Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система для організації збору коштів. Серверна частина для донорів та адміністраторів
(тема)

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

Матвій БОЙЧЕНКО
(Власне ім'я, ПРІЗВИЩЕ)

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

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

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

2025 р.

Харківський національний університет радіоелектроніки

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

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Бойченко Матвію Юрійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Програмна система для організації збору коштів. Серверна частина для донорів та адміністраторів
Затверджена наказом по університету від 19.05.2025 р. № 397 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 19.06. 2025
3. Вихідні дані до роботи Розробити серверну частину програмного рішення, що забезпечує управління ініціативами, створення зборів, облік пожертв та інтеграцію з платіжними платформами, використовуючи архітектуру Clean Architecture, мову програмування C# та платформу .NET 8, для зберігання даних – СУБД PostgreSQL.
4. Перелік питань, що потрібно опрацювати в роботі: Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	10.04.2025	*виконано*
2	Створення специфікації ПЗ	15.04.2025	*виконано*
3	Проектування ПЗ	16.04.2025	*виконано*
4	Розробка ПЗ	24.05.2025	*виконано*
5	Тестування ПЗ	27.05.2025	*виконано*
6	Оформлення пояснювальної записки	01.06.2025	*виконано*
7	Підготовка презентації та доповіді	06.06.2025	*виконано*
8	Попередній захист	16.06.2025	*виконано*
9	Нормоконтроль, рецензування	16.06.2025	*виконано*
10	Здача роботи у електронний архів	17.06.2025	*виконано*
11	Допуск до захисту у зав. кафедри	18.06.2025	*виконано*

Дата видачі завдання «08» «квітня» 2025 р.

Здобувач _____ Матвій БОЙЧЕНКО
(підпис)

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

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра, 52 стор., 10 рис., 1 табл., 13 джерел, 2 додатки.

ДОНАТОР, КОРИСТУВАЧ, ФАНДРЕЙЗИНГ, .NET 8, C#, CLEAN ARCHITECTURE, EMAIL, OAUTH 2.0, PUSH-СПОВІЩЕННЯ, STRIPE API

Об’єкт розробки – програмна система із серверною та мобільною частинами для організації збору коштів на соціальні, екологічні й медичні ініціативи.

Мета розробки – створити масштабовну та безпечну платформу, що забезпечує реєстрацію користувачів, ведення донорських профілів, автоматизовану розсилку сповіщень і адміністративний моніторинг активності.

Метод реалізації – стек технологій .NET 8 (C#), Entity Framework Core, Identity, PostgreSQL, ASP.NET Core Web API, SignalR, Firebase Cloud Messaging, React Native, а також інтеграція з платіжними сервісами (Stripe/LiqPay). Архітектурний підхід – Clean Architecture із розподілом на шари Domain, Application, Infrastructure та Presentation.

Результат розробки – серверне рішення, що дає змогу створювати ініціативи, збирати пожертви, автоматично надсилати email- і push-сповіщення підписникам, а також надає адміністраторам інструменти керування категоріями та моніторингу активності. Система підтримує автентифікацію через email і Google OAuth, верифікацію акаунтів, REST- та gRPC-API для зовнішніх сервісів і відповідає вимогам GDPR щодо захисту персональних даних.

ABSTRACT

.NET 8, C#, CLEAN ARCHITECTURE, EMAIL, OAUTH 2.0, PUSH NOTIFICATIONS, STRIPE API, DONOR, USER, FUNDRAISING.

Object of development – a software system with server and mobile parts for organising fundraising campaigns for social, environmental and medical initiatives.

Purpose of development – to create a scalable and secure platform that provides user registration, donor profile management, automated notifications and administrative activity monitoring.

Method of implementation – technology stack .NET 8 (C#), Entity Framework Core, Identity, PostgreSQL, ASP.NET Core Web API, SignalR, Firebase Cloud Messaging, React Native, with integration of payment services (Stripe/LiqPay). The architectural approach is Clean Architecture with Domain, Application, Infrastructure and Presentation layers.

Result of development – a web and mobile application that enables creation of initiatives, collection of donations, automatic distribution of email and push notifications to subscribers, and provides administrators with tools to manage categories and monitor activity. The system supports authentication via email and Google OAuth, account verification, REST and gRPC APIs for external services, and complies with GDPR personal-data protection requirements.

ЗМІСТ

ВСТУП

Актуальність роботи обумовлена широким застосуванням цифрових технологій у сфері збору коштів для соціальних, екологічних та медичних проектів. Потреба в зручних та ефективних системах для управління ініціативами та донатами є очевидною, оскільки значна частина сучасних благодійних організацій і волонтерських груп активно використовують онлайн-інструменти для взаємодії зі своїми донорами та підтримки постійної комунікації з ними. Розробка програмної системи, що дозволяє централізовано та прозоро організовувати збір коштів, відслідковувати ефективність кампаній та оперативно сповіщати учасників, є актуальною задачею з огляду на стрімке зростання популярності волонтерських та благодійних ініціатив.

Метою кваліфікаційної роботи є створення програмної системи, яка забезпечить автоматизацію та ефективне управління процесами реєстрації користувачів, адміністрування донорів, надсилання сповіщень та контролю активності.

Об’єкт дослідження – процес автоматизації збору коштів для різноманітних ініціатив.

Предмет дослідження – програмна система для реєстрації, керування профілями донорів та автоматизації сповіщень користувачів.

Для реалізації поставлених завдань використовувалися сучасні технології розробки, а саме платформа .NET 8 (мова C#), веб-фреймворк ASP.NET Core, інструменти для роботи з базами даних (Entity Framework Core, PostgreSQL), сервіси авторизації (Identity, OAuth 2.0), сервіси сповіщень (Firebase Cloud Messaging, SMTP), а також мобільні технології React Native.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

Сучасне суспільство стикається з численними соціальними, екологічними та медичними викликами, для вирішення яких постійно необхідні фінансові ресурси. Благодійні фонди, волонтерські рухи, громадські організації, а також індивідуальні активісти потребують ефективних інструментів для залучення фінансування, прозорого ведення зборів коштів та підтримки постійної комунікації з донорами.

Особливо актуальною є проблема швидкого реагування на соціальні та гуманітарні кризи, які вимагають негайного збору великих сум для надання допомоги постраждалим. Наприклад, стихійні лиха, військові конфлікти, епідемії вимагають швидкої мобілізації ресурсів. В таких умовах відсутність автоматизованих, прозорих та зручних систем збору коштів може значно уповільнити процес надання допомоги.

Крім того, прозорість та звітність перед донорами стали ключовими критеріями довіри до організацій, що збирають кошти. Люди хочуть бачити, на що саме йдуть їхні внески, мати можливість контролювати статус кампанії, а також отримувати зворотній зв’язок у вигляді звітів про використання коштів. Тому актуальним є створення програмної системи, яка надає повну прозорість у веденні зборів, дозволяє слідкувати за ходом реалізації ініціатив і забезпечує зручний спосіб сповіщення донорів про поточний статус кампаній.

Дослідження потреб сучасних благодійних організацій та соціальних рухів виявляє такі ключові запити:

– простота та зручність збору коштів. організації прагнуть використовувати сервіси, що не потребують значних витрат часу та ресурсів на адміністрування зборів та комунікацію з донорами;

– автоматизація комунікації. необхідність регулярно інформувати донорів про статус кампаній, нові ініціативи та їхні результати є важливим чинником залучення та утримання донорів;

– прозорість фінансових операцій. донори потребують впевненості, що їхні внески використовуються ефективно та за призначенням;

– інтеграція з іншими сервісами та платформами. організаціям важливо легко інтегрувати свої профілі та історію донорів з іншими цифровими платформами та зовнішніми сервісами;

– зручність для мобільних користувачів. зростання популярності мобільних пристроїв зумовлює необхідність мобільного доступу до систем збору коштів із можливістю швидких платежів і оперативного отримання сповіщень.

Практична цінність збору коштів у вигляді краудфандингу чи донорської підтримки є особливо високою завдяки можливості швидко залучати значну кількість небайдужих людей навколо конкретних соціальних, медичних чи екологічних ініціатив. На відміну від традиційних методів фінансування, таких як гранти, державні субсидії або позики, збір коштів через публічні кампанії забезпечує більш швидке та гнучке отримання фінансових ресурсів. При цьому ініціатори проєктів не обмежені складними бюрократичними процедурами, що дозволяє ефективно реагувати на термінові ситуації [1].

Однією з головних переваг зборів коштів є високий рівень залучення громадськості. Завдяки широкій інформаційній підтримці та активному поширенню через соціальні мережі й електронні ресурси, збори мають можливість охопити значну кількість потенційних донорів у короткий час. Такий підхід суттєво розширює аудиторію, залучаючи не тільки професійних благодійників чи великі організації, а й звичайних людей, які готові робити навіть невеликі, але регулярні внески.

Ще однією ключовою перевагою зборів коштів є прозорість використання отриманих ресурсів. На відміну від деяких традиційних способів фінансування, публічні збори зазвичай супроводжуються відкритим звітуванням та постійною комунікацією з донорами щодо витрачених коштів і результатів їх використання. Це формує високий рівень довіри серед донорів і спонукає до подальшої участі в наступних ініціативах.

Крім того, організація збору коштів у цифровій формі надає значні переваги з погляду зручності та доступності. Донори можуть здійснювати внески з будь-якої точки світу, використовуючи електронні платіжні сервіси, що значно спрощує

участь у благодійних кампаніях. Таким чином, цифрові збори є більш гнучкими та доступними порівняно з традиційними офлайн-методами, наприклад, фізичними благодійними заходами чи банківськими переказами, які потребують додаткових часових витрат або особистої присутності донорів.

Таким чином, збори коштів через цифрові платформи мають істотні переваги перед традиційними формами фінансування, забезпечуючи прозорість, гнучкість, швидкість і широке залучення громадськості до суспільно важливих ініціатив. Це дозволяє не тільки ефективно вирішувати актуальні проблеми суспільства, а й зміцнювати соціальні зв’язки, підвищувати рівень громадянської відповідальності та активно залучати людей до вирішення гострих соціальних питань.

Таким чином, аналіз предметної області демонструє очевидну необхідність створення спеціалізованої програмної системи, яка б ефективно задовольняла перелічені потреби та дозволяла організаціям і приватним особам організовувати прозорі та результативні кампанії з фінансування різноманітних суспільно важливих ініціатив.

