Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)
факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи ступеня бакалавра

(бакалавра, спеціаліста, магістра)

Студента Гнипа Олександра Романовича
(ПІБ)

академічної групи 126-20-1
(шифр)

спеціальності 126 «Інформаційні системи та технології»
(код і назва спеціальності)

за освітньо-професійною програмою «Інформаційні системи та технології»
(офіційна назва)

на тему Розробка мобільного застосунку інформаційної системи "Каса взаємодопомоги"
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікавційної роботи	проф. Коротенко Г.М.			
розділів:				

Рецензент	доц. Ширін А.Л.			

Нормоконтролер	проф. Коротенко Г.М.			

Дніпро
2024

ЗАТВЕРДЖЕНО:
завідувач кафедри
____інформаційних технологій____
___та комп'ютерної інженерії____
(повна назва)

_____ __Гнатушенко В.В._____
(підпис) (прізвище, ініціали)

«_____»_____2024_року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня _____бакалавр_____
(бакалавра, спеціаліста, магістра

студенту _____Гнипу О.Р._____ **академічної групи** ____126-20-1____
(прізвище та ініціали) (шифр)

спеціальності _____126 « Інформаційні системи та технології »_____

за освітньою-професійною програмою _____

«Інформаційні системи та технології»

на тему _____Розробка мобільного застосунку інформаційної системи _____

_____"Каса взаємодопомоги"_____

затверджену наказом ректора НТУ «Дніпровська політехніка» від 23.05.2024 р. № 469-с

Розділ	Зміст	Термін виконання
Розділ 1	Ознайомлення з матеріалами сайтів, літературою та виконання пошуку технологій і формування відповідних рішень	01.02.2024 – 30.03.2024
Розділ 2	Реалізація проектних рішень	01.04.2024 – 20.06.2024

Завдання видано _____ _____Коротенко Г.М._____
(підпис керівника) (прізвище, ініціали)

Дата видачі _____01.02.2024 р._____

Дата подання до екзаменаційної комісії ____02.07.2024 р.____

Прийнято до виконання _____ _____ Гнип О.Р._____
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 81 сторінка, 34 рисунків, 8 таблиць, 3 додатка, 20 джерел.

Об’єкт розробки: мобільний застосунок інформаційної системи «Каса взаємодопомоги».

Мета кваліфікаційної роботи: розробка програмного забезпечення мобільного застосунку інформаційної системи «Каса взаємодопомоги».

У першому розділі звертається увага на актуальність проблеми, проводиться аналіз існуючих аналогів мобільних застосунків для взаємодопомоги, і визначаються мета та задачі кваліфікаційної роботи.

Другий розділ присвячений проектуванню мобільного застосунку інформаційної системи «Каса взаємодопомоги», де наведено опис його архітектури, алгоритму роботи, та функціональності. Крім того, розглядаються технології, які будуть використовуватись для реалізації проекту.

У наступному розділі детально описується реалізація застосунку. Виконується розробка самого мобільного застосунку «Каси взаємодопомоги» з використанням інструментів фреймворків Flutter та Cubit, і мови програмування Dart. Також описується порядок взаємодії з графічним інтерфейсом користувача.

Результатом проведеної роботи є створений мобільний застосунок інформаційної системи «Каса взаємодопомоги». Цей застосунок реалізує особистий кабінет учасника фонду взаємодопомоги, та надає засоби для оплати внесків, керування подачею заяв та отриманням грошової допомоги.

Ключові слова: МОБІЛЬНИЙ, КАСА ВЗАЄМОДОПОМОГИ, ЗАСТОСУНОК, ЗАЯВКА, ВНЕСОК, ФРЕЙМВОРК, FLUTTER, CUBIT, DART.

ABSTRACT

Explanatory note: 81 pages, 34 fig., 8 tab, 3 appendices, 20 sources.

The object of development: mobile application of the information system "Mutual Assistance Cash Desk".

The purpose of the qualification work: development of software for mobile application of the information system "Mutual Assistance Cash Desk".

The first section draws attention to the relevance of the problem, analyzes existing analogs of mobile applications for mutual aid, and defines the purpose and tasks of the qualification work.

The second section is devoted to the design of the mobile application of the information system "Mutual Assistance Cash Desk", which describes its architecture, work algorithm, and functionality. In addition, the technologies that will be used for the implementation of the project are considered.

The next section describes the implementation of the application in detail. The development of the mobile application of the "Mutual Assistance Cash Desk" using the tools of the Flutter framework, the Cubit library and the Dart programming language is in progress. The order of interaction with the graphical user interface is also described.

The result of the work is the created mobile application of the information system "Mutual Assistance Cash Desk". This application implements the personal account of the member of the mutual aid fund, and provides means for paying contributions, managing the submission of applications and receiving financial assistance.

Keywords: MOBILE, MUTUAL ASSISTANCE CASH DESK, APPLICATION, CONTRIBUTION, FRAMEWORK, FLUTTER, CUBIT, DART.

ЗМІСТ

ВСТУП

Взаємодопомога означає співпрацю для служіння членам громади [1]. Серед війни, пандемії [2], економічної кризи та збільшення кліматичних катастроф потреба в мережах взаємодопомоги значно зросла. Ці кризи викривають системну нерівність у суспільстві, включно з потребами у охороні здоров'я та гуманітарній допомозі. І взаємодопомога завжди відігравала роль у заповненні цих прогалин.

Групи взаємодопомоги вирізняються своїм прагненням згладити ієрархію, шукаючи колективного консенсусу для прийняття рішень серед учасників, а не поміщаючи керівництво в закриту виконавчу команду. Завдяки спільному прийняттю рішень усі учасники отримують повноваження вносити зміни та брати на себе відповідальність за групу [3].

У групах взаємодопомоги ресурси зазвичай розподіляються беззастережно, що відрізняє цю модель від благодійності, де часто встановлюються умови для отримання доступу до допомоги, такі, як перевірка матеріального становища чи надання грантів. Ці групи часто виходять за межі обміну матеріалами чи послугами та створюються як форма політичної участі [3], у якій люди беруть на себе відповідальність за турботу один про одного та зміну політичних умов.

У рамках даної кваліфікаційної роботи було поставлено завдання розробити мобільний застосунок інформаційної системи «Каса взаємодопомоги». «Каса взаємодопомоги» є інформаційною системою, яка автоматизує процес функціонування спілки взаємодопомоги.

Цей проект має на меті розробити зручний та ефективний інструментарій для реалізації особистого кабінету учасника «Каси взаємодопомоги». З його допомогою буде здійснюватися регістрація користувачів для вступу у спілку, подача заяв, сплата внесків та завантаження файлів на сервер системи.

У процесі розробки мобільного застосунку інформаційної системи буде використана сучасна технологія програмування, що дозволить створити зручний та функціональний інструмент для забезпечення функціонування каси взаємодопомоги. Також будуть застосовані методи авторизації, зокрема, через інтеграцію з «Дією», і оплати за допомогою платіжної системи LiqPay.

Основною метою цієї розробки є створення зручного та інтуїтивно зрозумілого інтерфейсу, який дозволить користувачам ефективно взаємодіяти з фондом взаємодопомоги у якості його учасників.

Розроблений застосунок буде враховувати сучасні вимоги до дизайну та функціональності, що забезпечить ефективну та коректну роботу та приємний досвід користувачів. Очікується, що розроблений мобільний застосунок інформаційної системи «Каса взаємодопомоги» покращить доступність та автономність дій учасників руху солідарної фінансової підтримки, сприяючи розвитку благодійних ініціатив членів громади.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність проблеми

Взаємодопомога останніми роками стала модним словом серед активістів, громадських організаторів та науковців. Після пандемії COVID-19 громади в усьому світі мобілізувалися, щоб задовольнити потреби своїх сусідів, і термін «взаємодопомога» став загальноприйнятим. Проте сьогодні є прогалини в розумінні того, як насправді виглядає взаємодопомога на практиці. Коли існуючі системи не можуть задовольнити основні потреби людини під час кризи, взаємна допомога забезпечує ці потреби громади [5].

Фундаментальна робота Діна Спейда «Взаємодопомога: розбудова солідарності під час цієї кризи (і наступної)» визначає взаємодопомогу як «працю на виживання», яку виконують звичайні люди в надзвичайних обставинах, коли урядова політика неадекватно реагує на кризові ситуації та навіть посилює структурну нерівність [4]. Взаємна допомога полягає в мобілізації спільнот для усунення спільної несправедливості та спільної роботи для спільного пошуку рішень, які не покладаються на одну особу, корпорацію чи державну установу.

Оскільки взаємодопомога спонукає організаторів переосмислювати спосіб роботи традиційних соціальних служб і протистояти йому, багато груп взаємодопомоги відчувають труднощі з розробкою моделей керівництва, прийняття рішень і фінансування [5]. Групи взаємодопомоги часто є найбільш успішними, коли багато людей збираються разом і приймають рішення за весь колектив, але це набагато легше сказати, ніж зробити, особливо коли більшість звикли до ієрархії та кар’єризму. Дін Спейд окреслює деякі найкращі практики для організаторів [4], підкреслюючи, що процес є таким же важливим напрямком, як і мета допомоги людям.

По-перше, більшість груп взаємодопомоги дотримуватимуться горизонтальної структури прийняття рішень. Це означає, що часто немає

лідерів, директорів, генеральних директорів тощо. Натомість усі члени групи беруть участь у прийнятті рішень. Замість того, щоб покладатися на зарплату чи терміновість просування роботи вперед, добре структуровані групи взаємодопомоги розглядають, що внутрішньо спонукає волонтерів залишатися залученими. Що змушує їх продовжувати діяти? Дін Спейд стверджує, що участь у роботі та внесок у рішення групи змушує учасників відчувати співвласність, що запобігає вигоранню та плинності кадрів [4]. Він рекомендує приймати рішення на основі консенсусу, що дозволяє кожному мати право голосу в будь-якому рішенні, які на них впливають.

Фінансова підтримка роботи групи взаємодопомоги також може виглядати зовсім не так, як традиційні благодійні організації, які часто покладаються переважно на масове краудфандинг. Отримання статусу неприбуткової організації, отримання фінансування та наймання оплачуваного персоналу не обов’язково означає, що організація не є групою взаємодопомоги. Несправедливість може виникнути, коли деякі члени отримують гроші, а деякі є волонтерами. Раптом можна очікувати, що оплачувані учасники зроблять для групи більше, ніж неоплачувані колеги, і прийняття рішень може почати спотворюватись на користь думок оплачуваних учасників. Крім того, навички, необхідні для розуміння юридичних і канцелярських застережень, пов’язаних з управлінням грантовим фінансуванням, можуть зробити ці оплачувані посади доступними лише для більш привілейованих учасників. Нарешті, грантове фінансування часто супроводжується певними обмеженнями та застереженнями донорів про те, як можна використовувати гроші, а також про те, які політичні питання організація може публічно вирішувати.

Спільнота сама по собі є найважливішим компонентом взаємодопомоги, як і розширення можливостей зазначеної спільноти мати свободу дій у своєму колективному виживанні та добробуті. Взаємодопомога полягає в мобілізації спільнот для усунення спільної несправедливості та спільної роботи для спільного пошуку рішень [5].

Для подолання деяких з окреслених проблеми організацій взаємодопомоги можуть бути використані сучасні інформаційні системи, шляхом автоматизації процесів управління фондом взаємодопомоги, обробки звернень учасників та здійснення платежів. Розроблений застосунок повинен надавати зручний та ефективний функціонал для всіх осіб, які залучені до каси взаємодопомоги у якості учасників. Мобільний застосунок інформаційної системи, що розробляється у даній кваліфікаційній роботі, спрямований на обслуговування учасників благодійних програм фонду, і призначений спеціально для конкретної організації "Каса взаємодопомоги".

1.2 Аналіз аналогів мобільних застосунків фондів взаємодопомоги

Перед початком проектування застосунку необхідно провести пошук і аналіз існуючих аналогічних програмних продуктів, що належать до категорії мобільних застосунків інформаційних систем, фондів або сервісів взаємодопомоги. У результаті було розглянуто наступні програмні продукти:

1. Мобільний застосунок «Лікарняна каса» [6].

Додаток призначений для підтримки функціонування каси взаємодопомоги благодійної організації «Лікарняна каса Житомирської області».

Функціонал застосунку включає в себе:

- посвідчення члена благодійної організації «Лікарняна каса Житомирської області» в медичних закладах (є номер посвідчення, ПІБ, інформація про дітей, стан сплати внесків);
- можливість завантаження документів;
- нагадуваня про оплату;
- перевірку стану сплати внесків (рис. 1.1);

Рис. 1.1. Перевірка стану сплати внесків

- інформацію по філіях благодійної організації "Лікарняна каса Житомирської області" (рис. 1.2);

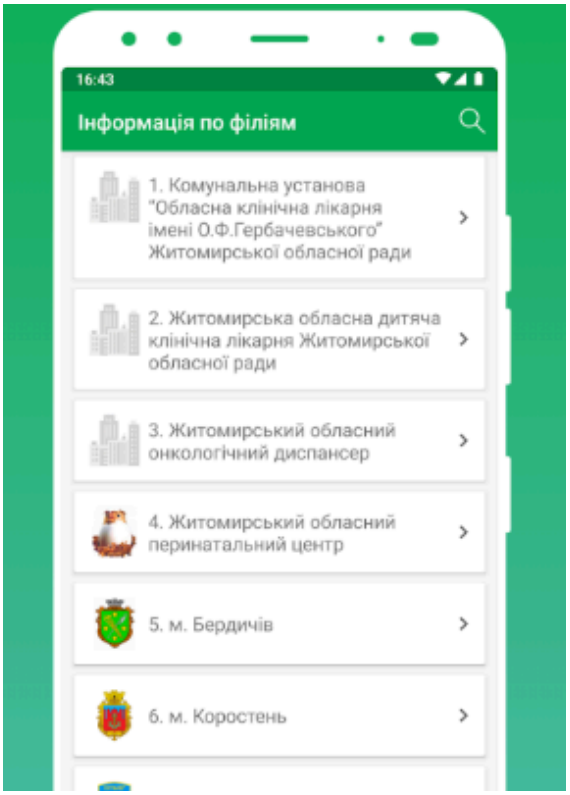

Рис. 1.2. Інформація про філії

- підтримка декількох користувачів (рис. 1.3).

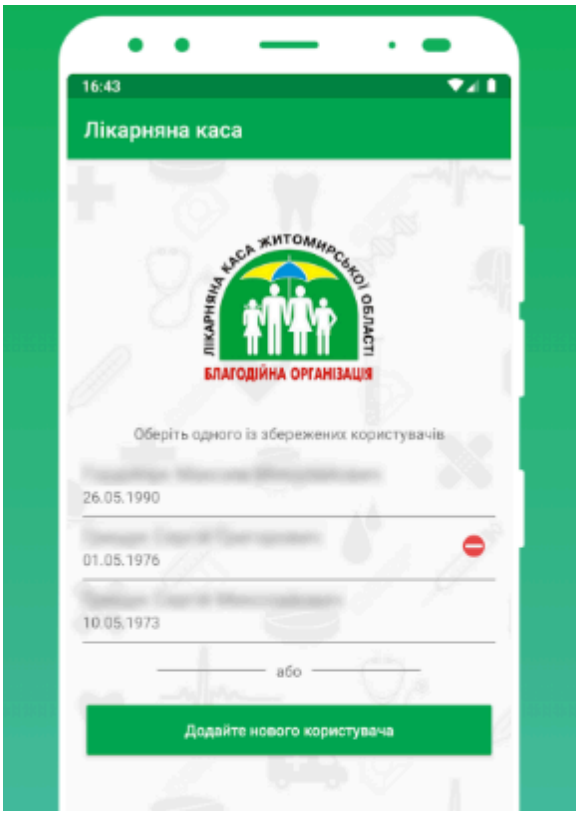

Рис. 1.3. Вибір користувача

Недоліки:

Застосунок призначений лише для обслуговування конкретної організації («Лікарняна каса Житомирської області»), не є універсальним, і тому не може бути використаним для інших фондів взаємодопомоги.

2. Мобільний застосунок «Dopomoha» [7]

Це сервіс взаємопомічі, призначений для координації роботи волонтерів, побудований по принципу «рівний-рівному». Допомога розділяється на три категорії: матеріальна, фізична та моральна. Можна просити про послуги, гуманітарну допомогу, транспортування.

Функціонал застосунку включає в себе:

- Створення заявки на отримання допомоги (рис. 1.4);

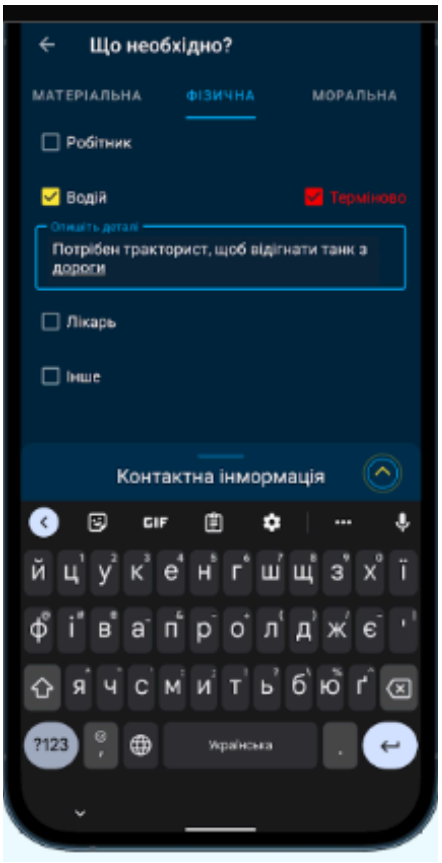

Рис. 1.4. Створення заявки

- Перегляд необроблених заявок (рис. 1.5);

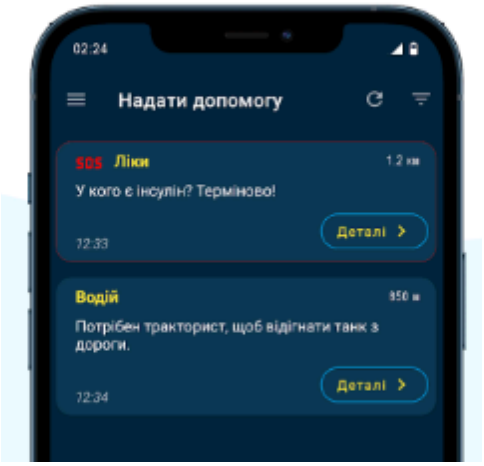

Рис. 1.5. Перегляд заявок

- Відгук на заявку;

- Відображення географічне розташування заявок на отримання допомоги на карті (рис. 1.6).

