НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Веб-сервіс для донатів

Виконав студент IV курсу, групи ІТ-91
(шифр групи)

Тонкошкур Олег Русланович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к.т.н., доц., Ліщук К.І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації ст. викл. Головченко М.М.. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доц. каф.ІСТ, к.т.н., доц. Писаренко А. В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ –2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення комп'ютерних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Тонкошкур Олег Русланович
(прізвище, ім'я, по батькові)

1. Тема проєкту Веб-сервіс для донатів
керівник проєкту Ліщук Катерина Ігорівна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «31»травня 2023 р. № 2101-с

2. Термін подання студентом проєкту « 17 » червня 2023 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Аналіз вимог до програмного забезпечення: загальні положення, змістовний опис і аналіз предметної області, аналіз існуючих технологій та успішних ІТ-проєктів, аналіз вимог до програмного забезпечення, постановка задачі.
2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, архітектура програмного забезпечення, конструювання програмного забезпечення, аналіз безпеки даних.
3) Аналіз якості та тестування програмного забезпечення: аналіз якості ПЗ, Опис процесів тестування, опис контрольного прикладу.
4) Впровадження та супровід програмного забезпечення: розгортання програмного забезпечення.
5) Технологічний розділ: керівництво користувача, методика випробувань програмного продукту.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема бази даних

3) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «10» березня 2023 року _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	10.03.2023	
2	Аналіз існуючих методів розв’язання задачі	18.03.2023	
3	Постановка та формалізація задачі	26.03.2023	
4	Розробка інформаційного забезпечення	05.04.2023	
5	Алгоритмізація задачі	12.04.2023	
6	Обґрунтування вибору використаних технічних засобів	20.04.2023	
7	Розробка програмного забезпечення	06.05.2023	
8	Налагодження програми	13.05.2023	
9	Виконання графічних документів	18.05.2023	
10	Оформлення пояснювальної записки	02.06.2023	
11	Подання ДП на попередній захист	02.06.2023	
12	Подання ДП рецензенту	12.06.2023	
13	Подання ДП на основний захист	17.06.2023	

Студент _____ Олег ТОНКОШКУР
(підпис) (ініціали, прізвище)

Керівник _____ Катерина ЛІЩУК
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 38 таблиць, 27 рисунків та 10 джерел – загалом 72 сторінки.

Дипломний проєкт присвячений архітектурному рішенню веб-сервісу для донатів

Мета створення архітектури програмного забезпечення веб-сервісу для донатів та імплементація його у повноцінне програмне забезпечення, що дозволить забезпечити ефективну, прозору та безпечну систему збору та розподілу коштів.

Об'єкт дослідження: веб-сервіс для збору пожертвувань. Об'єкт дослідження включає усі аспекти, компоненти та функціональність, пов'язані з розробкою та експлуатацією цього веб-сервісу.

Предмет дослідження: розробка та реалізація веб-сервісу для збору пожертвувань. Предмет дослідження охоплює концепцію, функціональність, технології, архітектуру та процеси, пов'язані з створенням та ефективним функціонуванням цього сервісу.

У першому розділі проаналізовано вимоги до програмного забезпечення, розроблено функціональні та нефункціональні вимоги до системи.

Другий розділ присвячений моделюванню та конструюванню програмного забезпечення. Тут описані деталі архітектури та реалізації системи.

В третьому описано аналіз якості розробленого програмного рішення та проведено тестування програмного забезпечення.

Останній, четвертий розділ, присвячено впровадженню та супроводу програмного забезпечення.

КЛЮЧОВІ СЛОВА: КЛІЄНТ-СЕРВЕР, REST, БАЗА ДАНИХ, API, РЕПОЗИТОРІЙ, ПАТТЕРН

ABSTRACT

The explanatory note of the diploma project consists of four sections, contains 38 tables, 27 figures and 10 sources – in total 72 pages.

The purpose of the diploma project is an architectural solution for delivery service.

The goal is to create the architecture of the software for a donation web service and implement it into a fully functional software solution that will ensure an efficient, transparent, and secure system for collecting and distributing funds.

Research object: the sphere of goods and services delivery, the needs of enterprise management and end users.

Research subject: the development of architecture and software implementation of a delivery service, which includes the analysis and selection of optimal architectural approaches and technologies.

The first chapter analyzes the requirements for the software and develops functional and non-functional requirements for the system.

The second chapter is dedicated to software modeling and design. It describes the details of the system's architecture and implementation.

The third chapter describes the analysis of the quality of the developed software solution and conducts software testing.

The final, fourth chapter is devoted to the deployment and maintenance of the software.

KEYWORDS: API, MICROSERVICES, CLIENT-SERVER, REST, SAAS, MULTITENANCY, DATABASE.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІТ-9126.045440.01.91	Технічне завдання	11	
3	А4	КПІ.ІТ-9126.045440.02.81	Пояснювальна записка	72	
4	А4	КПІ.ІТ-9126.045440.03.12	Текст програми	68	
5	А4	КПІ.ІТ-9126.045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІТ-9126.045440.05.34	Керівництво користувача	14	
7	А3	КПІ.ІТ-9126.045440.06.99	Схема структурна варіантів	1	
			використань		
8	А3	КПІ.ІТ-9123.045440.07.99	Схема бази даних	1	
9	А3	КПІ.ІТ-9123.045440.08.99	Креслення вигляду екранних форм	1	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІТ-9126.045440.00.90		
Розроб.		Тонкошкур О. Р.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Ліщук К. І.					
Н.контр.		Головченко М. М.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІТ-91	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

Веб-сервіс для донатів

Технічне завдання

КПІ. ІТ-9126.045440.01.91

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Катерина ЛІЩУК

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Олег ТОНКОШКУР

Київ – 2023

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Веб-сервіс для донатів

Галузь застосування:

Наведене технічне завдання поширюється на розробку програмного забезпечення для збору пожертвувань та їх ефективному розподілі між благодійними організаціями та проектами та призначене для загального користування.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки веб-сервісу для донатів є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для створення веб-сервісу для донатів, який надасть можливість користувачам здійснювати благодійні внески та пожертвування на підтримку різних благодійних організацій та проектів.

Метою розробки є створення архітектури програмного забезпечення веб-сервісу для донатів та імплементація його у повноцінне програмне забезпечення, що дозволить забезпечити ефективну, прозору та безпечну систему збору та розподілу коштів.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

- реєстрація користувача в системі:
  - реєстрація повинна здійснюватися через Google аккаунт.
- вхід користувача в систему:
  - вхід повинен здійснюватися через Google аккаунт.
- зміна унікального імені для користувача:
  - унікальне ім’я не може повторюватися.
- перегляд статусів цілей для донатів:
  - повинна надаватися інформація про назву, опис, загальну та поточну суму збору цілі.
- створення цілей для донатів:
  - ціль повинна містити такі дані: назва, опис, номер карти для виводу коштів та загальна сума збору.
- редагування цілей для донатів:
  - редагуватися повинні такі дані: опис та загальна сума збору.
- проведення донатів:
  - донат повиннен містити такі дані: ім’я автору донату, повідомленн донату та сума донату;
  - донат повинен здійснюватися в гривні.
- перегляд інформації про отримані донати:
  - повинна надаватися інформація про суму донату, дату донату, ім’я та текст повідомлення які залишив автор донату.
- вивід коштів:
  - вивід коштів повинен бути доступний лише на ті картки які були прикріплені до цілей для донатів.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i5;
– об‘єм ОЗП: 4 Гб;
– підключення до мережі Інтернет зі швидкістю від 20 мегабіт.

Рекомендована конфігурація технічних засобів:

– тип процесору: Intel Core i7;
– об‘єм ОЗП: 8 Гб;
– підключення до мережі Інтернет зі швидкістю від 100 мегабіт.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням усіх операційних систем які підтримують підключення до мережі інтернет, та можуть надати доступ до веб-сервісу через браузер.

4.5.1 Вимоги до вихідних даних

Результати роботи серверної частини повинні бути представлені в форматі http response, який за необхідності буде зберігати в собі дані в форматі json.

4.5.2 Вимоги до мови розробки

Розробку виконати на мові програмування C#.

4.5.3 Вимоги до середовища розробки

Розробку виконати на платформі Rider.

4.5.4 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді текстових файлів у форматі .cs для самого коду системи, у форматі .json для конфігураційних файлів та .csproj для файлів-проектів.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Розгорнути клієнтський та серверний застосунок в Microsoft Azure.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;

– технічне завдання;

– текст програми;

– програма та методика тестування;

– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання;

– схема бази даних;

– креслення вигляду екранних форм.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	27.02	
2.	Розробка технічного завдання	09.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	25.03	Схема варіантів використань
4.	Проектування структури програмного забезпечення, проектування бази даних	7.04	Схема бази даних
5.	Програмна реалізація програмного забезпечення	20.04	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	13.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	01.06	Пояснювальна записка
8.	Розробка матеріалів графічної частини проекту	20.04	Графічний матеріал проекту

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: Веб-сервіс для донатів

КПІ.ІТ-9126.045440.02.81

Київ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ІТ	–	Інформаційні технології
БД	–	База даних

ВСТУП

У сучасному світі, де технології стають неодмінною складовою нашого повсякденного життя, Інтернет здобуває все більшу популярність як важливий ресурс для отримання інформації, спілкування та здійснення різних дій. За допомогою Інтернету люди можуть швидко та зручно звертатися до інших людей з проханням про допомогу або надавати її самостійно.

Одним із найпоширеніших способів надання фінансової допомоги є система донатів. Донати, або внески, дозволяють людям пожертвувати гроші на підтримку різних благодійних, громадських та культурних проектів. Однак, в цьому процесі існують певні перешкоди, такі як недостатня інформаційна доступність, складнощі з організацією та моніторингом зібраних коштів, а також потенційна недостовірність деяких донатів.

Метою даної дипломної роботи є розробка та реалізація веб-сервісу для донатів, який буде вирішувати зазначені проблеми та забезпечувати ефективну, прозору та безпечну систему збору та розподілу коштів. Веб-сервіс буде базуватися на передових технологіях розробки веб-додатків та забезпечувати зручний інтерфейс для користувачів, що дозволить їм легко здійснювати донати та переглядати статистику щодо зібраних коштів.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Веб-сервіси для донатів є важливим інструментом для збору коштів на різноманітні цілі. Вони можуть бути використані благодійними організаціями, громадськими проектами, особистими ініціативами тощо. Такі сервіси забезпечують простий та безпечний механізм, який дозволяє зробити пожертвування за кілька клацань миші.

1.2 Змістовний опис і аналіз предметної області

У випадку веб-сервісу для донатів, предметна область полягає в зборі пожертвувань та їх ефективному розподілі між благодійними організаціями та проектами.

Процес імплементації предметної області у ІТ, у даному випадку, передбачає створення зручного та безпечного веб-сервісу, який надає можливість користувачам зробити пожертвування та благодійним організаціям ефективно керувати отриманими коштами.

Проте, існують певні недоліки в поточному стані речей з предметною областю у сфері ІТ. Деякі з них включають:

Низька доступність та складність процесу пожертвування: існуючі сервіси можуть бути складними для користувачів, що обмежує їх можливість зробити пожертвування.

Відсутність аналітики та звітності: інформація про пожертвування та їх використання не завжди є доступною, що ускладнює оцінку ефективності благодійних організацій та проектів.

Для покращення ситуації з розробками у сфері ІТ для веб-сервісів донатів, можливі шляхи включають:

Вдосконалення користувацького досвіду: спрощення та зручність процесу пожертвування, інтуїтивний інтерфейс та можливість швидкого здійснення донату.

Розробка аналітики та звітності: збір та аналіз даних про пожертвування, створення звітів для благодійних організацій та донорів, що дозволить ефективно використовувати отримані кошти.

У рамках дипломного проєкту обраний підхід полягає у розробці веб-сервісу для донатів, який буде забезпечувати зручний та безпечний процес пожертвування, з функціональністю аналітики та звітності для кращого керування пожертвуваннями та ефективним використанням отриманих коштів.

1.3 Аналіз існуючих технологій та успішних ІТ-проєктів

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації веб-сервісу для донатів. Далі будуть розглянуті допоміжні програмні засоби, засоби розробки та готові програмні рішення.

1.3.1 Аналіз відомих алгоритмічних та технічних рішень

Система позиціонується як веб-сервіс, вона складатиметься з серверного і клієнтського застосунків Серед архітектурних рішень для такої системи найкраще підходить варіант трирівневої клієнт-серверної архітектури [1].

Вибір клієнт-серверного підходу зумовлений тим, що застосунок може розширятись в майбутньому, наприклад створення різних версії клієнтів або створення мобільного застосунку.

1.3.2 Аналіз допоміжних програмних засобів та засобів розробки

Для розробки такої системи добре підходить мова програмування C#. Цей вибір обумовлений тим, що останні роки C# досить швидко розвивається і з кожним роком надає розробникам все більше і більше можливостей, також це конкурентна мова і застосунок буде легко підтримувати на пізніших стадіях розробки. Також вона глибоко інтегрована з cloud-середовищем «Azure», що дозволить легко інтегруватись та розгортувати застосунок в клауді.

Через глибоку інтеграцію з azure та C# базою даних було обрано MSSQL Server.

Досить важливою частиною системи є інтеграція з платіжною системою. Серед варіантів було обрано Fondy UA.

1.3.3 Аналіз відомих програмних продуктів

Існуючі сервіси з донатів фокусуються на підтримці авторів якогось контенту, а не на благодійні організації і волонтерів. У них користувач не може самостійно переглянути всі цілі для донатів іншого користувача. Коли користувач хоче зробити донат, то він автоматично робиться на першу активну ціль. Також присутні ліміти на загальну суму донатів за день і на максимальну суму окремого донату.

Для порівняння дипломного проекту (Donation App Service – далі DAP) з аналогами можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогом “Donatello”

Функціонал	DAP	Donatello	Пояснення
Можливість здійснити донат без входу в систему	Так	Так	
Можливість переглянути всі цілі для донатів іншого користувача	Так	Ні	Donatello показує лише першу активну ціль
Надає користувачам можливість зробити донат на довільну ціль користувача	Так	Ні	Donatello автоматично обирає першу активу ціль

Продовження таблиці 1.1

Надає користувачу можливість налаштувати систему сповіщень	Ні	Так	Для DAP це додатковий функціонал який може бути доданий пізніше
Можливість створити запит на вивід коштів	Так	Так	
Відсутність лімітів на максимальну суму донатів за день і суму окремого донату	Так	Ні	Donatello має статусу профілі від який залежать ліміти

1.4 Аналіз вимог до програмного забезпечення

Основною задачею системи є надання функціоналу для створення цілей для донатів і можливості робити донати на актуальні цілі.

Всі функції програмного забезпечення можна побачити на рисунку 1.1.

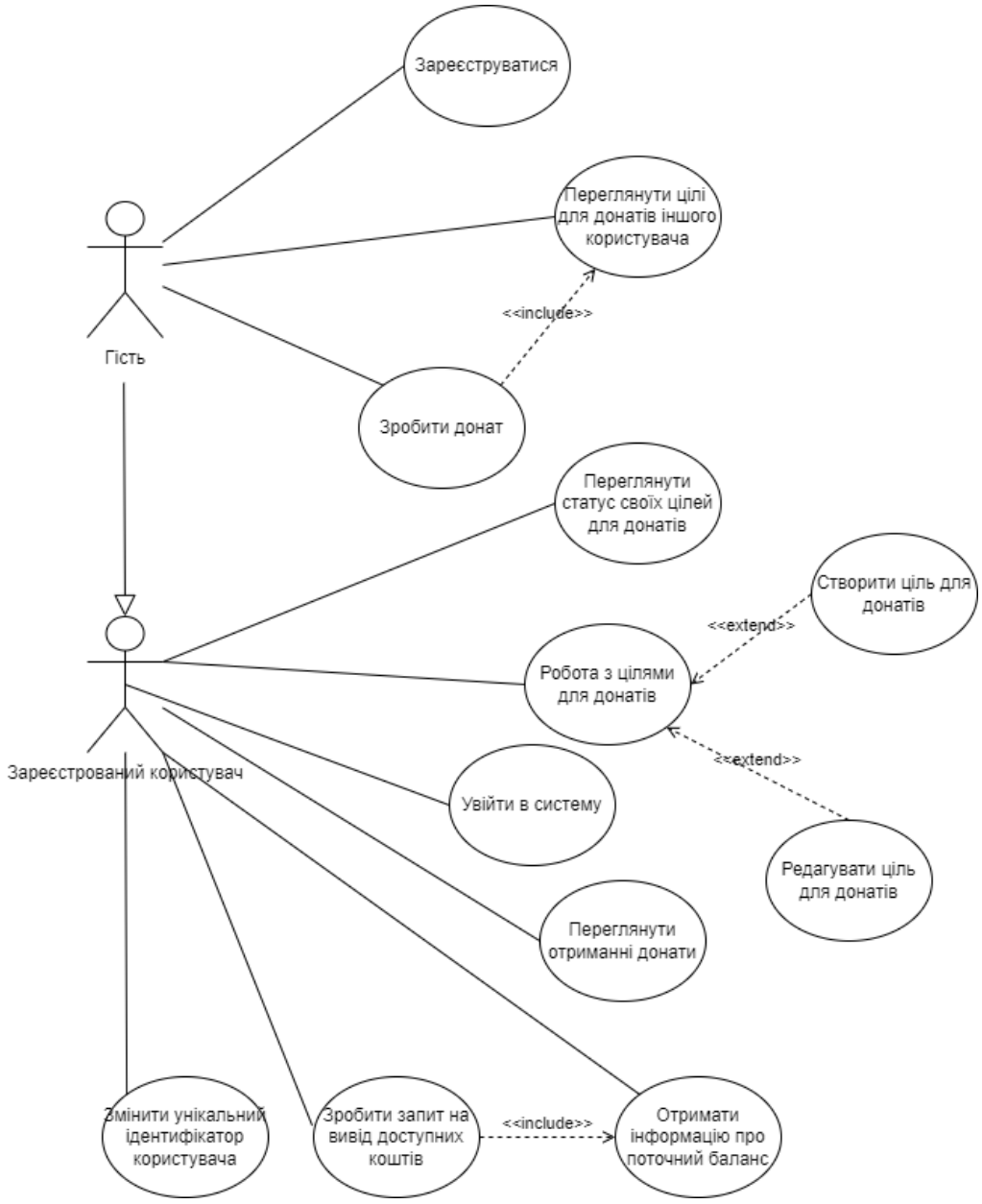


Рисунок 1.1 – Діаграма варіантів використання

В таблицях 1.2 - 1.12 наведені варіанти використання системи.

Таблиця 1.2 - Варіант використання UC-1

Use case name	Увійти в систему
Use case ID	UC-01
Goals	Авторизація користувача в системі
Actors	Користувач системи
Trigger	Користувач хоче авторизуватись в системі

Продовження таблиці 1.2

Pre-conditions	-
Flow of Events	Користувач натискає кнопку увійти/зареєструватися і у спливаючому вікні обирає гугл аккаунт для входу . Користувач перенаправляється на основну сторінку
Extension	-
Post-Condition	Користувач переходить до основної сторінки

Таблиця 1.3 - Варіант використання UC-2

Use case name	Зареєструватися в системі
Use case ID	UC-02
Goals	Реєстрація нового користувача в системі
Actors	Користувач системи
Trigger	Новий користувач бажає приєднатися до системи
Pre-conditions	-
Flow of Events	Користувач натискає кнопку увійти/зареєструватися і у спливаючому вікні обирає гугл аккаунт для входу . Користувач додається в систему і перенаправляється на основну сторінку
Extension	-
Post-Condition	Користувач успішно додається в систему і переходить до основної сторінки

Таблиця 1.4 - Варіант використання UC-3

Use case name	Змінити унікальний ідентифікатор користувача
Use case ID	UC-03
Goals	Надати ідентифікатору якийсь зміст
Actors	Користувач системи
Trigger	Користувач хоче змінити свій унікальний ідентифікатор

Продовження таблиці 1.4

Pre-conditions	Користувач авторизований в системі
Flow of Events	Користувач переходить на сторінку для редагування персональної інформації
Extension	-
Post-Condition	Користувач бачить оновлені дані

Таблиця 1.5 - Варіант використання UC-4

Use case name	Переглянути статус своїх цілей для донатів
Use case ID	UC-04
Goals	Отримання інформації про статус своїх цілей для донатів
Actors	Користувач системи
Trigger	Користувач хоче бачити поточну кількість донатів по своїх цілях
Pre-conditions	Користувач авторизований в системі Користувач відкрив сторінку з своїми цілями для донатів
Flow of Events	Користувач переходить на сторінку з своїми цілями для донатів
Extension	-
Post-Condition	-

Таблиця 1.6 - Варіант використання UC-5

Use case name	Створити ціль для донатів
Use case ID	UC-05
Goals	Створення цілі для донатів
Actors	Користувач системи
Trigger	Користувач хоче створити певну ціль для донатів

Продовження таблиці 1.6

Pre-conditions	Користувач авторизований в системі Користувач відкрив сторінку з даними про свої цілі для донатів
Flow of Events	Користувач заповнює відповідні дані про ціль і натискає кнопку “Створити”. Сторінка з даними про цілі користувача оновлюється
Extension	-
Post-Condition	Сторінка з даними про цілі користувача оновлюється

Таблиця 1.7 - Варіант використання UC-6

Use case name	Редагувати ціль для донатів
Use case ID	UC-06
Goals	Редагування даних певної цілі для донатів
Actors	Користувач системи
Trigger	Користувач хоче редагувати дані певної цілі для донатів
Pre-conditions	Користувач авторизований в системі Користувач відкрив сторінку з даними про свої цілі для донатів Ціль для донатів створена
Flow of Events	Менеджер натискає кнопку “Редагувати” біля відповідної цілі і вводить нові дані у відповідні поля . Сторінка з даними про цілі користувача оновлюється
Extension	-
Post-Condition	Сторінка з даними про цілі користувача оновлюється

Таблиця 1.8 - Варіант використання UC-7

Use case name	Переглянути цілі для донатів іншого користувача
Use case ID	UC-07
Goals	Переглянути цілі для донатів іншого користувача, щоб потім зробити донат

Продовження таблиці 1.8

Actors	Користувач системи
Trigger	Користувач хоче Переглянути цілі для донатів іншого користувача
Pre-conditions	-
Flow of Events	Користувач відкриває сторінку з цілями для донатів іншого користувача
Extension	-
Post-Condition	-

Таблиця 1.9 - Варіант використання UC-8

Use case name	Зробити донат
Use case ID	UC-08
Goals	Відправити кошти на певну ціль
Actors	Користувач системи
Trigger	Користувач хоче відправити свої кошти на певну ціль
Pre-conditions	Користувач відкривав сторінку з цілями для донатів іншого користувача
Flow of Events	Користувач вводить відповідні дані у форму для донату і підтверджує платіж
Extension	-
Post-Condition	Дані про донат додаються в систему

Таблиця 1.10 - Варіант використання UC-9

Use case name	Переглянути отриманні донати
Use case ID	UC-9
Goals	Отримати інформацію про отримані донати
Actors	Користувач системи
Trigger	Користувач хоче розуміти хто і скільки коштів відправив на його цілі для донатів

Продовження таблиці 1.10

Pre-conditions	Користувач авторизований в системі
Flow of Events	Користувач переходить на сторінку з отриманими донатами
Extension	-
Post-Condition	-

Таблиця 1.11 - Варіант використання UC-10

Use case name	Отримати інформацію про поточний баланс
Use case ID	UC-10
Goals	Користувач хоче знати загальну суму коштів доступних для виводу
Actors	Користувач системи
Trigger	Користувач переходить на сторінку з виводом коштів
Pre-conditions	Користувач авторизований в системі
Flow of Events	Користувач переходить на сторінку з виводом коштів
Extension	-
Post-Condition	-

Таблиця 1.12 - Варіант використання UC-11

Use case name	Зробити запит на вивід доступних коштів
Use case ID	UC-11
Goals	Користувач хоче отримати кошти зібрані з донатів
Actors	Користувач системи
Trigger	Користувач хоче отримати кошти зібрані з донатів
Pre-conditions	Користувач авторизований в системі

Продовження таблиці 1.12

Flow of Events	Користувач переходить на сторінку з виводом коштів, вводить номер карти і суму для виводу та натискає кнопку “Створити запит”
Extension	-
Post-Condition	-

1.4.1 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. У таблицях 1.13 – 1.20 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 1.5.

Таблиця 1.13 – Функціональна вимога FR-1

Назва	Вхід в систему
Опис	Система повинна надавати можливість входу користувачу до системи за допомогою його гугл аккаунту.

Таблиця 1.14 – Функціональна вимога FR-2

Назва	Реєстрація в системі
Опис	Система повинна надавати можливість зареєструватися користувачу в системі за допомогою його гугл аккаунту.

Таблиця 1.15 – Функціональна вимога FR-3

Назва	Зміна унікального імені для користувача
Опис	Система повинна надавати можливість користувачу обирати ім’я по якому інші користувачі зможуть бачити його цілі для донатів.

Таблиця 1.16 – Функціональна вимога FR-4

Назва	Перегляд статусів цілей для донатів
Опис	Система повинна надавати можливість отримання актуальних статусів цілей для донатів.

Таблиця 1.17 – Функціональна вимога FR-5

Назва	Робота з цілями для донатів
Опис	Система повинна надавати можливість створення/редагування цілей для донатів.

Таблиця 1.18 – Функціональна вимога FR-6

Назва	Проведення донату
Опис	Система повинна надавати можливість проведення донату користувачем на певну ціль.

Таблиця 1.19 – Функціональна вимога FR-7

Назва	Перегляд інформації про отримані донати
Опис	Система повинна надавати можливість отримання даних про отримані донати.

Таблиця 1.20 – Функціональна вимога FR-8

Назва	Вивід коштів
Опис	Система повинна надавати можливість створення запиту на виведення коштів.

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8
UC-1	+							
UC-2		+						
UC-3			+					
UC-4				+				
UC-5					+			
UC-6					+			
UC-7				+				
UC-8						+		
UC-9							+	
UC-10								+
UC-11								+

Рисунок 1.2 – Матриця трасування вимог

1.4.2 Розроблення нефункціональних вимог

Система розрахована на одночасну обробку даних багатьох клієнтів та великої кількості кінцевих користувачів. Ці факти формують наступні нефункціональні вимоги:

- надійність: система повинна бути стійкою та надійною, з мінімальною кількістю відмов. Донати повинні бути оброблені без помилок, а інформація про транзакції повинна бути збережена та захищена від втрати;
- швидкодія: сервіс повинен працювати швидко та ефективно, з мінімальними затримками. Відповідь на запити користувачів та обробка транзакцій мають бути проведені у максимально короткий термін;
- безпека: система повинна забезпечувати високий рівень безпеки для донатів та особистої інформації користувачів.

- аудит та звітність: сервіс повинен забезпечувати можливість ведення аудиту транзакцій та генерацію звітів для подальшого аналізу та контролю;
- вартість та масштабування: система повинна бути ефективною з точки зору витрат, а також здатною масштабуватися без необхідності великих інвестицій у обладнання та інфраструктуру.

1.5 Постановка задачі

Програмне забезпечення призначене для підтримки способів надання фінансової допомоги (донатів).

Метою дипломного проєкту є створення архітектури програмного забезпечення веб-сервісу для донатів та імплементація його у повноцінне програмне забезпечення, що дозволить забезпечити ефективну, прозору та безпечну систему збору та розподілу коштів.

Для реалізації поставленої мети необхідно вирішити комплекс наступних взаємопов’язаних задач:

- реєстрація та авторизація користувачів;
- ведення донатів;
- підтримка донатів;
- отримання поточного балансу;
- створення запитів на виведення коштів;
- редагування даних про користувача;
- створення і редагування цілей для донатів.

Висновки до розділу

В цьому розділі було розглянуто предметну область донатів, її поточний стан, розвиток та рівень її інтеграції з IT-технологіями.

Було здійснено порівняння з існуючими аналогами, з якого можна зрозуміти, що застосунок достатньо спеціалізований, щоб якісно забезпечити основні потреби користувачів.

Наступним кроком був аналіз існуючих рішень в плані архітектури застосунку, інструментів реалізації та зовнішніх залежностей. Цей аналіз дозволив сформувати рішення про те, що платформа буде розроблена як клієнт-серверна система.

Далі було сформовано основні функціональні, нефункціональні вимоги, варіанти використання застосунку і сформовано задачу.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Для опису основних бізнес процесів програмного забезпечення використовується BPMN модель [2]. На рисунку 2.1 зображений основнтй бізнес процес, інші процеси є лінійними і просто задані послідовністю дій.

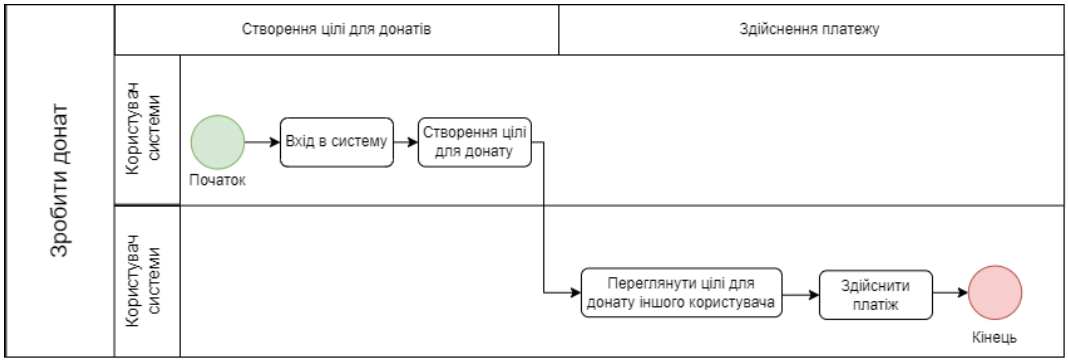


Рисунок 2.1 – Схема бізнес-процесу «Зробити донат»

Детально розглянемо цей бізнес-процес.

Бізнес-процес «Зробити донат» описує взаємодію різних користувачів системи. Процес починається з входу в систему одним з користувачів і створення певної цілі для донатів. Після того, як ціль для донатів була створена, інші користувачі мають можливість переглянути її. Після того як інший користувач переглянув певну ціль для донатів, він має можливість здійснити платіж.

Інші бізнес будуть подані у вигляді послідовностей. Далі надані послідовності для подальших бізнес процесів.

