чітко визначити цільову аудиторію проєкту (донорів та волонтерів-адміністраторів) і сформувати повний перелік функціональних та нефункціональних вимог до системи. Для наочного розуміння того, як

користувачі будуть взаємодіяти з вебдодатком, ми побудували UML-діаграму варіантів використання. Усе це дозволило сформувати чітке технічне завдання, яке задає конкретні рамки проєкту та значно спростить подальший процес розробки.

Маючи вичерпну специфікацію та розуміння того, як має працювати фінальний продукт, можна поставити конкретні задачі, які необхідно виконати на наступних етапах кваліфікаційної роботи.

Перелік поставлених задач:

− провести аналіз існуючих архітектурних підходів і патернів проєктування для вебдодатків та обрати найоптимальніші для нашого проєкту;

− спроєктувати структуру бази даних та описати зв’язки між основними сутностями (користувачі, збори, документи, транзакції);

− розробити макети користувацького інтерфейсу (UI) з урахуванням зручності для мобільних пристроїв;

− остаточно визначити технологічний стек (фреймворки та бібліотеки) для реалізації серверної та клієнтської частин системи;

− розробити основні модулі бізнес-логіки (обробка донатів, система гейміфікації з відзнаками, управління контентом);

− створити та запрограмувати клієнтський інтерфейс (Frontend), реалізувавши необхідні динамічні елементи та анімації;

− визначитися з методами тестування та провести перевірку готового вебдодатку на наявність помилок;

− проаналізувати результати тестування та зробити загальні висновки по виконаній роботі.

2. ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ

2.1 Аналіз та вибір архітектури веб-додатка

Етап проєктування архітектури є визначальним кроком у розробленні програмного забезпечення, оскільки він закладає фундамент для подальшої масштабованості, безпеки та стабільності системи. Архітектурне рішення визначає загальну організацію вебдодатка, принципи взаємодії його компонентів та розподіл функціональних обов'язків. З метою вибору найбільш оптимального підходу для створення системи обліку та автоматизації військових зборів було проведено детальний аналіз декількох класичних і сучасних архітектурних парадигм.

Монолітна архітектура являє собою підхід, за якого клієнтська частина, бізнес-логіка та рівень доступу до бази даних об'єднані в один цілісний проєкт (єдину кодову базу) і розгортаються як єдиний вузол. Цей підхід відзначається простотою на початкових етапах розроблення, оскільки значно полегшує процеси локального тестування, налаштування та первинного розгортання.

У монолітній системі всі компоненти мають прямий і швидкий доступ один до одного, що гарантує високу продуктивність при обробці внутрішніх запитів та спрощує контроль за цілісністю даних.

Мікросервісна архітектура пропонує розділення глобальної системи на набір дрібних, автономних сервісів, кожен з яких відповідає за виконання конкретної бізнес-задачі та взаємодіє з іншими через мережеві протоколи. Головною перевагою такої парадигми є абсолютна гнучкість у масштабуванні окремих модулів.

Водночас цей підхід вимагає розбудови надзвичайно складної інфраструктури розгортання та моніторингу мережевої взаємодії між вузлами, що суттєво ускладнює підтримку проєкту.

Архітектура на базі хмарних сервісів (Backend-as-a-Service) передбачає повне делегування серверної частини сторонньому провайдеру.

Це забезпечує високу швидкість розроблення інтерфейсу, проте створює жорстку прив'язку до конкретної платформи (Vendor lock-in) та суттєво обмежує розробника у можливостях реалізації складної, нестандартної бізнес-логіки на боці сервера.

Для узагальнення сильних та слабких сторін розглянутих підходів результати аналізу зведено у компактну таблицю 2.1.

Таблиця 2.1 - Порівняльний аналіз архітектурних підходів

Тип архітектури	Переваги	Недоліки
Монолітна	Єдина кодова база, швидке розгортання, простота тестування та висока внутрішня швидкодія.	Складність масштабування окремих вузлів при надвисоких навантаженнях.
Мікросервісна	Висока відмовостійкість, незалежне масштабування модулів.	Надмірна складність інфраструктури, високі витрати на підтримку.
BaaS (Хмарна)	Відсутність потреби адмініструвати сервери, швидкий старт	Обмеження кастомної логіки, залежність від стороннього провайдера

Вибір остаточного архітектурного рішення здійснюється на основі методу порівнянь та виключень відповідно до поставлених вимог. Розроблюваний вебдодаток повинен забезпечувати стабільну обробку фінансових даних, мати надійну систему авторизації та єдину точку контролю за контентом. Шляхом виключення відкидається мікросервісна архітектура через її надмірну інфраструктурну складність (Overengineering), яка є економічно та технологічно недоцільною для реалізації проєкту такого масштабу.

Хмарна архітектура (BaaS) також виключається, оскільки система обліку військових зборів вимагає повного контролю над серверним середовищем та базою даних для забезпечення максимальної безпеки і незалежності від сторонніх сервісів. Таким чином, найкращим рішенням, яке повністю задовольняє вимоги проєкту, визнано монолітну архітектуру.

Залежно від обраного типу архітектури, монолітний проєкт концептуально розбито на три взаємопов'язані логічні рівні.

Перший рівень представлення (Presentation Layer). Він відповідає за взаємодію з користувачем, візуалізацію даних та обробку подій у браузері. Для його реалізації використовується сучасна бібліотека React, що дозволяє створити динамічний, компонентно-орієнтований інтерфейс із високою швидкістю відгуку завдяки технології Virtual DOM.

Другий рівень бізнес-логіки (Business Logic Layer). Цей рівень інкапсулює всі правила роботи додатку: обробку вхідних запитів, валідацію даних, логіку агрегації транзакцій, управління сесіями та формування звітів. Він забезпечує зв'язок між клієнтським інтерфейсом та базою даних, гарантуючи коректність виконання всіх операцій.

Третій рівень доступу до даних (Data Access Layer). Він відповідає за безпечне зберігання та отримання інформації про користувачів, збори та документи. Цей рівень забезпечує виконання запитів до бази даних, підтримку цілісності зв'язків між сутностями та резервне копіювання інформації. Такий багаторівневий розподіл у межах єдиного моноліту гарантує структурованість коду та полегшує подальше супроводження програмного продукту.

Окрім загального вибору типу архітектури, невід'ємною частиною проєктування є визначення шаблонів (патернів) розробки, які застосовуватимуться безпосередньо на рівні програмного коду.

Відповідно до обраного підходу та використання бібліотеки React, для клієнтської частини системи ключовим є застосування компонентного шаблону (Component-Based Pattern). Цей підхід дозволяє декомпозувати складний

інтерфейс користувача на дрібні, незалежні та інкапсульовані блоки — наприклад, картки військових зборів, форми здійснення транзакцій, елементи навігації, модальні вікна та графіки статистики. Таке розділення гарантує можливість багаторазового використання коду (Reusability), знижує його надмірність та значно спрощує процеси локалізації і виправлення помилок.

Для ефективного управління глобальним станом додатку (State Management) збереженням даних про поточного авторизованого користувача, обраних фільтрів у каталозі зборів та статусу завантаження інформації застосовується шаблон централізованого сховища, що базується на архітектурі Flux (або аналогічних сучасних рішеннях). Цей патерн створює єдиний і суворо передбачуваний потік даних (Unidirectional Data Flow). Впровадження такого підходу є критично важливим для систем, що працюють з фінансами, оскільки воно повністю виключає ризик розсинхронізації інформації на різних екранах під час виконання пожертв.

На рівні взаємодії бізнес-логіки з базою даних активно застосовується поведінковий шаблон «Спостерігач» (Observer).

Цей патерн дозволяє клієнтському додатку «підписуватися» на оновлення специфічних сутностей у базі даних (наприклад, на зміну загальної суми зібраних коштів певного збору). Завдяки цьому будь-яка успішна транзакція миттєво генерує подію, яка автоматично та в реальному часі оновлює індикатори прогресу на екранах усіх активних користувачів без необхідності ручного оновлення сторінки.

Додатково, для ініціалізації головного з'єднання з базою даних використовується твірний шаблон «Одинак» (Singleton).

Він гарантує створення та існування лише одного екземпляра підключення в межах усього сеансу роботи програми, що запобігає витокам пам'яті та суттєво оптимізує споживання системних ресурсів.

У підсумку, інтеграція монолітної архітектури з визначеними сучасними шаблонами проєктування формує надійний, безпечний та високопродуктивний

базис. Затверджене архітектурне рішення повністю задовольняє функціональні та нефункціональні вимоги до вебдодатку і створює необхідне підґрунтя для переходу до наступного етапу розроблення детального проєктування структури бази даних та декомпозиції основних модулів системи.

2.2 Опис декомпозиції

2.2.1 Загальна декомпозиція

На найвищому рівні абстракції загальна декомпозиція системи відображає її фізичний та логічний поділ на основні середовища виконання. Відповідно до обраної клієнт-серверної архітектури (BaaS + SPA), розроблюваний вебдодаток жорстко декомпозується на дві глобальні підсистеми: клієнтську та серверну.

Клієнтська підсистема (Frontend-додаток): Ця частина реалізована на базі бібліотеки React і виконується виключно в середовищі веббраузера користувача. Вона повністю ізольована від прямого доступу до бази даних. Головними завданнями цієї підсистеми є:

− маршрутизація сторінок на стороні клієнта (забезпечення переходів без перезавантаження вікна);

− візуалізація інтерфейсу користувача та рендеринг компонентів (карток, графіків, форм);

− перехоплення та первинна валідація даних, введених користувачем (наприклад, перевірка формату суми донату перед відправкою на сервер);

− управління локальним станом додатка (збереження вибраних фільтрів, статусу авторизації сесії);

− виконання анімацій та забезпечення плавності користувацького досвіду.

Серверна підсистема (BaaS-інфраструктура): Ця частина реалізована через екосистему хмарних сервісів Firebase. Вона є повністю прихованою від кінцевого користувача і виступає єдиним джерелом істини (Single Source of Truth) для всієї платформи. До її завдань входить:

− агрегація та перманентне збереження даних у NoSQL базі Cloud Firestore;

− автентифікація користувачів та генерація захищених токенів доступу;

− забезпечення правил безпеки (Security Rules), які на рівні сервера блокують несанкціоновані транзакції або спроби зміни чужих даних;

− зберігання бінарних файлів (фотографій зборів, PDF-документів звітів) у хмарному сховищі. Такий жорсткий поділ гарантує, що критична бізнес-логіка та фінансові дані надійно захищені на рівні хмарної інфраструктури і не можуть бути скомпрометовані шляхом маніпуляцій із клієнтським кодом.

2.2.2 Модульна декомпозиція

Для безпосередньої програмної реалізації клієнтська та бізнес-логіка розбиваються на окремі функціональні модулі (підсистеми), кожен з яких інкапсулює певну частину предметної області. У розроблюваній системі виділено п'ять ключових взаємодіючих модулів:

− модуль ідентифікації та управління доступом (Identity & Access Module): Відповідає за життєвий цикл сесії користувача. Забезпечує процеси реєстрації, авторизації, відновлення паролів та логіку гостьового доступу (для здійснення неавторизованих донатів). Модуль також реалізує рольову модель управління доступом (RBAC), розмежовуючи права між звичайним донором та адміністратором платформи;

− модуль управління краудфандинговими кампаніями (Campaign Management Module): Відповідає за логіку відображення каталогу зборів. Включає підмодулі пагінації (поступового завантаження списку для економії трафіку), складного пошуку за текстовими описами, а також сортування та фільтрації карток за статусом, категорією потреби чи рівнем терміновості;

− модуль обробки транзакцій та гейміфікації (Transaction & Gamification Module): Є центральним фінансовим та інтерактивним ядром системи. Обробляє

логіку ініціалізації благодійних внесків та оновлює індикатори прогресу зборів у реальному часі за допомогою WebSocket-з'єднань. У фоновому режимі цей модуль аналізує історію транзакцій авторизованого користувача та автоматично нараховує віртуальні відзнаки (досягнення) при виконанні певних умов (наприклад, «Перший донат», «Меценат місяця»);

− модуль контенту та соціальної взаємодії : Інкапсулює логіку роботи з текстовим та медіаконтентом. Забезпечує відображення стрічки загальних новин платформи, публікацію звітних документів (чеків, актів прийому-передачі) безпосередньо у картках зборів. Також включає підмодуль коментування, що дозволяє користувачам залишати повідомлення під кампаніями та власними транзакціями;

− модуль адміністрування та аналітики : Закрита підсистема для волонтерів. Надає інтерфейси CRUD (створення, читання, оновлення, видалення) для управління всім контентом сайту. Включає аналітичний підмодуль, який агрегує фінансові дані зі всієї бази і формує візуальні графіки статистики (динаміка зборів, активність користувачів) для моніторингу ефективності платформи.

2.2.3 Декомпозиція моделі переходів станів

Для забезпечення абсолютної коректності роботи бізнес-логіки необхідно чітко алгоритмізувати життєвий цикл ключових сутностей системи. Декомпозиція моделі переходів станів фокусується на головному об'єкті предметної області — «Благодійному зборі» . Його життєвий цикл являє собою скінченний автомат і розбитий на строго регламентовані наступні стани.

Стан «Створено» : Початковий стан об'єкта. Волонтер ініціалізує картку збору, заповнює опис, додає реквізити та встановлює фінансову мету. У цьому стані об'єкт існує лише в базі даних адміністратора і є повністю невидимим для публічного каталогу (гостей та донорів). Транзакції заборонені.

Стан «Активний» : Перехід у цей стан відбувається після публікації збору адміністратором. Картка з'являється у загальному каталозі, активуються слухачі бази даних реального часу. Система починає приймати транзакції, оновлювати прогрес-бари та дозволяє користувачам залишати коментарі.

Стан «Завершено» : Перехід у цей стан відбувається автоматично, коли сума транзакцій досягає 100% встановленої мети, або ініціюється адміністратором вручну (у разі дострокового закриття). У цьому стані фінансовий шлюз для даного збору блокується здійснення нових донатів стає неможливим. Проте картка залишається видимою в каталозі з позначкою про успішне завершення.

Стан «Підзвітний / Архівний» : Фінальний етап життєвого циклу. Перехід здійснюється після того, як адміністратор завантажує підтверджувальні документи (чеки, акти, фотозвіти) про цільове використання зібраних коштів. Об'єкт переходить у статус «Лише для читання» і слугує для забезпечення прозорості та формування історії успішних кампаній фонду.

Моделювання таких строгих переходів гарантує, що система унеможливить логічні помилки наприклад, не дозволить донорам переказувати кошти на вже закритий збір або не дасть адміністратору опублікувати звіт для кампанії, збір коштів на яку ще активно триває.

2.3 Опис залежностей

Щоб вебдодаток працював стабільно та без помилок, потрібно чітко описати зв'язки між усіма його частинами. Це допоможе зрозуміти, як компоненти взаємодіють між собою, та уникнути ситуацій, коли вони блокують роботу один одного.

Міжрівневі залежності. Система має чітку ієрархію: клієнтський інтерфейс повністю залежить від сервера (бази даних та логіки), але не навпаки. Інтерфейс лише надсилає запити на збереження або отримання даних і чекає на відповідь.

Сервер ніколи сам не звертається до інтерфейсу. Завдяки цьому будь-які зміни в дизайні сайту не зламають логіку обробки платежів чи правила безпеки в базі даних.

Блок міжмодульної залежності. Модулі додатка постійно обмінюються інформацією. Головним є модуль авторизації від нього залежить робота більшості інших частин. Наприклад, модуль адміністрування спочатку перевіряє через нього, чи має користувач права волонтера, і лише потім дозволяє створити новий збір. Також модуль транзакцій безпосередньо залежить від каталогу зборів, адже жоден платіж неможливо провести без прив'язки до конкретної активної кампанії.

Блок міжпроцесних залежностей. Робота інтерфейсу у браузері користувача залежить від швидкості обміну даними з сервером. Щоб сторінка не "зависала" під час завантаження інформації, всі запити відбуваються паралельно. Крім того, інтерфейс залежить від фонових процесів бази даних. Наприклад, коли хтось робить донат, сервер обробляє цю транзакцію і сам надсилає сигнал додатку. Після цього на екранах усіх користувачів миттєво та автоматично оновлюється сума зібраних коштів.

Блок залежності всередині даних. У базі даних інформація тісно пов'язана між собою. Наприклад, запис про фінансову транзакцію не може існувати сам по собі: він обов'язково має посилатися на конкретний збір та на користувача (якщо внесок не анонімний). Відзнаки та коментарі також мають подвійну залежність вони прив'язані і до автора, і до кампанії. Важливою деталлю є фінансова залежність: якщо волонтер видаляє збір, пов'язані з ним транзакції не зникають із бази. Вони залишаються для правильної звітності, змінюється лише їхній статус.

Блок залежності між станами. Зміна статусу благодійного збору залежить від певних умов. Перехід від статусу «Створено» до «Активний» залежить лише від адміністратора — він має вручну опублікувати кампанію. А от перехід до статусу «Завершено» залежить від фінансових даних: він відбувається

автоматично, щойно зібрана сума досягає встановленої мети. Фінальний статус «Підзвітний» залежить від попереднього кроку: система просто не дозволить завантажити чеки та звіти, поки збір не буде успішно завершено.

2.4 Опис інтерфейсу модулів

З урахуванням розробленої трирівневої структури програмного забезпечення, у цьому підрозділі описуються механізми взаємодії між логічними модулями системи. Оскільки вебдодаток реалізовано з використанням сучасних технологій на базі компонентного підходу, роль класичних інтерфейсів об'єктної моделі виконують глобальні стани додатку, програмні контракти сервісних класів та спеціалізовані функції для управління потоками даних. Головною метою проєктування цих інтерфейсів є забезпечення слабкого зв'язування між візуальною частиною та базою даних, що гарантує високу безпеку та легкість подальшого масштабування продукту.

Опис міжмодульних інтерфейсів

Міжмодульні інтерфейси забезпечують обмін даними між незалежними логічними блоками системи без порушення їхньої внутрішньої структури та інкапсуляції. У розроблюваному вебдодатку головним вузлом зв'язку виступає глобальний контекст управління станом, який зберігає інформацію про поточну сесію та глобальні налаштування.

Взаємодія між рівнем представлення (інтерфейсом користувача) та рівнем бізнес-логіки відбувається через шар сервісних об'єктів. Візуальні компоненти ніколи не звертаються до бази даних напряму. Наприклад, коли модуль управління зборами повинен відобразити список актуальних потреб, він викликає відповідний метод сервісного класу. Цей метод формує правильний запит, звертається до хмарного середовища, отримує сирі дані, перетворює їх у стандартизований масив об'єктів і лише після цього повертає візуальному компоненту. Такий міжмодульний інтерфейс гарантує, що у разі зміни структури

бази даних розробнику доведеться оновити лише один сервісний клас, а не переписувати весь візуальний код програми.

Окремо виділяється міжмодульний інтерфейс перевірки прав доступу. Модуль авторизації надає відкритий програмний контракт для всіх інших підсистем. Коли користувач намагається відкрити захищену сторінку (наприклад, панель створення нового збору), модуль маршрутизації звертається до інтерфейсу ідентифікації з запитом на перевірку ролі поточного користувача. Якщо інтерфейс підтверджує наявність прав адміністратора, відбувається перехід; якщо ні — інтерфейс повертає команду блокування доступу та перенаправляє користувача на головну сторінку.

Опис інтерфейсів процесів

У розроблюваній системі всі основні класи бізнес-логіки є відкритими для рівня представлення. Тому інтерфейси процесів складаються з переліку ключових сервісних класів та їхніх публічних методів, які повністю описують усі можливі дії користувачів та адміністраторів на платформі.

Клас управління ідентифікацією користувачів відповідає за процеси реєстрації та безпеки. До його відкритого інтерфейсу належать наступні процеси. Метод реєстрації нового користувача: приймає на вхід адресу електронної пошти, надійний пароль та ім'я користувача; перевіряє унікальність даних, створює новий запис у системі безпеки та повертає згенерований об'єкт профілю. Метод авторизації: проводить звірку введених облікових даних із базою, генерує захищений маркер сесії та ініціалізує робочий простір користувача. Метод відновлення доступу: приймає електронну адресу та ініціює процес відправки посилання для безпечного скидання пароля. Метод отримання поточного стану сесії: працює у фоновому режимі та постійно повертає актуальну інформацію про те, чи перебуває користувач у системі, що дозволяє миттєво змінювати вигляд навігаційного меню.

Клас управління благодійними зборами інкапсулює процеси роботи з волонтерськими кампаніями. Його інтерфейс включає такі методи. Метод

створення нового збору: доступний лише адміністраторам; приймає текстовий опис потреби, цільову суму, банківські реквізити та медіафайли; створює новий документ у базі даних зі статусом чернетки та повертає його унікальний ідентифікатор. Метод отримання каталогу зборів: приймає параметри пошуку, категорію та вказівники для поступового завантаження списку; повертає відфільтрований масив карток зборів. Метод детального перегляду: приймає ідентифікатор конкретного збору та повертає повний набір даних про нього, включаючи прикріплені звіти та коментарі. Метод оновлення статусу: дозволяє адміністратору перевести кампанію з активного стану в статус успішно завершеної.

Клас обробки фінансових транзакцій забезпечує критично важливі процеси фінансового обліку платформи. Його публічний інтерфейс складається з таких функцій. Метод ініціалізації платежу: приймає суму благодійного внеску, ідентифікатор збору та дані донора (або позначку про анонімність); створює в базі даних запис про очікувану транзакцію. Метод прослуховування фінансового прогресу: встановлює постійне з'єднання з сервером та приймає функцію зворотного виклику; щоразу, коли сума збору змінюється, цей метод автоматично повертає нове значення, забезпечуючи плавну анімацію індикаторів на екрані. Метод отримання індивідуальної статистики: приймає ідентифікатор користувача та повертає повну історію всіх його успішних внесків для відображення в особистому кабінеті.

Клас управління системою гейміфікації відповідає за процеси соціального заохочення користувачів. Його інтерфейс містить метод перевірки умов досягнень, який автоматично викликається після кожної успішної транзакції; він аналізує історію користувача та повертає рішення про необхідність видачі нагороди. Метод нарахування відзнаки: додає віртуальну нагороду до профілю донора.

Клас управління документацією та контентом реалізує процеси публічної звітності та модерації. До його інтерфейсу належить метод завантаження

звітного документа, який приймає електронну копію чека чи акта прийому-передачі; він безпечно розміщує файл у хмарному сховищі, генерує публічне посилання і повертає його для подальшого використання. Метод прикріплення звіту: пов'язує завантажений документ із конкретним завершеним збором. Метод модерації коментарів: дозволяє адміністратору платформи приховати або назавжди видалити повідомлення, які порушують правила спільноти.

Такий розгорнутий опис сервісних класів та їхніх методів дозволяє повністю простежити всі інформаційні потоки додатку. Він формує вичерпне розуміння об'єктної моделі та процесів взаємодії, демонструючи, як загальні функціональні вимоги перетворюються на конкретні програмні алгоритми на рівні коду.

2.5 Проєктування інтерфейсу користувача

Етап проєктування інтерфейсу користувача є одним із найважливіших кроків розроблення, оскільки саме візуальна частина визначатиме загальну ефективність майбутнього програмного продукту. Для вебдодатку, який працюватиме з фінансовими транзакціями та волонтерськими зборами, вимоги до якості візуальної складової мають бути надзвичайно високими. Головною метою поточного етапу проєктування є закладення основ для створення інтуїтивно зрозумілого, доступного та безпечного середовища. Воно повинно мінімізувати когнітивне навантаження на людину та дозволяти виконати цільову дію, зокрема здійснити благодійний внесок, за мінімальну кількість кроків.

Особлива увага під час створення макетів має приділятися принципу мобільного пріоритету. Враховуючи специфіку швидкого поширення інформації, очікується, що абсолютна більшість транзакцій здійснюватиметься зі смартфонів. Тому архітектура майбутнього інтерфейсу, розміщення навігаційних панелей та розміри інтерактивних елементів проєктуватимуться з урахуванням зручності управління однією рукою на сенсорних екранах. Процес

здійснення переказу має стати максимально швидким і не повинен вимагати від користувача заповнення зайвих текстових полів. Лише після затвердження базових мобільних макетів інтерфейс буде масштабуватися для екранів настільних комп'ютерів.

Важливим аспектом майбутньої системи має стати забезпечення високого рівня візуальної доступності. Кольорову палітру планується підбирати з урахуванням суворих вимог до контрастності, щоб текст комфортно зчитувався за будь-яких умов зовнішнього освітлення. Загальний дизайн базуватиметься на стриманих відтінках, які мають викликати відчуття довіри та стабільності, не відвертаючи при цьому увагу від головного контенту індикаторів прогресу. Усі динамічні елементи, такі як повідомлення про успішну оплату чи здобуття відзнаки, повинні бути спроєктовані так, щоб інформувати користувача без зайвого перевантаження екрана.

Логіка навігації будуватиметься таким чином, щоб задовольнити вимоги двох цільових груп. Для звичайних відвідувачів і донорів передбачається створення простого каталогу із системою фільтрів та можливістю миттєвого переходу до оплати. Для адміністраторів має бути спроєктована окрема закрита панель управління, де головний акцент робитиметься на чітку візуалізацію статистичних даних у вигляді графіків та зручний доступ до завантаження звітної документації.

Згідно з сучасними стандартами інженерії програмного забезпечення, найкращою практикою перед початком написання коду є етап візуального прототипування. Створення попередніх макетів дозволяє наочно представити архітектуру інформації, перевірити зручність розташування елементів управління та протестувати користувацькі сценарії без витрат часу на програмування.

На основі раніше сформованих вимог розробляється серія візуальних прототипів, які демонструють концептуальний вигляд ключових сторінок майбутнього вебдодатку.

Першим базовим елементом, який зустрічатиме користувача, стане головна панель навігації. Її планується спроєктувати за принципом мінімалізму, щоб забезпечити безперешкодний доступ до основних розділів платформи. Панель міститиме логотип системи зліва, центральний блок посилань (головна сторінка, каталог зборів, розділ звітів та контакти), а також контрастну кнопку для входу в систему або реєстрації, яку буде логічно відокремлено у правій частині екрана (рисунок 2.1).

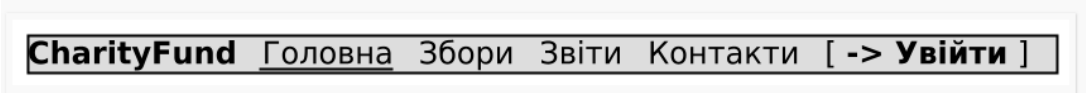


Рисунок 2.1 – Проєктний макет головної панелі навігації вебдодатку

Для забезпечення зручного пошуку актуальних волонтерських кампаній проєктується макет сторінки загального каталогу.

У верхній частині інтерфейсу має бути розміщена зручна панель фільтрації, яка дозволить в один клік відсортувати збори за категоріями (гуманітарні, військові, медичні тощо), а також поле для прямого текстового пошуку за назвою. Нижче розташовуватиметься сітка карток зборів.

Саму картку планується розробити з чіткою візуальною ієрархією: вона міститиме область для тематичного зображення, інформативні статусні позначки, заголовок, короткий опис та контрастний індикатор прогресу.