1.1 Аналіз подібних систем

Системи цифрового фандрейзингу бувають різних типів. Існують як класичні краудфандингові платформи, так і мобільні/банківські додатки з благодійними функціями. Однак наразі саме традиційні канали – колективні скриньки та особисті волонтерські збори – залишаються найпопулярнішими: близько 29% українців роблять внески у стаціонарні скриньки в магазинах і супермаркетах, ще 11% – жертвують через волонтерів на публічних заходах. Натомість лише незначна частина громадян користувалася краудфандингом.З технічної точки зору найпоширенішими є мобільні застосунки банків: наприклад, у Monobank – функція «банки», а в Приват24 – сервіс «конверти», де користувач може створити збір із описом, ціллю і оновленням прогресу. Також дедалі частіше застосовуються QR-донати і смс-пожертви, а платіжні шлюзи використовуються благодійними фондами для прийому онлайн-платежів.

Попри технічний прогрес, цифрова благодійність має низку суттєвих

викликів. Головна з них – низька довіра: багато донорів остерігаються, що їхні кошти підуть не за призначенням, а небагато хто перевіряє прозорість фондів. Як наголошується у дослідженні з UX-дизайну, відсутність чіткої інформації про те, куди потрапляють гроші, й нечітка подача мети проекту миттєво знижують довіру користувача.

Аналогічно, будь-яка «фрикція» у процесі пожертви – зайві кроки, заплутана навігація або неадаптованість під мобільні – часто призводить до відмови від донату. Ще один психологічний бар'єр – перевантаження вибором. Класичні спостереження показують, що коли донору пропонують забагато параметрів (різні суми чи категорії пожертв, багато опцій у формі), він просто «заморожується» й відмовляється діяти. Тому зайва складність форм може нашкодити конверсії.

Інтерфейс додатку Donate24 (див. рис. 1) показовий: він щоденно пропонує користувачам актуальні збори на потреби ЗСУ, дозволяючи відфільтрувати кампанії за темою, легко перейти до донату чи скопіювати реквізити з одним

Такі рішення посилюють ефект миттєвої дії – якщо користувач бачить кнопку «Пожертвувати зараз» на початку заклику, це значно підвищує відгук. Також мобільні застосунки використовують push-повідомлення для рекрутингу донорів і мотивації «дати зараз»: вони спонукатимуть зробити внесок своєчасно та нагадують «давати часто». Не менш важливий аспект – соціальний резонанс: коли люди діляться у мережах інформацією про власний донат, це може генерувати нові залучення (соціальне підтвердження) кліком.

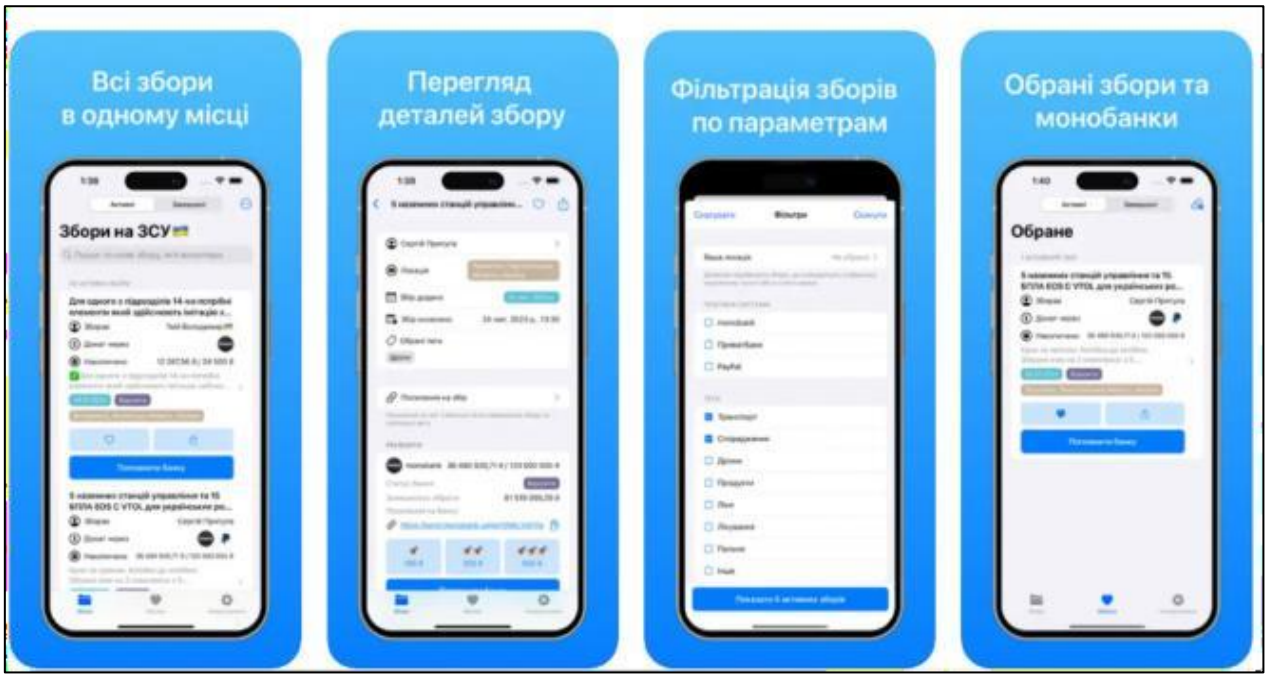

Рисунок 1.1 – Donate24 (за даними [2])

Найкращі практики у комунікації та технічному вирішенні формули руху донорів зосереджені на простоті та прозорості. Наприклад, показ реального прогресу кампанії у вигляді progress bar та конкретних числових статистик значно підвищують довіру користувача. Важливо також мінімізувати кількість полів у формі донату та забезпечити «UX у 2 кліки»: досвід показує, що перенесення головного заклику до дії на початок (наприклад, кнопка «Donate» у перших рядках листа) різко збільшує конверсію. Після завершення транзакції корисно одразу відображати екран-підтвердження з візуальною анімацією або коротким повідомленням-подякою, щоб донор відчув «ефект завершеності» (feedback) і розумів, що його пожертва зарахована. Автоматизовані листи чи підтвердження з докладними даними про витрати також допомагають утримати довіру і мотивують повторні внески.

Прикладами вдалих рішень є як українські розробки, так і світові сервіси. Зазначимо, що функція «банок» у мобільному додатку Monobank зібрала понад 1 млрд євро від початку війни, а минулого року через неї пройшло 43,68 млрд грн донатів (Monobank оприлюднив ці дані). Подібний інструмент – «Конверти» у Приват24 (див. рис. 1.2) – дозволяє групувати збори та прозоро показувати збірникам опис і прогрес. Ще один приклад – сайт VolunteeringUkraine, який

публічно відображає всі пожертви на фронтовий проєкт Front Line Kit, що надходять через платіжний шлюз WayForPay; так донори бачать повну прозорість усіх транзакцій. Міжнародні платформи як Patreon чи Donorbox стають стандартними для творчих і навчальних проєктів, а українські рішення активно запозичують ці підходи. Разом вони ілюструють прагнення зробити цифрову благодійність зручною та прозорою, що важливо для подальшого зростання фондів і залучення довіри суспільства.

Рисунок 1.2 – Конверт у Private24 (за даними [3])

1.2 Аналіз проблем, з якими стикаються системи збору коштів

Незважаючи на очевидні переваги використання цифрових платформ для збору коштів, існує низка суттєвих проблем, що перешкоджають їх повноцінному та ефективному використанню. Однією з найважливіших є проблема недостатньої прозорості та звітності перед донорами. Недостатнє або несвоєчасне інформування користувачів про використання їхніх внесків може викликати сумніви щодо чесності та надійності платформи, а отже, значно зменшує довіру потенційних

донорів.

Також гостро стоїть проблема шахрайських зборів та махінацій. З розвитком цифрових технологій зростає кількість фіктивних кампаній, метою яких є отримання коштів незаконним шляхом. Відсутність належного контролю та систем верифікації ініціаторів зборів може призводити до випадків шахрайства, що завдає значної шкоди репутації платформ і негативно впливає на готовність донорів робити внески в майбутньому.

Не менш актуальною є проблема технічної доступності та зручності взаємодії системами збору коштів. Складний або незручний інтерфейс, погано оптимізована мобільна версія, відсутність якісних механізмів пошуку та фільтрації ініціатив призводять до зменшення залученості користувачів. В умовах сучасного світу, де значна частина донорів взаємодіє з сервісами саме через мобільні пристрої, така проблема може значно знижувати ефективність кампаній.

Окрім того, складність інтеграції різних платіжних систем часто створює додаткові труднощі для донорів. Відсутність підтримки популярних платіжних сервісів, необхідність реєстрації або надмірно складні процедури оплати можуть призводити до відмови користувачів від донатів навіть у випадках, коли вони налаштовані підтримати ініціативу.

Додатковою проблемою є недостатній рівень автоматизації процесів комунікації з донорами. Якщо система не забезпечує регулярних автоматизованих сповіщень про статус кампаній, нові ініціативи чи успіхи зборів, це може знизити мотивацію донорів до повторних внесків та спричинити втрату постійних учасників кампаній.

Таким чином, для ефективного функціонування сучасних платформ для збору коштів важливою є розробка системних рішень, що забезпечать високий рівень прозорості, захисту від шахрайства, технічної доступності та інтегрованості з популярними платіжними системами, а також автоматизацію комунікації для підтримки постійного залучення донорів. Саме ці аспекти потребують ретельної уваги під час розробки та реалізації програмних систем для збору коштів.

1.3 Виявлення та виправлення проблеми

Для виявлення проблем, з якими стикаються сучасні системи збору коштів, було проведено аналіз існуючих цифрових платформ, огляд тематичних форумів і дослідження відгуків користувачів. Аналіз показав, що найбільш критичними проблемами є недостатня прозорість, складні процедури інтеграції платіжних сервісів, відсутність належної автоматизації комунікації з донорами та низький рівень захисту від шахрайства.

На підставі виявлених недоліків було сформовано основні напрями, за якими планувалось вирішити зазначені проблеми шляхом розробки та впровадження відповідного програмного рішення.

Для забезпечення прозорості використання коштів у запропонованій системі реалізовано механізм автоматичного звітування перед донорами. Користувачі отримують регулярні email- та push-сповіщення про поточний стан зборів, а також мають доступ до персонального кабінету, де представлені детальні відомості щодо їхніх внесків. Це підвищує довіру до системи та стимулює донорів до регулярної участі у зборі коштів.

Проблема шахрайства була вирішена шляхом впровадження системи верифікації користувачів. Реєстрація користувачів здійснюється з обов’язковою перевіркою email-адреси, а створення нових ініціатив передбачає додаткове підтвердження особи через отримання спеціального коду. Адміністративна панель дозволяє оперативно контролювати та блокувати підозрілі кампанії, що забезпечує високий рівень безпеки і зменшує ймовірність появи фальшивих зборів.

