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 підтримки волонтерської діяльності.
Мобільний застосунок
(тема)

Виконав:
здобувач 4 року навчання
групи ПЗПІ-21-8

Станіслав ГОЛЯ
(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 проф. кафедри ПІ Володимир БОНДАРЄВ
(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 Голі Станіславу Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система для підтримки волонтерської діяльності. Мобільний застосунок

Затверджена наказом по університету № 397Ст від 19.05.2025

2. Термін подання студентом роботи до екзаменаційної комісії 12.06.2025

3. Вихідні дані до роботи Розробити мобільний застосунок для підтримки волонтерської діяльності.

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, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	12.04.2025	*виконано*
2	Створення специфікації ПЗ	18.04.2025	*виконано*
3	Проектування ПЗ	24.04.2025	*виконано*
4	Розробка ПЗ	12.05.2025	*виконано*
5	Тестування ПЗ	26.05.2025	*виконано*
6	Оформлення пояснювальної записки	27.05.2025	*виконано*
7	Підготовка презентації та доповіді	28.05.2025	*виконано*
8	Попередній захист	02.06.2025	*виконано*
9	Нормоконтроль, рецензування	05.06.2025	*виконано*
10	Здача роботи у електронний архів	06.06.2025	*виконано*
11	Допуск до захисту у зав. кафедри	09.06.2025	*виконано*

Дата видачі завдання «09» «квітня» 2025р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. кафедри ПІ Володимир БОНДАРЄВ
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра, 59 стор., 24 рис., 1 табл., 12 джерел, 2 додатки.

БЕКЕНД, ВОЛОНТЕРСЬКА ДОПОМОГА, ВИМОГА, ЗБІР, ІНТЕГРАЦІЯ, МОБІЛЬНИЙ ДОДАТОК, МОНОБАНК, FLUTTER

Об’єкт розробки – клієнтська частина програмної системи у вигляді мобільного додатку, призначеного для організації волонтерської допомоги. Цей додаток дозволяє користувачам переглядати та створювати збори коштів і матеріальних ресурсів, керувати запитами на допомогу від отримувачів, працювати з профілями волонтера та отримувача, а також редагувати інформацію. Додаток взаємодіє із серверною частиною системи для обміну даними.

Мета розробки – створити зручний мобільний застосунок, який допоможе волонтерам ефективно координувати надання допомоги, керувати кампаніями зі збору коштів, оперативно реагувати на потреби людей та взаємодіяти з іншими учасниками волонтерського процесу.

Метод рішення – розробка мобільного застосунку з використанням технології Flutter, що забезпечує роботу на різних мобільних платформах (Android, iOS). Додаток отримує та відправляє дані на сервер через спеціально розроблений інтерфейс. Реалізовано функції входу користувачів, відображення списків зборів та потреб, можливість їх створення, а також підключення до сервісу Монобанку для показу прогресу зборів коштів.

Результат розробки – створено зручний та функціональний мобільний застосунок. Він надає волонтерам швидкий доступ до актуальної інформації щодо зборів коштів та матеріальних потреб, дозволяє ефективно керувати власним профілем та діями, сприяючи більш прозорій та дієвій організації волонтерської допомоги.

ABSTRACT

BACK-END, VOLUNTEER AID SYSTEM, REQUIREMENT, FUND, INTEGRATION, MOBILE APP, MONOBANK, FLUTTER

Object of development – the client side (mobile app) of a software system designed to automate volunteer aid processes. It includes displaying and creating fundraisers, recipient requests, volunteer and recipient profiles, the ability to edit information, and integration with the back-end for data exchange.

Goal of development – to create a convenient mobile application that enables volunteers to efficiently coordinate aid, manage fundraising campaigns, respond to needs, and interact with other participants in the process.

Solution method – using Flutter as a cross-platform framework to build a mobile interface that communicates with the back-end system via a defined interface. The application provides user authentication, displays and allows creation of fundraisers and requirements, and integrates with the Monobank service to show fundraising progress.

Result of development – a convenient and functional mobile application was created. It allows volunteers to quickly access up-to-date information about fundraisers and needs, manage their profile and activities effectively, and supports a more transparent and efficient organization of aid.

ЗМІСТ

ВСТУП

Сучасний світ характеризується динамічними змінами та складними викликами, які вимагають активної громадянської позиції та консолідації суспільних зусиль. В цьому контексті особливого значення набуває волонтерський рух та благодійна діяльність, що стали невід'ємною частиною реагування на кризові ситуації та підтримки вразливих груп населення. Однак, ефективність такої діяльності безпосередньо залежить від здатності швидко координувати дії, раціонально використовувати наявні ресурси, підтримувати прозорість процесів та забезпечувати надійну комунікацію між усіма залученими сторонами – волонтерами, координаторами, донорами та отримувачами допомоги.

Незважаючи на значні масштаби волонтерської активності, особливо в Україні, традиційні інструменти координації, такі як соціальні мережі, месенджери та електронні таблиці, часто виявляються недостатньо ефективними. Вони призводять до фрагментації інформації, ускладнюють відстеження актуального статусу потреб і зборів, не дозволяють належним чином пріоритезувати запити та ускладнюють процес звітності. Ці фактори можуть знижувати загальну продуктивність волонтерської роботи та рівень довіри до неї. Тому постає актуальна задача розробки спеціалізованих програмних рішень, які б могли автоматизувати рутинні процеси, централізувати інформаційні потоки та надати зручні інструменти для всіх учасників. Оскільки значна частина волонтерської діяльності відбувається "в полі", особливої ваги набуває забезпечення мобільного доступу до таких систем.

Метою даної дипломної роботи є дослідження та розробка концепції та програмної реалізації клієнтської частини інформаційної системи волонтерської допомоги у вигляді мобільного застосунку. Передбачається, що такий застосунок стане зручним та функціональним інструментом, перш за все, для волонтерів та отримувачів допомоги. Він має надати можливість ефективно управляти процесом створення та перегляду цільових зборів коштів і ресурсів, формувати та відстежувати запити на допомогу, керувати власними профілями та взаємодіяти з

іншими користувачами через централізовану систему. Ключовим аспектом є забезпечення інтуїтивно зрозумілого інтерфейсу та надійного доступу до актуальної інформації в будь-який час і з будь-якого місця.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

У період повномасштабної війни, соціально-економічної нестабільності та гуманітарних викликів в Україні, волонтерський рух відіграє надзвичайно важливу роль у забезпеченні базових потреб цивільного населення, медичних закладів, військових та постраждалих громад. Волонтерство стало не лише формою громадської активності, а й ключовим механізмом оперативного реагування на кризові ситуації. В умовах високої динаміки запитів на допомогу та обмежених ресурсів, критично важливо забезпечити ефективну організацію зборів, логістику передачі допомоги та прозорість її розподілу. Однак, традиційна модель волонтерської координації, що базується на чатах, таблицях та окремих формах Google, виявляється малоефективною при масштабуванні ініціатив, призводячи до дублювання запитів, втрати актуальності інформації, складності у звітності та зниження довіри до організаторів.

У відповідь на ці виклики активно розвиваються цифрові платформи, покликані автоматизувати процеси координації допомоги. Вони мають потенціал дозволити формувати структуровані вимоги на допомогу, створювати цільові збори, обліковувати отримувачів та формувати звіти. Водночас, по-справжньому ефективна система повинна не лише реєструвати дані, а й забезпечувати активну взаємодію між усіма учасниками – волонтерами, донорами, отримувачами – підтримувати гнучку пріоритезацію запитів, контроль за термінами виконання та надавати інтуїтивно зрозумілі інструменти для візуалізації процесів. Важливим аспектом є можливість інтеграції таких платформ з зовнішніми сервісами через API: мобільними застосунками, сайтами для публічного представлення, платіжними системами (як Monobank jars) для зручних пожертв та відстеження, аналітичними інструментами та хмарними сховищами для документації.

Одним із найвідоміших та масштабних прикладів цифрових рішень у сфері благодійності в Україні є платформа dobro.ua Української Біржі Благодійності [1]. Вона зарекомендувала себе як надійний інструмент для проведення цільових зборів

коштів на реалізацію проєктів, ініційованих перевіреними благодійними фондами та громадськими організаціями. Платформа забезпечує високий рівень прозорості саме фінансових потоків: кожен проєкт має детальний опис, обґрунтування потреби та чітко визначену цільову суму. Процес пожертви є максимально спрощеним – користувач може здійснити платіж з банківської картки, а інформація про надходження коштів оперативно відображається на сторінці проєкту (див. рис. 1.1). Кошти акумулюються на рахунках біржі, що гарантує їх збереження та цільове використання. Важливим елементом довіри є ретельна верифікація фондів-партнерів та детальна фінансова звітність по завершенню проєктів, яка надається самими фондами.