Цей індикатор виступатиме ключовим елементом, оскільки він одразу показуватиме співвідношення вже зібраних коштів до загальної фінансової мети, мотивуючи долучитися.

Застосування такого підходу до проектування UI-компонента базується на принципах ергономіки та психології сприйняття користувацьких інтерфейсів.

Візуалізація прогресу у вигляді динамічного індикатора дозволяє мінімізувати когнітивне навантаження на користувача, надаючи всю критично важливу інформацію одним поглядом що викликає в користувачів більше довіри до розробленої платформи (рисунок 2.2).

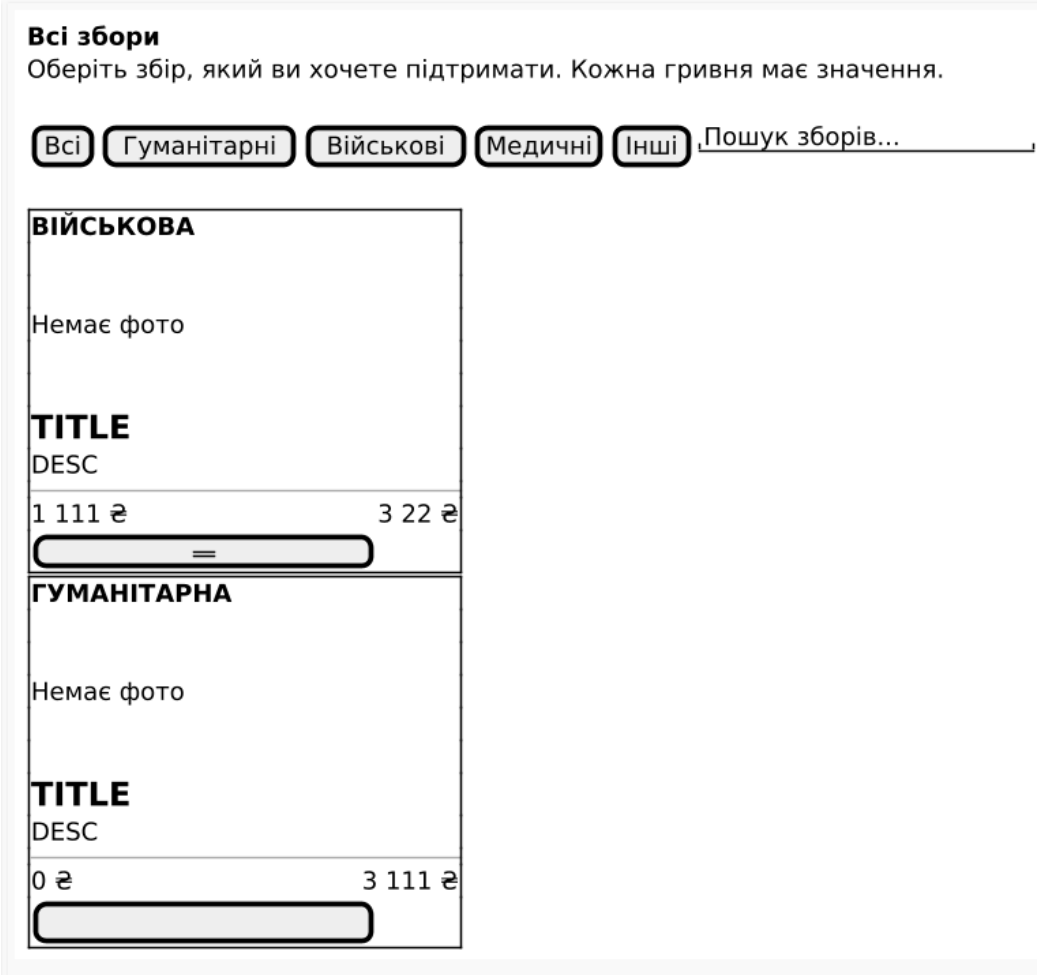


Рисунок 2.2 – Прототип сторінки каталогу благодійних зборів

Найбільш критичним етапом взаємодії користувача з платформою стане сторінка конкретної кампанії та форма здійснення внеску. Макет цієї сторінки передбачається логічно розділити на два функціональні блоки для ефективного використання простору екрана. Зліва буде відведено місце для візуального підтвердження потреби (фотографії чи відео), а справа зосереджуватиметься фінансова інформація та платіжний інтерфейс.

Форма оплати проєктуватиметься з метою максимального пришвидшення транзакції. Користувачу пропонуватиметься ввести суму власноруч або скористатися кнопками швидкого вибору із заздалегідь підготовленими номіналами. Поля для введення імені та короткого текстового повідомлення підтримки планується зробити необов'язковими. Таке рішення приймається свідомо, щоб не створювати штучних бар'єрів і не затримувати процес переказу

коштів. Завершуватиметься блок великою, добре помітною кнопкою підтвердження дії (рисунок 2.3).

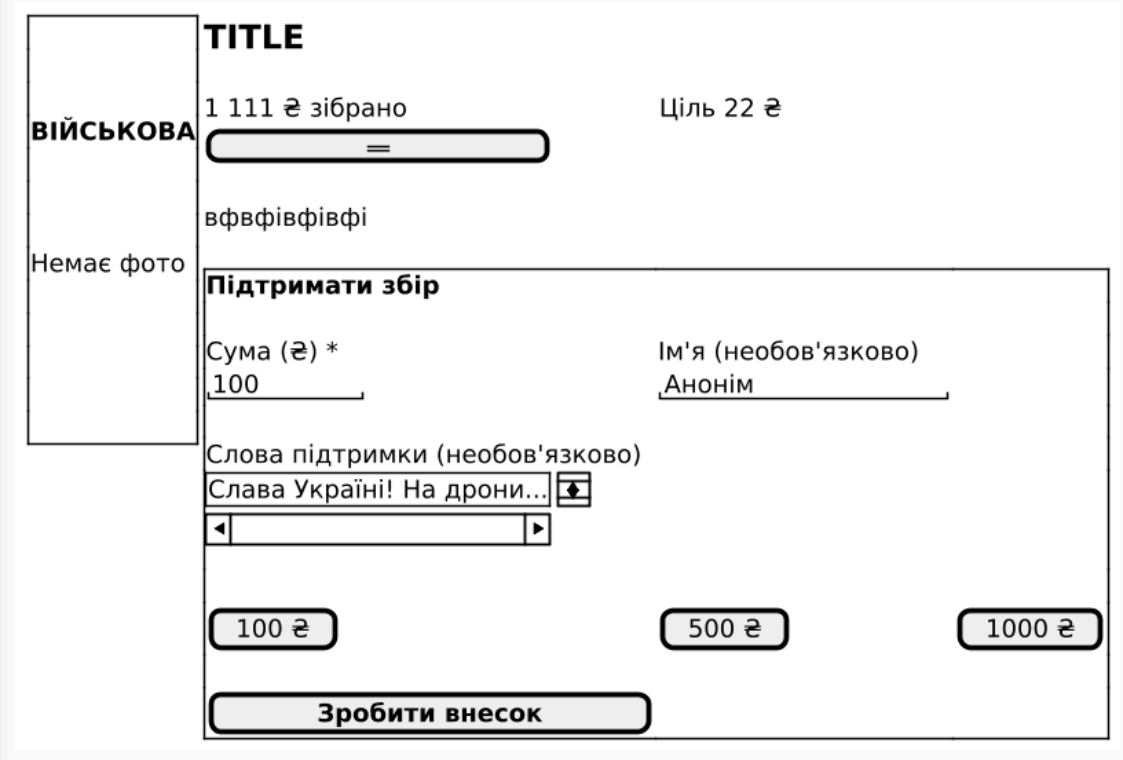


Рисунок 2.3 – Макет форми транзакції

Застосування подібної техніки візуального моделювання планується для всіх інших функціональних розділів вебдодатку, зокрема особистого кабінету донора, панелі управління для волонтерів та модулів завантаження звітності. Такий системний підхід до проєктування інтерфейсів формує чітке концептуальне бачення майбутнього продукту на ранніх етапах розроблення.

Створення повної серії макетів для кожної сторінки дозволить скласти цілісну фінальну картину взаємодії користувача із системою. Головною перевагою цього етапу є можливість детального аналізу логіки переходів між екранами. Це дасть змогу завчасно виявити можливі архітектурні недоліки, такі як перевантаженість певних сторінок даними, зайві кроки при оформленні транзакцій або неочевидність навігації, та оперативно їх усунути ще до початку безпосереднього програмування. Таким чином, затвердження повної бази

візуальних прототипів гарантуватиме високу ергономічність системи обліку зборів та значно мінімізує ризики виникнення логічних помилок під час її технічної реалізації.

2.6 Детальне проєктування модулів

Етап детального проєктування передбачає логічний перехід від загальної архітектури системи до специфікації алгоритмів роботи її окремих внутрішніх модулів. Метою цього етапу є чітке визначення того, як саме оброблятимуться дані всередині кожної підсистеми та як змінюватимуться стани головних об'єктів під час взаємодії з користувачем. Такий рівень деталізації є критично важливим для уникнення логічних колізій на етапі безпосереднього написання програмного коду. У розроблюваному вебдодатку найбільшої уваги потребують модуль управління сесіями та ідентифікацією, а також модуль адміністрування благодійних кампаній.

Логіка модуля управління доступом будуватиметься навколо життєвого циклу користувацької сесії. Після запуску вебдодатку кожен відвідувач за замовчуванням отримує статус неавторизованого гостя. У цьому стані система дозволяє йому переглядати каталог та здійснювати анонімні перекази, проте блокує доступ до персональної статистики. Процес авторизації ініціюватиме зміну стану: після успішної перевірки облікових даних сервер згенерує захищений маркер, і користувач перейде у стан авторизованого донора. Якщо ж система виявить у профілі наявність спеціальних прав доступу, користувачеві буде додатково відкрито перехід у стан адміністратора, що розблокує панель управління волонтера.

Завершення сесії з будь-якого стану гарантовано повертатиме користувача до початкового базового рівня гостя.

Для візуалізації цих процесів розроблено діаграму станів користувача, що демонструє маршрутизацію прав доступу в межах платформи (рисунок 2.4).

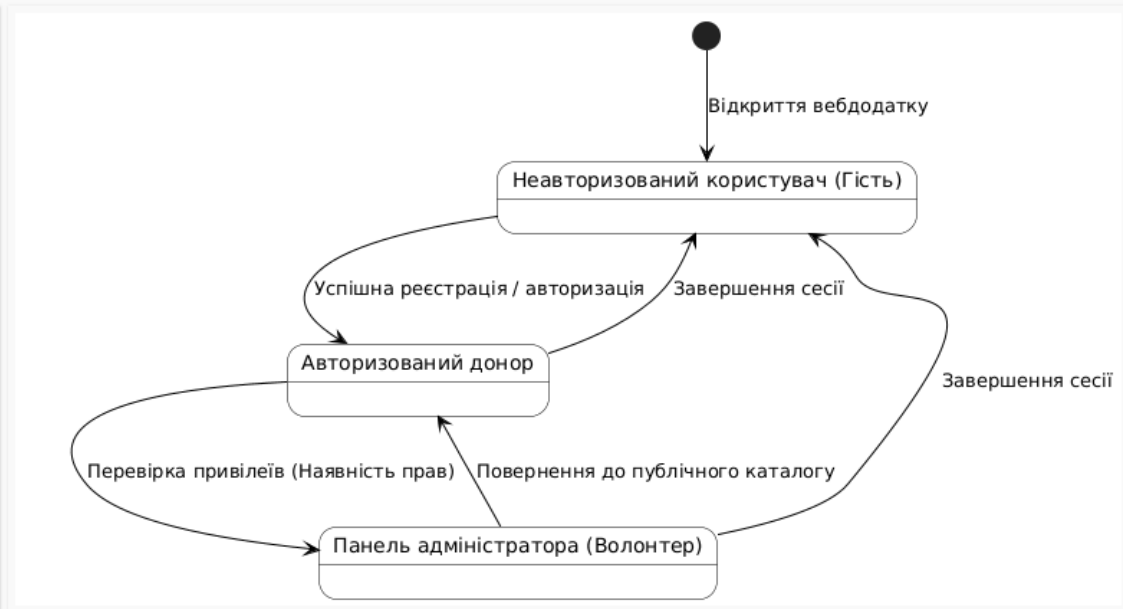


Рисунок 2.4 - Діаграми станів користувача

Модуль управління благодійними кампаніями проєктується як центральне ядро системи, що відповідає за строгий контроль фінансових потоків та звітності. Головним об'єктом цього модуля є картка збору, життєвий цикл якої має бути жорстко регламентованим для забезпечення абсолютної прозорості. Логіка модуля передбачає, що процес створення збору починається зі статусу чорнетки. На цьому етапі волонтер вносить усі необхідні реквізити, описи та фотографії, маючи можливість багаторазово редагувати інформацію без ризику публічного розголосу.

Після перевірки даних об'єкт переводиться в активний стан, під час якого алгоритми модуля відкривають платіжний шлюз та починають приймати транзакції, миттєво перераховуючи відсоток прогресу. Ключовим правилом проєктування є неможливість ручного переведення активного збору в статус підзвітного без попереднього його завершення. Система автоматично переводитиме кампанію у стан завершеної щойно сума транзакцій досягне встановленої мети. Лише після блокування платіжних функцій модуль розблокує для адміністратора інтерфейс завантаження підтверджувальних фінансових документів, переводячи об'єкт у фінальний архівний стан.

Для формалізації цього складного життєвого циклу спроєктовано відповідну діаграму станів благодійного збору (рисунок 2.5).

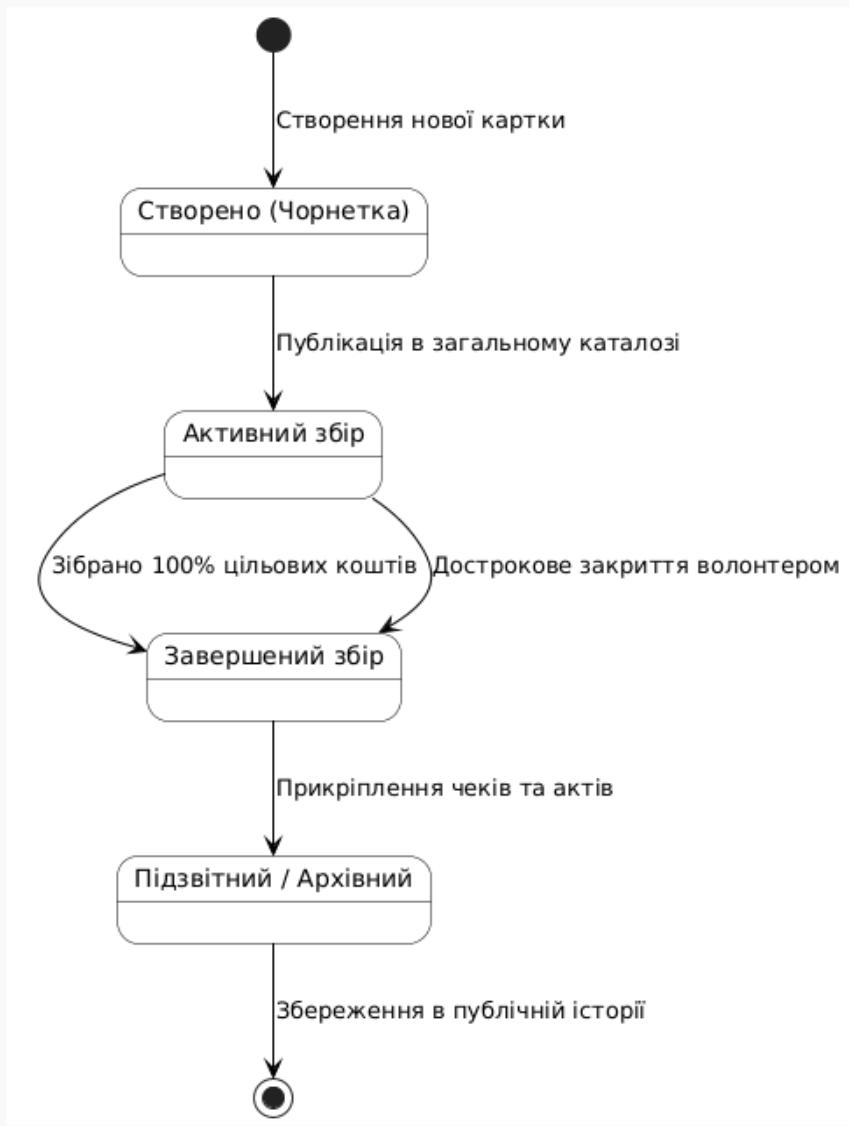


Рисунок 2.5 - Діаграми станів збору

Оскільки здійснення благодійного внеску є основною та критично важливою функцією розроблюваного вебдодатку, процес обробки транзакцій потребує найбільш ретельного проєктування.

Цей бізнес-процес є складним, адже він вимагає абсолютної синхронізації між діями користувача, клієнтським додатком, серверною бізнес-логікою, базою даних та стороннім платіжним шлюзом (еквайрингом).

Будь-яка помилка або розсинхронізація на цьому етапі може призвести до втрати фінансових даних або зниження довіри до платформи.

Планується, що життєвий цикл транзакції складатиметься з двох основних етапів: ініціалізації та асинхронного підтвердження.

На першому етапі донор вводить суму внеску в клієнтському додатку. Інтерфейс відправляє запит до модуля транзакцій, який створює у базі даних попередній запис зі статусом «Очікування».

Після цього модуль звертається до зовнішнього платіжного API, отримує унікальне захищене посилання на оплату та перенаправляє туди користувача.

Другий етап відбувається в асинхронному фоновому режимі. Коли користувач успішно вводить реквізити картки на стороні платіжної системи, банківський шлюз автоматично відправляє на сервер вебдодатку підтверджувальний сигнал (Callback).

Отримавши цей сигнал, бізнес-логіка перевіряє його валідність, змінює статус транзакції на «Успішно» та миттєво додає отриману суму до загального прогресу відповідного збору.

Завдяки використанню технології WebSocket (шаблон «Спостерігач»), оновлені дані з бази миттєво надсилаються до клієнтського додатку, де користувач бачить анімацію успішного переказу без необхідності перезавантажувати сторінку.

Для детальної візуалізації цього складного багатопотокового процесу спроєктовано UML-діаграму послідовностей.

Вона відображає хронологічний порядок обміну повідомленнями між усіма компонентами системи під час здійснення благодійного внеску.

Ця діаграма деталізує життєвий цикл запиту: від моменту ініціації дії користувачем у фронтенд-частині застосунку до фінальної обробки даних на сервері та оновлення стану бази даних у реальному часі та дозволяє чітко відстежити точки критичних затримок або можливих відмов у системі що є не прийнятним в проектах подібного типу (рисунок 2.6).

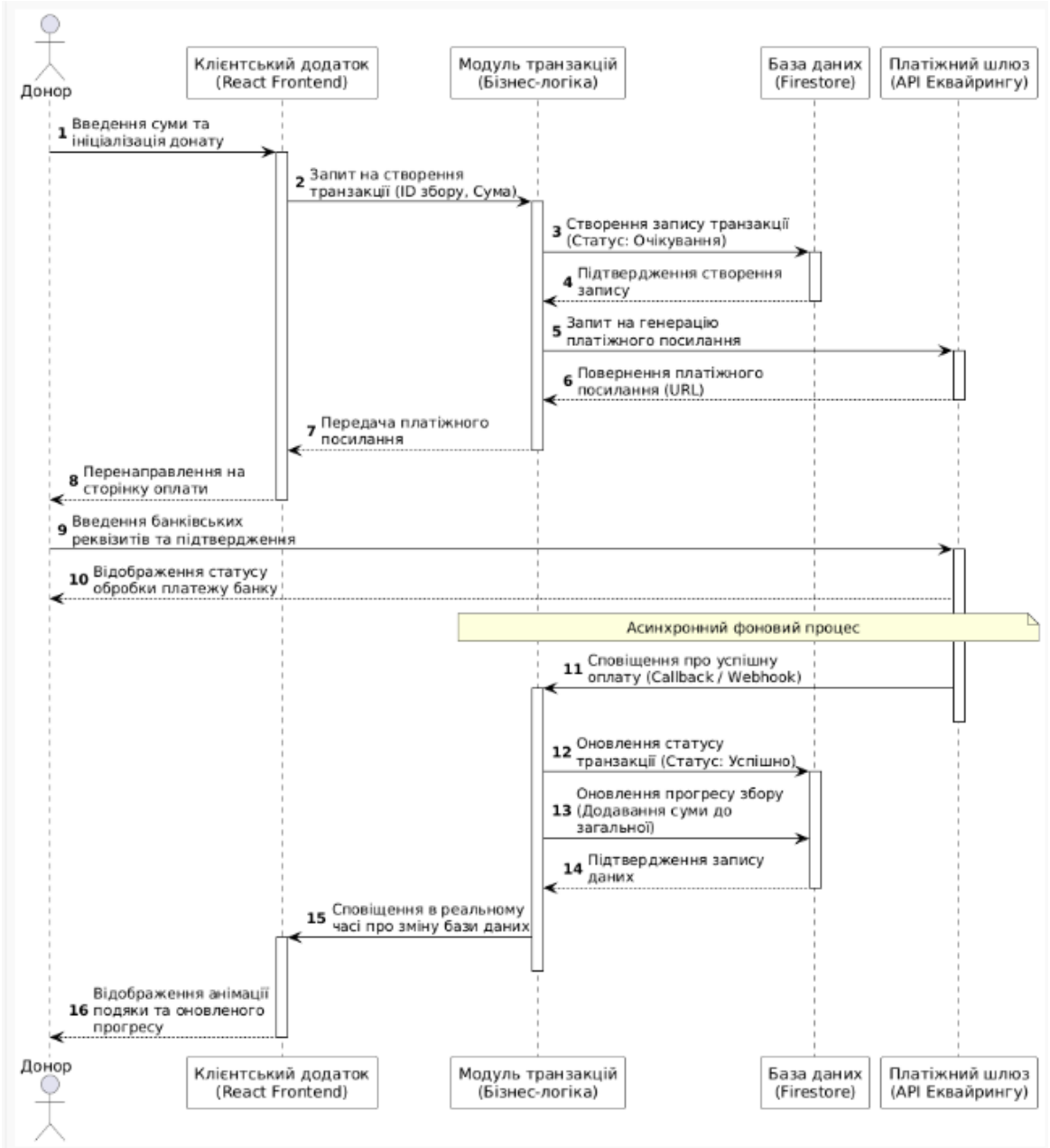


Рисунок 2.6 – Діаграма послідовностей взаємодії колистувача з донатом

Підсумовуючи етап детального проєктування модулів, варто наголосити, що враховуючи високу складність закладеної бізнес-логіки зокрема, необхідність асинхронної обробки фінансових транзакцій, суворого контролю життєвого циклу волонтерських зборів та забезпечення безпеки сесій таке глибоке опрацювання внутрішньої структури є беззаперечною запорукою успіху всього програмного продукту.

Створені алгоритми взаємодії та розроблені моделі переходів станів формують вичерпну, чітку специфікацію для подальшої програмної реалізації. Цей рівень деталізації дозволяє виявити та усунути приховані логічні прогалини чи архітектурні конфлікти ще до написання перших рядків коду, що суттєво мінімізує ризики виникнення критичних помилок на етапі тестування.

2.7 Аналіз та вибір технологій і методів реалізації

Після завершення етапу концептуального проєктування інтерфейсів та розробки архітектурних схем, наступним критичним кроком є вибір технологічного стека — набору інструментів, мов програмування та фреймворків, за допомогою яких система буде реалізована фізично. Від правильності цього вибору безпосередньо залежить швидкість розроблення, стабільність роботи системи під навантаженнями та легкість її подальшої підтримки. Оскільки розроблювана система є веборієнтованою, процес вибору технологій поділяється на два основні етапи: визначення інструментарію для клієнтської частини та вибір платформи для серверної логіки і зберігання даних.

Сучасна веб-розробка відійшла від використання класичного, неструктурованого JavaScript на користь потужних фреймворків та бібліотек. Вони дозволяють створювати односторінкові додатки (SPA - Single Page Application), які працюють без повного перезавантаження сторінки, що забезпечує користувачам плавний досвід, схожий на роботу з повноцінними настільними програмами. Це особливо важливо для нашого проєкту, де швидкість навігації між каталогом зборів та платіжною формою впливає на кількість успішних донатів. Для реалізації клієнтської частини було розглянуто три найпопулярніші на сьогодні інструменти: Angular, Vue.js та React.

Angular - це повноцінний, важкий фреймворк від компанії Google. Він надає розробнику жорстку структуру та містить з коробки всі необхідні інструменти (маршрутизацію, управління станом, валідацію форм). Його

головна перевага полягає у високій надійності для масштабних корпоративних (Enterprise) рішень. Проте для стартапів та проєктів середнього розміру він має занадто високий поріг входження та генерує багато надлишкового коду.

Vue.js - це прогресивний фреймворк, який здобув популярність завдяки простоті вивчення та дуже елегантній архітектурі. Він чудово підходить для швидкого створення невеликих додатків, проте при зростанні проєкту та ускладненні бізнес-логіки управління його компонентами може стати менш очевидним.

React - це бібліотека від компанії Meta (Facebook) для створення користувацьких інтерфейсів на основі компонентного підходу. React не нав'язує жорсткої структури, дозволяючи розробнику самостійно обирати додаткові інструменти (наприклад, для маршрутизації. Завдяки їй браузер не перемальовує всю сторінку при кожній дії, а оновлює лише ті компоненти, які дійсно змінилися (наприклад, лише індикатор суми збору під час донату).

Таблиця 2.2 - Порівняльний аналіз Frontend-інструментів

Критерій оцінки	Angular	Vue.js	React
Архітектура	Повний фреймворк (MVC)	Прогресивний фреймворк	Компонентна бібліотека
Швидкість рендерингу	Середня	Висока	Дуже висока (Virtual DOM
Поріг входження	Високий (вимагає TypeScript)	Низький	Середній
Масштабованість	Відмінна (для корпоративних систем)	Добра	Відмінна (гнучка екосистема)

Проаналізувавши вимоги до вебдодатку, можна зробити висновок, що Angular є занадто перевантаженим для реалізації платформи військових зборів, а Vue.js може не забезпечити необхідної гнучкості при ускладненні системи гейміфікації. Таким чином, React стає найбільш раціональним вибором. Його компонентний підхід ідеально відповідає спроєктованим макетам (де кожна картка збору чи форма є окремим компонентом), а використання Virtual DOM гарантує миттєве оновлення фінансових графіків без затримок.

Серверна частина відповідає за безпечне збереження фінансової історії, ідентифікацію користувачів та обробку транзакцій. При виборі серверних технологій розглядалися два кардинально різні підходи: написання власного сервера з нуля (на базі Node.js або Python) та використання готових хмарних платформ Backend-as-a-Service (BaaS).

Написання власного сервера (наприклад, використовуючи стек Node.js + Express + PostgreSQL) дає абсолютний контроль над кожним байтом інформації та дозволяє реалізувати бізнес-логіку будь-якої складності. Однак цей підхід вимагає значних витрат часу на налаштування інфраструктури: розробнику потрібно власноруч проєктувати реляційну базу даних, налаштовувати захист від мережевих атак, реалізовувати протоколи авторизації (JWT) та писати складну логіку для системи сповіщень у реальному часі (через WebSockets). Для проєкту, який має обмежені часові рамки розробки, це створює невиправдані ризики.