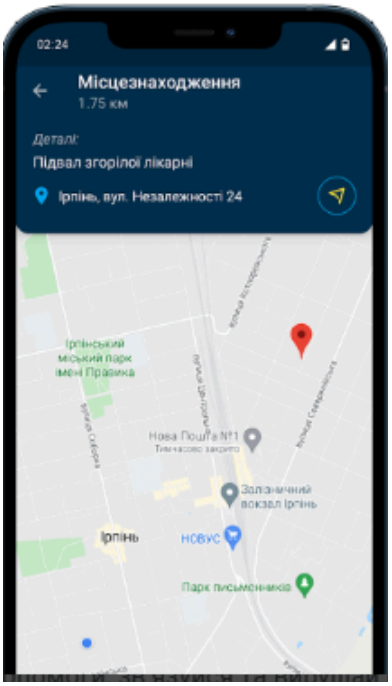

Рис. 1.6. Пошук заявок на мапі

Після відгуку на заявку з’являється можливість спілкування в чаті з учасником, який запропонував допомогу (рис. 1.7). Також є функція відстеження змін статусу заяви.

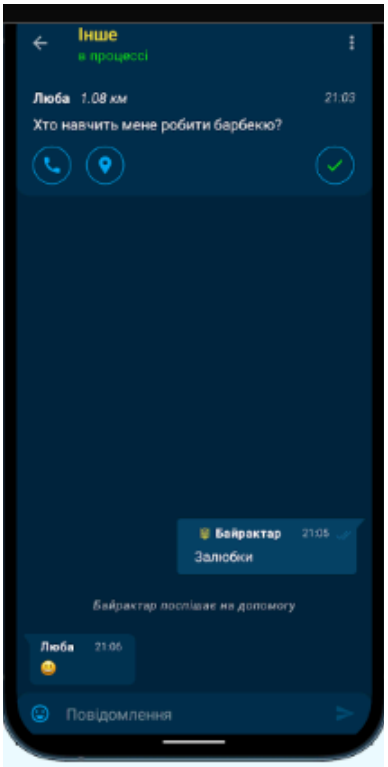

Рис. 1.7. Відгук на заявку

Недоліки:

- Мінімалістичний та не дуже зрозумілий дизайн;
- Застосунок не підтримує винонання фінансових операцій, таких як переказ коштів, перевірка оплат, відстеження руху коштів по проектах.

3. Мобільний застосунок «Бандеролька» [8]

Застосунок для надання волонтерської допомоги, дуже схожий на попередній описаний програмний продукт, проте з більш якісним та привабливим дизайном.

Функції:

- Реєстрація (за номером телефону);
- Подача завок на отримання допомоги (рис. 1.8);

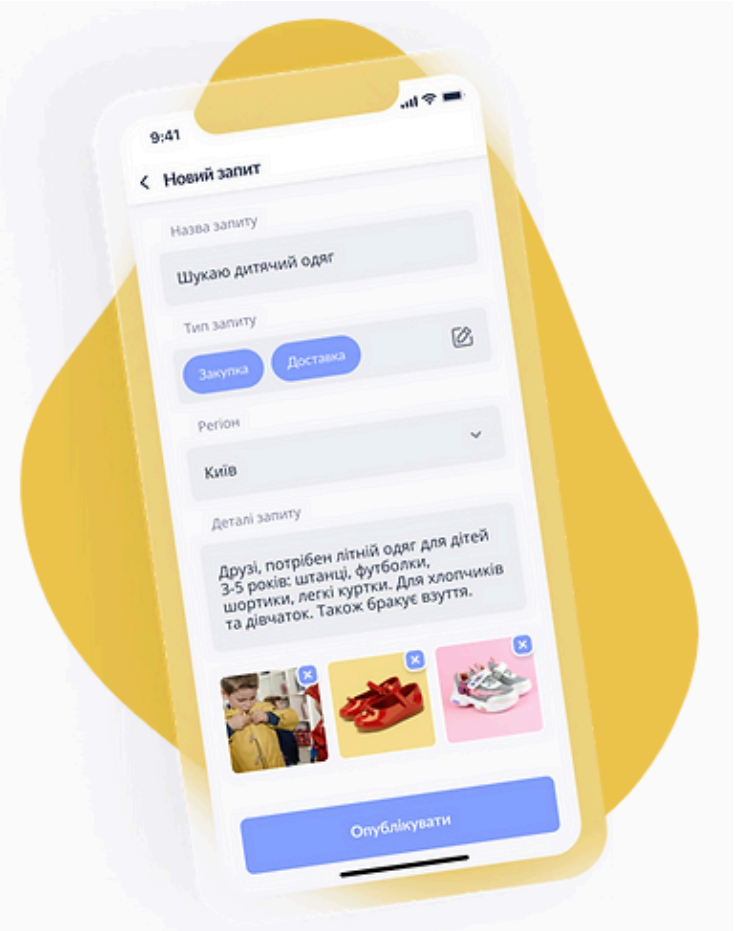

Рис. 1.8. Процес створення заявки

- Категоризація заявок за призначенням допомоги (продукти, гігієна, транспортування тощо), за територіальним знаходженням;
- Перегляд актуальних запитів у вигляді стрічки (рис. 1.9);
- Відгук на заявку.

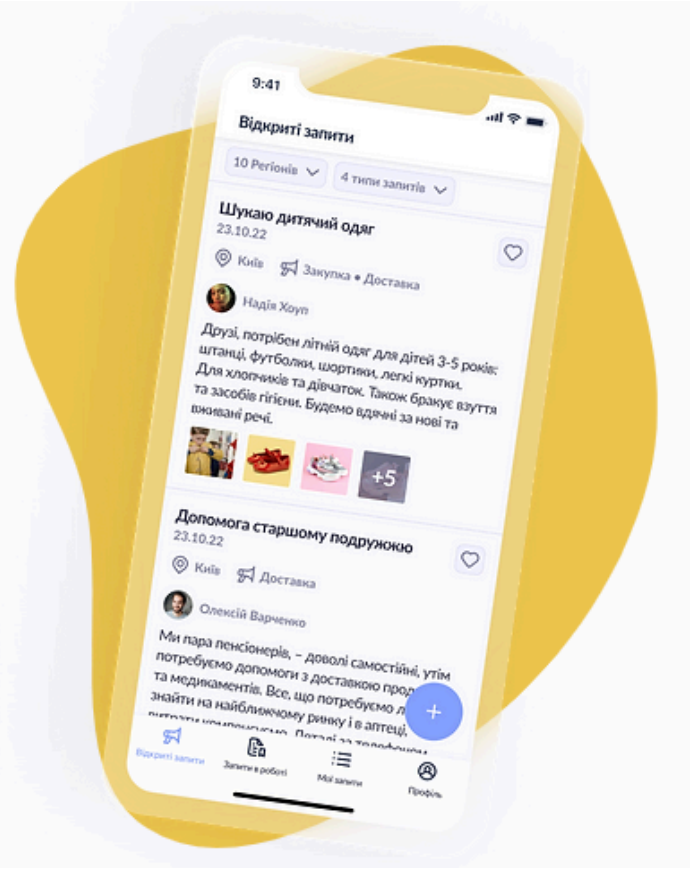

Рис. 1.9. Перегляд стрічки заявок

Недоліки:

- Неможливість створення комплексних програм взаємодопомоги;
- Відсутність функціоналу для обслуговування фінансових операцій;
- Відсутність звітності про результати виконання запитів.

4. Застосунок «Lepta» [9]

Даний безоплатний застосунок для взаємодопомоги був запущений Міністерством цифрової трансформації України. Особливість застосунку полягає в тому, що застосунок працює так, що в ньому немає посередників, а ті кому потрібна допомога, напряму зв`язуються з тими хто може надати таку допомогу.

Функціональні можливості Lepta:

- подання запитів про допомогу;

- відгук на запити;
- надання точного звіту про надану підтримку;
- відображення активних запитів у форматі стрічки, та на карті (рис. 1.10);

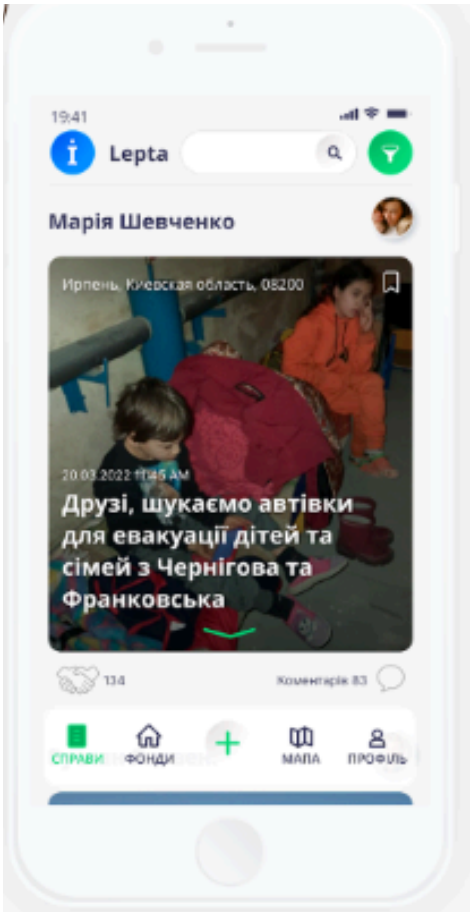

Рис. 1.10. Запит про допомогу

- картографічноий сервіс для відображення запитів, розташований найближче до користувача;

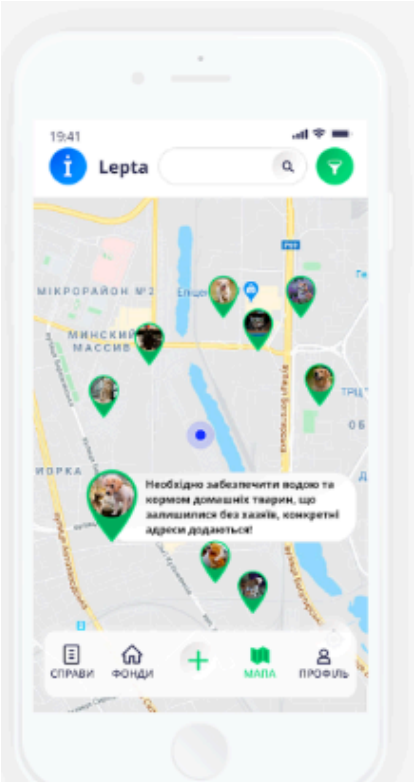

Рис. 1.11. Картографічний сервіс

- створення благодійних ініціатив;
- пошук зниклих людей;

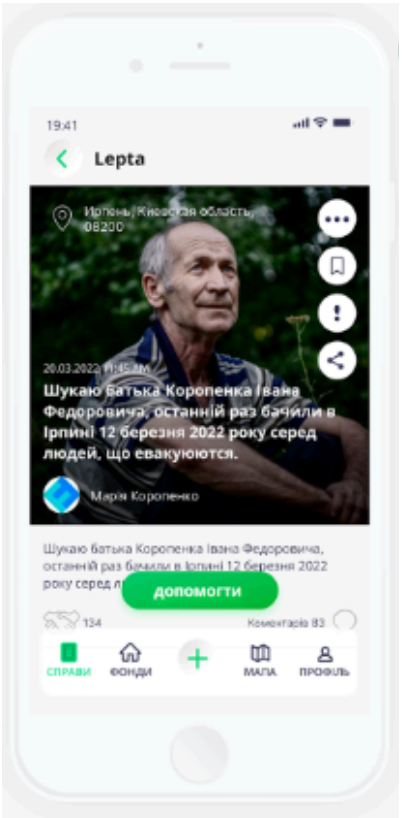

Рис. 1.12. Пошук зниклих людей

− можливість задонатити на фонди підтримки ЗСУ та людей, що постраждали внаслідок російської агресії.

Недоліки:

- орієнтованість на адресну допомогу;
- відсутність можливості робити регулярні грошові внески;
- відсутність керування ініціативними проектами (функція скоріше декларативна).

В результаті проведеного огляду можна зробити висновок, що жоден з застосунків не реалізує у повному обсязі весь інструментарій для учасника спілки взаємодопомоги. Тому актуальною задачею є розробка мобільного застосунку інформаційної системи «Каса взаємодопомоги», який би забезпечував зручний та функціональний інтерфейс для користування всіма можливостями «Каси взаємодопомоги».

1.3 Мета та задачі мобільного застосунку інформаційної системи «Каса взаємодопомоги»

Мета даної кваліфікаційної роботи полягає у розробці мобільного застосунку інформаційної системи «Каса взаємодопомоги». Цей застосунок реалізує зручний та функціональний інструмент для забезпечення функціонування каси взаємодопомоги. Він надасть можливість здійснювати регістрація користувачів для вступу у спілку, подача заяв, сплата внесків та завантаження файлів на сервер системи.

Також, враховуючи основну мету цієї програми, мобільний застосунок «Каси взаємодопомоги» буде використовуватись у якості зручного та інтуїтивно зрозумілого інтерфейсу, який дозволить користувачам ефективно взаємодіяти з фондом взаємодопомоги у якості його учасників.

Розроблений застосунок має забезпечувати виконання наступних функцій:

- подання запиту на регістрацію нових членів фонду;
- авторизація користувачів;
- перегляд благодійних програм;
- перегляд тарифів регулярних внесків;
- реєстрація на участь у благодійній програмі;
- перегляд історії здійснених платежів;
- перегляд історії поданих заяв;
- подання заяви на отримання допомоги;
- завантаження документів на сервер інформаційної системи;
- перевірка статусу заяви;
- здійснення фінансових внесків;
- редагування профіля користувача;
- отримання нагадувань та сповіщень про регулярні або термінові події.

Мобільний застосунок «Каси взаємодопомоги» має бути розроблений відповідно до сучасних вимог до дизайну та функціональності. Програмне забезпечення повинно підтримувати виконання таких дій:

- обмін REST запитами з сервером інформаційної системи та сервером «Дії»;
- обмін HTTP-запитами з сервісом liqPay;
- авторизація користувачів через СМС;
- авторизація через «Дію»;
- завантаження графічних файлів.

Для досягнення поставленої мети і відповідно до вимог, необхідно виконати наступні завдання:

- здійснити експертний аналіз предметної області;
- вибрати необхідні інструменти для створення застосунку;
- розробити артітектуру програмного продукту;
- розробити відповідний функціонал застосунку;
- провести тестування застосунку.

Щоб забезпечити ефективну реалізацію проекту та виконання всіх вимог, необхідно мати детальне технічне завдання. Це є необхідною умовою для успішної реалізації проекту та досягнення поставлених цілей.

1.4 Висновки до розділу 1

У першому розділі кваліфікаційної роботи бакалавра було розглянуто структуру та особливості функціонування фонду взаємодопомоги, а також виконано огляд мобільних застосунків, які можуть виступати у якості мобільного компоненту інформаційної системи фонду.

В результаті огляду були виявлені наступні недоліки:

– неінтуїтивний дизайн;
– відсутність фінансових операцій;
– відсутність підтримки благодійних програм;
– призначення лише для конкретної організації.

Вищезазначені недоліки обумовлюють необхідність розробки мобільного застосунку інформаційної системи «Каса взаємодопомоги», який би забезпечував учасників зручним та функціональним інтерфейсом для користування всіма можливостями фонду «Каса взаємодопомоги».

РОЗДІЛ 2. ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ ІНФОРМАЦІЙНОЇ СИСТЕМИ «КАСА ВЗАЄМОДОПОМОГИ»

2.1 Взаємодія компонентів інформаційної системи

Мобільний застосунок учасника благодійних програм для свого функціонування передбачає взаємодію із сервером мобільного застосунку, що забезпечує доступ до даних стосовно роботи благодійного фонду, для завантаження персональних даних може відбуватися взаємодія з серверами державного сервісу “Дія”, а також з митою сплати за участь у благодійних програмах фонду, учасник має взаємодіяти з сервісом LiqPay державного банку ПриватБанк. Схема взаємодії наведена на рисунку 2.1.

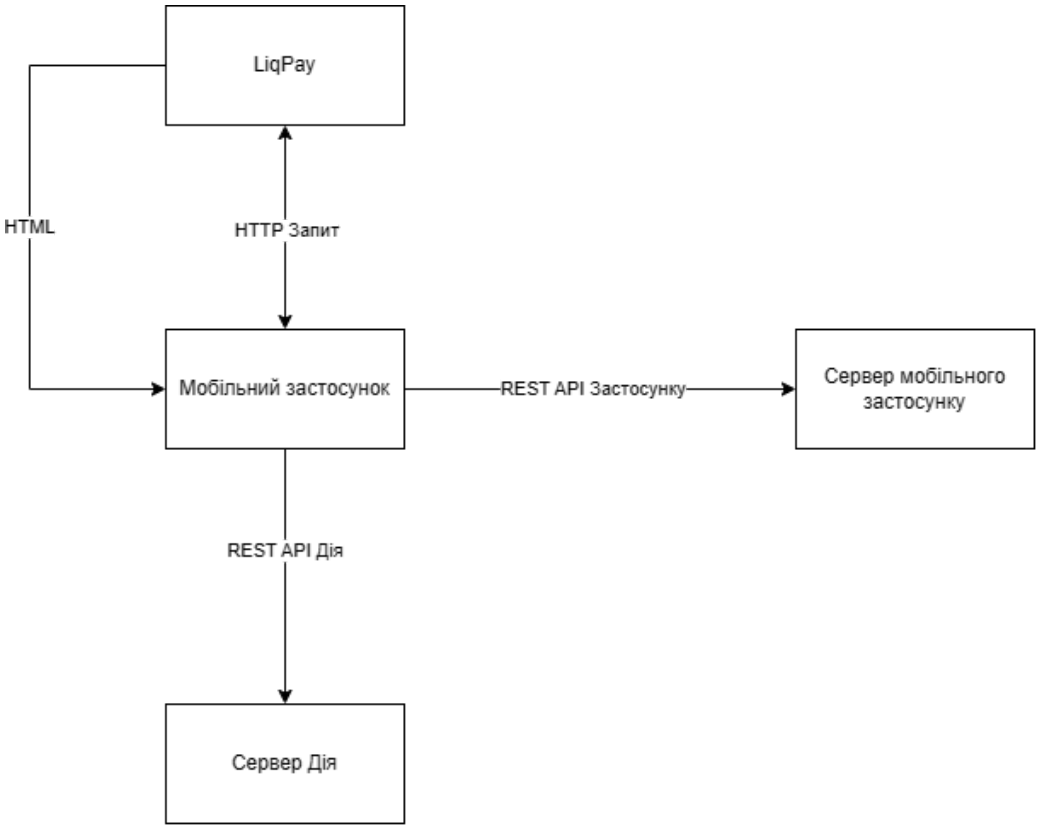

Рис. 2.1. Схема взаємодії компонентів системи

Мобільний застосунок виконує запити до сервера мобільного застосунку, з метою виконання всіх основних операцій, а саме: авторизації, управління профілем учасника програм, подання запитів на благодійну

допомогу, перегляд фінансових транзакцій та перерахування коштів. Частина з цих операцій вимагає завантаження файлів, тож запити у якості вхідних даних використовують як тип контенту JSON, так і x-www-form-urlencoded.