Для вирішення проблеми складності інтеграції платіжних систем у проекті було реалізовано інтеграцію з популярними та простими у використанні платіжними сервісами, такими як Stripe і LiqPay. Ці сервіси були обрані з огляду на їх простоту налаштування, високу швидкість проведення платежів та популярність серед користувачів.

Проблема недостатньої автоматизації комунікації з донорами вирішується через модуль сповіщень, який дозволяє здійснювати автоматичні email- і push-розсилки. Користувачі можуть налаштовувати частоту та тип сповіщень відповідно

до своїх уподобань, що підвищує залученість аудиторії, забезпечує регулярне інформування та мотивує до повторних пожертвувань.

Таким чином, за допомогою розробленого рішення було усунуто більшість проблем, притаманних існуючим системам збору коштів. Реалізація зазначених заходів забезпечила ефективність, прозорість, зручність користування і високий рівень безпеки платформи, що відповідає сучасним вимогам благодійних та громадських ініціатив.

1.4 Постановка задачі

Враховуючи проведений аналіз предметної області та визначені проблеми, у межах даної кваліфікаційної роботи було сформульовано такі задачі, що стосуються розробки функціоналу модулів користувачів, донорів, сповіщень та адміністративного контролю:

– розробити модуль реєстрації та авторизації користувачів, що підтримує стандартну реєстрацію через Email, а також автентифікацію за допомогою google oauth 2.0 [4];

– реалізувати профілі користувачів, що містять детальну інформацію про їх внески, налаштування профілю та особисті дані, необхідні для ведення історії взаємодії з системою;

– забезпечити механізм верифікації користувачів шляхом надсилання унікального коду на електронну пошту для отримання дозволу на створення власних зборів коштів;

– створити функціонал збереження історії донатів, який дозволить сортувати та здійснювати пошук донорів за їхньою активністю;

– розробити публічний api для інтеграції з іншими платформами, що дозволяє отримувати інформацію про донорів із застосуванням авторизації за секретними api-ключами;

– створити систему автоматизованих email- і push-сповіщень для користувачів, яка дозволить здійснювати підписку на конкретні ініціативи чи категорії, інформувати про нові збори та оновлення їхнього статусу;

– забезпечити можливість налаштовувати частоту, тип та формат отримуваних сповіщень відповідно до потреб кожного користувача;

– реалізувати адміністративний модуль для управління категоріями ініціатив, а також інструменти для моніторингу активності користувачів, перевірки та блокування потенційно фальшивих зборів.

Виконання цих задач сприятиме створенню надійного, зручного та прозорого функціоналу, що задовольняє потреби кінцевих користувачів, донорів і адміністраторів у процесі організації та проведення зборів коштів

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Функціональні вимоги

На основі проведеного аналізу предметної області, потреб потенційних користувачів і виявлених проблем було визначено такі загальні функціональні вимоги до програмної системи:

– реєстрація та авторизація користувачів. Система повинна підтримувати створення облікових записів, авторизацію через електронну пошту та зовнішні сервіси (наприклад, google oauth);

– управління профілями. Користувачі повинні мати можливість налаштовувати свій профіль, переглядати історію активності, свої внески та керувати підписками на різні ініціативи;

– керування ініціативами. Система має надавати можливість створювати, редагувати, видаляти та переглядати ініціативи із можливістю пошуку, фільтрації за категоріями та іншими параметрами;

– організація та проведення зборів коштів. Повинна забезпечуватися можливість створення та адміністрування кампаній для збору коштів, включаючи налаштування цільової суми, термінів збору, описів та інших атрибутів;

– інтеграція платіжних систем. Необхідно реалізувати підтримку інтеграції із загальновизнаними та безпечними платіжними системами (stripe, paypal, liqpay тощо) для здійснення транзакцій;

– моніторинг і звітність. Система повинна надавати інструменти для генерації звітів та статистичних даних за окремими кампаніями, загальною ефективністю зборів та активністю користувачів;

– автоматизована комунікація. Повинна бути реалізована можливість надсилати автоматичні сповіщення користувачам (email- та push-сповіщення) про нові ініціативи, статус поточних зборів, важливі оновлення та результати кампаній;

– адміністративне управління. Система повинна мати спеціалізований адміністративний інтерфейс для управління користувачами, ініціативами, категоріями зборів, контролю активності та блокування потенційно шахрайських

кампаній;

– підтримка мобільних пристроїв. Платформа має забезпечувати зручність використання через мобільні пристрої, включаючи мобільні додатки, адаптивну версію веб-інтерфейсу та інтеграцію з мобільними функціями (gps, push-сповіщення тощо);

– публічне API. Система повинна надавати зовнішнім сервісам можливість взаємодіяти з нею через захищений API для інтеграції з іншими цифровими платформами.

Задоволення цих функціональних вимог гарантує створення цілісної, ефективної, зручної та безпечної системи збору коштів, яка відповідає актуальним потребам сучасних користувачів і організацій.

2.2 Нефункціональні залежності

Поряд із функціональними вимогами, програмна система збору коштів має відповідати низці нефункціональних вимог, які визначають якість, безпеку та надійність розробленого програмного рішення:

– продуктивність. Система повинна забезпечувати високу швидкість реагування на дії користувачів, підтримувати стабільність роботи під навантаженням, зокрема в умовах великої кількості одночасних транзакцій і активних користувачів;

– масштабованість. Архітектура системи повинна бути спроектована з можливістю горизонтального та вертикального масштабування, що забезпечує стабільну роботу при зростанні навантаження або розширенні функціональності;

– безпека. Система повинна гарантувати безпеку персональних даних користувачів і платіжних транзакцій, відповідати стандартам gdpr, а також захищати дані від несанкціонованого доступу, атак і потенційних шахрайських дій;

– надійність і доступність. Система має підтримувати високий рівень доступності (не нижче 99,9%) і бути стійкою до збоїв, забезпечуючи коректну роботу навіть за умови виникнення технічних несправностей або помилок;

– зручність використання (usability). Інтерфейс системи повинен бути

інтуїтивно зрозумілим, простим і доступним для користувачів із різним рівнем підготовки, що дозволить їм швидко та ефективно взаємодіяти із функціоналом платформи;

– супроводжуваність і розширюваність. Архітектура має бути організована таким чином, щоб подальше супроводження, оновлення і доповнення новими функціями не потребували значних витрат часу і ресурсів розробників;

– переносимість. Система повинна мати можливість розгортання на різних хмарних платформах (наприклад, aws, azure) або локальних серверах без необхідності суттєвих змін у вихідному коді;

– сумісність. Забезпечення сумісності з основними сучасними браузерами (google chrome, firefox, safari, edge), а також підтримка роботи мобільних клієнтів на платформах Android та iOS.

Виконання зазначених нефункціональних вимог забезпечить високу якість роботи програмної системи, її стабільність, безпеку та зручність, що є критично важливим для ефективного вирішення поставлених завдань у сфері збору коштів.

2.3 Припущення та залежності

Передбачається, що користувачі мають стабільний доступ до Інтернету, що дозволяє їм безперешкодно взаємодіяти із системою, зокрема здійснювати транзакції та отримувати сповіщення в реальному часі. Робота системи залежить від стабільності та доступності інтегрованих зовнішніх сервісів, таких як сервіси електронної пошти, OAuth-провайдерів (Google), сервіси платіжних систем (Stripe, LiqPay тощо), а також сервісів доставки push-сповіщень (Firebase Cloud Messaging) [5]. Будь-які зміни або неполадки в роботі цих сервісів можуть впливати на стабільність роботи системи. Передбачається, що адміністратори системи мають достатні навички для роботи з адміністративною панеллю та оперативного реагування на можливі випадки шахрайства чи некоректного використання платформи. Функціонування системи передбачає дотримання нормативних вимог та стандартів щодо захисту персональних даних, зокрема регламенту GDPR, що потребує регулярного оновлення політики безпеки та періодичної перевірки

відповідності вимогам законодавства.

Очікується, що розробка подальших оновлень і додаткових функцій буде відбуватись відповідно до принципів Clean Architecture, які були обрані під час початкового проектування системи, з метою мінімізації складності майбутньої підтримки та масштабування.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Проектування UML

На основі визначених функціональних і нефункціональних вимог було проведено моделювання сценаріїв взаємодії користувачів з розроблюваною програмною системою. Для візуалізації цих сценаріїв було використано UML-діаграму прецедентів (Use Case Diagram), яка дозволяє чітко відобразити ролі основних акторів системи та типові взаємодії між ними [6].

Створення Use Case Diagram дозволяє структуровано зобразити взаємозв'язок акторів із ключовими функціями системи та слугує основою для подальшого проєктування та реалізації функціоналу програмної системи (див. рис. 3.1).

Рисунок 3.1 – Use-case діаграма (рисунок виконано самостійно)

Ключовими акторами запропонованої системи є:

– зареєстрований користувач – здійснює реєстрацію та авторизацію, управляє своїм профілем, підписується на ініціативи та виконує пожертвування;

– адміністратор – виконує керування ініціативами, категоріями, проводить моніторинг активності користувачів та перевірку зборів;

– незареєстрований користувач (гість) – переглядає доступні ініціативи та збори коштів без можливості здійснювати пожертвування чи підписки;

– зовнішній сервіс – взаємодіє з системою через API для отримання інформації про донорів або ініціативи.

На наведеній Use-case діаграмі представлено основні актори та їх взаємодії з програмною платформою для збору коштів. Актори поділені на чотири категорії: Гість, Зареєстрований користувач, Адміністратор та API-клієнт. Гість має змогу лише переглядати та здійснювати пошук ініціатив без авторизації. Зареєстрований користувач виконує широкий спектр дій: від авторизації, налаштування профілю, підписки на ініціативи, отримання сповіщень до безпосереднього створення власних ініціатив та пожертвувань. Адміністратор системи здійснює модерацію контенту, управління категоріями ініціатив, моніторинг активності користувачів, а також контролює та блокує потенційно шахрайські збори. API-клієнт використовує відкриті інтерфейси для отримання інформації про ініціативи та донорів із зовнішніх застосунків. Диференціація ролей на діаграмі дозволяє чітко визначити функціонал та повноваження кожного типу користувача системи.

3.2 Проєктування архітектури ПЗ

У проекті було використано .NET6(LTS) із EFCore та базою даних PostgreSQL, ці технології становлять основу серверної складової моєї програмної системи.

.NET 6 (LTS) є останньою версією платформи .NET від Microsoft, яка включає довготривалу підтримку (Long Term Support). Це означає, що Microsoft забезпечує підтримку цієї версії протягом тривалого періоду, зазвичай трьох років. .NET 6 є частиною нового об'єднаного підходу до розробки на платформі .NET, що дозволяє

створювати додатки для різноманітних платформ із використанням єдиної бази коду [7].