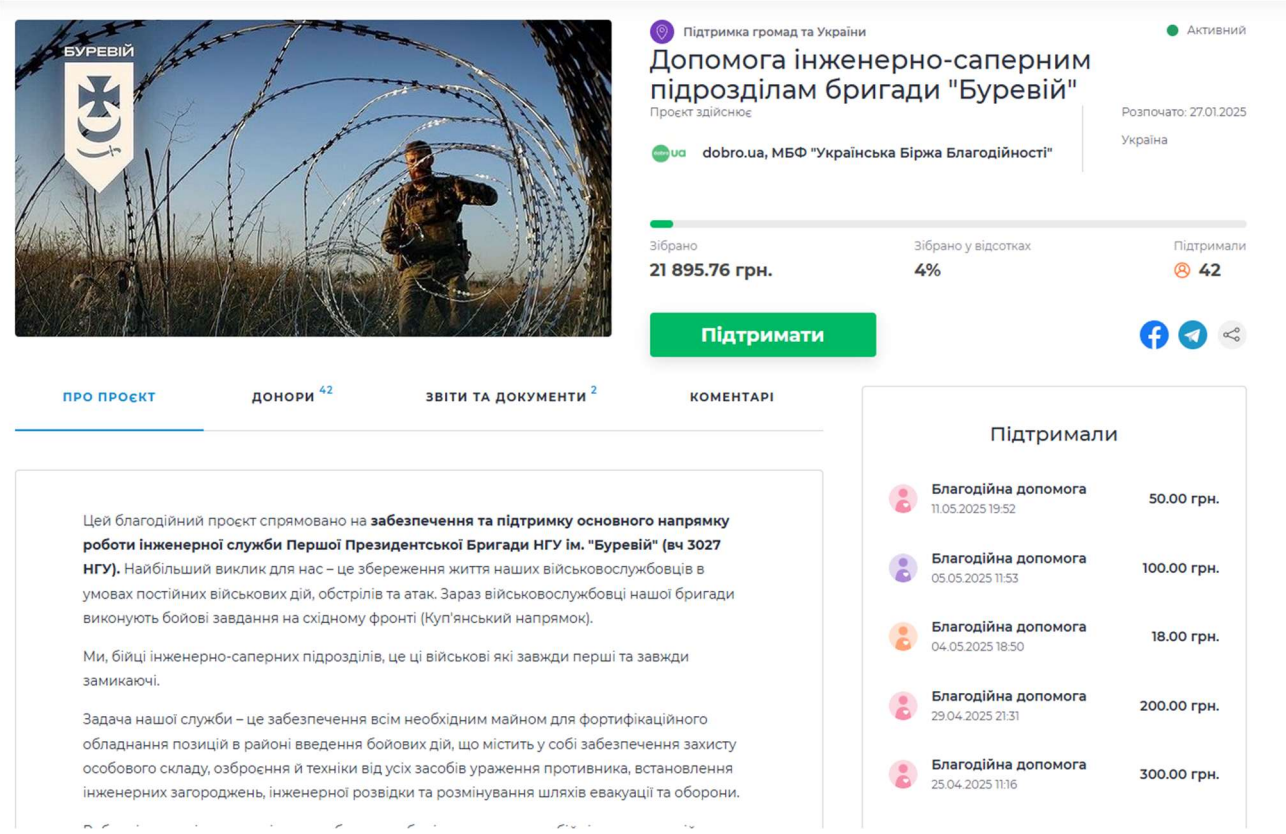

Рисунок 1.1 – Платформа благодійної підтримки Dobro.ua [1]

Однак, при детальному аналізі функціоналу dobro.ua з точки зору потреб операційної волонтерської діяльності, що передбачає управління не лише

фінансами, а й матеріальними ресурсами, логістикою та координацією волонтерів, виявляються певні обмеження:

- основний фокус платформи – збір коштів. Можливості для детального опису та управління потребами (конкретні товари, медикаменти, обладнання із зазначенням точної кількості, характеристик, категорій) є обмеженими. Потреби часто описуються узагальнено в текстовому описі проєкту, що ускладнює автоматизовану обробку, пошук конкретних позицій та контроль їх закриття;
- платформа є переважно донор-орієнтованою. Механізми для реєстрації волонтерів, які безпосередньо займаються пошуком, закупівлею, транспортуванням чи розподілом допомоги, відсутні. Немає інструментів для призначення завдань волонтерам, відстеження їхньої доступності чи координації логістичних ланцюжків;
- хоча фінансова звітність фондів є сильною стороною платформи, відсутня система звітування саме про операційну діяльність: хто, коли, кому і яку конкретно матеріальну допомогу доставив. Це важливо для відстеження всього циклу допомоги та підвищення довіри щодо розподілу ресурсів;
- платформа не надає інструментів для пріоритезації запитів, встановлення та контролю дедлайнів для закриття конкретних потреб, управління статусами окремих предметів у межах великого запиту.

Отже, хоча dobro.ua є важливим та успішним інструментом для залучення фінансових ресурсів на благодійні проєкти через верифіковані фонди, вона не покриває весь спектр завдань, що стоять перед системами для комплексної координації волонтерської допомоги, особливо тієї, що пов’язана з матеріальними ресурсами та активною роботою волонтерів на місцях. Залишається актуальною потреба у створенні гнучкої системи, яка дозволить ефективно управляти як фінансовими, так і матеріальними потоками, чітко структурувати потреби, координувати дії волонтерів та забезпечувати прозору операційну звітність.

Така система повинна підтримувати:

- створення деталізованих вимог із конкретним переліком необхідних предметів (назва, кількість, категорія);
- гнучку пріоритезацію запитів на допомогу;
- контроль статусу виконання по кожному окремому пункту вимоги;
- ведення обліку отримувачів допомоги;
- можливість встановлення та відстеження дедлайнів;
- інтеграцію з банківськими інструментами для відстеження фінансових зборів.

1.2 Виявлення та вирішення проблем

Детальний аналіз існуючих цифрових платформ та практик координації волонтерської допомоги виявляє низку системних проблем, які ускладнюють ефективну організацію діяльності, особливо в умовах великого потоку запитів та обмежених ресурсів. Багато існуючих рішень, хоч і пропонують певні інструменти, часто зосереджені лише на окремих аспектах, таких як збір фінансових пожертв або логістика гуманітарних вантажів, і не забезпечують комплексного підходу до управління всім циклом допомоги. Це створює необхідність у розробці інтегрованої системи, яка б вирішувала ключові виявлені проблеми.

Однією з найпоширеніших труднощів є відсутність стандартизованої структури для запитів на допомогу. У багатьох волонтерських ініціативах потреби формулюються та передаються у довільній формі – через текстові повідомлення в чатах, усні домовленості чи неструктуровані дописи в соціальних мережах. Це робить процес агрегації, фільтрації та автоматизованої обробки таких запитів надзвичайно складним, а часто й неможливим. Виникають труднощі з розумінням точного переліку необхідного, його кількості та специфікацій. Для вирішення цієї проблеми пропонується впровадження у програмній системі механізму формування структурованих вимог. Це передбачає створення запитів, де чітко вказується перелік необхідних предметів, їх належність до конкретних категорій (наприклад, "їжа", "медицина", "обладнання"), точна кількість та, за потреби, інші

важливі параметри. Такий підхід дозволить систематизувати інформацію, полегшить пошук та фільтрацію потреб, а також створить основу для автоматизації процесів обліку та звітності.

Іншою суттєвою проблемою є неможливість ефективної пріоритезації запитів. У великому потоці потреб критично важливі запити, що потребують негайного реагування, можуть легко загубитися серед менш термінових або бути поміченими із запізненням. Це призводить до неефективного розподілу уваги та ресурсів волонтерів. Рішенням цієї проблеми є введення в систему механізму пріоритезації вимог. Запровадження окремого атрибуту "пріоритет" (наприклад, зі значеннями "Стандартний", "Високий") для кожної вимоги або навіть окремого предмету в ній дозволить візуально виділяти найбільш нагальні потреби та фокусувати на них увагу волонтерів та координаторів. Це забезпечить більш раціональне планування діяльності та оперативне реагування на критичні ситуації.

Третім важливим викликом є низький рівень прозорості процесів для донорів та інших зацікавлених сторін. Часто люди, які надають фінансову чи матеріальну допомогу, не мають чіткого розуміння, як саме використовуються їхні внески. Непрозорість може підривати довіру до волонтерських ініціатив та знижувати мотивацію до подальшої підтримки. Для підвищення прозорості система повинна надавати інструменти для відстеження статусу виконання кожного пункту вимоги та формування детальних звітів. Можливість бачити, що конкретна потреба була закрита, що необхідні предмети закуплені або доставлені, разом із публікацією зведених звітів про діяльність, суттєво підвищить рівень довіри та підзвітності волонтерських організацій.

Нарешті, значною перешкодою для ефективної роботи є обмежена взаємодія існуючих систем з мобільними пристроями. Багато платформ є переважно веб-орієнтованими, що не завжди зручно для волонтерів, які активно працюють поза стаціонарними робочими місцями – займаються логістикою, розповсюдженням допомоги, безпосередньою комунікацією з отримувачами. Необхідність працювати з таблицями чи веб-інтерфейсами на маленькому екрані смартфона може бути

неефективною та часозатратною. Рішенням цієї проблеми є розробка повноцінного мобільного клієнта для системи. Такий застосунок, синхронізований в реальному часі з центральним сервером, надасть волонтерам зручний та оптимізований для мобільних пристроїв доступ до всієї необхідної інформації та функціоналу безпосередньо "в полі", значно підвищуючи їхню оперативність та продуктивність.

1.3 Постановка задачі

Виходячи з аналізу предметної галузі та виявлених проблем, задача даної дипломної роботи формулюється як проєктування та розробка мобільного застосунку, що слугуватиме клієнтською частиною комплексної інформаційної системи для ефективної організації та координації волонтерської допомоги. Основне призначення цього застосунку – стати основним робочим інструментом для волонтерів та, частково, для отримувачів допомоги, надаючи їм зручний та функціональний інтерфейс для взаємодії з централізованою базою даних та логікою системи. Передбачається, що застосунок об'єднає ключовий функціонал, необхідний для управління життєвим циклом волонтерських ініціатив, від створення зборів та фіксації потреб до звітування про їх виконання.

Метою розробки є суттєве спрощення процесу взаємодії між усіма учасниками волонтерського руху. Через надання інтуїтивно зрозумілого мобільного інтерфейсу ставиться за ціль дозволити волонтерам оперативно реагувати на нові запити допомоги, ефективно управляти статусом їх виконання, контролювати хід зборів та вести облік власної активності. Застосунок має сприяти підвищенню прозорості, підзвітності та загальної ефективності волонтерської діяльності.

Для досягнення поставленої мети мобільний застосунок буде реалізований як клієнтська частина, що взаємодіє з централізованим серверним програмним інтерфейсом (API). Це забезпечить синхронізацію даних у реальному часі та доступ до актуальної інформації для всіх користувачів. Функціонально, застосунок повинен реалізовувати можливості для створення та детального перегляду

цільових зборів коштів чи матеріальних ресурсів, дозволяючи користувачам бачити опис, мету, відповідальних осіб та актуальний статус збору. Невід'ємною частиною є механізм роботи зі структурованими вимогами: користувачі (переважно отримувачі або волонтери від їх імені) зможуть додавати деталізовані запити, вказуючи конкретні предмети, їх необхідну кількість, належність до певної категорії та пріоритетність. Також волонтери матимуть можливість переглядати загальний список активних вимог та, за наявності відповідних прав, долучатися до їх виконання або до вже існуючих активних зборів.

Важливим елементом є управління особистими даними та історією активності. Користувачі матимуть доступ до особистого кабінету, де зможуть переглядати та редагувати свій профіль, бачити історію своєї участі у волонтерських проєктах, включаючи виконані завдання чи закриті потреби. Передбачається також функціонал, що дозволить завантажувати звіти про виконання вимог, наприклад, фотопідтвердження доставки допомоги чи документи про закупівлю, що підвищить рівень прозорості. Для зручності навігації та швидкого доступу до ключової інформації будуть реалізовані головні екрани (дашборди) для різних ролей користувачів. Головна сторінка волонтера може відображати останні активні збори та найбільш пріоритетні вимоги, наприклад, у вигляді зручних для перегляду горизонтальних стрічок або списків. Головна сторінка отримувача може показувати статус його власних запитів та збори, які були відкриті спеціально для їх закриття. Детальна інформація про кожен збір чи вимогу буде доступна на окремих екранах-картках.