За наявності облікового запису у сервісі Дія, учасник може авторизуватися в ньому, та відправити запит на надання персональних даних, включно з файлами ID-карти та ідентифікаційного коду для серверної частини інформаційної системи благодійного фонду. В цьому випадку, відповідь на запит буде спрямована вже безпосередньо у серверну частину інформаційної системи.

У випадку використання сервісу LiqPay для перерахування коштів, за участю в благодійних програмах, мобільний застосунок має відправити зашифрований запит на отримання платіжної сторінки від сервісу LiqPay. Відповідне шифрування виконується на сервері мобільного застосунку і надається мобільному застосунку для подальшого використання. Користувач заповнює платіжну інформацію на наданій LiqPay сторінці і намагається підтвердити факт оплати. У випадку успіху, відповідь сервера LiqPay відправляється до серверної частини благодійного фонду, а у випадку виникнення проблеми LiqPay повертає мобільному застосунку сторінку з повідомленням про помилку. Такий варіант реалізації взаємодії з LiqPay обумовлений внутрішнєю архітектурою сервісу LiqPay, який приймає платіжну інформацію виняткова зі сторінок свого сайту.

2.2 Вибір технологій та мов програмування для створення програмного продукту

Для створення застосунку були обрані фреймворк Flutter, бібліотека Cubit, та мова програмування Dart.

Flutter [10] – це платформа з відкритим вихідним кодом, який розроблено та підтримується Google. Фронтенд-розробники і

фуллстек-розробники використовують Flutter для створення інтерфейсу застосунків (UI) для різних платформ із застосуванням єдиної бази коду.

Після випуску Flutter у 2018 році ця платформа переважно підтримувала розробку мобільних застосунків. Тепер Flutter підтримує розробку програм на шести платформах: iOS, Android, веб-інтерфейс, Windows, MacOS та Linux.

Flutter спрощує процес створення одноманітних привабливих інтерфейсів для застосунку на шести підтримуваних платформах.

Розробка кросплатформових додатків дає розробникам можливість використовувати одну мову програмування та одну базу коду, щоб створити програму для кількох платформ. Якщо програму випускається для декількох платформ, кросплатформова розробка вимагатиме менших витрат і меншого часу, ніж розробка платформозалежної програми. Також цей процес дозволяє розробникам створювати однаковіші інтерфейси для різних платформ.

Цей підхід може мати недоліки в порівнянні з розробкою платформозалежних програм, у тому числі обмежений доступ до функцій пристрою. Однак у Flutter реалізовані можливості, які роблять розробку кросплатформових додатків більш простою та високоефективною.

Flutter виділяється серед платформ для кроссплатформенної розробки завдяки наступним перевагам.

1. Продуктивність близька до продуктивності платформозалежних додатків. Flutter використовує мову програмування Dart та компілюється в машинний код. Пристрої розуміють цей код, що забезпечує швидку роботу та високу продуктивність.

2. Швидке, однакове відображення, що настроюється. Flutter не покладається на платформозалежні інструменти відображення, а використовує для відображення інтерфейсу користувача графічну бібліотеку Google Skia з відкритим вихідним кодом. Це надає користувачам однакові візуальні елементи незалежно від платформи, яка використовується для доступу до програми.

3. Зручні інструменти для розробників Google створила Flutter з акцентом на простоті використання. Завдяки таким функціям, як гаряче перезавантаження, розробники можуть попередньо переглядати, як виглядатимуть зміни в коді, без втрати стану. Інші інструменти, такі як інспектор віджетів, спрощують візуалізацію та вирішення проблем у макетах інтерфейсу користувача.

Розробники створюють макети інтерфейсу користувача на Flutter за допомогою віджетів. Це означає, що все, що користувач бачить на екрані від вікон і панелей до кнопок і тексту, складається з віджетів. Віджети Flutter розроблені так, щоб розробникам було просто налаштовувати їх. У Flutter це реалізовано з використанням композиційного підходу. Це означає, більшість віджетів складається з менших віджетів і більшість базових віджетів мають специфічне призначення. Це дозволяє розробникам комбінувати або редагувати віджети для створення нових віджетів.

Flutter відображає віджети за допомогою власного графічного двигуна, не покладаючись на вбудовані віджети платформи. Завдяки цьому користувачам надається той самий зовнішній вигляд програми Flutter на різних платформах. Крім того, цей підхід надає розробникам гнучкість, тому що деякі віджети Flutter можуть виконувати функції, недоступні для віджетів, специфічних для платформи.

Крім того, Flutter полегшує використання віджетів, розроблених спільнотою. Архітектура Flutter підтримує безліч бібліотек віджетів, і Flutter підтримує спільноту у розробці та підтримці нових.

Крім того, Flutter поставляється з макетами та темами, завдяки чому розробники можуть одразу приступати до компонування.

Flutter використовує мову програмування з відкритим кодом Dart [], який також розроблений в Google. Dart оптимізовано для створення інтерфейсу користувача, і багато його переваг використовуються у Flutter.

Dart [11] - це доступна, портативна та продуктивна мова для високоякісних програм на будь-якій платформі.

Її переваги:

- Доступність: строго типізована мова програмування, яка є послідовною, лаконічною та пропонує такі сучасні мовні функції, як нульовий захист і шаблони.
- Портативність: компілюється в машинний код ARM, x64 або RISC-V для мобільних пристроїв, настільних комп'ютерів і серверної частини. Компілюється в JavaScript або WebAssembly для Інтернету.
- Продуктивність: є можливість вносити зміни ітеративно: гаряче перезавантаження використовується, щоб миттєво побачити результат у запущеній програмі. Для діагностування проблем з програмою застосовується утиліта DevTools.

Гнучка технологія компілятора Dart дозволяє запускати код Dart різними способами, залежно від платформи та цілей:

Dart Native: для програм, націлених на пристрої (мобільні, настільні, серверні та інші), Dart Native включає як віртуальну машину Dart із компіляцією JIT (точно вчасно), так і компілятор AOT (завчасно) для створення машинного коду.

Dart Web: для програм, орієнтованих на Інтернет, Dart Web містить як компілятор часу розробки (dartdevc), так і компілятор часу виробництва (dart2js).

Одна з можливостей Dart, яка використовується у Flutter, – захист від нульових посилань. Захист від нульових покажчиків спрощує виявлення поширених помилок, які називаються помилками нульових посилань. Ця можливість скорочує час, що витрачається розробниками обслуговування коду, звільняючи час створення додатків.

Cubit [12] - це легка бібліотека управління станом для Flutter, створена на основі BLoC (Business Logic Component). Вона пропонує простий і зрозумілий інтерфейс для керування станом UI компонентів, роблячи код більш організованим, передбачуваним та керованим.

Переваги використання Cubit:

- Простота: Cubit має простий API, що робить його легким для вивчення та використання.
- Легкість: Cubit не потребує boilerplate коду, завдяки чому код стає більш лаконічним.
- Передбачуваність: Cubit використовує детерміновані оновлення стану, що робить поведінку UI більш передбачуваною.
- Тестованість: Cubit легко тестувати, що робить код більш надійним.

Як працює Cubit:

Cubit використовує клас Cubit для представлення стану UI. Цей клас містить методи для доступу до стану, оновлення стану та надсилання подій.

Для оновлення стану потрібно використовувати метод emit(). Цей метод приймає новий стан як аргумент і оновлює стан Cubit. Всі віджети, які залежать від цього стану, будуть автоматично перебудовані.

Для надсилання подій також використовується метод emit(). Цей метод приймає подію як аргумент і надсилає її всім віджетам, які підписані на цю подію.

2.3 Архітектура мобільного застосунку «Каси взаємодопомоги»

Рівень бізнес-логіки, або Domain Layer, відповідає за реалізацію основної логіки додатка, включаючи всі бізнес-правила та обробку даних. При проєктуванні цього рівня додатка був проведений аналіз рішень, які спрощують реалізацію.

StatefulWidgets — один з основних компонентів представлення (UI), в якому описується логіка роботи та правила відображення. Завдяки його можливостям, можна отримати дані, обробити їх та відобразити за допомогою setState(() {}) методу.

Переваги:

- Найпростіший у використанні з усіх;

- Мала кількість коду при простій логіці;

- Не потребує додаткових бібліотек або складних конфігурацій.

Недоліки:

- Складно масштабувати при великих проєктах;

- Немає розділення бізнес-логіки від UI компонентів;

- Тестування цього підходу є досить важким;

- При досить великих чи складних State-ах продуктивність додатка стає досить низькою.

ChangeNotifier є частиною бібліотеки provider у Flutter і використовується для управління станом різних компонентів додатка через сповіщення підписників про зміни.

Переваги:

- Простий у використанні;

- Дозволяє налаштувати складні взаємодії між різними частинами додатка;

- Бізнес-логіка може легко бути протестована окремо від візуальної частини додатка.

Недоліки:

- При реалізації великих додатків зі складними залежностями масштабування може бути досить складним.

BLoC (Business Logic Component) — архітектурний шаблон, що відокремлює бізнес-логіку від представлення. Використовує потоки (Streams) для управління станом та обробки подій. Найпопулярнішою бібліотекою з реалізацією цього шаблону є flutter_bloc.

Переваги:

- Чітке розділення логіки та UI, що полегшує масштабування додатка;

- Логіка має чітку і сформовану структуру, кожна подія має визначений обробник;

- Логіка легко тестується окремо від UI.

Недоліки:

- Має досить складну для розуміння структуру;

- Потребує досить багато коду, якщо порівнювати з попередніми варіантами.

Однак, flutter_bloc містить також спрощену версію реалізації BLoC-шаблону, яка називається Cubit. Він має набагато простішу структуру, легший для розуміння, та менший за обсягом коду, що нівелює перераховані недоліки.

Архітектура мобільного застосунку «Каси взаємодопомоги» основана на концепцію чистої архітектури [13]. Основна ідея полягає в тому, щоб розділити код на слабозв’язані рівні, отже, можна досягти зручності обслуговування, масштабованості та тестування [14]. Ця структура забезпечує кращий розподіл завдань, де кожен рівень має чіткий API свого функціоналу, а класи функцій знаходяться на різних рівнях, де посилання на них неможливе без явної залежності модуля.

Застосунок містить наступні рівні архітектури (рис. 2.2):

 Рівень даних: цей рівень містить логіку, необхідну для взаємодії з джерелами даних, такими як кеш (бази даних, безпечне сховище, спільні параметри), віддалене (служби API) або будь-яке інше джерело даних, яке може знадобитися програмі. Іншими словами, цей рівень забезпечує єдине джерело правдивих даних.

 Рівень домену: він містить випадки використання, які інкапсулюють одне й дуже конкретне завдання, яке потрібно виконати. Цей рівень витягує бізнес-логіку з рівня презентації, наприклад із BLoC, щоб зробити клас BLoC презентації простішим із єдиною роботою для координації між представленнями та варіантами використання.

 Рівень презентації: цей рівень містить інтерфейс користувача (перегляди, віджети) і керування станом BLoC, яке розкриває стани, які споживатиме інтерфейс користувача.

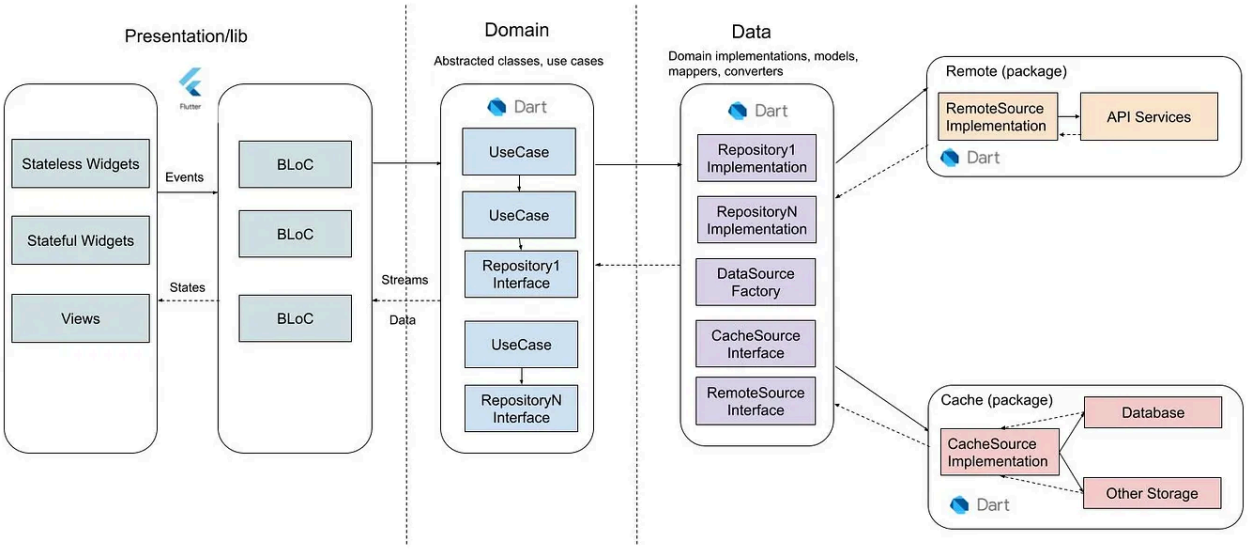

Рис. 2.2. Архітектура Flutter застосунку [14]

Рівень даних має доступ лише до рівня домену та містить наступне:

- Реалізація репозиторіїв для інтерфейсів репозиторіїв, визначених доменним рівнем. Наприклад, Domain визначає контракт для функцій, пов’язаних з featureX, а Data створює реалізацію та отримує дані з джерела даних. Це корисно, оскільки є можливість змінювати або додавати кілька реалізацій без взаємодії з доменом.
- Джерела даних, які реалізують логіку доступу до даних із різних джерел, наприклад Cache та Remote. Завдяки такому підходу можна легко додати нове джерело даних.
- Фабрика джерел даних, яка забезпечує механізм вибору правильного джерела даних, коли це необхідно.

Рівень домену не має доступу та не має інформації про зовнішній світ. Він включає наступне:

- Варіанти використання, які виконують конкретне завдання. Ці випадки можуть об’єднувати дані з одного чи кількох сховищ, наприклад, після виконання віддаленого входу зберегти маркер доступу в безпечному сховищі в кеші.
- Моделі, які декларують необхідний формат даних.

Презентаційний рівень у даному підході представляє каталог lib, який містить залежності, пов’язані з фреймворком Flutter. Цей рівень матиме доступ до всіх інших рівнів для цілей впровадження залежностей, де в ідеалі потрібен лише рівень домену. Крім того, він містить наступне:

- Перегляди Flutter і віджети, такі як віджети Stateless і Stateful.
- BLoC, що означає компонент бізнес-логіки — підхід до управління станом у Flutter — класи, які виконуватимуть один або кілька варіантів використання та випромінюватимуть стани для прив’язаного до нього перегляду/віджета.

Структура рівня даних

Як згадувалося раніше, рівень даних має джерела даних:

- Remote: це чистий код Dart, який оброблятиме віддалений зв’язок API із серверною частиною.
- Кеш: це чистий код Dart, який оброблятиме збереження даних в автономному режимі. Тут можна використовувати SQLite DB, Shared Preferences, Secure Storage тощо.
- Будь-які інші джерела: це можуть бути будь-які інші джерела даних, які можуть знадобитися для проекту.

Розроблюваний застосунок не потребує збереження поточної інформації на самому пристрої, тому питання про структуру та розміщення бази даних у цій роботі не розглядались. Проте протягом роботи застосунку він отримує доступ до даних, які містяться на віддаленому сервері. Для організації доступу до цих даних, застосунок мусить мати набір відповідних класів, з відповідними полями, щоб забезпечити інтерфейс доступу візуальним компонентам. На рисунку 2.3 наведено діаграму класів шару даних застосунку.