Опис послідовності входу користувачів в систему:

- користувач переходить на сторінку для входу;
- користувач натискає кнопку увійти/зареєструватися;
- користувач обирає гугл аккаунт для входу.

Опис послідовності реєстрації користувачів в системі:

– користувач переходить на сторінку для входу;
– користувач натискає кнопку увійти/зареєструватися;
– користувач обирає гугл аккаунт для реєстрації.

Опис послідовності зміни унікального ідентифікатора користувача:

– користувач переходить на сторінку з налаштуваннями;
– користувач вводить новий унікальний ідентифікатор.

Опис послідовності перегляду статусу власних цілей для донату:

– користувач переходить на сторінку з власними цілями для донату.

Опис послідовності створення власної цілі для донатів:

– користувач переходить на сторінку з власними цілями для донату;
– користувач вводить необхідні дані і натискає кнопку «Створити».

Опис послідовності редагування власної цілі для донатів:

– користувач переходить на сторінку з власними цілями для донату;
– користувач обирає ціль для редагування;
– користувач вводить нові дані і натискає кнопку «Змінити».

Опис послідовності перегляду статусу цілей для донату іншого користувача:

– користувач переходить на сторінку з цілями для донату іншого користувача.

Опис послідовності перегляду отриманих донатів:

– користувач переходить на сторінку з отриманими донатами.

Опис послідовності отримання інформації про поточний баланс:

– користувач переходить на сторінку з виводом коштів.

Опис послідовності створення запиту на виведення коштів:

– користувач переходить на сторінку з виводом коштів;
– користувач вводить номер карти і суму для виводу та натискає кнопку “Створити запит”.

2.2 Архітектура програмного забезпечення

Архітектура програмного забезпечення веб-сервісу для донатів базується на трирівневій клієнт-серверній моделі, яка розділяє функціональні компоненти та взаємодії системи на три рівні: клієнтський рівень, серверний рівень та рівень зберігання даних. Кожен рівень відповідає за свої завдання та взаємодії з іншими рівнями.

Клієнтський рівень включає у себе інтерфейс користувача, що дозволяє користувачам взаємодіяти з веб-сервісом для донатів. Клієнтський рівень може бути представлений у вигляді веб-додатку або мобільного додатку, який надає зручний та інтуїтивно зрозумілий інтерфейс для здійснення донатів та керування персональними налаштуваннями.

Серверний рівень відповідає за обробку запитів користувачів, бізнес-логіку та забезпечення виконання функціональних вимог системи. Він включає в себе набір серверів та компонентів, які обробляють та аналізують запити, взаємодіють з базою даних та іншими зовнішніми системами, здійснюють логіку опрацювання платежів, ведення обліку та надсилання підтверджень до користувачів.

Рівень зберігання даних відповідає за збереження та керування даними, які використовуються в системі. Він включає в себе базу даних, яка зберігає інформацію про користувачів, транзакції, налаштування та інші важливі дані, необхідні для роботи веб-сервісу. Рівень зберігання даних також може

включати механізми резервного копіювання, відновлення даних та забезпечення їх цілісності та конфіденційності.

Взаємодія між цими рівнями здійснюється за допомогою протоколів передачі даних, таких як HTTP або HTTPS, а також за допомогою API (Application Programming Interface), які дозволяють передавати дані та виконувати запити між клієнтським та серверним рівнями.

Трирівнева архітектура [3] дозволяє розділити функціональності та забезпечити більшу гнучкість, масштабованість та повторне використання компонентів системи. Вона також сприяє покращенню безпеки, оскільки зовнішні користувачі не мають прямого доступу до бази даних або бізнес-логіки, а взаємодіють лише через контрольовані інтерфейси.

Також в архітектурі програмного забезпечення веб-сервісу для донатів, використовується підхід REST [4] (Representational State Transfer). REST є архітектурним стилем, який використовується для побудови розподілених систем, зокрема веб-сервісів.

За підходом REST, веб-сервіс вважається як набір ресурсів, до яких можна отримати доступ через стандартні HTTP-методи, такі як GET, POST, PUT та DELETE. Клієнти взаємодіють з ресурсами, надсилаючи HTTP-запити на відповідні URL-шляхи і отримуючи HTTP-відповіді з відповідними статусами та даними.

REST забезпечує простоту та легкість розуміння інтерфейсу, а також дозволяє розділити клієнтський та серверний код, сприяючи модульності та переносимості. Він також підтримує безстанність, що означає, що кожен запит вважається самодостатнім і сервер не зберігає жодного стану про клієнта між запитами.

2.3 Конструювання програмного забезпечення

Конструювання програмного забезпечення є важливим етапом у розробці веб-сервісу для донатів. На цьому етапі відбувається створення програмного коду та налаштування компонентів, що входять до складу

системи. Система не має якихось складних алгоритмів, чи формул для розрахунків. Під час розробки архітектури в цій системі основна увага буде звернена на серверну її частину.

В серверному застосунку використовувалася трирівнева архітектура. Застосунок розбивається на 3 окремих шари, кожен з яких може взаємодіяти лише з шаром нижче, але не може звертатись до шарів вище.

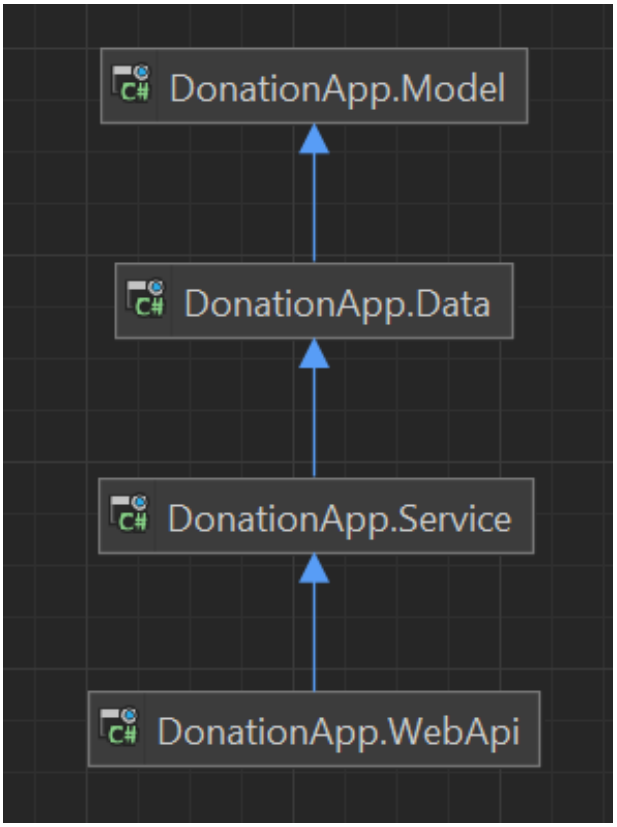


Рисунок 2.2 – Діаграма проекту серверного застосунку

Вигляд архітектури серверного застосунку зображено на рисунку 2.3.

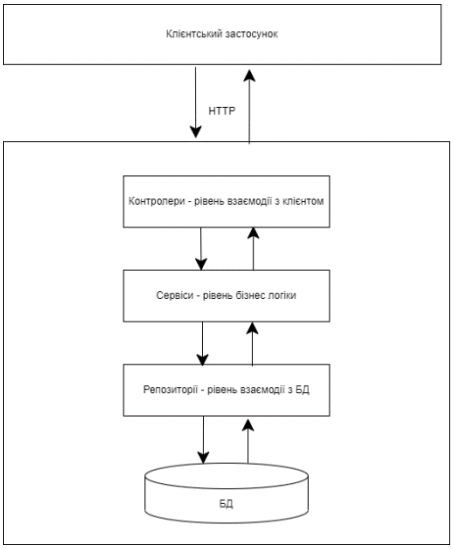


Рисунок 2.3 – Архітектура серверного застосунку

Рівень доступу до даних в системі використовує підхід з використанням Entity Framework [5] та паттерн Репозиторій [6]. Цей рівень відповідає за взаємодію з базою даних та забезпечення операцій збереження, оновлення, вилучення та запиту даних.

Entity Framework є об'єктно-реляційним маппером (ORM), який дозволяє взаємодіяти з базою даних через об'єктно-орієнтовані моделі. Він дозволяє легко відображати таблиці бази даних на класи об'єктів та здійснювати операції з ними безпосередньо в коді.

Паттерн Репозиторій (Repository) використовується для абстрагування роботи з базою даних. Він визначає інтерфейси та методи для виконання операцій збереження, оновлення, вилучення та запиту даних. Репозиторії приховують деталі доступу до бази даних та надають зручний спосіб роботи з даними на рівні бізнес-логіки. Класи репозиторіїв визначаються для кожного типу даних або сутності, які зберігаються в базі даних. Ці репозиторії надають методи для створення, читання, оновлення та видалення сутностей.

Цей рівень дозволяє абстрагуватися від деталей бази даних і забезпечує гнучкість та повторне використання коду. Використання паттерну Репозиторій сприяє легкій зміні механізму доступу до даних без впливу на бізнес-логіку вищих рівнів системи.

Рівень бізнес-логіки веб-сервісу відповідає за обробку основних функціональних процесів та правил, які регулюють функціонування системи. Його завдання - забезпечити логічну та коректну роботу сервісу, включаючи приймання та обробку донатів, управління користувачами, взаємодію з платіжної системою та інші бізнес-процеси.

Рівень контролерів відповідає за обробку вхідних запитів від користувачів та керування виконанням бізнес-логіки системи. Контролери взаємодіють з рівнем представлення (інтерфейсом користувача) та рівнем бізнес-логіки, забезпечуючи потрібну функціональність та поведінку системи.

Інші аспекти розробки системи, які не пов’язані з архітектурним підходом:

Асинхронна взаємодія [7] між компонентами системи. Асинхронне програмування дозволяє ефективніше використовувати ресурси системи та забезпечувати відгуки на запити користувачів без блокування виконання інших задач.

Використання SOLID принципів [8]. Використання цих принципів сприяє створенню гнучкої, розширюваної та підтримуваної архітектури системи.

Практичний підхід – Чистий Код [9]. Цей підхід включає в себе використання зрозумілих імен змінних та функцій, відмову від надлишкового коду, дотримання стандартів форматування, написання коротких та зрозумілих методів та уникнення заплутаних конструкцій.

Використання DI [10]. Це сприяє зменшенню залежностей між класами, полегшує тестування та робить код більш гнучким.

Для взаємодії з даними буде використано СУБД MSSQl server завдяки його високій інтеграції з .NET платформою. Діаграми сутностей зображена на рисунку 2.5 і описана в таблицях 2.1-2.6.

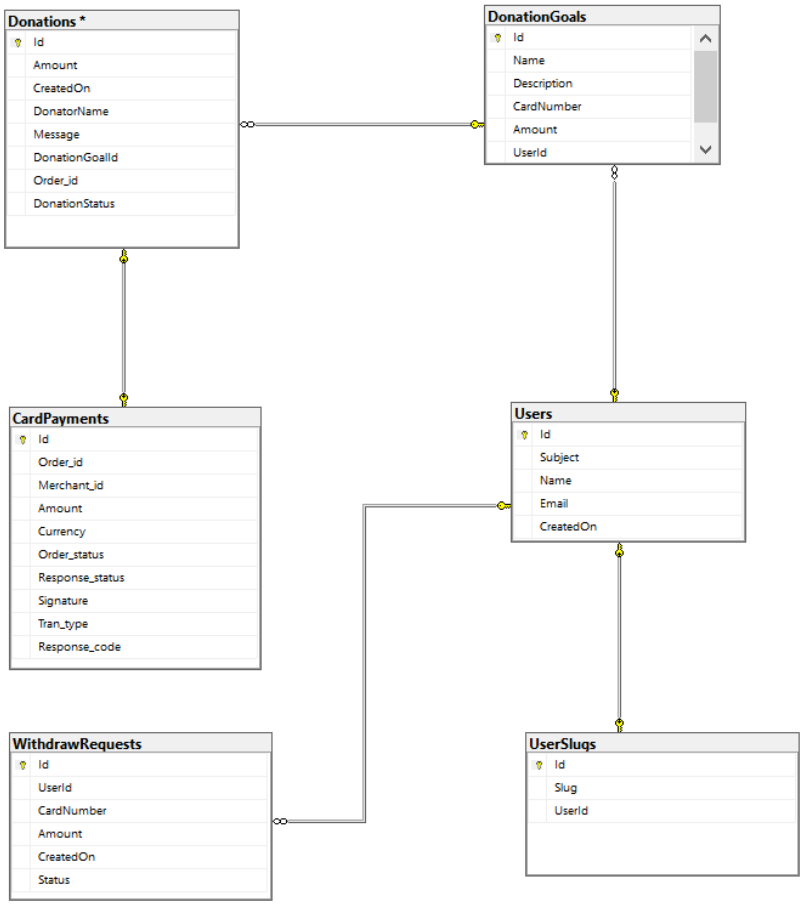


Рисунок 2.4 – Діаграма сутностей БД

Таблиця 2.1 – Опис таблиці Users

Таблиця	Назва поля	Тип даних	Опис
Users	Id	Guid	Унікальний ідентифікатор
	Subject	nvarchar	Ідентифікаційний номер гугл аккаунту
	Name	nvarchar	Ім’я користувача
	Email	nvarchar	Електронна адреса користувача
	CreatedOn	datetime	Дата реєстрації користувача

Таблиця 2.2 – Опис таблиці UserSlugs

Таблиця	Назва поля	Тип даних	Опис
UserSlugs	Id	Guid	Унікальний ідентифікатор
	Slug	string	Текстовий ідентифікатор користувача
	UserId	Guid	Унікальний ідентифікатор користувача

Таблиця 2.3 – Опис таблиці WithdrawRequests

Таблиця	Назва поля	Тип даних	Опис
Withdraw Requests	Id	Guid	Унікальний ідентифікатор
	UserId	Guid	Унікальний ідентифікатор користувача
	CardNumber	nvarchar	Номер карти для виводи коштів
	Amount	int	Сума виводу
	CreatedOn	datetime	Дата створення запиту
	Status	int	Статус обробки запиту

Таблиця 2.4 – Опис таблиці Donations

Таблиця	Назва поля	Тип даних	Опис
Donations	Id	Guid	Унікальний ідентифікатор
	Amount	int	Сума донату
	CreatedOn	datetime	Дата створення донату
	DonatorName	nvarchar	Ім’я користувача, що здійснив донат
	Message	nvarchar	Повідомлення в донаті
	DonationGoalId	Guid	Унікальний ідентифікатор цілі для донатів
	Order_id	string	Унікальний ідентифікатор платежа
	DonationStatus	int	Статус донату

Таблиця 2.5 – Опис таблиці DonationGoals

Таблиця	Назва поля	Тип даних	Опис
Donation Goals	Id	int	Ідентифікаційний номер користувача
	Name	nvarchar	Назва цілі для донатів
	Description	nvarchar	Опис цілі для донатів
	CardNumber	nvarchar	Номер карти для збору

Продовження таблиці 2.5

	Amount	int	Загальна сума збору
	UserId	Guid	Унікальний ідентифікатор користувача

Таблиця 2.6 – Опис таблиці CardPayments

Таблиця	Назва поля	Тип даних	Опис
CardPayments	Id	int	Ідентифікаційний номер закладу
	Order_id	string	Унікальний ідентифікатор платежа
	Merchant_id	int	Ідентифікатор мерчанту
	Amount	int	Сума платежу
	Currency	string	Валюта
	Order_status	nvarchar	Статус платежу
	Response_status	nvarchar	Статус відповіді від платжної системи
	Signature	nvarchar	Підпис
	Tran_type	nvarchar	Тип транзакції
	Response_code	nvarchar	Код відповіді від платіжної системи

Далі надано опис основних розроблених класів серверної частини програмного забезпечення та опис їх методів.

Клас: RepositoryBase<T,K>

Опис: Цей клас є базовим репозиторієм для доступу до даних і реалізує загальний набір операцій для роботи з сутностями.

Методи:

1) Create(T entity) Опис: Метод додає нову сутність до бази даних. Параметри:
   - entity: об'єкт типу T, який потрібно створити. Повертає:
   - Об'єкт типу T, який був доданий до бази даних.
2) GetById(K id) Опис: Метод отримує сутність з бази даних за заданим ідентифікатором.

   Параметри:
   - id: ідентифікатор типу K, за яким потрібно знайти сутність.

   Повертає:
   - Об'єкт типу T, який має вказаний ідентифікатор.
3) Delete(K id) Опис: Метод видаляє сутність з бази даних за заданим ідентифікатором. Параметри:
   - id: ідентифікатор типу K, за яким потрібно знайти сутність для видалення. Повертає:
   - Відсутнє значення.
4) Update(T entity) Опис: Метод оновлює вказану сутність в базі даних. Параметри:
   - entity: об'єкт типу T, який потрібно оновити.

   Повертає:
   - Об'єкт типу T, який був оновлений.
5) SaveChanges() Опис: Метод зберігає всі внесені зміни в базу даних. Параметри:
   - Відсутні.

   Повертає:
   - Відсутнє значення.

Клас: DonationGoalRepository

Опис: Цей клас є репозиторієм для доступу до даних про мети збору пожертв для конкретного користувача.

Наслідує: RepositoryBase<DonationGoal, Guid>, IDonationGoalRepository

Методи:

1) GetByUserId(Guid userId) Опис: Метод отримує всі мети збору пожертв, пов'язані з конкретним користувачем за його ідентифікатором. Параметри:
   - userId: ідентифікатор типу Guid, який представляє ідентифікатор користувача.

   Повертає:
   - Об'єкт типу IQueryable<DonationGoal>, який містить всі мети збору пожертв, пов'язані з вказаним користувачем.

Клас: DonationRepository

Опис: Цей клас є репозиторієм для доступу до даних про пожертви.

Наслідує: RepositoryBase<Donation, Guid>, IDonationRepository

Методи:

1) GetAll() Опис: Метод отримує всі пожертви з бази даних. Параметри:
   - Відсутні. Повертає:
   - Об'єкт типу IQueryable<Donation>, який містить всі пожертви.
2) GetByUserId(Guid userId) Опис: Метод отримує всі пожертви, пов'язані з конкретним користувачем за його ідентифікатором, які мають статус "Success".

   Параметри:
   - userId: ідентифікатор типу Guid, який представляє ідентифікатор користувача.

   Повертає:
   - Об'єкт типу IQueryable<Donation>, який містить всі пожертви, пов'язані з вказаним користувачем та мають статус "Success".

3) GetUnProcessedByUserId(Guid userId) Опис: Метод отримує всі невиконані пожертви, пов'язані з конкретним користувачем за його ідентифікатором, які мають статус "New".

Параметри:

- userId: ідентифікатор типу Guid, який представляє ідентифікатор користувача.

Повертає:

- Об'єкт типу IQueryable<Donation>, який містить всі невиконані пожертви, пов'язані з вказаним користувачем та мають статус "New".

4) GetByOrderId(string orderId) Опис: Метод отримує пожертву за заданим ідентифікатором замовлення.

Параметри:

- orderId: рядок, який представляє ідентифікатор замовлення пожертви.

Повертає:

- Об'єкт типу Task<Donation?>, який містить пожертву або null, якщо пожертва не знайдена.

Клас: UserRepository

Опис: Цей клас є репозиторієм для доступу до даних про користувачів.

Наслідує: RepositoryBase<User, Guid>, IUserRepository

Методи:

1) GetUserBySubject(string subject) Опис: Метод отримує користувача за заданим значенням "subject".

Параметри:

- subject: рядок, що представляє значення "subject" користувача.

Повертає:

- Об'єкт типу Task<User?>, який містить користувача або null, якщо користувача не знайдено.

2) GetUserBySlug(string userSlug) Опис: Метод отримує користувача за заданим значенням "userSlug".

Параметри:

- userSlug: рядок, що представляє значення "userSlug" користувача.

Повертає:

- Об'єкт типу Task<User?>, який містить користувача або null, якщо користувача не знайдено.

3) GetUserIdBySubject(string subject) Опис: Метод отримує ідентифікатор користувача за заданим значенням "subject".

Параметри:

- subject: рядок, що представляє значення "subject" користувача.

Повертає:

- Значення типу Guid або null, якщо користувача не знайдено.

4) GetById(Guid id) Опис: Метод отримує користувача за заданим ідентифікатором.

Параметри:

- id: ідентифікатор типу Guid, який представляє ідентифікатор користувача.

Повертає:

- Об'єкт типу Task<User>, який містить користувача або генерує виключення, якщо користувача не знайдено.

Клас: WithdrawRequestRepository

Опис: Цей клас є репозиторієм для доступу до даних про запити на виведення коштів.

Наслідує: RepositoryBase<WithdrawRequest, Guid>, IWithdrawRequestRepository

Методи:

1) GetUserWithdrawRequests(Guid userId) Опис: Метод отримує список запитів на виведення коштів для заданого користувача.

Параметри:

- userId: ідентифікатор типу Guid, який представляє ідентифікатор користувача.

Повертає:

- Об'єкт типу Task<IEnumerable<WithdrawRequest>>, який містить список запитів на виведення коштів або пустий список, якщо запитів не знайдено.

Клас: DonationGoalService

Опис: Цей клас представляє сервіс для управління цілями збору пожертв для користувачів.

Наслідує: IDonationGoalService

1) Create(DonationGoalRequestDto donationGoalDto) Опис: Метод створює нову ціль збору пожертв.

   Параметри:

   - donationGoalDto: Об'єкт типу DonationGoalRequestDto, який містить дані для створення цілі збору пожертв.

   Повертає:

   - Об'єкт типу Task<DonationGoalResponseDto>, який містить дані про створену ціль збору пожертв.

2) Update(DonationGoalUpdateDto donationGoalDto) Опис: Метод оновлює існуючу ціль збору пожертв.

   Параметри:

   - donationGoalDto: Об'єкт типу DonationGoalUpdateDto, який містить дані для оновлення цілі збору пожертв.

   Повертає:

   - Об'єкт типу Task<DonationGoalResponseDto>, який містить оновлені дані про ціль збору пожертв.

3) GetByUser(string userSlug) Опис: Метод отримує список цілей збору пожертв для заданого користувача.

Параметри:

- userSlug: Рядок, що представляє унікальний ідентифікатор користувача.

Повертає:

- Об'єкт типу Task<IEnumerable<DonationGoalResponseDto>>, який містить список цілей збору пожертв або порожній список, якщо цілей не знайдено.

Клас: DonationService

Опис: Цей клас представляє сервіс для управління пожертвами.

Наслідує: IDonationService

Методи:

1) GetByFilter(DonationFilter filter) Опис: Метод отримує список пожертв за заданим фільтром.

   Параметри:

   - filter: Об'єкт типу DonationFilter, який містить параметри фільтрації пожертв.

   Повертає:

   - Об'єкт типу Task<IList<Donation>>, який містить список пожертв, які задовольняють вказаний фільтр.

2) GetByUserId(Guid userId) Опис: Метод отримує список пожертв для заданого користувача.

   Параметри:

   - userId: Об'єкт типу Guid, який представляє унікальний ідентифікатор користувача.

   Повертає:

   - Об'єкт типу Task<IList<Donation>>, який містить список пожертв для заданого користувача.

Клас: PaymentService

Опис: Цей клас представляє сервіс для обробки платежів.

Наслідує: IPaymentService

Методи:

1) GenerateDonationLink(DonationDto donationDto) Опис: Метод генерує посилання для пожертвування.

   Параметри:

   - donationDto: Об'єкт типу DonationDto, який містить дані пожертви.

   Повертає:

   - Об'єкт типу Task<string>, який представляє згенероване посилання для пожертвування.

2) ProcessPaymentCallback(ResponseModel responseModel) Опис: Метод обробляє зворотний виклик платежу.

   Параметри:

   - responseModel: Об'єкт типу ResponseModel, який містить дані зворотного виклику платежу.

   Повертає:

   - Об'єкт типу Task, який вказує на асинхронну операцію обробки зворотного виклику платежу.

Клас: UserService

Опис: Цей клас представляє сервіс для операцій з користувачами.

Наслідує: IUserService

Методи:

1) CreateIfNotExist(string tokenId) Опис: Метод створює нового користувача, якщо користувач не існує за вказаним токеном. Параметри:
   - tokenId: Рядок, що представляє токен користувача. Повертає:
   - Об'єкт типу Task<UserDto>, який представляє створеного користувача або існуючого, якщо користувач вже існує.
2) SetUserSlug(UserSlugDto userSlugDto) Опис: Метод встановлює "слаг" користувача.

Параметри:

- userSlugDto: Об'єкт типу UserSlugDto, який містить дані "слага" користувача.

Повертає:

- Об'єкт типу Task<bool>, який показує, чи було успішно встановлено "слаг" користувача.

3) GetUserSlug(Guid userId) Опис: Метод отримує "слаг" користувача за його ідентифікатором.

Параметри:

- userId: Об'єкт типу Guid, який представляє ідентифікатор користувача.

Повертає:

- Об'єкт типу Task<string>, який представляє "слаг" користувача.

Клас: WithdrawService

Опис: Цей клас представляє сервіс для операцій з виведенням коштів.

Наслідує: IWithdrawService

Методи:

1) GetUserAvailableWithdrawAmount(Guid userId) Опис: Метод отримує доступну суму для виведення коштів користувача.

Параметри:

- userId: Об'єкт типу Guid, який представляє ідентифікатор користувача.

Повертає:

- Об'єкт типу Task<IEnumerable<AvailableWithdrawAmountResponse>>, який представляє колекцію доступних сум для виведення коштів для кожного номеру картки.

2) CreateWithdrawRequest(WithdrawRequestDto withdrawRequestDto) Опис: Метод створює запит на виведення коштів. Параметри:

- withdrawRequestDto: Об'єкт типу WithdrawRequestDto, який містить дані запиту на виведення коштів. Повертає:
- Об'єкт типу Task<WithdrawRequestDto>, який представляє створений запит на виведення коштів.

Клас: AuthorizeController

Опис: Цей клас представляє контролер для авторизації користувача.

Наслідує: ControllerBase

Методи:

1) CreateIfNotExist(UserIn userIn) Опис: Метод для створення користувача, якщо він ще не існує.

Параметри:

- userIn: Об'єкт типу UserIn, який містить дані користувача.

Повертає:

- Об'єкт типу Task<IActionResult>, який представляє результат операції. Відповідь містить статус Ok і об'єкт користувача у разі успішного створення, або статус BadRequest з повідомленням про помилку у разі невдалої спроби створення.

Клас: DonationController

Опис: Цей клас представляє контролер для операцій з донатів.

Наслідує: UserControllerBase

Методи:

1) GetByFilter(DonationFilter filter) Опис: Метод для отримання списку донатів з використанням фільтра.

Параметри:

- filter: Об'єкт типу DonationFilter, який містить фільтр для запиту.

Повертає:

- Об'єкт типу Task<IActionResult>, який представляє результат операції. Відповідь містить статус Ok і список донатів у разі

успішного отримання, або статус BadRequest з повідомленням про помилку у разі невдалих спроб.

2) GetByUserId() Опис: Метод для отримання списку донатів для поточного користувача.

Параметри:

- Відсутні.

Повертає:

- Об'єкт типу Task<IActionResult>, який представляє результат операції. Відповідь містить статус Ok і список донатів у разі успішного отримання, або статус BadRequest з повідомленням про помилку у разі невдалих спроб.

Клас: DonationGoalController

Опис: Цей клас представляє контролер для операцій з цілями збору пожертв.

Наслідує: UserControllerBase

Методи:

1) Create(DonationGoalRequestDto donationGoalDto) Опис: Метод для створення нової цілі збору пожертв.

Параметри:

- donationGoalDto: Об'єкт типу DonationGoalRequestDto, який містить дані про нову ціль збору пожертв.

Повертає:

- Об'єкт типу Task<IActionResult>, який представляє результат операції. Відповідь містить статус Ok і створену ціль збору пожертв у разі успішного створення, або статус BadRequest з повідомленням про помилку у разі невдалих спроб.

2) Update(DonationGoalUpdateDto donationGoalDto) Опис: Метод для оновлення існуючої цілі збору пожертв.

Параметри:

- donationGoalDto: Об'єкт типу DonationGoalUpdateDto, який містить дані для оновлення цілі збору пожертв.

Повертає:

- Об'єкт типу Task<IActionResult>, який представляє результат операції. Відповідь містить статус Ok і оновлену ціль збору пожертв у разі успішного оновлення, або статус BadRequest з повідомленням про помилку у разі невдалих спроб.

3) GetByUser(string userSlug) Опис: Метод для отримання списку цілей збору пожертв для конкретного користувача за допомогою його ідентифікатора користувача.

Параметри:

- userSlug: Рядок, що містить унікальний ідентифікатор користувача.

Повертає:

- Об'єкт типу Task<IActionResult>, який представляє результат операції. Відповідь містить статус Ok і список цілей збору пожертв у разі успішного отримання, або статус BadRequest з повідомленням про помилку у разі невдалих спроб.

Клас: PaymentController

Опис: Цей клас представляє контролер для операцій з платежами та генерації посилань на пожертви.

Наслідує: ControllerBase

Методи:

1) PaymentCallback(ResponseModel response) Опис: Метод для обробки зворотного виклику платежу.

Параметри:

- response: Об'єкт типу ResponseModel, який містить дані про статус платежу.

Повертає:

- Об'єкт типу Task<IActionResult>, який представляє результат операції. Відповідь містить статус Ok і статус відповіді у разі успішної обробки зворотного виклику платежу, або статус BadRequest з повідомленням про помилку у разі невдалих спроб.

2) GenerateDonationLink(DonationDto dto) Опис: Метод для генерації посилання на пожертву.

Параметри:

- dto: Об'єкт типу DonationDto, який містить дані для генерації посилання на пожертву.

Повертає:

- Об'єкт типу Task<ActionResult>, який представляє результат операції. Відповідь містить статус Ok і згенероване посилання на пожертву у разі успішної генерації, або статус BadRequest з повідомленням про помилку у разі невдалих спроб.

Клас: UserController

Опис: Цей клас представляє контролер для операцій з користувачами.

Наслідує: UserControllerBase

Методи:

1) SetUserSlug(UserSlugDto userSlugDto) Опис: Метод для встановлення ідентифікатора користувача (UserSlug).

Параметри:

- userSlugDto: Об'єкт типу UserSlugDto, який містить дані для встановлення UserSlug.

Повертає:

- Об'єкт типу Task<IActionResult>, який представляє результат операції. Відповідь містить статус Ok і значення true у разі успішного встановлення UserSlug, або статус BadRequest з значенням false у разі невдалих спроб.