Вона включає покращення продуктивності, крос-платформенної функціональності та нових функцій для спрощення розробки.

Entity Framework Core (EF Core) є легкою, розширюваною та високопродуктивною версією Entity Framework, яка є об'єктно-реляційним відображувачем (ORM) для .NET. EF Core дозволяє розробникам працювати з базою даних, використовуючи .NET об'єкти, забезпечуючи абстракцію від конкретних команд SQL, які необхідні для взаємодії з даними.

Завдяки цьому, розробники можуть концентруватися на бізнес-логіці додатків, а не на деталях взаємодії з базою даних.

PostgreSQL є високопродуктивною, надійною, та безпечною системою управління реляційними базами даних (RDBMS). Вона широко використовується у корпоративних додатках для зберігання даних, обробки транзакцій та аналітики.

Зв'язок між цими технологіями:

– розробка: розробники використовують .NET 6 для створення додатків, які можуть виконуватися на різних платформах (наприклад, Windows, Linux, macOS). EF Core використовується як шар між додатком .NET і базою даних для управління даними. EF Core дозволяє легко створювати, читати, оновлювати та видаляти дані, використовуючи об'єктні моделі .NET;

– конфігурація: конфігурація з'єднання EF Core з PostgreSQL визначається у файлі конфігурації додатку .NET або програмним кодом. Вказується рядок підключення до PostgreSQL, який містить необхідні параметри для доступу до конкретної бази даних;

– міграція: в EF Core можна визначати моделі даних у коді .NET, які потім мапляться на відповідні таблиці та поля в PostgreSQL. EF Core генерує SQL запити, що виконуються в базі даних, автоматично керуючи створенням, оновленням, видаленням та запитами даних;

– оптимізація та продуктивність: EF Core дозволяє використовувати переваги розширених функцій PostgreSQL, таких як транзакції, індекси, згортання

та розгортання запитів, що оптимізують виконання запитів і підвищують продуктивність застосунків;

– безпека: використання EF Core дозволяє реалізувати засоби безпеки на рівні додатку, такі як перевірка вхідних даних і управління доступом на рівні рядків, що доповнює можливості безпеки, наявні в PostgreSQL, такі як аутентифікація, авторизація та шифрування даних [8].

Більш детальний зв’язок зображено на рисунку 3.2.

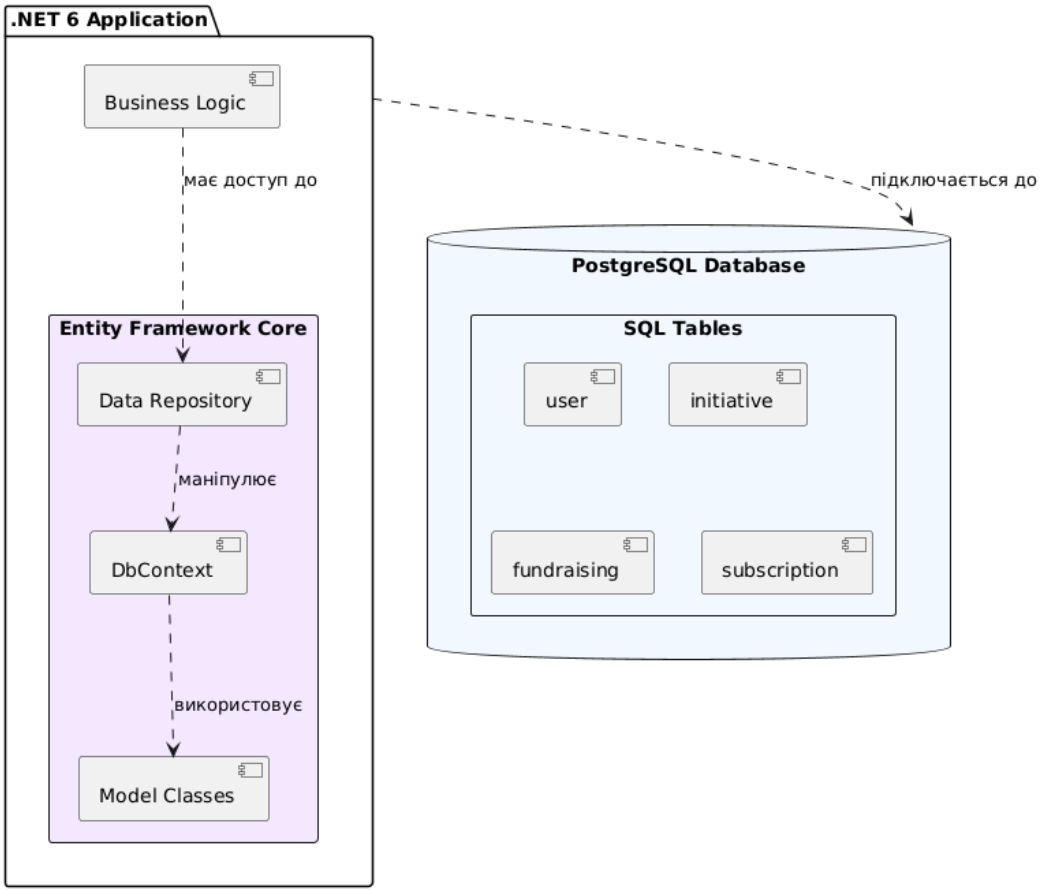

Рисунок 3.2 – Архітектура серверної частини (рисунок виконано самостійно)

З рисунку видно що:

– .net 6 application: загальна оболонка для додатку, що використовує .net 6, включає всі компоненти;

– entity framework core: вбудовано у додаток як частина основної бізнес-логіки та даних, що включає;

– dbcontext: взаємодіє з базою даних, є основним класом, що дозволяє

здійснювати операції з даними;

– model classes: об'єкти, що відображають таблиці бази даних у вигляді класів;

– data repository: абстракція для доступу до даних, що використовує моделі ef core;

– business logic: логіка додатку, яка використовує репозиторій для взаємодії з базою даних;

– postgreSQL database: фізична база даних, де зберігаються дані;

– sql tables: реальні таблиці в postgreSQL, такі як user, initiative, fundraising.

Ця тріада технологій (.NET 6, EF Core та PostgreSQL) разом формують потужну платформу для розробки сучасних, масштабованих та високо надійних корпоративних додатків, забезпечуючи ефективне управління даними, високу продуктивність та сильні засоби безпеки.

Далі більш детально розглянемо архітектуру бек-енд частини, а саме – веб додатку.

У системі використано Clean-architecture архітектуру.

Завдяки чистій архітектурі доменний і прикладний рівні знаходяться в центрі дизайну, відомого як ядро системи. Бізнес-логіка розміщується в цих двох рівнях, хоча вони містять різні види бізнес-логіки. Вони розглядаються як деталі, і бізнес-рівні не повинні залежати від рівнів презентації та інфраструктури. Замість того, щоб бізнес-логіка залежала від доступу до даних або інших питань інфраструктури, ця залежність інвертується: деталі інфраструктури та реалізації залежать від прикладного рівня.

Ця функціональність досягається шляхом визначення абстракцій або інтерфейсів на прикладному рівні, які потім реалізуються типами, визначеними на інфраструктурному рівні. Поширеним способом візуалізації цієї архітектури є використання серії концентричних кіл, подібних до цибулевої архітектури.

Доменний рівень містить корпоративну логіку та типи, а прикладний рівень містить бізнес-логіку та типи. Різниця полягає в тому, що корпоративна логіка може бути спільною для багатьох систем, тоді як бізнес-логіка зазвичай

використовується лише в системі [9].

Ядро не повинно залежати від доступу до даних та інших питань інфраструктури, тому ці залежності інвертуються. Це досягається шляхом додавання інтерфейсів або абстракцій у Core, які реалізуються на рівнях за межами Core (див. рис. 3.3). Прикладний результат імплементації архітектури можна побачити на рисунку 3.4.

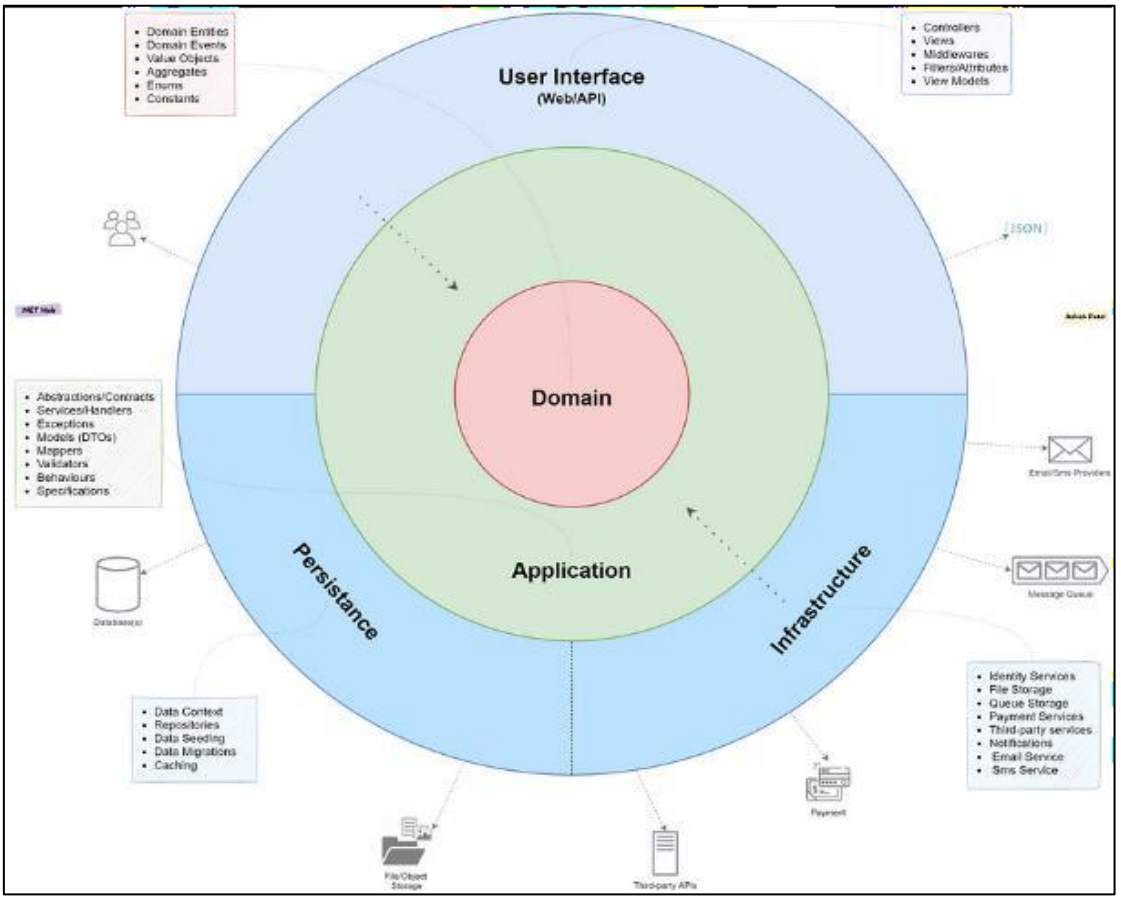

Рисунок 3.3 – Clean-architecture (рисунок виконано самостійно)

3.3 Проектування структури БД

Також важливо розглянути набір сутностей, та їх взаємодію у БД, розглянемо спрощену діаграму цю діаграму наведену на рисунку 3.4.