Принциповим рішенням є те, що застосунок не буде безпосередньо обробляти фінансові транзакції. Це дозволить уникнути значних юридичних та технічних складнощів, пов'язаних із впровадженням платіжних шлюзів та дотриманням фінансових регуляцій. Натомість, для фінансових зборів акцент буде зроблено на інтеграцію з зовнішніми сервісами, наприклад, через відображення посилань на банківські "банки" (як Monobank Jars), де користувачі зможуть здійснити пожертву, а застосунок зможе (через відповідне API банку) показувати

прогрес збору. Таким чином, фокус мобільного застосунку зміщується на організаційно-логістичні аспекти волонтерської діяльності.

Особливу увагу приділено модулю управління вимогами, оскільки структурований підхід до фіксації потреб є ключовим для ефективного планування та розподілу ресурсів. Кожна вимога матиме чітку структуру з переліком предметів, що дозволить системі точно визначати потреби, зв'язувати їх із конкретними зборами чи волонтерами-виконавцями, а також автоматизувати частину звітності. Синхронізація даних з сервером у реальному часі гарантуватиме актуальність інформації для всіх користувачів.

Розробка такого мобільного застосунку розглядається як ключовий компонент для створення сучасної та ефективної системи координації волонтерської допомоги. Надання зручного інтерфейсу для швидкої участі, управління процесами та підвищення прозорості сприятиме залученню більшої кількості волонтерів, кращому розподілу наявних ресурсів та, в кінцевому рахунку, посиленню спроможності громадянського суспільства оперативно та дієво реагувати на соціальні виклики.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Розроблюваний мобільний застосунок є клієнтською частиною комплексної системи координації волонтерської допомоги. Його основна мета – надати зручний та ефективний інструмент для ключових учасників волонтерського процесу, таких як волонтери та отримувачі допомоги, для взаємодії з централізованою системою в режимі реального часу. Фокус розробки зосереджений на створенні інтуїтивно зрозумілого інтерфейсу для мобільних пристроїв, що забезпечить оперативне управління ресурсами та комунікацію.

Реалізація застосунку здійснюється за допомогою фреймворку Flutter [2], що забезпечує кросплатформеність (роботу на Android та iOS) з єдиної кодової бази, оптимізуючи розробку та підтримку. Усі дані синхронізуються з серверною частиною через програмний інтерфейс (API), гарантуючи актуальність інформації. Нижче наведено основні сценарії використання застосунку, що визначають його функціональні вимоги.

Сценарій 1. Автентифікація та доступ до системи. Будь-який користувач, чи то волонтер, чи отримувач допомоги, для початку роботи із захищеними функціями системи повинен пройти процес автентифікації. Користувач відкриває застосунок та переходить на екран входу, де вводить свою електронну пошту та пароль. Ці дані відправляються на сервер для перевірки. У разі успішної автентифікації сервер повертає унікальний маркер доступу (токен), який застосунок зберігає на пристрої для подальших сесій. Користувач отримує доступ до функціоналу відповідно до своєї ролі (волонтер або отримувач). Якщо дані невірні, користувач отримує повідомлення про помилку і залишається на екрані входу. Також користувач повинен мати можливість вийти зі свого облікового запису, що призводить до видалення збереженого токена та повернення на екран входу.

Сценарій 2. Робота волонтера зі зборами. Волонтер, увійшовши до системи, може взаємодіяти зі зборами коштів або матеріальних ресурсів. Він має можливість переглянути загальний список активних зборів, використовуючи фільтри (наприклад, за категорією потреб, статусом) та пошук (за ключовими словами в

назві чи описі). Волонтер може відкрити детальний екран будь-якого збору, щоб ознайомитися з його метою, описом, поточним прогресом (якщо це фінансовий збір, може бути відображена інформація з пов'язаного рахунку Monobank), відповідальним волонтером та списком пов'язаних потреб (вимог). Важливою функцією є можливість для волонтера створити новий збір: для цього він заповнює відповідну форму, вказуючи назву, детальний опис, цільову суму (для фінансових зборів) або перелік необхідних ресурсів, прикріплює зображення та вказує посилання на банківський рахунок (якщо є). Волонтер також може редагувати інформацію про власні збори та змінювати їх статус (наприклад, перевести у "Завершений" або "Скасований").

Сценарій 3. Робота з вимогами (запитами на допомогу). Цей сценарій є ключовим як для отримувачів, так і для волонтерів. Отримувач допомоги може самостійно створити новий запит (вимогу) через відповідний інтерфейс у застосунку. При створенні він вказує загальну назву потреби, бажаний термін виконання (дедлайн, якщо відомий) та, що найважливіше, додає перелік конкретних предметів, які необхідні. Для кожного предмета зазначається його назва, потрібна кількість та категорія (їжа, медицина, одяг, обладнання тощо). Також є можливість вказати пріоритетність вимоги (наприклад, "Висока" для критичних потреб). Отримувач може переглядати список своїх створених вимог та відстежувати статус їх виконання. Волонтер, у свою чергу, може переглядати загальний список активних вимог від різних отримувачів, застосовуючи фільтри за категорією, пріоритетом чи місцем розташування (якщо така інформація передбачена). Волонтер може ознайомитися з деталями конкретної вимоги та взяти її в роботу або долучитися до її виконання, якщо система підтримує таку логіку координації. Також волонтер може мати можливість створювати вимогу від імені отримувача, якщо той не має доступу до застосунку.

Сценарій 4. Управління профілем користувача. І волонтер, і отримувач допомоги мають доступ до особистого кабінету, де відображається їхня профільна інформація. Користувачі можуть переглядати свої дані (ім'я, контактну

інформацію, роль у системі, історію активності – створені збори/вимоги, виконані завдання тощо). Важливою функцією є можливість редагування профілю, зокрема оновлення контактних даних (телефон, email), зміна аватара, а для волонтера – також вказівка своєї спеціалізації чи навичок та статусу доступності для нових завдань.

Сценарій 5. Дашборд волонтера. Для зручності орієнтації в поточних завданнях та можливостях волонтер має доступ до персоналізованого дашборду (головного екрану). На цьому екрані може бути згрупована найактуальніша інформація: наприклад, відображатися кілька останніх зборів, ініційованих цим волонтером, та список найбільш пріоритетних чи нещодавно створених вимог, які потребують уваги. Це дозволяє волонтеру швидко оцінити ситуацію та перейти до відповідних дій.

Сценарій 6. Звітність та прозорість. Хоча застосунок не обробляє фінансові транзакції напряму, він сприяє прозорості. Для фінансових зборів, при натисканні на банківське посилання, користувач перенаправляється у безпечне середовище банку для здійснення пожертви. Прогрес збору може візуалізуватися через інтеграцію з API Монобанку. Щодо матеріальної допомоги, система передбачає можливість відстеження статусу виконання конкретних пунктів вимог. У майбутньому можлива реалізація функціоналу завантаження звітів волонтерами (наприклад, фотопідтвердження доставки), які будуть прив'язані до конкретного збору або вимоги, що ще більше підвищить рівень прозорості для донорів.

Нефункціональні вимоги:

- забезпечення швидкого завантаження застосунку при запуску;
- гарантування плавного переходу між екранами та швидкого відгуку інтерфейсу на дії користувача;
- оптимізація завантаження списків даних (збори, вимоги) для уникнення тривалих затримок;
- мінімізація впливу взаємодії зі сторонніми сервісами (наприклад, API Монобанку) на загальну швидкість роботи інтерфейсу;

- реалізація коректної обробки можливих помилок (відсутність інтернет-з'єднання, помилки відповіді сервера, некоректні дані) з наданням зрозумілих повідомлень користувачеві;
- забезпечення стабільної роботи застосунку та мінімізація ймовірності непередбачених завершень роботи;
- впровадження безпечного механізму зберігання токенів автентифікації на пристрої користувача;
- унеможливлення зберігання паролів користувачів у відкритому вигляді на стороні клієнта;
- розробка інтуїтивно зрозумілого інтерфейсу користувача, легкого для освоєння користувачами з різним рівнем технічної підготовки;
- забезпечення простої та логічної навігаційної структури застосунку;
- гарантування легкої доступності ключової інформації та основних функцій;
- забезпечення коректної роботи застосунку на широкому спектрі пристроїв під управлінням актуальних версій операційних систем Android та iOS;
- реалізація адаптивного дизайну, що коректно відображається на екранах різних розмірів та співвідношень сторін;
- написання чистого, добре структурованого та (за необхідності) коментованого коду для полегшення його подальшої модифікації, виправлення помилок та додавання нового функціоналу;
- використання архітектурних підходів, що сприяють легкості підтримки та масштабування застосунку.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір технологій та інструментів

Ключовим рішенням на етапі проєктування став вибір основного інструментарію для розробки мобільного застосунку. Зважаючи на потребу забезпечити швидку розробку, високу продуктивність та доступність програми для користувачів основних мобільних платформ, було обрано фреймворк Flutter, розроблений Google, у поєднанні з мовою програмування Dart [3]. Цей вибір обґрунтовується низкою вагомих переваг Flutter, що роблять його оптимальним для даного проєкту.

Насамперед, однією з визначальних переваг Flutter є його кросплатформеність. Завдяки Flutter, розробники можуть створювати мобільні застосунки для операційних систем Android та iOS, використовуючи єдину кодову базу. Це не лише значно прискорює сам процес розробки, оскільки не потрібно писати та підтримувати окремий код для кожної платформи, але й суттєво скорочує витрати ресурсів та часу на подальшу підтримку, оновлення та виправлення помилок. Водночас, Flutter дозволяє створювати інтерфейси, які виглядають і відчуваються як нативні на кожній платформі, або ж реалізувати єдиний унікальний дизайн.

Іншою важливою характеристикою є висока продуктивність застосунків, створених на Flutter. На відміну від деяких інших кросплатформених рішень, Flutter не використовує інтерпретацію коду або проміжні мости для взаємодії з нативними компонентами під час виконання. Замість цього, код Dart компілюється безпосередньо в нативний ARM або x64 код, що забезпечує швидкість роботи, близьку до тієї, що демонструють застосунки, написані на Swift/Objective-C для iOS або Kotlin/Java для Android [4]. Це особливо важливо для забезпечення плавного відгуку інтерфейсу, швидкої анімації та комфортної взаємодії користувача з програмою, навіть при роботі зі значними обсягами даних чи складною логікою.