2) GetUserSlug() Опис: Метод для отримання ідентифікатора користувача (UserSlug).

Параметри:

- Відсутні.

Повертає:

- Об'єкт типу Task<IActionResult>, який представляє результат операції. Відповідь містить статус Ok і значення UserSlug у разі успішного отримання, або статус BadRequest з повідомленням про помилку у разі невдалих спроб.

Клас: WithdrawController

Опис: Цей клас представляє контролер для операцій з виведенням коштів.

Наслідує: UserControllerBase

Методи:

1) GetUserAvailableWithdrawAmount() Опис: Метод для отримання доступної суми для виведення коштів користувача.

Параметри:

- Відсутні.

Повертає:

- Об'єкт типу Task<IActionResult>, який представляє результат операції. Відповідь містить статус Ok і список об'єктів AvailableWithdrawAmountResponse у разі успішного отримання, або статус BadRequest з повідомленням про помилку у разі невдалих спроб.

2) CreateWithdrawRequest(WithdrawRequestDto withdrawRequestDto) Опис: Метод для створення запиту на виведення коштів.

Параметри:

- withdrawRequestDto: Об'єкт типу WithdrawRequestDto, який містить дані для створення запиту на виведення коштів.

Повертає:

- Об'єкт типу Task<IActionResult>, який представляє результат операції. Відповідь містить статус Ok і об'єкт типу WithdrawRequestDto у разі успішного створення запиту, або статус BadRequest з повідомленням про помилку у разі невдалих спроб.

Клас: GoogleAuthorizeAttribute

Опис: Цей атрибут використовується для автентифікації користувачів з використанням Google-акаунту. Валідує токен авторизації.

Наслідує: TypeFilterAttribute

.

Клас: GoogleAuthorizeFilter

Опис: Цей фільтр авторизації перевіряє наявність та валідність токену авторизації, а також перевіряє та отримує ідентифікатор користувача з бази даних.

Реалізує: IAuthorizationFilter

Метод OnAuthorization(AuthorizationFilterContext context) Опис: Метод, який виконується під час авторизації і перевіряє наявність та валідність токену авторизації, а також отримує ідентифікатор користувача з бази даних.

Параметри:

- context: Об'єкт типу AuthorizationFilterContext, який містить контекст авторизації.

Повертає:

- Відсутнє значення.

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 2.7.

Таблиця 2.7 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	Rider	Головне середовище розробки програмного забезпечення серверної частини курсової роботи.
2	WebStorm	Додаткове середовище розробки для імплементації одного з клієнтських застосунків.
3	Postman	Програмне забезпечення необхідне для тестування HTTP запитів. Використовувалось для тестування API інтерфейсів, та клієнтських запитів.
4	MSSQL SERVER Management studio	Програмне забезпечення яке надає легкий графічний інтерфейс для доступу до бази даних.
5	Entity framework	Фреймворк для реалізації легкої взаємодії застосунку з БД.
6	Google.Apis.Auth	Зовнішня бібліотека для взаємодії з Google сервісами.

2.4 Аналіз безпеки даних

Забезпечення безпеки даних є важливим аспектом веб-сервісу для донатів.

На стороні сервера, використання GoogleJsonWebSignature та Google.Apis.Auth забезпечує потужні механізми безпеки. GoogleJsonWebSignature (JWT) є стандартом для цифрових підписів JSON-даних. Він дозволяє перевірити цілісність та автентичність даних, переданих між сервером та клієнтом. Google.Apis.Auth, у свою чергу, надає можливості аутентифікації та авторизації через Google-акаунт, що забезпечує безпеку ідентифікації користувача.

На стороні клієнта, використання react-google-login дозволяє здійснювати безпечний процес аутентифікації через Google-акаунт. Ця бібліотека надає інтеграцію з Google Sign-In, що дозволяє клієнтам надати доступ до своїх особистих даних з облікового запису Google з мінімальним ризиком для безпеки.

Висновки до розділу

В цьому розділі було детально розглянути бізнес процеси, архітектуру системи, архітектуру окремих модулів та БД, було розглянуто технологічний стек, бібліотеки, інструменти для реалізації системи та проаналізовано безпеку даних.

Було виділено і описано основний бізнес процес системи «Зробити донат», інші бізнес процеси є просто різними лінійними комбінаціями послідовностей використання системи.

Наступним кроком було розглянуто і описано основні обрані підходи до розробки архітектури всієї системи. Система має трирівневу клієнт серверну архітектуру і складається з серверного і клієнтського застосунку.

Після цього було розглянуто більш детально архітектуру серверного застосунку, яким конкретним чином він взаємодіє з іншими компонентами. Також у цьому підрозділі розглянуто структуру БД, її таблиці та описано технології, бібліотеки та платформи, які було використано в розробці.

Наприкінці було проаналізовано безпеку даних застосунку.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Для аналізу якості ПЗ було використано Embold. Embold – це безкоштовний продукт, який надає функціонал для статичного аналізу коду. За результатами прогону коду серверної частини системи через Embold отримано результат, який зображено на рисунку 3.1.

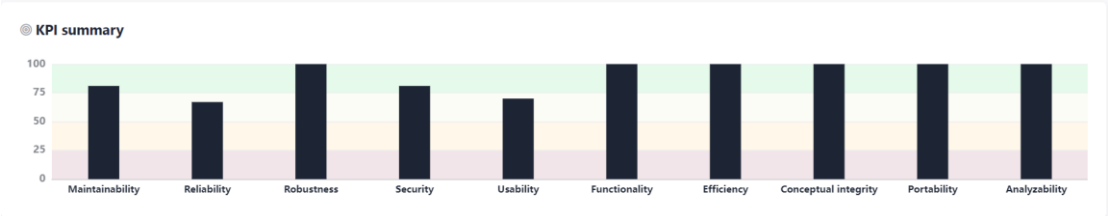


Рисунок 3.1 – Результат статичного аналізу коду

Відповідно до цих результатів, можна побачити, що показники якості програмного забезпечення задовільні. Найнижчий рівень у Reliability – 67/100, також не максимальні у Maintainability, Security та Usability. В інших категоріях аналізатор коду не знайшов жодних помилок в коді системи. Аналізатор загалом оцінює код як «дуже добре».

3.2 Опис процесів тестування

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 3.1 – 3.16.

Таблиця 3.1 – Тест 1.1

Тест	Вхід користувача
Модуль	Вхід користувача
Номер тесту	1.1
Початковий стан системи	Користувач знаходиться на сторінці входу

Продовження таблиці 3.1

Вхідні данні	Google аккаунт користувача
Опис проведення тесту	Натискається кнопка увійти/зареєструватися і у спливаючому вікні обирається Google через який раніше здійснювався вхід.
Очікуваний результат	Вхід проходить успішно, користувач перенаправляється на головну сторінку.
Фактичний результат	Вхід проходить успішно, користувач перенаправляється на головну сторінку.

Таблиця 3.2 – Тест 1.2

Тест	Реєстрація користувача
Модуль	Реєстрація користувача
Номер тесту	1.2
Початковий стан системи	Користувач знаходиться на сторінці входу
Вхідні данні	Google аккаунт користувача
Опис проведення тесту	Натискається кнопка увійти/зареєструватися і у спливаючому вікні обирається Google через який раніше не здійснювався вхід.
Очікуваний результат	Реєстрація проходить успішно, користувач додається у систему, головна сторінка оновлена.
Фактичний результат	Реєстрація проходить успішно, користувач додається у систему, головна сторінка оновлена.

Таблиця 3.3 – Тест 1.3

Тест	Зміна унікального ідентифікатора користувача
Модуль	Конфігурація користувача
Номер тесту	1.3

Продовження таблиці 3.3

Початковий стан системи	Користувач знаходиться на сторінці з налаштуваннями
Вхідні данні	Новий унікальний ідентифікатор користувача
Опис проведення тесту	У відповідне поле вводиться текст. Натискається кнопка зберегти.
Очікуваний результат	Унікальний ідентифікатор користувача оновлюється новим значенням.
Фактичний результат	Унікальний ідентифікатор користувача оновлюється новим значенням.

Таблиця 3.4 – Тест 1.4

Тест	Переглянути статус своїх цілей для донатів
Модуль	Цілі для донатів
Номер тесту	1.4
Початковий стан системи	Користувач увійшов в систему
Вхідні данні	-
Опис проведення тесту	Користувач переходить на сторінку з цілями для донатів.
Очікуваний результат	На сторінці відображаются цілі для донатів користувача.
Фактичний результат	На сторінці відображаются цілі для донатів користувача.

Таблиця 3.5 – Тест 1.5

Тест	Створення цілі для донатів
Модуль	Цілі для донатів
Номер тесту	1.5
Початковий стан системи	Користувач знаходиться на сторінці з цілями для донатів
Вхідні данні	Назва цілі для донатів. Опис цілі для донатів. Номер карти для виводу коштів. Загальна сума збору.
Опис проведення тесту	Заповнюється форма для створення цілі для донатів. Натискається кнопка створити.
Очікуваний результат	Створення проходить успішно. Дані про ціль додаються в БД і відображаються на сторінці з цілями для донатів.
Фактичний результат	Створення проходить успішно. Дані про ціль додаються в БД і відображаються на сторінці з цілями для донатів.

Таблиця 3.6 – Тест 1.6

Тест	Редагування цілі для донатів
Модуль	Цілі для донатів
Номер тесту	1.6
Початковий стан системи	Користувач знаходиться на сторінці з цілями для донатів
Вхідні данні	Новий опис цілі. Нова загальна сума збору.
Опис проведення тесту	Обирається ціль для редагування і у відповідні поля вводиться новий опис цілі і нова загальна суму збору. Натискається кнопка редагувати.
Очікуваний результат	Редагування проходить успішно. Нові дані зберігаються в БД.
Фактичний результат	Редагування проходить успішно. Нові дані зберігаються в БД.

Таблиця 3.7 – Тест 1.7

Тест	Перегляд цілей для донатів іншого користувача
Модуль	Цілі для донатів
Номер тесту	1.7
Початковий стан системи	Довільний
Вхідні данні	Унікальний ідентифікатор іншого користувача
Опис проведення тесту	Користувач переходить на сторінку профілю іншого користувача за допомогою його унікального ідентифікатору.
Очікуваний результат	На сторінці відображаются цілі для донатів іншого користувача.
Фактичний результат	На сторінці відображаются цілі для донатів іншого користувача.

Таблиця 3.8 – Тест 1.8

Тест	Зробити донат
Модуль	Донати
Номер тесту	1.8
Початковий стан системи	Користувач знаходиться на сторінці іншого користувача
Вхідні данні	Сума донату. Повідомлення донату. Ім’я автору донату. Ціль для донату.
Опис проведення тесту	У відповідні поля вводяться: Сума донату, ім’я автору донату, повідомлення донату, ціль для донату. Після цього натискається кнопка зробити донат.

Продовження таблиці 3.8

Очікуваний результат	Дані про донат зберігаються в БД. Користувач переходить на екран оплати.
Фактичний результат	Дані про донат зберігаються в БД. Користувач переходить на екран оплати.

Таблиця 3.9 – Тест 1.9

Тест	Перегляд отриманих донатів
Модуль	Донати
Номер тесту	1.9
Початковий стан системи	Користувач увійшов в систему
Вхідні данні	-
Опис проведення тесту	Користувач переходить на сторінку з даними про донати.
Очікуваний результат	На сторінці відображаються дані про донати користувача.
Фактичний результат	На сторінці відображаються дані про донати користувача.

Таблиця 3.10 – Тест 1.10

Тест	Отримати інформацію про поточний баланс
Модуль	Вивід коштів
Номер тесту	1.10
Початковий стан системи	Користувач увійшов в систему
Вхідні данні	-
Опис проведення тесту	Користувач переходить на сторінку з виводом коштів.
Очікуваний результат	На сторінці відображаються номера карток з доступними сумами для виводу.
Фактичний результат	На сторінці відображаються номера карток з доступними сумами для виводу.

Таблиця 3.11 – Тест 1.11

Тест	Здійснення запиту на вивід коштів
Модуль	Вивід коштів
Номер тесту	1.11
Початковий стан системи	Користувач знаходиться на сторінці замовлення
Вхідні данні	Сума для виводу коштів. Карта для виводу коштів.
Опис проведення тесту	Користувач вводить суму для виводу коштів. Обирає карту для виводу коштів. Натискає кнопку створити запит на виведення коштів.

Продовження таблиці 3.11

Очікуваний результат	Запит на виведення коштів успішно створюється і зберігається в БД.
Фактичний результат	Запит на виведення коштів успішно створюється і зберігається в БД.

3.3 Опис контрольного прикладу

Взаємодія з системою починається з того, що користувач потрапляє на сторінку входу, де він може натиснути кнопку зареєструватися/увійти.

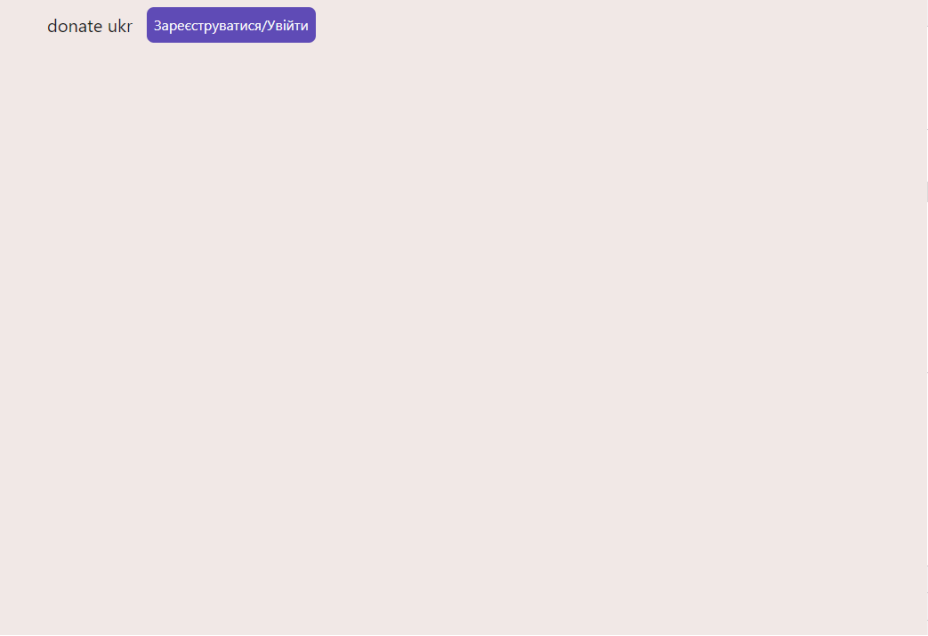


Рисунок 3.2 – Стартова сторінка

Після натискання кнопки зареєструватися/увійти, з’являється форма з кнопкою Log in with Google.

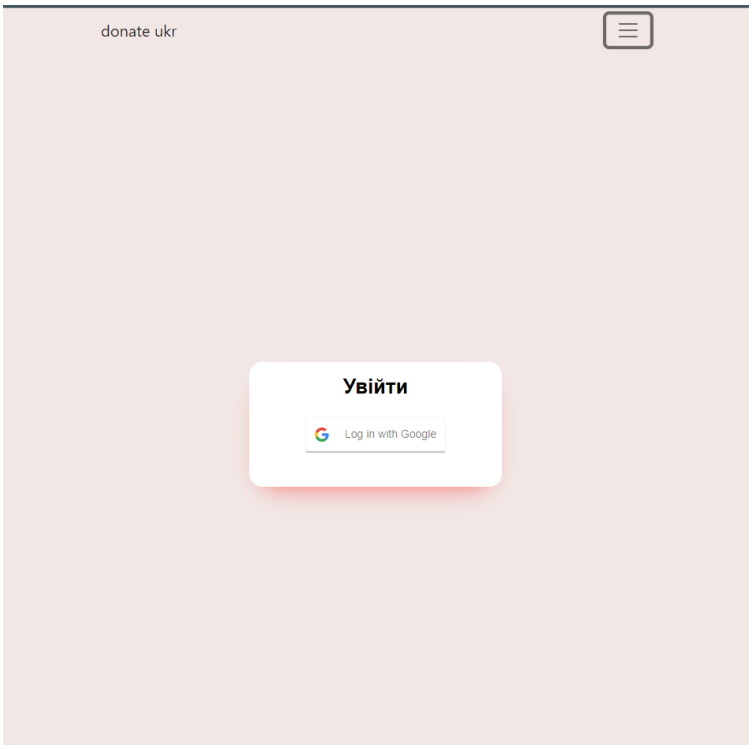


Рисунок 3.3 – Стартова сторінка з кнопкою для входу

Після натискання кнопки Log in with Google з’являється спливаюче вікно з формою для входу в Google аккаунт.

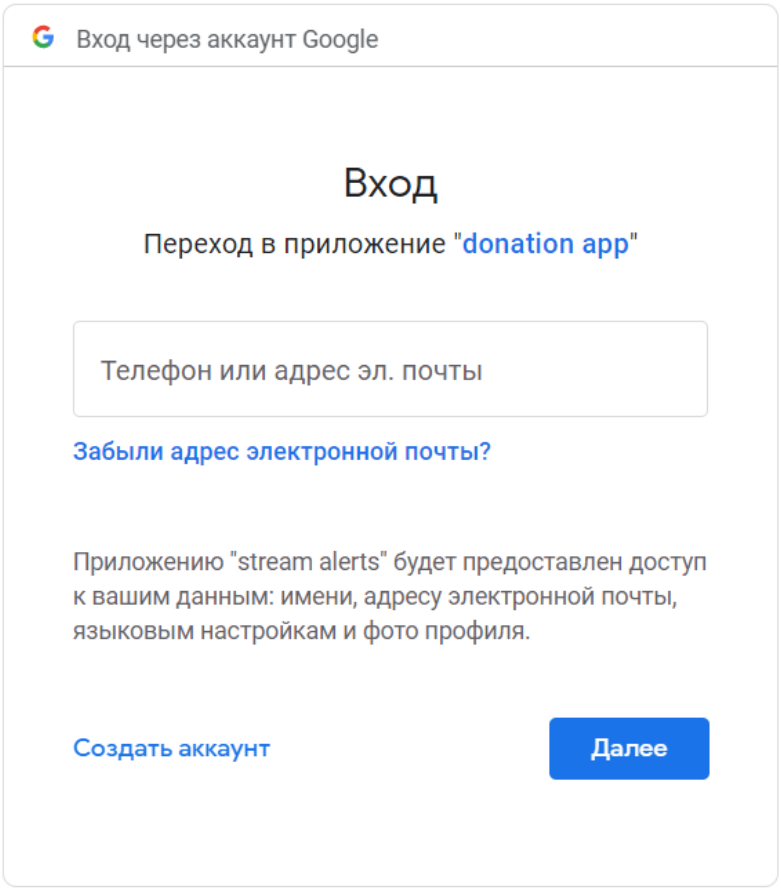


Рисунок 3.4 – Форма для входу в Google аккаунт

Якщо реєстрація/вхід пройшли успішно, то користувач потрапляє на сторінку з навігацією. Сторінка зображено на рисунку 3.5.

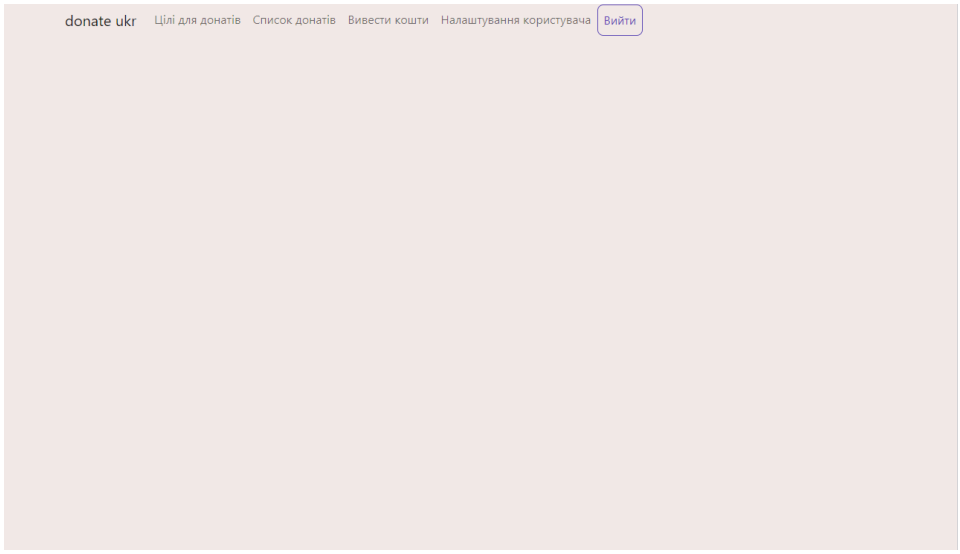


Рисунок 3.5 – Сторінка з навігацією

Користувач може перейти на такі сторінки:

- цілі для донатів;
- список донатів;
- вивести кошти;
- налаштування користувача.

Сторінка з списком донатів містить таблицю, що показує всі донати отримані користувачем на його цілі. Таблиця містить 4 поля: дата, ім’я, повідомлення та сума. Сторінка з списком донатів зображена на рисунку 3.6.

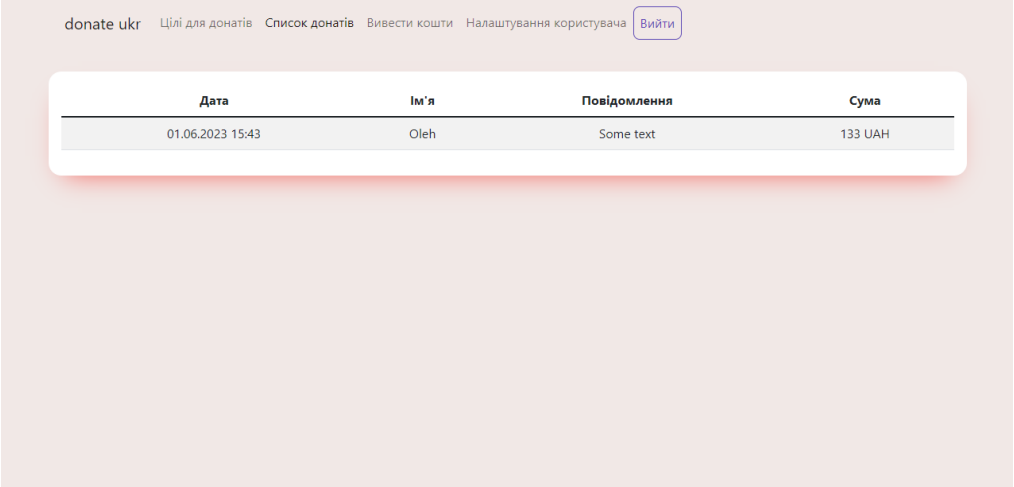


Рисунок 3.6 – Сторінка з списком донатів

Сторінка з цілями для донатів поточного користувача містить всі цілі для донатів користувача. Кожна ціль для донатів містить такі дані: назва цілі, опис цілі, загальна сума збору та поточна сума збору. Також користувач може редагувати кожну свою ціль. Він може змінювати поле опис цілі та поле загальна сума збору. Для того щоб редагувати ціль, користувач повинен змінити дані у відповідних полях та натиснути кнопку зберегти. Також внизу сторінки є форма для створення нових цілей для донатів. Форма містить 4 обов'язкові поля: назва цілі, опис цілі, номер карти для виводу коштів та загальна сума збору. Щоб створити ціль потрібно ввести дані у всі поля форми та натиснути кнопку створити ціль. Після цього ціль для донатів успішно створиться і з'явиться у списку з цілями користувача.

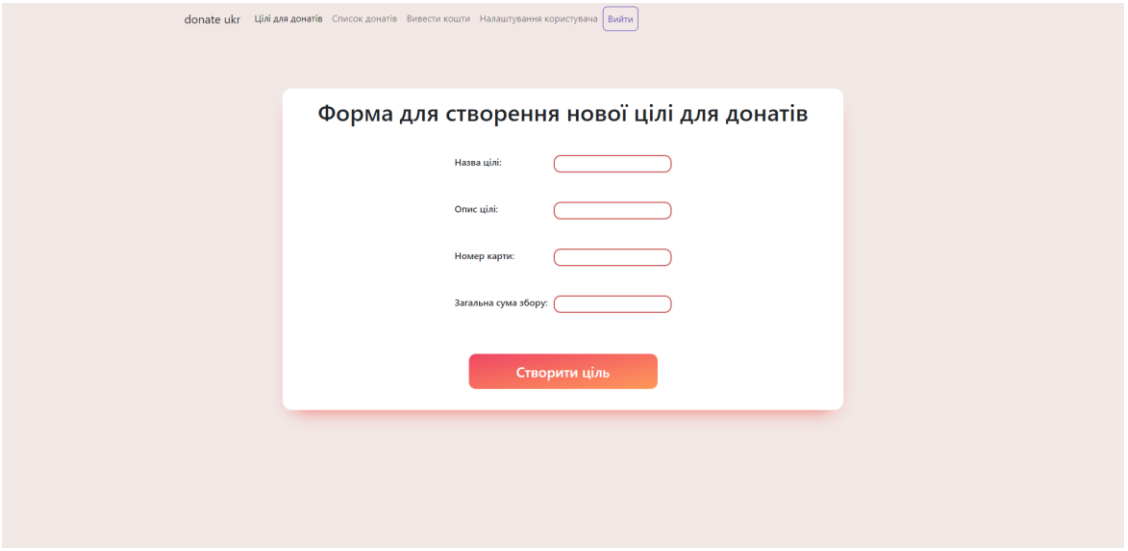


Рисунок 3.7 – Форма для створення нових цілей для донатів

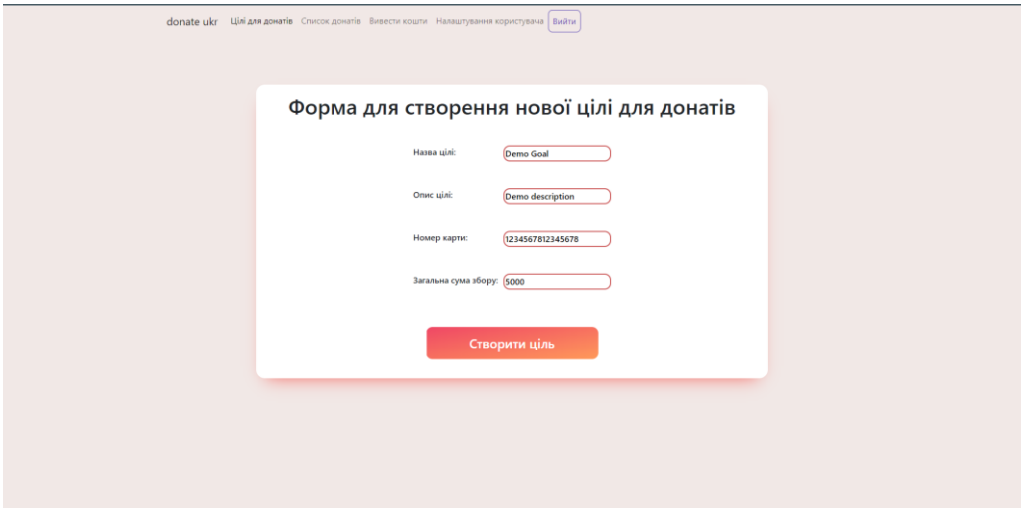


Рисунок 3.8 – Форма із заповненими даними

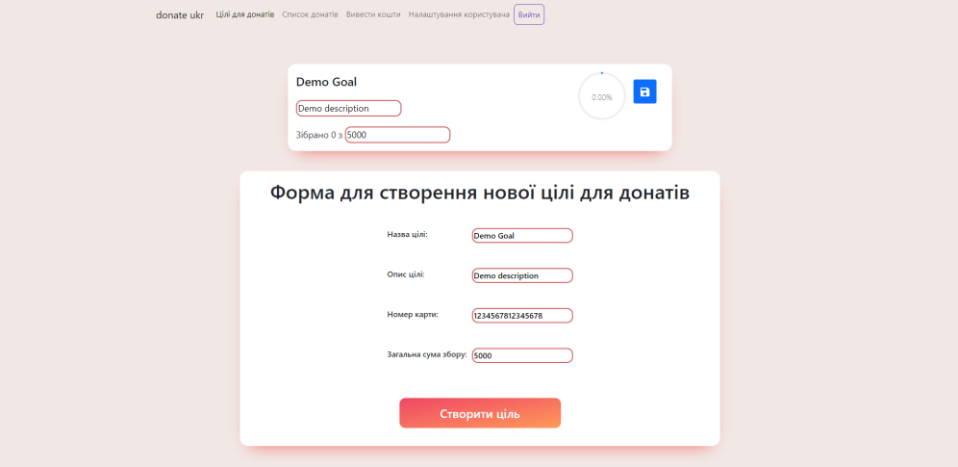


Рисунок 3.9 – Вигляд сторінки після натискання кнопки створити ціль

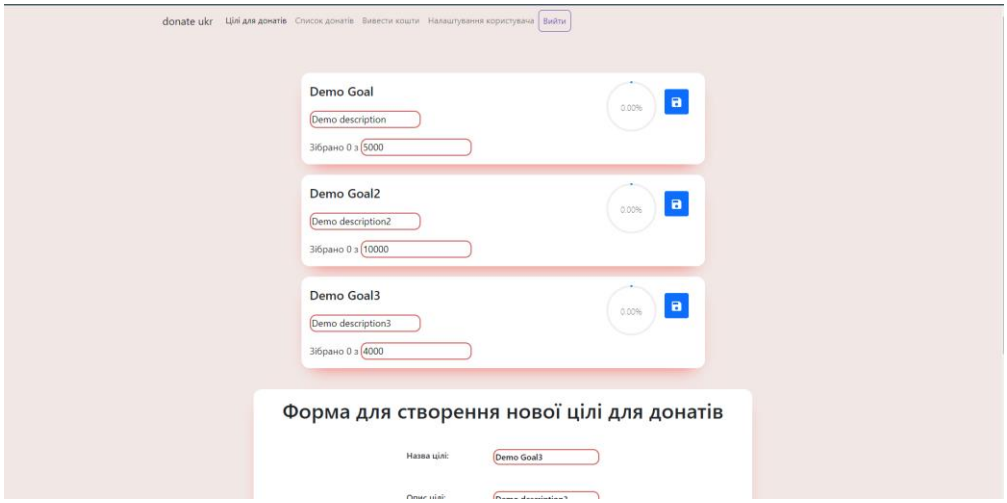


Рисунок 3.10 – Вигляд сторінки коли користувач має декілька цілей

Сторінка з налаштуваннями користувача містить посилання на сторінку донацій користувача. Посилання на сторінку донацій користувача складається з статичної частини та унікального ідентифікатора користувача. На цій сторінці користувач може ввести нове значення в поле з унікальним ідентифікатором користувача і натиснути кнопку зберегти, цим самим змінити свій унікальний ідентифікатор та своє посилання на сторінку донацій. Також на цій сторінці є кнопка яка дозволяє скопіювати своє посилання на сторінку донацій, щоб мати зручну можливість поділитися ним з іншими користувачами.