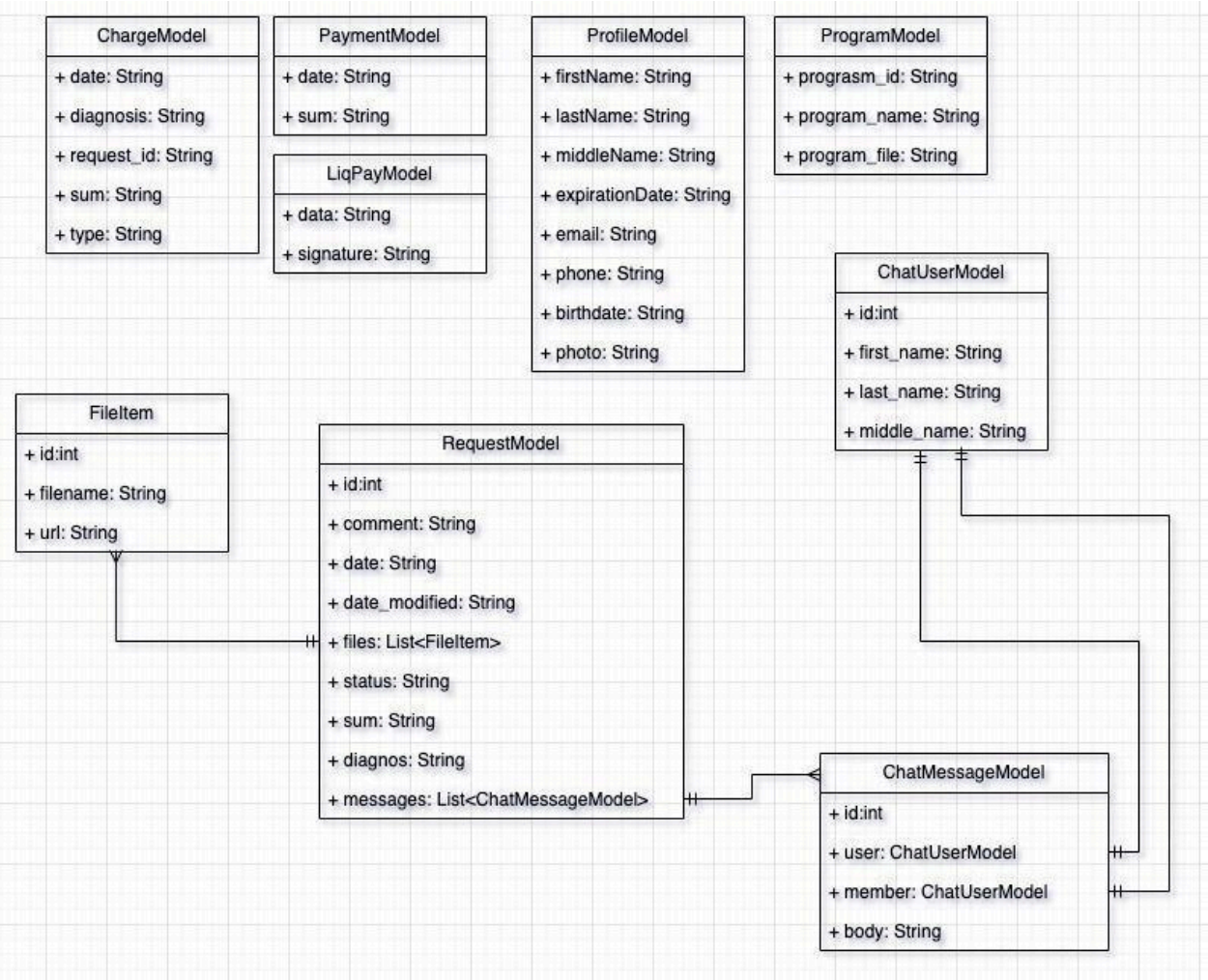

Рис. 2.3. Діаграма класів шару даних застосунку

У таблицях 2.1-2.8 наведено короткий опис полів класів, що складають шари даних застосунку.

Таблиця 2.1 - Опис полів класів шару даних моделі PaymentModel

Назва поля	**Тип даних**	**Призначення**
date	STRING	Дата внеску
sum	STRING	Сума внеску

Таблиця 2.2 - Опис полів класів шару даних моделі ChargeModel

Назва поля	**Тип даних**	**Призначення**
date	STRING	Дата виплати
diagnosis	STRING	Діагноз
request_id	STRING	Ідентифікатор запиту
sum	STRING	Сума виплати

Назва поля	Тип даних	Призначення
type	STRING	Тип виплати

Таблиця 2.3 - Опис полів класів шару даних моделі LiqPayModel

Назва поля	Тип даних	Призначення
data	STRING	Дата виконання операції
signature	STRING	Електронний підпис

Таблиця 2.4 - Опис полів класів шару даних моделі ProfileModel

Назва поля	Тип даних	Призначення
firstName	STRING	Ім’я
lastName	STRING	Прізвище
middleName	STRING	По-батькові
expirationDate	STRING	Дата завершення участі у благодійних програмах
email	STRING	Електронна адреса
phone	STRING	Номер телефону
birthdate	STRING	Дата народження
photo	STRING	Фото для відображення у мобільному застосунку

Таблиця 2.5 - Опис полів класів шару даних моделі ProgramModel

Назва поля	Тип даних	Призначення
program_id	STRING	Ідентифікатор програми
program_name	STRING	Назва програми
program_file	STRING	Повний URL файлу благодійної програми

Таблиця 2.6 - Опис полів класів шару даних моделі RequestModel

Назва поля	Тип даних	Призначення
id	INT	Ідентифікатор запиту
comment	STRING	Коментар до запиту
date	STRING	Дата створення запиту
date_modified	STRING	Дата оновлення запиту
files	List<FileItem>	Файли додані до запиту
Назва поля	**Тип даних**	**Призначення**
status	STRING	Статус запиту

sum	STRING	Сума запиту
diagnosis	STRING	Діагноз запиту
messages	List<ChatMessageModel>	Повідомлення листування щодо запиту

Таблиця 2.7 - Опис полів класів шару даних моделі ChatMessageModel

Назва поля	**Тип даних**	**Призначення**
id	INT	Ідентифікатор повідомлення
user	UserModel	Об’єкт що містить інформацію про адміністратора, який написав повідомлення
member	UserModel	Об’єкт що містить дані про учасника програм, який написав повідомлення
body	STRING	Текст повідомлення

Таблиця 2.8 - Опис полів класів шару даних моделі ChatUserModel

Назва поля	**Тип даних**	**Призначення**
id	INT	Ідентифікатор листування
first_name	STRING	Ім’я
last_name	STRING	Прізвище
middle_name	STRING	По-батькові

2.4 Висновки до розділу 2

У другому розділі кваліфікаційної роботи було проведено проєктування мобільного застосунку «Каси взаємодопомоги». Спочатку була розглянута структура та бізнес-логіка застосунку.

Було розглянуто фреймворки для кросплатформної мобільної розробки, і для створення програмного коду застосунку було обрано Flutter та мову програмування Dart. У якості основи для архітектури застосунку був обраний архітектурний шаблон BLoC (Business Logic Component) та концепція чистої архітектури. Код застосунку розділяється на слабозв’язані рівні, що допоможе досягти зручності обслуговування, масштабованості та тестування програми.

Для організації доступу до даних інформаційної системи застосунок буде мати набір відповідних класів, з відповідними полями, щоб забезпечити інтерфейс доступу візуальним компонентам із рівня презентації. Було наведено графічне зображення даної класової моделі, та її детальний опис.

РОЗДІЛ 3. ОПИС ПРОГРАМНОГО КОМПОНЕНТУ МОБІЛЬНОГО ЗАСТОСУНКУ «КАСИ ВЗАЄМОДОПОМОГИ»

3.1 Варіанти використання застосунку

Взаємодія учасника благодійних програм з благодійним фондом передбачає певний порядок дій, які реалізує розроблений застосунок. Діаграма використання застосунку наведена на рисунку 3.1

Рис. 3.1. Діаграма використання застосунку благодійного фонду

Користувач має підтвердити свій номер телефону, введенням одноразового пароля, який відправляється йому через СМС. Це підтвердження фактично є авторизацією у застосунку. Якщо авторизація відбувається вперше, користувач має надати свої персональні дані та фотокопії документів, аби співробітники фонду могли їх перевірити та укласти с учасником угоду. Після успішної авторизації, користувач може перераховувати внески за участь у благодійних програмах, та переглядати їх перелік, оформлювати запити на виплати, згідно з умовами благодійної

програми (додаючи потрібні файли), а також даними свого профілю, такими як номер телефону, адреса електронної пошти тощо.

3.2 Структура проєкту

Оскільки «чиста» архітектура Flutter застосунків не пов'язана з жодною конкретною методологією управління станом, для управління станом мобільного застосунку фонду взаємодопомоги було вирішено використати техніку «Постачальник» (Provider). На рисунку 3.2 показано потік даних кожної події, ініційованої користувачем, у рамках реалізації даного шаблона.

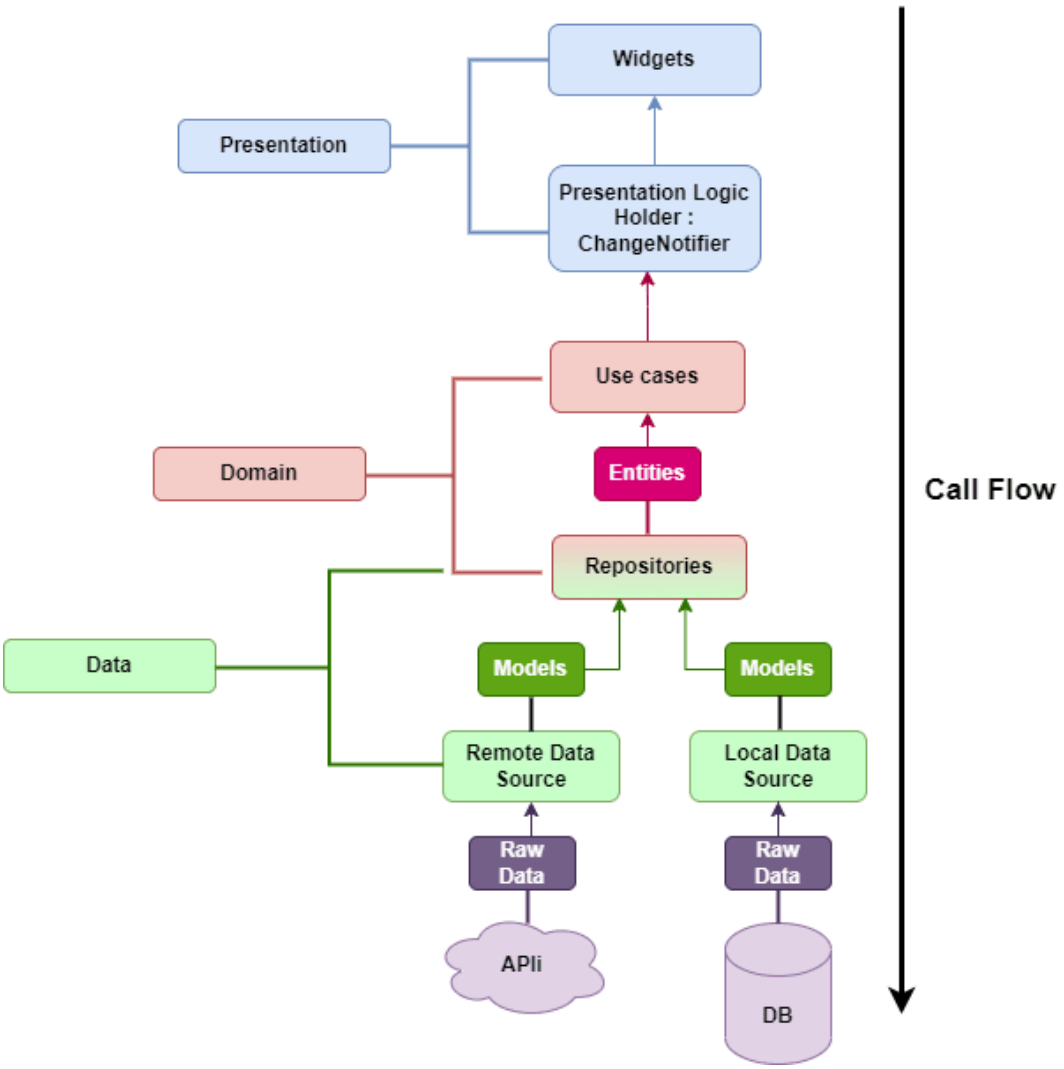

Рис. 3.2. Потік даних у застосунку Flutter [15]

Коли користувач взаємодіє з віджетом, його дія повідомляється класу Provider, який потім підключається до useCase, щоб отримати результат дії. Після цього useCase взаємодіє з класом Repository, щоб отримати рішення з віддаленого або локального джерела даних.

Для того, щоб здійснити впровадження шарів «чистої» архітектури, проект розділяється на підпапки. Структура дерева папок проекту показана на рис. 3.3.

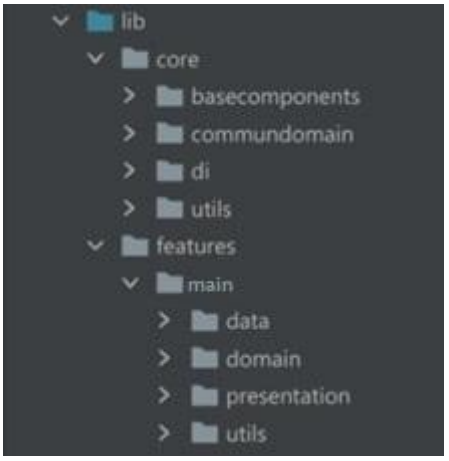

Рис. 3.3. Дерево папок проекту

В папці lib створено дві вкладені папки. Перша називається «core», і вона містить усі спільні та базові компоненти, а також реалізацію основних функцій, таких як впровадження залежностей. Друга папка має назву «features» і містить усі «функції» програми. Кожна функція також буде розділена на три рівні: дані, домен, презентація та утиліти.

Папка utils не потрібна для кожної функції; це потрібно лише тоді, коли у проекті є окремі утиліти функцій, такі як переліки, класи, функції розширення тощо.

Рівні features:

- Презентація

Цей рівень відповідає за представлення даних користувачеві за допомогою віджетів. Цей рівень також відповідає за прослуховування станів і підключення до класу Provider, який делегує всю роботу конкретному

випадку використання. У більшості випадків основним завданням рівня презентації є обробка основних перевірок введення, анімації та взаємодії з користувачем. Структура папок презентації показана на рис. 3.4.

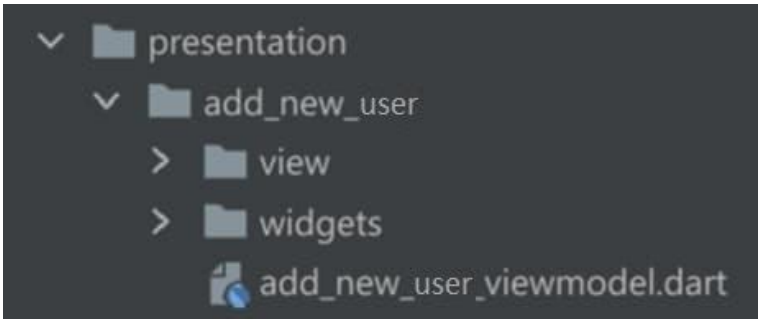

Рис. 3.4. Папки рівня презентації

Кожен інтерфейс застосунка матиме дві вкладені папки, одну для перегляду файлу, а іншу для його віджетів. Також включається файл провайдера.

- Домен

Це клейовий шар, який з'єднує два інших шари. Він використовуватиме бізнес-об'єкти (сутності) і варіанти використання для реалізації бізнес-логіки програми. Крім того, кожен варіант використання покладається на репозиторій для отримання даних.

Рівень домену буде поділено на три окремі рівні: сутності, сховища та варіанти використання, як показано на рис. 3.5.

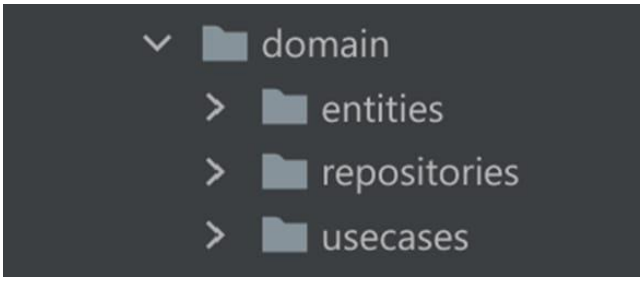

Рис. 3.5. Папки рівня домен

Сутність: інкапсулює бізнес-правила високого рівня для всього підприємства. Лише ті поля, які мають сенс з «ділової точки зору», зберігаються сутністю. Крім того, сутність є кінцевим результатом, який

використовуватиме представлення, і найменша ймовірність її зміни під час зовнішніх змін. Крім того, необов’язково, щоб сутність мала однакові поля, які повертає веб-служба - можна налаштувати поля відповідно до потреб інтерфейсу.

Варіант використання: Варіант використання організовує потік даних до та з сутності. Варіант використання, у своїй основній формі, представляє дію користувача. Щоб відповідати принципу єдиної відповідальності SOLID, кожен варіант використання має бути незалежним від інших.

Рівень домену повинен бути повністю незалежним від усіх інших рівнів. Цього важко досягти, коли будь-який варіант використання отримує дані зі сховища, яке використовується спільно з рівнем даних. Тому був використаний один із принципів SOLID, інверсії залежностей. Фактично, на рівні домену ми створюється абстрактний клас Repository, щоб визначити умови того, що має робити Repository, а потім на рівні даних пишеться його фактична реалізація.

Репозиторій: клас репозиторію служить єдиним джерелом даних, відокремлюючи логіку, яка отримує дані та відображає їх у моделі сутності. Цей клас збирає дані з різних джерел (REST API, локальні бази даних, кеш тощо) і надає їх решті програми. Інші компоненти не «знають», звідки беруться дані; вони просто «споживають» їх.

- Дані

Цей рівень відповідає за отримання даних з багатьох джерел. Він складається з класу сховища, який реалізує договір домену та визначає, чи повертати свіжі чи кешовані дані, а також коли їх кешувати. Крім того, клас джерела даних обробляє вибірку даних із певного джерела - віддаленого API (або локальної бази даних).

Як показано на рис. 3.6, три підрівні складатимуть рівень даних: джерела даних, реалізації репозиторіїв і моделі.

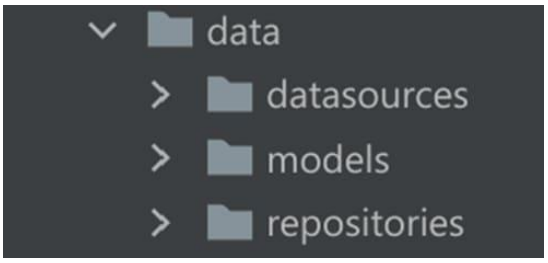

Рис. 3.6. Папки рівня даних

Через те, що їх надає кожне джерело даних, створюється папка models, а не сутності. Насправді кожна модель відповідає за перетворення неструктурованих даних (JSON тощо) в об’єкти Dart за допомогою певних методів (fromJSON, toJSON тощо).