Flutter також пропонує надзвичайно багатий та гнучкий інструментарій для побудови користувацького інтерфейсу (UI). В основі лежить концепція "все є віджет". Фреймворк надає велику бібліотеку готових до використання, високоякісних та легко кастомізованих віджетів, що реалізують як гайдлайни Material Design від Google [5], так і Cupertino від Apple [6]. Це дозволяє розробникам швидко створювати візуально привабливі, сучасні та інтуїтивно зрозумілі інтерфейси. Гнучка система компонування віджетів дає повний контроль над кожним пікселем на екрані, дозволяючи реалізувати практично будь-які дизайнерські задуми без компромісів.

Не менш важливою є активна та постійно зростаюча спільнота розробників Flutter, а також величезна екосистема готових пакетів та бібліотек, доступних через репозиторій pub.dev. Це означає, що для багатьох типових завдань, таких як робота з мережею та REST API, управління станом застосунку [10], навігація між екранами, робота з базами даних, інтеграція з апаратними можливостями пристрою чи сторонніми сервісами (як API Монобанку [9]), вже існують перевірені та надійні рішення. Використання цих пакетів дозволяє значно прискорити розробку, уникнути "винаходу велосипеда" та зосередитися на реалізації унікальної бізнес-логіки волонтерського застосунку.

Отже, вибір Flutter як основного фреймворку надає оптимальне поєднання кросплатформеності, високої продуктивності, гнучкості у побудові UI та підтримки активної спільноти, що робить його ідеальним рішенням для розробки сучасного та ефективного мобільного застосунку для координації волонтерської допомоги. Для середовища розробки було обрано Visual Studio Code через його зручність та потужні інструменти для роботи з Flutter. Для управління кодом використовується система контролю версій Git. Інші специфічні інструменти та бібліотеки, що стосуються управління станом, мережевої взаємодії та навігації, детально розглядаються у відповідних розділах проєктування.

3.2 Проєктування архітектури ПЗ

Для забезпечення структурованості, підтримуваності та масштабованості мобільного застосунку, його архітектура була спроєктована за трирівневою моделлю. Цей підхід дозволяє чітко розмежувати відповідальність між різними частинами програми: відображенням інтерфейсу, обробкою бізнес-логіки та взаємодією з зовнішніми даними. Використання такої структури спрощує розробку, тестування окремих компонентів та подальше внесення змін до функціоналу застосунку. Основними компонентами цієї архітектури є шар представлення, сервісний шар та шар даних, що взаємодіє з API.

Перший рівень, шар представлення або Presentation Layer, є тим компонентом, з яким безпосередньо контактує кінцевий користувач. Його завдання полягає у візуалізації інформації та наданні елементів управління. У контексті технології Flutter цей шар реалізується через систему віджетів, які формують окремі екрани (Screens) застосунку. Сюди входять кнопки, поля для введення тексту, списки, зображення та інші візуальні елементи. Цей шар не містить складної бізнес-логіки; його основна функція – відображати дані, що надходять від нижчих шарів архітектури, та приймати вхідні події від користувача, такі як натискання кнопок чи введення тексту, передаючи їх для подальшої обробки. Для управління навігацією між різними екранами застосунку використовується спеціалізований пакет go_router [7], який дозволяє визначати маршрути та керувати переходами у структурований спосіб.

Другий рівень, сервісний шар або Service Layer, виступає як проміжна ланка, що містить основну бізнес-логіку клієнтського застосунку. Він отримує події та дані від шару представлення, обробляє їх згідно з визначеними правилами та координує взаємодію з шаром даних. Наприклад, коли користувач ініціює дію на екрані, як показано на діаграмі послідовності (рисунок 4.1), ця дія передається саме до сервісного шару. Тут відбувається перевірка вхідних даних (валідація), застосовуються бізнес-правила, специфічні для програми (наприклад, перевірка можливості виконати певну дію відповідно до ролі користувача), і формується

запит на отримання або відправку даних до наступного шару. Також сервісний шар відповідає за підготовку даних у форматі, зручному для відображення на шарі представлення.

Третій рівень, шар даних або Data Layer, відповідає за всю комунікацію з зовнішнім джерелом даних, яким у даному випадку є віддалений backend-сервер, доступний через програмний інтерфейс (REST API). Цей шар повністю інкапсулює деталі мережевої взаємодії. Його завдання – приймати запити від сервісного шару, формувати відповідні HTTP-запити до конкретних ендпоінтів backend API, відправляти їх на сервер, отримувати відповіді, обробляти їх (включаючи коди стану та можливі помилки) та перетворювати отримані дані (зазвичай у форматі JSON) у структуровані об'єкти мови Dart (моделі даних). Ці моделі даних потім повертаються на сервісний шар. Аналогічно, при необхідності відправити дані на сервер (наприклад, при створенні нового збору), цей шар приймає Dart-об'єкти від сервісного шару, перетворює їх у формат JSON та включає у тіло відповідного HTTP-запиту. Використання цього шару дозволяє ізолювати решту застосунку від деталей конкретної реалізації API та полегшує можливу зміну джерела даних у майбутньому.

Послідовність взаємодії між цими шарами, як ілюструє діаграма на рисунку 3.1, є наступною. Користувач ініціює дію на шарі представлення. Ця дія передається сервісному шару. Сервісний шар обробляє дію та формує запит до шару даних для отримання або зміни інформації. Шар даних, використовуючи API клієнт, надсилає відповідний HTTP-запит до backend API. Після отримання відповіді (у форматі JSON) від сервера, шар даних перетворює її у зрозумілі для Dart моделі даних і повертає на сервісний шар. Сервісний шар може виконати додаткову обробку отриманих даних і потім передає оновлені дані або повідомлення про зміну стану на шар представлення. Шар представлення, отримавши ці оновлення, перебудовує відповідні частини інтерфейсу, щоб відобразити актуальну інформацію для користувача на його екрані. Такий чіткий

потік даних та розподіл відповідальностей робить архітектуру передбачуваною та керованою.

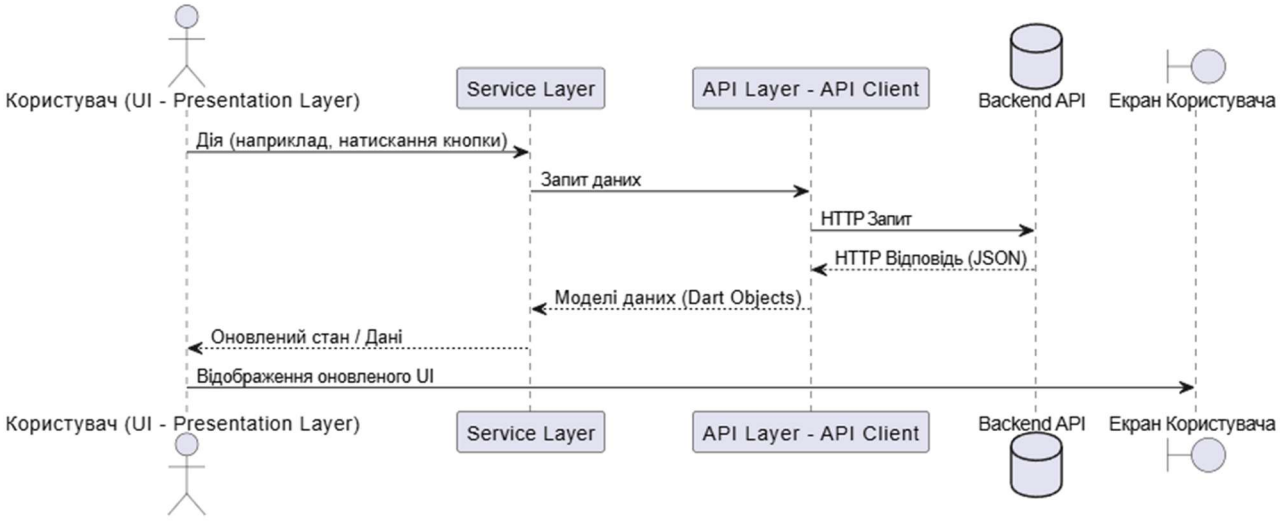

Рисунок 3.1 – Потік даних та взаємодія шарів в архітектурі мобільного застосунку

3.3 UML проєктування мобільного застосунку

Для детального візуального представлення функціональних можливостей розроблюваного мобільного застосунку та способів взаємодії з ним користувачів було створено діаграму прецедентів (Use Case diagram), наведену на рисунку 3.2. Ця діаграма чітко визначає основних дійових осіб (акторів), які працюють із системою через мобільний клієнт, та завдання (прецеденти), які вони можуть виконувати. Такий підхід дозволяє структурувати вимоги до функціоналу та слугує основою для подальшого проєктування інтерфейсів і логіки програми.