Альтернативою є платформи BaaS, які надають базу даних, систему авторизації та хмарне сховище як готові сервіси. Найвідомішими представниками цього напрямку є Supabase та Firebase. Supabase позиціонується як відкрита альтернатива, яка працює поверх класичної реляційної бази PostgreSQL. Вона має сувору структуру даних, але вимагає глибшого розуміння мови SQL. Firebase — це потужна хмарна екосистема від Google, що базується на нереляційній (NoSQL) базі даних Firestore. Вона відрізняється неймовірною швидкістю інтеграції. Її головна перевага — вбудована реактивність: будь-яка зміна в базі даних автоматично (без написання додаткового коду) надсилає

сигнал на клієнтський додаток. Це ідеально підходить для нашої системи індикаторів зібраних коштів. Крім того, Firebase має готові, надзвичайно надійні модулі безпечної авторизації через Google, електронну пошту та телефони.

Таблиця 2.3 – Порівняльний аналіз Backend-рішень

Критерій оцінки	Власний сервер (Node.js)	Supabase	Firebase
Швидкість старту розробки	Дуже низька (багато налаштувань)	Середня	Дуже висока
Тип бази даних	Реляційна (SQL)	Реляційна (PostgreSQL)	Документо-орієнтована (NoSQL)
Підтримка реального часу	Вимагає ручного написання (Socket.io)	Підтримується (складно налаштувати)	Вбудована базова функція
Система авторизації	Вимагає розробки з нуля	Надається з коробки	Надається з коробки (інтеграція з Google)
Хостинг та масштабування	Ручне адміністрування	Автоматичне	Автоматичне (хмарна інфраструктура)

З огляду на специфіку волонтерської платформи, де швидкість оновлення інформації про донати в реальному часі є критично важливою, а також зважаючи на високі вимоги до безпеки персональних даних користувачів, використання власного сервера визнано нераціональним.

Тому перевага віддається екосистемі Firebase. Її нереляційна база даних Firestore ідеально підходить для зберігання гнучких структур (карток зборів з різним набором полів), а готові модулі авторизації дозволять заощадити значний час на етапі програмування.

Підсумовуючи цей етап аналізу, можна констатувати, що зв'язка бібліотеки React для фронтенду та платформи Firebase для бекенду формує потужний, сучасний та гнучкий технологічний стек. Проте вибір базових фреймворків не є остаточним кроком у формуванні інструментарію.

Вибір базових технологій у вигляді React та Firebase формує лише загальний фундамент системи. Для забезпечення повноцінної роботи сучасного односторінкового додатка, гнучкої взаємодії з користувачем та безпечної обробки фінансових даних, цей стек необхідно доповнити низкою спеціалізованих бібліотек. Екосистема React відрізняється високою модульністю, тому з нею ідеально поєднуються наступні інструменти, які планується інтегрувати в проєкт:

Бібліотека маршрутизації React Router. Оскільки система проєктується як односторінковий додаток, класичний перехід між сторінками через запити до сервера є неефективним. Ця бібліотека бере на себе керування навігацією виключно на стороні браузера. Вона дозволить миттєво перемикатися між каталогом зборів, особистим кабінетом та формами оплати без перезавантаження вікна, а також забезпечить захист приватних маршрутів (наприклад, блокування доступу до панелі адміністратора для звичайних гостей).

Інструмент управління станом Redux Toolkit. Для складних систем, де різні компоненти повинні знати інформацію один про одного (наприклад, навігаційне меню має знати, чи авторизований користувач, а сторінка каталогу які фільтри зараз обрано), необхідне єдине централізоване сховище даних. Цей інструмент дозволяє надійно зберігати глобальний стан платформи, ефективно розподіляти дані між компонентами та запобігати розсинхронізації інформації під час виконання фінансових транзакцій.

Фреймворк стилізації Tailwind CSS. Відповідно до затвердженого раніше принципу мобільного пріоритету, процес перенесення візуальних прототипів у програмний код вимагає гнучкого інструменту стилізації. Використання цього утилітарного CSS-фреймворку дозволяє швидко створювати адаптивні інтерфейси без написання громіздких окремих файлів стилів. Він ідеально поєднується з компонентним підходом React і гарантує високу швидкість завантаження сторінок завдяки автоматичному видаленню невикористаного коду.

Бібліотеки обробки та валідації форм React Hook Form та Yup. Оформлення благодійного внеску чи створення нової картки збору волонтером передбачає роботу з формами введення. Для забезпечення безпеки система повинна миттєво перевіряти коректність введених даних (правильність формату суми, наявність обов'язкових реквізитів). Обрана зв'язка бібліотек дозволяє проводити складну валідацію введеної інформації безпосередньо під час набору тексту, мінімізуючи навантаження на пристрій користувача та запобігаючи відправці некоректних запитів до бази даних бази даних.

Для забезпечення ефективного процесу написання програмного коду, зручної навігації файловою системою проєкту та швидкого пошуку помилок необхідно обрати оптимальне інтегроване середовище розробки. Оскільки розробка сучасного клієнт-серверного вебдодатку вимагає одночасної роботи з розміткою, стилями та складною логікою на базі обраних фреймворків, інструмент має бути потужним, але не перевантажувати операційну систему комп'ютера.

На етапі аналізу розглянуто три популярні програмні рішення для вебпрограмування:

WebStorm - професійне комерційне середовище від компанії JetBrains. Цей інструмент пропонує максимальний набір вбудованих функцій «з коробки», включаючи глибокий аналіз коду та власні інструменти для роботи з базами даних. Його головними недоліками є високе споживання системних ресурсів

(оперативної пам'яті) та необхідність придбання платної ліцензії для комерційного чи тривалого використання (рисунок 2.7).

Рисунок 2.7 – Логотип середовища розробки WebStorm

Sublime Text надзвичайно легкий та швидкий текстовий редактор. Він миттєво запускається та ідеально підходить для редагування окремих файлів. Проте він не є повноцінним інтегрованим середовищем: для організації комфортної роботи зі складною архітектурою обраного компонентного фреймворку цей редактор вимагає тривалого ручного пошуку, встановлення та конфігурації десятків сторонніх доповнень (рисунок 2.8).

Рисунок 2.8 – Логотип текстового редактора Sublime Text

Visual Studio Code безкоштовний редактор коду з відкритим вихідним кодом від компанії Microsoft, який на сьогодні є індустріальним стандартом у сфері веброзробки. Він поєднує в собі швидкість легких редакторів та потужність повноцінних середовищ завдяки величезній екосистемі розширень. Інструмент має ідеальну вбудовану підтримку обраних нами технологій, зручний термінал та засоби для роботи з контролем версій (рисунок 2.9).

Рисунок 2.9 – Логотип середовища розробки Visual Studio Code

Проаналізувавши наведені варіанти, як основний інструмент для подальшої фізичної реалізації платформи було обрано Visual Studio Code. Його гнучкість дозволить встановити лише ті доповнення, які дійсно необхідні для розробки (наприклад, форматери коду та інструменти для підсвічування синтаксису компонентів). Наявність потужного вбудованого термінала дозволить зручно запускати локальний сервер та взаємодіяти з хмарною інфраструктурою бази даних в одному робочому вікні. Такий вибір забезпечить максимальний комфорт програмування, знизить ризик синтаксичних помилок та дозволить повністю зосередитися на реалізації складної бізнес-логіки системи.

Підсумовуючи етап вибору технологій та методів реалізації, варто зазначити, що сформований технологічний стек повністю відповідає заявленим вимогам до розроблюваної платформи. Використання компонентної бібліотеки React гарантує високу швидкодію клієнтської частини та миттєве оновлення

динамічних елементів інтерфейсу. Інтеграція з хмарною екосистемою Firebase знімає необхідність розробки власної серверної інфраструктури з нуля, забезпечуючи при цьому надійний захист фінансових даних та вбудовану підтримку оновлень у реальному часі. Використання перевірених бібліотек стилізації та управління станом у середовищі Visual Studio Code дозволить суттєво оптимізувати процес написання коду. Таким чином, обраний інструментарій мінімізує технічні ризики та створює надійний фундамент для фізичної реалізації проєкту.

Загалом було виконано повний цикл проєктування вебдодатку для автоматизації та обліку військових зборів, у результаті якого концептуальні вимоги було перетворено на чітку інженерну специфікацію. За підсумками проведеного дослідження та проєктування було досягнуто наступних результатів:

На основі порівняльного аналізу архітектурних підходів обґрунтовано вибір монолітної клієнт-серверної архітектури з використанням хмарних сервісів. Це забезпечить високу швидкість обробки даних та достатній рівень безпеки для роботи з фінансовими транзакціями.

− проведено глибоку декомпозицію системи, у результаті якої вебдодаток було розділено на незалежні логічні модулі: ідентифікації, управління кампаніями, обробки транзакцій та гейміфікації. Для кожного модуля детально описано міжмодульні та внутрішні залежності;

− спроєктовано інтерфейс користувача з урахуванням принципу мобільного пріоритету.

Розроблено візуальні прототипи ключових сторінок платформи: головної панелі навігації, каталогу зборів та оптимізованої форми здійснення благодійного внеску;

− виконано детальне проєктування бізнес-логіки головних об'єктів системи. За допомогою діаграм станів формалізовано життєвий цикл благодійного збору та сесії користувача. Побудована діаграма послідовностей

повністю описала алгоритм асинхронної обробки платежів та взаємодію з базою даних;

− затверджено кінцевий стек технологій для програмної реалізації. Обрано бібліотеку React для побудови фронтенду, платформу Firebase для збереження даних та авторизації, а також редактор Visual Studio Code як головне середовище розробки.

Таким чином, розроблена архітектура, спроєктовані візуальні макети та алгоритми взаємодії утворюють вичерпну технічну базу.

3.ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Реалізація логіки зборів та донатів

Головним функціональним ядром розроблюваного вебдодатку є модуль обробки фінансових надходжень. Оскільки платформа орієнтована на максимальне охоплення аудиторії донорів, програмна логіка була спроєктована таким чином, щоб підтримувати три незалежні канали фінансування: міжнародний картковий еквайринг, інтеграцію з накопичувальними рахунками українських банків та прямі банківські перекази за реквізитами. Така мультиканальність вимагала створення складної архітектури синхронізації даних для коректного відображення загального прогресу збору.

Для автоматизованого приймання платежів з банківських карток (Visa, Mastercard) безпосередньо в інтерфейсі вебдодатку було обрано міжнародну систему Stripe. Головною особливістю цієї системи є відповідність стандарту безпеки PCI DSS: розроблюваний сервер ніколи не торкається відкритих номерів карток, працюючи виключно із зашифрованими токенами.

Програмна інтеграція Stripe розділена на клієнтську та серверну частини і працює за наступним алгоритмом:

Ініціалізація налаштувань: У React-додатку використовується провайдер Elements з бібліотеки @stripe/react-stripe-js, який обгортає форму оплати та підключає публічний ключ Stripe API.

Створення наміру платежу: Коли користувач вводить суму та натискає «Пожертвувати», клієнтський додаток не відправляє платіж одразу. Замість цього він робить запит до Firebase Cloud Functions. Серверна функція за допомогою секретного ключа звертається до Stripe API та створює об'єкт PaymentIntent. Цей об'єкт резервує суму та повертає на клієнт спеціальний токен client_secret.

Клієнтська обробка: Отримавши client_secret, компонент CardElement безпечно відмальовує поля для введення реквізитів у фреймі, ізольованому від

нашого домену. Після введення даних бібліотека Stripe самостійно виконує транзакцію.

Асинхронне підтвердження: Найважливішим етапом є підтвердження. Оскільки користувач може закрити вкладку до завершення транзакції, клієнтський додаток не має права самостійно оновлювати базу даних. Для цього налаштовано Webhook захищену кінцеву точку у Firebase Cloud Functions. Коли банк списує кошти, Stripe відправляє на цей Webhook POST-запит із подією payment_intent.succeeded. Сервер перевіряє криптографічний підпис запиту, після чого атомарно оновлює документ збору у базі Firestore, додаючи суму до поля currentAmount.

Враховуючи специфіку українського простору благодійності, обов'язковою вимогою стала підтримка цільових рахунків Monobank. Програмна реалізація цього методу кардинально відрізняється від Stripe, оскільки процес оплати відбувається поза межами сайту.

Логіка роботи з «Банкою» побудована на комбінації генерації графічних кодів та роботи з відкритим API банку:

Генерація QR-коду: Під час створення збору волонтер зберігає у базі даних URL-посилання на свою «Банку».

На стороні клієнта за допомогою бібліотеки qrcode.react це посилання «на льоту» перетворюється на інтерактивний QR-код. Користувачі десктопних версій можуть просканувати його смартфоном, а для мобільних користувачів генерується кнопка «Відкрити в додатку», яка використовує технологію Deep Link для прямого запуску програми Monobank.

Синхронізація балансу: Для того щоб сума на сторонній «Банці» враховувалася у загальному прогресі на нашому сайті, реалізовано механізм опитування відкритого API.

Адміністратор збору надає свій персональний токен Monobank.

Створена у Firebase запланована функція періодично робить GET-запит до api.monobank.ua/personal/client-info. З отриманого масиву рахунків алгоритм

витягує об'єкт цільової «Банки», зчитує поле балансу та синхронізує його з даними в нашій колекції campaigns.

Третім методом є прямий переказ за реквізитами (IBAN або номер картки). Цей метод не піддається повній автоматизації з боку системи, тому вимагає впровадження алгоритму ручної модерації:

Буферна транзакція: Користувач копіює реквізити, здійснює переказ через свій банк і натискає у вебдодатку кнопку «Я здійснив переказ». Система відкриває форму, де користувач вводить точну суму, час переказу та (опціонально) прикріплює скріншот квитанції.

Статус очікування: Модуль створює у колекції transactions новий запис зі спеціальним статусом pending_manual_verification. Ця сума не додається до загального прогресу збору миттєво, щоб уникнути штучної накрутки показників зловмисниками.

Модерація адміністратором: Волонтер у своїй захищеній панелі бачить список транзакцій зі статусом очікування. Звіривши їх з випискою свого реального банківського рахунку, він натискає кнопку «Підтвердити». Лише після цього система змінює статус транзакції на success і запускає тригер оновлення лічильника currentAmount.

Оскільки кошти надходять із трьох різних джерел (автоматичні через Stripe, напівавтоматичні через Monobank API та ручні через IBAN), критично важливим уникнути конфліктів під час одночасного запису в базу.

Для вирішення цієї проблеми буде використано транзакційні операції Firestore.

Незалежно від джерела платежу, оновлення поля загальної суми завжди відбувається шляхом блокування документа збору на мілісекунди: сервер зчитує поточний стан, додає нову суму та записує оновлений результат.

Це гарантує математичну точність навіть у випадках, коли десятки донорів одночасно здійснюють оплату різними методами, забезпечуючи стовідсоткову надійність обліку благодійних внесків.

3.2 Розробка бази даних

Оскільки архітектура вебдодатку базується на принципах безсерверних обчислень, основним сховищем даних та платформою для бекенду було обрано екосистему Firebase від Google.

Процес її налаштування складався з ініціалізації проєкту, проєктування структури бази даних, конфігурації модуля авторизації та написання суворих правил безпеки.

Процес розробки розпочався зі створення нового проєкту в консолі Firebase. Після реєстрації вебдодатку в панелі керування було згенеровано унікальний об'єкт конфігурації який містить ключі доступу.

Для інтеграції цих налаштувань у React-додаток було створено окремий конфігураційний файл налаштування , що відповідає за збереження та розподіл існуючих ключів конфігурації між проектом, у директорії services.

Використовуючи офіційний Firebase SDK, у цьому файлі відбувається ініціалізація головного об'єкта додатка, а також експорт інстансів бази даних модуля авторизації та хмарного сховища.

Такий підхід гарантує створення єдиного підключення що запобігає витокам пам'яті під час навігації між сторінками (рисунок 3.1).

```
{
  "projectId": "Charity",
  "appId": "example-key-",
  "apiKey": "example-key-",
  "authDomain": "example-key-",
  "firestoreDatabaseId": "example-key-",
  "storageBucket": "example-key-",
  "messagingSenderId": "example-key-",
  "measurementId": "example-key-"
}
```

Рисунок 3.1 – Приклад налаштування ключів доступа Firebase

В якості основної бази даних було обрано сучасне хмарне рішення. Це документоорієнтована база, яка зберігає інформацію у вигляді окремих груп та записів. На відміну від класичних табличних баз, вона дозволяє зберігати гнучкі структури даних та забезпечує високу швидкість читання завдяки автоматичному впорядкуванню інформації.

Структуру даних було спроєктовано таким чином, щоб мінімізувати кількість звернень до сервера під час завантаження головних сторінок. Було створено три основні групи записів.

Перша група зберігає інформацію про користувачів. Записи тут створюються автоматично під час першої реєстрації.

Ідентифікатором кожного такого запису виступає унікальний код, згенерований модулем авторизації. Кожен запис містить ім'я, електронну пошту, посилання на фотографію та критично важливе поле ролі.

Саме роль визначає рівень доступу в системі, розділяючи користувачів на благодійників, волонтерів та адміністраторів.

Друга група є центральною сутністю системи і містить дані про благодійні збори.

Кожен запис репрезентує окремий збір і містить його назву, опис, цільову суму, поточну зібрану суму, посилання на творця збору, поточний стан збору від чорнетки до завершеного, а також зашифрований ключ для з'єднання з банківською системою.

Третя група слугує журналом фінансових операцій. Записи тут містять ідентифікатор збору, куди пішли кошти, суму, спосіб оплати, точний час переказу та статус виконання операції.

Для цих записів налаштовано складні покажчики, що дозволяє швидко будувати графіки надходжень у панелі адміністратора завдяки чому контролювати благодійні внески стало ще простіше.

Розглянемо вигляд одного запису в базі , на прикладі створеного збору в базі даних (рисунок. 3.2)

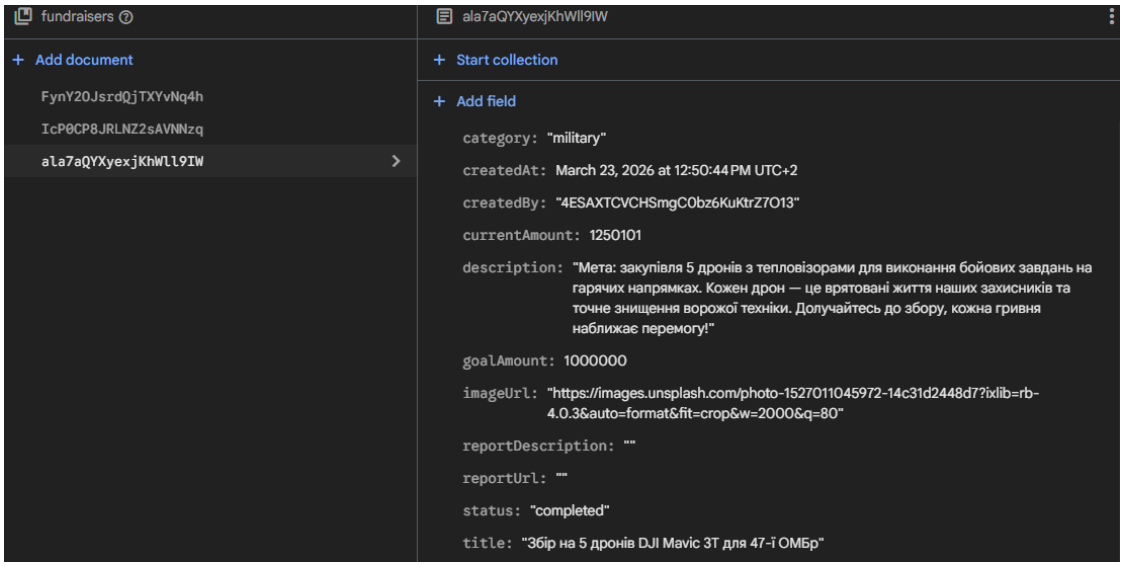


Рисунок 3.2 - Структура записів благодійних зборів у базі даних

Для управління сеансами та захисту персональних даних було налаштовано модуль розпізнавання користувачів. Було активовано два способи входу. Перший спосіб передбачає класичну реєстрацію. Для неї реалізовано логіку відправки автоматичних листів для підтвердження адреси електронної пошти, щоб уникнути створення фальшивих облікових записів. Другий спосіб налаштовано для забезпечення швидкого входу в один клік через облікові записи Гугл. Це суттєво підвищує залученість донорів, оскільки позбавляє їх необхідності створювати та запам'ятовувати нові паролі.

Оскільки обрана архітектура дозволяє клієнтській програмі звертатися до бази даних безпосередньо без проміжного сервера, критично важливим етапом стала розробка суворих правил безпеки. Без них будь-хто міг би змінити суми зборів через інструменти браузера.

Правила були написані спеціальною мовою та розгорнуті на хмарних серверах. Основна логіка безпеки реалізована за кількома принципами.

По-перше, налаштовано публічний доступ на читання. Будь-який відвідувач, навіть неавторизований гість, має право бачити записи про активні або завершені збори. По-друге, забезпечено захист запису для користувачів. Люди можуть редагувати лише власні профільні дані.

По-третє, впроваджено рольовий доступ до зборів. Створювати, редагувати або видаляти кампанії можуть лише ті користувачі, які пройшли авторизацію та мають статус волонтера або адміністратора. Це перевіряється спеціальною допоміжною функцією безпосередньо у правилах безпеки.

Останнім і найважливішим принципом є захист фінансів. Пряма зміна фінансових переказів або поточних сум у зборах з боку користувача суворо заборонена. Будь-які зміни балансу дозволено здійснювати виключно через захищені серверні алгоритми після успішної перевірки платежу з боку платіжної системи або банку (рисунок 3.3).

```
match /users/{userId} {
  allow read: if isOwner(userId) || isAdmin();

  // Allow user creation. Only admins can create admin users, unless it's the default admin email.
  allow create: if isAuthenticated() && isOwner(userId) && isValidUser() && uidUnchanged() && (
    request.resource.data.role == 'user' || isAdmin()
  );

  // Allow users to update their own profile (but not role or uid), and admins can update anything.
  allow update: if isAuthenticated() && isValidUser() && uidNotModified() && (
    isAdmin() ||
    (isOwner(userId) && request.resource.data.role == resource.data.role && request.resource.data.createdAt == resource.data.createdAt)
  );

  allow delete: if isAdmin();
}

match /fundraisers/{fundraiserId} {
  allow read: if true;
  allow create: if isAdmin() && isValidFundraiser();
  allow update: if (isAdmin() && isValidFundraiser()) || isValidDonationUpdate();
  allow delete: if isAdmin();

  match /updates/{updateId} {
    allow read: if true;
    allow write: if isAdmin();
  }
}

match /donations/{donationId} {
  allow read: if true;
  allow create: if isValidDonation();
  allow update: if false;
  allow delete: if isAdmin();
}
```

Рисунок 3.3 – Конфігурація правил безпеки для бази даних

Комплексне налаштування хмарного середовища дозволило створити високопродуктивну, масштабовану та безпечну серверну частину. Використання нереляційної бази даних забезпечило гнучкість структури зборів, а суворі правила безпеки повністю унеможливили несанкціоноване втручання у фінансову статистику платформи.

3.3 Розробка програмних модулів

Фізична реалізація вебдодатку базується на компонентному підході, який є фундаментальним для обраної бібліотеки створення користувацьких

інтерфейсів. Архітектурно додаток будується як сукупність незалежних, ізольованих модулів або візуальних блоків, які можуть багаторазово використовуватися у різних частинах програми. Кожен такий блок містить власну розмітку, стилі та логіку поведінки. Головною перевагою такої архітектури є односпрямований потік даних, що робить поведінку інтерфейсу передбачуваною та значно спрощує пошук помилок під час тестування.

Процес розробки розпочався з ініціалізації базового середовища та налаштування архітектури файлів.

Для забезпечення високої швидкості компіляції коду та оптимізації готового продукту було використано сучасний збирач модулів. Крім того, весь проєкт налаштовано для роботи зі строго типізованою мовою програмування, про що свідчать відповідні конфігураційні файли в корені проєкту. Це дозволяє виявляти логічні помилки ще на етапі написання коду.

У кореневій директорії проєкту знаходяться файли налаштувань середовища, файл управління залежностями програми, а також конфігурації для розгортання та безпеки хмарної бази даних, зокрема файл правил безпеки бази даних.

Там само розміщено скрипт для початкового наповнення бази даних тестовою інформацією, що є необхідним для етапу розробки.

Така структура проєкту була впроваджена для забезпечення максимальної модульності та масштабованості системи.

Завдяки відокремленню конфігураційних параметрів від логіки виконання, вдалося досягти високого рівня гнучкості: середовище розробки, тестування та продуктового розгортання (production) легко адаптується шляхом зміни значень у відповідних змінних оточення (environment variables), без необхідності внесення правок у вихідний код застосунку.

Використання файлів управління залежностями дозволяє чітко контролювати версійність бібліотек, що мінімізує ризики виникнення конфліктів сумісності під час оновлення компонентів системи (рисунок 3.5).

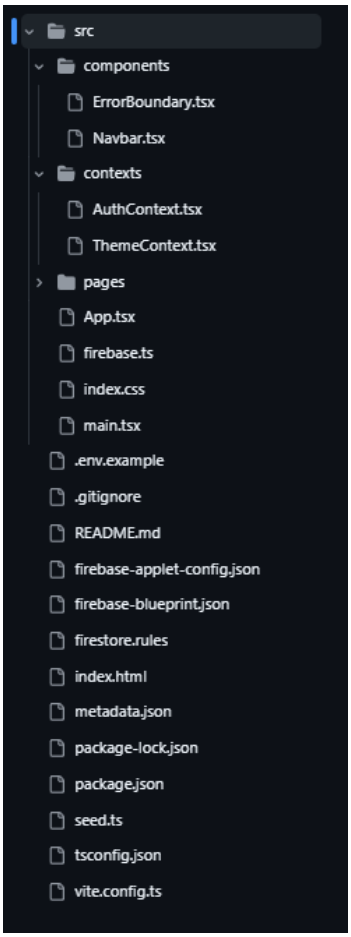


Рисунок 3.5 - Структура директорій та файлів вебдодатку

Основна логіка програми та візуальні елементи зосереджені у директорії вихідного коду. Ця директорія логічно поділена на кілька ключових підкаталогів, кожен з яких виконує свою специфічну роль у системі.

Підкаталог компонентів містить ізольовані візуальні елементи інтерфейсу, які не прив'язані до конкретного маршруту. Згідно з структурою, тут реалізовано головну навігаційну панель програми та спеціальний модуль обробки помилок. Останній виступає в ролі запобіжника: у разі критичного збою в одному з дочірніх елементів, він перехоплює помилку та не дозволяє всьому додатку припинити роботу, виводячи на екран резервний інтерфейс.