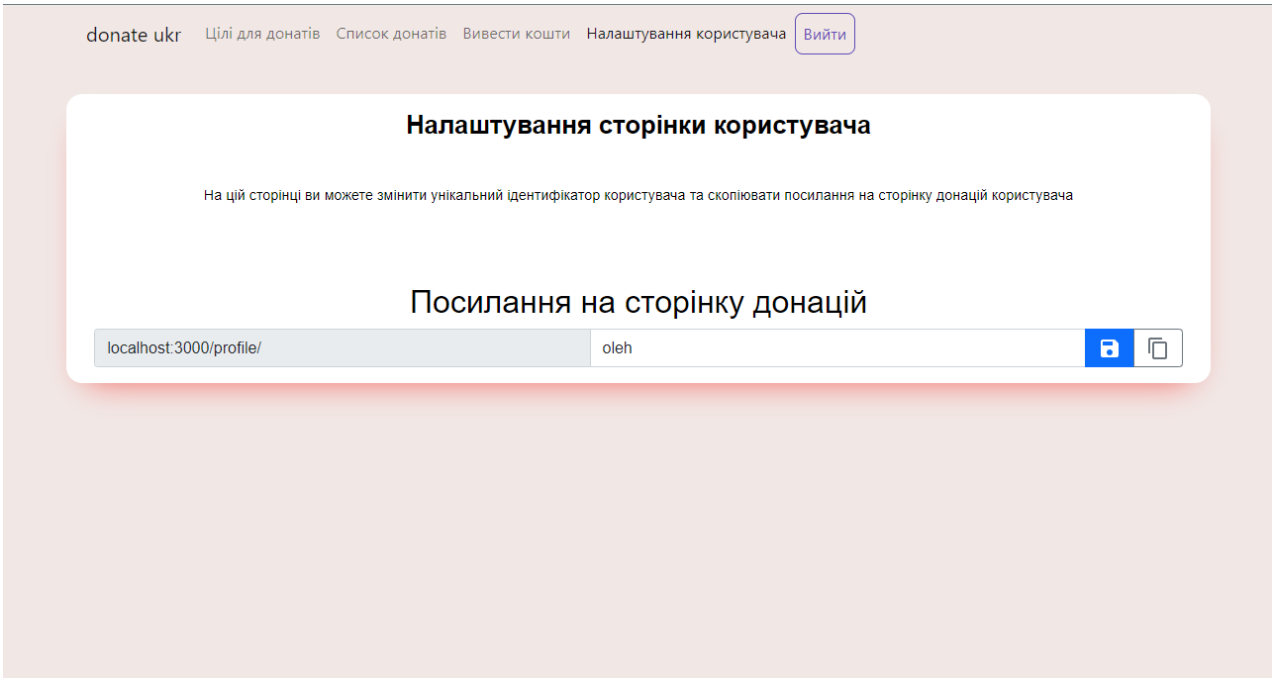


Рисунок 3.11 – Сторінка з налаштуваннями користувача

Якщо перейти по посиланню на сторінку донацій користувача, то побачимо сторінку з усіма цілями для донатів відповідного користувача та формою для здійснення донатів на обрану ціль користувача. Кожна ціль для донатів містить такі дані: назва цілі, опис цілі, загальна сума збору та поточна сума збору.

Форма для здійснення донатів містить 4 обов’язкові поля: ім’я автору донату, повідомлення донату, сума донату та ціль на яку робиться донат. Для того щоб здійснити донат, потрібно заповнити всі поля форми та обрати з списку ціль на яку потрібно зробити донат. Після цього натиснути кнопку зробити донат.

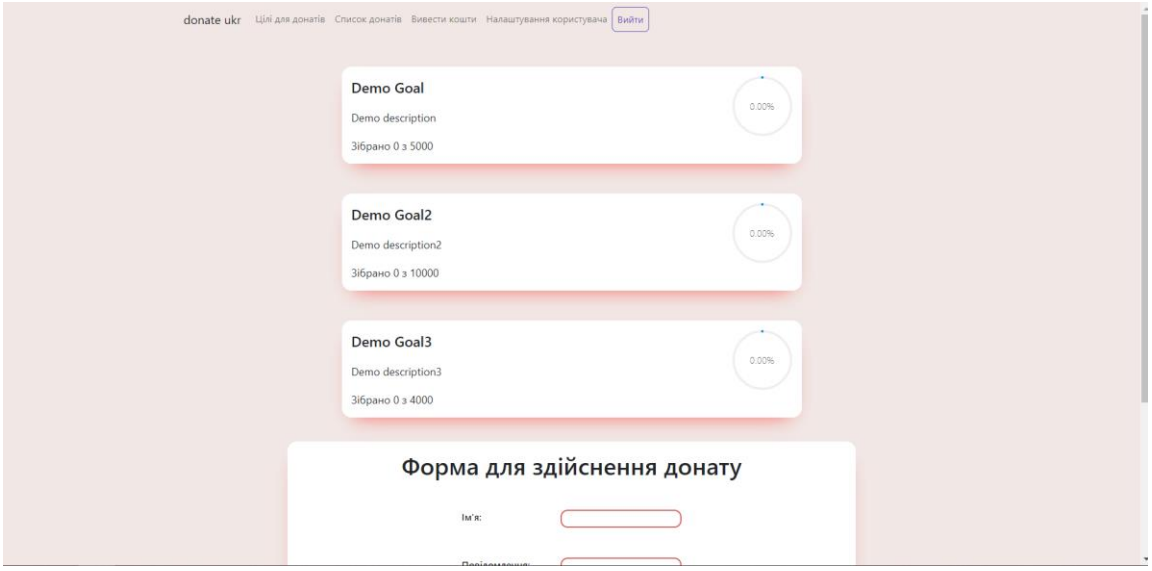


Рисунок 3.12 – Сторінка з цілями для донату конкретного користувача

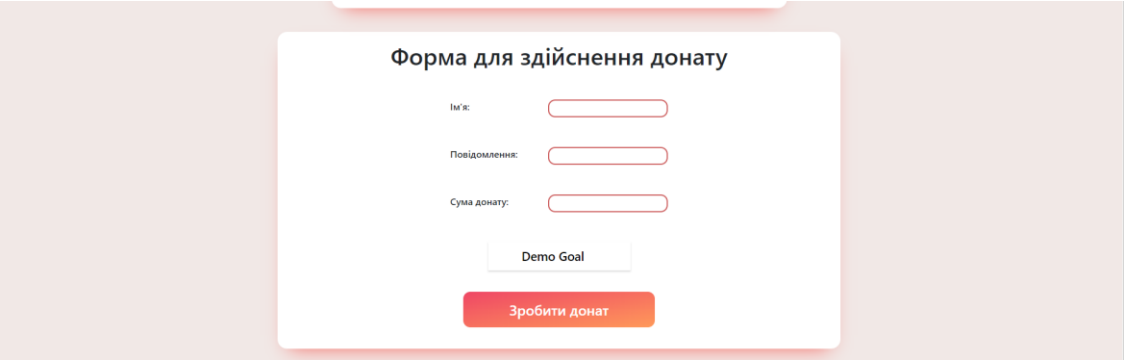


Рисунок 3.13 – Форма для здійснення донату

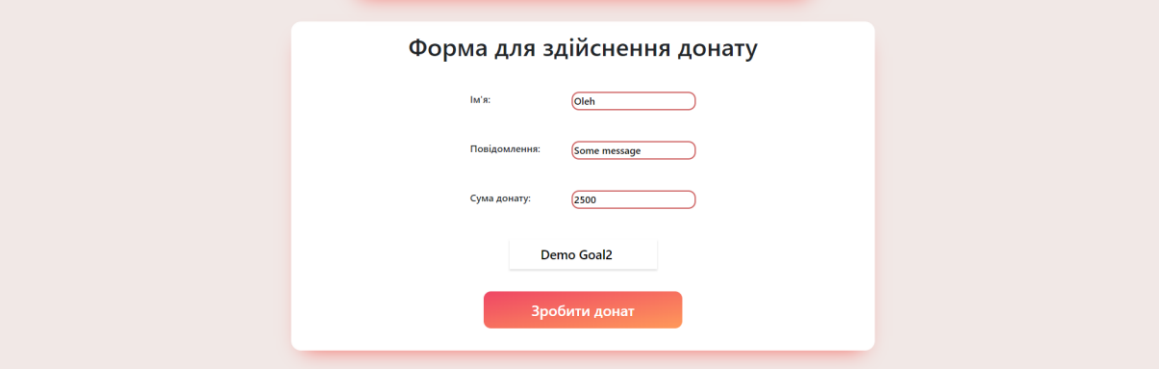


Рисунок 3.14 – Форма для здійснення донату із заповненими даними

Після натискання кнопки у новій вкладці браузеру відкриється сторінка платіжної системи Fondy з формою для оплати.

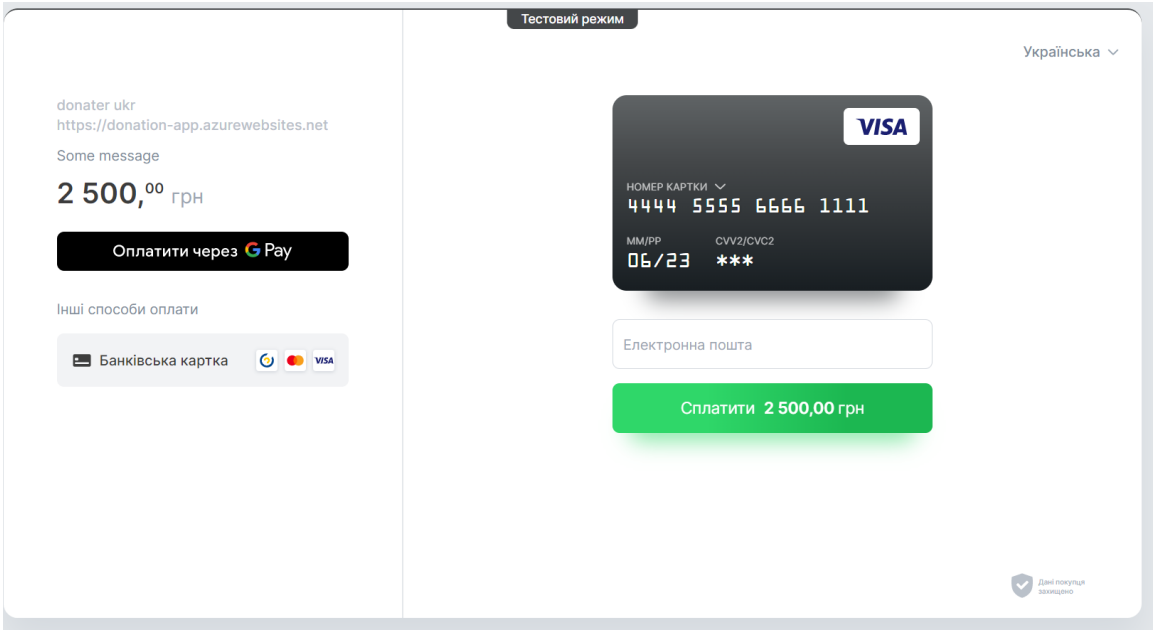


Рисунок 3.15 – Сторінка для оплати

Після оплати на сторінці відображається відповідне повідомлення з інформацією про платіж.

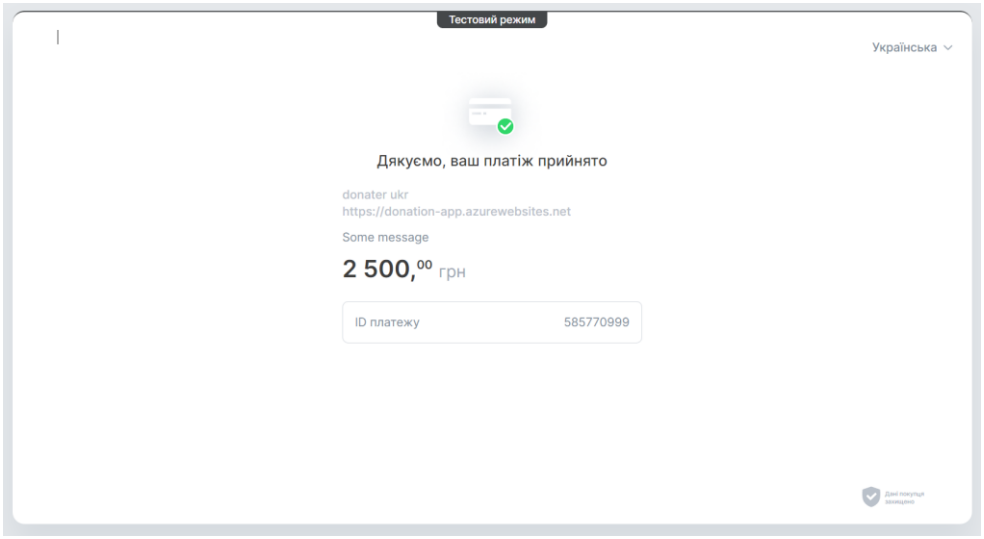


Рисунок 3.16 – Сторінка з успішним платежом

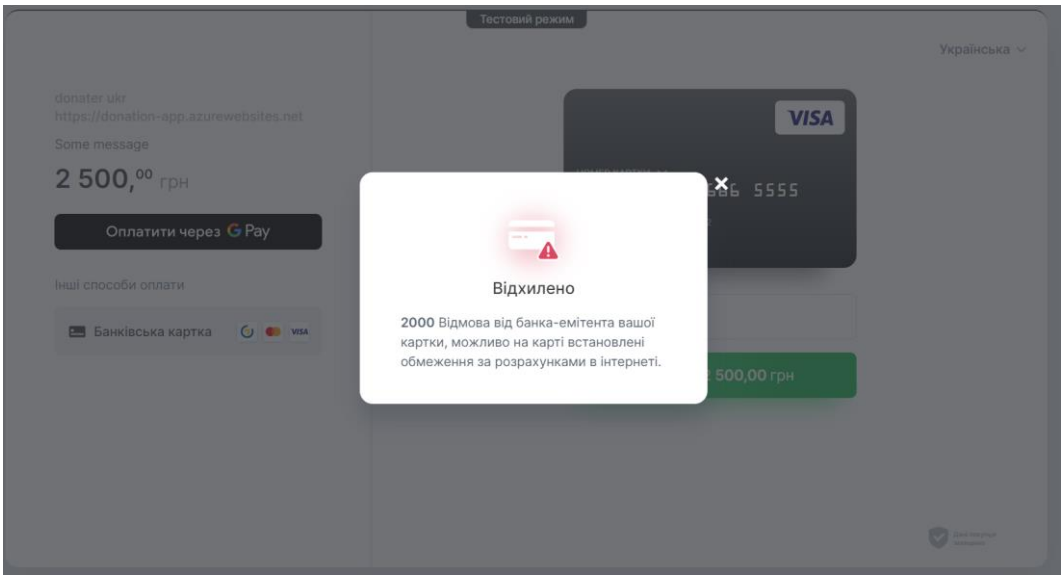


Рисунок 3.17 – Сторінка з не успішним платежом

Якщо платіж пройшов успішно, то поточна сума збору для цілі, на яку здійснювався донат, збільшується на суму донату.

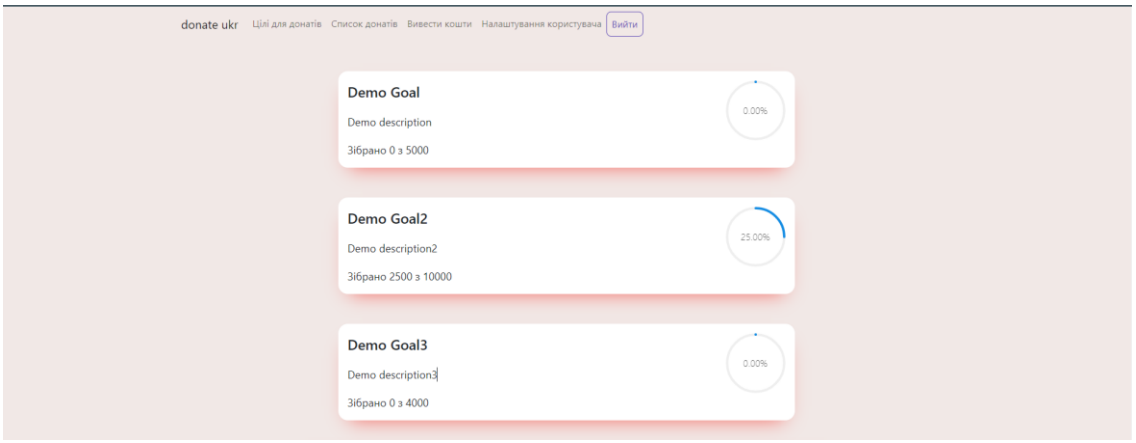


Рисунок 3.18 – Сторінка з цілями для донатів користувача

Сторінка для виводу коштів містить список карток з доступними сумами для виводу.

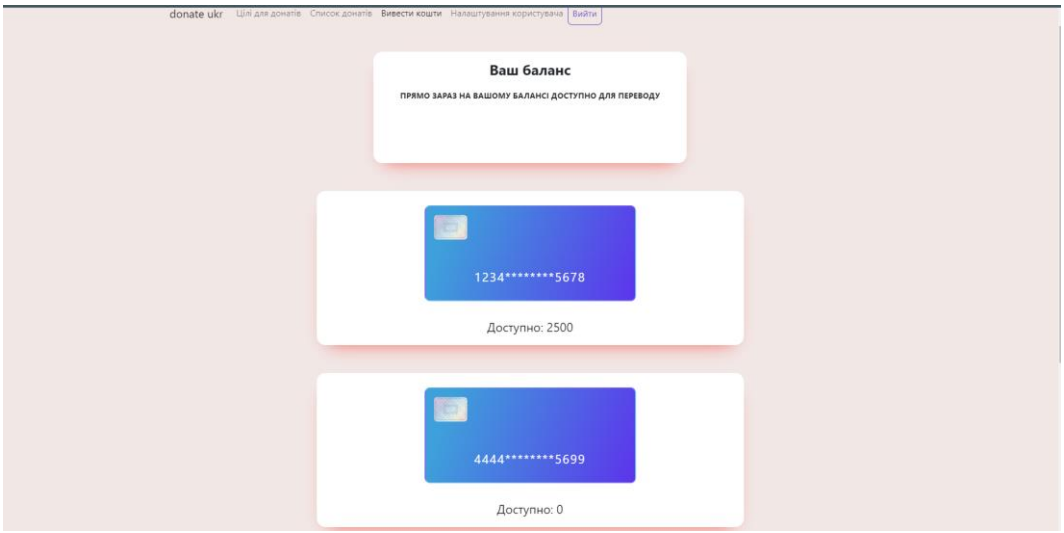


Рисунок 3.19 – Сторінка для виводу коштів

Внизу сторінки знаходиться форма для створення запиту на виведення коштів. Форма містить два поля: номер карти для виводу, який обирається з доступних карток та сума для виводу. Щоб створити запит на виведення коштів потрібно заповнити відповідні поля форми і натиснути кнопку створити запит.

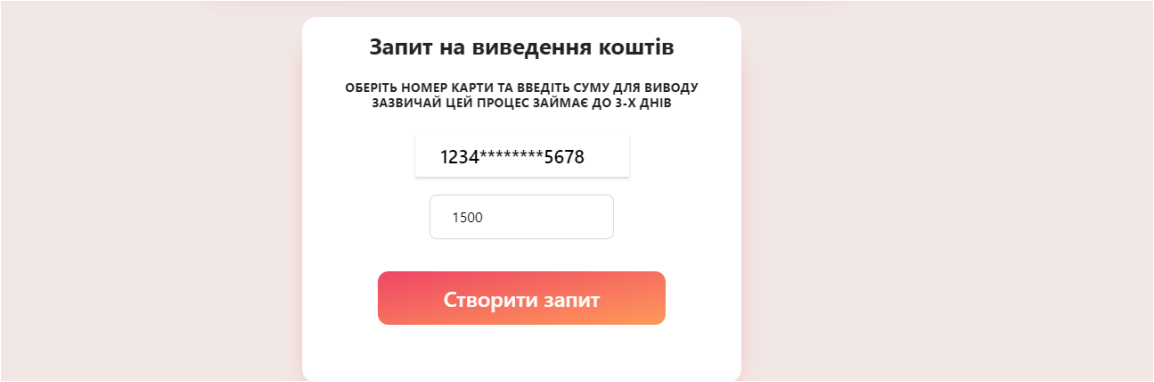


Рисунок 3.20 – Форма для виводу коштів

Після того як запит буде створено, доступний баланс на картці буде оновлено.

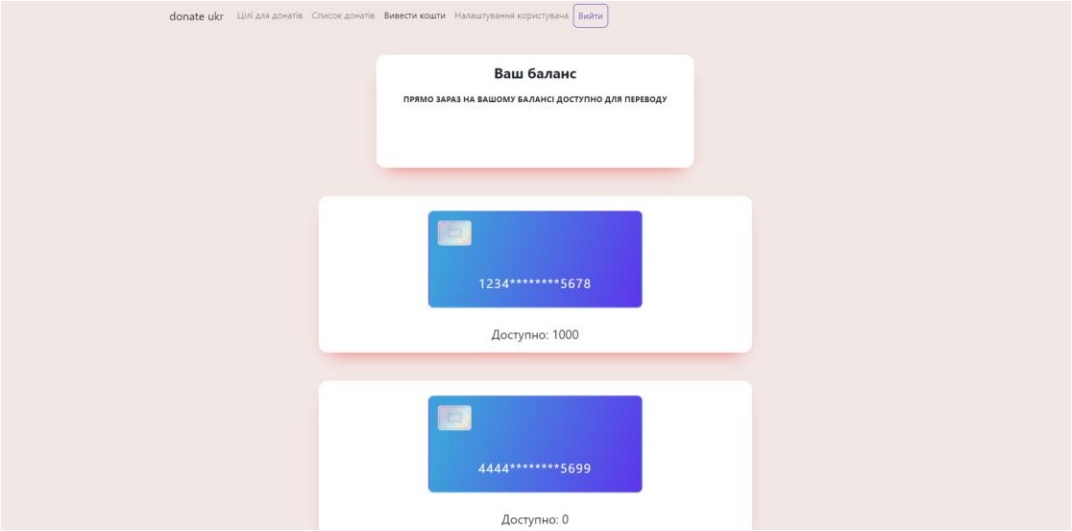


Рисунок 3.21 – Сторінка для виводу коштів

Висновки до розділу

У даному розділі було проведено аналіз якості системи, використовуючи два підходи: статичний аналіз коду та мануальне тестування.

Результати статичного аналізу коду показали задовільні результати. Цей підхід дозволив виявити слабкі місця системи та визначити напрямки подальшої роботи для виправлення технічного боргу та покращення параметрів системи.

Також система пройшла успішно всі основні мануальні сценарії, що свідчить про її відповідність бізнес-вимогам. Усі необхідні сценарії були описані та перевірені в системі.

Після завершення тестування було зроблено опис поточного стану системи, на якому відбувалося тестування. Був представлений основний поточний функціонал системи, який був підданий тестуванню разом з усіма необхідними зображеннями.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Для розгортання веб-сервісу було використано Azure App Service, який надає зручну платформу для розміщення та керування веб-додатками. Для автоматизації процесу розгортання було використано CI/CD (Continuous Integration/Continuous Deployment) засоби GitHub Actions.

GitHub Actions надає потужні можливості для автоматизації робочих процесів, у тому числі для забезпечення неперервної поставки програмного забезпечення. За допомогою GitHub Actions був створений та налаштований робочий процес, який включає кроки компіляції, тестування та розгортання веб-сервісу.

Після кожного змінення в репозиторії GitHub, робочий процес автоматично спрацьовує. Він компілює джереловий код, запускає тести для перевірки функціональності та якості системи, а потім проводить розгортання на Azure App Service. Цей підхід дозволяє забезпечити швидке та автоматизоване впровадження оновлень та змін до веб-сервісу.

Розгортання в Azure App Service надає багато переваг, включаючи масштабованість, високу доступність та зручний інтерфейс для керування сервісом. Також, завдяки використанню CI/CD GitHub Actions, процес розгортання стає автоматизованим, прозорим та ефективним.

В рамках розробки веб-сервісу для донатів використовується SQL Azure база даних. SQL Azure є повноцінною хмарною платформою для управління та збереження даних, що надає високий рівень доступності, масштабованості та надійності.

SQL Azure дозволяє легко створювати та керувати базами даних, а також забезпечує можливість резервного копіювання, моніторингу та управління безпекою даних. Вона підтримує реляційну модель даних і має сумісність зі

стандартами SQL, що дозволяє використовувати звичайні SQL-запити для роботи з даними.

Висновки до розділу

В цьому розділі було описано розгортання веб-сервісу в Azure App Service за допомогою GitHub Actions, що дозволяє забезпечити швидке та автоматизоване впровадження змін та оновлень у системі.

Застосування CI/CD дозволяє прискорити розробку, знизити ризик впровадження та покращити якість програмного забезпечення. Цей підхід допомагає автоматизувати рутинні процеси, забезпечує швидкий цикл зворотного зв'язку та дозволяє командам розробників більш ефективно працювати над проектом.

Використання SQL Azure бази даних забезпечує централізоване та надійне зберігання даних, що використовуються в системі для донатів. Це дозволяє забезпечити постійний доступ до даних, їх безпеку та можливість масштабування бази даних залежно від потреб системи.

ВИСНОВКИ

У цьому дипломному проекті було спроектовано архітектурне рішення та створено програмне забезпечення, яке призначене для здійснення благодійних внески та пожертвуввань на підтримку різних благодійних організацій та проектів.

Було виконано всі задачі, які було поставлено для досягнення мети роботи і було створено веб-серів, який відповідає всім описаним вимогам.

В результаті виконання дипломного проєкту було спроєктовано та реалізовано веб-сервіс для донатів.

Цей веб-сервіс надає зручний та безпечний процес збору та розподілу благодійних коштів для підтримки благодійних організацій та проектів, забезпечуючи прозорість та ефективність.

В якості основної технології розробки було обрано мову C# та середовище розробки Rider.

У якості БД використано SQL Azure. Застосування SQL Azure бази даних у проекті допомагає забезпечити ефективне та надійне управління даними, необхідними для функціонування веб-сервісу для донатів.

Під час проектування архітектури веб-сервісу для донатів було враховано різноманітні атрибути якості програмного забезпечення, які є важливими для систем такого типу. З метою забезпечення цих атрибутів було обрано використання декількох архітектурних підходів.

Після завершення розробки системи було проведено тестування з метою перевірки відповідності всім функціональним вимогам. Додатково, застосувався статичний аналіз коду, який дозволив перевірити відповідність всім стандартам написання коду.

Для розгортання веб-сервісу було використано Azure App Service.

Також було здійснено автоматизацію процесу інтеграції змін у коді та неперервного розгортання змін до виробничого середовища.

Для автоматизації процесу розгортання було використано CI/CD засоби GitHub Actions.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Alex Berson Client/Server Architecture (McGraw-Hill Computer Communications Series). New York: McGraw-Hill, 1996. 8 с.

2) Пістунов І.М. Моделювання бізнес процесів: навч. посіб. Дніпро: НТУ «ДП» 2021. 23 с.

3) Kenneth Frampton Modern Architecture: A Critical History. London: Thames & Hudson 2020. 236 с.

4) Pethuru Raj, Anupama Raman, Harihara Subramanian Architectural Patterns. Birmingham: Packt Publishing, 2017. 31 с.

5) Leonard Richardson, Sam Ruby RESTful Web Services. Sebastopol, CA: O'Reilly Media, 2007. 60 с.

6) Eric Vogel Beginning Entity Framework Core 5: From Novice to Professional. New York City: Apress Berkeley, 2021. 57 с.

7) Stephen Cleary Concurrency in C# Cookbook: Asynchronous, Parallel, and Multithreaded Programming 2nd Edition. Sebastopol, CA: O'Reilly Media, 2014. 142 с.

8) Bipin Joshi Beginning SOLID Principles and Design Patterns for ASP.NET Developers. New York: Apress, 2016. 32 с.

9) Robert C. Martin Clean Code: A Handbook of Agile Software Craftsmanship. London: Pearson, 2008. 174 с.

10) Steven van Deursen and Mark Seemann Dependency Injection Principles, Practices, and Patterns. New York: Manning, 2019. 54 с.