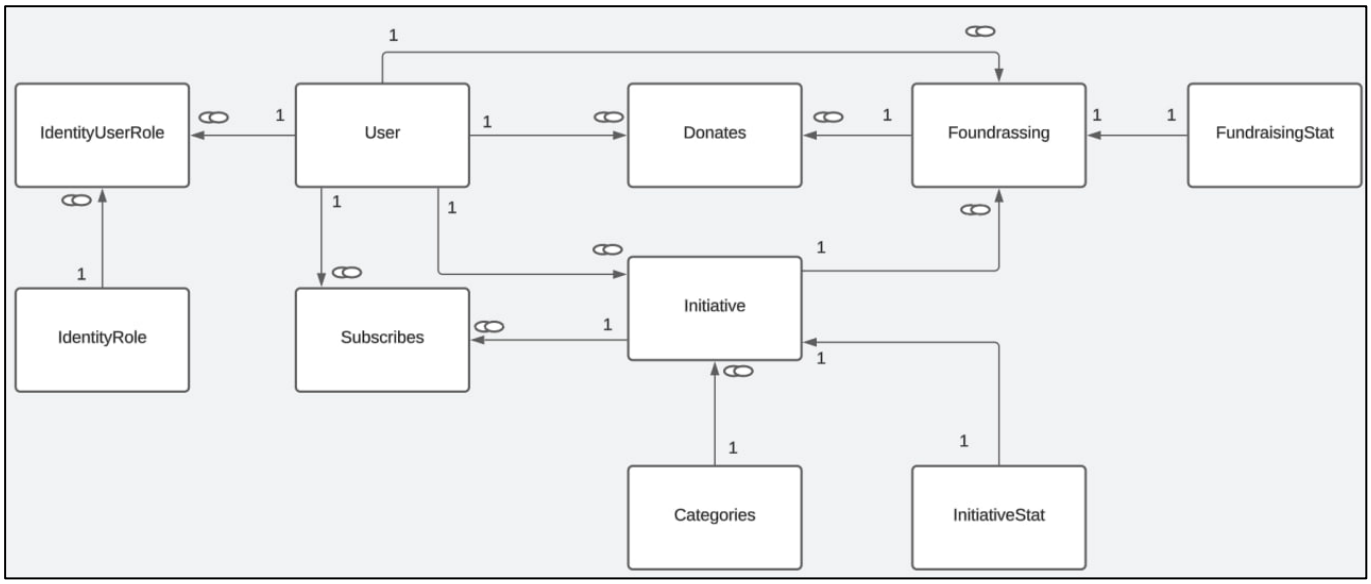

Рисунок 3.4 – Структура бази даних (рисунок виконано самостійно)

Натомість на рисунку 3.5 наведено ER діаграму із деталізацією по полям сутностей.

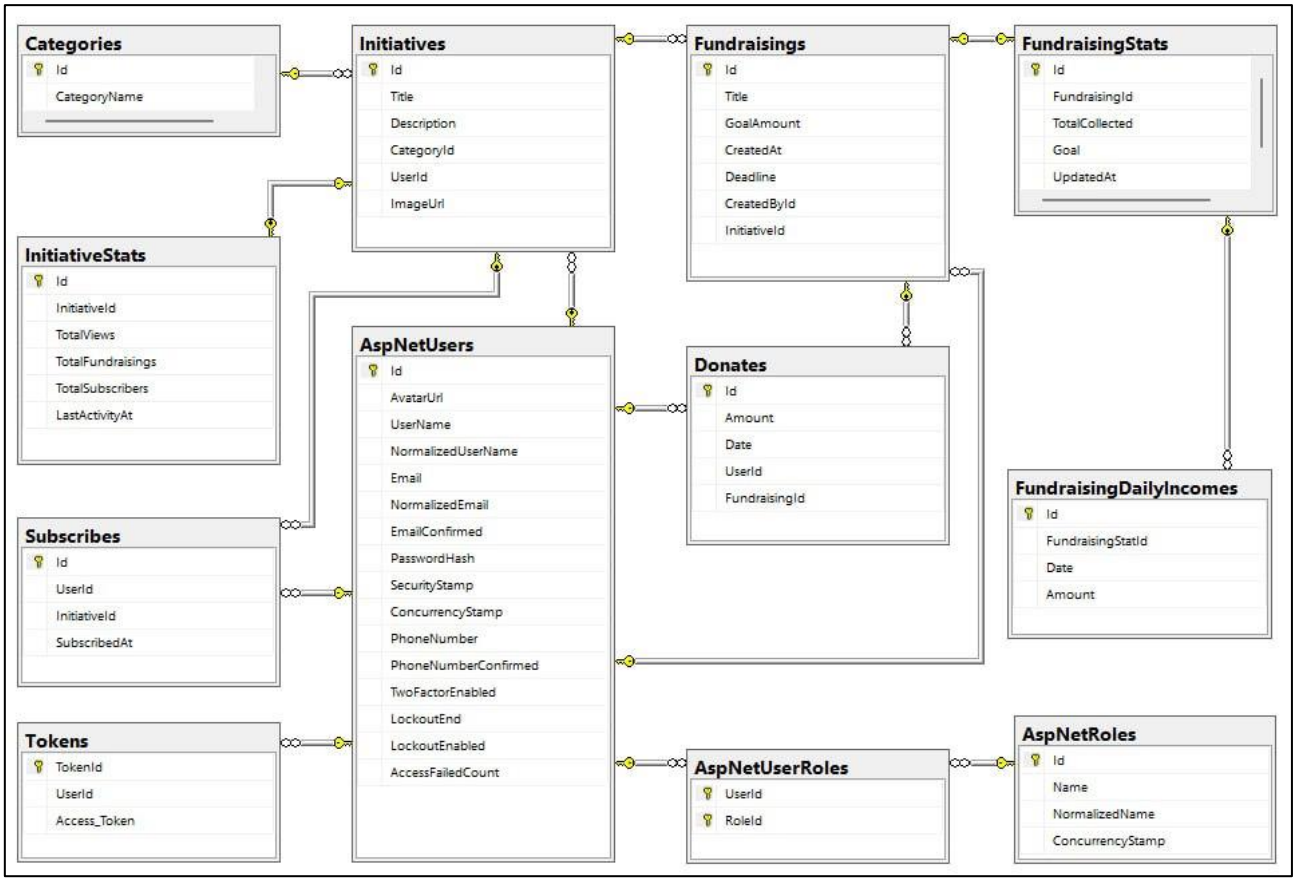

Рисунок 3.5 – ER діаграма (рисунок виконано самостійно)

У даній діаграмі відображено ключові сутності бази даних та їх взаємозв’язки у контексті роботи системи для збору коштів. Центральною сутністю є користувач

(User), який може створювати ініціативи, підписуватись на інші ініціативи, а також робити пожертви у рамках конкретних зборів. Користувач має зв’язок із ролями (IdentityUserRole та IdentityRole), що дозволяє розмежовувати права доступу, наприклад, розділяючи адміністратора та звичайного користувача. Ініціатива (Initiative) безпосередньо пов’язана з користувачем, що її створив, а також належить до певної категорії (Categories), забезпечуючи зручну навігацію та фільтрацію у системі.

Підписки (Subscribes) відображають, хто підписався на оновлення певної ініціативи, що дозволяє автоматично надсилати сповіщення про нові кампанії чи оновлення. Користувачі також можуть здійснювати пожертви, що фіксується у таблиці Donates та прив’язується як до конкретної ініціативи, так і до самого користувача. Кампанія збору коштів (Foundrassing) тісно пов’язана з ініціативою, для якої вона створена, і має власну статистику (FundraisingStat), що дозволяє адміністраторам або ініціаторам збору коштів відслідковувати динаміку пожертвувань та успішність кампанії.

Крім того, для кожної ініціативи ведеться окрема статистика (InitiativeStat), яка включає загальну інформацію про кількість підписок, кількість донатів та інші метрики активності. Така структура даних дозволяє системі працювати ефективно та прозоро: адміністратори можуть контролювати дії користувачів, швидко ідентифікуючи шахрайські кампанії, а користувачі — відслідковувати власну історію участі, брати участь у різних ініціативах та пожертвах, а також підписуватись на цікаві проєкти. Всі сутності взаємопов’язані між собою через чіткі відношення "один-до-багатьох", що гарантує логічну цілісність та дозволяє масштабувати функціонал системи без необхідності порушувати існуючу структуру.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Файлова структура проекту

Файлова структура проекту відображає основні компоненти та логіку побудови архітектури застосунку, дозволяючи зрозуміти, як організовані шари та залежності між ними. Вона чітко демонструє принципи Clean Architecture, у якій ядро системи, що складається з доменної та прикладної логіки, є незалежним від інфраструктурних деталей. Зокрема, окремі проекти (наприклад, Fun.Application, Fun.Domain, Fun.Infrastructure, Fun.Persistance, Fun.Plan.v2) відповідають за конкретні аспекти роботи системи: від бізнес-логіки й абстракцій (домен) до роботи з базами даних і зовнішніми API (інфраструктура).

Така структура забезпечує прозору організацію коду, полегшує розуміння логіки роботи застосунку, спрощує внесення змін і додавання нових функцій без порушення цілісності всього рішення. Крім того, вона дозволяє легко виділити окремі шари відповідальності та зменшує ймовірність виникнення тісних залежностей між компонентами, що сприяє надійності, масштабованості та тестованості системи.

Тож далі варто розглянути файлову структуру проекту та проаналізувати структуру та призначення слоїв архітектурної логіки. Файлову структуру наведено на рисунку 4.1.