Підкаталог контекстів відповідає за управління глобальним станом програми. Оскільки передача даних через глибоке дерево елементів є неефективною, було реалізовано механізм контекстів. Тут розміщено модуль стану авторизації, який зберігає дані про поточного користувача та надає доступ

до них будь-якій частині системи, а також модуль управління темою оформлення, який відповідає за перемикання візуальних стилів (рисунок 3.6).

```tsx
import React, { createContext, useContext, useEffect, useState } from 'react';
import { User, onAuthStateChanged, signInWithPopup, GoogleAuthProvider, signOut, signInWithEmailAndPassword } from 'firebase/auth';
import { doc, getDoc, setDoc, serverTimestamp } from 'firebase/firestore';
import { auth, db } from '../firebase';

interface AuthContextType {
  user: User | null;
  role: 'admin' | 'user' | null;
  loading: boolean;
  signInWithGoogle: () => Promise<void>;
  signInWithEmail: (email: string, pass: string) => Promise<void>;
  logout: () => Promise<void>;
}

const AuthContext = createContext<AuthContextType | undefined>(undefined);

export function AuthProvider({ children }: { children: React.ReactNode }) {
  const [user, setUser] = useState<User | null>(null);
  const [role, setRole] = useState<'admin' | 'user' | null>(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const unsubscribe = onAuthStateChanged(auth, async (firebaseUser) => {
      if (firebaseUser) {
        setUser(firebaseUser);

        // Check if user exists in Firestore
        const userDocRef = doc(db, 'users', firebaseUser.uid);
        const userDoc = await getDoc(userDocRef);

        if (userDoc.exists()) {
          let currentRole = userDoc.data().role;
          const isDefaultAdmin = firebaseUser.email?.toLowerCase() === 'grebeniuk1180406@gmail.com';

          // Upgrade to admin if it's the default admin email but currently set to user
          if (isDefaultAdmin && currentRole !== 'admin') {
            try {
              await setDoc(userDocRef, { role: 'admin' }, { merge: true });
            } catch (e) {
              console.error("Could not update role in DB", e);
            }
            currentRole = 'admin';
          }
          setRole(currentRole);
        } else {
          // Create new user document
          const isDefaultAdmin = firebaseUser.email?.toLowerCase() === 'grebeniuk1180406@gmail.com';
          const newRole = isDefaultAdmin ? 'admin' : 'user';
```

Рисунок 3.6 – Модуль стану авторизації користувача

Підкаталог сторінок включає компоненти вищого рівня. Вони виконують роль контейнерів, які об'єднують дрібні візуальні елементи та прив'язуються до конкретних адрес маршрутизатора.

Це повноцінні екрани, які бачить користувач під час навігації системою.

У добре спроектованій системі сторінка має бути максимально «тонкою» Якщо логіка сторінки стає занадто об'ємною, її слід виносити у спеціальні хуки або сервісні модулі, залишаючи саму сторінку лише як структуру для розміщення компонентів (рисунок 3.7).

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

type Theme = 'light' | 'dark';

interface ThemeContextType {
  theme: Theme;
  toggleTheme: () => void;
}

const ThemeContext = createContext<ThemeContextType | undefined>(undefined);

export function ThemeProvider({ children }: { children: React.ReactNode }) {
  const [theme, setTheme] = useState<Theme>(() => {
    const saved = localStorage.getItem('theme');
    return (saved as Theme) || 'light';
  });

  useEffect(() => {
    localStorage.setItem('theme', theme);
    if (theme === 'dark') {
      document.documentElement.classList.add('dark');
    } else {
      document.documentElement.classList.remove('dark');
    }
  }, [theme]);

  const toggleTheme = () => {
    setTheme(prev => prev === 'light' ? 'dark' : 'light');
  };

  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
}

export function useTheme() {
  const context = useContext(ThemeContext);
  if (context === undefined) {
    throw new Error('useTheme must be used within a ThemeProvider');
  }
  return context;
}
```

Рисунок 3.7 – Модуль стану теми

Окремої уваги заслуговують файли ініціалізації в корені папки вихідного коду. Файл конфігурації бази даних містить налаштування підключення до хмарної інфраструктури. Головний файл програми визначає загальну розмітку та підключає глобальні постачальники даних, тоді як точка входу відповідає за безпосереднє монтування створеного віртуального дерева елементів у реальне середовище вебпереглядача. Також тут розміщено файл глобальних стилів, який містить базові візуальні налаштування інтерфейсу.

Такий суворий архітектурний поділ повністю відповідає принципу єдиної відповідальності. Кожен файл та директорія виконують виключно свою задачу, що значно спрощує навігацію по коду та робить продукт готовим до подальшого розширення функціоналу.

3.4 Розробка інтерфейсів користувача

Розробка візуальної частини вебдодатку базувалася на принципі мобільного пріоритету, що було затверджено ще на етапі початкового проєктування. Оскільки сучасні користувачі здебільшого здійснюють благодійні внески зі смартфонів, інтерфейс першочергово оптимізувався під екрани мобільних пристроїв, після чого його архітектура масштабувалася для настільних комп'ютерів.

Для перетворення чорно-білих візуальних прототипів у робочий програмний код використовувався утилітарний фреймворк стилізації Tailwind. Цей інструмент дозволив повністю відмовитися від написання окремих об'ємних файлів каскадних таблиць стилів. Замість цього стилізація відбувалася шляхом додавання готових класів безпосередньо в розмітку компонентів. Такий підхід значно пришвидшив процес розробки, зменшив загальний розмір кінцевого коду та забезпечив високу однорідність візуального оформлення всієї платформи.

Кожен графічний елемент системи, починаючи від кнопок і закінчуючи складними картками благодійних зборів, розроблявся як незалежний візуальний модуль. Адаптивність інтерфейсу досягалася завдяки вбудованим механізмам медіазапитів обраного фреймворку. На екранах мобільних телефонів елементи каталогу автоматично шикуються в одну вертикальну колонку для зручності гортання однією рукою, тоді як на широких моніторах вони розгортаються у повноцінну багатоколонкову сітку (рисунок 3.8).

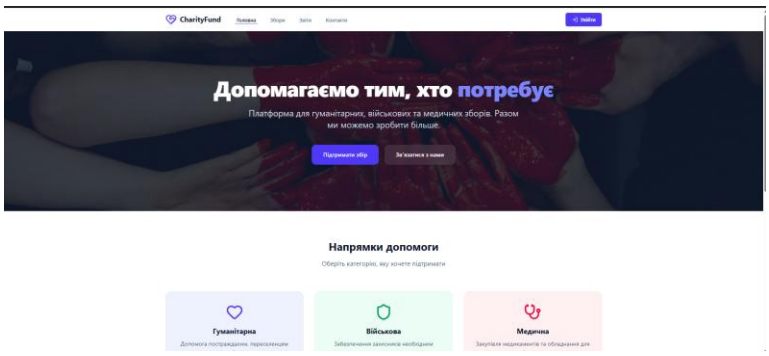


Рисунок 3.8 – Візуальна реалізація головної сторінки

Особлива увага під час розробки інтерфейсу приділялася динамічним елементам та станам взаємодії з користувачем. Візуальна частина була глибоко інтегрована з логікою отримання даних із хмарної бази. Для забезпечення плавного користувацького досвіду були реалізовані проміжні стани завантаження. Поки система отримує актуальну інформацію про збори, користувач бачить анімовані контури майбутніх карток, що запобігає відчуттю зависання програми та знижує когнітивне навантаження.

Динамічні елементи, такі як індикатор зібраних коштів, математично обчислюють свою ширину на основі отриманих даних і плавно змінюють розмір залежно від відсотка виконання фінансової мети. Крім того, усі кольорові рішення, відступи та шрифти були винесені у глобальні конфігураційні змінні проєкту. Це дозволило централізовано керувати темою оформлення та гарантувати, що кольори кнопок, текстів та фону залишатимуться ідентичними на всіх сторінках вебдодатку (рисунок 3.9).

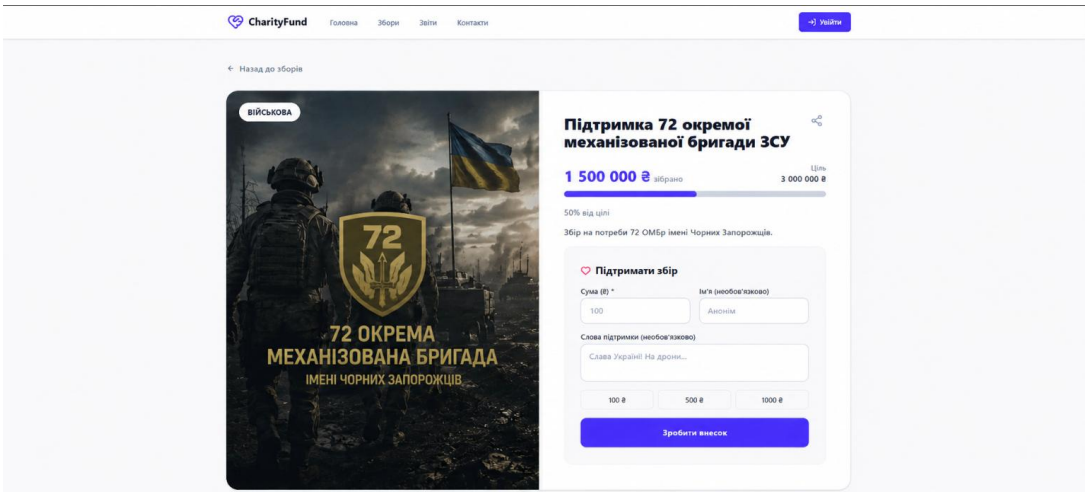


Рисунок 3.9 – Реалізація інтерфейсу сторінки окремого збору та форми здійснення внеску.

Данний збір не містить можливості оплати через моно, або картою напряму,але якщо увімкнути таку функцію в налаштуваннях адмін панелі, тоді збір буде виглядати так по іншому (рисунок 3.10).

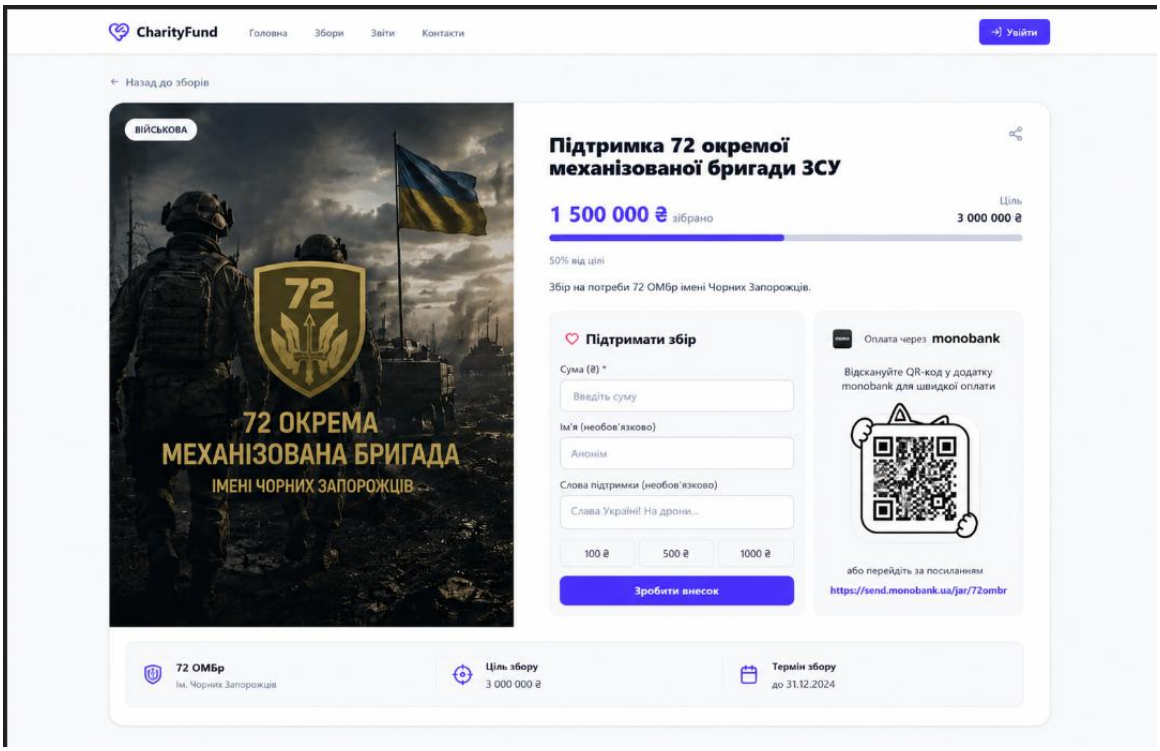


Рисунок 3.10 – Реалізація інтерфейсу сторінки окремого збору та форми здійснення внеску з можливістю донату на моно банку.

Логічним продовженням розробки публічного інтерфейсу стало створення закритої частини системи, призначеної для координації благодійних ініціатив. Оскільки платформа передбачає постійну взаємодію з фінансовими потоками, аналітикою та звітністю, для волонтерів та адміністраторів було розроблено багатофункціональну панель керування. Цей модуль дозволяє централізовано управляти всіма процесами без необхідності прямого втручання в базу даних, що значно підвищує безпеку та зручність обслуговування системи.

Візуальна архітектура панелі керування побудована за класичним принципом інформаційних панелей. Ліва частина екрана відведена під статичне навігаційне меню, яке забезпечує швидкий перехід між розділами аналітики, управління зборами, налаштуваннями та базою благодійників. Основна робоча область починається з блоку ключових показників ефективності. Тут у режимі реального часу відображаються зведені дані: загальна сума залучених коштів за певний період, кількість поточних активних кампаній, загальна кількість

унікальних донорів та відсоток успішних конверсій. Таке розташування дозволяє адміністратору миттєво оцінити загальний стан платформи одразу після входу в систему (рисунок 3.11).

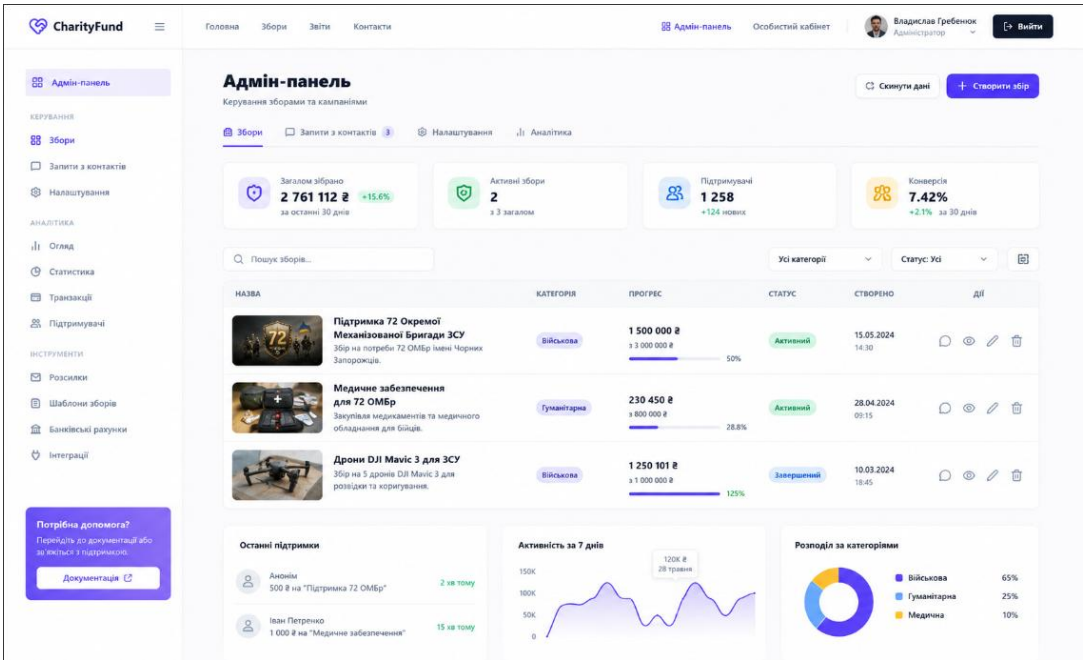


Рисунок 3.11 – Інтерфейс панелі керування для адміністраторів та волонтерів

Центральним елементом робочої області є інтерактивна таблиця поточних благодійних кампаній. Вона надає адміністратору вичерпну інформацію про кожен збір, включаючи його назву, категорію, статус активності та точну дату створення. Для швидкого розуміння фінансової ситуації кожна кампанія супроводжується візуальною шкалою прогресу, яка показує відсоткове співвідношення зібраних коштів до кінцевої мети. Навпроти кожного запису розташовані інструменти швидкої дії, що дозволяють відповідальній особі в один клік відредагувати опис, переглянути деталі або змінити статус кампанії.

Для глибокого аналізу ефективності платформи в нижній частині панелі реалізовано блок візуалізації статистичних даних. За допомогою інтегрованих інструментів побудови графіків система автоматично формує криву активності за останні сім днів, що дозволяє відстежувати дні найбільшої активності

благодійників. Поруч розташована кругова діаграма, яка наочно демонструє розподіл пожертв за різними напрямками, виділяючи частку військових, гуманітарних чи медичних потреб. Окремим блоком виведено стрічку останніх транзакцій, де фіксуються імена донорів, час переказу та сума. Усі ці аналітичні елементи динамічно пов'язані з хмарною базою даних і оновлюються миттєво при кожній новій успішній операції на платформі.

3.5 Вимоги до технічних та програмних засобів

Для успішної фізичної реалізації та повноцінного функціонування спроєктованої платформи необхідно чітко визначити мінімальні вимоги до апаратного та програмного забезпечення. Ці вимоги поділяються на три основні категорії: ресурси для робочої станції розробника, інфраструктурні потужності для серверної частини та характеристики пристроїв кінцевих користувачів.

Робоча станція інженера-програміста повинна мати достатню обчислювальну потужність для одночасного запуску середовища розробки, локального сервера компіляції та вебпереглядача з інструментами аналізу коду. Для забезпечення комфортної роботи рекомендується використовувати сучасний багатоядерний процесор з тактовою частотою від двох гігагерців та щонайменше вісім гігабайтів оперативної пам'яті. Для збереження файлів проєкту, модулів та забезпечення швидкого доступу до них доцільно використовувати твердотільний накопичувач із вільним простором не менше десяти гігабайтів.

Програмна складова робочого місця вимагає встановлення однієї із сучасних операційних систем, наприклад Windows, macOS або будь-якого популярного дистрибутиву Linux. Ключовими програмними інструментами, необхідними для компіляції проєкту, є середовище виконання Node.js та менеджер пакетів npm. Для безпосереднього написання програмного коду використовується редактор Visual Studio Code, а для надійного збереження історії змін та резервного копіювання встановлюється система контролю версій Git.

Оскільки архітектура розробленого додатка базується на хмарних обчисленнях, потреба в оренді та налаштуванні фізичного серверного обладнання повністю відсутня. Уся логіка бекенду, нереляційна база даних та модулі ідентифікації розгорнуті на потужностях платформи Firebase. Такий підхід гарантує автоматичне масштабування обчислювальних ресурсів залежно від кількості активних відвідувачів. Це забезпечує безперебійну роботу системи та швидке оброблення транзакцій навіть під час раптових пікових навантажень, які часто виникають після публікації термінових благодійних зборів у соціальних мережах.

Вимоги до пристроїв кінцевих користувачів, якими виступають благодійники та волонтери, зведені до абсолютного мінімуму. Для повноцінного доступу до системи достатньо мати будь-який пристрій з підключенням до мережі Інтернет, будь то настільний комп'ютер, планшет або мобільний телефон. Єдиною програмною вимогою є наявність оновленого вебпереглядача, такого як Google Chrome, Safari або Edge, з активованою підтримкою виконання сценаріїв JavaScript. Завдяки глибокій оптимізації коду та використанню адаптивного дизайну, платформа надзвичайно швидко завантажується та коректно працює навіть за умов слабкого або нестабільного мобільного зв'язку, що є критично важливим фактором для забезпечення безперервного процесу фінансування кампаній.

3.6 Керівництво користувача

Розроблений програмний продукт має інтуїтивно зрозумілий графічний інтерфейс, який не вимагає від користувачів додаткового навчання чи спеціальних технічних навичок. Взаємодія з платформою логічно розділена на два основні сценарії використання, які залежать від ролі відвідувача. Перший сценарій передбачений для звичайних гостей та благодійників, а другий розроблений для волонтерів та адміністраторів системи.

Для звичайних відвідувачів робота з вебдодатком розпочинається з головної сторінки. На ній представлено загальний каталог усіх поточних благодійних ініціатив. Відвідувач має змогу гортати стрічку кампаній, користуватися інструментами пошуку за назвою та застосовувати фільтри за категоріями потреб. Кожна кампанія представлена у вигляді компактної картки з тематичним зображенням, короткою назвою та індикатором зібраних коштів. Натиснувши на обрану картку, людина переходить на детальну сторінку збору, де розміщено розгорнутий опис мети, офіційні документи та повна статистика надходжень.

Процес здійснення фінансового внеску максимально спрощений. На сторінці обраного збору благодійнику пропонується кілька безпечних варіантів переказу. При виборі прямої оплати користувач вводить бажану суму та реквізити своєї банківської картки у захищене вікно, після чого система автоматично списує кошти. Альтернативний варіант дозволяє здійснити переказ через особистий банківський додаток. Для цього достатньо навести камеру смартфона на згенерований графічний код або скопіювати номер рахунку. Після успішного завершення операції індикатор на екрані плавно оновлюється, демонструючи новий загальний баланс кампанії.

Для волонтерів та адміністраторів передбачено розширений функціонал, доступний виключно після проходження процедури ідентифікації. Вхід у закриту частину системи здійснюється через відповідний розділ навігаційного меню. Авторизація відбувається за допомогою електронної пошти та пароля або шляхом швидкого входу через обліковий запис Гугл. Після підтвердження особи волонтер автоматично перенаправляється до закритої панелі керування.

У панелі керування адміністратор отримує повний контроль над своїми ініціативами. Для запуску нового збору необхідно перейти до відповідного розділу та заповнити форму створення кампанії. Процес вимагає вказання назви, детального опису потреб, встановлення цільової суми та завантаження візуальних матеріалів. Також на цьому етапі волонтер додає посилання на свої

накопичувальні рахунки для автоматичної генерації графічних кодів. Після збереження даних нова кампанія миттєво публікується в загальному каталозі і стає доступною для фінансування.

Окремим важливим інструментом панелі керування є сторінка аналітики та модерації транзакцій. Тут адміністратор відслідковує статистику надходжень за допомогою наочних графіків та діаграм. У разі надходження коштів прямим банківським переказом за реквізитами, які система не може відстежити автоматично, волонтер повинен самостійно перевірити виписку свого рахунку. Переконавшись у надходженні коштів, він підтверджує відповідну операцію в системі натисканням однієї кнопки, після чого підтверджена сума остаточно додається до загального прогресу збору.

Загалом логіка взаємодії з платформою побудована таким чином, щоб мінімізувати кількість кроків для виконання будь-якої цільової дії. Ергономічний дизайн та чітка структура дозволяють благодійникам швидко здійснювати перекази, а волонтерам ефективно керувати кампаніями та звітувати перед суспільством.

3.7 Тестування вебсайту

3.7.1 Аналіз методів тестування вебсайтів

Для забезпечення всебічної перевірки розробленого програмного забезпечення було обрано та проаналізовано кілька ключових методів тестування. Першим і фундаментальним підходом є функціональне тестування. Цей метод спрямований на перевірку працездатності кожного окремого модуля системи. У межах цього підходу детально перевіряються процеси ідентифікації користувачів, створення нових благодійних кампаній волонтерами, а також коректність роботи алгоритмів зміни статусів зборів. Головне завдання функціональної перевірки полягає у підтвердженні того, що кожна кнопка та

форма вводу виконує саме ту дію, яка була закладена на етапі початкового проєктування (рисунок 3.12).

ID	Назва тест-кейсу	Опис кроків	Очікуваний результат	Статус
TC-01	Реєстрація через Email	Введення пошти та пароля, натискання кнопки реєстрації.	Створення профілю в базі Firestore, успішна авторизація.	Успішно
TC-02	Вхід через Google	Вибір акаунту за допомогою Google OAuth провайдера.	Отримання даних та токена доступу, відображення профілю.	Успішно
TC-03	Створення збору (Волонтер)	Заповнення форми: назва, опис, мета, фото. Натискання "Опублікувати".	Кампанія з'являється у загальному каталозі активних зборів.	Успішно
TC-04	Платіж Stripe (Test Mode)	Введення суми, тестових реквізитів картки, підтвердження оплати.	Анімація успіху, миттєве збільшення балансу збору.	Успішно
TC-05	QR-код Monobank	Динамічна генерація коду на основі посилання на рахунок.	Відображення QR-коду, що веде на вірну сторінку оплати.	Успішно
TC-06	Доступ до	Спроба	Автоматично	Успішно

Рисунок 3.12 – Результати функціонального тестування.

Наступним важливим підходом є тестування інтеграції компонентів. Оскільки архітектура платформи передбачає тісну взаємодію клієнтської частини з хмарними серверними функціями та сторонніми фінансовими сервісами, необхідно переконатися у надійності каналів передачі інформації. Цей вид тестування сфокусований на перевірці коректності відповідей від міжнародного платіжного шлюзу, правильності зчитування даних із відкритих програмних інтерфейсів українських банків та безпомилковому атомарному оновленні записів у нереляційній базі даних після успішного проведення благодійної транзакції.

Окрім перевірки внутрішньої логіки, обов'язковим етапом є тестування користувацького інтерфейсу та загальної зручності використання. Цей метод

передбачає системну перевірку адаптивності візуальних елементів на екранах різних розмірів. Особлива увага приділяється коректному відображенню шрифтів, відступів та динамічних індикаторів прогресу на мобільних телефонах, планшетах та настільних моніторах. Також у межах цього методу аналізується зрозумілість навігації та рівень когнітивного навантаження на донора під час проходження шляху від вибору кампанії до успішного підтвердження платежу (рисунок 3.13).

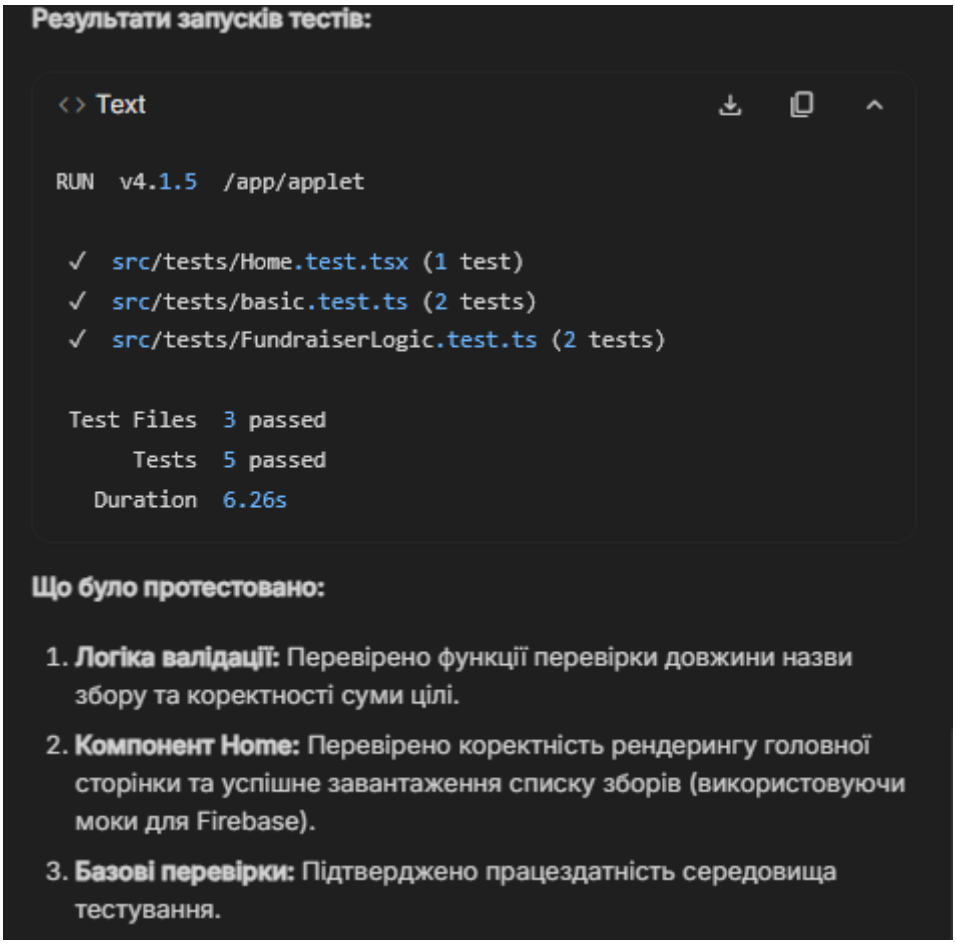


Рисунок 3.13 – Результати автоматизованого тестування.

Завершальним методом у комплексному підході є тестування продуктивності та навантаження. Цей вид перевірки дозволяє оцінити швидкість завантаження сторінок та загальний час реакції системи на дії відвідувача. Оскільки вебдодаток використовує реактивну архітектуру з постійним

спостереженням за змінами в реальному часі, критично важливо проаналізувати поведінку системи під час одночасного оновлення даних у тисяч клієнтів. Тестування продуктивності допомагає виявити вузькі місця у швидкодії програмного коду та оптимізувати розмір графічних ресурсів для забезпечення миттєвого відгуку інтерфейсу навіть за умов використання повільного мобільного зв'язку.

3.8 Висновки

Було виконано повний цикл програмної реалізації та комплексного тестування вебплатформи. Процес розробки підтвердив високу ефективність обраного технологічного стека на основі бібліотеки React та хмарної інфраструктури Firebase. Завдяки застосуванню компонентної архітектури вдалося створити гнучку та надійну систему, яка легко піддається подальшому масштабуванню. Впровадження сучасних інструментів розробки дозволило реалізувати складну логіку фінансових операцій, забезпечуючи при цьому високу швидкість відгуку інтерфейсу та стабільність роботи системи під навантаженням.

Особливу увагу в ході реалізації було приділено інтеграції різних методів здійснення благодійних внесків. Програмна розробка модулів для роботи з міжнародними платіжними шлюзами, банківськими сервісами через відкриті програмні інтерфейси та прямими переказами за реквізитами дозволила створити універсальний інструмент для залучення допомоги. Використання захищених серверних функцій для обробки транзакцій та суворих правил безпеки безпосередньо на рівні бази даних гарантувало цілісність фінансової інформації та надійний захист від несанкціонованого втручання.

Результати проведеного тристадійного тестування підтвердили повну відповідність готового програмного продукту встановленим інженерним вимогам. Перший етап перевірки продемонстрував безпомилкову роботу

основної логіки системи, включаючи процеси розпізнавання користувачів та автоматизованого обліку коштів. Другий етап тестування засвідчив високу якість графічного інтерфейсу та його повну адаптивність до різних типів мобільних та настільних пристроїв. Таким чином, розроблена платформа є стабільною, безпечною та повністю готовою до практичної експлуатації у сфері автоматизації волонтерської діяльності.

ВИСНОВКИ

Завершивши виконання кваліфікаційної роботи на тему «Розробка вебдодатку для автоматизації та обліку волонтерських зборів», можна зробити такі висновки. Виконання розпочалось з дослідження предметної області та постановки задач.

Спочатку було визначено рівень актуальності теми, проаналізовано проблематику процесу збору коштів (прозорість, довіра, швидкість) та досліджено існуючі платформи для благодійності.

Це показало високу актуальність теми, пов’язану зі зростанням кількості волонтерських ініціатив, надало можливість розглянути процеси, які типові для цієї галузі, та дозволило краще розуміти критично важливі завдання, які стоять перед розроблюваним вебдодатком. В результаті виконання аналізу предметної області було сформовано вимоги до застосунку, побудовано діаграму варіантів використання та поставлено задачі для проєктування і реалізації. Після визначення вимог і задач було проведено проєктування вебдодатку.

Першим важливим етапом був розгляд архітектурних підходів, які часто використовують під час реалізації сучасних клієнт-серверних рішень. На основі аналізу було обрано поєднання компонентного підходу для клієнтської частини та безсерверної (Serverless) архітектури для бекенду, що забезпечило модульність, безпеку і гнучкість системи.

На основі обраного архітектурного підходу було спроєктовано модулі для вебдодатку та побудовано низку діаграм:

− діаграма бази даних;

− діаграми послідовності;

− діаграми станів;

− діаграми діяльності;

− діаграма компонентів.

Після цього було визначено вимоги для інтерфейсу користувача та створено макети всіх потрібних екранів з урахуванням принципу мобільного

пріоритету. Наявність цих діаграм та макетів надала можливість краще розуміти композицію системи та ефективно почати реалізацію. В кінці проєктування було обрано технології та методи реалізації.

Для клієнтської частини було обрано бібліотеку React та утилітарний фреймворк Tailwind CSS, через їх високу гнучкість, швидкість рендерингу та підтримку компонентного підходу. В якості серверної інфраструктури та бази даних було вибрано екосистему Firebase по причині її надійності та реактивності. Як метод було обрано ітеративний, через високу потребу у швидкому впроваджені змін у проєкті та тестуванні платіжних шлюзів.

Після завершення проєктування перед початком реалізації було розглянуто особливості процесу побудови фінансових систем. Налаштування нереляційної бази даних Firestore та конфігурація правил безпеки (Security Rules) було першим етапом реалізації.

Після цього було створено візуальні React-компоненти інтерфейсу, які дали можливість гнучко взаємодіяти з даними. Використовуючи хмарні функції (Cloud Functions) було реалізовано інтеграцію з платіжною системою Stripe та генерацію QR-кодів для Monobank, що було об’єднано з розробленими програмними модулями адмін-панелі та публічного каталогу. Останнім етапом було проведення тестування системи. Було визначено, що основними методами тестування для фінансових вебдодатків є функціональне тестування, тестування інтерфейсу користувача (UI/UX) на адаптивність та тестування продуктивності через їх високу ефективність у виявленні критичних дефектів, по цій причині ці методи були обрані для тестування мого вебдодатку.

Результати тестування визначили високу стабільність та безпеку розробленого застосунку. Отже, можна стверджувати, що мета кваліфікаційної роботи, яка полягала у створенні вебдодатку для автоматизації та обліку волонтерських зборів, була досягнута.

Програмний продукт успішно розроблений, пройшов тестування, підтвердивши свою відповідність встановленим технічним вимогам. Вебдодаток

забезпечує зручний процес створення кампаній, прозорий облік пожертв, високий рівень безпеки транзакцій та адаптивний інтерфейс.

Таким чином, розроблений вебдодаток відповідає сучасним стандартам та вимогам індустрії веб-розробки. Результати кваліфікаційної роботи підтверджують її успішне завершення і відповідність поставленій меті та завданням.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Офіційна документація React. URL: https://react.dev/learn (дата звернення: 12.02.2026).

2. Firebase Documentation. Cloud Firestore та Authentication. URL: https://firebase.google.com/docs (дата звернення: 15.02.2026).

3. Stripe API Reference. Робота з платіжними намірами та вебхуками. URL: https://stripe.com/docs/api (дата звернення: 22.02.2026).

4. Tailwind CSS Documentation. Утилітарний підхід до стилізації вебдодатків. URL: https://tailwindcss.com/docs (дата звернення: 28.02.2026).

5 .Відкритий API Monobank. Отримання даних про баланс та виписки. URL: https://api.monobank.ua/docs/ (дата звернення: 05.03.2026).

6. Opendatabot. Аналітика благодійних зборів та донатів в Україні. URL: https://opendatabot.ua/analytics/charity-2023 (дата звернення: 10.03.2026).

7. Nielsen Norman Group. Mobile-First Design: Why It’s Great and Why It Sucks. URL: https://www.nngroup.com/articles/mobile-first/ (дата звернення: 15.03.2026).

8. Recharts. Побудова графіків та візуалізація даних у React. URL: https://recharts.org/en-US/guide (дата звернення: 20.03.2026).

9. React Router Documentation. Маршрутизація в односторінкових додатках. URL: https://reactrouter.com/en/main (дата звернення: 25.03.2026).

10. React Hook Form. Ефективна робота з формами та валідація даних у React. URL: https://react-hook-form.com/ (дата звернення: 28.03.2026).

11. Redux Toolkit. Сучасний підхід до управління станом додатку. URL: https://redux-toolkit.js.org/ (дата звернення: 02.04.2026).

12. Firebase Security Rules Documentation. Захист хмарних баз даних. URL: https://firebase.google.com/docs/rules (дата звернення: 05.04.2026).

13. Stripe Testing Guide. Використання тестового режиму для симуляції транзакцій. URL: https://stripe.com/docs/testing (дата звернення: 10.04.2026).

14. MDN Web Docs. Робота з WebSockets та асинхронними запитами. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 15.04.2026).

15. UX/UI Design Principles. Основи проєктування користувацьких інтерфейсів. URL: https://www.interaction-design.org/literature/topics/ui-design (дата звернення: 18.04.2026).

16. Jest Documentation. Інфраструктура для тестування JavaScript-застосунків. URL: https://jestjs.io/docs/getting-started (дата звернення: 22.04.2026).

17. OWASP Top Ten. Найбільш критичні ризики безпеки вебдодатків. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 25.04.2026).

18. Google Lighthouse. Автоматизований інструмент для перевірки якості вебсторінок. URL: https://developer.chrome.com/docs/lighthouse/overview/ (дата звернення: 28.04.2026).

20. GitHub. Платформа для хостингу IT-проєктів та спільної розробки. URL: https://github.com/ (дата звернення: 02.05.2026).

21. PlantUML. Створення UML-діаграм з текстового опису. URL: https://plantuml.com/ (дата звернення: 04.05.2026).

22. TypeScript Documentation. Строга типізація у JavaScript-додатках для підвищення надійності коду. URL: https://www.typescriptlang.org/docs/ (дата звернення: 05.05.2026).

23. Vite. Сучасний інструмент для швидкої збірки фронтенд-додатків. URL: https://vitejs.dev/guide/ (дата звернення: 06.05.2026).

24. Axios. Проміс-орієнтований HTTP-клієнт для браузера та Node.js. URL: https://axios-http.com/docs/intro (дата звернення: 06.05.2026).

25. React Testing Library. Інструментарій для тестування React-компонентів з акцентом на поведінку користувача. URL: https://testing-library.com/docs/react-testing-library/intro/ (дата звернення: 07.05.2026).

26. Cypress. Фреймворк для наскрізного (E2E) тестування вебдодатків. URL: https://docs.cypress.io/ (дата звернення: 08.05.2026).

27. Figma. Хмарна платформа для спільного проєктування інтерфейсів та прототипування. URL: https://help.figma.com/ (дата звернення: 09.05.2026).

28. TanStack Query (React Query). Асинхронне управління серверним станом та кешування даних у React. URL: https://tanstack.com/query/latest/docs/react/overview (дата звернення: 09.05.2026).

29. Framer Motion. Декларативна бібліотека для створення продуктивних анімацій у React. URL: https://www.framer.com/motion/ (дата звернення: 10.05.2026).

30. ESLint. Статичний аналізатор коду для виявлення та виправлення проблем у JavaScript/TypeScript. URL: https://eslint.org/docs/latest/ (дата звернення: 10.05.2026).

31. Web Content Accessibility Guidelines (WCAG) 2.1. Міжнародні рекомендації щодо забезпечення доступності вебконтенту. URL: https://www.w3.org/TR/WCAG21/ (дата звернення: 11.05.2026).

32. JSON Web Token (JWT). Відкритий стандарт безпечної передачі даних між сторонами. URL: https://jwt.io/introduction (дата звернення: 11.05.2026).

33. Vercel Documentation. Платформа для оптимізації та розгортання сучасних фронтенд-додатків. URL: https://vercel.com/docs (дата звернення: 12.05.2026).

34. Docker Documentation. Технології контейнеризації для ізоляції та розгортання програмного забезпечення. URL: https://docs.docker.com/ (дата звернення: 12.05.2026).

35. Postman. Платформа для розробки, документування та тестування API. URL: https://learning.postman.com/docs/introduction/overview/ (дата звернення: 13.05.2026).

36. Sentry. Моніторинг помилок та продуктивності вебдодатків у реальному часі. URL: https://docs.sentry.io/ (дата звернення: 13.05.2026).

37. Swagger (OpenAPI). Інструментарій для проєктування та інтерактивного документування RESTful API. URL: https://swagger.io/docs/ (дата звернення: 14.05.2026).

38. NPM Documentation. Управління залежностями та пакетами в екосистемі Node.js. URL: https://docs.npmjs.com/ (дата звернення: 14.05.2026).

39. Git Documentation. Розподілена система управління версіями для контролю змін у вихідному коді. URL: https://git-scm.com/doc (дата звернення: 15.05.2026).

40. Gamma App. Платформа на базі штучного інтелекту для генерації презентацій та візуалізації даних. URL: https://gamma.app/ (дата звернення: 15.05.2026).

41. Refactoring.Guru. Патерни проєктування та принципи рефакторингу коду в інженерії програмного забезпечення. URL: https://refactoring.guru/uk (дата звернення: 16.05.2026).

ДОДАТОК А

(обов'язковий)

ЛІСТИНГИ ПРОГРАМНОГО КОДУ