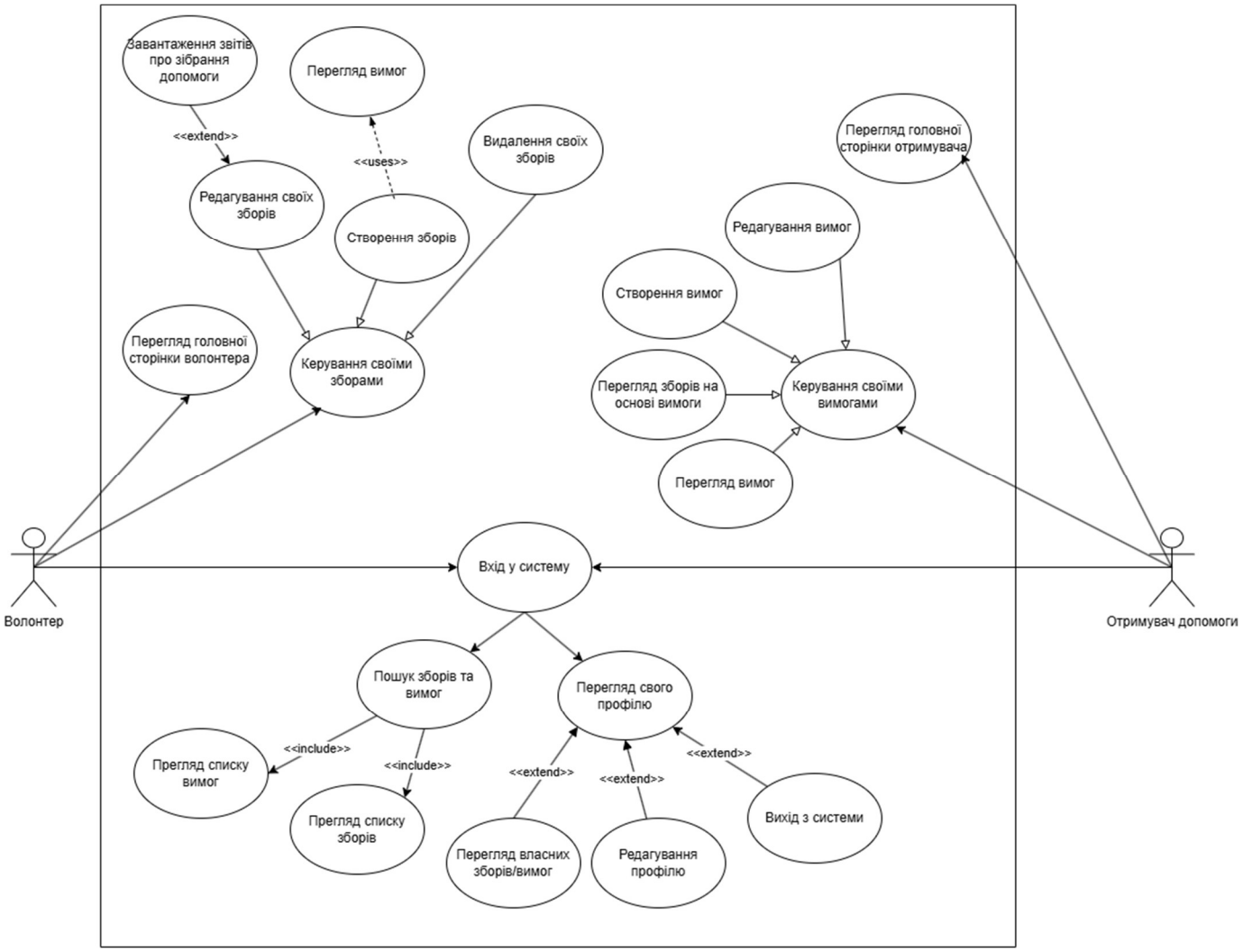

Рисунок 3.2 – Діаграма прецедентів мобільного застосунку

На діаграмі визначено двох основних акторів: "Волонтер" та "Отримувач допомоги". Обидва актори взаємодіють із системою, починаючи з базового прецеденту "Вхід у систему". Після успішної автентифікації кожен актор отримує доступ до свого набору функцій. Загальними для обох акторів є також прецедент "Перегляд свого профілю". При цьому функціональність перегляду профілю може бути розширена прецедентами "Вихід з системи", "Редагування профілю" та "Перегляд власних зборів/вимог", надаючи користувачам можливість не тільки бачити інформацію, але й змінювати її або переглядати пов'язану активність безпосередньо з екрану профілю (якщо така навігація передбачена).

Специфічний функціонал для актора "Волонтер" пов'язаний переважно з управлінням зборами та вимогами. Він може ініціювати прецедент "Керування

своїми зборами". Цей загальний прецедент, в свою чергу, охоплює більш конкретні дії: "Створення зборів", "Редагування своїх зборів" та "Видалення своїх зборів". Процес створення збору ("Створення зборів") використовує інформацію з прецеденту "Перегляд вимог", оскільки для формування збору необхідно обрати потреби, на які він спрямований. Прецедент "Редагування своїх зборів" може бути опціонально розширений прецедентом "Завантаження звітів про зібрання допомоги", наприклад, при завершенні збору. Також волонтеру доступний прецедент "Перегляд головної сторінки волонтера", який агрегує інформацію з інших прецедентів, таких як перегляд зборів та вимог.

Актор "Отримувач допомоги" має доступ до функціоналу, пов'язаного з управлінням власними потребами – прецедент "Керування своїми вимогами". Цей прецедент включає такі дії як "Створення вимог", "Перегляд вимог" та "Редагування вимог". Крім того, отримувачу доступний "Перегляд зборів на основі вимоги", що дозволяє бачити, які кампанії були ініційовані для задоволення його потреб. Також для отримувача передбачено "Перегляд головної сторінки отримувача", який надає швидкий огляд його поточних запитів та пов'язаних зборів.

Окремо на діаграмі виділено функцію "Пошук зборів та вимог", яка є доступною для обох типів користувачів після входу в систему. Результатом виконання пошуку є відображення списків, тому цей прецедент включає (<<include>>) функціональність "Перегляд списку зборів" та "Перегляд списку вимог".

3.4 Проєктування UI/UX

Етап проєктування користувацького інтерфейсу та досвіду взаємодії для мобільного застосунку волонтерської допомоги мав на меті не лише створити візуально привабливу оболонку, а й забезпечити максимальну ефективність та зручність для кінцевих користувачів у виконанні їхніх специфічних завдань. Враховуючи динамічний та часто стресовий характер волонтерської роботи,

ключовими принципами проєктування стали ясність, швидкий доступ до критично важливої інформації та інтуїтивність навігації. Прототипи та візуальний стиль були розроблені в середовищі Figma [8], що дозволило ітераційно вдосконалювати макети перед початком реалізації у Flutter.

Загальна концепція дизайну базується на принципах мінімалізму та функціональності. Обрано світлу колірну гаму, яка сприяє кращому сприйняттю інформації та не втомлює очі при тривалому використанні. Акцентні кольори використовуються помірно для виділення важливих елементів керування чи статусів. Типографіка підібрана з урахуванням читабельності на мобільних пристроях, з достатнім розміром шрифтів та міжрядковими інтервалами. Для структурування контенту активно використовуються картки із закругленими кутами та легкими тінями, що візуально групує пов'язану інформацію та покращує загальну організацію простору екрану. Навігація між основними функціональними блоками реалізована через стандартну нижню панель, що забезпечує швидкий та передбачуваний доступ до головного екрану, пошуку та профілю користувача з будь-якої точки застосунку.

Проєктування екрану профілю користувача (ілюстрованого на рисунку 3.3) передбачало створення єдиного простору для перегляду та редагування персональної інформації волонтера. Ключова ідентифікаційна інформація, така як аватар, ім'я, прізвище та контактні дані, винесена на чільне місце у верхню картку. Використання іконок поряд з текстовими даними (пошта, телефон, вік, спеціалізація) покращує швидкість сканування інформації користувачем. Розміщення кнопки редагування у верхньому правому куті відповідає загальноприйнятим патернам мобільного дизайну, роблячи функцію редагування легко доступною, але не нав'язливою. Нижче основної інформації профілю виводиться секція, присвячена активності користувача, наприклад, збори, якими він опікується. Представлення кожного збору у вигляді інформативної картки з візуальним елементом (зображенням), назвою, статусом, прогресом та ключовими учасниками (волонтер, отримувач) дозволяє швидко оцінити стан проєкту, не

переходячи на окремий екран деталей, хоча така можливість також передбачається при натисканні на картку.

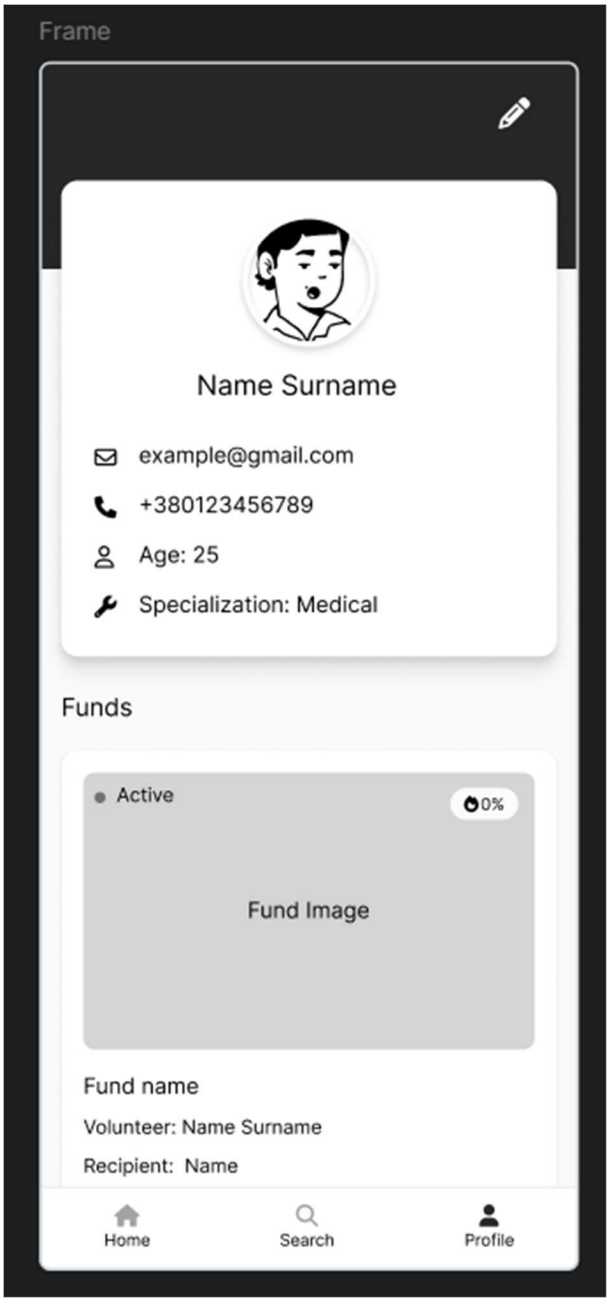

Рисунок 3.3 – Прототип екрану профілю користувача

Дизайн головного екрану волонтера, або дашборду (як показано на рисунку 4.4), був спроєктований так, щоб надавати максимально релевантний та актуальний зріз інформації одразу після входу в систему. Він діє як інформаційний хаб, агрегуючи дані з різних частин системи. Верхня частина дашборду відведена під

горизонтально прокручуваний список власних зборів ("My Funds"). Такий формат дозволяє компактно розмістити кілька проєктів, не займаючи багато вертикального простору, та надає швидкий огляд їх статусів і прогресу за допомогою візуальних індикаторів (статусних міток та кругових діаграм відсотка виконання). Нижня частина дашборду присвячена актуальним потребам ("Last requirements"). Представлення вимог у вигляді вертикального списку карток дозволяє відобразити більше деталей по кожній потребі, включаючи назву, автора запиту, ключові позиції та термін виконання. Наявність візуальних індикаторів (іконка біля пріоритету, іконка годинника біля дедлайну) допомагає користувачеві швидше оцінити терміновість та характер потреби. Розділення дашборду на секції власних завдань та загальних потреб дозволяє волонтеру ефективно балансувати між управлінням своїми проєктами та реагуванням на нові запити системи.