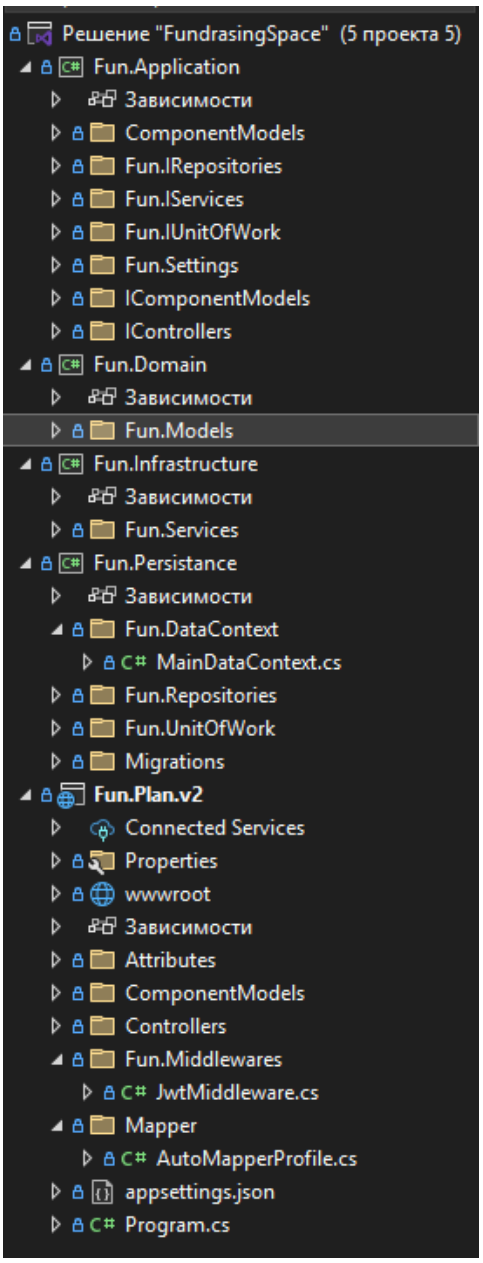

Рисунок 4.1 – Файлова структура (рисунок виконано самостійно)

У системі реалізовано чітку Clean Architecture з поділом на проєкти за ролями: Fun.Application містить бізнес-логіку у вигляді сервісів (DonateService, InitiativeService, TokenService, UserService), інтерфейсів для залежностей (IFun.Services, IRepositories, IUnitOfWork) і DTO (ComponentModels), які служать для передачі даних між шарами; Fun.Domain описує доменні сутності у Fun.Models, що є чистими моделями без залежності від інфраструктури — це основа бізнес-логіки; Fun.Infrastructure реалізує сервіси, що взаємодіють із зовнішніми API або середовищем (наприклад, OAuth, Email, TokenProvider), а також реалізації інтерфейсів, оголошених у Application; Fun.Persistance відповідає за роботу з БД:

тут знаходиться MainDataContext.cs (EF Core DbContext), реалізація UnitOfWork і Repositories, а також міграції; нарешті, Fun.Plan.v2 — це ваш API-шар (Presentation Layer), що містить Controllers (вхідні точки HTTP), Program.cs (точка входу), Middlewares, конфігурацію AutoMapper і роботу з UI (наприклад, wwwroot для SPA); саме тут налаштовуються DI, Authentication, CORS, Swagger, маршрути та запускається веб-сервер. Уся структура підтримує принципи SOLID, забезпечує слабку зв’язаність і тестованість компонентів, а також чітке розділення обов’язків між логікою, зберіганням і представленням.

Далі варто також розглянути код-флоу для системної взаємодії наступного функціоналу – підписка на ініціативи (вибір категорій чи конкретних ініціатив), відправка email та push-сповіщень про нові збори чи статус існуючих, налаштування частоти та типу сповіщень (див. рис. 4.2).

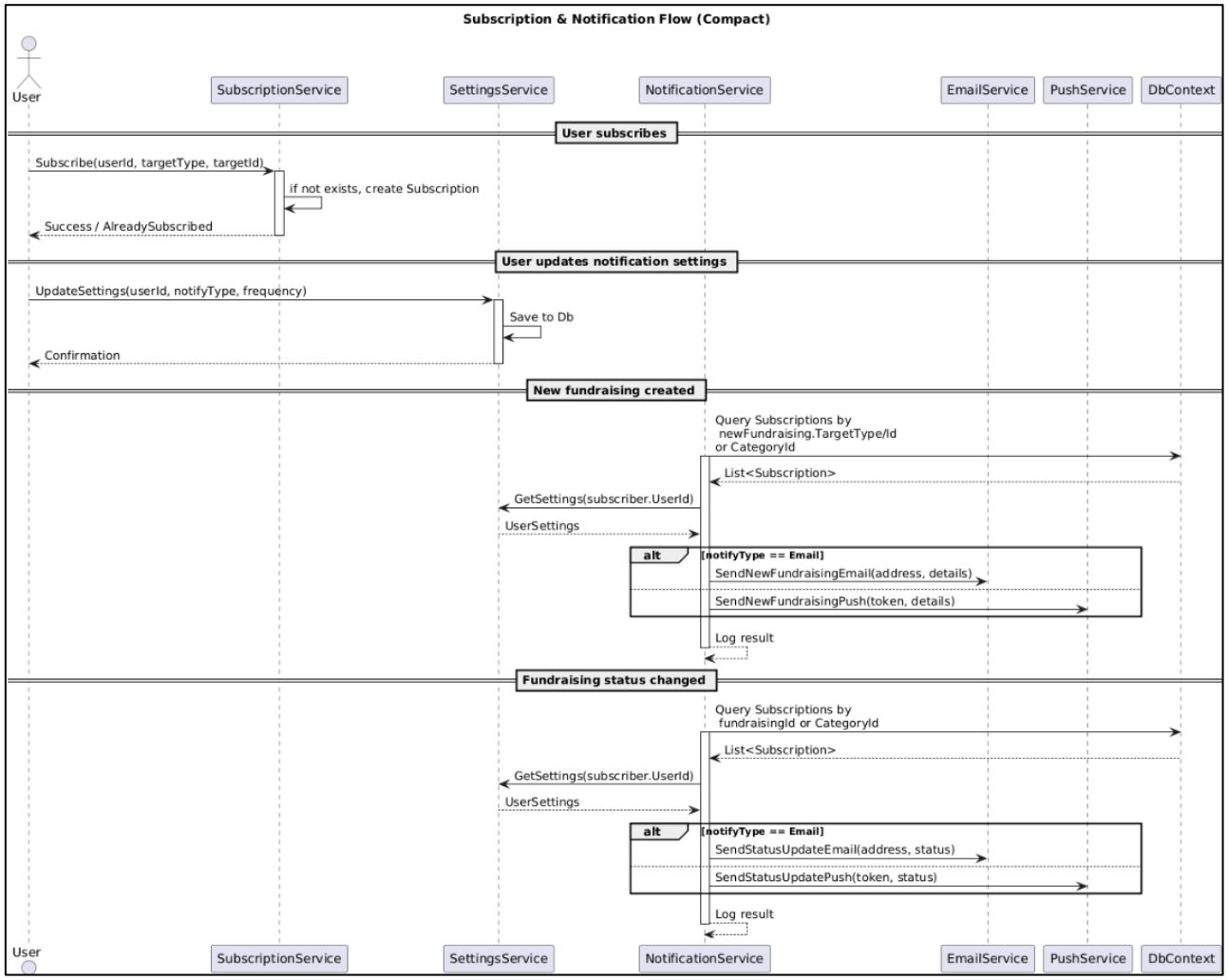

Рисунок 4.2 – Код-флоу підписок (рисунок виконано самостійно)

З діаграми видно що користувач може підписатися на сповіщення про нові збори або зміну статусу існуючих, обравши конкретну ініціативу чи категорію. Запит на підписку надходить у SubscriptionService, який перевіряє наявність аналогічного запису і, якщо підписки ще немає, зберігає її в таблиці Subscriptions (UserId, TargetType, TargetId). Якщо підписка вже існує, нічого не робиться або повертається інформація «вже підписаний». Далі користувач може вказати, як саме та з якою частотою він бажає отримувати сповіщення (Email або Push), і ці налаштування фіксуються у SettingsService у таблиці NotificationSettings (UserId, NotifyType, Frequency).

Коли створюється новий збір або змінюється статус існуючого, запускається NotificationService. Він спочатку отримує з Subscriptions список UserId, які підписані на відповідну InitiativeId або CategoryId, потім для кожного з них завантажує їхні налаштування з NotificationSettings (NotifyType, Frequency). Якщо зрозуміло, що слід надіслати повідомлення миттєво, NotificationService передає дані (адресу/ідентифікатор користувача, тему й текст повідомлення) відповідно до обраного NotifyType в EmailService або PushService. EmailService відправляє лист через поштовий шлюз, PushService — пуш-повідомлення через наприклад Firebase Cloud Messaging.

Щоб уникнути дублювання, NotificationService може запобігати повторним відправкам про ту саму подію (наприклад, перевіряючи, чи вже надсилалося повідомлення про завершення збору). Усе це відбувається асинхронно: підписки й налаштування оновлюються в реальному часі, а NotificationService — через Domain Events або виклики після збереження Fundraising — розподіляє завдання надсилання в бекґраунд (через власний Background Queue чи зовнішню чергу), не блокуючи користувацький запит. Завдяки такому розподілу відповідальності можна незалежно розвивати підсистеми підписок (SubscriptionService), налаштувань (SettingsService) та різних каналів розсилки (EmailService, PushService), не змінюючи базової бізнес-логіки.

4.2 Цікаві алгоритми та методи

Використання OAuth-автентифікації (наприклад через Google) у поєднанні з власною токенізацією всередині системи дає змогу взяти найкраще з обох світів: зовнішнім провайдером делегуємо всю складну і чутливу частину верифікації особи та управління паролями, водночас після підтвердження особи маємо можливість випустити власний JWT із точно підібраними клеймами й ролями, які потрібні бізнес-логіці.

Це дозволяє не зберігати й не обробляти паролі користувачів у базі, знижуючи ризики витоку, і водночас зберегти повний контроль над тим, хто, коли і з якими правами заходить до API: самі визначаємо термін життя токена, його алгоритм підпису, набір клейм (userId, email, roles, permissions тощо) і можемо у будь-який момент відкликати чи заблокувати токен через зміну ключів чи використання чорного списку [10].

OAuth забезпечує зручність користувача (одноразовий вхід через існуючий обліковий запис, без потреби реєструвати новий пароль), стандартизацію (використання OpenID Connect, вбудовані механізми захисту через state і PKCE) та можливість швидко підключати інші провайдери соцмереж чи корпоративних каталогів, а власна токенізація – гарантує, що всі внутрішні мікросервіси або фронтенд-додатки отримують єдиний формат авторизації (Bearer JWT), легко перевіряють підпис і зчитують необхідні дані з клеймів без звернення до бази чи сторонніх серверів. Такий підхід оптимізує як безпеку, так і продуктивність: зовнішній провайдер гарантує автентичність, а, випустивши свій JWT, гарантується цілісність й авторизацію доступу до ресурсів згідно з встановленими правилами (див. рис. 4.3).