```
const functions = require('firebase-functions');
const admin = require('firebase-admin');
const stripe = require('stripe')(process.env.STRIPE_SECRET_KEY);

admin.initializeApp();
const db = admin.firestore();

const endpointSecret = process.env.STRIPE_WEBHOOK_SECRET;

exports.stripeWebhookHandler = functions.https.onRequest(async (req, res) => {
  const sig = req.headers['stripe-signature'];
  let event;

  try {
    event = stripe.webhooks.constructEvent(req.rawBody, sig, endpointSecret);
  } catch (err) {
    console.error('⚠️ Помилка верифікації вебхука:', err.message);
    return res.status(400).send(`Webhook Error: ${err.message}`);
  }

  if (event.type === 'payment_intent.succeeded') {
    const paymentIntent = event.data.object;
    const amountReceived = paymentIntent.amount_received / 100;
    const campaignId = paymentIntent.metadata.campaignId;
    const donorId = paymentIntent.metadata.userId || 'anonymous';

    try {
      const campaignRef = db.collection('campaigns').doc(campaignId);
      await campaignRef.update({
        currentAmount: admin.firestore.FieldValue.increment(amountReceived),
        updatedAt: admin.firestore.FieldValue.serverTimestamp()
      });

      await db.collection('transactions').add({
        campaignId: campaignId,
        amount: amountReceived,
        donorId: donorId,
        paymentMethod: 'stripe',
        paymentIntentId: paymentIntent.id,
        status: 'success',
        createdAt: admin.firestore.FieldValue.serverTimestamp()
      });

      console.log(`✅ Успішно зараховано ${amountReceived} до збору ${campaignId}`);
    } catch (dbError) {
      console.error('Помилка запису в базу даних:', dbError);
      return res.status(500).send('Database Update Failed');
    }
  }

  res.status(200).json({ received: true });
});
```

Рисунок А.1. - Хмарна функція для обробки вебхуків від Stripe

```
rules_version = '2';

service cloud.firestore {
  match /databases/{database}/documents {

    function isAuthenticated() {
      return request.auth != null;
    }

    function hasRole(role) {
      return isAuthenticated() &&
        get(/databases/$(database)/documents/users/$(request.auth.uid)).data.role == role;
    }

    match /users/{userId} {
      allow read: if isAuthenticated();
      allow write: if isAuthenticated() && request.auth.uid == userId;
    }

    match /campaigns/{campaignId} {
      allow read: if true;
      allow create, update, delete: if hasRole('volunteer') || hasRole('admin');
    }

    match /transactions/{transactionId} {
      allow read: if hasRole('admin') || hasRole('volunteer');
      allow write: if false;
    }
  }
}
```

Рисунок А.2. - Конфігурація правил безпеки бази даних

```
import { useState, useEffect } from 'react';
import { doc, onSnapshot } from 'firebase/firestore';
import { db } from '../services/firebaseConfig';

export const useFundraiser = (campaignId) => {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    if (!campaignId) {
      setLoading(false);
      return;
    }

    setLoading(true);

    const campaignRef = doc(db, 'campaigns', campaignId);

    const unsubscribe = onSnapshot(
      campaignRef,
      (docSnapshot) => {
        if (docSnapshot.exists()) {
          setData({ id: docSnapshot.id, ...docSnapshot.data() });
          setError(null);
        } else {
          setError('Благодійний збір не знайдено або його було видалено.');
          setData(null);
        }
        setLoading(false);
      },
      (err) => {
        console.error('Помилка отримання даних збору:', err);
        setError('Не вдалося завантажити інформацію. Перевірте з\'єднання.');
        setLoading(false);
      }
    );

    return () => {
      unsubscribe();
    };
  }, [campaignId]);

  return { data, loading, error };
};
```

Рисунок А.3. - Користувацький React-хук для реактивного оновлення даних

ДОДАТОК Б

(обов’язковий)

ПРОГРАМНИЙ КОД ОСНОВНИХ МОДУЛІВ

Б.1 Код Точки входу

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';
import { AuthProvider, useAuth } from './contexts/AuthContext';
import { ErrorBoundary } from './components/ErrorBoundary';
import Navbar from './components/Navbar';
import Home from './pages/Home';
import Fundraisers from './pages/Fundraisers';
import FundraiserDetail from './pages/FundraiserDetail';
import AdminDashboard from './pages/AdminDashboard';
import AdminLogin from './pages/AdminLogin';
import Contact from './pages/Contact';
import Profile from './pages/Profile';
import Reports from './pages/Reports';

function ProtectedAdminRoute({ children }: { children: React.ReactNode }) {
  const { user, role, loading } = useAuth();

  if (loading) return <div className="flex justify-center items-center h-screen">Завантаження...</div>;
  if (!user) return <AdminLogin />;
  if (role !== 'admin') return <Navigate to="/" replace />;

  return <>{children}</>;
}

export default function App() {
  return (
```

```
    <ErrorBoundary>
      <AuthProvider>
        <Router>
          <div className="min-h-screen bg-slate-50 flex flex-col font-sans">
            <Navbar />
            <main className="flex-grow">
              <Routes>
                <Route path="/" element={<Home />} />
                <Route path="/fundraisers" element={<Fundraisers />} />
                <Route path="/fundraisers/:id" element={<FundraiserDetail />} />
                <Route path="/reports" element={<Reports />} />
                <Route path="/contact" element={<Contact />} />
                <Route path="/profile" element={<Profile />} />
                <Route
                  path="/admin"
                  element={
                    <ProtectedAdminRoute>
                      <AdminDashboard />
                    </ProtectedAdminRoute>
                  }
                />
              </Routes>
            </main>
            <footer className="bg-slate-900 text-slate-400 py-8 text-center">
              <p>&copy; {new Date().getFullYear()} Charity Fundraisers. Всі права захищені.</p>
            </footer>
          </div>
        </Router>
      </AuthProvider>
    </ErrorBoundary>
  );
}
```

Б.2 Код форми логіна для Адміністрування

```
import React, { useState } from 'react';
import { useAuth } from '../contexts/AuthContext';
import { Lock } from 'lucide-react';
import { useNavigate } from 'react-router-dom';

export default function AdminLogin() {
  const { signInWithEmail } = useAuth();
  const navigate = useNavigate();
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const [loading, setLoading] = useState(false);

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    setError('');
    setLoading(true);
    try {
      await signInWithEmail(email, password);
      navigate('/admin');
    } catch (err: any) {
      console.error(err);
      setError('Помилка входу. Перевірте email та пароль, або переконайтеся, що вхід за
паролем увімкнено у Firebase Console.');
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="min-h-screen bg-slate-50 flex flex-col justify-center py-12 sm:px-6 lg:px-8">
      <div className="sm:mx-auto sm:w-full sm:max-w-md">
        <div className="flex justify-center">
          <div className="h-12 w-12 bg-indigo-100 rounded-full flex items-center justify-center">
            <Lock className="h-6 w-6 text-indigo-600" />
          </div>
        </div>
        <h2 className="mt-6 text-center text-3xl font-extrabold text-slate-900">
          Вхід для адміністратора
        </h2>
        <p className="mt-2 text-center text-sm text-slate-600">
          Увійдіть за допомогою Email та пароля
        </p>
      </div>

      <div className="mt-8 sm:mx-auto sm:w-full sm:max-w-md">
```

```
      <div className="bg-white py-8 px-4 shadow sm:rounded-lg sm:px-10 border border-slate-200">
        <form className="space-y-6" onSubmit={handleSubmit}>
          {error && (
            <div className="bg-red-50 border border-red-200 text-red-600 px-4 py-3 rounded-lg text-sm">
              {error}
            </div>
          )}
          <div>
            <label className="block text-sm font-medium text-slate-700">Email</label>
            <input type="email" required value={email} onChange={e => setEmail(e.target.value)} className="mt-1 block w-full border border-slate-300 rounded-md shadow-sm py-2 px-3 focus:outline-none focus:ring-indigo-500 focus:border-indigo-500 sm:text-sm" />
          </div>
          <div>
            <label className="block text-sm font-medium text-slate-700">Пароль</label>
            <input type="password" required value={password} onChange={e => setPassword(e.target.value)} className="mt-1 block w-full border border-slate-300 rounded-md shadow-sm py-2 px-3 focus:outline-none focus:ring-indigo-500 focus:border-indigo-500 sm:text-sm" />
          </div>
          <button type="submit" disabled={loading} className="w-full flex justify-center py-2 px-4 border border-transparent rounded-md shadow-sm text-sm font-medium text-white bg-indigo-600 hover:bg-indigo-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-indigo-500">
            {loading ? 'Вхід...' : 'Увійти'}
          </button>
        </form>
      </div>
    </div>
  </div>
 );
}
```