ДОДАТОК А ЗВІТ ПОДІБНОСТІ

UNICHECK

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1015410536

Дата перевірки:
04.06.2023 11:34:26 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
04.06.2023 11:56:02 EEST

ID користувача:
76913

Назва документа: **ІТ-91_Тонкошкур_ПЗ**

Кількість сторінок: **67** Кількість слів: **8867** Кількість символів: **70942** Розмір файлу: **1.72 MB** ID файлу: **1015073853**

549 слів позначені як "вилучені" та не враховуються у підрахунку слів

13.5%
Схожість

Найбільша схожість: **9.28%** з джерелом з Бібліотеки (ID файлу: **1015060909**)

0.3% Джерела з Інтернету	5	Сторінка 69
13.5% Джерела з Бібліотеки	114	Сторінка 69

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0.01%
Вилучень

Деякі джерела вилучено автоматично (фільтри вилучення: кількість знайдених слів є меншою за **8 слів** та **0%**)

0.01% Вилучення з Інтернету	37	Сторінка 70
0.01% Вилученого тексту з Бібліотеки	104	Сторінка 70

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

Веб-сервіс для донатів

Текст програми

КПІ.ІТ-9126.045440.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Катерина ЛІЩУК

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Олег ТОНКОШКУР

Київ – 2023

Файл ICardPaymentRepository.cs

```
using DonationApp.Model;

namespace DonationApp.Data.Abstract;

public interface ICardPaymentRepository : IRepository<CardPayment, Guid>
{

}
```

Файл IDonationGoalRepository.cs

```
using DonationApp.Model;

namespace DonationApp.Data.Abstract;

public interface IDonationGoalRepository : IRepository<DonationGoal, Guid>
{
    IQueryable<DonationGoal> GetByUserId(Guid userId);
}
```

Файл IDonationRepository.cs

```
using DonationApp.Model;

namespace DonationApp.Data.Abstract;

public interface IDonationRepository : IRepository<Donation, Guid>
{
    IQueryable<Donation> GetAll();
    IQueryable<Donation> GetByUserId(Guid userId);
    IQueryable<Donation> GetUnProcessedByUserId(Guid userId);
    Task<Donation?> GetByOrderId(string orderId);
}
```

Файл IRepository.cs

```
namespace DonationApp.Data.Abstract;

public interface IRepository<T, K>
    where T : class
    where K : struct
{
    Task<T> Create(T entity);
    Task<T> GetById(K id);
```

```
    void Delete(K id);
    T Update(T entity);
    void SaveChanges();
}
```

Файл IUserRepository.cs

```
using DonationApp.Model;

namespace DonationApp.Data.Abstract;

public interface IUserRepository : IRepository<User, Guid>
{
    public Task<User?> GetUserBySubject(string subject);
    public Task<User?> GetUserBySlug(string userSlug);
    public Guid? GetUserIdBySubject(string subject);
}
```

Файл IWithdrawRequestRepository.cs

```
using DonationApp.Model;

namespace DonationApp.Data.Abstract;

public interface IWithdrawRequestRepository : IRepository<WithdrawRequest, Guid>
{
    Task<IEnumerable<WithdrawRequest>> GetUserWithdrawRequests(Guid userId);
}
```

Файл CardPaymentRepository.cs

```
using DonationApp.Data.Abstract;
using DonationApp.Model;

namespace DonationApp.Data.Repositories;

public class CardPaymentRepository : RepositoryBase<CardPayment, Guid>, ICardPaymentRepository
{
    public CardPaymentRepository(DonationAppContext context) : base(context)
    {
    }
}
```

Файл DonationGoalRepository.cs

```
using DonationApp.Data.Abstract;
using DonationApp.Model;

namespace DonationApp.Data.Repositories;

public class DonationGoalRepository : RepositoryBase<DonationGoal, Guid>, IDonationGoalRepository
{
    public DonationGoalRepository(DonationAppContext context) : base(context)
    {

    }

    public IQueryable<DonationGoal> GetByUserId(Guid userId)
    {
        return dbSet.Where(dg => dg.UserId == userId);
    }
}
```

Файл DonationRepository.cs

```
using DonationApp.Data.Abstract;
using DonationApp.Model;
using DonationApp.Model.Enums;
using Microsoft.EntityFrameworkCore;

namespace DonationApp.Data.Repositories;

public class DonationRepository : RepositoryBase<Donation, Guid>, IDonationRepository
{
    public DonationRepository(DonationAppContext context): base(context)
    { }

    public IQueryable<Donation> GetAll()
    {
        return dbSet.AsQueryable();
    }

    public IQueryable<Donation> GetByUserId(Guid userId)
    {
        return dbSet.Where(d => d.DonationGoal.UserId == userId && d.DonationStatus == DonationStatus.Success);
    }
```

```
    public IQueryable<Donation> GetUnProcessedByUserId(Guid userId)
    {
        return dbSet.Where(d => d.DonationGoal.UserId == userId && d.DonationStatus == DonationStatus.New);
    }

    public async Task<Donation?> GetByOrderId(string orderId)
    {
        return await dbSet.FirstOrDefaultAsync(d => d.Order_id.Equals(orderId)).ConfigureAwait(false);
    }
}
```

Файл RepositoryBase.cs

```
using DonationApp.Data.Abstract;
using Microsoft.EntityFrameworkCore;

namespace DonationApp.Data.Repositories;

public class RepositoryBase<T,K> : IRepository<T, K>
    where T : class
    where K : struct
{
    private readonly DonationAppContext _context;
    protected readonly DbSet<T> dbSet;

    public RepositoryBase(DonationAppContext context)
    {
        _context = context;
        dbSet = context.Set<T>();
    }
    public async Task<T> Create(T entity)
    {
        var entry = await dbSet.AddAsync(entity).ConfigureAwait(false);
        return entry.Entity;
    }

    public async Task<T> GetById(K id)
    {
        return (await dbSet.FindAsync(id).ConfigureAwait(false))!;
    }

    public async void Delete(K id)
```

```
    {
        var entity = await dbSet.FindAsync(id).ConfigureAwait(false);
        if (entity is not null)
        {
            dbSet.Remove(entity);
        }
    }

    public T Update(T entity)
    {
        return dbSet.Update(entity).Entity;
    }

    public void SaveChanges()
    {
        _context.SaveChanges();
    }
}
```

Файл UserRepository.cs

```
using DonationApp.Data.Abstract;
using DonationApp.Model;
using Microsoft.EntityFrameworkCore;

namespace DonationApp.Data.Repositories;

public class UserRepository : RepositoryBase<User, Guid>, IUserRepository
{
    public UserRepository(DonationAppContext context) : base(context)
    {

    }

    public async Task<User?> GetUserBySubject(string subject)
    {
        return await dbSet.FirstOrDefaultAsync(u => u.Subject.Equals(subject)).ConfigureAwait(false);
    }

    public async Task<User?> GetUserBySlug(string userSlug)
    {
        return await dbSet.FirstOrDefaultAsync(u => u.UserSlug.Slug.Equals(userSlug)).ConfigureAwait(false);
```

```
    }

    public Guid? GetUserIdBySubject(string subject)
    {
        return dbSet.FirstOrDefault(u => u.Subject.Equals(subject))?.Id;
    }

    public new async Task<User> GetById(Guid id)
    {
        return (await dbSet.Include(x => x.UserSlug)
            .FirstOrDefaultAsync(x => x.Id == id).ConfigureAwait(false))!;
    }
}
```

Файл WithdrawRequestRepository.cs

```
using DonationApp.Data.Abstract;
using DonationApp.Model;
using DonationApp.Model.Enums;
using Microsoft.EntityFrameworkCore;

namespace DonationApp.Data.Repositories;

public class WithdrawRequestRepository : RepositoryBase<WithdrawRequest, Guid>,
IWithdrawRequestRepository
{
    public WithdrawRequestRepository(DonationAppContext context) : base(context)
    {
    }

    public async Task<IEnumerable<WithdrawRequest>> GetUserWithdrawRequests(Guid userId)
    {
        return await dbSet.Where(x => x.UserId == userId && x.Status !=
WithdrawRequestStatus.Rejected).ToListAsync().ConfigureAwait(false);
    }
}
```

Файл DonationAppContext.cs

```
using DonationApp.Model;
using Microsoft.EntityFrameworkCore;

namespace DonationApp.Data;
```

```
public class DonationAppContext : DbContext
{
    public DbSet<Donation> Donations { get; set; }
    public DbSet<User> Users { get; set; }
    public DbSet<WithdrawRequest> WithdrawRequests { get; set; }
    public DbSet<UserSlug> UserSlugs { get; set; }
    public DbSet<DonationGoal> DonationGoals { get; set; }
    public DbSet<CardPayment> CardPayments { get; set; }

    public DonationAppContext(DbContextOptions<DonationAppContext> options)
        : base(options)
    {
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Donation>()
            .HasOne(p => p.CardPayment)
            .WithOne(b => b.Donation)
            .HasForeignKey<CardPayment>(p => p.Order_id)
            .HasPrincipalKey<Donation>(b => b.Order_id);
    }
}
```

Файл DonationStatus.cs

```
namespace DonationApp.Model.Enums;

public enum DonationStatus
{
    New,
    Failure,
    Success
}
```

Файл WithdrawRequestStatus.cs

```
namespace DonationApp.Model.Enums;

public enum WithdrawRequestStatus
{
    New,
```

```
    Rejected,
    Processed
}
```

Файл CardPayment.cs

```
namespace DonationApp.Model;

public class CardPayment
{
    public Guid Id { get; set; }
    public string Order_id { get; set; }
    public int Merchant_id { get; set; }
    public int Amount { get; set; }
    public string Currency { get; set; }
    public string Order_status { get; set; }
    public string Response_status { get; set; }
    public string Signature { get; set; }
    public string Tran_type { get; set; }
    public string Response_code { get; set; }

    public Donation Donation { get; set; }
}
```

Файл Donation.cs

```
using DonationApp.Model.Enums;

namespace DonationApp.Model;

public class Donation
{
    public Guid Id { get; set; }
    public int Amount { get; set; }
    public DateTime CreatedOn { get; set; }
    public string DonatorName { get; set; }
    public string Message { get; set; }
    public Guid DonationGoalId { get; set; }
    public string Order_id { get; set; }
    public DonationStatus DonationStatus { get; set; }

    public DonationGoal DonationGoal { get; set; }
    public CardPayment CardPayment { get; set; }
```

```
}
```

Файл DonationGoal.cs

```
namespace DonationApp.Model;

public class DonationGoal
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public string CardNumber { get; set; }
    public int Amount { get; set; }
    public Guid UserId { get; set; }
    public User User { get; set; }
}
```

Файл User.cs

```
namespace DonationApp.Model;

public class User
{
    public Guid Id { get; set; }
    public string Subject { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    public DateTime CreatedOn { get; set; }
    public UserSlug UserSlug { get; set; }
    public ICollection<DonationGoal> DonationGoals { get; set; }
}
```

Файл UserSlug.cs

```
namespace DonationApp.Model;

public class UserSlug
{
    public Guid Id { get; set; }
    public string Slug { get; set; }
    public Guid UserId { get; set; }
}
```

Файл WithdrawRequest.cs

```
using DonationApp.Model.Enums;

namespace DonationApp.Model;

public class WithdrawRequest
{
    public Guid Id { get; set; }
    public Guid UserId { get; set; }
    public string CardNumber { get; set; }
    public int Amount { get; set; }
    public DateTime CreatedOn { get; set; }
    public WithdrawRequestStatus Status { get; set; }

    public User User { get; set; }
}
```

Файл IDonationGoalService.cs

```
using DonationApp.Service.Dtos;

namespace DonationApp.Service.Abstract;

public interface IDonationGoalService
{
    Task<DonationGoalResponseDto> Create(DonationGoalRequestDto donationGoalDto);
    Task<DonationGoalResponseDto> Update(DonationGoalUpdateDto donationGoalDto);
    Task<IEnumerable<DonationGoalResponseDto>> GetByUser(string userSlug);
}
```

Файл IDonationService.cs

```
using DonationApp.Model;
using DonationApp.Service.Dtos;

namespace DonationApp.Service.Abstract;

public interface IDonationService
{
    Task<IList<Donation>> GetByFilter(DonationFilter filter);
    Task<IList<Donation>> GetByUserId(Guid userId);
}
```

Файл IPaymentService.cs

```
using CloudIpspSDK.Models;
using DonationApp.Service.Dtos;

namespace DonationApp.Service.Abstract;

public interface IPaymentService
{
    public Task<string> GenerateDonationLink(DonationDto dto);
    public Task ProcessPaymentCallback(ResponseModel paymentResponse);
}
```

Файл IUserService.cs

```
using DonationApp.Service.Dtos;

namespace DonationApp.Service.Abstract;

public interface IUserService
{
    public Task<UserDto?> CreateIfNotExist(string tokenId);
    public Task<bool> SetUserSlug(UserSlugDto userSlugDto);
    public Task<string> GetUserSlug(Guid userId);
}
```

Файл IWithdrawService.cs

```
using DonationApp.Service.Dtos;

namespace DonationApp.Service.Abstract;

public interface IWithdrawService
{
    Task<IEnumerable<AvailableWithdrawAmountResponse>> GetUserAvailableWithdrawAmount(Guid userId);
    Task<WithdrawRequestDto> CreateWithdrawRequest(WithdrawRequestDto withdrawRequestDto);
}
```

Файл AvailableWithdrawAmountResponse.cs

```
namespace DonationApp.Service.Dtos;
```

```
public class AvailableWithdrawAmountResponse
{
    public string CardNumber { get; set; }
    public int Amount { get; set; }
}
```

Файл DonationDto.cs

```
namespace DonationApp.Service.Dtos;

public class DonationDto
{
    public int Amount { get; set; }
    public string DonatorName { get; set; }
    public string Message { get; set; }
    public Guid DonationGoalId { get; set; }
}
```

Файл DonationFilter.cs

```
namespace DonationApp.Service.Dtos;

public class DonationFilter
{
    public int Skip { get; set; } = 0;
    public int Take { get; set; } = 0;
}
```

Файл DonationGoalRequestDto.cs

```
namespace DonationApp.Service.Dtos;

public class DonationGoalRequestDto
{
    public string Name { get; set; }
    public string Description { get; set; }
    public string CardNumber { get; set; }
    public int Amount { get; set; }
    public Guid UserId { get; set; }
}
```

Файл DonationGoalResponseDto.cs

```
namespace DonationApp.Service.Dtos;

public class DonationGoalResponseDto
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public string CardNumber { get; set; }
    public int Amount { get; set; }
    public int CurrentAmount { get; set; }
}
```

Файл DonationGoalUpdateDto.cs

```
namespace DonationApp.Service.Dtos;

public class DonationGoalUpdateDto
{
    public Guid Id { get; set; }
    public string Description { get; set; }
    public int Amount { get; set; }
    public Guid UserId { get; set; }
}
```

Файл Response.cs

```
namespace DonationApp.Service.Dtos;

public class Response
{
    public string Order_id { get; set; }
    public int Merchant_id { get; set; }
    public int Amount { get; set; }
    public string Currency { get; set; }
    public string Order_status { get; set; }
    public string Response_status { get; set; }
    public string Signature { get; set; }
    public string Tran_type { get; set; }
    public string Response_code { get; set; }
}
```

Файл UserDto.cs

```
namespace DonationApp.Service.Dtos;
```

```
public class UserDto
{
   public string Name { get; set; }
   public string Email { get; set; }
}
```

Файл UserIn.cs

```
namespace DonationApp.Service.Dtos;

public class UserIn
{
   public string TokenId { get; set; }
}
```

Файл UserSlugDto.cs

```
namespace DonationApp.Service.Dtos;

public class UserSlugDto
{
   public string Slug { get; set; }
   public Guid UserId { get; set; }
}
```

Файл WithdrawRequestDto.cs

```
namespace DonationApp.Service.Dtos;

public class WithdrawRequestDto
{
   public Guid Id { get; set; }
   public Guid UserId { get; set; }
   public string CardNumber { get; set; }
   public int Amount { get; set; }
}
```

Файл CardPaymentProfile.cs

```
using AutoMapper;
using CloudIpspSDK.Models;
using DonationApp.Model;
```

```
namespace DonationApp.Service.Profiles;

public class CardPaymentProfile : Profile
{
   public CardPaymentProfile()
   {
      CreateMap<ResponseModel, CardPayment>();
   }
}
```

Файл DonationGoalProfile.cs

```
using AutoMapper;
using DonationApp.Model;
using DonationApp.Service.Dtos;

namespace DonationApp.Service.Profiles;

public class DonationGoalProfile : Profile
{
   public DonationGoalProfile()
   {
      CreateMap<DonationGoalRequestDto, DonationGoal>();

      CreateMap<DonationGoal, DonationGoalResponseDto>();

      CreateMap<DonationGoalUpdateDto, DonationGoal>();
   }
}
```

Файл DonationProfile.cs

```
using AutoMapper;
using DonationApp.Model;
using DonationApp.Service.Dtos;

namespace DonationApp.Service.Profiles;

public class DonationProfile : Profile
{
   public DonationProfile()
   {
      CreateMap<DonationDto, Donation>()
```

```
            .ForMember(
                dest => dest.CreatedOn,
                opt => opt.MapFrom(src => DateTime.Now)
            );
        CreateMap<Donation, DonationDto>();
    }
}
```

Файл UserProfile.cs

```
using AutoMapper;
using DonationApp.Model;
using DonationApp.Service.Dtos;

namespace DonationApp.Service.Profiles;

public class UserProfile : Profile
{
    public UserProfile()
    {
        CreateMap<User, UserDto>();
        CreateMap<UserSlugDto, UserSlug>();
    }
}
```

Файл WithdrawRequestProfile.cs

```
using AutoMapper;
using DonationApp.Model;
using DonationApp.Model.Enums;
using DonationApp.Service.Dtos;

namespace DonationApp.Service.Profiles;

public class WithdrawRequestProfile : Profile
{
    public WithdrawRequestProfile()
    {
        CreateMap<WithdrawRequestDto, WithdrawRequest>()
            .ForMember(dest => dest.User, opt=> opt.Ignore())
            .ForMember(
                dest => dest.CreatedOn,
```

```
            opt => opt.MapFrom(src => DateTime.Now)
        )
        .ForMember(
            dest => dest.Status,
            opt => opt.MapFrom(src => WithdrawRequestStatus.New)
        );
    CreateMap<WithdrawRequest, WithdrawRequestDto>()
        .ForMember(dest => dest.UserId, opt=> opt.Ignore());
  }
}
```

Файл DonationGoalService.cs

```
using AutoMapper;
using DonationApp.Data.Abstract;
using DonationApp.Model;
using DonationApp.Service.Abstract;
using DonationApp.Service.Dtos;
using Microsoft.EntityFrameworkCore;

namespace DonationApp.Service.Services;

public class DonationGoalService : IDonationGoalService
{
  private readonly IDonationGoalRepository _donationGoalRepository;
  private readonly IDonationRepository _donationRepository;
  private readonly IUserRepository _userRepository;
  private readonly IMapper _mapper;

  public DonationGoalService(
    IDonationGoalRepository donationGoalRepository,
    IDonationRepository donationRepository,
    IUserRepository userRepository,
    IMapper mapper)
  {
    _donationGoalRepository = donationGoalRepository;
    _donationRepository = donationRepository;
    _userRepository = userRepository;
    _mapper = mapper;
  }

  public async Task<DonationGoalResponseDto> Create(DonationGoalRequestDto donationGoalDto)
```

```
    {
        var created = await
_donationGoalRepository.Create(_mapper.Map<DonationGoal>(donationGoalDto)).ConfigureAwait(false);

        _donationGoalRepository.SaveChanges();

        return _mapper.Map<DonationGoalResponseDto>(created);
    }

    public async Task<DonationGoalResponseDto> Update(DonationGoalUpdateDto donationGoalDto)
    {
        var entityToUpdate = await _donationGoalRepository.GetById(donationGoalDto.Id).ConfigureAwait(false);

        entityToUpdate.Description = donationGoalDto.Description;
        entityToUpdate.Amount = donationGoalDto.Amount;

        _donationGoalRepository.Update(entityToUpdate);
        _donationGoalRepository.SaveChanges();

        return _mapper.Map<DonationGoalResponseDto>(entityToUpdate);
    }

    public async Task<IEnumerable<DonationGoalResponseDto>> GetByUser(string userSlug)
    {
        var user = await _userRepository.GetUserBySlug(userSlug).ConfigureAwait(false);
        if (user == null)
        {
            return Array.Empty<DonationGoalResponseDto>();
        }

        var donationGoals =  await
_donationGoalRepository.GetByUserId(user.Id).ToListAsync().ConfigureAwait(false);
        var donations = await _donationRepository.GetByUserId(user.Id).ToListAsync().ConfigureAwait(false);
        var currentDonationGoalAmounts = new Dictionary<Guid, int>();
        foreach (var donation in donations)
        {
            if (currentDonationGoalAmounts.ContainsKey(donation.DonationGoalId))
            {
                currentDonationGoalAmounts[donation.DonationGoalId] += donation.Amount;
            }
            else
            {
```

```
                currentDonationGoalAmounts.Add(donation.DonationGoalId, donation.Amount);
            }
        }

        return donationGoals.Select(x => new DonationGoalResponseDto
        {
            Id = x.Id,
            Name = x.Name,
            Description = x.Description,
            Amount = x.Amount,
            CardNumber = x.CardNumber,
            CurrentAmount = currentDonationGoalAmounts.ContainsKey(x.Id) ? currentDonationGoalAmounts[x.Id] :
0
        });
    }
}
```

Файл DonationService.cs

```
using AutoMapper;
using DonationApp.Data.Abstract;
using DonationApp.Model;
using DonationApp.Service.Abstract;
using DonationApp.Service.Dtos;
using Microsoft.EntityFrameworkCore;

namespace DonationApp.Service.Services;

public class DonationService : IDonationService
{
    private readonly IDonationRepository _donationRepository;

    public DonationService(
        IDonationRepository donationRepository)
    {
        _donationRepository = donationRepository;
    }

    public async Task<IList<Donation>> GetByFilter(DonationFilter filter)
    {
        var donations = _donationRepository.GetAll();
        var skip = filter.Skip;
```

```
        var take = filter.Take;

        if (skip > 0)
        {
            donations = donations.Skip(skip);
        }

        if (take > 0)
        {
            donations = donations.Take(take);
        }

        return await donations.ToListAsync();
    }

    public async Task<IList<Donation>> GetByUserId(Guid userId)
    {
        return await _donationRepository.GetByUserId(userId).ToListAsync().ConfigureAwait(false);;
    }
}
```

Файл PaymentService.cs

```
using AutoMapper;
using CloudIpspSDK.Checkout;
using CloudIpspSDK.Models;
using DonationApp.Data.Abstract;
using DonationApp.Model;
using DonationApp.Model.Enums;
using DonationApp.Service.Abstract;
using DonationApp.Service.Dtos;
using Microsoft.Extensions.Logging;

namespace DonationApp.Service.Services;

public class PaymentService : IPaymentService
{
    private readonly IDonationRepository _donationRepository;
    private readonly ICardPaymentRepository _cardPaymentRepository;
    private readonly IDonationGoalRepository _donationGoalRepository;
    private readonly IMapper _mapper;
    private readonly ILogger<PaymentService> _logger;
```

```
    public PaymentService(
        IDonationRepository donationRepository,
        ICardPaymentRepository cardPaymentRepository,
        IDonationGoalRepository donationGoalRepository,
        IMapper mapper,
        ILogger<PaymentService> logger)
    {
        _donationRepository = donationRepository;
        _cardPaymentRepository = cardPaymentRepository;
        _donationGoalRepository = donationGoalRepository;
        _mapper = mapper;
        _logger = logger;
    }

    public async Task<string> GenerateDonationLink(DonationDto donationDto)
    {
        var donationGoal = await
_donationGoalRepository.GetById(donationDto.DonationGoalId).ConfigureAwait(false);
        if (donationGoal == null)
        {
            throw new ArgumentException("Invalid donation goal");
        }

        var donation = _mapper.Map<Donation>(donationDto);
        donation.Order_id = Guid.NewGuid().ToString("N");

        await _donationRepository.Create(donation).ConfigureAwait(false);
        _donationRepository.SaveChanges();

        var request = new CheckoutRequest {
            order_id = donation.Order_id,
            amount = donation.Amount * 100,
            order_desc = donation.Message,
            currency = "UAH",
            sender_first_name = donation.DonatorName
        };

        var response = new Url().Post(request);
        if (response.Error == null)
        {
            return response.checkout_url;
```

```
        }

        throw new Exception(response.Error.ErrorMessage);
    }

    public async Task ProcessPaymentCallback(ResponseModel responseModel)
    {
        var donation = await _donationRepository.GetByOrderId(responseModel.order_id).ConfigureAwait(false);
        if (donation == null)
        {
            return;
        }

        var cardPayment = _mapper.Map<CardPayment>(responseModel);
        if (cardPayment.Response_status == "success" && cardPayment.Amount == donation.Amount * 100)
        {
            donation.DonationStatus = DonationStatus.Success;
        }
        else
        {
            donation.DonationStatus = DonationStatus.Failure;
        }

        await _cardPaymentRepository.Create(cardPayment).ConfigureAwait(false);
        _donationRepository.Update(donation);
        _donationRepository.SaveChanges();
    }
}
```

Файл UserService.cs

```
using AutoMapper;
using DonationApp.Data.Abstract;
using DonationApp.Model;
using DonationApp.Service.Abstract;
using DonationApp.Service.Dtos;
using Google.Apis.Auth;

namespace DonationApp.Service.Services;

public class UserService : IUserService
{
    private readonly IUserRepository _userRepository;
```

```
private readonly IMapper _mapper;

public UserService(
    IUserRepository userRepository,
    IMapper mapper)
{
    _userRepository = userRepository;
    _mapper = mapper;
}

public async Task<UserDto?> CreateIfNotExist(string tokenId)
{
    var payload = GoogleJsonWebSignature.ValidateAsync(tokenId).Result;
    var user = await _userRepository.GetUserBySubject(payload.Subject).ConfigureAwait(false);
    if (user != null)
    {
        return _mapper.Map<UserDto>(user);
    }

    var newUserId = Guid.NewGuid();
    var newUser = await _userRepository.Create(new User
    {
        Id = newUserId,
        Email = payload.Email,
        Name = payload.Name,
        CreatedOn = DateTime.Now,
        Subject = payload.Subject,
        UserSlug = new UserSlug
        {
            UserId = newUserId,
            Slug = Guid.NewGuid().ToString()
        }
    }).ConfigureAwait(false);

    _userRepository.SaveChanges();

    return _mapper.Map<UserDto>(newUser);
}

public async Task<bool> SetUserSlug(UserSlugDto userSlugDto)
{
    var user = await _userRepository.GetUserBySlug(userSlugDto.Slug).ConfigureAwait(false);
```

```
        if (user != null)
        {
            return false;
        }

        user = await _userRepository.GetById(userSlugDto.UserId).ConfigureAwait(false);
        user.UserSlug = _mapper.Map<UserSlug>(userSlugDto);

        _userRepository.Update(user);
        _userRepository.SaveChanges();

        return true;
    }

    public async Task<string> GetUserSlug(Guid userId)
    {
        var user = await _userRepository.GetById(userId).ConfigureAwait(false);
        return user.UserSlug.Slug;
    }
}
```

Файл WithdrawService.cs

```
using AutoMapper;
using DonationApp.Data.Abstract;
using DonationApp.Model;
using DonationApp.Service.Abstract;
using DonationApp.Service.Dtos;

namespace DonationApp.Service.Services;

public class WithdrawService : IWithdrawService
{
    private readonly IWithdrawRequestRepository _withdrawRequestRepository;
    private readonly IDonationGoalService _donationGoalService;
    private readonly IUserRepository _userRepository;
    private readonly IMapper _mapper;

    public WithdrawService(
        IWithdrawRequestRepository withdrawRequestRepository,
        IDonationGoalService donationGoalService,
        IUserRepository userRepository,
        IMapper mapper)
    {
```

```
        _withdrawRequestRepository = withdrawRequestRepository;
        _donationGoalService = donationGoalService;
        _userRepository = userRepository;
        _mapper = mapper;
    }

    public async Task<IEnumerable<AvailableWithdrawAmountResponse>>
GetUserAvailableWithdrawAmount(Guid userId)
    {
        var user = await _userRepository.GetById(userId).ConfigureAwait(false);
        var donationGoals = await _donationGoalService.GetByUser(user.UserSlug.Slug).ConfigureAwait(false);
        var cardNumberAvailableAmount = new Dictionary<string, int>();
        foreach (var donationGoal in donationGoals)
        {
            if (cardNumberAvailableAmount.ContainsKey(donationGoal.CardNumber))
            {
                cardNumberAvailableAmount[donationGoal.CardNumber] += donationGoal.CurrentAmount;
            }
            else
            {
                cardNumberAvailableAmount.Add(donationGoal.CardNumber, donationGoal.CurrentAmount);
            }
        }

        var processedWithdrawRequests = await
_withdrawRequestRepository.GetUserWithdrawRequests(userId).ConfigureAwait(false);
        foreach (var processedWithdrawRequest in processedWithdrawRequests)
        {
            cardNumberAvailableAmount[processedWithdrawRequest.CardNumber] -=
processedWithdrawRequest.Amount;
        }

        return cardNumberAvailableAmount.Select(x => new AvailableWithdrawAmountResponse
        {
            CardNumber = x.Key,
            Amount = x.Value
        });
    }

    public async Task<WithdrawRequestDto> CreateWithdrawRequest(WithdrawRequestDto withdrawRequestDto)
    {
```

```
        var availableWithdrawAmount = await
GetUserAvailableWithdrawAmount(withdrawRequestDto.UserId).ConfigureAwait(false);
        var availableWithdrawAmountForCardNumber =
            availableWithdrawAmount.FirstOrDefault(x => x.CardNumber == withdrawRequestDto.CardNumber);
        if (availableWithdrawAmountForCardNumber == null || availableWithdrawAmountForCardNumber.Amount <
withdrawRequestDto.Amount)
        {
            throw new ArgumentException("Not enough amount");
        }

        withdrawRequestDto.Id = Guid.NewGuid();
        var result = await
_withdrawRequestRepository.Create(_mapper.Map<WithdrawRequest>(withdrawRequestDto)).ConfigureAwait(fal
se);

        _withdrawRequestRepository.SaveChanges();

        return _mapper.Map<WithdrawRequestDto>(result);
    }
}
```

Файл GoogleAuthorizeAttribute.cs

```
using Google.Apis.Auth;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;
using DonationApp.Data.Abstract;

namespace DonationApp.WebApi.Attributes
{
    /// <summary>
    /// Custom Google Authentication authorize attribute which validates the bearer token.
    /// </summary>
    public class GoogleAuthorizeAttribute : TypeFilterAttribute
    {
        public GoogleAuthorizeAttribute() : base(typeof(GoogleAuthorizeFilter)) { }
    }



    public class GoogleAuthorizeFilter : IAuthorizationFilter
    {
        private readonly IUserRepository _userRepository;
```

```
        public GoogleAuthorizeFilter(IUserRepository userRepository)
        {
            _userRepository = userRepository;
        }

        public void OnAuthorization(AuthorizationFilterContext context)
        {
            try
            {
                // Verify Authorization header exists
                var headers = context.HttpContext.Request.Headers;
                if (!headers.ContainsKey("Authorization"))
                {
                    context.Result = new UnauthorizedResult();
                }
                var authHeader = headers["Authorization"].ToString();

                // Verify authorization header starts with bearer and has a token
                if (!authHeader.StartsWith("Bearer ") && authHeader.Length > 7)
                {
                    context.Result = new UnauthorizedResult();
                }

                // Grab the token and verify through google. If verification fails, and exception will be thrown.
                var token = authHeader.Remove(0, 7);
                var validated = GoogleJsonWebSignature.ValidateAsync(token).Result;

                context.HttpContext.Items.Add("userId", _userRepository.GetUserIdBySubject(validated.Subject));
            }
            catch (Exception)
            {
                context.Result = new UnauthorizedResult();
            }
        }
    }
}
```