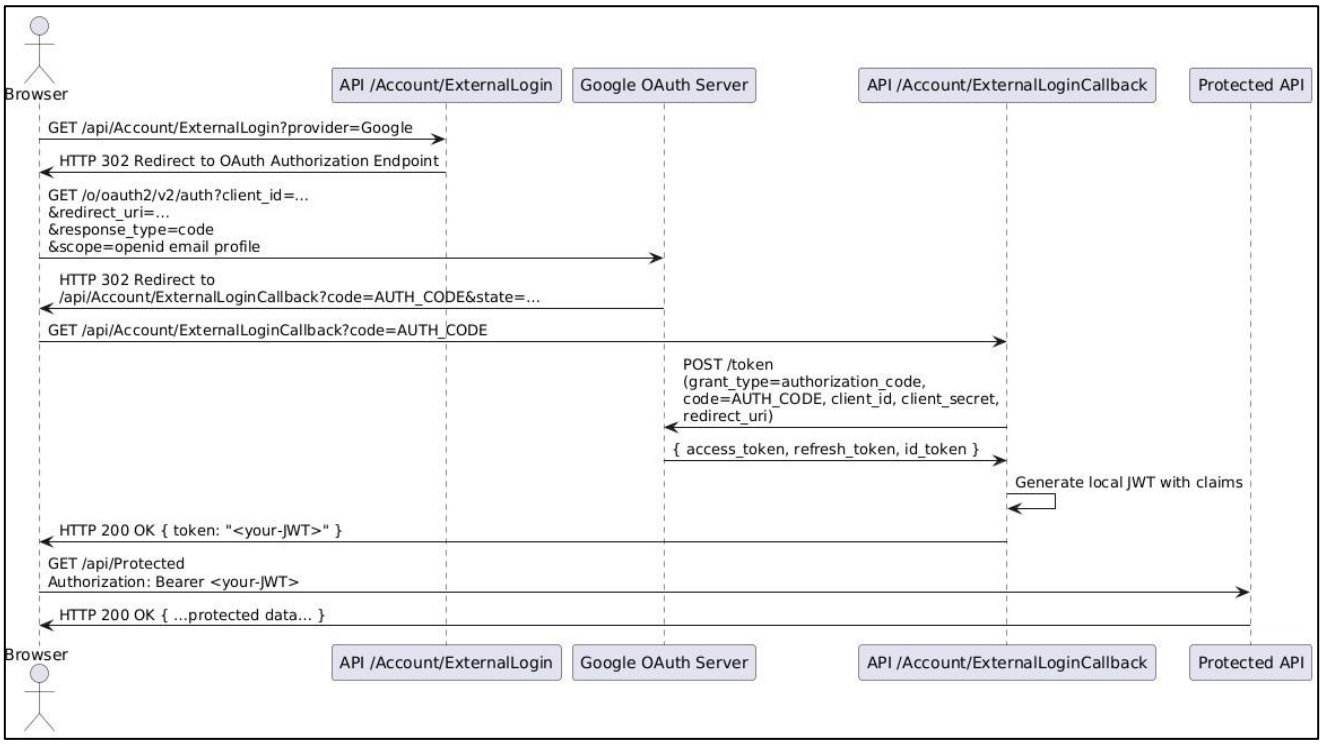

Рисунок 4.3 – OAuth + JWT pipline (рисунок виконано самостійно)

Надалі надано код сервісу юзера по обробці потоку OAuth:

```
public AuthenticationProperties GetAuthenticationProperties(string returnUrl)
{
var properties =
_signInManager.ConfigureExternalAuthenticationProperties(
GoogleDefaults.AuthenticationScheme,

$"{_redirectUri}?returnUrl={Uri.EscapeDataString(returnUrl)}"); return properties;
}public async Task<User> HandleCallbackAsync()
{
var info = await _signInManager.GetExternalLoginInfoAsync(); if (info == null)
throw new InvalidOperationException("Cannot load external login information.");
var result = await _signInManager.ExternalLoginSignInAsync( info.LoginProvider, info.ProviderKey, isPersistent: false);
User user;
if (result.Succeeded)
{
user = await _userManager.FindByLoginAsync( info.LoginProvider, info.ProviderKey);
}
else
{
var email = info.Principal.FindFirstValue(ClaimTypes.Email); user = new User { UserName = email, Email = email };
await _userManager.CreateAsync(user);
await _userManager.AddLoginAsync(user, info);
}
```

```
return user;
}
```

Тут метод GetAuthenticationProperties формує об’єкт

AuthenticationProperties, який потрібен для запуску OAuth-автентифікації через Google; тут викликається ConfigureExternalAuthenticationProperties, де вказується схема (GoogleDefaults.AuthenticationScheme) і повний callback-URL, який включає в себе параметр returnUrl, що зберігає шлях, куди фронтенд має повернути користувача після входу (наприклад, /dashboard). Цей

AuthenticationProperties потім передається в Challenge(...), щоб middleware міг зберегти стан і знати, куди повернутись. Метод HandleCallbackAsync обробляє зворотній виклик від Google після авторизації: він викликає

GetExternalLoginInfoAsync, який повертає інформацію про зовнішній обліковий запис (LoginProvider. ProviderKey, ClaimsPrincipal); якщо така зв’язка вже існує в базі (через ExternalLoginSignInAsync), система просто повертає пов’язаного користувача (FindByLoginAsync); якщо ні — витягується email із claims, створюється новий користувач, і виконується AddLoginAsync, що додає запис у таблицю AspNetUserLogins, зв’язуючи нового користувача з зовнішнім обліковим записом (Google). Обидва методи разом забезпечують безпечну, автоматичну авторизацію через Google із підтримкою повернення на фронтенд і зберіганням зовнішніх логінів.

4.2.1 Архітектура адміністративного функціоналу системи

У нас адміністративна частина побудована на базі стандартного ASP NET Core Identity та ролей, поширених через JWT-токени. Коли користувач реєструється, його профіль зберігається в таблиці AspNetUsers разом із записом у багатозначній зв’язці IdentityUserRole–IdentityRole, де йому одразу може бути призначена роль “User” або “Admin”. Далі, під час входу в систему, генеруємо JWT-токен, в якому крім email і userId додаємо всі ролі користувача як окремі claim типу “role”. У налаштуваннях сервісів (Program.cs) під’єднуємо middleware для аутентифікації через JWT (AddAuthentication().AddJwtBearer(...)) і викликаємо

AddAuthorization, щоб ASP NET Core перевіряв ці ролі на вході в кожен контролер або ендпоїнт [11].

У контролерах адміністративні методи (додавання/видалення категорій, модерація ініціатив, робота зі статистикою тощо) відмічені атрибутом [Authorize(Roles = "Admin")] і механізм автоматично відкидає будь-які запити від не-адмінів із 403 Forbidden. Якщо ж точково потрібні більш тонкі права, можна описати власні політики через services.AddAuthorization(options => options.AddPolicy("CanManageFundraising", …)) і застосувати їх через [Authorize(Policy = "CanManageFundraising")].

Весь обмін даними між фронтендом і бекендом відбувається через захищені JWT-заголовки, тому ролі ніколи не передаються у відкритому вигляді. Сам механізм авторизації спирається на стандартні класи UserManager та RoleManager: при створенні або оновленні користувача викликаємо userManager.AddToRoleAsync(user, "Admin") чи .RemoveFromRoleAsync, а далі при кожному запиті middleware JwtBearer перевіряє валідність токена, відновлює об’єкт ClaimsPrincipal з його вмістом і дивиться, чи є в ньому claim “role”:“Admin”. Якщо є – запит пускається, якщо ні –повертається “403 Forbidden”.

Таким чином, адміністративна зона мого API виділена чітким розподілом відповідальності: Identity відповідає за зберігання й перевірку ролей, генерація токенів вкладає ролі в claims, а атрибути [Authorize(Roles=…)] у контролерах забезпечують захищений доступ тільки для користувачів-адмінів. Це дає нам гнучкість керувати привілеями без написання власних складних перевірок у бізнес-логіці. Нижче розберемо метод автентифікації користувача:

```
    public async Task<IActionResult>
AuthenticateUser(IUserAuthenticateModel authenticateUser)
    {
        var user = _mapper.Map<User>(authenticateUser);
        var joinUser = await _userManager.FindByEmailAsync(user.Email);
        if (joinUser == null)
        {
            return new BadRequestObjectResult(new { Message = "User not
found" });
        }
        var userauth = await
_signInManager.CheckPasswordSignInAsync(joinUser,
authenticateUser.Password,
```

```
lockoutOnFailure: false);
          if (userauth.Succeeded)
          {
              var token = _jwtService.GenerateToken(authenticateUser);
              var isAdmin = authenticateUser.Email;
              string supportadmin = "alexsemenov@gmaill.com";
              bool IsSupportAdmin = string.Equals(isAdmin, supportadmin,
StringComparison.OrdinalIgnoreCase);;
              return new OkObjectResult(new
              {
                  AuthToken = token,
                  IsAdminSupport = IsSupportAdmin
              }
              );
          }
          return new BadRequestObjectResult(new { Message = "Wrong password"
});
      }
```

Цей метод AuthenticateUser обробляє запит на вхід користувача та повертає JWT-токен разом з позначкою, чи є він «саппорт-адміном». По черзі він виконує такі кроки:

– за допомогою automapper перетворює передану dto-модель iuserauthenticatemodel у внутрішню сутність user, щоб дістати з неї email;

– через usermanager шукає в базі користувача з таким email; якщо його немає, негайно повертає 400 bad request із повідомленням «user not found»;

– якщо ж користувач знайдений, викликає signinmanager.checkpasswordsigninasync, щоб перевірити правильність пароля (без блокування після помилок);

– у разі успішної аутентифікації генерує jwt-токен (jwtservice.generatetoken) на основі вхідних даних;

– оскільки в цій системі є «саппорт-адмін» з фіксованою поштою, перевіряє, чи співпадає email поточного користувача з цим значенням (alexsemenov@gmail.com), і встановлює булеву змінну issupportadmin;

– віддає клієнту 200 ok із json-об’єктом, що містить згенерований токен та прапорець isadminsupport;

– якщо перевірка пароля не пройшла, повертає 400 bad request із повідомленням «wrong password».