3.3 Порядок взаємодії з графічним інтерфейсом користувача

Після завантаження додатка, користувач попадає на секцію подачі заявки на реєстрацію, де йому також пропонується авторизуватися. Для подачі заявки, спершу, потрібно вписати номер і СМС код, який буде відправлено на телефонний номер. Приклад цього можна побачити на рисунку 3.7.

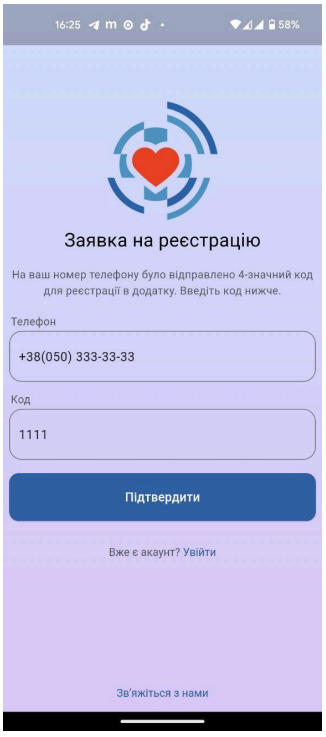

Рис. 3.7. Сторінка заявки на реєстрацію

Після цього, користувача буде перенаправлено на наступну секцію реєстрації, де він повинен вписати персональні дані по наступним полям або авторизуватися через додаток “Дія”, звідки буде завантажено всю необхідну інформацію:

- Прізвище
- Ім’я
- По-батькові
- E-mail
- IBAN
- Завантажити копію паспорта старого зразка або ID-картку
- ІПН

Приклад цього можна побачити на рисунках 3.8, 3.9, 3.10, 3.11.

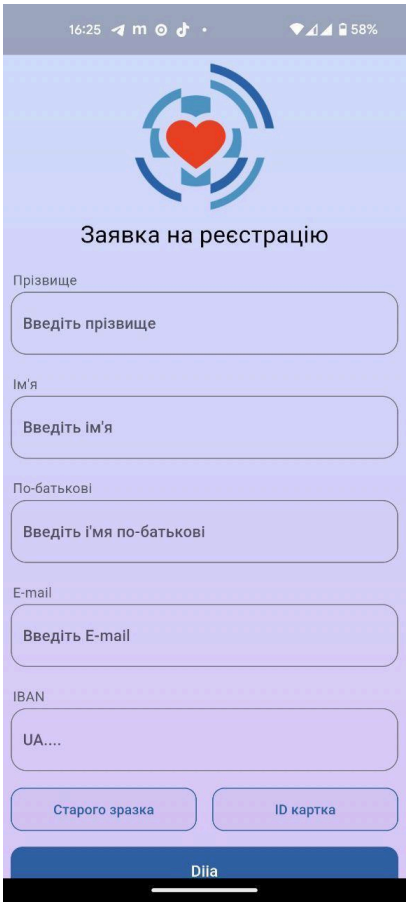

Рис. 3.8. Сторінка заявки на реєстрацію

Рис. 3.9. Сторінка заявки на реєстрацію

Рис. 3.10. Сторінка заявки на реєстрацію

Рис. 3.11. Сторінка заявки на реєстрацію

Після натискання на кнопку “Стати учасником”, користувача буде перенаправлено на сторінку з вибором програм, як можна побачити на рисунку 3.12.

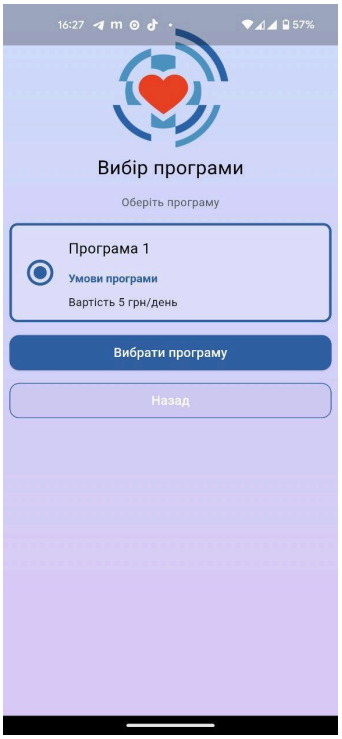

Рис. 3.12. Сторінка заявки на реєстрацію - вибір програми

Коли користувач визначиться з програмами, йому буде необхідно натиснути на кнопку "Вибрати програму", після чого реєстрацію користувача буде завершено, як показано на рисунку 3.13.

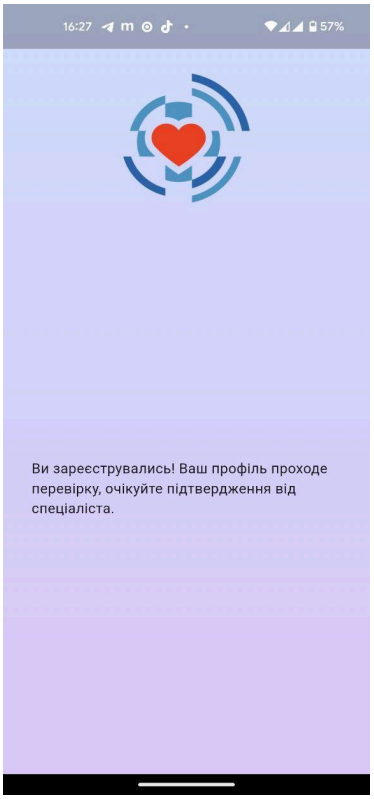

Рис. 3.13. Завершення реєстрації

Після завершення реєстрації, профіль нового користувача повинен бути підтверджений спеціалістами.

Після підтвердження профілю, для авторизації потрібно ввести номер телефону, отримати СМС-код і вписати його в поле.

Після успішної авторизації користувач потрапляє на головний екран застосунку. Внизу екрану розташоване меню з п'ятьма сторінками, які можна побачити на рисунку 3.14: "Внески", "Виплати", "Заявки", "Програми" та "Профіль". За замовчуванням відкривається сторінка "Внески". Користувач може переміщатися між сторінками, натискаючи на кожну з них.

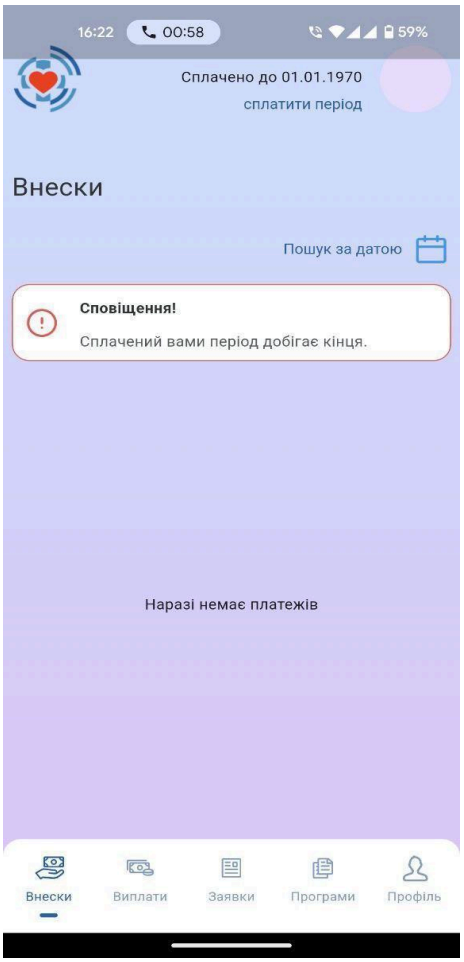

Рис. 3.14. Сторінка “Внески”

На сторінці “Внески” можна переглянути всі зроблені користувачем внески, по яким можна перейти натиснувши на наявний запис. За необхідності, вони можуть бути відфільтровані за датами.

На сторінці “Виплати” можна переглянути всі виплати за запитами користувача, що можна побачити на рисунку 3.15. За необхідності, вони можуть бути відфільтровані за датами.

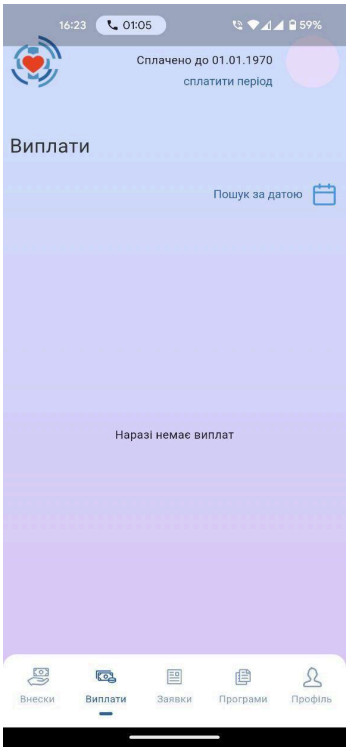

Рис. 3.15. Сторінка “Виплати”

На сторінці “Заявки” можна переглянути всі заявки отримані благодійним фондом, по яким можна перейти натиснувши на наявний запис. За необхідності, вони можуть бути відфільтровані за датами. Це можна побачити на рисунку 3.16.

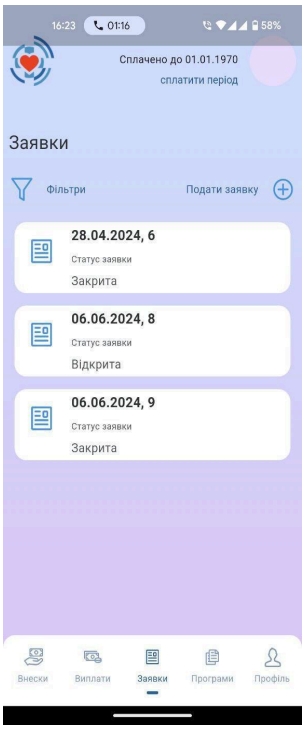

Рис. 3.16. Сторінка “Заявки”

При переході на вже наявну заявку, користувач побачить наступну інформацію про заявку: її статус, коментар, дату подачі та суму виплати призначеної за цим запитом.

При натисканні на кнопку “Подати заявку” відкривається форма створення нової заявки, де треба завантажити необхідний документ та ввести коментар. Для того, щоб зберегти заявку до її розгляду, потрібно натиснути на кнопку "Зберегти", для того, щоб скасувати - "Скасувати". Побачити це можна на рисунку 3.17.

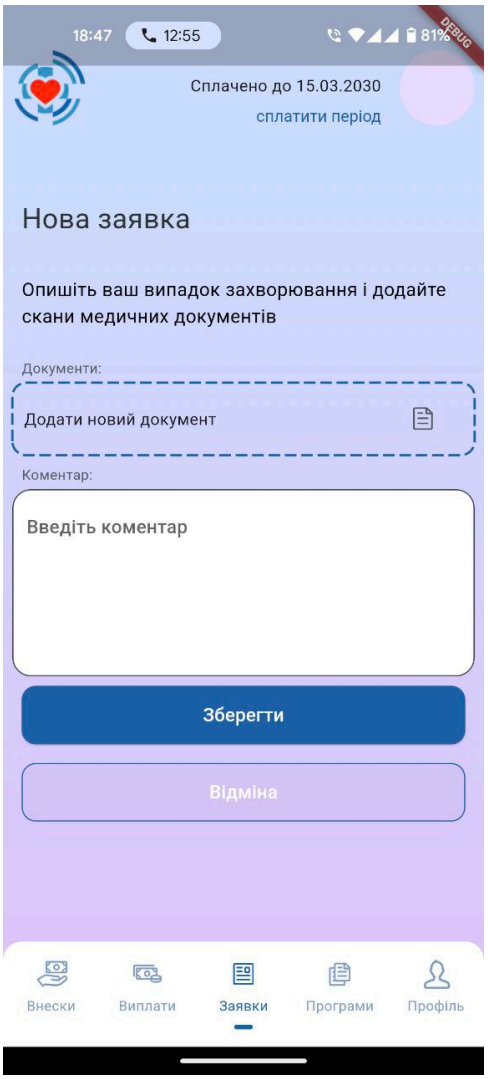

Рис. 3.17. Форма подачі нової заявки

На сторінці “Програми” можна побачити список всіх програм, а також поточну програму, яку, за потреби, можна змінити, обравши одну з наявних і натиснувши кнопку "Обрати". Це можна побачити на рисунку 3.18.

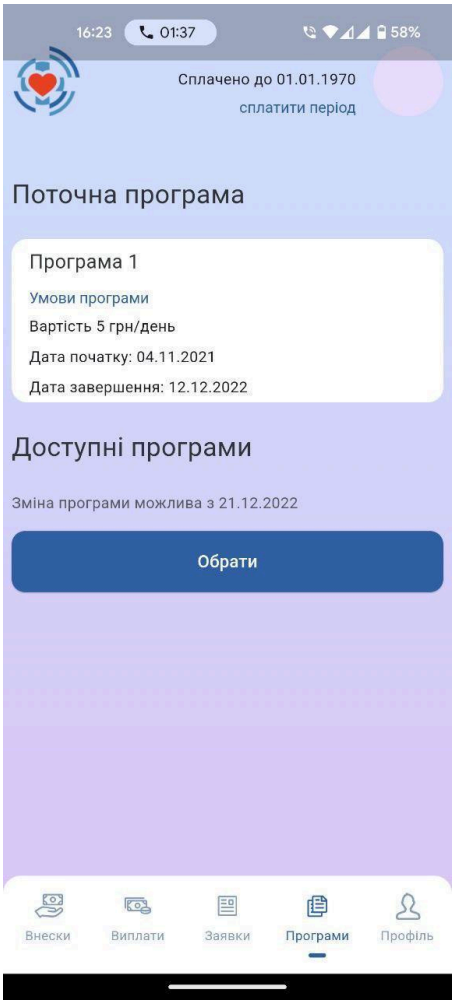

Рис. 3.18. Сторінка “Програми”

При переході на сторінку “Профіль” користувач бачить всю інформацію по його профілю, зокрема: прізвище, ім’я, по-батькові, e-mail, дату народження, номер телефону. За необхідності, користувач зможе поміняти аватар, номер телефону, e-mail, ім'я, прізвище, або паспортні дані.

У разі натискання на кнопку праворуч зверху "Оплатити період", відкривається вікно оплати, де потрібно обрати необхідну кількість місяців для оплати та натиснути на кнопку "Поповнити", після чого учасника буде

перенаправлено на сторінку оплати LiqPay. Приклад цього можна побачити на рисунку 3.19.

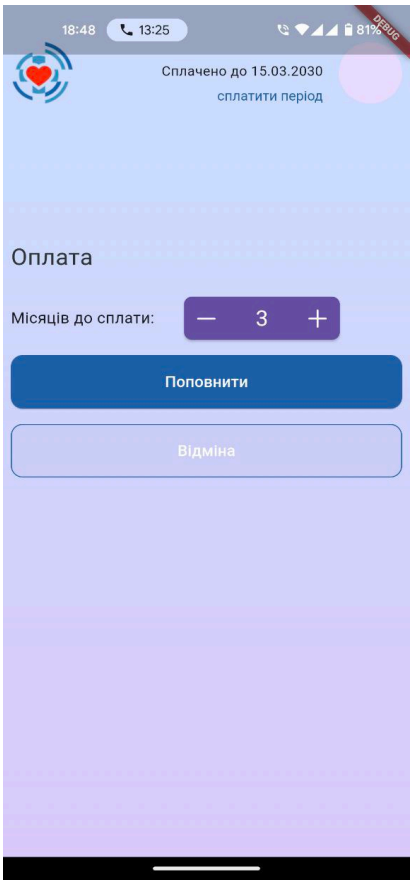

Рис. 3.19. Сторінка з оплатою

3.4 Висновки до розділу 3

В ході виконання третьої частини кваліфікаційної роботи був розроблений Flatter застосунок, що реалізує функції з управління участю в програмах благодійного фонду взаємодопомоги. Було створено діаграму варіантів використання застосунку, досліджено архітектуру застосунку, включаючи його відокремлені складові, зміст і принцип роботи. Також було описано структуру програмного компоненту, включаючи всі функції та класи, з яких він складається, з детальним описом кожного з них.

Також було розроблено опис графічного інтерфейсу програми. Було створено керівництво користувача, яке дозволяє користувачам ознайомитися з функціональністю мобільного застосунку інформаційної системи «Каса взаємодопомоги» та отримати обізнаність щодо його використання та взаємодії з ним. Керівництво надає детальні вказівки щодо використання можливостей застосунку та призначення окремих елементів дизайну.

Розроблений застосунок був протестований безпосередньо виконавцем кваліфікаційної роботи вручну, але для виходу на PlayMarket або PlayStore мають бути виконані ґрунтовніші тести.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи "Розробка мобільного застосунку інформаційної системи «Каса взаємодопомоги»" було успішно реалізовано ряд завдань та досягнуто мету роботи.

Застосунок написаний мовою Dart з використанням фреймворку Flutter та бібліотеки Cubit. У якості основи для архітектури застосунку був обраний архітектурний шаблон BLoC (Business Logic Component) та концепція чистої архітектури.

Розроблений застосунок дозволяє:

- подавати запити на регістрацію нових членів фонду;
- завантажувати файли на сервер благодійного фонду;
- запитувати персональну інформацію с державного сервісу “Дія”;
- переглядати історію здійснених платежів та історію поданих заяв;
- подавати заяви на отримання допомоги;
- робити благодійні внески на користь тих програм, у яких бере участь користувач, за допомогою платіжного сервісу LiqPay.

Було проведено тестування функціоналу застосунку. Результати тестування були успішними, що підтверджує його стабільну та ефективну роботу.

Документація, включаючи опис функціоналу застосунку та графічного інтерфейсу користувача, була ретельно підготовлена. Це допоможе користувачам швидко ознайомитися з програмним компонентом, та ефективно використовувати його.

Загальним висновком є те, що розроблений мобільний застосунок «Каси взаємодопомоги» відповідає поставленим вимогам та має потенціал для подальшого розвитку. У проєкті застосовані сучасні технології програмування і відповідні методи та шаблони побудови архітектури мобільних застосунків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Rankins M. G. We Keep Us Safe, Venmo Doesn't: The Impact on Peer-to-Peer Payment Apps on Mutual Aid and Community Organizing, 27 N.C. BANKING INST, 2023. 240 p.

2. Dominguez D. et al. Leveraging the Power of Mutual Aid, Coalitions, Leadership, and Advocacy during COVID-19, 75 AM. PSYCH, 2020. 909 p.