Файл UserControllerBase.cs

```
using Microsoft.AspNetCore.Mvc;
```

```
namespace DonationApp.WebApi.Controllers.Abstract;

public class UserControllerBase : ControllerBase
{
    protected Guid GetUserId()
    {
        HttpContext.Items.TryGetValue("userId", out var userId);
        return (Guid)userId!;
    }
}
```

Файл AuthorizeController.cs

```
using DonationApp.Service.Abstract;
using DonationApp.Service.Dtos;
using Microsoft.AspNetCore.Mvc;

namespace DonationApp.WebApi.Controllers;

[ApiController]
[Route("api/[controller]/[action]")]
public class AuthorizeController : ControllerBase
{
    private readonly IUserService _userService;

    public AuthorizeController(IUserService userService)
    {
        _userService = userService;
    }

    [HttpPost]
    public async Task<IActionResult> CreateIfNotExist([FromBody] UserIn userIn)
    {
        try
        {
            if (userIn == null)
                throw new ArgumentNullException(nameof(userIn));

            var user = await _userService.CreateIfNotExist(userIn.TokenId).ConfigureAwait(false);

            return Ok(user);
        }
```

```
        catch (Exception ex)
        {
            return BadRequest(ex.Message);
        }
    }
}
```

Файл DonationController.cs

```
using DonationApp.Service.Abstract;
using DonationApp.Service.Dtos;
using DonationApp.WebApi.Attributes;
using DonationApp.WebApi.Controllers.Abstract;
using Microsoft.AspNetCore.Mvc;

namespace DonationApp.WebApi.Controllers;

[GoogleAuthorize]
[ApiController]
[Route("api/[controller]/[action]")]
public class DonationController : UserControllerBase
{
    private readonly IDonationService _donationService;

    public DonationController(IDonationService donationService)
    {
        _donationService = donationService;
    }

    [HttpGet]
    public async Task<IActionResult> GetByFilter([FromQuery]DonationFilter filter)
    {
        if (filter == null)
            throw new ArgumentNullException(nameof(filter));

        var entities = await _donationService.GetByFilter(filter).ConfigureAwait(false);;
        return Ok(entities);
    }

    [HttpGet]
    public async Task<IActionResult> GetByUserId()
    {
```

```
        var entities = await _donationService.GetByUserId(GetUserId()).ConfigureAwait(false);;

        return Ok(entities);
    }
}
```

Файл DonationGoalController.cs

```
using DonationApp.Service.Abstract;
using DonationApp.Service.Dtos;
using DonationApp.WebApi.Attributes;
using DonationApp.WebApi.Controllers.Abstract;
using Microsoft.AspNetCore.Mvc;

namespace DonationApp.WebApi.Controllers;


[ApiController]
[Route("api/[controller]/[action]")]
public class DonationGoalController : UserControllerBase
{
    private readonly IDonationGoalService _donationGoalService;

    public DonationGoalController(IDonationGoalService donationGoalService)
    {
        _donationGoalService = donationGoalService;
    }

    [GoogleAuthorize]
    [HttpPost]
    public async Task<IActionResult> Create(DonationGoalRequestDto donationGoalDto)
    {
        if (donationGoalDto == null)
            throw new ArgumentNullException(nameof(donationGoalDto));

        donationGoalDto.UserId = GetUserId();
        var entity = await _donationGoalService.Create(donationGoalDto).ConfigureAwait(false);
        return Ok(entity);
    }

    [GoogleAuthorize]
    [HttpPut]
```

```
    public async Task<IActionResult> Update(DonationGoalUpdateDto donationGoalDto)
    {
        if (donationGoalDto == null)
            throw new ArgumentNullException(nameof(donationGoalDto));

        donationGoalDto.UserId = GetUserId();
        var entity = await _donationGoalService.Update(donationGoalDto).ConfigureAwait(false);
        return Ok(entity);
    }

    [HttpGet]
    [Route("{userSlug}")]
    public async Task<IActionResult> GetByUser(string userSlug)
    {
        if (userSlug == null)
            throw new ArgumentNullException(nameof(userSlug));

        var entities = await _donationGoalService.GetByUser(userSlug).ConfigureAwait(false);

        return Ok(entities);
    }
}
```

Файл PaymentController.cs

```
using CloudIpspSDK.Models;
using DonationApp.Service.Abstract;
using DonationApp.Service.Dtos;
using Microsoft.AspNetCore.Mvc;

namespace DonationApp.WebApi.Controllers;

[ApiController]
[Route("api/[controller]/[action]")]
public class PaymentController : ControllerBase
{
    private readonly IPaymentService _paymentService;
    private readonly ILogger<PaymentController> _logger;

    public PaymentController(
        IPaymentService paymentService,
        ILogger<PaymentController> logger)
```

```
    {
        _paymentService = paymentService;
        _logger = logger;
    }

    [HttpPost]
    public async Task<IActionResult> PaymentCallback(ResponseModel response)
    {
        if (response == null)
            throw new ArgumentNullException(nameof(response));

        await _paymentService.ProcessPaymentCallback(response).ConfigureAwait(false);
        return Ok(response.response_status);
    }

    [HttpPost]
    public async Task<ActionResult> GenerateDonationLink([FromBody]DonationDto dto)
    {
        if (dto == null)
            throw new ArgumentNullException(nameof(dto));

        var result = await _paymentService.GenerateDonationLink(dto).ConfigureAwait(false);

        return Ok(result);
    }
}
```

Файл UserController.cs

```
using DonationApp.Service.Abstract;
using DonationApp.Service.Dtos;
using DonationApp.WebApi.Attributes;
using DonationApp.WebApi.Controllers.Abstract;
using Microsoft.AspNetCore.Mvc;

namespace DonationApp.WebApi.Controllers;

[GoogleAuthorize]
[ApiController]
[Route("api/[controller]/[action]")]
public class UserController : UserControllerBase
{
```

```
    private readonly IUserService _userService;

    public UserController(IUserService userService)
    {
        _userService = userService;
    }

    [HttpPost]
    public async Task<IActionResult> SetUserSlug(UserSlugDto userSlugDto)
    {
        if (userSlugDto == null)
            throw new ArgumentNullException(nameof(userSlugDto));

        userSlugDto.UserId = GetUserId();
        var result = await _userService.SetUserSlug(userSlugDto).ConfigureAwait(false);

        return Ok(result);
    }


    [HttpGet]
    public async Task<IActionResult> GetUserSlug()
    {
        var result = await _userService.GetUserSlug(GetUserId()).ConfigureAwait(false);

        return Ok(result);
    }
}
```

Файл WithdrawController.cs

```
using DonationApp.Service.Abstract;
using DonationApp.Service.Dtos;
using DonationApp.WebApi.Attributes;
using DonationApp.WebApi.Controllers.Abstract;
using Microsoft.AspNetCore.Mvc;

namespace DonationApp.WebApi.Controllers;

[GoogleAuthorize]
[ApiController]
[Route("api/[controller]/[action]")]
```

```
public class WithdrawController : UserControllerBase
{
    private readonly IWithdrawService _withdrawRequestService;

    public WithdrawController(IWithdrawService withdrawRequestService)
    {
        _withdrawRequestService = withdrawRequestService;
    }

    [HttpGet]
    public async Task<IActionResult> GetUserAvailableWithdrawAmount()
    {
        var result = await
_withdrawRequestService.GetUserAvailableWithdrawAmount(GetUserId()).ConfigureAwait(false);

        return Ok(result);
    }

    [HttpPost]
    public async Task<IActionResult> CreateWithdrawRequest([FromBody] WithdrawRequestDto
withdrawRequestDto)
    {
        try
        {
            if (withdrawRequestDto == null)
                throw new ArgumentNullException(nameof(withdrawRequestDto));

            withdrawRequestDto.UserId = GetUserId();
            var result = await
_withdrawRequestService.CreateWithdrawRequest(withdrawRequestDto).ConfigureAwait(false);

            return Ok(result);
        }
        catch (Exception ex)
        {
            return BadRequest(ex.Message);
        }
    }
}
```

Файл App.js

```
import React, {useEffect, useState} from "react"
import './App.css'
import DonateForm from "../donate-page"
import HomePage from "../home-page"
import NavigationBar from "../navigation-bar"
import LoginPage from "../login-page"
import {Switch, Route, useHistory} from "react-router-dom"
import WithdrawForm from "../withdraw-form"
import UserSettingsPage from "../user-settings-page"
import DonationGoal from "../donation-goal/donation-goal";
import Profile from "../profile/Profile";


function App() {
    const [loginData, setLoginData] = useState(getLoginData())
    const [showNav, setShowNav] = useState(true)
    let history = useHistory().location.pathname;
    useEffect(() => {
        if (history === "/donate")
            setShowNav(false)
    }, [history])

    function getLoginData(){
        if (sessionStorage.getItem("loginData"))
            return JSON.parse(sessionStorage.getItem("loginData"))
        return null
    }

    const logOut = () => {
        sessionStorage.removeItem("loginData")
        setLoginData(null)
    }

  return (
    <div className="App">
        {showNav &&
            <NavigationBar loginData={loginData} logout={logOut}/>
        }

        <Switch>
```

```
          <Route path="/home">
            <HomePage loginData={loginData} />
          </Route>
          <Route path="/withdraw">
            <WithdrawForm loginData={loginData} />
          </Route>
          <Route path="/user-settings">
            <UserSettingsPage loginData={loginData} />
          </Route>
          <Route path="/donate">
            <DonateForm />
          </Route>
          <Route path="/donation-goal">
            <DonationGoal loginData={loginData} />
          </Route>
          {!loginData &&   <Route path={["/login","/signup"]}>
                        <LoginPage setLoginData={setLoginData}/>
                    </Route>
          }
          <Route exact path="/profile/:id" render={(props) => <Profile loginData={loginData} userSlug =
{props.match.params.id}/>}/>
        </Switch>
      </div>
    );
}

export default App;
```

Файл DonationGoal.js

```
import React, {useEffect, useRef, useState} from 'react';
import {Progress} from 'react-sweet-progress';
import "react-sweet-progress/lib/style.css";
import "./donation-goal.css"
import DonationGoalsService from "../../services/DonationGoalsService";
import UserService from "../../services/UserService";
import styled from "styled-components";
import SaveIcon from "@mui/icons-material/Save";
import {Button} from "react-bootstrap";



const FormInput = styled("input")`
```

```
  background: linear-gradient(to bottom right, #EF4765, #FF9A5A);
  margin: 20px auto;
  min-height: 48px;
  border: 0;
  text-align: center;
  border-radius: 12px;
  color: #FFFFFF;
  cursor: pointer;
  display: inline-block;
  font-family: -apple-system, system-ui, "Segoe UI", Roboto, Helvetica, Arial, sans-serif;
  font-size: 24px;
  font-weight: 500;
  line-height: 2.5;
  outline: transparent;
  padding: 0 1rem;
  text-decoration: none;
  transition: box-shadow .2s ease-in-out;
  user-select: none;
  -webkit-user-select: none;
  touch-action: manipulation;
  white-space: nowrap;
  :hover{
    cursor: pointer;
  }`

const DonationGoal = ({loginData}) => {

  const [name, setName] = useState('')
  const [description, setDescription] = useState('')
  const [cardNumber, setCardNumber] = useState('')
  const [amount, setAmount] = useState(null)
  const [userSlug, setSlug] = useState("")


  const donationGoalService = new DonationGoalsService(loginData)
  const userService = new UserService(loginData)


  const [goalsData, setGoalsData] = useState({
    name: 'Назва цілі',
    description: 'Опис цілі',
    id: 'Id',
```

```
  amount: 'Загальна сума збору',
  currentAmount: 'Уже зібрано',
  cardNumber: 'cardNumber'
})

let initialUserSlug = ''

useEffect(async () => {
  await userService.getSlug()
    .then(res => {
      setSlug(res)
      initialUserSlug = res
    })
  await handleGetGoalsData()
}, [])


const createDonationGoal = async (data) => {
  await donationGoalService.createDonationGoal(data);
}

function handleGetGoalsData() {
  donationGoalService.getDonationGoals(initialUserSlug || userSlug)
    .then(res => {
      const result = [goalsData, ...res]
      setGoalsData(result)
    })
}

function handleSetName(event) {
  setName(event.target.value)
}

function handleSetDescription(event) {
  setDescription(event.target.value)
}

function handleSetCardNumber(event) {
  setCardNumber(event.target.value)
}

function handleSetAmount(event) {
```

```
  setAmount(event.target.value)
}

function countPercent(currentAmount, amount) {
  const percent = currentAmount / amount
  return (percent * 100).toFixed(2)
}

async function handleSubmit() {
  const data = {name, description, cardNumber, amount}
  await createDonationGoal(data)
  await handleGetGoalsData()
}

return (
  <div>
    <div className={"goalList"}>
      {goalsData.length > 0 ? goalsData.map(
        (el, index) =>
          <div>
            {index > 0 &&
              <div className={"goalListElement"} key={el.id}>
                <div className={"goalListElementInfo"}>
                  <div className={"goalListElementName"}>
                    {el.name}
                  </div>
                  <div className={"goalListElementDescription"}>
                    <input className={"formInputElement"} value={el.description}/>
                  </div>
                  <div className={"goalListElementAmount"}>
                    Зібрано {el.currentAmount} з <input className={"formInputElement"} value={el.amount}/>
                  </div>
                </div>
                <Progress
                  className={"goalListElementBar"}
                  width="100px"
                  type='circle'
                  strokeWidth="4"
                  theme={{
                    success: {
                      symbol: '',
                      color: 'rgb(223, 105, 180)'
```

```
                }
                }}
                percent={countPercent(el.currentAmount, el.amount)}/>
              <Button className={"saveIcon"}><SaveIcon/></Button>
            </div>
          }

        </div>
      )
      : <div> No data </div>}
    </div>
    <form >
      <div className={"goalListElementOleh"}>
        <h1>Форма для створення нової цілі для донатів</h1>
        <div className={"formTableElement"}>
          <label className={"formLabelElement"}>
            <span>Назва цілі:</span>
            <input className={"formInputElement"} type="text" name="name" onChange={handleSetName}/>
          </label>
        </div>
        <div className={"formTableElement"}>
          <label className={"formLabelElement"}>
            <span>Опис цілі:</span>
            <input className={"formInputElement"} type="text" name="description"
onChange={handleSetDescription}/>
          </label>
        </div>
        <div className={"formTableElement"}>
          <label className={"formLabelElement"}>
            <span>Номер карти:</span>
            <input className={"formInputElement"} type="text" name="cardNumber"
onChange={handleSetCardNumber}/>
          </label>
        </div>
        <div className={"formTableElement"}>
          <label className={"formLabelElement"}>
            <span>Загальна сума збору:</span>
            <input className={"formInputElement"} type="number" name="Amount"
onChange={handleSetAmount}/>
          </label>
        </div>
        <FormInput onClick={handleSubmit} defaultValue="Створити ціль"/>
```

```
        </div>
      </form>
    </div>
  );
};

export default DonationGoal;
```

Файл DonationsTable.js

```
import React, {useEffect, useState} from "react"
import {Table} from "react-bootstrap"
import TableItem from "./TableItem"
import DonationService from "../../services/DonationService"

function DonationsTable({loginData}) {
    const [donationsInfo, setDonationsInfo] = useState([
        {id: 1, donatorName: "Viktor", amount: 200, currency: "UAH"},
        {id:2, name: "Viktor", amount: 200, currency: "UAH"}])


    useEffect(() => {
        const donationService = new DonationService(loginData)
        async function fetchApi() {
            const data = await donationService.getDonationsOfUser()
                .catch(err => alert(err));
            console.log(data)
            setDonationsInfo(data)
        }
        fetchApi()
    }, [])
    const tableItems = donationsInfo.map(d => {return <TableItem key={d.id}{...d}/>})

    return(
        <Table striped>
            <thead>
            <tr>
                <th>Дата</th>
                <th>Ім'я</th>
                <th>Повідомлення</th>
                <th>Сума</th>
            </tr>
```

```
            </thead>
            <tbody>
            {tableItems}
            </tbody>
        </Table>
    )
}
export default DonationsTable
```

Файл TableItem.js

```
import React from "react"
function TableItem({id, donatorName, amount, createdOn, message}) {
    let date = new Date(createdOn)
    return(
        <tr>
            <td>{`${date.toLocaleDateString()} ${date.getHours()}:${date.getMinutes()}`}</td>
            <td>{donatorName}</td>
            <td>{message}</td>
            <td>{amount} UAH</td>
        </tr>
    )
}
export default TableItem
```

Файл HomePage.js

```
import React from "react"
import DonationsTable from "../donations-table"
import "./HomePage.css"

function HomePage({loginData}) {
    return(
        <>
            <div className="table-background">
                <DonationsTable loginData={loginData}/>
            </div>
        </>
    )
}
export default HomePage
```

Файл LoginPage.js

```
import React from "react"
import SignupForm from "./SignupForm"
import SigninForm from "./SigninForm"
import {Route, Switch} from "react-router-dom"

function LoginPage({setLoginData}) {
  return(
    <>
      <Switch>
        <Route path="/login">
          <SigninForm setLoginData={setLoginData}/>
        </Route>
        <Route path="/signup">
          <SignupForm />
        </Route>
      </Switch>
    </>
  )
}
export default LoginPage
```

Файл SigninForm.js

```
import React from "react"
import {Col, Form, Row} from "react-bootstrap"
import GoogleLogin from "react-google-login"
import AuthorizeService from "../../services/AuthorizeService"

function SigninForm({setLoginData}) {
  const handleFailure = (result) => {
    alert(result)
  }
  const handleLogin = async (googleData) => {
    const userService = new AuthorizeService(googleData)
    const res = await userService.loginUser()
    const constructed = {
      tokenId: googleData.tokenId,
      userName: res.name,
      userEmail: res.email
    }
```

```
        setLoginData(constructed)
        sessionStorage.setItem("loginData", JSON.stringify(constructed))
        console.log(constructed)
    }

    return(
        <div className="wrapper-card">
            <Form className="container-card" >
                <div className="content">
                    <h1 className="title">Увійти</h1>
                    <Row className="justify-content-sm-center">
                        <Col xs={12}>
                            <GoogleLogin
                                clientId={"108433003367-gtacrot2h32ch66dqjpvrlqnfoi1q4mq.apps.googleusercontent.com"}
                                buttonText={"Log in with Google"}
                                onSuccess={handleLogin}
                                onFailure={handleFailure}
                                cookiePolicy={"single_host_origin"} />
                        </Col>
                    </Row>
                </div>
            </Form>
        </div>
    )
}
export default SigninForm
```

Файл NavigationBar.js

```
import React from "react"
import {Container, Nav, Navbar} from "react-bootstrap"
import {LinkContainer} from "react-router-bootstrap"

function NavigationBar({loginData, logout}) {
    return (
        <Navbar expand={"lg"}>
            <Container>
                <Navbar.Brand className={"logo"}>donate ukr</Navbar.Brand>
                <Navbar.Toggle aria-controls="basic-navbar-nav"/>
                <Navbar.Collapse>
                    {loginData && <Nav>
                        <LinkContainer to="/donation-goal">
```

```
                    <Nav.Link>
                        Цілі для донатів
                    </Nav.Link>
                </LinkContainer>
                <LinkContainer to="/home">
                    <Nav.Link>
                        Список донатів
                    </Nav.Link>
                </LinkContainer>
                <LinkContainer to="/withdraw">
                    <Nav.Link>
                        Вивести кошти
                    </Nav.Link>
                </LinkContainer>
                <LinkContainer to="/user-settings">
                    <Nav.Link>
                        Налаштування користувача
                    </Nav.Link>
                </LinkContainer>
                <LinkContainer to="/">
                    <Nav.Link className="logout-nav-link" onClick={logout}>
                        Вийти
                    </Nav.Link>
                </LinkContainer>
            </Nav>
            }

            {!loginData && <Nav>
                <LinkContainer to="/login">
                    <Nav.Link className="login-nav-link">
                        Зареєструватися/Увійти
                    </Nav.Link>
                </LinkContainer>
            </Nav>
            }
        </Navbar.Collapse>
    </Container>
</Navbar>
)
}

export default NavigationBar
```

Файл Profile.js

```
import React, {useEffect, useRef, useState} from 'react';
import DonationGoalsService from "../../services/DonationGoalsService";
import "./profile.css"


import styled from "styled-components";
import PaymentService from "../../services/PaymentService";
import {Progress} from "react-sweet-progress";


const DropDownContainer = styled("div")`
  width: 15em;
  margin: 0 auto;
   & > div > ul{
     padding-left: 0;
   }
`;

const DropDownHeader = styled("div")`
  margin-bottom: 0.8em;
  padding: 0.4em 2em 0.4em 1em;
  box-shadow: 0 2px 3px rgba(0, 0, 0, 0.15);
  font-weight: 500;
  font-size: 1.3rem;
  text-align: center;
  color: black;
`;

const DropDownListContainer = styled("div")``;

const DropDownList = styled("ul")`
  margin: 0;
  box-sizing: border-box;
  color: #3faffa;
  font-size: 1.3rem;
  font-weight: 500;
`;
const ListItem = styled("li")`
    list-style: none;
```

```
    width: 100%;
    color: black;
    text-align: center;
    margin-bottom: 0.8em;
    background: white;

`;

const FormInput = styled("input")`
  background: linear-gradient(to bottom right, #EF4765, #FF9A5A);
  margin: 20px auto;
  min-height: 48px;
  border: 0;
  text-align: center;
  border-radius: 12px;
  color: #FFFFFF;
  cursor: pointer;
  display: inline-block;
  font-family: -apple-system, system-ui, "Segoe UI", Roboto, Helvetica, Arial, sans-serif;
  font-size: 24px;
  font-weight: 500;
  line-height: 2.5;
  outline: transparent;
  padding: 0 1rem;
  text-decoration: none;
  transition: box-shadow .2s ease-in-out;
  user-select: none;
  -webkit-user-select: none;
  touch-action: manipulation;
  white-space: nowrap;
  :hover{
    cursor: pointer;
  }`

const Profile = ({loginData, userSlug}) => {

    const donationGoalService = new DonationGoalsService(loginData)
    const paymentService = new PaymentService()

    const [goalsData, setGoalsData] = useState('')
    const [selectedDonation, setSelectedDonation] = useState('')
    const [isOpen, setIsOpen] = useState(false);
```

```
const amountRef = useRef(null)
const donatorNameRef = useRef('')
const messageRef = useRef('')

const toggling = () => setIsOpen(!isOpen);

const onOptionClicked = value => () => {
    setSelectedDonation(value);
    setIsOpen(false);
};

useEffect(()=>{
    handleGetGoalsData()
}, [])
function handleGetGoalsData(){
    donationGoalService.getDonationGoals(userSlug)
        .then(res => {
            setGoalsData(res)
            setSelectedDonation(res[0])
        })
}

function countPercent(currentAmount, amount) {
    const percent = currentAmount / amount
    return (percent * 100).toFixed(2)
}
const createDonate = async() => {
    const donate = {
        "amount": amountRef.current.value,
        "donatorName": donatorNameRef.current.value,
        "message": messageRef.current.value,
        "donationGoalId": selectedDonation.id
    }
    await (await paymentService.donate(donate)).text()
        .then(res => window.open(res, "_blank", "noreferrer"));
}

return (
    <div>
        <div>
            {goalsData.length > 0 ? goalsData.map(
```

```
        (el) =>
          <div className={"goalList"}>
          <div className={"goalListElement"} key={el.id}>
            <div className={"goalListElementInfo"}>
              <div className={"goalListElementName"}>
                {el.name}
              </div>
              <div className={"goalListElementDescription"}>
                {el.description}
              </div>
              <div className={"goalListElementAmount"}>
                Зібрано {el.currentAmount} з {el.amount}
              </div>
            </div>
            <Progress
              className={"goalListElementBar"}
              width="100px"
              type='circle'
              strokeWidth="4"
              theme={{
                success: {
                  symbol: '🤣',
                  color: 'rgb(223, 105, 180)'
                }
              }}
              percent={countPercent(el.currentAmount, el.amount)}/>
          </div>
          </div>
      )
      : <div> No data </div>}
    </div>
    <form >
      <div className={"goalListElementOleh"}>
        <h1>Форма для здійснення донату</h1>
        <div className={"formTableElement"}>
          <label className={"formLabelElement"}>
            <span>Ім'я:</span>
            <input className={"formInputElement"} type="text" name="name" ref={donatorNameRef}/>
          </label>
        </div>
        <div className={"formTableElement"}>
          <label className={"formLabelElement"}>
```

```
                    <span>Повідомлення:</span>
                    <input className={"formInputElement"} type="text" name="name" ref={messageRef}/>
                  </label>
                </div>
                <div className={"formTableElement"}>
                  <label className={"formLabelElement"}>
                    <span>Сума донату:</span>
                    <input className={"formInputElement"} type="text" name="name" ref={amountRef}/>
                  </label>
                </div>
                <DropDownContainer>
                  <DropDownHeader onClick={toggling}>{selectedDonation.name || 'Name'}</DropDownHeader>
                  {isOpen && (
                    <DropDownListContainer>
                      <DropDownList>
                        {
                          goalsData.length > 0 ? goalsData.map((el) =>
                            <ListItem onClick={onOptionClicked(el)} key={el.id}> {el.name} </ListItem>
                          ) : <div>No Data</div>
                        }
                      </DropDownList>
                    </DropDownListContainer>
                  )}
                </DropDownContainer>
                <FormInput onClick={createDonate} defaultValue="Зробити донат"/>
              </div>
            </form>
          </div>
    );
};

export default Profile;
```

Файл UserSettingsPage.js

```
import React, {useEffect, useState} from "react"
import "./UserSettingsPage.css"
import {Button, Container, FormControl, InputGroup, Row} from "react-bootstrap"
import ContentCopyIcon from '@mui/icons-material/ContentCopy';
import SaveIcon from '@mui/icons-material/Save';
import UserService from "../../services/UserService"
```

```
function UserSettingsPage({loginData}) {
    const [slug, setSlug] = useState("")



    useEffect(()=>{
        const userService = new UserService(loginData)
        userService.getSlug()
            .then(res => {
                setSlug(res)
            })
    },[])

    const saveSlug = async () => {
        const userService = new UserService(loginData)
        const res = await userService.setSlug(slug);
        console.log(res);
    }
    return(
        <>
            <div className="widget-page-background">
                <h1 className="title">Налаштування сторінки користувача</h1>
                <br/>
                <h2 className="widget-page-subtitle">На цій сторінці ви можете змінити унікальний ідентифікатор
користувача та скопіювати посилання на сторінку донацій користувача</h2>
                <Container>
                    <Row className="widget-setting">
                        <h2 className="setting-name">Посилання на сторінку донацій</h2>
                            <InputGroup>
                                <FormControl type="text" value="localhost:3000/profile/" readOnly/>
                                <FormControl type="text" onChange={(e) => setSlug(e.target.value)} value={slug}/>
                                <Button><SaveIcon onClick={() => saveSlug()}/></Button>
                                <Button variant="outline-secondary"><ContentCopyIcon onClick={() =>
{navigator.clipboard.writeText("localhost:3000/profile/" + slug)}}/></Button>
                            </InputGroup>
                    </Row>
                </Container>
            </div>
        </>
    )
}
export default UserSettingsPage
```

Файл WithdrawForm.js

```
import React, {useEffect, useRef, useState} from "react"
import {Col, Form, Row} from "react-bootstrap"
import "./WithdrawForm.css"
import WithdrawService from "../../services/WithdrawService"
import styled from "styled-components";

const DropDownContainer = styled("div")`
  width: 15em;
  margin: 20px auto 20px;
  & > div > ul{
    padding-left: 0;
  }
`;

const DropDownHeader = styled("div")`
  margin-bottom: 0.8em;
  padding: 0.4em 2em 0.4em 1em;
  box-shadow: 0 2px 3px rgba(0, 0, 0, 0.15);
  font-weight: 500;
  font-size: 1.3rem;
  text-align: center;
  color: black;
`;

const DropDownListContainer = styled("div")``;

const DropDownList = styled("ul")`
  margin: 0;
  box-sizing: border-box;
  color: #3faffa;
  font-size: 1.3rem;
  font-weight: 500;
`;
const ListItem = styled("li")`
  list-style: none;
  width: 100%;
  color: black;
  text-align: center;
  margin-bottom: 0.8em;
  background: white;
```

```
`;

function WithdrawForm({loginData}) {
    const [userAvailableWithdrawAmount, setUserAvailableWithdrawAmount] = useState({
cardNumber:'cardNumber', amount:'Amount'})
    const [selectedCardNumber, setSelectedCardNumber] = useState('')
    const [isOpen, setIsOpen] = useState(false);
    const toggling = () => setIsOpen(!isOpen);
    const withdrawService = new WithdrawService(loginData)
    const amountRef = useRef(null)

    useEffect(async () => {
        await handleGetGoalsData()

    },[])

    async function handleGetGoalsData(){
        const data = await withdrawService.getUserAvailableWithdrawAmount()
            .then(res => {
                setUserAvailableWithdrawAmount(res)
                setSelectedCardNumber(res[0])
            }).catch(err => alert(err));
        console.log(data)
    }

    const onOptionClicked = value => () => {
        setSelectedCardNumber(value);
        setIsOpen(false);
    };

    const createWithdrawRequest = async() => {
        const withdrawRequestData = {
            "cardNumber": selectedCardNumber.cardNumber,
            "amount": amountRef.current.value,
        }
        await withdrawService.createWithdrawRequest(withdrawRequestData);
        await handleGetGoalsData()
    }

    return(<>
            <div className="wrapper-balance">
```

```
            <div className="container-card container-balance" >
              <div className="content">
                <h1 className="title">Ваш баланс</h1>
                <h2 className="subtitle">Прямо зараз на вашому балансі доступно для переводу<br/></h2>
              </div>
            </div>
          </div>
          <div className={"withdrawList"}>
            {userAvailableWithdrawAmount.length > 0 ? userAvailableWithdrawAmount.map(
                (el) =>
                  <div className={"goalList"}>
                    <div className={"goalListElementWrapper"} key={el.id}>
                      <div className="card">
                        <div className="visa_info">
                          <img src="https://raw.githubusercontent.com/muhammederdem/credit-card-
form/master/src/assets/images/chip.png" alt=""/>
                          <p>{el.cardNumber}</p>
                        </div>
                      </div>
                      <div className="cardAmount">
                        Доступно: {el.amount}
                      </div>
                    </div>
                  </div>
              )
              : <div> No data </div>}
          </div>
            <div className="wrapper-balance">
              <Form className="container-card" >
                <div className="content">
                  <h1 className="title">Запит на виведення коштів</h1>
                  <h2 className="subtitle">Оберіть номер карти та введіть суму для виводу <br/> Зазвичай
цей процес займає до 3-х днів</h2>
                  <Row className="justify-content-sm-center">
                    <Col xs={12}>
                      <DropDownContainer>
                        <DropDownHeader onClick={toggling}>{selectedCardNumber.cardNumber ||
'CardNumber'}</DropDownHeader>
                          {isOpen && (
                            <DropDownListContainer>
                              <DropDownList>
                                {
```

```
                                    userAvailableWithdrawAmount.length > 0 ?
userAvailableWithdrawAmount.map((el) =>
                                        <ListItem onClick={onOptionClicked(el)} key={el.cardNumber}>
{el.cardNumber} </ListItem>
                                    ) : <div>No data</div>
                                }
                            </DropDownList>
                        </DropDownListContainer>
                    )}
                </DropDownContainer>
                <input type="number" className="donate-input" placeholder="0" name="amount" required
ref={amountRef}/>
                <br/>
                <input className={"formButton"} onClick={createWithdrawRequest}
defaultValue="Створити запит"/>
                </Col>
            </Row>
        </div>
    </Form>
</div>
</>
)
}
export default WithdrawForm
```