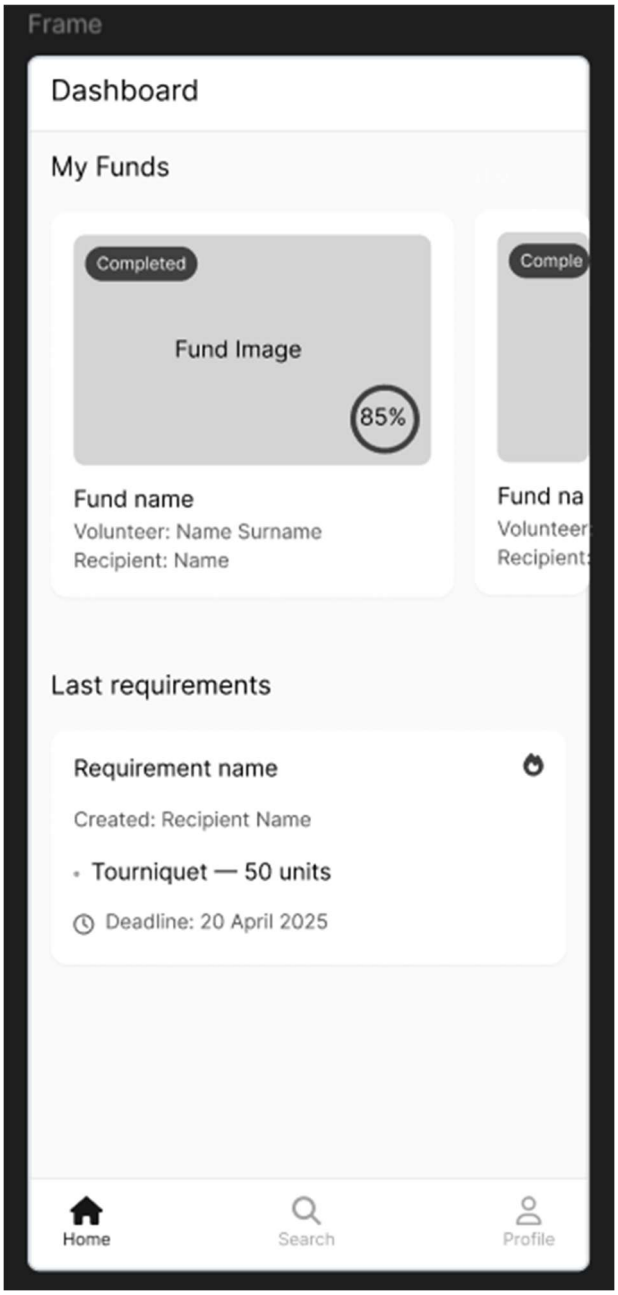

Рисунок 3.4 – Прототип екрану головної сторінки волонтера

В цілому, проєктування UI/UX було спрямоване на створення не просто набору екранів, а цілісного досвіду, що підтримує користувача у його волонтерській діяльності, мінімізує час на пошук інформації та виконання рутинних операцій, а також забезпечує відчуття контролю та поінформованості щодо процесів, у яких він бере участь.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Організація кодової бази та навігації

Для забезпечення належної організації кодової бази, її читабельності та можливості легкого розширення, проєкт мобільного застосунку було структуровано за функціональними та архітектурними ознаками. Весь основний вихідний код програми розміщується в стандартній для Flutter-проєктів директорії lib. Усередині цієї директорії створено спеціалізовані піддиректорії для логічного групування файлів. Зокрема, моделі даних, що описують структуру об'єктів системи (такі як Fund, Requirement, User), розташовані в піддиректорії models. Кожен екран застосунку (наприклад, екран входу, дашборд, профіль) реалізовано як окремий віджет і розміщено у піддиректорії screens. Класи, відповідальні за взаємодію з зовнішнім API (сервіси), знаходяться в директорії services (або api). Віджети, які використовуються на декількох екранах і не мають специфічної бізнес-логіки, винесені в common_widgets. Різноманітні допоміжні функції та конфігураційні файли містяться в utils, а компоненти, що відповідають за управління станом застосунку, можуть бути згруповані в окремій директорії. Такий підхід полегшує навігацію по коду та його підтримку.

Навігація між різними екранами є фундаментальним аспектом будь-якого мобільного застосунку. Для реалізації потужної та декларативної системи маршрутизації в даному проєкті було обрано популярний пакет go_router [7]. Цей пакет дозволяє чітко визначати всі доступні маршрути (шляхи) в застосунку, управляти стеком навігації, обробляти перенаправлення на основі стану (наприклад, стану автентифікації користувача) та передавати параметри між екранами. Вся конфігурація маршрутизації інкапсульована в окремому файлі (наприклад, router.dart), що сприяє централізованому управлінню навігаційною логікою.

```
GoRouter createRouter(AuthState authState) {
  return GoRouter(
```

```
    initialLocation: '/splash',
    refreshListenable: authState,
    // ... логіка redirect та визначення routes ...
  );
}
```

Наведений фрагмент показує функцію createRouter, яка створює та повертає екземпляр GoRouter. Параметр initialLocation вказує, який маршрут має бути завантажений при першому запуску застосунку (у даному випадку це /splash для сплеш-екрану). Важливим є параметр refreshListenable, якому передається об'єкт authState. Це дозволяє роутеру автоматично реагувати на зміни стану автентифікації користувача (наприклад, вхід або вихід) та ініціювати перевірку правил перенаправлення.

GoRouter надає потужний механізм redirect для керування доступом до певних маршрутів залежно від різних умов, наприклад, статусу автентифікації користувача.

```
redirect: (context, state) {
  final loggedIn = authState.isLoggedIn;
  final goingToLogin = state.fullPath == '/login';
  final goingToSplash = state.fullPath == '/splash';

  if (!authState.isInitialized) return null;
  if (!loggedIn) {
    return goingToLogin ? null : '/login';
  }

  if (goingToSplash || goingToLogin) return '/dashboard';

  return null;
},
```

Функція redirect викликається перед кожним переходом. Вона аналізує поточний стан автентифікації (loggedIn) та маршрут, на який користувач

намагається перейти (state.fullPath). Якщо стан автентифікації ще не визначено (!authState.isInitialized), перенаправлення не відбувається, дозволяючи відобразити сплеш-екран. Якщо користувач не автентифікований (!loggedIn), будь-яка спроба перейти на маршрут, відмінний від /login, буде перенаправлена на екран входу. Якщо ж користувач вже автентифікований і випадково намагається перейти на сплеш-екран або екран входу, його буде автоматично перенаправлено на головний екран (/dashboard). Ця логіка забезпечує захист маршрутів, доступних лише для авторизованих користувачів.

Основна конфігурація маршрутів відбувається у списку routes. Тут визначаються шляхи та віджети-конструктори (builders) для кожного екрану.

```
      routes: [
        GoRoute(
          path: '/splash',
          builder: (context, state) => const SplashScreen(),
        ),
        GoRoute(
          path: '/login',
          builder: (context, state) => const LoginScreen(),
        ),
        GoRoute(
          path: '/profile/edit',
          builder: (context, state) {
            final volunteer = state.extra as Volunteer;
            return EditProfileScreen(volunteer: volunteer);
          },
        ),

        ShellRoute(
          builder: (context, state, child) => MainScaffold(child:
child),
          routes: [
            GoRoute(
              path: '/dashboard',
              builder: (context, state) {
```

```
                final role = authState.role;
                if (role == 'Volunteer') return const
DashboardVolunteerPage();
                else if (role == 'Recipient') return const
DashboardRecipientPage();
                else return const Text('Unknown role');
              },
            ),
            GoRoute(
              path: '/search',
              builder: (context, state) => const FundSearchPage(),
            ),
            GoRoute(
                path: '/profile',
                builder: (context, state) {
                  final role = authState.role;
                  if (role == 'Volunteer') return const
ProfileScreen();
                  else if (role == 'Recipient') return const
RecipientProfileScreen();
                  else return const Text('Unknown role');
                }),
          ],
        ),
      ],
```

Кожен GoRoute визначає шлях (path) та функцію builder, яка повертає віджет відповідного екрану. Для маршруту /profile/edit показано приклад отримання додаткових даних (state.extra), переданих при навігації, для ініціалізації екрану редагування профілю конкретного волонтера. Використання ShellRoute дозволяє застосувати загальну структуру (наприклад, MainScaffold, що може містити AppBar та BottomNavigationBar) до групи вкладених маршрутів (/dashboard, /search, /profile). Це забезпечує консистентний вигляд та поведінку для основних екранів застосунку, доступних після автентифікації. Також на прикладі маршрутів

/dashboard та /profile продемонстровано можливість динамічного вибору віджета екрану залежно від ролі поточного користувача, отриманої з authState.

На рисунку 4.1 наведено приклад реалізації загальної оболонки застосунку MainScaffold, яка включає нижню панель навігації з основними розділами: "Головна", "Пошук", "Профіль", що забезпечує швидкий доступ до ключового функціоналу.

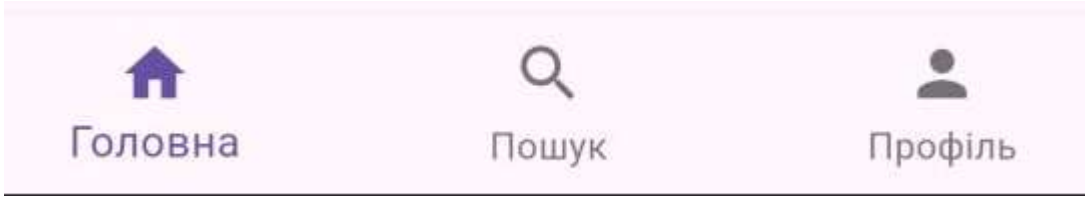

Рисунок 4.1 – Скріншот прикладу реалізації навігаційної панелі

На рисунку 4.2 зображено екран входу до системи (LoginScreen), де користувач може ввести свої облікові дані для автентифікації.