Завдяки цій логіці гарантується, що лише існуючі користувачі з коректними обліковими даними отримають токен доступу, а у відповідь клієнту надходить як сам токен, так і інформація про те, чи володіє цей користувач розширеними правами саппорт-адміна [12].

Далі наведено діаграму в якій зображено функціональні елементи до яких має доступ тільки адміністратор системи (див. рис. 4.3).

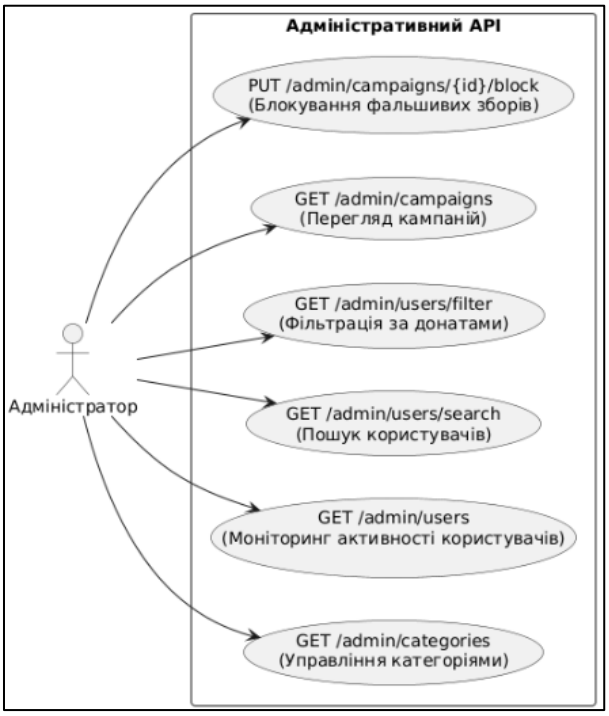

Рисунок 4.3 – Функціональні елементи до яких має доступ тільки адміністратор (рисунок виконано самостійно)

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

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

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

Інформація про тест-кейс			
Ідентифікатор тесту:		Тест-кейс №1	
Власник тесту:		Бойченко Матвій Юрійович	
Дата створення:		23.05.2025	
Мета тесту:		перевірити, що життєвий цикл акаунта (реєстрація, верифікація, OAuth-логін, донати) й модуль сповіщень (підписки, push/email, частота) працюють коректно та що адмінський інтерфейс безпечно обробляє фільтри й масові операції.	
Реєстрація, email-верифікація, Google OAuth і профіль			
№	Опис випадку	Очікуваний результат	Висновок
1	POST /auth/register {email, pwd}	201 Created, статус PendingVerify, у Redis з’явився лист-шаблон	Пройдено
2	Перейти за /auth/verify?token=…	200, статус → Verified, refresh+access JWT	Пройдено
3	POST /auth/login з валідними cred	200, повернено пари токенів, claim role=User	Пройдено
4	GET /profile із JWT	200, donations=[], subscriptions=[]	Пройдено
5	PATCH /profile змінити avatar URL & bio	200, updatedAt оновився, avatar URL пройшов валідацію MIME	Пройдено
6	GET /auth/google/url → redirect, OAuth consent, callback /auth/google/callback	302 на returnUrl, видано JWT з claim LoginProvider=Google	Пройдено
7	POST /auth/login з неправильним pwd (5 разів)	Після 5-ї спроби – 423 Locked 10 хв, audit-entry LoginLock	Пройдено

Кінець таблиці 5.1.

8	Спроба POST /donations без email-верифікації (новий користувач)	403 EmailNotVerified	Пройдено
9	DELETE /auth/logout → GET /profile	401 Unauthorized	Пройдено
10	DELETE /auth/user із JWT Google-логіна	204, акаунт помічено IsSoftDeleted=true, refresh-токени відкликано	Пройдено
Підписки, сповіщення та адмін-дашборд донорів			
№	Опис випадку	Очікуваний результат	Висновок
1	POST /subscriptions {type=Category, id=medical}	204, запис у Subscriptions	Пройдено
2	PATCH /settings/notifications {email=daily, push=instant}	200, налаштування збережено	Пройдено
3	Створюється новий збір категорії medical	У очереді PushTasks — завдання → FCM надсилає push; SMTP — лист у digest-чергу	Пройдено
4	Відкрити push, перейти за deep-link	Мобільний клієнт відкриває деталі ініціативи ID	Пройдено
5	POST /donations {amount=75} → Stripe webhook succeeded	У Donations новий запис, totalSum(user) +75	Пройдено
6	GET /admin/donors?sort=totalSum&dir=desc&take=5 (JWT Admin)	JSON ТОП-5; перший totalSum ≥ 75	Пройдено
7	GET /admin/donors?minCount=0&maxCount=0	Порожній масив (фільтр «без донатів»)	Пройдено
8	PATCH /admin/donors/{id}/tag {tag=VIP}	200, тег додано; повторний PATCH → 409 TagExists	Пройдено
9	POST /admin/broadcast/email {target=VIP, template=thanks}	Створено batch-job в Redis; у SMTP-інбоксі > 0 листів	Пройдено
10	Запит /admin/donors з токеном звичайного User	403 Forbidden	Пройдено

ВИСНОВОК

У результаті проходження передатестаційної практики було спроєктовано програмну систему для організації збору коштів, яка передбачає повноцінний набір функцій для взаємодії з користувачами, донорами, адміністраторами та зовнішніми сервісами. Основну увагу під час роботи було приділено проєктуванню модулів реєстрації та авторизації користувачів (включаючи можливість інтеграції з Google OAuth 2.0), структури ведення історії донатів, формування профілів, а також логіки системи сповіщень — як email, так і push-повідомлень. Окремо опрацьовано концепцію адміністративного інтерфейсу для модерації ініціатив, перевірки зборів і управління категоріями.

Архітектура майбутнього застосунку базується на підході Clean Architecture з чітким розділенням на доменну, прикладну, інфраструктурну та презентаційну частини. Такий підхід має забезпечити високу підтримуваність коду, модульність і гнучкість при подальшому масштабуванні. У межах проєктування передбачено використання сучасного технологічного стеку: .NET 8, ASP.NET Core, EF Core, PostgreSQL, Firebase Cloud Messaging, а також Stripe або LiqPay для реалізації онлайн-платежів.

У процесі роботи було проаналізовано ключові проблеми цифрових платформ фандрейзингу: низьку прозорість, відсутність гнучкої системи верифікації користувачів, складну інтеграцію з платіжними інструментами та брак ефективної комунікації зі сторони сервісу. Запропонована структура проєкту покликана усунути ці недоліки, забезпечити масштабовану взаємодію з великою кількістю донорів і створити зручну систему адміністрування ініціатив.

Проєктування демонструє відповідність майбутнього рішення сучасним вимогам до безпечних, гнучких і прозорих систем цифрового фандрейзингу. Створена архітектурна основа є повністю готовою до подальшої розробки та практичного впровадження у рамках реальних соціальних або благодійних ініціатив.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Мазаракі А. А., Волосович С. І. Краудфандинг благодійності в умовах протидії збройній агресії [Електронний ресурс] // Scientia fructuosa. – 2023. – Вип. 1. – Режим доступу: https://doi.org/10.31617/1.2023(147)01, вільний. – Дата звернення: 03.06.2025.

2. Apple Inc. Donate24 [Електронний ресурс] (офіційна сторінка App Store) [Зображення]. — Режим доступу: https://apps.apple.com/us/app/donate24-%D0%B7%D0%B1%D0%BE%D1%80%D0%B8-%D0%BF%D1%96%D0%B4%D1%82%D1%80%D0%B8%D0%BC%D0%BA%D0%B0/id6450211519 — Дата звернення: 03.06.2025.

3. Конверти в Приват24 [Електронний ресурс] (офіційна сторінка Private24) [Зображення]. — Режим доступу: https://privatbank.ua/envelopes-p24 — Дата звернення: 03.06.2025.

4. Google. Using OAuth 2.0 to Access Google APIs [Електронний ресурс]. – Режим доступу: https://developers.google.com/identity/protocols/oauth2 – Дата звернення: 03.06.2025.

5. Google. Firebase Cloud Messaging [Електронний ресурс]. – Режим доступу: https://firebase.google.com/docs/cloud-messaging – Дата звернення: 03.06.2025.

6. Booch, G., Rumbaugh, J., & Jacobson, I. The Unified Modeling Language User Guide (2nd Edition) [Електронний ресурс]. – Addison-Wesley Professional, 2005. – 512 с. – ISBN 0321267974. – Режим доступу: https://www.oreilly.com/library/view/the-unified-modeling/0321267974/ – Дата звернення: 03.06.2025.

7. Freeman, A., & Sanderson, S. Pro Entity Framework Core 6: Modern Cross-Platform Development (2nd Edition) [Електронний ресурс]. – Apress, 2022. – 700 с. – ISBN 978-1-4842-8649-7. – Режим доступу: https://www.apress.com/gp/book/9781484286497 – Дата звернення: 03.06.2025.

8. Michael's Coding Spot. PostgreSQL in C# .NET with Npgsql, Dapper, and Entity Framework: The Complete Guide [Електронний ресурс]. – Режим доступу: https://michaelscodingspot.com/postgres-in-csharp/ – Дата звернення: 03.06.2025.

9. Tytenko S., Polienova V., Fedenko V. Чиста архітектура: повний посібник із розробки програмного забезпечення [Електронний ресурс]. – Режим доступу: https://er.auk.edu.ua/handle/234907866/73 – Дата звернення: 03.06.2025

10. Richer, J., & Sanso, A. OAuth 2 in Action [Електронний ресурс]. – Manning Publications, 2017. – 360 с. – ISBN 9781617293276. – Режим доступу: https://www.manning.com/books/oauth-2-in-action – Дата звернення: 03.06.2025.

11. Freeman, A. Pro ASP.NET Core Identity: Under the Hood with Authentication and Authorization in ASP.NET Core 5 and 6 Applications [Електронний ресурс]. – Apress, 2021. – 725 с. – ISBN 978-1-4842-6857-5. – Режим доступу: https://www.vitalsource.com/products/pro-asp-net-core-identity-adam-freeman-v9781484268582 – Дата звернення: 03.06.2025.

12. Jaskula, T. Securing ASP.NET Core with JWT [Електронний ресурс]. – Packt Publishing, 2020. – 250 с. – ISBN 978-1-78917-648-7. – Режим доступу: https://www.packtpub.com/product/securing-asp-net-core-with-jwt/9781789176487 – Дата звернення: 03.06.2025.

13. Github репозиторій проекту [Електронний ресурс]. — Режим доступу: https://github.com/NureBoichenkoMatvii/2025_B_PI_PZPI-21-1_Boichenko_M_Y.