Файл AuthorizeService.js

```
export default class AuthorizeService {
    constructor(googleData) {
        this.api = process.env["REACT_APP_API"]+"/api/authorize"
        this.tokenId = googleData.tokenId
    }

    async loginUser() {
        const body = JSON.stringify({
            tokenId: this.tokenId,
            loginType: 0
        })
        const req = await fetch(this.api + "/createIfNotExist", {
            method: "POST",
            headers: {
                "Content-type": "application/json",
```

```
            },
            body: body,
            cache: "default",
            cors: "cors"
        }).then(res => res.json(), reject => alert("Будь ласка, спробуйте авторизуватися заново")).catch(err =>
console.log(err))
        return req
    }
}
```

Файл DonationGoalsService.js

```
export default class DonationGoalsService {
    constructor(userData) {
        this.api = process.env["REACT_APP_API"]+"/api/DonationGoal"
        this.tokenId = userData.tokenId
        this.name = userData.name
    }

    async createDonationGoal(donationGoalData) {
        const body = JSON.stringify({
            name: donationGoalData.name,
            description: donationGoalData.description,
            cardNumber: donationGoalData.cardNumber,
            amount: donationGoalData.amount
        })
        return await fetch(this.api + "/Create", {
            method: "POST",
            headers: {
                "Content-type": "application/json",
                "Authorization": `Bearer ${this.tokenId}`
            },
            body: body,
            cache: "default",
            cors: "cors"
        }).then(res => res.json(), () => alert("Будь ласка, спробуйте ще раз")).catch(err => console.log(err))
    }

    async getDonationGoals(userSlug){
        return fetch(this.api + "/GetByUser/"+ userSlug)
            .then(res => res.json())
            .then(res => res)
```

```
    }
}
```

Файл DonationService.js

```
export default class DonationService {
    constructor(userData) {
        this.api = process.env["REACT_APP_API"]+"/api/donation"
        this.tokenId = userData.tokenId
    }

    async getDonationsOfUser() {
        const res = await fetch(this.api + "/getByUserId", {
            method: "GET",
            headers: {
                "Content-type": "application/json",
                "Authorization": `Bearer ${this.tokenId}`
            },
            cache: "default",
            cors: "cors"
        })
        const data = await res.json()
        return data
    }
}
```

Файл PaymentService.js

```
export default class PaymentService {
    constructor() {
        this.api = process.env["REACT_APP_API"]+"/api/Payment"
    }

    async donate(donate){
        const body = JSON.stringify({
            "amount":donate.amount,
            "donatorName": donate.donatorName,
            "message": donate.message,
            "donationGoalId": donate.donationGoalId
        })
        return await fetch(this.api + "/GenerateDonationLink", {
```

```
            method: "POST",
            headers: {
                "Content-type": "application/json",
            },
            body: body,
            cache: "default",
            cors: "cors"
        })
    }
}
```

Файл UserService.js

```
export default class UserService {
    constructor(userData) {
        this.api = process.env["REACT_APP_API"]+"/api/user"
        this.tokenId = userData.tokenId
    }

    async setSlug(userSlug) {
        const body = JSON.stringify({
            slug: userSlug,
            userId: "87785743-D4C2-442D-B12D-D8CDA9EC74C4"
        })
        return await fetch(this.api + "/setUserSlug", {
            method: "POST",
            headers: {
                "Content-type": "application/json",
                "Authorization": `Bearer ${this.tokenId}`
            },
            body: body,
            cache: "default",
            cors: "cors"
        }).then(res => res.json(), () => alert("Будь ласка, спробуйте ще раз")).catch(err => console.log(err))
    }
    async getSlug(){
        return fetch(this.api + "/getUserSlug",{
            headers: {
                "Content-type": "application/json",
                "Authorization": `Bearer ${this.tokenId}`
            },
        })
```

```
            .then(res => res.text())
            .then(res => res)
    }
}
```

Файл WithdrawService.js

```
export default class WithdrawService {
    constructor(userData) {
        this.api = process.env["REACT_APP_API"]+"/api/withdraw"
        this.tokenId = userData.tokenId
    }

    async getUserAvailableWithdrawAmount() {
        const res = await fetch(this.api + "/getUserAvailableWithdrawAmount", {
            method: "GET",
            headers: {
                "Content-type": "application/json",
                "Authorization": `Bearer ${this.tokenId}`
            },
            cache: "default",
            cors: "cors"
        })

        return await res.json()
    }

    async createWithdrawRequest(withdrawRequestData) {
        const body = JSON.stringify({
            cardNumber: withdrawRequestData.cardNumber,
            amount: withdrawRequestData.amount
        })
        return await fetch(this.api + "/CreateWithdrawRequest", {
            method: "POST",
            headers: {
                "Content-type": "application/json",
                "Authorization": `Bearer ${this.tokenId}`
            },
            body: body,
            cache: "default",
            cors: "cors"
        }).then(res => res.json(), () => alert("Будь ласка, спробуйте ще раз")).catch(err => console.log(err))
```

```
    }
}
```

Файл Index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './components/app/App';
import './index.css';
import {BrowserRouter} from "react-router-dom"
ReactDOM.render(
    <React.StrictMode>
      <BrowserRouter>
         <App />
      </BrowserRouter>
    </React.StrictMode>
 ,
  document.getElementById('root')
);
```

Файл App.css

```
.App {
  text-align: center;
}
```

Файл donation-goal.css

```
.goalList{
    margin-top: 50px;
    display: flex;
    flex-direction: column;
    gap: 20px;
}

.goalListElementOleh{
    background-color: #fff;
    box-shadow: 0 17px 34px -20px #f55951;
    margin: 40px auto;
    border-radius: 1em;
    padding: 1em;
    width: 50%;
```

```
}

.goalListElement{
    width: 50%;
    margin: 0 auto;
    background-color: #fff;
    padding: 1em;
    border-radius: 1em;
    max-width: 768px;
    min-width: 320px;
    box-shadow: 0 17px 34px -20px #f55951;
    display: flex;
    flex-direction: row;
    justify-content: space-between;
}

.goalListElementInfo{
    width: 80%;
    display: flex;
    flex-direction: column;
    justify-content: start;
    gap: 20px;
    text-align: left;
}

.goalListElementName{
    font-size: 24px;
    font-weight: 500;
}

.goalListElementDescription{
    font-size: 18px;
}

.goalListElementAmount {
    font-size: 18px;
}

.goalListElementBar{
    margin-top: 20px;
}
```

```
.formTable {
    margin: 40px auto;
    display: flex;
    flex-direction: column;
    width:40%;
}

.formTableElement {
    width: 40%;
    margin: 50px auto;
}

.formInputElement{
    border: indianred solid 2px;
    border-radius: 12px;
    min-height: 30px;

}

.formLabelElement {
    display: flex;
    flex-direction: row;
    justify-content: space-between;
    font-weight: 500;
}


.formButton {
    background: linear-gradient(to bottom right, #EF4765, #FF9A5A);
    margin: 20px auto;
    border: 0;
    text-align: center;
    border-radius: 12px;
    color: #FFFFFF;
    cursor: pointer;
    display: inline-block;
    font-family: -apple-system, system-ui, "Segoe UI", Roboto, Helvetica, Arial, sans-serif;
    font-size: 24px;
    font-weight: 500;
    line-height: 2.5;
    outline: transparent;
    padding: 0 1rem;
```

```
    text-decoration: none;
    transition: box-shadow .2s ease-in-out;
    user-select: none;
    -webkit-user-select: none;
    touch-action: manipulation;
    white-space: nowrap;
}

.formButton:not([disabled]):focus {
    box-shadow: 0 0 .25rem rgba(0, 0, 0, 0.5), -.125rem -.125rem 1rem rgba(239, 71, 101, 0.5), .125rem .125rem
1rem rgba(255, 154, 90, 0.5);
}

.formButton:not([disabled]):hover {
    box-shadow: 0 0 .25rem rgba(0, 0, 0, 0.5), -.125rem -.125rem 1rem rgba(239, 71, 101, 0.5), .125rem .125rem
1rem rgba(255, 154, 90, 0.5);
}

.saveIcon {
    height: 50%;
    padding: 10px;
    margin: 15px;
}
```

Файл HomePage.css

```
.table-background {
    background-color: #fff;
    padding: 1em;
    border-radius: 1em;
    max-width: 90vw;
    min-width: 320px;
    -webkit-box-shadow: 0px 17px 34px -20px #f55951;
    box-shadow: 0px 17px 34px -20px #f55951;
    margin-top: 2em;
    margin-left: auto;
    margin-right: auto;
}
```

Файл profile.css

```
$purple: #663554;
```

```
$lightpurple: #b38570;
$gold: #ffd58c;
$highlight: #6d5e46;
$time: 3.4;


body {
    font-family: 'Verdana', sans-serif;
    padding: 0;
    margin: 0;
    background: #ff7b4a;
}

.credit-card {
    position: absolute;
    width: 100%;
    height: 100%;
}
```

Файл UserSettingPage.css

```
.widget-page-background {
    background-color: #fff;
    padding: 1em;
    border-radius: 1em;
    max-width: 90vw;
    min-width: 320px;
    -webkit-box-shadow: 0 17px 34px -20px #f55951;
    box-shadow: 0 17px 34px -20px #f55951;
    margin-top: 2em;
    margin-left: auto;
    margin-right: auto;
    color: #000;
    font-family: 'Muli', sans-serif;
    font-size: 1rem;
}

.widget-page-subtitle {
    text-transform: none;
    font-size: 0.9rem;
}
```

```
.widget-setting {
    margin-top: 5em;
}

.slug-editor {
    max-width: 30%;
    min-width: 120px;
}
```

Файл WihdrawForm.css

```
@import url("https://fonts.googleapis.com/css2?family=Rubik:wght@300;400;500;600;700&display=swap");

.wrapper-balance {
    display: grid;
    place-items: center;
    margin-top: 40px;
    min-width: 320px;
}

.goalListElementWrapper{
    width: 50%;
    margin: 0 auto;
    background-color: #fff;
    padding: 1em;
    border-radius: 1em;
    max-width: 768px;
    min-width: 320px;
    box-shadow: 0 17px 34px -20px #f55951;
    display: block
}

.card-wrapper {
    padding: 0 2em;
    margin: auto 0;
}


.container-balance {
    max-width: 576px;
    min-width: 320px;
    min-height: 200px;
```

```
}

.wrapper-card {
    color: #000;
    font-family: 'Muli', sans-serif;
    font-size: 1rem;
    display: -ms-grid;
    display: grid;
    min-width: 320px;
    place-items: center;
}

.cardAmount{
    font-size: 22px;
}

.card {
    margin: 10px auto 30px;
    width: 380px;
    border: 2px solid rgba(255, 255, 255, 0.2);
    background-image: linear-gradient(
            109.6deg,
            rgba(62, 161, 219, 1) 11.2%,
            rgba(93, 52, 236, 1) 100.2%
    );
    border-radius: 10px;
    z-index: 1;
    overflow: hidden;
    backdrop-filter: blur(5px);
    -webkit-backdrop-filter: blur(5px);
    position: relative;
}


.visa_info {
    padding: 15px;
}
.visa_info img {
    width: 60px;
    height: 45px;
}
```

```
.visa_info p {
    font-size: 22px;
    letter-spacing: 2px;
    color: #ffffff;
    padding-top: 50px;
    padding-bottom: 10px;
    margin-bottom: 0;
}

.visa_crinfo {
    display: flex;
    justify-content: space-between;
    color: #ffffff;
}
```

Файл Index.css

```
* {
  margin: 0;
  box-sizing: border-box;
}

body {
  margin: 0;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  background-color: #F1E8E6;
}

.logout-nav-link {
  color: #5F4BB6!important;
  border: 1px solid #5F4BB6;
  background-color: transparent;
  border-radius: 0.5em;
}

.login-nav-link {
  background-color: #5F4BB6;
  color: #fff!important;
  border-radius: 0.5em;
}
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

Веб-сервіс для донатів

Програма та методика тестування

КПІ.ІТ-9126. 045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Катерина ЛІЩУК

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Олег ТОНКОШКУР

Київ – 2023

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є програмне забезпечення веб-сервісу для донатів. Серверна частино програмного забезпечення написана на мові програмування – C# за допомогою ASP.NET Web API, клієнтська частина за допомогою JavaScript з використанням бібліотеки React.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка відповідності програмного забезпечення вимогам технічного завдання;
- перевірка збереження даних;
- знаходження проблем, помилок і недоліків з метою їх усунення;
- перевірка зручності графічного інтерфейсу;
- працездатність компонентів сторінок веб-додатку.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- функціональне тестування;
- системне тестування;
- мануальне тестування.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з використанням наскрізного тестування та за допомогою Postman, з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності користування.

Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

- динамічне тестування на відповідність функціональним вимогам;
- тестування інтерфейсу користувача;
- тестування зручності використання.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

Веб-сервіс для донатів

Керівництво користувача

КПІ.ІТ-9126.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Катерина ЛІЩУК

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Олег ТОНКОШКУР

Київ – 2023

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

«Donation App Service» - це система для збору пожертвувань та керування цим процесом. Додаток створений з метою допомоги благодійним організаціям, фондам та іншим благодійним проектам у зборі коштів для своїх благодійних цілей. Весь вихідний код системи заходиться в двох репозиторіях. В одному репозиторії знаходиться код серверного застосунку, а в іншому клієнтського. Система складається з клієнтського та серверного застосунків.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

– наявність браузеру;

– наявність доступу до Інтернету;

– для завантаження всіх модулів системи на пк повинно бути не менше 100 МБ вільної пам’яті.

2.2 Завантаження застосунку

Для повноцінного налаштування системи в локальному середовищі, спочатку необхідно налаштувати всі зовнішні залежності системи та інструменти розробки. На локальний комп’ютер необхідно завантажити MS SQL Server, .NET SDK, Node.js, Rider та WebStorm та також необхідно завантажити код з обох репозиторіїв.

Наступним кроком необхідно створити базу даних використовуючи MS SQL. Рядок підключення до бази даних необхідно вказати у appsettings.json серверного застосунку.

Для налаштування клієнтської частини необхідно відкрити клієнтський застосунок у WebStorm, та написати в командному рядку дві команди: npm install та npm start.

2.3 Перевірка коректної роботи

Після запуску команди npm start користувач перенаправляється на відкриту вкладку у веб-браузері на якій повинна відображатися стартова сторінка.

3 ВИКОНАННЯ ПРОГРАМИ

Взаємодія з системою починається з того, що користувач потрапляє на сторінку входу, де він може натиснути кнопку зареєструватися/увійти.

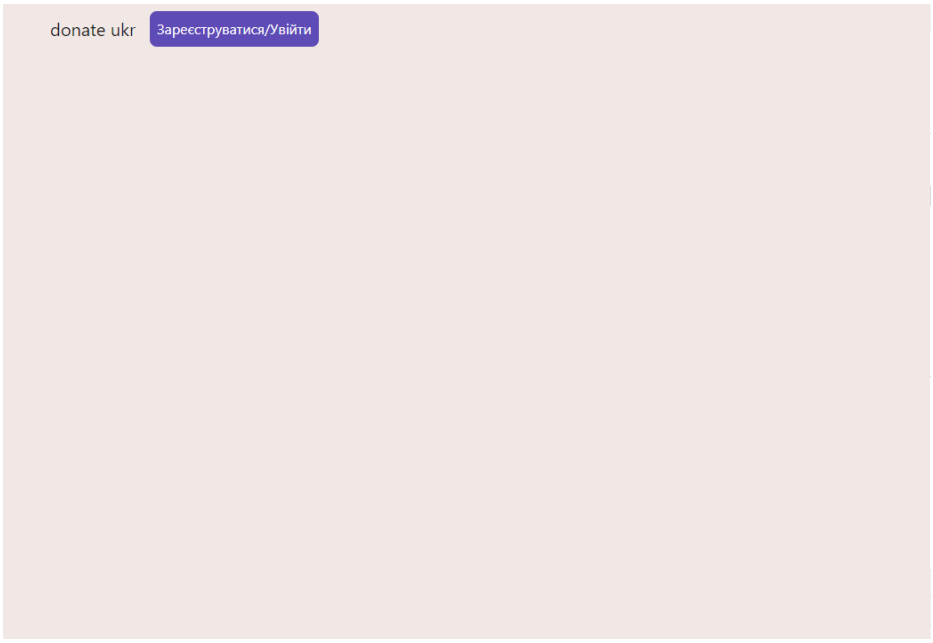


Рисунок 3.1 – Стартова сторінка

Після натискання кнопки зареєструватися/увійти, з’являється форма з кнопкою Log in with Google.

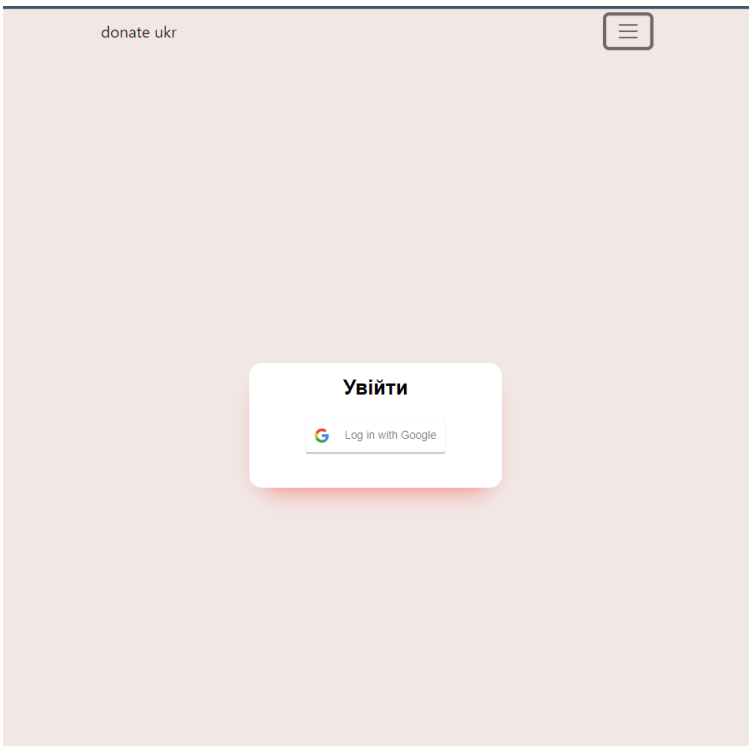


Рисунок 3.2 – Стартова сторінка з кнопкою для входу

Після натискання кнопки Log in with Google з’являється спливаюче вікно з формою для входу в Google аккаунт.

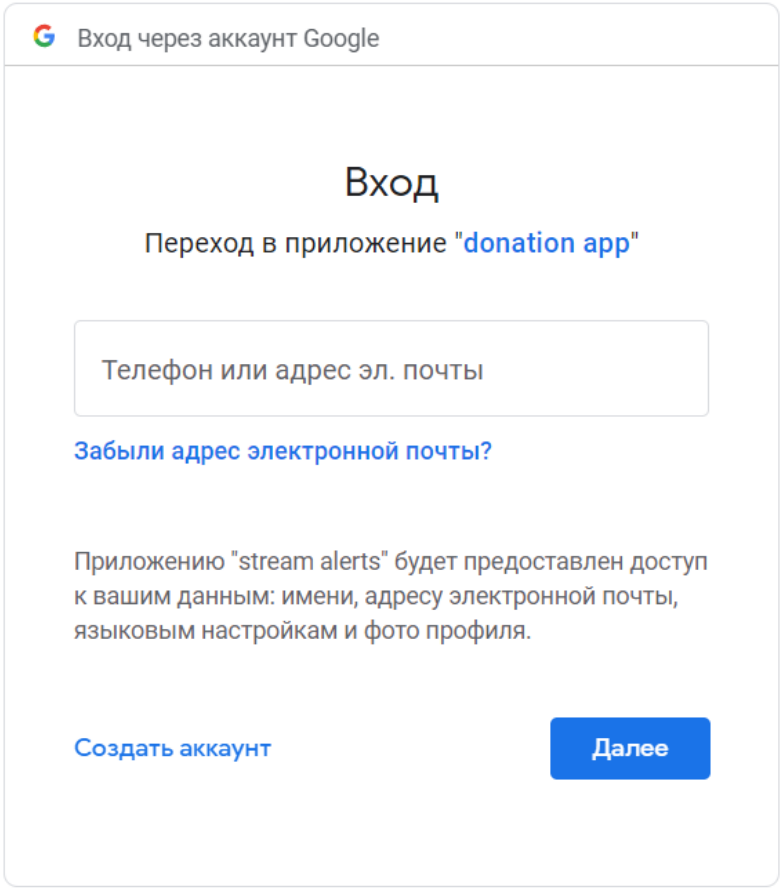


Рисунок 3.3 – Форма для входу в Google аккаунт

Якщо реєстрація/вхід пройшли успішно, то користувач потрапляє на сторінку з навігацією. Сторінка зображено на рисунку 3.5.

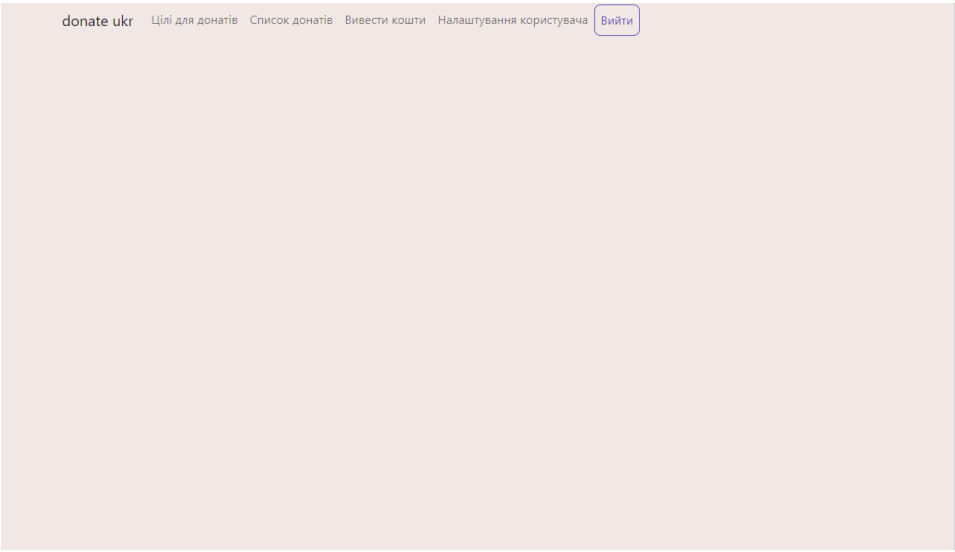


Рисунок 3.4 – Сторінка з навігацією

Користувач може перейти на такі сторінки:

– цілі для донатів;
– список донатів;
– вивести кошти;
– налаштування користувача.

Сторінка з списком донатів містить таблицю, що показує всі донати отримані користувачем на його цілі. Таблиця містить 4 поля: дата, ім’я, повідомлення та сума. Сторінка з списком донатів зображена на рисунку 3.6.

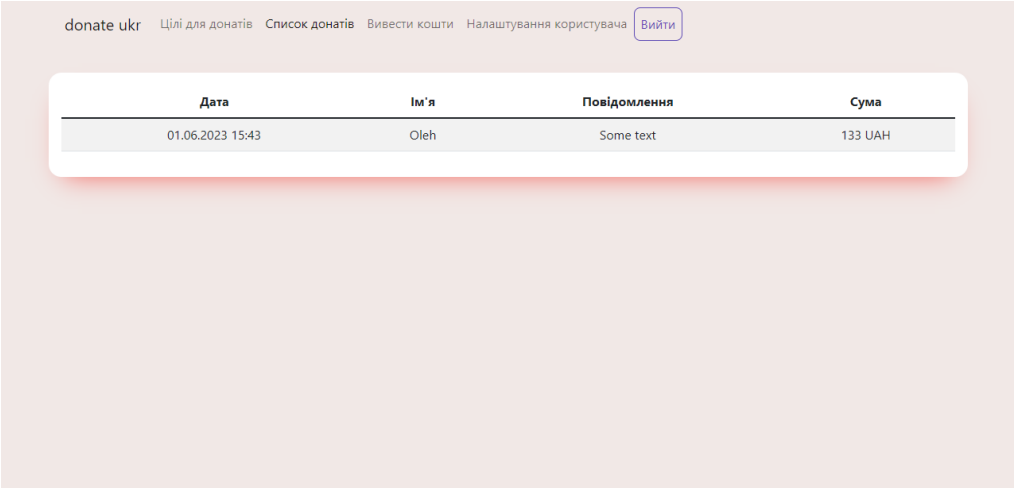


Рисунок 3.5 – Сторінка з списком донатів

Сторінка з цілями для донатів поточного користувача містить всі цілі для донатів користувача. Кожна ціль для донатів містить такі дані: назва цілі, опис цілі, загальна сума збору та поточна сума збору. Також користувач може редагувати кожну свою ціль. Він може змінювати поле опис цілі та поле загальна сума збору. Для того щоб редагувати ціль, користувач повинен змінити дані у відповідних полях та натиснути кнопку зберегти. Також внизу сторінки є форма для створення нових цілей для донатів. Форма містить 4 обов’язкові поля: назва цілі, опис цілі, номер карти для виводу коштів та загальна сума збору. Щоб створити ціль потрібно ввести дані у всі поля форми та натиснути кнопку створити ціль. Після цього ціль для донатів успішно створиться і з’явиться у списку з цілями користувача.

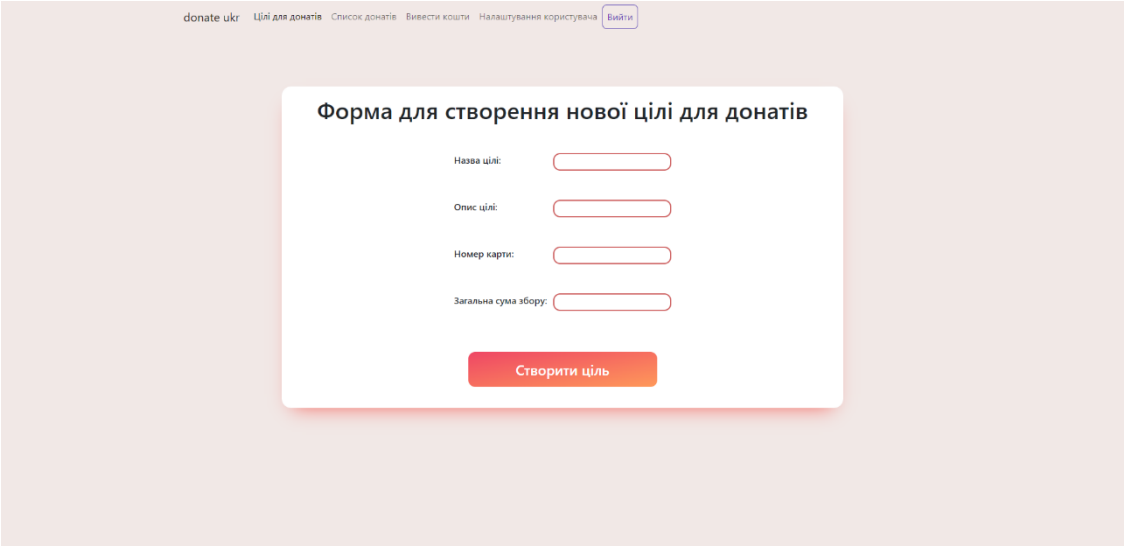


Рисунок 3.6 – Форма для створення нових цілей для донатів

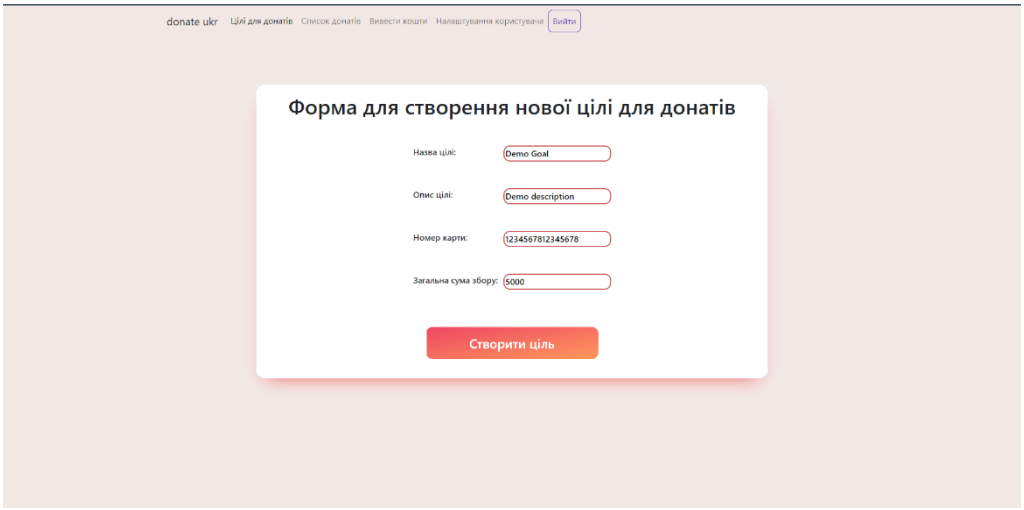


Рисунок 3.7– Форма із заповненими даними

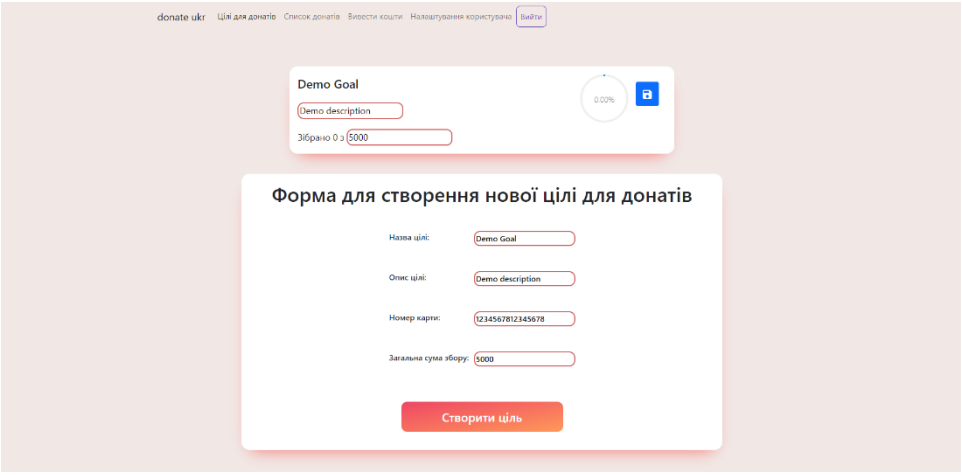


Рисунок 3.8 – Вигляд сторінки після натискання кнопки створити ціль

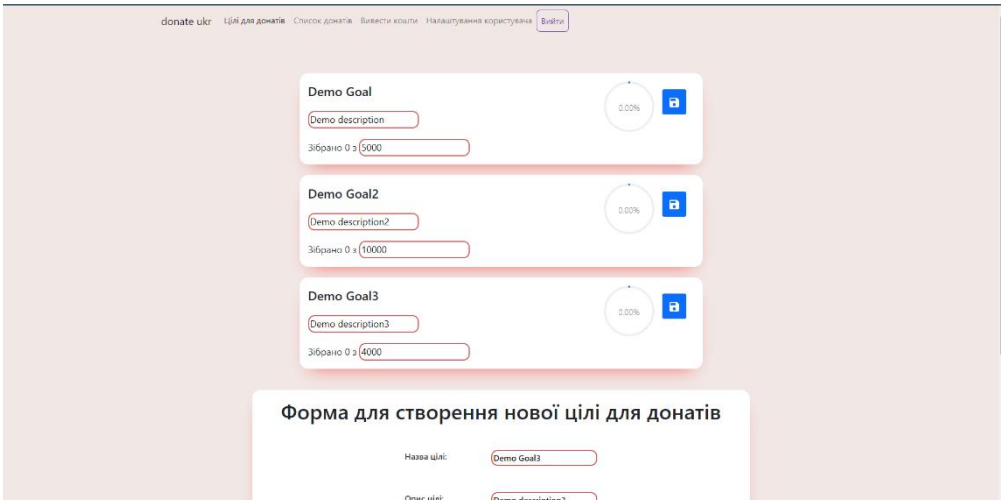


Рисунок 3.9 – Вигляд сторінки коли користувач має декілька цілей

Сторінка з налаштуваннями користувача містить посилання на сторінку донацій користувача. Посилання на сторінку донацій користувача складається з статичної частини та унікального ідентифікатора користувача. На цій сторінці користувач може ввести нове значення в поле з унікальним ідентифікатором користувача і натиснути кнопку зберегти, цим самим змінити свій унікальний ідентифікатор та своє посилання на сторінку донацій. Також на цій сторінці є кнопка яка дозволяє скопіювати своє посилання на сторінку донацій, щоб мати зручну можливість поділитися ним з іншими користувачами.