3. Mutual aid [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Mutual_aid_(organization_theory)

4. Spade D. Mutual aid: Building Solidarity during this crisis (and the next). Verso, 2020.

5. Mutual Aid 101: History, Politics, and Organizational Structures of Community Care. 2023. [Електронний ресурс] – Режим доступу: https://cunyurbanfoodpolicy.org/news/2023/08/22/mutual-aid-101-history-politics-and-organizational-structures-of-community-care/

6. "Лікарняна каса Житомирської області" [Електронний ресурс] – Режим доступу: https://play.google.com/store/apps/details?id= likkasa.com.ua&hl

7. Dopomoha [Електронний ресурс] – Режим доступу: https://play.google.com/store/apps/details?id=com.leviancode.android.dopomoga&hl=uk

8. Бандеролька [Електронний ресурс] – Режим доступу: https://www.banderolka.support/

9. Lepta [Електронний ресурс] – Режим доступу: https://play.google.com/store/apps/details?id=com.lepta&hl

10. Flutter [Електронний ресурс] – Режим доступу: https://aws.amazon.com/what-is/flutter/

11. Dart [Електронний ресурс] – Режим доступу: https://dart.dev/

12. Cubit [Електронний ресурс] – Режим доступу: https://pub.dev/documentation/bloc/latest/bloc/Cubit-class.html

13. Martin R.C. The Clean Architecture. 2012. [Електронний ресурс] – Режим доступу: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

14. Elaziz A. Shehadeh Flutter Mobile — Clean architecture (part 1). 2022. [Електронний ресурс] – Режим доступу: https://elaziz-shehadeh.medium.com/flutter-mobile-clean-architecture-part-1-6a78d33e651d

15. Mejri M. Flutter Clean Architecture [1]: An Overview. 2023. [Електронний ресурс] – Режим доступу: https://dev.to/marwamejri/flutter-clean-architecture-1-an-overview-project-structure-4bhf

16. Bracha G. The Dart Programming Language 1st Edition. Addison-Wesley Professional PTG, 2016.

17. Buckett C. Dart in Action. O'Reilly, 2013.

18. Dart tutorial [Електронний ресурс] – Режим доступу: https://dart-tutorial.com/#google_vignette

19. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення: ДСТУ 3008-95. – Чинний від 1996–01–01. – К.: Держстандарт України, 1996. – 39 с.

20. Скорочення слів в українській мові у бібліографічному описі. Загальні правила та вимоги: ДСТУ 3582-97. – Чинний від 1998–07–01. – К.: Держстандарт України, 1998. – 24 с. – (Державний стандарт України).

ДОДАТОК А. ЛІСТИНГ

/data/remote/payment/ConcordModel.dart

```
import 'dart:core';

import 'package:json_annotation/json_annotation.dart';

part 'ConcordModel.g.dart';

@JsonSerializable()
class ConcordModel {
  List<String> add_params;
  int amount;
  String callback_url;
  String client_id;
  String currency_iso;
  String description;
  String merchant_id;
  String operation;
  String order_id;
  String signature;

  ConcordModel(
      {required this.add_params,
      required this.amount,
      required this.callback_url,
      required this.client_id,
      required this.currency_iso,
      required this.description,
      required this.merchant_id,
      required this.operation,
      required this.order_id,
      required this.signature});

   factory ConcordModel.fromJson(Map<String, dynamic>
json) =>
      _$ConcordModelFromJson(json);

            Map<String, dynamic> toJson() =>
_$ConcordModelToJson(this);

  factory ConcordModel.empty() {
    return ConcordModel(
      add_params: List<String>.empty(),
      amount: 0,
      callback_url: "",
      client_id: "",
      currency_iso: "",
      description: "",
      merchant_id: "",
      operation: "",
      order_id: "",
      signature: "");
  }
}
```

/data/remote/payment/LiqPayModel.dart

```
import 'package:json_annotation/json_annotation.dart';


part 'LiqPayModel.g.dart';

@JsonSerializable()
class LiqPayModel {
  String data;
  String signature;


  LiqPayModel(
      {required this.data,
       required this.signature});

    factory LiqPayModel.fromJson(Map<String, dynamic>
json) =>
      _$LiqPayModelFromJson(json);

            Map<String, dynamic> toJson() =>
_$LiqPayModelToJson(this);

  factory LiqPayModel.empty() {
    return LiqPayModel(data: "", signature: "");
  }
```

```
}
```

/data/remote/payment/PaymentModel.dart

```
import 'package:json_annotation/json_annotation.dart';
part 'PaymentModel.g.dart';
@JsonSerializable()
class PaymentModel {
  String date;
  String sum;
  PaymentModel(
      {required this.date,
       required this.sum});

   factory PaymentModel.fromJson(Map<String, dynamic>
json) =>
      _$PaymentModelFromJson(json);

            Map<String, dynamic> toJson() =>
_$PaymentModelToJson(this);

}
```

/data/remote/payment/LiqPayModel.dart

```
import 'package:json_annotation/json_annotation.dart';
part 'LiqPayModel.g.dart';

@JsonSerializable()
class LiqPayModel {
  String data;
  String signature;

  LiqPayModel(
      {required this.data,
       required this.signature});

    factory LiqPayModel.fromJson(Map<String, dynamic>
json) =>
      _$LiqPayModelFromJson(json);

            Map<String, dynamic> toJson() =>
_$LiqPayModelToJson(this);

  factory LiqPayModel.empty() {
    return LiqPayModel(data: "", signature: "");
  }
}
```

/data/remote/charges/ChargeModel.dart

```
import 'package:json_annotation/json_annotation.dart';
part 'ChargeModel.g.dart';

@JsonSerializable()
class ChargeModel {
  String date;
  String diagnosis;

  @JsonKey(includeIfNull: true, defaultValue: "123")
  String request_id;
  String sum;
          @JsonKey(includeIfNull: true, defaultValue:
"Стационар")
  String type;

  ChargeModel(
      {required this.date,
       required this.diagnosis,
       required this.request_id,
       required this.sum,
       required this.type}
       );

    factory ChargeModel.fromJson(Map<String, dynamic>
json) =>
      _$ChargeModelFromJson(json);
            Map<String, dynamic> toJson() =>
_$ChargeModelToJson(this);
}
```

/data/remote/profile/ProfileModel.dart

```
import 'package:json_annotation/json_annotation.dart';
```

```
import 'package:l_kassa/constants/const_values.dart';
import
'package:l_kassa/data/remote/program/ProgramModel.dart';
import 'package:l_kassa/service/ServiceLocator.dart';
import
'package:shared_preferences/shared_preferences.dart';

part 'ProfileModel.g.dart';

@JsonSerializable()
class ProfileModel {
  @JsonKey(name: "first_name")
  String firstName;

  @JsonKey(name: "last_name")
  String lastName;

  @JsonKey(name: "middle_name")
  String middleName;

  ProgramModel program;

      @JsonKey(name:    "expiration_date",    defaultValue:
"01.01.1970")
  String expirationDate;

  @JsonKey(name: "email")
  String email;

  @JsonKey(name: "phone")
  String phone;

  String certificate;

  @JsonKey(name: "birth_date", defaultValue: "")
  String birthdate;

  @JsonKey(name: "photo", defaultValue: "")
  String photo;

  ProfileModel(
      {required this.firstName,
      required this.lastName,
      required this.middleName,
      required this.program,
      required this.expirationDate,
      required this.email,
      required this.phone,
      required this.certificate,
      required this.birthdate,
      required this.photo});

    factory  ProfileModel.fromJson(Map<String,  dynamic>
json) =>
      _$ProfileModelFromJson(json);

            Map<String,      dynamic>      toJson()      =>
_$ProfileModelToJson(this);

  factory ProfileModel.empty() {
    return ProfileModel(
        firstName: "",
        lastName: "",
        middleName: "",
        program: ProgramModel.empty(),
        expirationDate: "",
        email: "",
        phone: "",
        certificate: "",
        birthdate: "",
        photo: "");
  }

  factory ProfileModel.fromStore() {
    SharedPreferences pref = locator<SharedPreferences>();
    return ProfileModel(
                                                  firstName:
pref.getString(AppConstants.PROFILE_FIRST_NAME) ??
"",
                                                   lastName:
pref.getString(AppConstants.PROFILE_LAST_NAME)  ??
"",
                                               middleName:
pref.getString(AppConstants.PROFILE_MIDDLE_NAME)
?? "",
        expirationDate:

pref.getString(AppConstants.PROFILE_EXPIRATION_D
ATE) ?? "",
                                                      email:
pref.getString(AppConstants.PROFILE_EMAIL) ?? "",
```

```
                                                    phone:
pref.getString(AppConstants.PROFILE_PHONE) ?? "",
                                                certificate:
pref.getString(AppConstants.PROFILE_CERTIFICATE)
?? "",
                                                    birthdate:
pref.getString(AppConstants.PROFILE_BIRTH_DATE)  ??
"",
                                                    photo:
pref.getString(AppConstants.PROFILE_PHOTO_URL)??"
",
      program: ProgramModel.fromStore());
  }
  void save() {
    SharedPreferences pref = locator<SharedPreferences>();
     pref.setString(AppConstants.PROFILE_FIRST_NAME,
firstName);
      pref.setString(AppConstants.PROFILE_LAST_NAME,
lastName);

pref.setString(AppConstants.PROFILE_MIDDLE_NAME,
middleName);

pref.setString(AppConstants.PROFILE_EXPIRATION_D
ATE, expirationDate);
    pref.setString(AppConstants.PROFILE_EMAIL, email);
            pref.setString(AppConstants.PROFILE_PHONE,
phone);
     pref.setString(AppConstants.PROFILE_CERTIFICATE,
certificate);
      pref.setString(AppConstants.PROFILE_BIRTH_DATE,
birthdate);
      pref.setString(AppConstants.PROFILE_PHOTO_URL,
photo);
    program.save();
  }
}
```

/data/remote/program/ProgramModel.dart

```
import 'package:json_annotation/json_annotation.dart';
import 'package:l_kassa/constants/const_values.dart';
import 'package:l_kassa/service/ServiceLocator.dart';
import
'package:shared_preferences/shared_preferences.dart';

part 'ProgramModel.g.dart';

@JsonSerializable()
class ProgramModel {
  int program_id;
  String program_name;
  String program_file;

  ProgramModel(
      {required this.program_id,
      required this.program_file,
      required this.program_name});

   factory  ProgramModel.fromJson(Map<String,  dynamic>
json) =>
      _$ProgramModelFromJson(json);

            Map<String,      dynamic>      toJson()      =>
_$ProgramModelToJson(this);

  factory ProgramModel.fromStore() {
    SharedPreferences pref = locator<SharedPreferences>();
    return ProgramModel(
                                                    program_id:
pref.getInt(AppConstants.PROFILE_PROGRAM_ID)     ??
-1,
                                                  program_file:
pref.getString(AppConstants.PROFILE_PROGRAM_FILE
) ?? "",
                                                  program_name:
pref.getString(AppConstants.PROFILE_PROGRAM_NA
ME) ?? "");
  }

  factory ProgramModel.empty(){
     return ProgramModel(program_id: -1, program_file: "",
program_name: "");
  }

  void save(){
    SharedPreferences pref = locator<SharedPreferences>();
        pref.setInt(AppConstants.PROFILE_PROGRAM_ID,
program_id);
```

```
pref.setString(AppConstants.PROFILE_PROGRAM_FILE
, program_file);

pref.setString(AppConstants.PROFILE_PROGRAM_NA
ME, program_name);

  }

}
```

/data/remote/request/FileItem.dart

```
import 'package:json_annotation/json_annotation.dart';

part 'FileItem.g.dart';

@JsonSerializable()
class FileItem {
  int id;
  String filename;
  String url;

  FileItem({required this.id, required this.filename, required
this.url});

   factory  FileItem.fromJson(Map<String,  dynamic> json)
=>
     _$FileItemFromJson(json);

          Map<String,      dynamic>      toJson()      =>
_$FileItemToJson(this);
}
```

/data/remote/request/RequestModel.dart

```
import 'package:json_annotation/json_annotation.dart';
import 'FileItem.dart';

part 'RequestModel.g.dart';

@JsonSerializable()
class RequestModel {
  String comment;
  String date;
  String date_modified;
  List<FileItem> files;
  int id;
  String status;
  double sum;
  String diagnos;

  RequestModel(
     {required this.comment,
     required this.date,
     required this.date_modified,
     required this.files,
     required this.id,
     required this.status,
     required this.sum,
     required this.diagnos});

  factory RequestModel.empty() {
    return RequestModel(
      comment: "",
      date: "",
      date_modified: "",
      files: List<FileItem>.empty(),
      id: 0,
      status: "",
      sum: 0.0,
      diagnos: ""
    );
  }

   factory  RequestModel.fromJson(Map<String,  dynamic>
json) =>
     _$RequestModelFromJson(json);

          Map<String,      dynamic>      toJson()      =>
_$RequestModelToJson(this);
}
```

/data/remote/request/RequestModelDetail.dart

```
import 'package:json_annotation/json_annotation.dart';
```

```
import 'FileItem.dart';
import 'chat/ChatMessageModel.dart';

part 'RequestModelDetail.g.dart';

@JsonSerializable()
class RequestModelDetail {
  String comment;
  String date;
  String date_modified;
  List<FileItem> files;
  int id;
  String status;
  double sum;
  String diagnos;
  List<ChatMessageModel> messages;

  RequestModelDetail(
      {required this.comment,
        required this.date,
        required this.date_modified,
        required this.files,
        required this.id,
        required this.status,
        required this.sum,
        required this.diagnos,
        required this.messages});

  factory RequestModelDetail.empty() {
    return RequestModelDetail(
        comment: "",
        date: "",
        date_modified: "",
        files: List<FileItem>.empty(),
        id: 0,
        status: "",
        sum: 0.0,
        diagnos: "",
        messages: List<ChatMessageModel>.empty()
    );
  }

      factory    RequestModelDetail.fromJson(Map<String,
dynamic> json) =>
    _$RequestModelDetailFromJson(json);

          Map<String,      dynamic>      toJson()      =>
_$RequestModelDetailToJson(this);
}
```

/data/remote/request/chat/ChatMessageModel.dart

```
import 'package:json_annotation/json_annotation.dart';

import 'UserModel.dart';

part 'ChatMessageModel.g.dart';

@JsonSerializable()
class ChatMessageModel {
  int id;

  UserModel member;

  UserModel user;

  @JsonKey(name: "message")
  String body;

        ChatMessageModel({required      this.id,      required
this.member,
    required this.user, required this.body});

      factory    ChatMessageModel.fromJson(Map<String,
dynamic> json) =>
    _$ChatMessageModelFromJson(json);

          Map<String,      dynamic>      toJson()      =>
_$ChatMessageModelToJson(this);
}
```

/data/remote/request/chat/ChatUserModel.dart

```
import 'package:json_annotation/json_annotation.dart';

part 'UserModel.g.dart';
```

```
@JsonSerializable()
class UserModel {
  int id;
  String first_name;
  String last_name;
  String middle_name;

    UserModel({required this.id, required this.first_name, required this.last_name, required this.middle_name});

  factory UserModel.fromJson(Map<String, dynamic> json) =>
      json != null
          ? _$UserModelFromJson(json)
              : UserModel(id:-1, first_name: "", last_name: "", middle_name: "");

              Map<String, dynamic> toJson() => _$UserModelToJson(this);
}
```