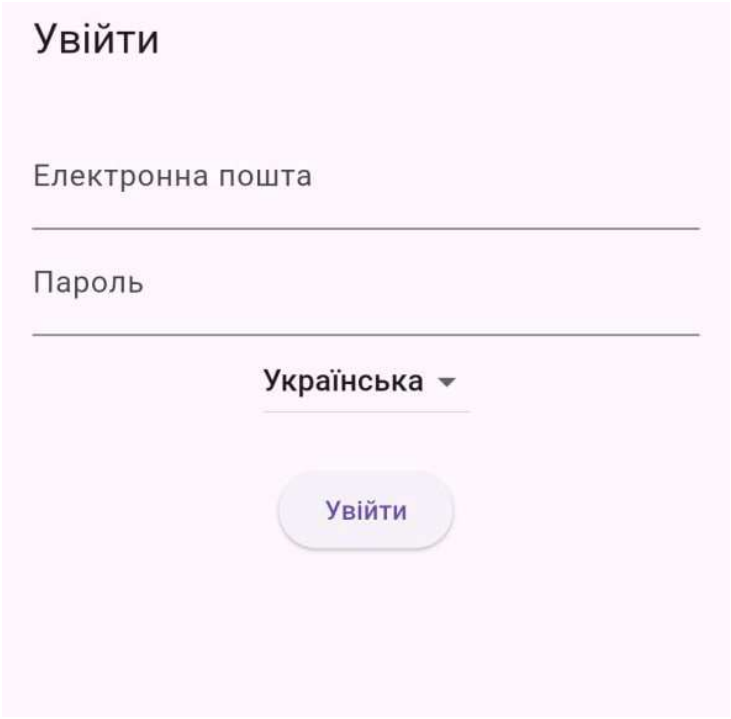

Рисунок 4.2 – Скріншот прикладу сторінки логіну

4.2 Реалізація користувацького інтерфейсу та компонентів

Для візуалізації інформації про волонтерські збори в мобільному застосунку було розроблено спеціалізований компонент – віджет FundCard. Основне призначення цього віджету – надати користувачеві компактне, але інформативне представлення кожного окремого збору в списках (наприклад, на дашборді або в результатах пошуку), а також слугувати інтерактивним елементом для переходу до більш детальної інформації про обраний збір. FundCard реалізований як StatefulWidget у Flutter, що дозволяє йому управляти власним внутрішнім станом, зокрема, відображати динамічно оновлюваний прогрес збору коштів.

При створенні екземпляру FundCard на екрані, в його методі initState, ініціюється процес отримання даних про поточний прогрес фінансового збору. Цей процес є асинхронним, оскільки вимагає звернення до зовнішнього сервісу.

```
double _percentageCollected = 0.0;

  @override
  void initState() {
    super.initState();
    _loadProgress();
  }

  Future<void> _loadProgress() async {
    final jarId = widget.fund.longJarId;
    try {
      final jarData = await getMonoJarData(jarId);
      if (mounted) {
         setState(() {
           _percentageCollected = jarData.progressPercent / 100;
         });
      }
    } catch (e) {
      print("Error fetching MonoJar data: $e");
      if (mounted) {
        setState(() {
```

```
          _percentageCollected = 0.0;
        });
      }
    }
  }
```

Метод _loadProgress відповідає за отримання даних про суму зібраних коштів. Він використовує longJarId з об'єкту widget.fund (дані про поточний збір, передані до віджету) для ідентифікації відповідної "банки" в системі Monobank. За допомогою функції getMonoJarData (що інкапсулює логіку звернення до API Монобанку [9]) отримуються дані про стан "банки". Оскільки це асинхронна операція, використовується async/await. Після успішного отримання даних, відсоток зібраних коштів обчислюється та зберігається у змінній стану _percentageCollected. Виклик setState сигналізує Flutter, що стан віджету змінився, і його потрібно перемалювати для відображення оновленого прогресу. Додано перевірку if (mounted) перед викликом setState для уникнення помилок, якщо віджет буде видалено з дерева до завершення асинхронної операції.

Метод build відповідає за побудову візуального представлення FundCard на основі поточних даних про збір та його прогрес.

```
// fund_card.dart (фрагмент методу build)
  @override
  Widget build(BuildContext context) {
    // ... отримання даних fund, fundName, volunteerName тощо ...
    return GestureDetector(
      onTap: () {
        Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) => FundDetailScreen(fundId:
widget.fund.id),
          ),
        );
      },
```

```
      child: SizedBox(
        width: 280,
        height: 280,
        child: Card( /* ... стилізація Card: форма, тінь ... */
          child: Padding( /* ... внутрішні відступи ... */
            child: Column( /* ... розміщення елементів ... */
              // ... Блок зображення з індикаторами ...
              // ... Блок текстової інформації ...
            ),
          ),
        ),
      ),
    );
  }
```

Картка обгорнута у віджет GestureDetector для реалізації переходу на екран FundDetailScreen при натисканні, передаючи fund.id як параметр. Розміри картки фіксовані (SizedBox) для забезпечення однакового вигляду в списках. Основою є віджет Card, який надає стандартний вигляд картки з тінню та закругленими кутами. Внутрішній вміст організовано у Column для вертикального розташування блоку з зображенням та текстової інформації.

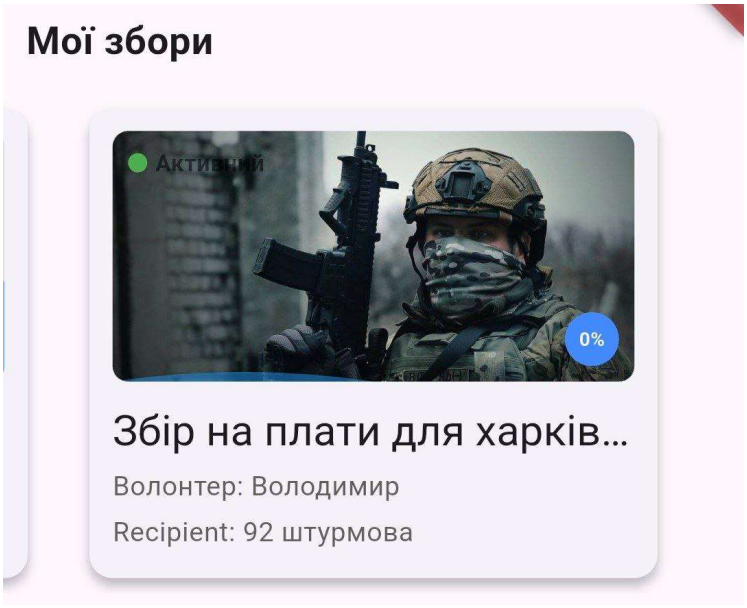

Рисунок 4.3 – Скріншот картки збору.

4.3 Взаємодія з Backend API та управління даними

Фундаментом для динамічної роботи мобільного застосунку є його здатність надійно та ефективно обмінюватися даними з серверною частиною системи. Ця взаємодія реалізована через звернення до REST API, наданого backend-ом, за допомогою стандартних HTTP-запитів. У Flutter-застосунку для виконання цих запитів використовується пакет http. Особливу увагу приділено процесу автентифікації користувача, оскільки доступ до більшості функцій системи має бути захищеним. Управління токенами доступу та інформацією про роль користувача є критично важливим для забезпечення як безпеки, так і коректного надання відповідних функцій кожному типу користувача.

Для інкапсуляції всієї логіки, пов'язаної з автентифікацією, було створено спеціальний сервісний клас AuthService. Його завданням є не тільки виконання запиту на вхід до системи, але й обробка відповіді сервера, безпечне збереження отриманого токена доступу та іншої важливої інформації про сесію користувача, а також надання методів для доступу до цих даних іншим компонентам застосунку.

Ключовим методом класу AuthService є функція login, яка приймає електронну пошту та пароль користувача і намагається авторизувати його в системі.

```
class AuthService {
  static Future<(Token, String)?> login(String email, String
password) async {
    try {
      final url = Uri.parse('${ApiConfig.baseUrl}/profile/login');
      final headers = {'Content-Type': 'application/json'};
      final body = jsonEncode({'email': email, 'password':
password});

      final response = await http.post(url, headers: headers, body:
body);

      if (response.statusCode == 200) {
```

```
            final jsonData = jsonDecode(response.body);
            final accessToken = jsonData['access_token'];
            final role = jsonData['role'] as String;
            final userId = (jsonData['user'] != null &&
jsonData['user']['id'] != null)
                            ? jsonData['user']['id'] as String
                            : null;

            if (accessToken != null && userId != null) {
              final token = Token(accessToken: accessToken, tokenType:
"Bearer");
              await _saveToken(token.accessToken);
              await _saveRole(role);
              await _saveUserId(userId);
              return (token, role);
            }
          } else {
            print('Login failed with status ${response.statusCode}');
          }
          return null;
        } catch (e) {
          print('Error in AuthService.login: $e');
          return null;
        }
      }
      // ... решта методів класу ...
```

У наведеному фрагменті метод login асинхронно відправляє POST-запит на серверний ендпоінт /profile/login. Для цього використовується базовий URL з ApiConfig та передані email і password, які кодуються у формат JSON для тіла запиту. Заголовок Content-Type: application/json інформує сервер про формат переданих даних.

Після отримання відповіді, її статус-код перевіряється. Якщо статус 200 (OK), тіло відповіді (також у форматі JSON) декодується [11]. З отриманих даних витягуються токен доступу (access_token), роль користувача (role) та його

унікальний ідентифікатор (userId). Була додана більш безпечна перевірка наявності шляху jsonData['user']['id'] перед спробою доступу, щоб уникнути помилок, якщо структура відповіді може варіюватися. Якщо всі необхідні дані присутні, створюється об'єкт Token, і токен, роль та ID користувача зберігаються локально за допомогою приватних методів, описаних нижче. У разі успіху метод повертає кортеж з об'єктом Token та рядком ролі. В іншому випадку (невдалий статус-код або помилка під час виконання) повертається null.

Для збереження даних сесії між запусками застосунку використовується пакет SharedPreferences. Це простий механізм для зберігання невеликих обсягів даних типу "ключ-значення".

```
// auth_service.dart (фрагмент методів для SharedPreferences)
  static Future<void> _saveToken(String token) async {
    final prefs = await SharedPreferences.getInstance();
    await prefs.setString('access_token', token);
  }

  static Future<String?> getToken() async {
    final prefs = await SharedPreferences.getInstance();
    return prefs.getString('access_token');
  }

  static Future<void> _saveRole(String role) async {
    final prefs = await SharedPreferences.getInstance();
    await prefs.setString('user_role', role);
  }

  static Future<String?> getRole() async {
    final prefs = await SharedPreferences.getInstance();
    return prefs.getString('user_role');
  }

  static Future<void> _saveUserId(String userId) async {
    final prefs = await SharedPreferences.getInstance();
    await prefs.setString('user_id', userId);
```

```
}

static Future<String?> getUserId() async {
  final prefs = await SharedPreferences.getInstance();
  return prefs.getString('user_id');
}
```

Приватні методи _saveToken, _saveRole, _saveUserId отримують екземпляр SharedPreferences і зберігають відповідні значення (токен, роль, ID користувача) за унікальними ключами ('access_token', 'user_role', 'user_id'). Публічні статичні методи getToken, getRole, getUserId дозволяють іншим частинам застосунку асинхронно отримати ці збережені значення.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Забезпечення якості та надійності розробленого мобільного застосунку для координації волонтерської допомоги є критично важливим завданням. Тестування розглядалося як невід'ємний етап розробки, спрямований на перевірку відповідності реалізованого функціоналу визначеним вимогам, а також на забезпечення стабільної та коректної роботи застосунку в різних умовах використання. Основна мета тестування полягала у виявленні та усуненні можливих дефектів, а також у підтвердженні того, що застосунок є зручним та інтуїтивно зрозумілим для цільових користувачів – волонтерів та отримувачів допомоги.