Б.3 Код адміністративної панелі

```
import React, { useState, useEffect } from 'react';
import { collection, query, getDocs, addDoc, updateDoc, deleteDoc, doc, serverTimestamp, orderBy, getDoc, setDoc } from 'firebase/firestore';
import { ref, uploadBytes, getDownloadURL } from 'firebase/storage';
import { db, storage } from '../firebase';
import { useAuth } from '../contexts/AuthContext';
import { Plus, Edit2, Trash2, X, Check, Eye, MessageSquare, Settings, Save, BarChart2, Download, MessageCircle } from 'lucide-react';
import { Link } from 'react-router-dom';
import { BarChart, Bar, XAxis, YAxis, CartesianGrid, Tooltip, ResponsiveContainer } from 'recharts';

export default function AdminDashboard() {
```

```
  const { user } = useAuth();
  const [fundraisers, setFundraisers] = useState<any[]>([]);
  const [loading, setLoading] = useState(true);
  const [isModalOpen, setIsModalOpen] = useState(false);
  const [editingId, setEditingId] = useState<string | null>(null);
  const [editingFundraiser, setEditingFundraiser] = useState<any>(null);
  const [error, setError] = useState<string | null>(null);
  const [deleteConfirmId, setDeleteConfirmId] = useState<string | null>(null);
  const [imageFiles, setImageFiles] = useState<File[]>([]);
  const [newImageUrl, setNewImageUrl] = useState('');
  const [uploading, setUploading] = useState(false);
  const [activeTab, setActiveTab] = useState<'fundraisers' | 'requests' | 'settings' |
'analytics'>('fundraisers');
  const [requests, setRequests] = useState<any[]>([]);
  const [donations, setDonations] = useState<any[]>([]);
  const [contactEmail, setContactEmail] = useState('');
  const [savingSettings, setSavingSettings] = useState(false);
  const [isUpdateModalOpen, setIsUpdateModalOpen] = useState(false);
  const [updateContent, setUpdateContent] = useState('');
  const [updateFundraiserId, setUpdateFundraiserId] = useState<string | null>(null);

  const [formData, setFormData] = useState({
    title: '',
    description: '',
    category: 'humanitarian',
    goalAmount: '',
    imageUrl: '',
    images: [] as string[],
    videoUrl: '',
    status: 'active',
    reportUrl: '',
    reportDescription: '',
    hasTimeframe: false,
    startDate: '',
    endDate: ''
  });

  const handleAddImageUrl = () => {
    if (newImageUrl.trim()) {
      setFormData(prev => ({
        ...prev,
        images: [...(prev.images || []), newImageUrl.trim()]
      }));
      setNewImageUrl('');
    }
  };

  const handleRemoveImageUrl = (index: number) => {
    setFormData(prev => {
      const updatedImages = [...(prev.images || [])];
```

```
      updatedImages.splice(index, 1);
      return { ...prev, images: updatedImages };
    });
  };

  const fetchFundraisers = async () => {
    setLoading(true);
    try {
      const q = query(collection(db, 'fundraisers'), orderBy('createdAt', 'desc'));
      const snapshot = await getDocs(q);

      if (snapshot.empty && user) {
        // Seed default fundraiser
        await addDoc(collection(db, 'fundraisers'), {
          title: "Збір на 5 дронів DJI Mavic 3T для 47-ї ОМБр",
          description: "Мета: закупівля 5 дронів з тепловізорами для виконання бойових завдань на гарячих напрямках.\n\nКожен дрон — це врятовані життя наших захисників та точне знищення ворожої техніки. Долучайтесь до збору, кожна гривня наближає перемогу!",
          category: "military",
          goalAmount: 1000000,
          currentAmount: 150000,
          imageUrl: "https://images.unsplash.com/photo-1527011045972-14c31d2448d7?ixlib=rb-4.0.3&auto=format&fit=crop&w=2000&q=80",
          videoUrl: "https://www.youtube.com/embed/u3xKvwE2XmI",
          status: "active",
          createdAt: serverTimestamp(),
          createdBy: user.uid
        });

        const newSnap = await getDocs(q);
        setFundraisers(newSnap.docs.map(doc => ({ id: doc.id, ...doc.data() })));
      } else {
        setFundraisers(snapshot.docs.map(doc => ({ id: doc.id, ...doc.data() })));
      }
    } catch (error) {
      console.error('Error fetching fundraisers:', error);
    } finally {
      setLoading(false);
    }
  };

  const fetchRequests = async () => {
    try {
      const q = query(collection(db, 'contact_requests'), orderBy('createdAt', 'desc'));
      const snapshot = await getDocs(q);
      const data = snapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));
      setRequests(data);
    } catch (err) {
      console.error('Error fetching requests:', err);
    }
```

```
};

const fetchSettings = async () => {
  try {
    const docRef = doc(db, 'settings', 'general');
    const docSnap = await getDoc(docRef);
    if (docSnap.exists()) {
      setContactEmail(docSnap.data().contactEmail || '');
    }
  } catch (err) {
    console.error('Error fetching settings:', err);
  }
};

const fetchDonations = async () => {
  try {
    const q = query(collection(db, 'donations'), orderBy('createdAt', 'desc'));
    const snapshot = await getDocs(q);
    const data = snapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));
    setDonations(data);
  } catch (err) {
    console.error('Error fetching donations:', err);
  }
};

useEffect(() => {
  fetchFundraisers();
  fetchRequests();
  fetchSettings();
  fetchDonations();
}, []);

const handleSaveSettings = async () => {
  setSavingSettings(true);
  try {
    const docRef = doc(db, 'settings', 'general');
    await setDoc(docRef, { contactEmail }, { merge: true });
    alert('Налаштування збережено');
  } catch (err) {
    console.error('Error saving settings:', err);
    alert('Помилка при збереженні налаштувань');
  }
  setSavingSettings(false);
};

const handleMarkRequestRead = async (id: string, currentStatus: string) => {
  try {
    const newStatus = currentStatus === 'new' ? 'read' : 'new';
    await updateDoc(doc(db, 'contact_requests', id), { status: newStatus });
    fetchRequests();
```

```
    } catch (err) {
      console.error('Error updating request:', err);
    }
  };

  const handleDeleteRequest = async (id: string) => {
    if (window.confirm('Ви впевнені, що хочете видалити цей запит?')) {
      try {
        await deleteDoc(doc(db, 'contact_requests', id));
        fetchRequests();
      } catch (err) {
        console.error('Error deleting request:', err);
      }
    }
  };

  const exportCSV = (data: any[], filename: string) => {
    if (!data || !data.length) return;
    const headers = Object.keys(data[0]).join(',');
    const rows = data.map(row =>
      Object.values(row).map(val => {
        if (typeof val === 'string') return `"${val.replace(/"/g, '""')}"`;
        if (val && typeof val === 'object' && 'toDate' in val && typeof (val as any).toDate ===
'function') return `"${(val as any).toDate().toISOString()}"`;
        return `"${val}"`;
      }).join(',')
    ).join('\n');
    const csv = `${headers}\n${rows}`;
    const blob = new Blob([csv], { type: 'text/csv;charset=utf-8;' });
    const url = window.URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url;
    a.download = filename;
    a.click();
    window.URL.revokeObjectURL(url);
  };

  const handlePostUpdate = async (e: React.FormEvent) => {
    e.preventDefault();
    if (!updateFundraiserId || !updateContent.trim()) return;
    setUploading(true);
    try {
      await addDoc(collection(db, 'fundraisers', updateFundraiserId, 'updates'), {
        content: updateContent,
        createdAt: serverTimestamp()
      });
      setIsUpdateModalOpen(false);
      setUpdateContent('');
      setUpdateFundraiserId(null);
      alert('Оновлення успішно додано!');
```

```
    } catch (err) {
      console.error('Error posting update:', err);
      alert('Помилка при додаванні оновлення');
    } finally {
      setUploading(false);
    }
  };

  const handleInputChange = (e: React.ChangeEvent<HTMLInputElement | HTMLTextAreaElement |
HTMLSelectElement>) => {
    const { name, value, type } = e.target;
    if (type === 'checkbox') {
      const checked = (e.target as HTMLInputElement).checked;
      setFormData(prev => ({ ...prev, [name]: checked }));
    } else {
      setFormData(prev => ({ ...prev, [name]: value }));
    }
  };

  const handleFileChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    if (e.target.files) {
      setImageFiles(Array.from(e.target.files));
    }
  };

  const uploadImages = async (fundraiserId: string): Promise<string[]> => {
    if (imageFiles.length === 0) return [];

    const uploadPromises = imageFiles.map(async (file) => {
      const fileRef = ref(storage, `fundraisers/${fundraiserId}/images/${Date.now()}_${file.name}`);
      await uploadBytes(fileRef, file);
      return getDownloadURL(fileRef);
    });

    return Promise.all(uploadPromises);
  };

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    if (!user) return;
    setUploading(true);
    setError(null);

    try {
      const dataToSave: any = {
        title: formData.title,
        description: formData.description,
        category: formData.category,
        goalAmount: Number(formData.goalAmount),
        videoUrl: formData.videoUrl,
```

```
  status: formData.status,
  hasTimeframe: formData.hasTimeframe,
  startDate: formData.hasTimeframe ? formData.startDate : null,
  endDate: formData.hasTimeframe ? formData.endDate : null,
};

if (formData.status === 'completed') {
  dataToSave.reportUrl = formData.reportUrl;
  dataToSave.reportDescription = formData.reportDescription;
}

if (editingId) {
  const updateData = { ...dataToSave };
  // Fix for older documents that might be missing required fields
  if (editingFundraiser) {
    if (editingFundraiser.currentAmount === undefined) updateData.currentAmount = 0;
    if (!editingFundraiser.createdBy) updateData.createdBy = user.uid;
    if (!editingFundraiser.createdAt) updateData.createdAt = serverTimestamp();
  }

  const newImageUrls = await uploadImages(editingId);
  updateData.images = [...(formData.images || []), ...newImageUrls];
  if (updateData.images.length > 0) {
    updateData.imageUrl = updateData.images[0];
  } else {
    updateData.imageUrl = '';
  }

  await updateDoc(doc(db, 'fundraisers', editingId), updateData);
} else {
  const docRef = await addDoc(collection(db, 'fundraisers'), {
    ...dataToSave,
    images: formData.images || [],
    currentAmount: 0,
    createdAt: serverTimestamp(),
    createdBy: user.uid
  });

  const newImageUrls = await uploadImages(docRef.id);
  if (newImageUrls.length > 0 || (formData.images && formData.images.length > 0)) {
    const finalImages = [...(formData.images || []), ...newImageUrls];
    await updateDoc(docRef, {
      images: finalImages,
      imageUrl: finalImages[0] || ''
    });
  }
}

setIsModalOpen(false);
setEditingId(null);
```

```
      setEditingFundraiser(null);
      setImageFiles([]);
      setError(null);
      setFormData({ title: '', description: '', category: 'humanitarian', goalAmount: '', imageUrl: '',
images: [], videoUrl: '', status: 'active', reportUrl: '', reportDescription: '', hasTimeframe: false,
startDate: '', endDate: '' });
      fetchFundraisers();
    } catch (err: any) {
      console.error('Error saving fundraiser:', err);
      setError('Помилка при збереженні: ' + (err.message || 'Невідома помилка'));
    } finally {
      setUploading(false);
    }
  };

  const handleEdit = (fundraiser: any) => {
    setFormData({
      title: fundraiser.title || '',
      description: fundraiser.description || '',
      category: fundraiser.category || 'humanitarian',
      goalAmount: fundraiser.goalAmount ? fundraiser.goalAmount.toString() : '',
      imageUrl: fundraiser.imageUrl || '',
      images: fundraiser.images || (fundraiser.imageUrl ? [fundraiser.imageUrl] : []),
      videoUrl: fundraiser.videoUrl || '',
      status: fundraiser.status || 'active',
      reportUrl: fundraiser.reportUrl || '',
      reportDescription: fundraiser.reportDescription || '',
      hasTimeframe: fundraiser.hasTimeframe || false,
      startDate: fundraiser.startDate || '',
      endDate: fundraiser.endDate || ''
    });
    setEditingId(fundraiser.id);
    setEditingFundraiser(fundraiser);
    setImageFiles([]);
    setNewImageUrl('');
    setIsModalOpen(true);
  };

  const handleDelete = async (id: string) => {
    setDeleteConfirmId(id);
  };

  const confirmDelete = async () => {
    if (!deleteConfirmId) return;
    try {
      await deleteDoc(doc(db, 'fundraisers', deleteConfirmId));
      setDeleteConfirmId(null);
      fetchFundraisers();
    } catch (err: any) {
      console.error('Error deleting fundraiser:', err);
```

```
      setError('Помилка при видаленні: ' + (err.message || 'Невідома помилка'));
      setDeleteConfirmId(null);
    }
  };

  const handleResetData = async () => {
    if (!window.confirm('Ви впевнені, що хочете видалити всі збори та створити нові?')) return;
    setLoading(true);
    try {
      // Delete all
      const q = query(collection(db, 'fundraisers'));
      const snapshot = await getDocs(q);
      const deletePromises = snapshot.docs.map(d => deleteDoc(doc(db, 'fundraisers', d.id)));
      await Promise.all(deletePromises);

      // Create new
      await addDoc(collection(db, 'fundraisers'), {
        title: "Збір на 5 дронів DJI Mavic 3T для 47-ї ОМБр",
        description: "Мета: закупівля 5 дронів з тепловізорами для виконання бойових завдань на гарячих напрямках.\n\nКожен дрон — це врятовані життя наших захисників та точне знищення ворожої техніки. Долучайтесь до збору, кожна гривня наближає перемогу!",
        category: "military",
        goalAmount: 1000000,
        currentAmount: 150000,
        imageUrl: "https://images.unsplash.com/photo-1527011045972-14c31d2448d7?ixlib=rb-4.0.3&auto=format&fit=crop&w=2000&q=80",
        videoUrl: "https://www.youtube.com/embed/u3xKvwE2XmI",
        status: "active",
        createdAt: serverTimestamp(),
        createdBy: user?.uid || "admin"
      });

      await addDoc(collection(db, 'fundraisers'), {
        title: "Медикаменти для дитячої лікарні 'Охматдит'",
        description: "Збираємо кошти на закупівлю життєво необхідних медикаментів для відділення інтенсивної терапії. Ваша допомога рятує дитячі життя.",
        category: "medical",
        goalAmount: 500000,
        currentAmount: 250000,
        imageUrl: "https://images.unsplash.com/photo-1516549655169-df83a0774514?ixlib=rb-4.0.3&auto=format&fit=crop&w=2000&q=80",
        status: "active",
        createdAt: serverTimestamp(),
        createdBy: user?.uid || "admin"
      });

      await addDoc(collection(db, 'fundraisers'), {
        title: "Гуманітарна допомога переселенцям з Бахмута",
        description: "Забезпечення продуктовими наборами, теплим одягом та засобами гігієни сімей, які втратили свої домівки.",
```

```
        category: "humanitarian",
        goalAmount: 200000,
        currentAmount: 200000,
        imageUrl: "https://images.unsplash.com/photo-1469571486292-0ba58a3f068b?ixlib=rb-
4.0.3&auto=format&fit=crop&w=2000&q=80",
        status: "completed",
        reportUrl: "https://example.com/report.pdf",
        reportDescription: "Завдяки вашій підтримці ми змогли забезпечити 500 сімей необхідними
речами на зимовий період. Дякуємо кожному!",
        createdAt: serverTimestamp(),
        createdBy: user?.uid || "admin"
      });

      fetchFundraisers();
    } catch (err: any) {
      console.error('Error resetting data:', err);
      setError('Помилка при скиданні даних: ' + (err.message || 'Невідома помилка'));
      setLoading(false);
    }
  };

  const openNewModal = () => {
    setEditingId(null);
    setEditingFundraiser(null);
    setError(null);
    setImageFiles([]);
    setNewImageUrl('');
    setFormData({ title: '', description: '', category: 'humanitarian', goalAmount: '', imageUrl: '', images:
[], videoUrl: '', status: 'active', reportUrl: '', reportDescription: '', hasTimeframe: false, startDate: '',
endDate: '' });
    setIsModalOpen(true);
  };

  return (
    <div className="bg-slate-50 min-h-screen py-12">
      <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
        <div className="flex justify-between items-center mb-8">
          <div>
            <h1 className="text-3xl font-extrabold text-slate-900">Адмін-панель</h1>
            <p className="mt-2 text-slate-600">Керування зборами та кампаніями</p>
          </div>
          <div className="flex gap-3">
            <button
              onClick={handleResetData}
              className="inline-flex items-center px-4 py-2 border border-slate-300 rounded-xl shadow-
sm text-sm font-medium text-slate-700 bg-white hover:bg-slate-50 focus:outline-none focus:ring-2
focus:ring-offset-2 focus:ring-indigo-500 transition-colors"
            >
              Скинути дані
            </button>
```

```
          {activeTab === 'fundraisers' && (
           <button
            onClick={openNewModal}
            className="inline-flex items-center px-4 py-2 border border-transparent rounded-xl shadow-sm text-sm font-medium text-white bg-indigo-600 hover:bg-indigo-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-indigo-500 transition-colors"
           >
            <Plus className="mr-2 h-5 w-5" />
            Створити збір
           </button>
          )}
         </div>
        </div>

        {/* Tabs */}
        <div className="mb-8 border-b border-slate-200">
         <nav className="-mb-px flex space-x-8">
          <button
           onClick={() => setActiveTab('fundraisers')}
           className={`${
            activeTab === 'fundraisers'
             ? 'border-indigo-500 text-indigo-600'
             : 'border-transparent text-slate-500 hover:text-slate-700 hover:border-slate-300'
           } whitespace-nowrap py-4 px-1 border-b-2 font-medium text-sm flex items-center`}
          >
           Збори
          </button>
          <button
           onClick={() => setActiveTab('requests')}
           className={`${
            activeTab === 'requests'
             ? 'border-indigo-500 text-indigo-600'
             : 'border-transparent text-slate-500 hover:text-slate-700 hover:border-slate-300'
           } whitespace-nowrap py-4 px-1 border-b-2 font-medium text-sm flex items-center`}
          >
           <MessageSquare className="mr-2 h-4 w-4" />
           Запити з контактів
           {requests.filter(r => r.status === 'new').length > 0 && (
            <span className="ml-2 bg-rose-100 text-rose-600 py-0.5 px-2 rounded-full text-xs">
             {requests.filter(r => r.status === 'new').length}
            </span>
           )}
          </button>
          <button
           onClick={() => setActiveTab('settings')}
           className={`${
            activeTab === 'settings'
             ? 'border-indigo-500 text-indigo-600'
             : 'border-transparent text-slate-500 hover:text-slate-700 hover:border-slate-300'
           } whitespace-nowrap py-4 px-1 border-b-2 font-medium text-sm flex items-center`}
```

```
          >
            <Settings className="mr-2 h-4 w-4" />
            Налаштування
          </button>
          <button
            onClick={() => setActiveTab('analytics')}
            className={`${
              activeTab === 'analytics'
                ? 'border-indigo-500 text-indigo-600'
                : 'border-transparent text-slate-500 hover:text-slate-700 hover:border-slate-300'
            } whitespace-nowrap py-4 px-1 border-b-2 font-medium text-sm flex items-center`}
          >
            <BarChart2 className="mr-2 h-4 w-4" />
            Аналітика
          </button>
        </nav>
      </div>

      {activeTab === 'fundraisers' && (
        /* Table */
        <div className="bg-white shadow-sm rounded-2xl border border-slate-200 overflow-
hidden">
          <div className="overflow-x-auto">
            <table className="min-w-full divide-y divide-slate-200">
              <thead className="bg-slate-50">
                <tr>
                <th scope="col" className="px-6 py-4 text-left text-xs font-bold text-slate-500 uppercase
tracking-wider">
                  Назва
                </th>
                <th scope="col" className="px-6 py-4 text-left text-xs font-bold text-slate-500 uppercase
tracking-wider">
                  Категорія
                </th>
                <th scope="col" className="px-6 py-4 text-left text-xs font-bold text-slate-500 uppercase
tracking-wider">
                  Прогрес
                </th>
                <th scope="col" className="px-6 py-4 text-left text-xs font-bold text-slate-500 uppercase
tracking-wider">
                  Статус
                </th>
                <th scope="col" className="px-6 py-4 text-right text-xs font-bold text-slate-500
uppercase tracking-wider">
                  Дії
                </th>
              </tr>
            </thead>
            <tbody className="bg-white divide-y divide-slate-200">
              {loading ? (
```

```
            <tr>
              <td colSpan={5} className="px-6 py-12 text-center text-slate-500">
                Завантаження...
              </td>
            </tr>
          ) : fundraisers.length === 0 ? (
            <tr>
              <td colSpan={5} className="px-6 py-12 text-center text-slate-500">
                Немає зборів. Створіть перший збір.
              </td>
            </tr>
          ) : (
            fundraisers.map((fundraiser) => (
              <tr key={fundraiser.id} className="hover:bg-slate-50 transition-colors">
                <td className="px-6 py-4 whitespace-nowrap">
                  <div className="flex items-center">
                    <div className="h-10 w-10 flex-shrink-0 rounded-lg bg-slate-200 overflow-
hidden">
                      {fundraiser.images && fundraiser.images.length > 0 ? (
                        <img className="h-10 w-10 object-cover" src={fundraiser.images[0]} alt=""
referrerPolicy="no-referrer" />
                      ) : fundraiser.imageUrl ? (
                        <img className="h-10 w-10 object-cover" src={fundraiser.imageUrl} alt=""
referrerPolicy="no-referrer" />
                      ) : (
                        <div className="h-10 w-10 bg-slate-200"></div>
                      )}
                    </div>
                    <div className="ml-4">
                      <div className="text-sm font-medium text-slate-900 max-w-[200px] truncate"
title={fundraiser.title}>
                        {fundraiser.title}
                      </div>
                    </div>
                  </div>
                </td>
                <td className="px-6 py-4 whitespace-nowrap">
                  <span className="px-2.5 py-1 inline-flex text-xs leading-5 font-semibold rounded-
full bg-indigo-100 text-indigo-800">
                    {fundraiser.category === 'humanitarian' ? 'Гуманітарна' :
                     fundraiser.category === 'military' ? 'Військова' :
                     fundraiser.category === 'medical' ? 'Медична' : 'Інше'}
                  </span>
                </td>
                <td className="px-6 py-4 whitespace-nowrap">
                  <div className="text-sm text-slate-900">{(fundraiser.currentAmount ||
0).toLocaleString()} ₴</div>
                  <div className="text-xs text-slate-500">з {(fundraiser.goalAmount ||
0).toLocaleString()} ₴</div>
                </td>
```

```
                <td className="px-6 py-4 whitespace-nowrap">
                  <span className={`px-2.5 py-1 inline-flex text-xs leading-5 font-semibold rounded-
full ${
                    fundraiser.status === 'active' ? 'bg-emerald-100 text-emerald-800' :
                    fundraiser.status === 'completed' ? 'bg-blue-100 text-blue-800' :
                    'bg-amber-100 text-amber-800'
                  }`}>
                    {fundraiser.status === 'active' ? 'Активний' :
                     fundraiser.status === 'completed' ? 'Завершений' : 'На паузі'}
                  </span>
                </td>
                <td className="px-6 py-4 whitespace-nowrap text-right text-sm font-medium">
                  <div className="flex justify-end gap-3">
                    <button
                      onClick={() => {
                        setUpdateFundraiserId(fundraiser.id);
                        setIsUpdateModalOpen(true);
                      }}
                      className="text-slate-400 hover:text-blue-600 transition-colors"
                      title="Додати оновлення"
                    >
                      <MessageCircle className="h-5 w-5" />
                    </button>
                    <Link to={`/fundraisers/${fundraiser.id}`} className="text-slate-400 hover:text-
indigo-600 transition-colors" title="Переглянути">
                      <Eye className="h-5 w-5" />
                    </Link>
                    <button onClick={() => handleEdit(fundraiser)} className="text-slate-400
hover:text-emerald-600 transition-colors" title="Редагувати">
                      <Edit2 className="h-5 w-5" />
                    </button>
                    <button onClick={() => handleDelete(fundraiser.id)} className="text-slate-400
hover:text-rose-600 transition-colors" title="Видалити">
                      <Trash2 className="h-5 w-5" />
                    </button>
                  </div>
                </td>
              </tr>
            ))
          )}
        </tbody>
      </table>
    </div>
  </div>
  )}

  {activeTab === 'requests' && (
    <div className="bg-white shadow-sm rounded-2xl border border-slate-200 overflow-
hidden">
      {requests.length === 0 ? (
```

```
          <div className="px-6 py-12 text-center text-slate-500">
            Немає запитів
          </div>
        ) : (
          <div className="divide-y divide-slate-200">
            {requests.map((request) => (
              <div key={request.id} className={`p-6 ${request.status === 'new' ? 'bg-indigo-50/50' :
'bg-white'}`}>
                <div className="flex justify-between items-start">
                  <div className="flex-1">
                    <div className="flex items-center gap-3 mb-2">
                      <h3 className="text-lg font-medium text-slate-900">{request.name}</h3>
                      {request.status === 'new' && (
                        <span className="inline-flex items-center px-2.5 py-0.5 rounded-full text-xs font-
medium bg-indigo-100 text-indigo-800">
                          Новий
                        </span>
                      )}
                      <span className="text-sm text-slate-500">
                        {request.createdAt?.toDate().toLocaleDateString('uk-UA', {
                          day: '2-digit', month: '2-digit', year: 'numeric',
                          hour: '2-digit', minute: '2-digit'
                        })}
                      </span>
                    </div>
                    <div className="flex items-center gap-2 text-sm text-slate-600 mb-4">
                      <a href={`mailto:${request.email}`} className="hover:text-indigo-600 transition-
colors">
                        {request.email}
                      </a>
                    </div>
                    <div className="bg-slate-50 rounded-xl p-4 text-slate-700 whitespace-pre-wrap">
                      {request.message}
                    </div>
                  </div>
                  <div className="ml-6 flex flex-col gap-2">
                    <button
                      onClick={() => handleMarkRequestRead(request.id, request.status)}
                      className={`inline-flex items-center justify-center p-2 rounded-xl border transition-
colors ${
                        request.status === 'new'
                          ? 'border-indigo-200 text-indigo-600 hover:bg-indigo-50'
                          : 'border-slate-200 text-slate-400 hover:bg-slate-50'
                      }`}
                      title={request.status === 'new' ? 'Позначити як прочитане' : 'Позначити як нове'}
                    >
                      <Check className="h-5 w-5" />
                    </button>
                    <button
                      onClick={() => handleDeleteRequest(request.id)}
```

```
                    className="inline-flex items-center justify-center p-2 rounded-xl border border-rose-200 text-rose-600 hover:bg-rose-50 transition-colors"
                    title="Видалити"
                   >
                    <Trash2 className="h-5 w-5" />
                   </button>
                  </div>
                 </div>
                </div>
               ))}
              </div>
             )}
            </div>
           )}

           {activeTab === 'settings' && (
            <div className="bg-white shadow-sm rounded-2xl border border-slate-200 p-8 max-w-2xl">
             <h2 className="text-xl font-bold text-slate-900 mb-6">Налаштування сповіщень</h2>
             <div className="space-y-6">
              <div>
               <label htmlFor="contactEmail" className="block text-sm font-medium text-slate-700 mb-2">
                Email для отримання запитів з форми контактів
               </label>
               <p className="text-sm text-slate-500 mb-4">
                На цю адресу будуть надходити листи, коли хтось заповнює форму на сторінці "Контакти".
               </p>
               <input
                type="email"
                id="contactEmail"
                value={contactEmail}
                onChange={(e) => setContactEmail(e.target.value)}
                placeholder="admin@example.com"
                className="block w-full px-4 py-3 border border-slate-300 rounded-xl bg-white placeholder-slate-400 focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:border-transparent transition-shadow"
               />
              </div>
              <div className="pt-4">
               <button
                onClick={handleSaveSettings}
                disabled={savingSettings}
                className="inline-flex items-center px-6 py-3 border border-transparent rounded-xl shadow-sm text-base font-medium text-white bg-indigo-600 hover:bg-indigo-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-indigo-500 disabled:opacity-50 transition-colors"
               >
                <Save className="mr-2 h-5 w-5" />
                {savingSettings ? 'Збереження...' : 'Зберегти налаштування'}
               </button>
```

```
          </div>
        </div>
      </div>
    )}

    {activeTab === 'analytics' && (
      <div className="space-y-8">
        <div className="flex justify-end gap-4">
          <button
            onClick={() => exportCSV(fundraisers, 'fundraisers.csv')}
            className="inline-flex items-center px-4 py-2 border border-slate-300 rounded-xl shadow-sm text-sm font-medium text-slate-700 bg-white hover:bg-slate-50 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-indigo-500 transition-colors"
          >
            <Download className="mr-2 h-4 w-4" /> Експорт зборів
          </button>
          <button
            onClick={() => exportCSV(donations, 'donations.csv')}
            className="inline-flex items-center px-4 py-2 border border-slate-300 rounded-xl shadow-sm text-sm font-medium text-slate-700 bg-white hover:bg-slate-50 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-indigo-500 transition-colors"
          >
            <Download className="mr-2 h-4 w-4" /> Експорт донатів
          </button>
        </div>

        <div className="grid grid-cols-1 lg:grid-cols-2 gap-8">
          <div className="bg-white p-6 rounded-2xl shadow-sm border border-slate-200">
            <h3 className="text-lg font-bold text-slate-900 mb-6">Донати по днях</h3>
            <div className="h-80">
              <ResponsiveContainer width="100%" height="100%">
                <BarChart data={
                  Object.entries(
                    donations.reduce((acc, curr) => {
                      const date = curr.createdAt?.toDate ? curr.createdAt.toDate().toLocaleDateString('uk-UA') : 'Невідомо';
                      acc[date] = (acc[date] || 0) + (curr.amount || 0);
                      return acc;
                    }, {} as Record<string, number>)
                  ).map(([date, amount]) => ({ date, amount })).slice(0, 7).reverse()
                }>
                  <CartesianGrid strokeDasharray="3 3" vertical={false} stroke="#e2e8f0" />
                  <XAxis dataKey="date" axisLine={false} tickLine={false} tick={{ fill: '#64748b', fontSize: 12 }} />
                  <YAxis axisLine={false} tickLine={false} tick={{ fill: '#64748b', fontSize: 12 }} />
                  <Tooltip cursor={{ fill: '#f1f5f9' }} contentStyle={{ borderRadius: '12px', border: 'none', boxShadow: '0 4px 6px -1px rgb(0 0 0 / 0.1)' }} />
                  <Bar dataKey="amount" fill="#4f46e5" radius={[4, 4, 0, 0]} />
                </BarChart>
              </ResponsiveContainer>
```

```
            </div>
          </div>

          <div className="bg-white p-6 rounded-2xl shadow-sm border border-slate-200">
            <h3 className="text-lg font-bold text-slate-900 mb-6">Збори за категоріями</h3>
            <div className="h-80">
              <ResponsiveContainer width="100%" height="100%">
                <BarChart data={
                  Object.entries(
                    fundraisers.reduce((acc, curr) => {
                      const cat = curr.category === 'humanitarian' ? 'Гуманітарна' : curr.category ===
'military' ? 'Військова' : curr.category === 'medical' ? 'Медична' : 'Інше';
                      acc[cat] = (acc[cat] || 0) + (curr.currentAmount || 0);
                      return acc;
                    }, {} as Record<string, number>)
                  ).map(([category, amount]) => ({ category, amount }))
                }>
                  <CartesianGrid strokeDasharray="3 3" vertical={false} stroke="#e2e8f0" />
                  <XAxis dataKey="category" axisLine={false} tickLine={false} tick={{ fill: '#64748b',
fontSize: 12 }} />
                  <YAxis axisLine={false} tickLine={false} tick={{ fill: '#64748b', fontSize: 12 }} />
                  <Tooltip cursor={{ fill: '#f1f5f9' }} contentStyle={{ borderRadius: '12px', border:
'none', boxShadow: '0 4px 6px -1px rgb(0 0 0 / 0.1)' }} />
                  <Bar dataKey="amount" fill="#10b981" radius={[4, 4, 0, 0]} />
                </BarChart>
              </ResponsiveContainer>
            </div>
          </div>
        </div>
      )}
    </div>

    {/* Update Modal */}
    {isUpdateModalOpen && (
      <div className="fixed inset-0 z-50 overflow-y-auto" aria-labelledby="modal-title"
role="dialog" aria-modal="true">
        <div className="flex items-end justify-center min-h-screen pt-4 px-4 pb-20 text-center
sm:block sm:p-0">
          <div className="fixed inset-0 bg-slate-900/50 transition-opacity" aria-hidden="true"
onClick={() => setIsUpdateModalOpen(false)}></div>
          <span className="hidden sm:inline-block sm:align-middle sm:h-screen" aria-
hidden="true">&#8203;</span>
          <div className="relative z-10 inline-block align-bottom bg-white rounded-3xl text-left
overflow-hidden shadow-xl transform transition-all sm:my-8 sm:align-middle sm:max-w-2xl w-full">
            <div className="bg-white px-4 pt-5 pb-4 sm:p-8 sm:pb-6">
              <div className="flex justify-between items-center mb-6">
                <h3 className="text-2xl leading-6 font-bold text-slate-900" id="modal-title">
                  Додати оновлення до збору
                </h3>
```

```
            <button onClick={() => setIsUpdateModalOpen(false)} className="text-slate-400
hover:text-slate-500">
              <X className="h-6 w-6" />
            </button>
          </div>
          <form onSubmit={handlePostUpdate} className="space-y-6">
            <div>
              <label className="block text-sm font-medium text-slate-700 mb-2">Текст оновлення
*</label>
              <textarea
                value={updateContent}
                onChange={(e) => setUpdateContent(e.target.value)}
                className="block w-full px-4 py-3 border border-slate-300 rounded-xl bg-white
placeholder-slate-400 focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:border-transparent
transition-shadow resize-none"
                placeholder="Що нового?"
                rows={6}
                required
              />
            </div>
            <div className="pt-4 flex justify-end gap-3 border-t border-slate-100">
              <button type="button" onClick={() => setIsUpdateModalOpen(false)} className="px-6
py-3 border border-slate-300 rounded-xl shadow-sm text-base font-medium text-slate-700 bg-white
hover:bg-slate-50 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-indigo-500
transition-colors">
                Скасувати
              </button>
              <button type="submit" disabled={uploading} className="inline-flex justify-center px-6
py-3 border border-transparent rounded-xl shadow-sm text-base font-medium text-white bg-indigo-
600 hover:bg-indigo-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-indigo-500
disabled:opacity-50 transition-colors">
                {uploading ? 'Публікація...' : 'Опублікувати'}
              </button>
            </div>
          </form>
        </div>
      </div>
    </div>
  )}

  {/* Modal */}
  {isModalOpen && (
    <div className="fixed inset-0 z-50 overflow-y-auto" aria-labelledby="modal-title"
role="dialog" aria-modal="true">
      <div className="flex items-end justify-center min-h-screen pt-4 px-4 pb-20 text-center
sm:block sm:p-0">
        <div className="fixed inset-0 bg-slate-900/50 transition-opacity" aria-hidden="true"
onClick={() => setIsModalOpen(false)}></div>
        <span className="hidden sm:inline-block sm:align-middle sm:h-screen" aria-
```

```
hidden="true">&#8203;</span>
        <div className="relative z-10 inline-block align-bottom bg-white rounded-3xl text-left
overflow-hidden shadow-xl transform transition-all sm:my-8 sm:align-middle sm:max-w-2xl w-full">
          <div className="bg-white px-4 pt-5 pb-4 sm:p-8 sm:pb-6">
            <div className="flex justify-between items-center mb-6">
              <h3 className="text-2xl leading-6 font-bold text-slate-900" id="modal-title">
                {editingId ? 'Редагувати збір' : 'Новий збір'}
              </h3>
              <button onClick={() => setIsModalOpen(false)} className="text-slate-400 hover:text-
slate-500">
                <X className="h-6 w-6" />
              </button>
            </div>

            {error && (
              <div className="mb-6 bg-red-50 border border-red-200 text-red-700 px-4 py-3 rounded-
xl">
                {error}
              </div>
            )}

            <form id="fundraiser-form" onSubmit={handleSubmit} className="space-y-6">
              <div>
                <label htmlFor="title" className="block text-sm font-medium text-slate-700 mb-
2">Назва збору *</label>
                <input type="text" name="title" id="title" required value={formData.title}
onChange={handleInputChange} className="block w-full px-4 py-3 border border-slate-300
rounded-xl bg-white placeholder-slate-400 focus:outline-none focus:ring-2 focus:ring-indigo-500
focus:border-transparent transition-shadow" />
              </div>

              <div>
                <label htmlFor="description" className="block text-sm font-medium text-slate-700 mb-
2">Опис *</label>
                <textarea
                  name="description"
                  id="description"
                  required
                  rows={6}
                  value={formData.description}
                  onChange={handleInputChange}
                  className="block w-full px-4 py-3 border border-slate-300 rounded-xl bg-white
placeholder-slate-400 focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:border-transparent
transition-shadow resize-none"
                ></textarea>
              </div>

              <div className="grid grid-cols-1 sm:grid-cols-2 gap-6">
                <div>
                  <label htmlFor="category" className="block text-sm font-medium text-slate-700 mb-
```

```
2">Категорія *</label>
                <select name="category" id="category" required value={formData.category}
onChange={handleInputChange} className="block w-full px-4 py-3 border border-slate-300
rounded-xl bg-white focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:border-transparent
transition-shadow">
                  <option value="humanitarian">Гуманітарна</option>
                  <option value="military">Військова</option>
                  <option value="medical">Медична</option>
                  <option value="other">Інше</option>
                </select>
              </div>

              <div>
                <label htmlFor="goalAmount" className="block text-sm font-medium text-slate-700
mb-2">Ціль (₴) *</label>
                <input type="number" name="goalAmount" id="goalAmount" min="1" required
value={formData.goalAmount} onChange={handleInputChange} className="block w-full px-4 py-
3 border border-slate-300 rounded-xl bg-white placeholder-slate-400 focus:outline-none focus:ring-2
focus:ring-indigo-500 focus:border-transparent transition-shadow" />
              </div>
            </div>

            <div className="bg-slate-50 p-4 rounded-xl border border-slate-200">
              <div className="flex items-center mb-4">
                <input
                  id="hasTimeframe"
                  name="hasTimeframe"
                  type="checkbox"
                  checked={formData.hasTimeframe}
                  onChange={handleInputChange}
                  className="h-4 w-4 text-indigo-600 focus:ring-indigo-500 border-slate-300 rounded"
                />
                <label htmlFor="hasTimeframe" className="ml-2 block text-sm font-medium text-
slate-700">
                  Задати часові рамки збору
                </label>
              </div>

              {formData.hasTimeframe && (
                <div className="grid grid-cols-1 sm:grid-cols-2 gap-6 mt-4">
                  <div>
                    <label htmlFor="startDate" className="block text-sm font-medium text-slate-700
mb-2">Дата початку</label>
                    <input type="datetime-local" name="startDate" id="startDate"
value={formData.startDate} onChange={handleInputChange} className="block w-full px-4 py-3
border border-slate-300 rounded-xl bg-white focus:outline-none focus:ring-2 focus:ring-indigo-500
focus:border-transparent transition-shadow" />
                  </div>
                  <div>
                    <label htmlFor="endDate" className="block text-sm font-medium text-slate-700
```

```
mb-2">Дата завершення</label>
                    <input type="datetime-local" name="endDate" id="endDate"
value={formData.endDate} onChange={handleInputChange} className="block w-full px-4 py-3
border border-slate-300 rounded-xl bg-white focus:outline-none focus:ring-2 focus:ring-indigo-500
focus:border-transparent transition-shadow" />
                  </div>
                 </div>
                )}
               </div>

               <div>
                <label htmlFor="images" className="block text-sm font-medium text-slate-700 mb-
2">Зображення (завантаження або посилання)</label>

                <div className="mb-4">
                 <p className="text-xs text-slate-500 mb-2">Варіант 1: Завантажити з
комп'ютера</p>
                 <input type="file" id="images" multiple accept="image/*"
onChange={handleFileChange} className="block w-full text-sm text-slate-500 file:mr-4 file:py-2
file:px-4 file:rounded-full file:border-0 file:text-sm file:font-semibold file:bg-indigo-50 file:text-
indigo-700 hover:file:bg-indigo-100 transition-colors" />
                </div>

                <div className="mb-4">
                 <p className="text-xs text-slate-500 mb-2">Варіант 2: Додати за посиланням</p>
                 <div className="flex gap-2">
                  <input
                   type="url"
                   value={newImageUrl}
                   onChange={(e) => setNewImageUrl(e.target.value)}
                   className="block w-full px-4 py-2 border border-slate-300 rounded-xl bg-white
placeholder-slate-400 focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:border-transparent
transition-shadow"
                   placeholder="https://example.com/image.jpg"
                  />
                  <button
                   type="button"
                   onClick={handleAddImageUrl}
                   disabled={!newImageUrl.trim()}
                   className="px-4 py-2 bg-indigo-50 text-indigo-600 rounded-xl font-medium
hover:bg-indigo-100 disabled:opacity-50 disabled:cursor-not-allowed transition-colors flex items-
center"
                  >
                   <Plus className="h-5 w-5" />
                  </button>
                 </div>
                </div>

                {formData.images && formData.images.length > 0 && (
                 <div className="mt-4 grid grid-cols-4 gap-2">
```

```
                {formData.images.map((img, idx) => (
                  <div key={idx} className="relative aspect-square rounded-lg overflow-hidden
border border-slate-200 group">
                    <img src={img} alt="" className="w-full h-full object-cover" referrerPolicy="no-
referrer" />
                    <button
                      type="button"
                      onClick={() => handleRemoveImageUrl(idx)}
                      className="absolute top-1 right-1 bg-rose-500 text-white p-1 rounded-full
opacity-0 group-hover:opacity-100 transition-opacity"
                    >
                      <X className="h-3 w-3" />
                    </button>
                  </div>
                ))}
              </div>
            )}
          </div>

          <div>
            <label htmlFor="videoUrl" className="block text-sm font-medium text-slate-700 mb-
2">URL відео (YouTube Embed)</label>
            <input type="url" name="videoUrl" id="videoUrl" value={formData.videoUrl}
onChange={handleInputChange} className="block w-full px-4 py-3 border border-slate-300
rounded-xl bg-white placeholder-slate-400 focus:outline-none focus:ring-2 focus:ring-indigo-500
focus:border-transparent transition-shadow" placeholder="https://www.youtube.com/embed/..." />
          </div>

          <div>
            <label htmlFor="status" className="block text-sm font-medium text-slate-700 mb-
2">Статус *</label>
            <select name="status" id="status" required value={formData.status}
onChange={handleInputChange} className="block w-full px-4 py-3 border border-slate-300
rounded-xl bg-white focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:border-transparent
transition-shadow">
              <option value="active">Активний</option>
              <option value="completed">Завершений</option>
              <option value="paused">На паузі</option>
            </select>
          </div>

          {formData.status === 'completed' && (
            <div className="space-y-6 border-t border-slate-200 pt-6 mt-6">
              <h4 className="text-lg font-bold text-slate-900">Звітність по збору</h4>
              <div>
                <label htmlFor="reportUrl" className="block text-sm font-medium text-slate-700
mb-2">URL документа зі звітом (PDF, Google Drive тощо)</label>
                <input type="url" name="reportUrl" id="reportUrl" value={formData.reportUrl}
onChange={handleInputChange} className="block w-full px-4 py-3 border border-slate-300
rounded-xl bg-white placeholder-slate-400 focus:outline-none focus:ring-2 focus:ring-indigo-500
```

```
focus:border-transparent transition-shadow" placeholder="https://..." />
                </div>
                <div>
                  <label htmlFor="reportDescription" className="block text-sm font-medium text-slate-700 mb-2">Опис звіту (куди пішли кошти)</label>
                  <textarea name="reportDescription" id="reportDescription" rows={3} value={formData.reportDescription} onChange={handleInputChange} className="block w-full px-4 py-3 border border-slate-300 rounded-xl bg-white placeholder-slate-400 focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:border-transparent transition-shadow resize-none" placeholder="Коротко опишіть результати..."></textarea>
                </div>
              </div>
            )}
          </form>
        </div>
        <div className="bg-slate-50 px-4 py-4 sm:px-8 sm:flex sm:flex-row-reverse border-t border-slate-200">
          <button type="submit" form="fundraiser-form" disabled={uploading} className="w-full inline-flex justify-center rounded-xl border border-transparent shadow-sm px-6 py-3 bg-indigo-600 text-base font-medium text-white hover:bg-indigo-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-indigo-500 sm:ml-3 sm:w-auto sm:text-sm transition-colors disabled:opacity-50">
            {uploading ? 'Збереження...' : <><Check className="mr-2 h-5 w-5" /> Зберегти</>}
          </button>
          <button type="button" onClick={() => setIsModalOpen(false)} disabled={uploading} className="mt-3 w-full inline-flex justify-center rounded-xl border border-slate-300 shadow-sm px-6 py-3 bg-white text-base font-medium text-slate-700 hover:bg-slate-50 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-indigo-500 sm:mt-0 sm:ml-3 sm:w-auto sm:text-sm transition-colors disabled:opacity-50">
            Скасувати
          </button>
        </div>
      </div>
    </div>
  )}
  {/* Delete Confirmation Modal */}
  {deleteConfirmId && (
    <div className="fixed inset-0 z-50 overflow-y-auto" aria-labelledby="modal-title" role="dialog" aria-modal="true">
      <div className="flex items-end justify-center min-h-screen pt-4 px-4 pb-20 text-center sm:block sm:p-0">
        <div className="fixed inset-0 bg-slate-900/50 transition-opacity" aria-hidden="true" onClick={() => setDeleteConfirmId(null)}></div>
        <span className="hidden sm:inline-block sm:align-middle sm:h-screen" aria-hidden="true">&#8203;</span>
        <div className="relative z-10 inline-block align-bottom bg-white rounded-3xl text-left overflow-hidden shadow-xl transform transition-all sm:my-8 sm:align-middle sm:max-w-md w-full">
          <div className="bg-white px-4 pt-5 pb-4 sm:p-6 sm:pb-4">
            <div className="sm:flex sm:items-start">
              <div className="mt-3 text-center sm:mt-0 sm:ml-4 sm:text-left">
```

```
                <h3 className="text-lg leading-6 font-bold text-slate-900" id="modal-title">
                  Підтвердження видалення
                </h3>
                <div className="mt-2">
                  <p className="text-sm text-slate-500">
                    Ви впевнені, що хочете видалити цей збір? Цю дію неможливо скасувати.
                  </p>
                </div>
              </div>
            </div>
          </div>
          <div className="bg-slate-50 px-4 py-3 sm:px-6 sm:flex sm:flex-row-reverse rounded-b-3xl">
            <button type="button" onClick={confirmDelete} className="w-full inline-flex justify-center rounded-xl border border-transparent shadow-sm px-4 py-2 bg-rose-600 text-base font-medium text-white hover:bg-rose-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-rose-500 sm:ml-3 sm:w-auto sm:text-sm transition-colors">
              Видалити
            </button>
            <button type="button" onClick={() => setDeleteConfirmId(null)} className="mt-3 w-full inline-flex justify-center rounded-xl border border-slate-300 shadow-sm px-4 py-2 bg-white text-base font-medium text-slate-700 hover:bg-slate-50 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-indigo-500 sm:mt-0 sm:ml-3 sm:w-auto sm:text-sm transition-colors">
              Скасувати
            </button>
          </div>
        </div>
```