/data/remote/request/ApiService.dart

```
import 'dart:convert';
import 'dart:io';

import 'package:dio/dio.dart';
import 'package:l_kassa/constants/colors.dart';
import 'package:l_kassa/constants/const_values.dart';
import 'package:l_kassa/data/remote/auth/response/LoginResponse.dart';
import 'package:l_kassa/data/remote/auth/response/RegisterResponse.dart';
import 'package:l_kassa/data/remote/auth/response/UserProfileFilled.dart';
import 'package:l_kassa/data/remote/charges/response/ChargeResponse.dart';
import 'package:l_kassa/data/remote/payment/LiqPayModel.dart';
import 'package:l_kassa/data/remote/payment/response/PaymentResponse.dart';
import 'package:l_kassa/data/remote/payment/response/ConcordResponse.dart';
import 'package:l_kassa/data/remote/payment/response/LiqPayResponse.dart';
import 'package:l_kassa/data/remote/profile/ProfileModel.dart';
import 'package:l_kassa/data/remote/profile/response/EditProfileResponse.dart';
import 'package:l_kassa/data/remote/profile/response/ProfileResponse.dart';
import 'package:l_kassa/data/remote/profile/response/ProfileWrongResponse.dart';
import 'package:l_kassa/data/remote/program/response/ChangeProgramResponse.dart';
import 'package:l_kassa/data/remote/program/response/GetProgramResponse.dart';
import 'package:l_kassa/data/remote/request/response/RequestListResponse.dart';
import 'package:l_kassa/data/remote/request/response/RequestResponse.dart';
import 'package:l_kassa/service/ServiceLocator.dart';
import 'package:pretty_dio_logger/pretty_dio_logger.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'package:path/path.dart';

class ApiService {
        static const String SERVER = "https://test3.kozach.pp.ua";

    static const String SEND_SMS_URL = SERVER + "/user/signrequest";
   static const String CHECK_SMS_URL = SERVER + "/user/signin";
```

```
  static const String CREATE_PROFILE_URL = SERVER
+ "/user/profile/create";
    static const String PROFILE_URL = SERVER +
"/user/profile/get";
  static const String PROFILE_WRONG_URL = SERVER
+ "/user/profile/wrong";
  static const String PROFILE_EDIT_URL = SERVER +
"/user/profile/edit";

  static const String PROGRAM_LIST_URL = SERVER +
"/program/list";
    static const String PROGRAM_CHANGE_URL =
SERVER + "/program/change";

  static const String PAYMENTS_LIST_URL = SERVER +
"/user/payments/list";
  static const String LIQPAY_PAYLOAD_URL = SERVER
+ "/user/payments/liqpay";
    static const String CONCORD_PAYLOAD_URL =
SERVER + "/user/payments/concord";

  static const String CHARGE_LIST_URL = SERVER +
"/user/charges/list";

  static const String REQUEST_LIST_URL = SERVER +
"/user/request/list";
    static const String REQUEST_UPDATE_URL =
SERVER + "/user/request/update";
    static const String REQUEST_CREATE_URL =
SERVER + "/user/request/create";

    static const String DIIA_URL = SERVER +
"/user/profile/diia";

 String token = "";

 final Dio _authDio = Dio();
 final Dio _DataDio = Dio();

 SharedPreferences pref = locator<SharedPreferences>();

 ApiService() {
  _DataDio.interceptors.add(PrettyDioLogger(
   requestHeader: false,
   requestBody: true,
   responseBody: true,
   responseHeader: false,
   compact: true,
  ));

  token = pref.getString(AppConstants.TOKEN_KEY) ??
"";

  _DataDio.interceptors
          .add(InterceptorsWrapper(onRequest: (options,
handler) {
   String token =
     pref.getString(AppConstants.TOKEN_KEY) ?? "";
   if (token != "") {
    options.headers["Authorization"] = "Bearer " + token;
   }
   return handler.next(options);

  }, onResponse: (response, handler) {
   return handler.next(response);

  }, onError: (DioError e, handler) {
   print(
     "ERROR DIO url: ${e.requestOptions.path} \ncode:
${e.response!.statusCode}\n------");

   return handler.next(e);

  }));
 }

 Future<LoginResponse> login(String phone) async {
  try {
   var payload = {'create': false, 'phone': phone};

                Response response = await
_authDio.post(SEND_SMS_URL,
          data: payload, options: Options(responseType:
ResponseType.plain));

   var responseJson = json.decode(response.data);

   return LoginResponse(responseJson["message"], true);
  } on DioError catch (e) {
   if (e.response == null) {
    return LoginResponse("no_internet", false);
```

```
    } else {
     if (e.response?.statusCode == 500 ){
      return LoginResponse("", false);
     } else {
      var responseJson = json.decode(e.response?.data);
          return  LoginResponse(responseJson["message"],
false);
     }
    }
   }
  }

  Future<RegisterResponse> register(String phone) async {
   try {
    var payload = {'create': true, 'phone': phone};

                    Response response = await
_authDio.post(SEND_SMS_URL,
            data: payload, options: Options(responseType:
ResponseType.plain));
    print(response.statusCode);
    print(response.data);
    var responseJson = json.decode(response.data);
        return  RegisterResponse(responseJson["message"],
true);
   } on DioError catch (e) {
    print(e.message);
    print(e.response?.statusCode);
    print(e.response?.data);
    var responseJson = json.decode(e. response?.data);
        return  RegisterResponse(responseJson["message"],
false);
   }
  }

  Future<bool> sendCode(String phone, String code) async
{
   try {
    var payload = {'code': code, 'phone': phone};

                    Response response = await
_authDio.post(CHECK_SMS_URL,
            data: payload, options: Options(responseType:
ResponseType.plain));
    print(response.statusCode);
    print(response.data);
    final responseJson = json.decode(response.data);
                pref.setString(AppConstants.TOKEN_KEY,
responseJson["access_token"]);
    token = responseJson["access_token"];
    return true;
   } on DioError catch (e) {

    print("send code error");
    print(e.message);
    print(e.response?.data.toString());

    return false;
   }
  }

  Future<UserProfileFilled> createProfileBook(
    String firstName,
    String lastName,
    String middleName,
    String email,
    String IBAN,
    int programId,
    File inn,
    File book_1,
    File book_2,
    File book_address) async {
   var formData = FormData.fromMap({
    'first_name': firstName,
    'last_name': lastName,
    'middle_name': middleName,
    'email': email,
    'iban':IBAN,
    'program_id': programId,
     'inn': await MultipartFile.fromFile(inn.path, filename:
'inn.jpg'),
    'book_1':
       await MultipartFile.fromFile(book_1.path, filename:
'book_1.jpg'),
    'book_2':
       await MultipartFile.fromFile(book_2.path, filename:
'book_2.jpg'),
                              'book_address':  await
MultipartFile.fromFile(book_address.path,
      filename: 'book_address.jpg'),
   });
```

```
  try {
                    var response = await _DataDio.post(CREATE_PROFILE_URL, data: formData,options: Options(responseType: ResponseType.plain));
    print(response.statusCode);
    print(response.data);
    final responseJson = json.decode(response.data);
                    UserProfileFilled result = UserProfileFilled.fromJson(responseJson);
    return result;
  } on DioError catch (e) {
    print(e.message);
    print(e.response?.statusCode);
    print(e.response?.data);
      return UserProfileFilled.withError("something went wrong");
  }
 }

     Future<UserProfileFilled> createProfileId(String firstName, String lastName, String middleName,
      String email, String IBAN,int programId, File inn, File id_1, File id_2) async {
  var formData = FormData.fromMap({
    'first_name': firstName,
    'last_name': lastName,
    'middle_name': middleName,
    'email': email,
    'iban': IBAN,
    'program_id': programId,
      'inn': await MultipartFile.fromFile(inn.path, filename: 'inn.jpg'),
          'id_1': await MultipartFile.fromFile(id_1.path, filename: 'id_1.jpg'),
          'id_2': await MultipartFile.fromFile(id_2.path, filename: 'id_2.jpg'),
  });
  try {

                    var response = await _DataDio.post(CREATE_PROFILE_URL, data: formData, options: Options(responseType: ResponseType.plain));
    print(response.statusCode);
    print(response.data);
    final responseJson = json.decode(response.data);
                    UserProfileFilled result = UserProfileFilled.fromJson(responseJson);
    return result;
  } on DioError catch (e) {
    print(e.message);
    print(e.response?.statusCode);
    print(e.response?.data);
      return UserProfileFilled.withError("something went wrong");
  }
 }

 Future<GetProgramResponse> getPrograms() async {
  try {
                    Response response = await _DataDio.get(PROGRAM_LIST_URL,
                        options: Options(responseType: ResponseType.plain));
    List responseJson = json.decode(response.data);
                    GetProgramResponse result = GetProgramResponse.fromJson(responseJson);
    return result;
  } on DioError catch (error) {
                        final responseJson = json.decode(error.response!.data.toString());
                    GetProgramResponse result = GetProgramResponse.withError(responseJson);
    return result;
  } on Exception catch (e) {
    return GetProgramResponse.withError(e.toString());
  }
 }

  Future<GetProgramResponse> getAvailablePrograms() async {
  ProfileModel p = ProfileModel.fromStore();
  try {
                    Response response = await _DataDio.get(PROGRAM_LIST_URL,
                        options: Options(responseType: ResponseType.plain));
    List responseJson = json.decode(response.data);
                    GetProgramResponse result = GetProgramResponse.fromJson(responseJson);
```

```
                result.results.removeWhere((element) =>
element.program_id == p.program.program_id);
    return result;
   } on DioError catch (error) {
                              final responseJson =
json.decode(error.response!.data.toString());
                        GetProgramResponse result =
GetProgramResponse.withError(responseJson);
    return result;
   } on Exception catch (e) {
    return GetProgramResponse.withError(e.toString());
   }
  }

  Future<ChangeProgramResponse> changeProgram(int id)
async {
   try {
    FormData formData = FormData.fromMap({
     'program_id': id,
    });

                        Response response = await
_DataDio.post(PROGRAM_CHANGE_URL,
                            options: Options(responseType:
ResponseType.plain),
        data: formData);
    final responseJson = json.decode(response.data);
                      ChangeProgramResponse result =
ChangeProgramResponse.fromJson(responseJson);
    return result;
   } on DioError catch (error) {
                              final responseJson =
json.decode(error.response!.data.toString());
                      ChangeProgramResponse result =
ChangeProgramResponse.withError(responseJson);
    return result;
   } on Exception catch (e) {
                                                  return
ChangeProgramResponse.withError(e.toString());
   }
  }

  Future<ProfileResponse> getProfile() async {
   try {
                        Response response = await
_DataDio.get(PROFILE_URL,
                            options: Options(responseType:
ResponseType.json));
                Map<String, dynamic> responseJson =
Map.from(response.data);
                          ProfileResponse result =
ProfileResponse.fromJson(responseJson);
    return result;
   } on DioError catch (e) {
    if (e.response == null) {
     return ProfileResponse.withError("No internet.");
    } else {
                 Map<String, dynamic> responseJson =
Map.from(e.response?.data);
     ProfileResponse result =

ProfileResponse.withError(responseJson["message"]);
     return result;
    }
   } on Exception catch (e) {
    print("profile unexpectable error");
    return ProfileResponse.withError("Shit happens ...");
   }
  }

                          Future<ProfileWrongResponse>
requestEditWrongProfile(String firstName,
      String lastName, String middleName, String email,
String iban) async {
   try {
    var formData = FormData.fromMap({
     'first_name': firstName,
     'last_name': lastName,
     'middle_name': middleName,
     'email': email,
     'iban': iban
    });

    Response response =
          await _DataDio.post(PROFILE_WRONG_URL,
data: formData);
                Map<String, dynamic> responseJson =
Map.from(response.data);
                    ProfileWrongResponse result =
ProfileWrongResponse.fromJson(responseJson);
    return result;
   } on DioError catch (error) {
```

```
    if (error.response == null) {
                return ProfileWrongResponse.withError("No
internet.");
    } else {
                                final responseJson =
json.decode(error.response!.data);
      ProfileWrongResponse result =

ProfileWrongResponse.withError(responseJson["message"]
);
      return result;
    }
  } on Exception catch (e) {
    print("profile unexpectable error");
     return ProfileWrongResponse.withError("Shit happens
...");
  }
 }

 Future<EditProfileResponse> updateProfile(
    String email, String phone, File? photo) async {
  try {
    Map<String, dynamic> requestData = {
     'email': email,
     'phone': phone,
    };
    if (photo != null) {
     requestData["photo"] =
         await MultipartFile.fromFile(photo.path, filename:
'photo.jpg');
    }
    var formData = FormData.fromMap(requestData);

                    Response response = await
_DataDio.post(PROFILE_EDIT_URL, data: formData);
              Map<String, dynamic> responseJson =
Map.from(response.data);
                    EditProfileResponse result =
EditProfileResponse.fromJson(responseJson);
    return result;
  } on DioError catch (error) {
    if (error.response == null) {
     return EditProfileResponse.withError("No internet.");
    } else {
                                final responseJson =
json.decode(error.response!.data);
      EditProfileResponse result =

EditProfileResponse.withError(responseJson["message"]);
      return result;
    }
  } on Exception catch (e) {
    print("profile unexpectable error");
      return EditProfileResponse.withError("Shit happens
...");
  }
 }

 Future<PaymentsResponse> getPayments() async {
  try {
                    Response response = await
_DataDio.get(PAYMENTS_LIST_URL,
                        options: Options(responseType:
ResponseType.plain));
    List responseJson = json.decode(response.data);
                    PaymentsResponse result =
PaymentsResponse.fromJson(responseJson);
    return result;
  } on DioError catch (error) {
                                final responseJson =
json.decode(error.response!.data.toString());
                    PaymentsResponse result =
PaymentsResponse.withError(responseJson);
    return result;
  } on Exception catch (e) {
    return PaymentsResponse.withError(e.toString());
  }
 }

 Future<ChargeResponse> getCharges() async {
  try {
                    Response response = await
_DataDio.post(CHARGE_LIST_URL,
                        options: Options(responseType:
ResponseType.plain), data: {});
    List responseJson = json.decode(response.data);
                    ChargeResponse result =
ChargeResponse.fromJson(responseJson);
    return result;
  } on DioError catch (error) {
                    final responseJson =
json.decode(error.response!.data.toString());
```

```
                ChargeResponse result =
ChargeResponse.withError(responseJson);
    return result;
   } on Exception catch (e) {
    return ChargeResponse.withError(e.toString());
   }
  }

    Future<ChargeResponse> getFilteredCharges(String
begin, String end) async {
   try {
                Response response = await
_DataDio.post(CHARGE_LIST_URL,
                options: Options(responseType:
ResponseType.plain),
      data: {"begin": begin, "end": end});
    List responseJson = json.decode(response.data);
                ChargeResponse result =
ChargeResponse.fromJson(responseJson);
    return result;
   } on DioError catch (error) {
                final responseJson =
json.decode(error.response!.data.toString());
                ChargeResponse result =
ChargeResponse.withError(responseJson);
    return result;
   } on Exception catch (e) {
    return ChargeResponse.withError(e.toString());
   }
  }

  Future<RequestListResponse> getRequests() async {
   try {
                Response response = await
_DataDio.post(REQUEST_LIST_URL,
                options: Options(responseType:
ResponseType.plain), data: {});
    List responseJson = json.decode(response.data);
                RequestListResponse result =
RequestListResponse.fromJson(responseJson);
    return result;
   } on DioError catch (error) {
    print(error.response!.data);
                final responseJson =
json.decode(error.response!.data.toString());
                RequestListResponse result =
RequestListResponse.withError(responseJson);
    return result;
   } on Exception catch (e) {
    return RequestListResponse.withError(e.toString());
   }
  }

                Future<RequestListResponse>
getFilteredRequests(List<String> statuses) async {
   try {
                Response response = await
_DataDio.post(REQUEST_LIST_URL,
                options: Options(responseType:
ResponseType.plain),
      data: {"status": statuses});
    List responseJson = json.decode(response.data);
                RequestListResponse result =
RequestListResponse.fromJson(responseJson);
    return result;
   } on DioError catch (error) {
    print(error.response!.data);
                final responseJson =
json.decode(error.response!.data.toString());
                RequestListResponse result =
RequestListResponse.withError(responseJson);
    return result;
   } on Exception catch (e) {
    return RequestListResponse.withError(e.toString());
   }
  }

  Future<RequestResponse> createRequest(
    String comment, List<File> files) async {
   FormData formData = FormData.fromMap({'comment':
comment});

   for (int i = 0; i < files.length; i++) {
    String filename = basename(files[i].path);
          MapEntry<String, MultipartFile> value =
MapEntry("file_${i}",
        await MultipartFile.fromFile(files[i].path, filename:
filename));
    formData.files.add(value);
   }
   try {
```

```
                var response = await
_DataDio.post(REQUEST_CREATE_URL, data:
formData);
            Map<String, dynamic> responseJson =
Map.from(response.data);
                RequestResponse result =
RequestResponse.fromJson(responseJson);
    return result;
  } on DioError catch (e) {
            Map<String, dynamic> responseJson =
Map.from(e.response?.data);
    RequestResponse result =

RequestResponse.withError(responseJson["message"]);
    return result;
  }
 }

 Future<RequestResponse> updateRequest(int id, File file)
async {
  FormData formData = FormData.fromMap({'id': id});
  String filename = basename(file.path);
  MapEntry<String, MultipartFile> value = MapEntry(
        "file", await MultipartFile.fromFile(file.path,
filename: filename));
  formData.files.add(value);

  try {
                var response = await
_DataDio.post(REQUEST_UPDATE_URL, data:
formData);
            Map<String, dynamic> responseJson =
Map.from(response.data);
                RequestResponse result =
RequestResponse.fromJson(responseJson);
    return result;
  } on DioError catch (e) {
            Map<String, dynamic> responseJson =
Map.from(e.response?.data);
    RequestResponse result =

RequestResponse.withError(responseJson["message"]);
    return result;
  }
 }

  Future<LiqPayResponse> getLiqPayBody(int amount)
async {
  try {
    var payload = {'amount': amount};

                Response response = await
_DataDio.post(LIQPAY_PAYLOAD_URL,
        data: payload, options: Options(responseType:
ResponseType.plain));
    final responseJson = json.decode(response.data);
                LiqPayResponse result =
LiqPayResponse.fromJson(responseJson);
    return result;
  } on DioError catch (error) {
                final responseJson =
json.decode(error.response!.data.toString());
                LiqPayResponse result =
LiqPayResponse.withError(responseJson);
    return result;
  } on Exception catch (e) {
    return LiqPayResponse.withError(e.toString());
  }
 }

 Future<ConcordResponse> getConcordBody(int amount)
async {
  try {
    var payload = {'amount': amount};

                Response response = await
_DataDio.post(CONCORD_PAYLOAD_URL,
        data: payload, options: Options(responseType:
ResponseType.plain));
    final responseJson = json.decode(response.data);
                ConcordResponse result =
ConcordResponse.fromJson(responseJson);
    return result;
  } on DioError catch (error) {
                final responseJson =
json.decode(error.response!.data.toString());
                ConcordResponse result =
ConcordResponse.withError(responseJson);
    return result;
  } on Exception catch (e) {
    return ConcordResponse.withError(e.toString());
  }
```

```
  }

  Future<String> getDiiaDeeplink() async {
    try {
      Response response = await _DataDio.post(DIIA_URL,
                                options:    Options(responseType:
ResponseType.plain));
      final responseJson = json.decode(response.data);
      String result = responseJson["deeplink"];
      return result;
    } on DioError catch (error) {
                                    final       responseJson       =
json.decode(error.response!.data.toString());

      return "Error";
    } on Exception catch (e) {
      return "Error";
    }
  }
}
```

/data/repository/AuthRepository.dart

```
import 'dart:io';

import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:l_kassa/constants/const_values.dart';
import
'package:l_kassa/cubit/registration_setup/registration_setup
_state.dart';
import
'package:l_kassa/data/remote/auth/response/LoginResponse
.dart';
import
'package:l_kassa/data/remote/auth/response/UserProfileFill
ed.dart';
import
'package:l_kassa/data/remote/profile/response/ProfileRespo
nse.dart';

import 'package:l_kassa/service/ServiceLocator.dart';
import 'package:l_kassa/data/remote/ApiService.dart';

import '../remote/auth/response/RegisterResponse.dart';

class AuthRepository {
  final ApiService _apiProvider = locator<ApiService>();

  Future<LoginResponse> login(String phone) {
    return _apiProvider.login(phone);
  }

  Future<RegisterResponse> register(String phone) {
    return _apiProvider.register(phone);
  }

  Future<bool> sendCode(String phone, String code) {
    return _apiProvider.sendCode(phone, code);
  }

  Future<UserProfileFilled> createProfileBook(
      String firstName,
      String lastName,
      String middleName,
      String email,
      String IBAN,
      int programId,
      File inn,
      File book_1,
      File book_2,
      File book_address) {
    return _apiProvider.createProfileBook(
      firstName,
      lastName,
      middleName,
      email,
      IBAN,
      programId,
      inn,
      book_1,
      book_2,
      book_address,
    );
  }

      Future<UserProfileFilled>      createProfileId(String
firstName, String lastName, String middleName,
      String email,String IBAN, int programId, File inn, File
id_1, File id_2) {
    return _apiProvider.createProfileId(
```

```
        firstName, lastName, middleName, email,IBAN,
programId, inn, id_1, id_2);
  }
}
```