Процес тестування охоплював перевірку всіх ключових аспектів застосунку. Особливу увагу було приділено коректності роботи інтерфейсу користувача: перевірялася реакція елементів управління на дії користувача (натискання кнопок, заповнення форм), правильність відображення даних на екранах (списки зборів та вимог, деталі профілів, прогрес зборів). Були розроблені сценарії використання, що імітують реальні дії користувачів у системі, з метою перевірки наскрізної функціональності – від процесу автентифікації, створення та перегляду зборів і вимог, до управління профілем. Проводилося функціональне тестування для кожного реалізованого програмного модуля та функції, щоб переконатися, що вони працюють згідно з технічним завданням та задовільняють потреби користувачів.

Окрім функціональної коректності, важливим аспектом тестування була оцінка зручності використання (юзабіліті) мобільного застосунку. Перевірялася логічність та простота навігації між екранами, зрозумілість назв елементів інтерфейсу та загальне враження від взаємодії з програмою.

Особливий акцент у тестуванні було зроблено на коректності взаємодії мобільного клієнта з серверною частиною (backend API). Перевірялася правильність відправлення запитів, обробки відповідей від сервера (включаючи успішні відповіді та різні коди помилок), а також коректність відображення отриманих даних.

У таблиці 5.1 наведено приклад детального тест-кейсу, який охоплює ключові сценарії роботи з мобільним застосунком: вхід в систему як волонтер, перегляд дашборду, створення нового збору та створення нової вимоги. Цей тест-кейс демонструє підхід до перевірки основної функціональності.

Таблиця 5.1 – Тест-кейс №1

Інформація про тест-кейс			
Ідентифікатор тесту:	Тест-кейс №1		
Опис функції:	Перевірка основних сценаріїв: вхід, перегляд дашборду, створення збору та вимоги.		
Власник тесту:	Голя Станіслав Володимирович		
Дата створення:	14.05.2025		
Мета тесту:	Перевірити коректність роботи ключових функцій застосунку		
Передумова			
№	Опис випадку	Очікуваний результат	Висновок
1	Застосунок встановлено на тестовому пристрої. Серверна частина (API) доступна.	Користувач має доступ до застосунку, та може його відкрити.	Пройдено
2	Існують тестові облікові записи в системі.	Користувач може увійти в тестові записи, використавши надані йому пароль та логін.	Пройдено
Вхід у систему як волонтер			
№	Опис випадку	Очікуваний результат	Висновок
1	Відкрити мобільний застосунок.	Відображається екран входу.	Пройдено
2	На екрані входу ввести коректні email та пароль волонтера.	Поля заповнюються введеними даними.	Пройдено
3	Натиснути кнопку "Увійти".	Відбувається запит до сервера. У разі успіху, користувач перенаправлений на дашборд волонтера.	Пройдено

Продовження таблиці 5.1

Перегляд дашборду волонтера та створення збору			
№	Опис випадку	Очікуваний результат	Висновок
1	Переглянути дашборд волонтера.	Відображаються секції "Мої збори" та "Останні вимоги".	Пройдено
2	Перейти на сторінку вимоги.	Відображається детальне представлення вимоги.	Пройдено
3	Заповнити поля форми: Назва збору, Опис, Посилання на Монобанку.	Дані коректно вводяться у відповідні поля.	Пройдено
4	Завантажити зображення для збору.	Зображення успішно завантажується.	Пройдено
5	Натиснути кнопку "Створити збір"	Збір успішно створено на сервері, відображається повідомлення про успіх.	Пройдено
Вхід у систему як отримувач			
№	Опис випадку	Очікуваний результат	Висновок
1	Відкрити мобільний застосунок.	Відображається екран входу.	Пройдено
2	На екрані входу ввести коректні email та пароль отримувача.	Поля заповнюються введеними даними.	Пройдено
3	Натиснути кнопку "Увійти".	Відбувається запит до сервера. У разі успіху, користувач перенаправлений на дашборд отримувача.	Пройдено
Перегляд дашборду отримувача та створення вимоги			
№	Опис випадку	Очікуваний результат	Висновок
1	Переглянути дашборд отримувача.	Відображаються секції "Мої збори" та "Мої вимоги".	Пройдено
2	Перейти до розділу створення нової вимоги.	Відкривається форма створення вимоги.	Пройдено
3	Заповнити поле "Назва вимоги".	Дані коректно вводяться.	Пройдено
4	Додати декілька предметів до вимоги: вказати назву, кількість та категорію.	Предмети успішно додаються до списку у формі.	Пройдено
5	Вказати пріоритет вимоги.	Пріоритет успішно обрано.	Пройдено
6	Натиснути кнопку "Створити вимогу".	Вимога успішно створена на сервері, відображається повідомлення про успіх.	Пройдено

Кінець таблиці 5.1

Результати тестування		
Тестувальник: Голя Станіслав Володимирович	Дата прогону тесту: 14.05.2025	Результат тесту (P/F/B): ПРОЙДЕНО (P)

Тестування функціональності розробленого мобільного застосунку для волонтерів проводилося систематично на різних етапах розробки, починаючи від тестування окремих віджетів та модулів до наскрізного тестування повних сценаріїв взаємодії користувача з системою. Окрім спеціально розроблених тестових випадків, подібних до наведеного в таблиці 5.1, велика увага приділялася перевірці реакції застосунку на нетипові дії користувача, валідації вхідних даних та обробці помилок, що надходять від сервера. Такий комплексний підхід до тестування дозволяє забезпечити високий рівень якості, стабільності та надійності мобільного застосунку, що є критично важливим для інструменту, який буде використовуватися в такій відповідальній сфері, як волонтерська допомога.

ВИСНОВКИ

У результаті виконання передатестаційної практики було досягнуто значного прогресу в розробці клієнтської частини (мобільного застосунку) програмної системи, призначеної для координації та управління процесами волонтерської допомоги. Початковим етапом роботи став ґрунтовний аналіз предметної галузі та існуючих цифрових рішень у сфері волонтерства та благодійності. Цей аналіз дозволив виявити ключові виклики, з якими стикаються волонтери та координатори: проблеми зі структуризацією запитів на допомогу, відсутність ефективних інструментів для пріоритезації потреб, низька прозорість процесів для донорів та обмежені можливості для мобільної роботи. Виявлені проблеми та обмеження існуючих платформ (як-от орієнтація лише на фінансування або логістику) стали фундаментом для формулювання чітких та обґрунтованих вимог до мобільного застосунку, який мав би стати комплексним інструментом для волонтерів.

На наступному етапі були детально сформульовані функціональні та нефункціональні вимоги до розроблюваного мобільного застосунку. Функціональні вимоги охоплювали ключові сценарії використання для основних ролей користувачів (волонтер, отримувач допомоги), включаючи автентифікацію, створення та управління зборами, формування та відстеження структурованих вимог на допомогу, керування профілями, а також інтеграцію з зовнішніми сервісами, зокрема з API Монобанку для відображення прогресу зборів. Нефункціональні вимоги визначили критерії якості застосунку, такі як висока продуктивність, надійність, безпека передачі та зберігання даних, зручність використання (юзабіліті) та сумісність з основними мобільними платформами. Ці вимоги лягли в основу подальших етапів проєктування архітектури та інтерфейсу застосунку.

Під час етапу архітектури та проєктування мобільного застосунку було прийнято рішення щодо використання фреймворку Flutter та мови Dart, що забезпечує кросплатформеність та високу продуктивність. Було спроєктовано

трирівневу архітектуру, яка забезпечує чіткий поділ відповідальностей та сприяє тестованості й підтримуваності коду. Значну увагу було приділено проєктуванню користувацького інтерфейсу (UI/UX), орієнтованого на інтуїтивність та ефективність виконання завдань волонтерами. Були створені прототипи ключових екранів (профіль, головна сторінка, списки зборів/вимог), що визначають візуальний стиль та навігаційну логіку застосунку. Вибір технологій та архітектурних рішень дозволить створити надійний, гнучкий та зручний мобільний інструмент для волонтерської спільноти, здатний ефективно вирішувати поставлені завдання та забезпечувати високий рівень користувацького досвіду.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Dobro.ua – Українська Біржа Благодійності [Електронний ресурс]. – URL: https://dobro.ua/ (дата звернення: 08.05.2025).

2. Flutter – Official Documentation [Електронний ресурс]. – URL: https://flutter.dev/docs (дата звернення: 11.05.2025).

3. Dart Programming Language – Official Website [Електронний ресурс]. – URL: https://dart.dev/ (дата звернення: 11.05.2025).

4. Effective Dart – Style Guide [Електронний ресурс]. – URL: https://dart.dev/guides/language/effective-dart/style (дата звернення: 11.05.2025).

5. Material Design – Guidelines by Google [Електронний ресурс]. – URL: https://material.io/design (дата звернення: 11.05.2025).

6. Cupertino (iOS-style) widgets – Flutter Documentation [Електронний ресурс]. – URL: https://docs.flutter.dev/ui/widgets/cupertino (дата звернення: 11.05.2025).

7. GoRouter – A declarative routing package for Flutter [Електронний ресурс]. – URL: https://pub.dev/packages/go_router (дата звернення: 12.05.2025).

8. Figma – Collaborative interface design tool [Електронний ресурс]. – URL: https://www.figma.com/ (дата звернення: 12.05.2025).

9. API Monobank – Офіційна документація [Електронний ресурс]. – URL: https://api.monobank.ua/docs/ (дата звернення: 12.05.2025).

10. State management approaches in Flutter [Електронний ресурс]. – URL: https://docs.flutter.dev/data-and-backend/state-mgmt/options (дата звернення: 12.05.2025).

11. Бондарєв В., Черепанова Ю. Комп’ютерна симуляція небесної механіки з навчальними цілями // Інформаційні системи та технології : матеріали 13-ї Міжнар. наук.-техн. конф. (Харків, 26–28 листоп. 2024 р.). Харків : ХНУРЕ, 2024. URL: https://ist-conf-nure.com.ua/ (дата звернення: 12.05.2025).

12. Github.com – 2025_B_PI_PZPI-21-8_Holia_S_V [Електронний ресурс]. – URL: https://github.com/NureHoliaStanislav/2025_B_PI_PZPI-21-8_Holia_S_V