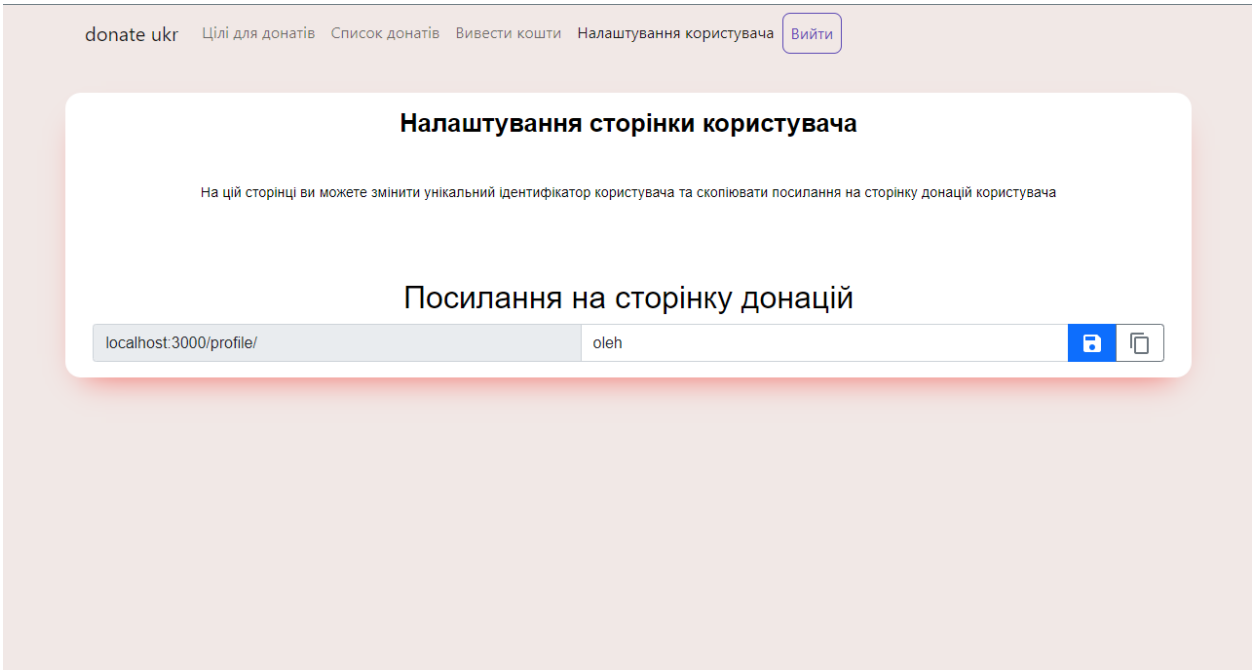


Рисунок 3.10 – Сторінка з налаштуваннями користувача

Якщо перейти по посиланню на сторінку донацій користувача, то побачимо сторінку з усіма цілями для донатів відповідного користувача та формою для здійснення донатів на обрану ціль користувача. Кожна ціль для донатів містить такі дані: назва цілі, опис цілі, загальна сума збору та поточна сума збору.

Форма для здійснення донатів містить 4 обов'язкові поля: ім'я автору донату, повідомлення донату, сума донату та ціль на яку робиться донат. Для того щоб здійснити донат, потрібно заповнити всі поля форми та обрати з списку ціль на яку потрібно зробити донат. Після цього натиснути кнопку зробити донат.

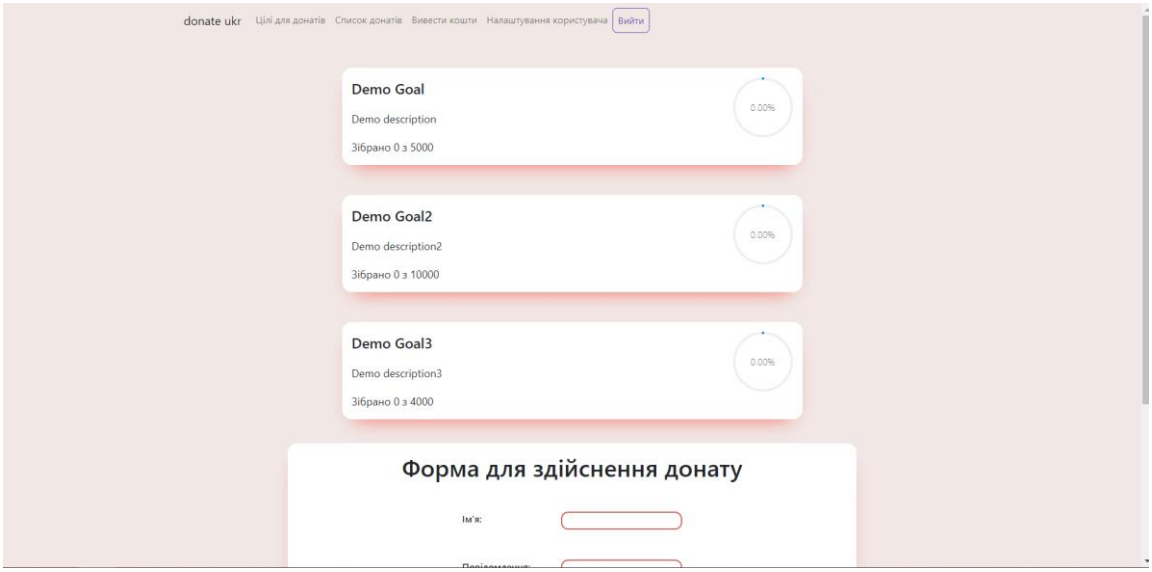


Рисунок 3.11 – Сторінка з цілями для донату конкретного користувача

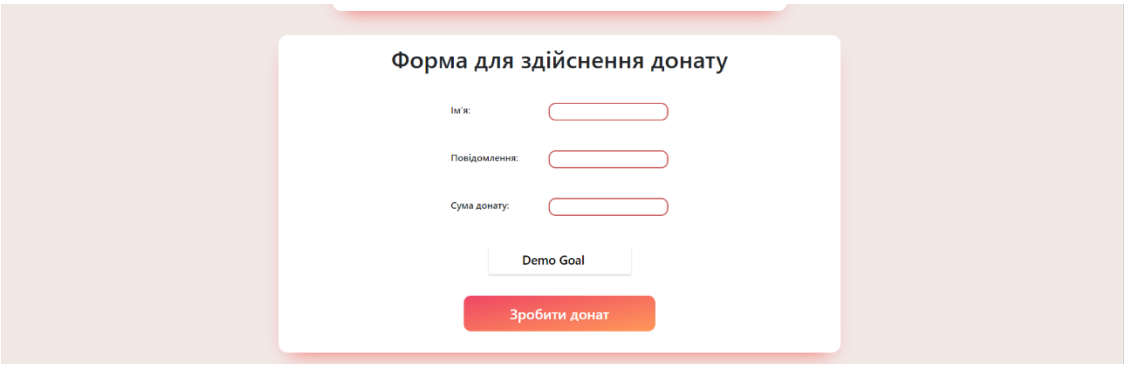


Рисунок 3.12 – Форма для здійснення донату

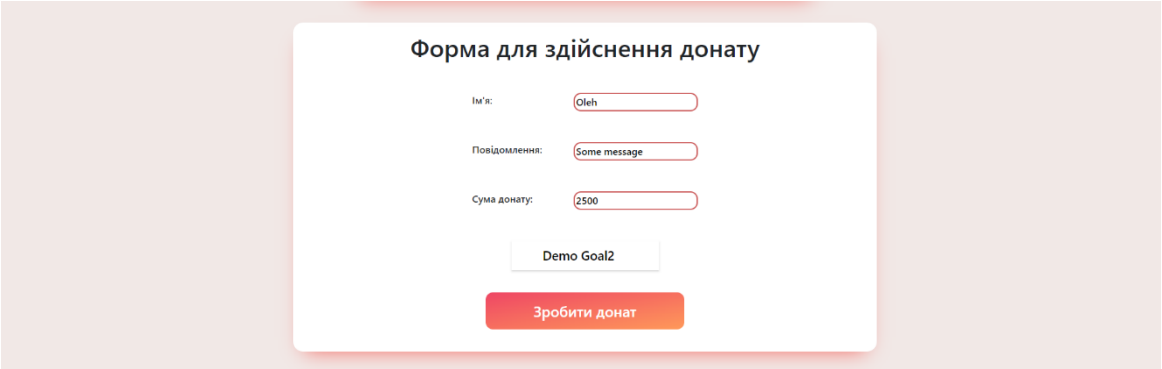


Рисунок 3.13 – Форма для здійснення донату із заповненими даними

Після натискання кнопки у новій вкладці браузеру відкриється сторінка платіжної системи Fondy з формою для оплати.

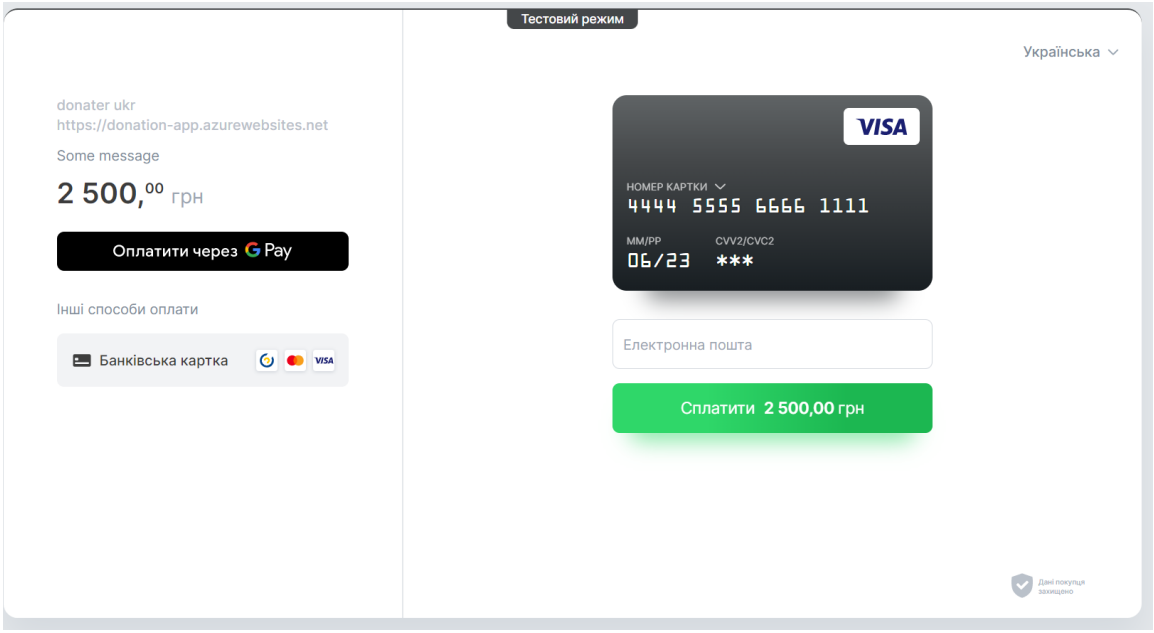


Рисунок 3.14 – Сторінка для оплати

Після оплати на сторінці відображається відповідне повідомлення з інформацією про платіж.

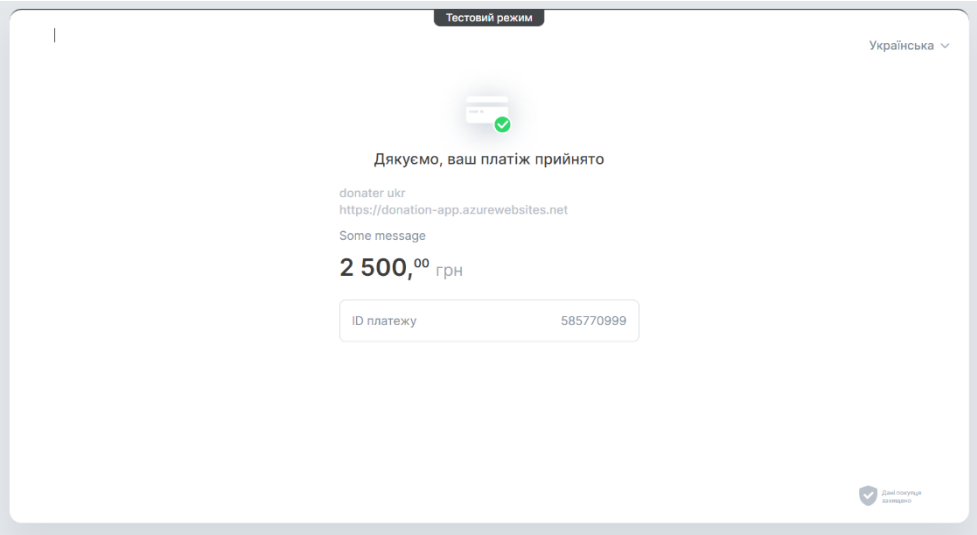


Рисунок 3.15 – Сторінка з успішним платежом

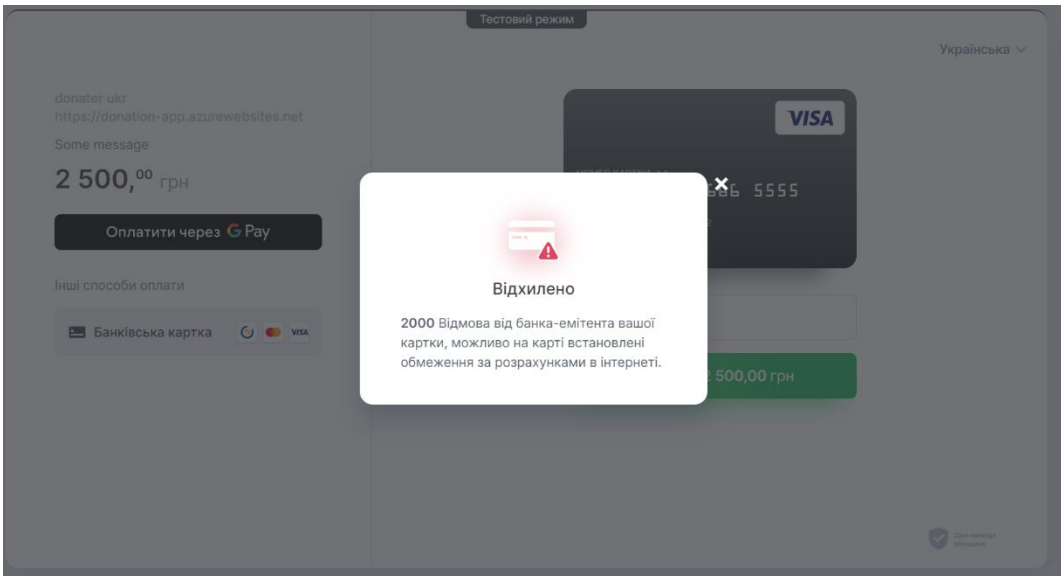


Рисунок 3.16 – Сторінка з не успішним платежом

Якщо платіж пройшов успішно, то поточна сума збору для цілі, на яку здійснювався донат, збільшується на суму донату.

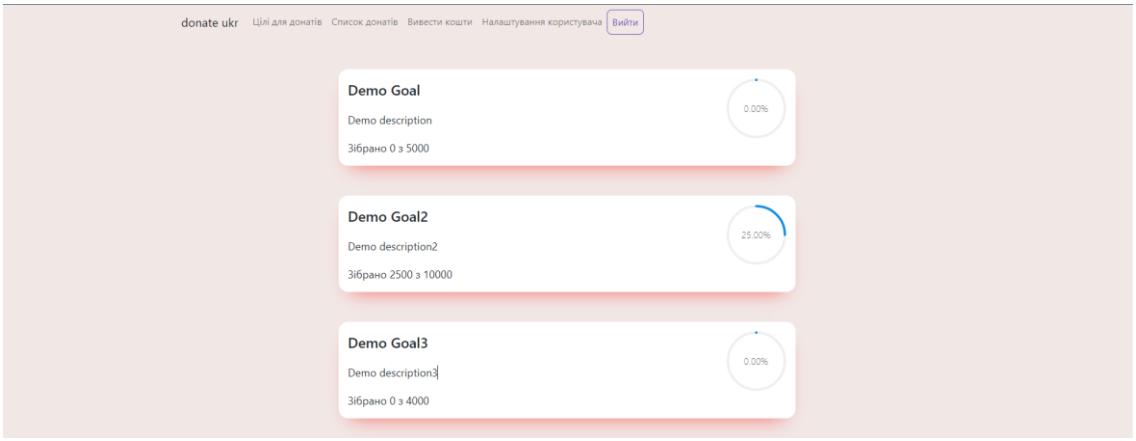


Рисунок 3.17 – Сторінка з цілями для донатів користувача

Сторінка для виводу коштів містить список карток з доступними сумами для виводу.

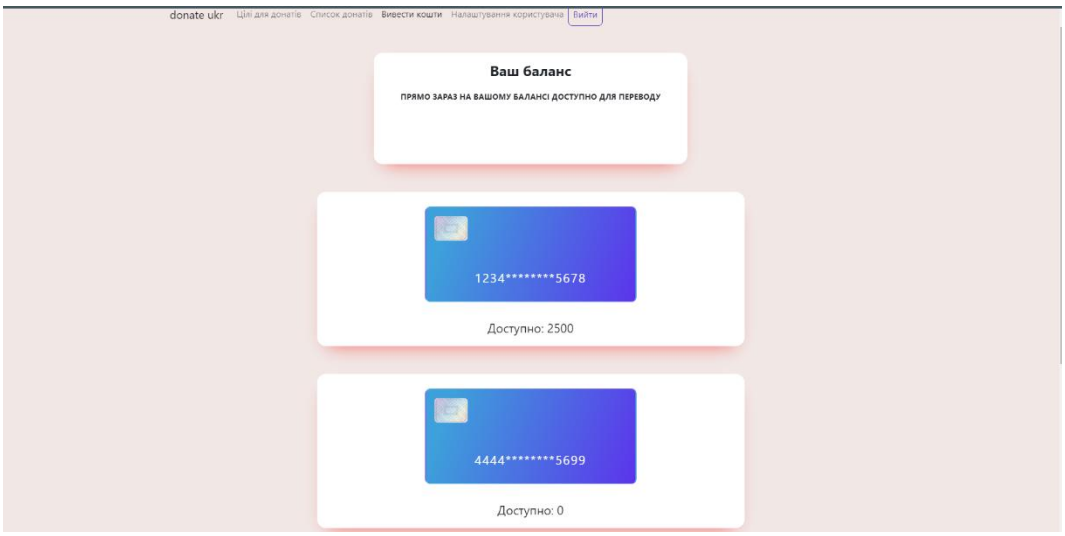


Рисунок 3.18 – Сторінка для виводу коштів

Внизу сторінки знаходиться форма для створення запиту на виведення коштів. Форма містить два поля: номер карти для виводу, який обирається з доступних карток та сума для виводу. Щоб створити запит на виведення коштів потрібно заповнити відповідні поля форми і натиснути кнопку створити запит.

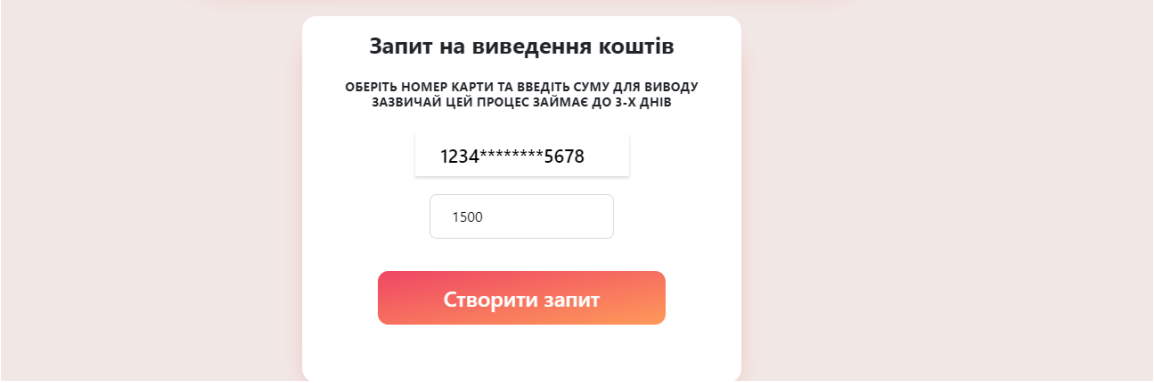


Рисунок 3.19 – Форма для виводу коштів

Після того як запит буде створено, доступний баланс на картці буде оновлено.

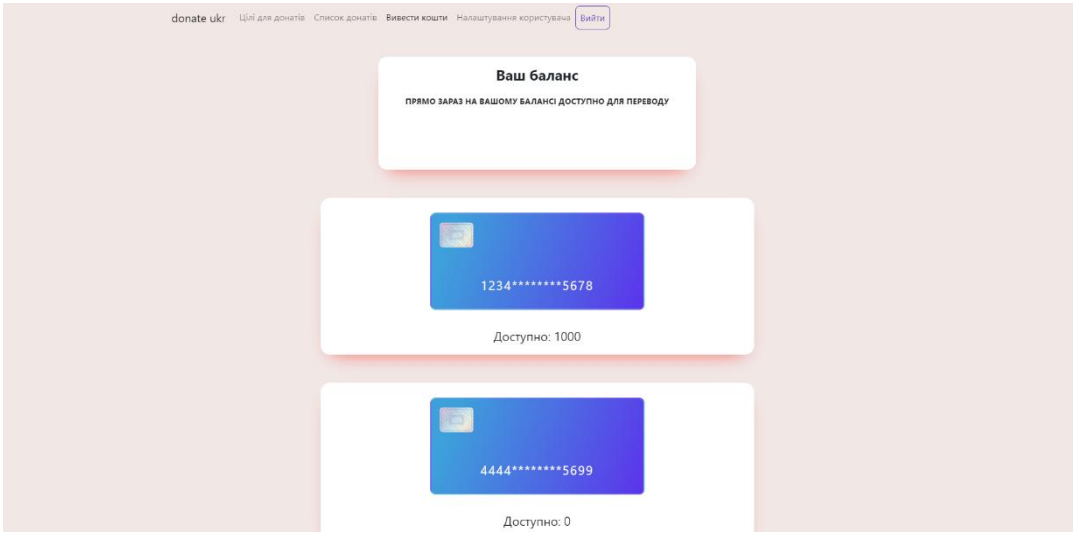
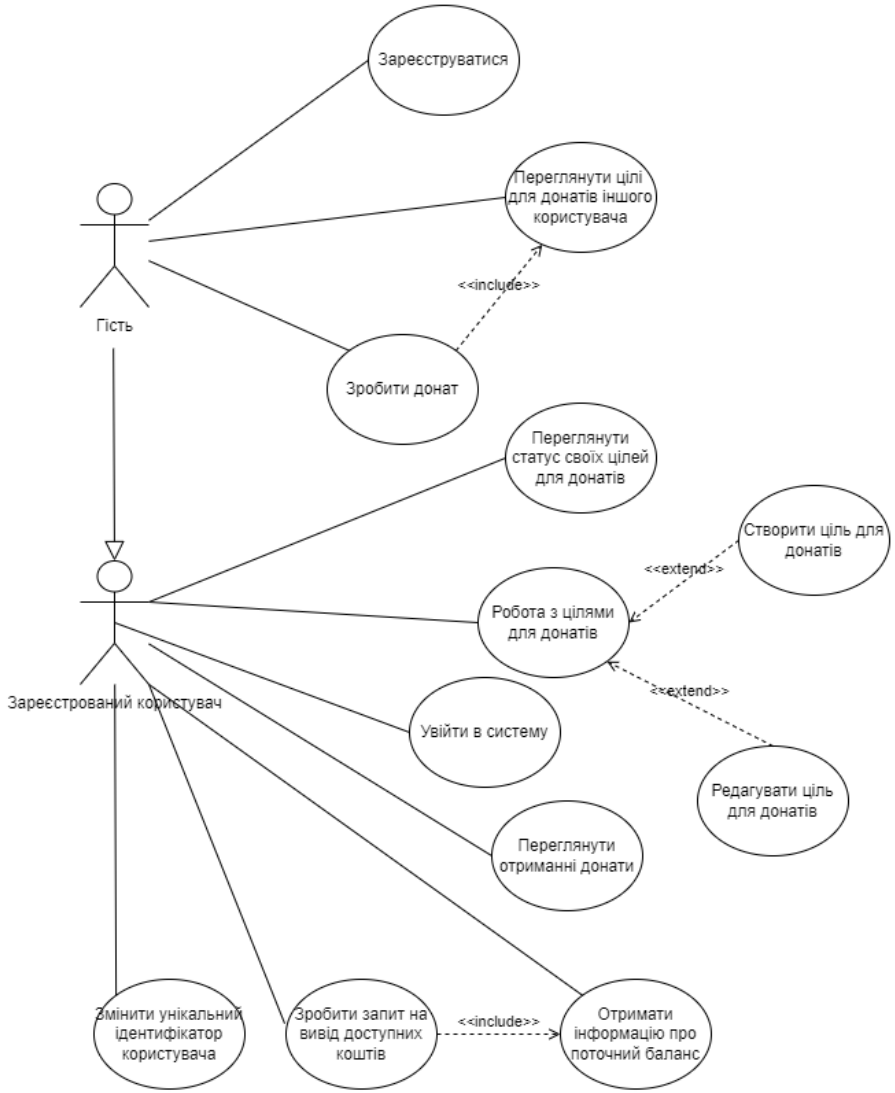


Рисунок 3.20 – Сторінка для виводу коштів



					КПІ.ІТ-9126.045440.06.99.ССВ	
						Літера / Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна варіантів використань	
Розробив		Тонкошкур О. Р.				Аркуш / Аркушів
Перевірив		Ліщук К. І.				
Т. кон.						
Н. кон.		Головченко М.			Веб-сервіс для донатів	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-91
Затвердив		Жаріков Е. В.				

КПІ.ІТ-9126.045440.07.99.СБД

					КПІ.ІТ-9126.045440.07.99.СБД	
Зм.	Арк.	№ документа	Підпис	Дата	Схема бази даних	Літера / Маса / Масштаб
Розробив		Тонкошкур О. Р.				
Перевірив		Ліщук К. І.				Аркуш / Аркушів
Т. кон.						
Н. кон.		Головченко М.			Веб-сервіс для донатів	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-91
Затвердив		Жаріков Е. В.				

					КПІ.ІТ-9126.045440.08.99.КЕ	
						Літера / Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	Креслення вигляду екранних форм	
Розробив		Тонкошкур О. Р.				Аркуш / Аркушів
Перевірив		Ліщук К. І.				
Т. кон.						
Н. кон.		Головченко М.			Веб-сервіс для донатів	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-91
Затвердив		Жаріков Е. В.				