ДОДАТОК В
(обов’язковий)

КЕРІВНИЦТВО КОРИСТУВАЧА

1. Призначення вебдодатку Вебдодаток «CharityFund» призначений для централізованого розміщення волонтерських зборів, безпечного проведення фінансових пожертв та автоматизованого обліку зібраних коштів. У системі передбачено два рівні доступу: публічний (для благодійників) та закритий (для волонтерів і адміністраторів).

2. Інструкція для благодійника (гостя системи)

Пошук збору: На головній сторінці перегляньте загальний каталог поточних кампаній. Для зручності ви можете скористатися панеллю фільтрації за категоріями (військові, медичні, гуманітарні потреби).

Здійснення донату через картку (Stripe): Оберіть збір та натисніть кнопку «Підтримати». У вікні, що з'явиться, введіть суму пожертви та реквізити вашої банківської картки. Натисніть «Оплатити». Після підтвердження банком шкала прогресу збору миттєво оновиться.

Здійснення донату через Monobank: На сторінці збору знайдіть згенерований QR-код. Наведіть на нього камеру вашого смартфона, щоб автоматично відкрити додаток банку, або натисніть кнопку «Відкрити в додатку» (якщо ви зайшли зі смартфона).

3. Інструкція для волонтера (адміністратора)

Авторизація: Натисніть кнопку «Увійти» у правому верхньому куті. Ви можете авторизуватися за допомогою електронної пошти та пароля або здійснити швидкий вхід через обліковий запис Google.

Створення нової кампанії: Після входу перейдіть до «Адмін-панелі». Натисніть «Створити збір». Заповніть обов'язкові поля: назва, детальний опис, цільова сума та додайте тематичне зображення. Вставте посилання на вашу

«Банку», якщо плануєте приймати альтернативні платежі. Натисніть «Опублікувати».

Керування транзакціями (Модерація): Якщо благодійник здійснив прямий переказ за реквізитами (IBAN), перейдіть у розділ «Транзакції». Звірте заявлену суму з випискою вашого банку. Натисніть кнопку «Підтвердити», щоб система зарахувала ці кошти до загального прогресу відповідного збору.

Аналітика: На головній сторінці Адмін-панелі ви можете відслідковувати загальну статистику залучених коштів по днях та аналізувати рівень активності за допомогою графіків.