/data/repository/ChargeRepository.dart

```
import
'package:l_kassa/data/remote/charges/response/ChargeResp
onse.dart';

import 'package:l_kassa/service/ServiceLocator.dart';
import 'package:l_kassa/data/remote/ApiService.dart';
class ChargeRepository{

  final ApiService _apiProvider = locator<ApiService>();

  Future<ChargeResponse> getCharges(){
    return _apiProvider.getCharges();
  }
}
```

/data/repository/PaymentRepository.dart

```
import
'package:l_kassa/data/remote/payment/response/PaymentR
esponse.dart';
import 'package:l_kassa/service/ServiceLocator.dart';
import 'package:l_kassa/data/remote/ApiService.dart';
class PaymentRepository{

  final ApiService _apiProvider = locator<ApiService>();

  Future<PaymentsResponse> getPayments(){
    return _apiProvider.getPayments();
  }
}
```

/data/repository/ProfileRepository.dart

```
import 'dart:io';

import
'package:l_kassa/data/remote/profile/response/EditProfileR
esponse.dart';
import
'package:l_kassa/data/remote/profile/response/ProfileRespo
nse.dart';
import
'package:l_kassa/data/remote/profile/response/ProfileWron
gResponse.dart';

import 'package:l_kassa/service/ServiceLocator.dart';
import 'package:l_kassa/data/remote/ApiService.dart';

class ProfileRepository {
  final ApiService _apiProvider = locator<ApiService>();

  Future<ProfileResponse> getProfile() {
    return _apiProvider.getProfile();
  }

  Future<ProfileWrongResponse> editWrongProfile(
      String firstName, String lastName, String middleName,
String email, String iban) {
    return _apiProvider.requestEditWrongProfile(
        firstName, lastName, middleName, email, iban);
  }

   Future<EditProfileResponse> editProfile(String phone,
String email, File? photo){
    return _apiProvider.updateProfile(email, phone, photo);
  }
}
```

/data/repository/ProgramRepository.dart

```
import
'package:l_kassa/data/remote/program/response/GetProgra
mResponse.dart';
import 'package:l_kassa/service/ServiceLocator.dart';
import 'package:l_kassa/data/remote/ApiService.dart';
class ProgramRepository{
```

```
  final ApiService _apiProvider = locator<ApiService>();

  Future<GetProgramResponse> getPrograms(){
    print("get news repository call");
    return _apiProvider.getPrograms();
  }
}
```

/data/repository/RequestRepository.dart

```
import 'package:l_kassa/service/ServiceLocator.dart';
import 'package:l_kassa/data/remote/ApiService.dart';

import
'../remote/request/response/RequestListResponse.dart';
class RequestRepository{

  final ApiService _apiProvider = locator<ApiService>();

  Future<RequestListResponse> getRequests(){
    return _apiProvider.getRequests();
  }
}
```

/service/ServiceLocator.dart

```
import 'package:get_it/get_it.dart';
import 'package:l_kassa/data/remote/ApiService.dart';
import
'package:l_kassa/data/repository/AuthRepository.dart';
import
'package:l_kassa/data/repository/ChargeRepository.dart';
import
'package:l_kassa/data/repository/PaymentRepository.dart';
import
'package:l_kassa/data/repository/ProfileRepository.dart';
import
'package:l_kassa/data/repository/ProgramRepository.dart';
import
'package:l_kassa/data/repository/RequestRepository.dart';
import
'package:shared_preferences/shared_preferences.dart';
import
'package:solid_bottom_sheet/solid_bottom_sheet.dart';

GetIt locator = GetIt.instance;


setupLocator () async {
  locator.allowReassignment = true;
          SolidController bottomSheetcontroller =
SolidController();

locator.registerSingleton<SolidController>(bottomSheetcon
troller);


          SharedPreferences prefs = await
SharedPreferences.getInstance();
  locator.registerSingleton<SharedPreferences>(prefs);

  ApiService _api = ApiService();
  locator.registerSingleton<ApiService>(_api);

          ProgramRepository programRepository =
ProgramRepository();

locator.registerSingleton<ProgramRepository>(programRe
pository);

          ProfileRepository profileRepository =
ProfileRepository();

locator.registerSingleton<ProfileRepository>(profileReposi
tory);

  AuthRepository authRepository = AuthRepository();

locator.registerSingleton<AuthRepository>(authRepository
);

          PaymentRepository paymentRepository =
PaymentRepository();

locator.registerSingleton<PaymentRepository>(paymentRe
pository);
```

```
        ChargeRepository chargeRepository = ChargeRepository();

locator.registerSingleton<ChargeRepository>(chargeRepository);

        RequestRepository requestRepository = RequestRepository();

locator.registerSingleton<RequestRepository>(requestRepository);

}
```

/main.dart

```
import 'dart:io';
import 'dart:ui';

import 'dart:async';

import 'package:app_links/app_links.dart';
import 'package:easy_localization/easy_localization.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:l_kassa/cubit/main/bottonsheet/bottomsheet_state.dart';
import 'package:l_kassa/cubit/main/payment/payment_state.dart';
import 'package:l_kassa/cubit/main/program/program_state.dart';
import 'package:l_kassa/cubit/main/requets/filter/request_filter_state.dart';
import 'package:l_kassa/cubit/main/requets/request_state.dart';
import 'package:l_kassa/cubit/profile/profile_photo_state.dart';
import 'package:l_kassa/cubit/registration_setup/registration_setup_state.dart';
import 'package:l_kassa/ui/auth/registration/RegistrationPage.dart';
import 'package:l_kassa/ui/auth/registration/RegistrationSuccessfulPage.dart';
import 'package:l_kassa/ui/auth/registration/SelectProgramPage.dart';
import 'package:l_kassa/ui/contacts/ContactsPage.dart';
import 'package:l_kassa/ui/intro/IntroPage.dart';
import 'package:l_kassa/ui/splash/SplashScreen.dart';

import 'constants/colors.dart';
import 'cubit/main/charge/charge_state.dart';
import 'cubit/profile/profile_state.dart';
import 'cubit/splash/splash_state.dart';
import 'service/ServiceLocator.dart';
import 'ui/auth/login/LoginPage.dart';
import 'ui/auth/registration/SetupProfilePage.dart';
import 'ui/main/HomePage.dart';

Future<void> main() async {
  print("start app...");



  WidgetsFlutterBinding.ensureInitialized();
  await setupLocator();
  await EasyLocalization.ensureInitialized();
  print("locator setted up");


  runApp(EasyLocalization(child: LKassaApp(),

    supportedLocales: const [
      Locale("uk"),
      Locale('ru')

    ],
    useOnlyLangCode: true,
    useFallbackTranslations: true,
    path: 'assets/languages',
    fallbackLocale:  Locale('uk'),));
}

class LKassaApp extends StatefulWidget {
  const LKassaApp({Key? key}) : super(key: key);
```

```
  @override
  State<LKassaApp> createState() => _LKassaAppState();
}

class _LKassaAppState extends State<LKassaApp> {

  final _navigatorKey = GlobalKey<NavigatorState>();
  late AppLinks _appLinks;
  StreamSubscription<Uri>? _linkSubscription;

  Future<void> initDeepLinks() async {
    _appLinks = AppLinks();
    final appLink = await _appLinks.getInitialLink();
    if (appLink != null) {
      print('getInitialAppLink: $appLink');
    }

                                        _linkSubscription          =
_appLinks.uriLinkStream.listen((uri) {
      print('onAppLink: $uri');
      openAppLink(uri);
    });
  }

  void openAppLink(Uri uri) {

_navigatorKey.currentState?.pushNamed("/registration_co
mpleted");
  }


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

  @override
  void dispose() {
    _linkSubscription?.cancel();

    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    print("app build");

    initDeepLinks();
    return MultiBlocProvider(
      providers: [
        BlocProvider<RegistrationSetupCubit>(
                          create:  (BuildContext  context)  =>
RegistrationSetupCubit(),
        ),

        BlocProvider<SplashCubit>(
          create: (BuildContext context) => SplashCubit(),
        ),

        BlocProvider<ProgramCubit>(
          create: (BuildContext context) => ProgramCubit(),
        ),
        BlocProvider<BottomSheetCubit>(
                          create:  (BuildContext  context)  =>
BottomSheetCubit(),
        ),
        BlocProvider<ProfileCubit>(
          create: (BuildContext context) => ProfileCubit(),
        ),
        BlocProvider<ProfilePhotoCubit>(
                          create:  (BuildContext  context)  =>
ProfilePhotoCubit(),
        ),
        BlocProvider<RequestCubit>(
          create: (BuildContext context) => RequestCubit(),
        ),
        BlocProvider<RequestFilterCubit>(
                          create:  (BuildContext  context)  =>
RequestFilterCubit(),
        ),

        BlocProvider<ChargeCubit>(
          create: (BuildContext context) => ChargeCubit(),
        ),
        BlocProvider<PaymentCubit>(
          create: (BuildContext context) => PaymentCubit(),
        ),
      ],
      child: MaterialApp(
        navigatorKey: _navigatorKey,
        title: 'LKassa',
```

```
      theme: ThemeData(
        primarySwatch: AppColors.primary,
      ),

      localizationsDelegates: context.localizationDelegates,
      supportedLocales: context.supportedLocales,
      locale: context.locale,

      initialRoute: '/',
      routes: {
        '/': (context) => const SplashScreen(),
        '/intro': (context) => IntroPage(),
        '/login': (context) => const LoginPage(),
        '/contacts': (context) => ContactsPage(),
        '/registration': (context) => const RegistrationPage(),
                          '/registration_profile': (context) =>
SetupProfilePage(),
                  '/registration_select_program': (context) =>
SelectProgramPage(),
                        '/registration_completed': (context) =>
RegistrationSuccessfulPage(),
        '/home': (context) => HomePage(),
      },
    ),
   );
  }
}
```

/cubit/main/bottonsheet/bottonsheet_cubit.dart

```
class BottomSheetCubit extends
Cubit<BottomSheetState>{
  BottomSheetCubit() : super(InitialBottomSheetState());

  bool showChooseLanguage = true;


  bool getShowChooseLanguage(){
    return showChooseLanguage;
  }


  void emitFilterRequest(){
    emit(FilterRequestBottomSheetState(maxHeight: 200));
  }

  void emitLanguage(){
    emit(LanguageBottomSheetState(maxHeight: 350));
  }

  void emitFilterCharge(){
    emit(FilterChargeBottomSheetState(maxHeight: 250));
  }

}
```

/cubit/main/bottonsheet/bottonsheet_state.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

part 'bottomsheet_cubit.dart';

@immutable
abstract class BottomSheetState {}

class InitialBottomSheetState extends BottomSheetState {}

class LanguageBottomSheetState extends
BottomSheetState {
  final int maxHeight;
  LanguageBottomSheetState({required this.maxHeight});

  @override
  List<Object> get props => [maxHeight];
}

class FilterRequestBottomSheetState extends
BottomSheetState {
  final int maxHeight;
                FilterRequestBottomSheetState({required
this.maxHeight});

  @override
  List<Object> get props => [maxHeight];
}
```

```
class FilterChargeBottomSheetState extends BottomSheetState {
  final int maxHeight;
                    FilterChargeBottomSheetState({required this.maxHeight});

  @override
  List<Object> get props => [maxHeight];
}
```

/cubit/main/charge/charge_cubit.dart

```
class ChargeCubit extends Cubit<ChargeState> {
  ChargeCubit() : super(ChargeInitState());

  String begin = "";
  String end = "";

  void resetFilter(){
    begin = "";
    end = "";
  }

  void emitLoadCharges() async {
    emit(ChargeLoadingState());
                ChargeResponse response = await locator<ApiService>().getCharges();
    emit(ChargeLoadState(chargeList: response.results));
  }

  void emitLoadFilteredCharges() async {
    emit(ChargeLoadingState());
                ChargeResponse response = await locator<ApiService>().getFilteredCharges(begin, end);
    emit(ChargeLoadState(chargeList: response.results));
  }
}
```

/cubit/main/charge/charge_state.dart

```
import 'package:flutter/cupertino.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:l_kassa/data/remote/ApiService.dart';
import 'package:l_kassa/data/remote/charges/ChargeModel.dart';
import 'package:l_kassa/service/ServiceLocator.dart';

import '../../../data/remote/charges/response/ChargeResponse.dart';
import '../../../data/remote/request/RequestModel.dart';
import '../../../data/remote/request/response/RequestListResponse.dart';

part 'charge_cubit.dart';

@immutable
abstract class ChargeState {}

class ChargeInitState extends ChargeState {}
class ChargeLoadingState extends ChargeState {}

class ChargeLoadState extends ChargeState {

  final List<ChargeModel> chargeList;
  ChargeLoadState({required this.chargeList});

  @override
  List<Object> get props => [chargeList];
}
```

/cubit/main/payment/payment_cubit.dart

```
class PaymentCubit extends Cubit<PaymentState> {
  PaymentCubit() : super(PaymentLoadingState());

  List<PaymentModel> srcList = List.empty();

  String begin = "";
  String end = "";

  loadPayments() async {
    emit(PaymentLoadingState());
    PaymentsResponse response =
        await locator<PaymentRepository>().getPayments();
    srcList = response.results;
    emit(PaymentLoadedState(paymentList: srcList));
  }

  filterResults() {
```

```
   emit(PaymentLoadingState());

                              DateTime dateFrom =
DateFormat('dd.MM.yyyy').parse(begin);
                               DateTime dateTo =
DateFormat('dd.MM.yyyy').parse(end);

       dateTo.add(const Duration(hours: 23, minutes: 59,
seconds: 59));

    List<PaymentModel> dstList = srcList.where((element)
{
                                  DateTime actual =
DateFormat('dd.MM.yyyy').parse(element.date);
                      return actual.isAfter(dateFrom) &&
actual.isBefore(dateTo);
   }).toList();

   emit(PaymentLoadedState(paymentList: dstList));
  }

  resetFilters() async {
   emit(PaymentLoadingState());

   begin = "";
   end = "";

   if (srcList.isEmpty) {
    PaymentsResponse response =
        await locator<PaymentRepository>().getPayments();
    srcList = response.results;
   }

   emit(PaymentLoadedState(paymentList: srcList));
  }
}
```

/cubit/main/payment/payment_state.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import
'package:l_kassa/data/remote/payment/PaymentModel.dart'
;
import
'package:l_kassa/data/repository/PaymentRepository.dart';
import 'package:l_kassa/service/ServiceLocator.dart';
import 'package:intl/intl.dart';

import
'../../../data/remote/payment/response/PaymentResponse.dar
t';

part 'payment_cubit.dart';

@immutable
abstract class PaymentState {}


class PaymentLoadingState extends PaymentState {}

class PaymentLoadedState extends PaymentState {

  final List<PaymentModel> paymentList;
  PaymentLoadedState({required this.paymentList});

  @override
  List<Object> get props => [paymentList];

}
```

/cubit/main/program/program_cubit.dart

```
class ProgramCubit extends Cubit<ProgramState>{
  ProgramCubit() : super(ProgramInfoState());


  void navigate(BuildContext context) {
    Navigator.push(
      context,
      MaterialPageRoute(
            builder: (context) => const MyHomePage(title:
"Cubit !!!")),
    );
  }
}
```

/cubit/main/program/program_state.dart

```
import 'package:flutter/material.dart';
```

```
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:l_kassa/ui/sample/MyHomePage.dart';

part 'program_cubit.dart';

@immutable
abstract class ProgramState {}

class ProgramInfoState extends ProgramState {}

class ProgramSelectState extends ProgramState {}
```

/cubit/main/requests/requests_cubit.dart

```
class RequestCubit extends Cubit<RequestState> {
 RequestCubit() : super(InitialRequestState());

 void emitLoadRequests() async {
  emit(LoadingRequestState());
            RequestListResponse response = await
locator<ApiService>().getRequests();
  emit(RequestLoadState(requestList: response.results));
 }

   void emitLoadFilteredRequests(List<String> statuses)
async {
  emit(LoadingRequestState());
  RequestListResponse response =
                                                  await
locator<ApiService>().getFilteredRequests(statuses);
  emit(RequestLoadState(requestList: response.results));
 }
}
```

/cubit/main/requests/requests_state.dart

```
import 'package:flutter/cupertino.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:l_kassa/data/remote/ApiService.dart';
import 'package:l_kassa/service/ServiceLocator.dart';

import '../../../data/remote/request/RequestModel.dart';
import
'../../../data/remote/request/response/RequestListResponse.d
art';

part 'request_cubit.dart';

@immutable
abstract class RequestState {}

class InitialRequestState extends RequestState {}

class LoadingRequestState extends RequestState {}

class RequestLoadState extends RequestState {

 final List<RequestModel> requestList;
 RequestLoadState({required this.requestList});

 @override
 List<Object> get props => [requestList];
}
```

/cubit/main/requests/filter/request_filter_cubit.dart

```
class RequestFilterCubit extends
Cubit<RequestFilterState>{
 RequestFilterCubit() : super(RequestFilterInitState());

 List<String> selected = [];

 void addSelected(String item)async{
  selected.add(item);
  emit(RequestSelectedFilterState(selected: selected));
 }

 void deleteSelected(String item) {
  selected.remove(item);
  emit(RequestSelectedFilterState(selected: selected));
 }

 void clearSelected(){
  selected.clear();
  emit(RequestSelectedFilterState(selected: selected));
 }
}
```

/cubit/main/requests/filter/request_filter_state.dart

```
import 'package:flutter/cupertino.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

part 'request_filter_cubit.dart';

@immutable
abstract class RequestFilterState {}

class RequestFilterInitState extends RequestFilterState {

  final List<String> selected = List<String>.empty();

  RequestFilterInitState();

  @override
  List<Object> get props => [selected];
}

class RequestSelectedFilterState extends RequestFilterState
{

  final List<String> selected;
  RequestSelectedFilterState({required this.selected});

  @override
  List<Object> get props => [selected];
}
```