ДОДАТОК Г

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

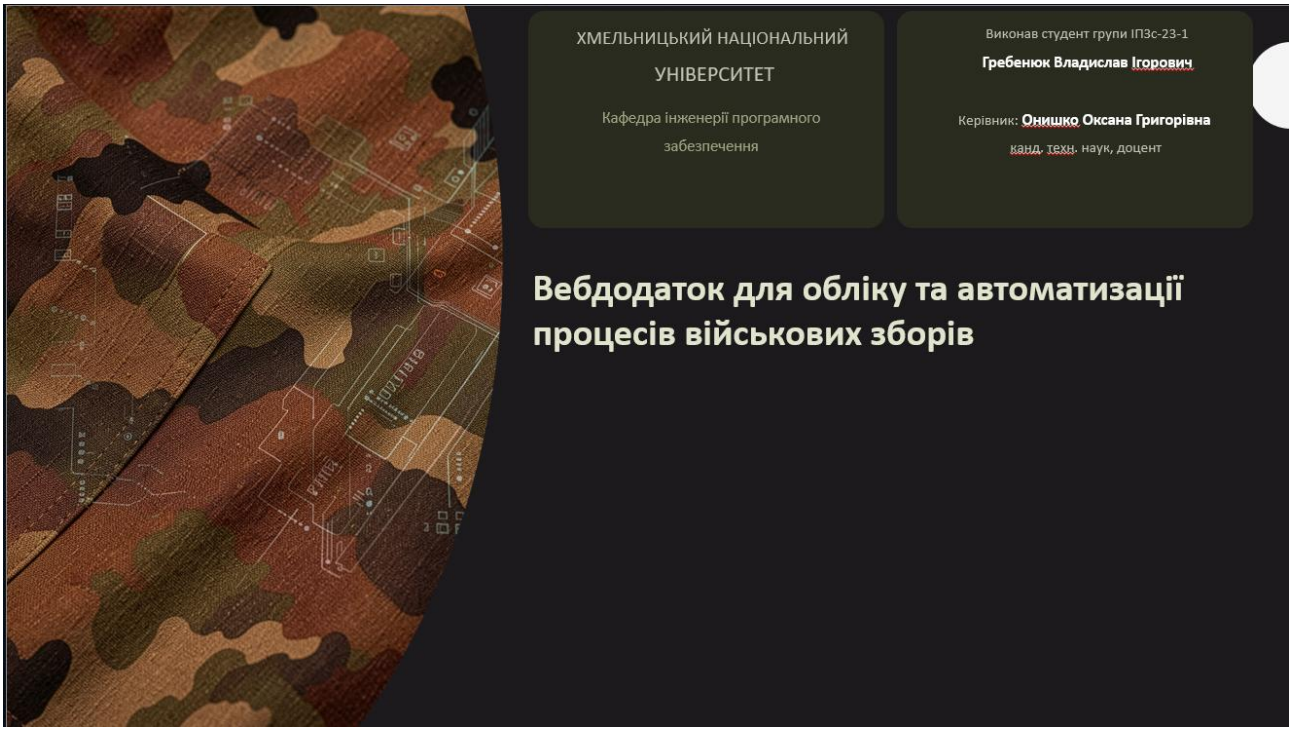


Рисунок Г.1 – Слайд 1

Рисунок Г.2 – Слайд 2

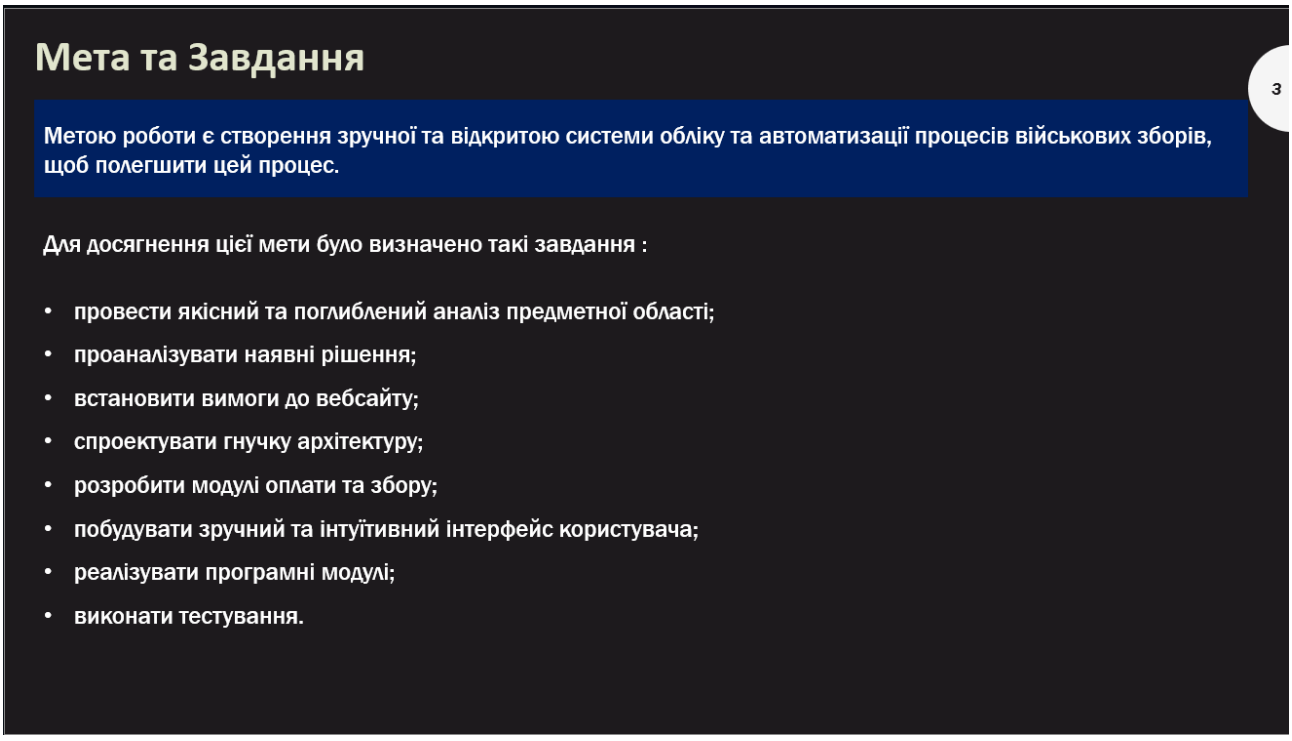


Рисунок Г.3 – Слайд 3

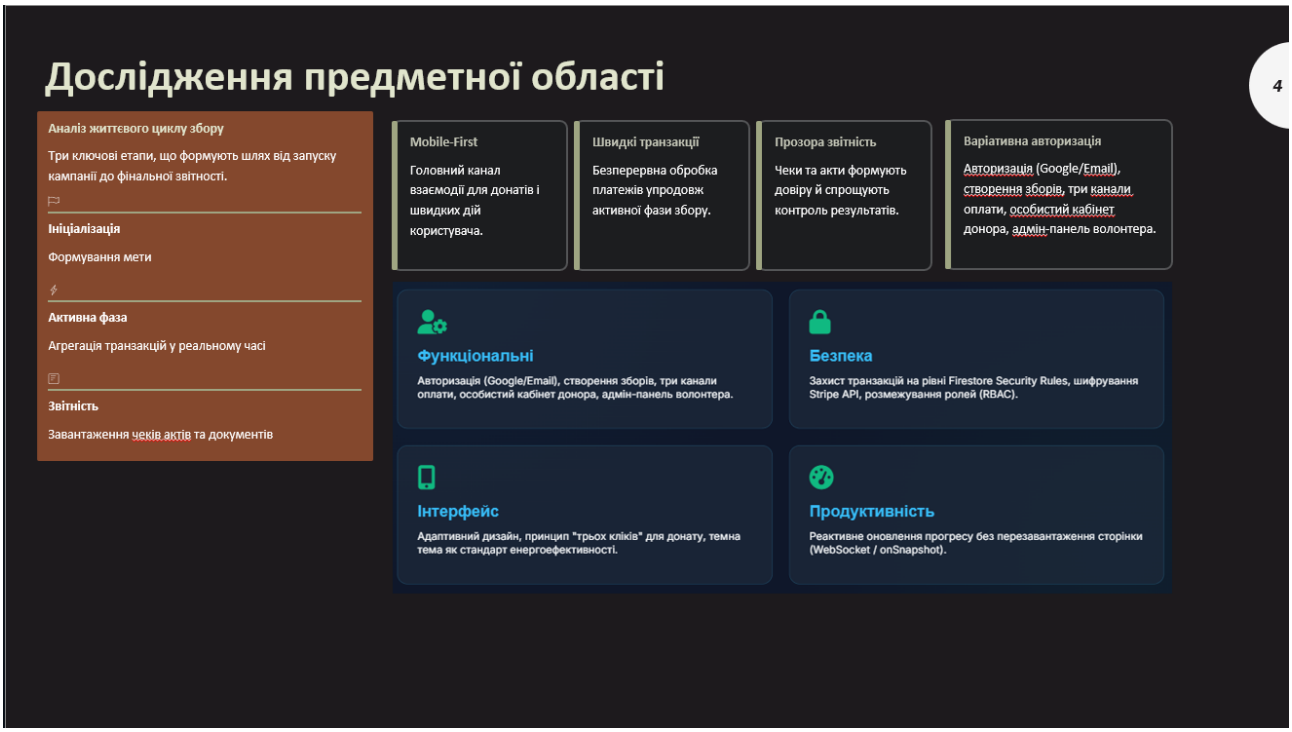


Рисунок Г.4 – Слайд 4

Рисунок Г.5 – Слайд 5

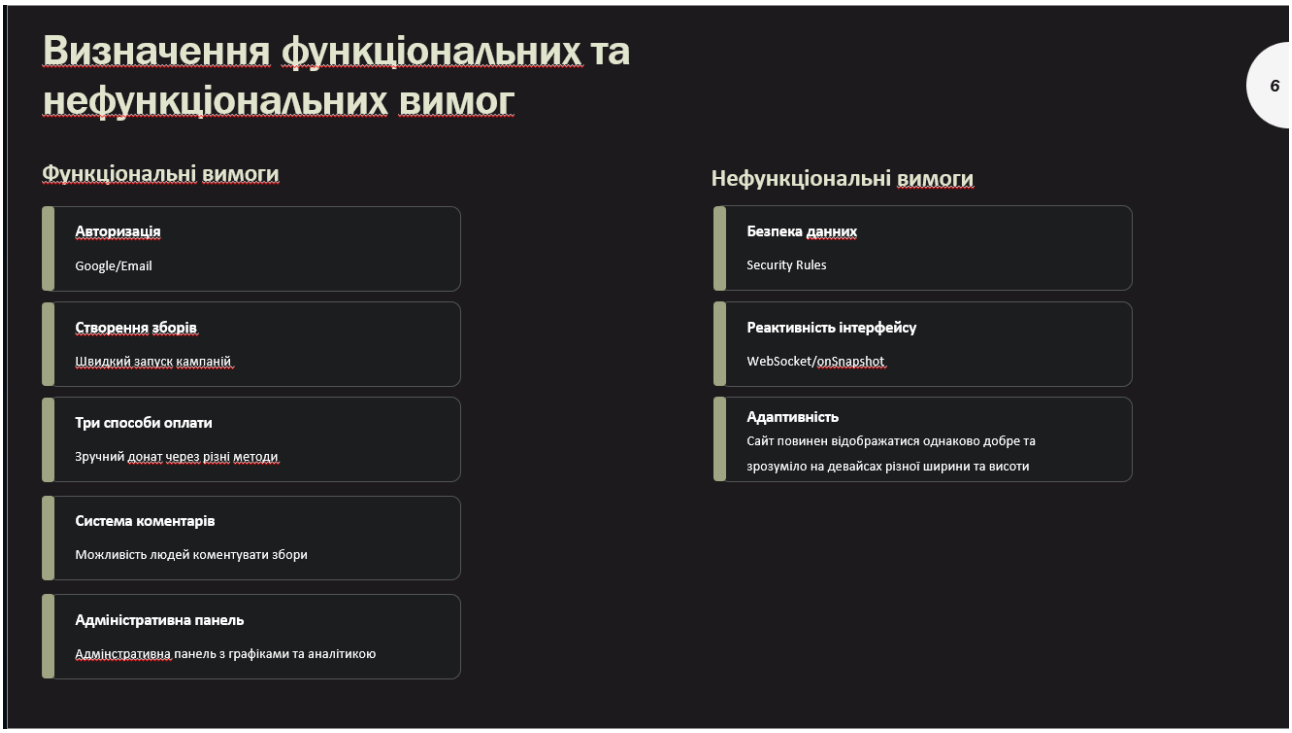


Рисунок Г.6 – Слайд 6

Рисунок Г.7 – Слайд 7

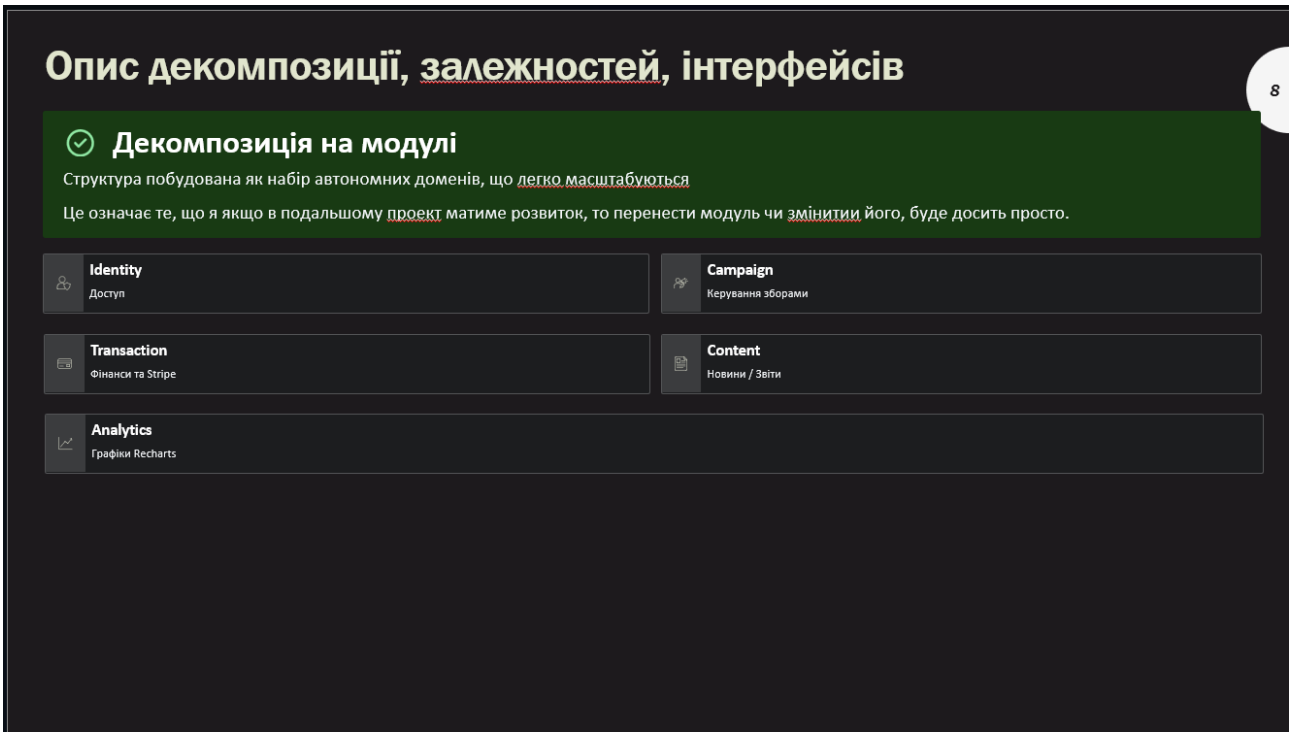


Рисунок Г.8 – Слайд 8

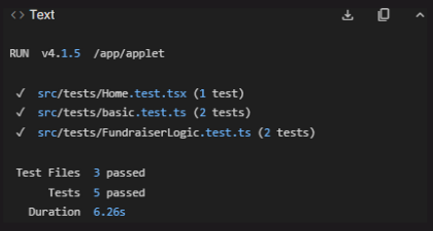
Рисунок Г.9 – Слайд 9

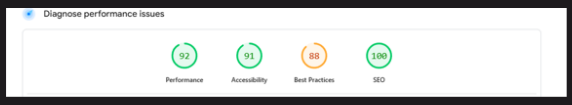
Рисунок Г.10 – Слайд 10

Рисунок Г.11 – Слайд 11

Тестування

12





За результатами проведеного комплексного тестування можна зробити висновок про повну працездатність розробленого вебдодатку та його відповідність усім функціональним і нефункціональним вимогам.
Усі ключові модулі системи (автентифікація користувачів, створення волонтерських кампаній, інтеграція платіжних шлюзів Stripe та Monobank) працюють коректно.
Механізми атомарного оновлення балансу в хмарній базі даних Firestore функціонують стабільно, забезпечуючи точний облік зібраних коштів у реальному часі.
Автоматизований аудит за допомогою Google Lighthouse показав відмінні результати (показники швидкодії, доступності та SEO на рівні 95–100%). Користувацький інтерфейс є повністю адаптивним і безпомилково відображається на мобільних пристроях.

Загальний висновок: Розроблена система є стабільною, безпечною, стійкою до навантажень і повністю готовою до практичної експлуатації.

Рисунок Г.12 – Слайд 12

Висновки

13

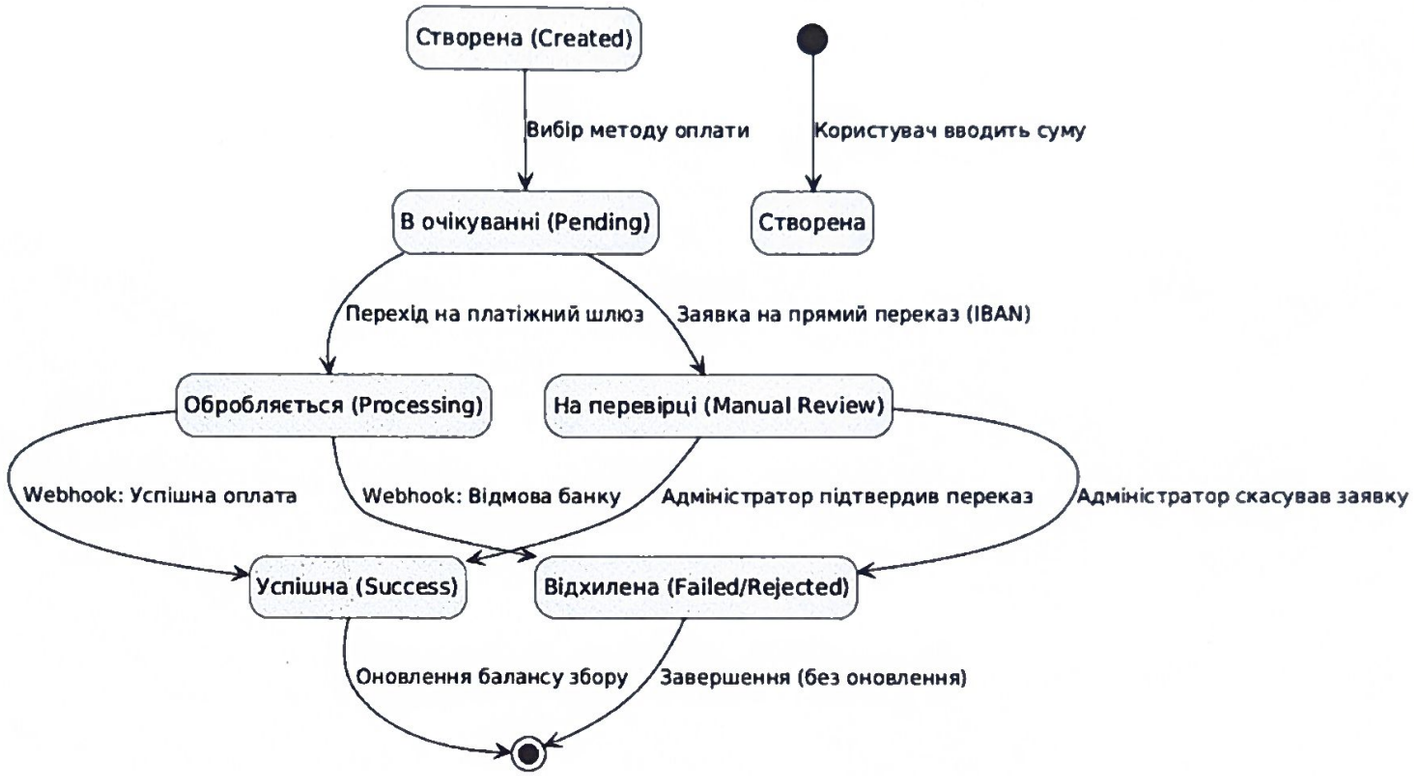
У результаті виконання кваліфікаційної роботи мету повністю досягнуто — розроблено готовий до експлуатації вебдодаток для автоматизації та обліку волонтерських зборів.
Основні результати:
Розробка системи: Створено масштабовану клієнт-серверну платформу на базі React та безсерверної архітектури Firebase.
Фінансова інтеграція: Успішно впроваджено модулі безпечного прийому платежів (Stripe, Monobank API) з автоматичним оновленням загального балансу в реальному часі.
Керування даними: Спроєктовано надійну базу даних Firestore із суворими правилами безпеки (Security Rules) та розроблено функціональну адмін-панель для волонтерів.

Рисунок Г.13 – Слайд 13

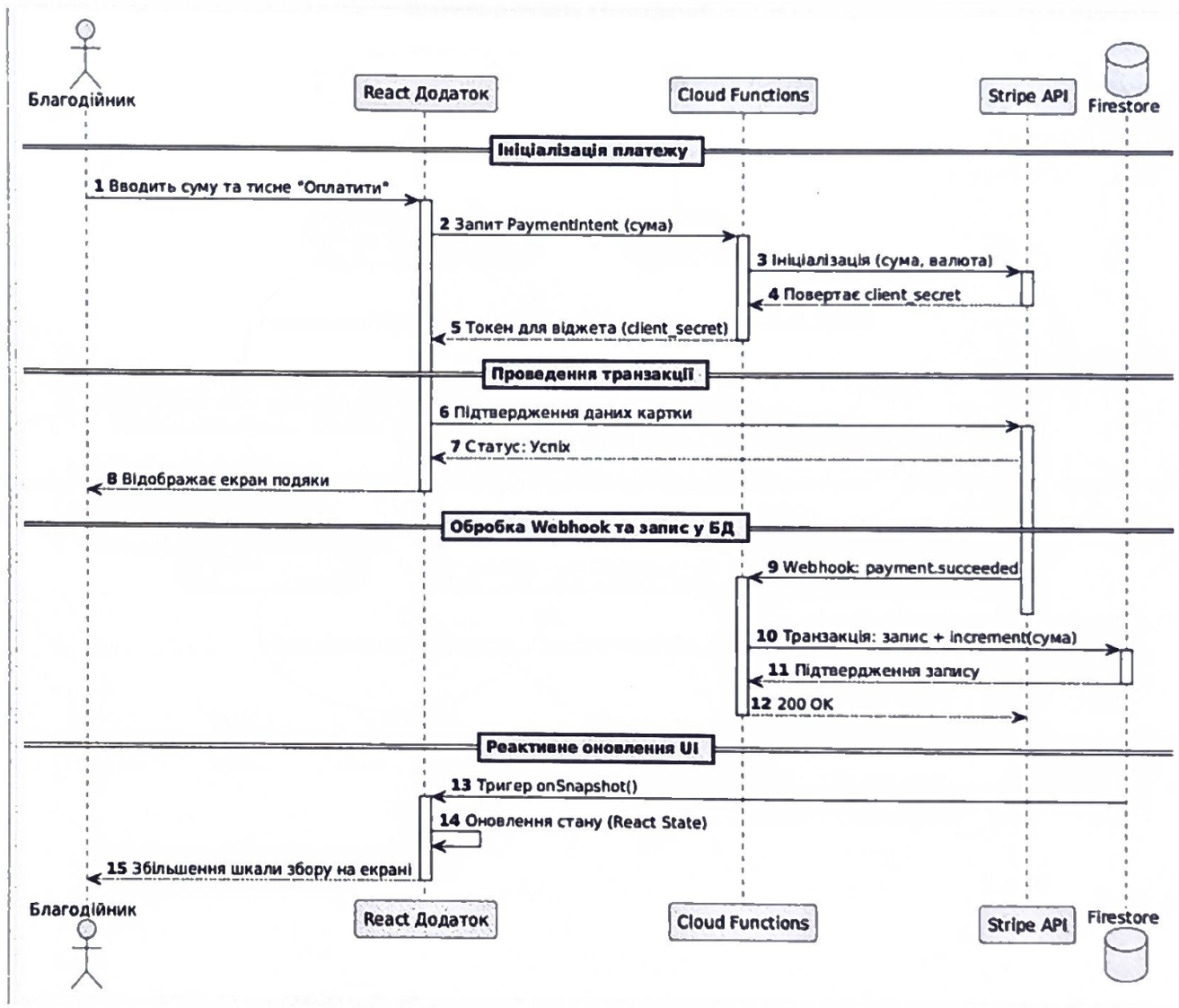
Рисунок Г.14 – Слайд 14

Графічна частина

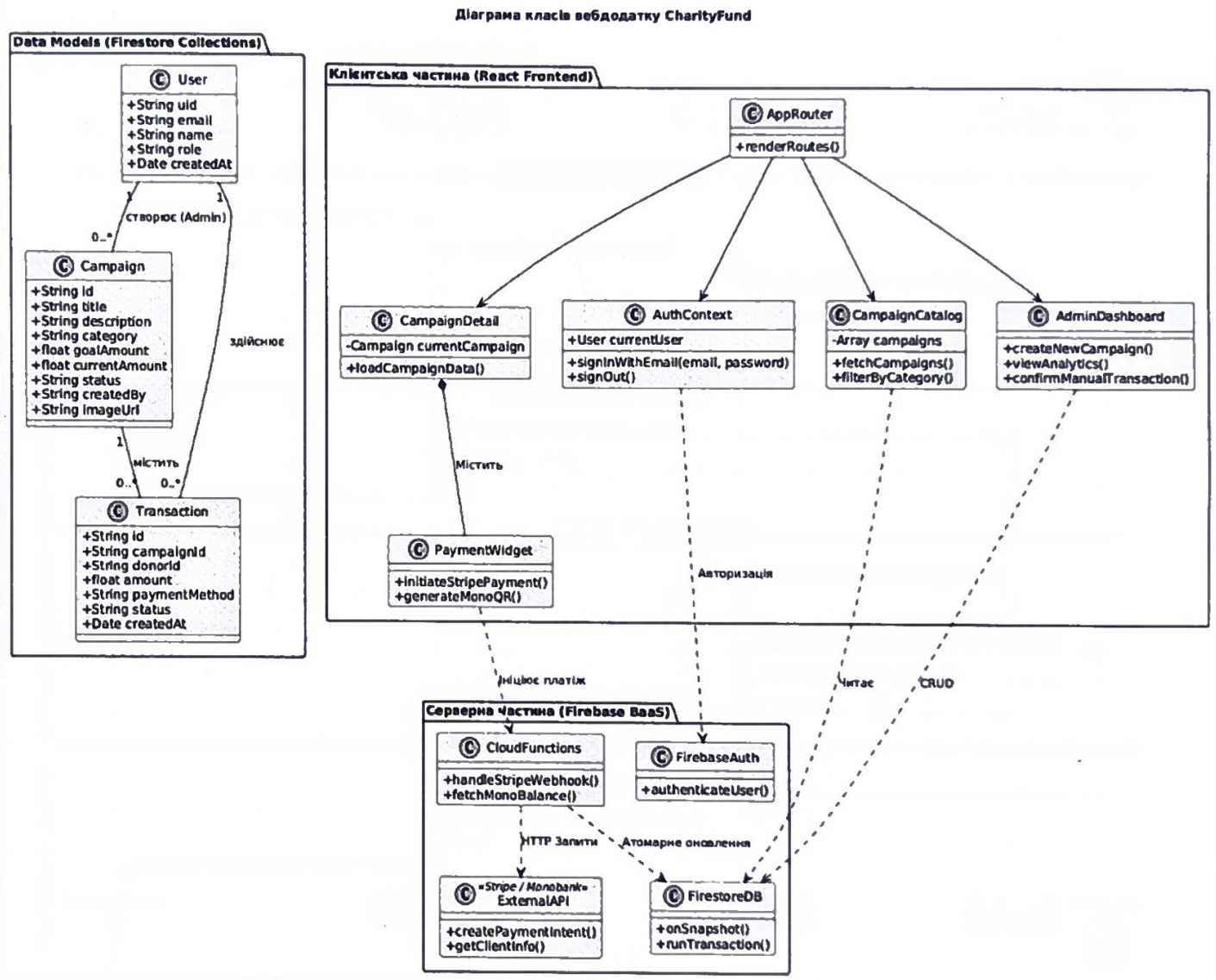
Діаграма станів транзакції



					КвРІПЗ. 230132.01.05.Е8		
Змн.	Лист	№ докум.	Підпис	Дата	Діаграма станів	Літ.	Маса / Масштаб
Розробив		Гребенюк В. І.		23.05			
Керівник		Онишко О. Г.				Арк. 3	Аркушів 3
Консульт.							
Н. Контр.		Форкун Ю. В.		23.05		ХНУ, ІПЗс–23–1	
Затверд.		Бодратюк Л.П.					



					КвРІПЗ. 230132.01.05.Е8		
Змн.	Лист	№ докум.	Підпис	Дата			
Розробив		Гребенюк В. І.		25.05	Діаграма послідовностей	Літ.	Маса / Масштаб
Керівник		Онишко О. Г.					
Консульт.						Арк. 2	Аркушів 3
Н. Контр.		Форкун Ю. В.		25.05		ХНУ, ІПЗс–23–1	
Затверд.		Бедратюк Л.П.					



					КвРІПЗ. 230132.01.05.Е8		
Змн.	Лист	№ докум.	Підпис	Дата			
Розробив		Гребенюк В. І.		26.05	Діаграма класів	Літ.	Маса / Масштаб
Керівник		Онишко О. Г.					
Консульт.						Арк. 1	Аркушів 3
Н. Контр.		Форкун Ю. В.		26.05		ХНУ, ІПЗс–23–1	
Затверд.		Бедратюк Л.П.					

Супровідні документи

Fri May 22 12:48:34 EEST 2026, Форкун Юрій Вікторович, Хмельницький національний університет, ХНУ

Anti-Plagiarism (http://ap.km.ua) v-16.718

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: UA, US, RU. **Помилок в документах: 18%**

	Документ		Сумарний збіг по Базі Даних	
ID: 271999 Назва: БКР Вебдодаток для обліку та автоматизації процесів військових зборів Додано в БД: 2026-05-22 Автора: Владислав ГРЕБЕНЮК Керівники: канд. пед. наук, доцент Оксана ОНИШКО Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	160591	1198	5761 (4%)	76 (6%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

Завідувачу кафедри
інженерії програмного забезпечення
проф. Бедратюку Л. П.
студента групи ІПЗс-23-1
Гребенюка В.І
Прізвище, ініціали

ЗАЯВА

Прошу закріпити за мною тему кваліфікаційної роботи освітнього ступеня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення»: Вебдодаток для обліку та автоматизації процесів військових

(керівник роботи – Онишко Оксана Григорівна)
Прізвище, ім'я, по батькові

25.05.2026
Дата

Підпис студента

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів перевірки роботи, продукованими програмно-технічним засобом (ами), на наявність текстових збігів.
Назва кваліфікаційної роботи: «Вебдодаток для обліку та автоматизації процесів військових хборів»
Автор: Гребенюк Владислав Ігорович
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Онишко Оксана Григорівна, кандидат педагогічних наук, доцент
Після аналізу звіту/звітів зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається до захисту.	**відповідає**
2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована.	
3	Виявлені запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Виявлені запозичення частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнуті. Робота може бути допущена до захисту після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені у роботі, є законними і не є плагіатом, оскільки:

1. у тексті кваліфікаційної роботи системою перевірки на плагіат Anti-Plagiarism виявлено схожість з деякими документами у частині загальновживаних обов'язкових словосполучень у стандартних бланках, у структурі змісту, назвах назвах розділів/підрозділів, рамках форм, у назвах та URL-адресах публікацій переліку джерел посилання;
2. запозичення, виявлені у тексті роботи, є фрагментарними. Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає 1.0% з одного джерела. Загальна сумарна подібність у базі даних складає 4% за символами та 6% за лексемами. Крім того, за результатами додаткового аналізу коефіцієнт подібності 1 становить 11.67%, коефіцієнт подібності 2 – 2.59%. Виявлено мікропробіли, не виявлено зайвих білих знаків або маніпуляцій з інтервалами. З урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 3.06.2026

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Оксана ОНИШКО

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Гребенюка Владислава Ігоровича
факультет ІТ, ІІІ курс, група ІПЗс-23-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

25.05.2026
дата

підпис

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ДЕКЛАРАЦІЯ УЧАСНИКА ОСВІТНЬОГО ПРОЦЕСУ

щодо дотримання академічної доброчесності

Цією декларацією я, Гребенюк Владислав Ігорович,

студент III курсу спеціальності 121 – Інженерія програмного забезпечення, група ІПЗс-23-1

здобувач вищої освіти (шифр та назва спец-ті, курс, академічна група)

підтверджую, що ознайомився (-лась) з Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті та Кодексом академічної доброчесності і **зобов'язуюсь** дотримуватися їх вимог під час освітнього процесу, проведення наукової діяльності, виконання організаційно-адміністративних функцій тощо.

Усвідомлюю, що у разі порушення мною принципів академічної доброчесності нестиму відповідальність перед академічною спільнотою ХНУ згідно з нормами, визначеними Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, законодавства України.

2 вересня 2026 р.

Підпис

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Гребенюк Владислав Ігорович

Тема Вебдодаток для обліку та автоматизації процесів військових зборів

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3; кількість сторінок записки 82
1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі досліджено і проаналізовано предметну область, визначено усі функціональні та нефункціональні вимоги. Був проведений аналіз існуючих програм на ринку, розглянуто їх переваги і недоліки, та доведено актуальність розробки нового програмного забезпечення. Розглянуто інструменти для реалізації спроектованих рішень, в результаті чого створено програмне забезпечення. Також було проведено тестування програми, за результатами якого доведено, що розроблене програмне забезпечення працює коректно та готове до експуатації.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі доведено актуальність теми, визначено мету та завдання дипломного проектування. У першому розділі проведено аналіз предметної області, розглянуто існуючі рішення та визначені функціональні і нефункціональні вимоги до розроблюваного програмного забезпечення. У другому розділі проведено аналіз сучасних архітектур, розглянуто їх переваги і недоліки та визначено, що система буде відповідати монолітній архітектурі та моделі клієнт-сервер. У третьому розділі підготовлено всі залежності для написання коду та виконано практичну розробку програмних модулів і описано їх особливості, в результаті чого створено програмний продукт. Також у цьому розділі виконано модульне тестування системи та проведено його у відповідності до функціональних вимог, в результаті було підтверджено коректну роботу програми.

4. Позитивні сторони роботи Тематика кваліфікаційної роботи є актуальною, оскільки на сьогодні в Україні тема військових зборів не є достатньо розвинутими та не мають достатньої кількості функціональних можливостей. Також було застосовано новітні технології для побудови програмного продукту та актуальні архітектурні рішеня.

5. Негативні сторони роботи У роботі всі збори формуються вручну адміністратором сайту, що не є правильним. Також відсутня можливість конвертації валюти під час донату

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки. Матеріал пояснювальної записки структурований, послідовний, чіткий та простий, що дозволяє чітко зрозуміти викладений матеріал у рамках тематики проектування. Графічний матеріал дає можливість наочно побачити деталі проектування системи.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «відмінно».

РЕЦЕНЗЕНТ кандидат технологічних наук, доцент
Кожухар Андрій Олександрович

“_____” ________________________ 2026 р. ____________________
(підпис)

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Владислав ГРЕБЕНЮК

Співавтор:

Назва: Вебдодаток для обліку та автоматизації процесів військових зборів

Науковий керівник: канд. пед. наук, доцент Оксана ОНИШКО

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:11.67%

Коефіцієнт подібності 2:2.59%

Мікропробіли: 11

Заміна букв: 2

Інтервали: 19

Білі знаки: 0

Дата створення звіту: 2026-05-25 02:00:42.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

Дата

25.05.2026

експерт