Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Мобільний застосунок для ведення бюджету

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-4
Іброхімбек БОЯКОВ
(власне ім'я, прізвище)

Спеціальність
123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп'ютерна інженерія
(повна назва освітньої програми)

Керівник: доц. Юрій КОЛТУН
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Боякову Іброхімбеку Саідмуродовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Мобільний застосунок для ведення бюджету

затверджена наказом по університету від “ 26 ” травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи

1) використання базових архітектурних підходів та принципів написання коду;
2) логіка взаємодії з базою даних SQLite;
3) використання спеціальних плагінів та пакетів для підвищення функціональності;
4) привабливий та функціональний інтерфейс та комфортний досвід користувача.

4. Перелік питань, що потрібно опрацювати у роботі

1) аналіз предметної області;
2) аналіз технологій, що використовуються при розробці мобільного застосунку;
3) опис програмної реалізації застосунку;
4) інструкція користувача;
5) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд-презентація – 10 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
	Аналіз задачі та огляд існуючих рішень	27.05.25-29.05.25	
	Вибір технологій та інструментів розробки	30.05.25-31.05.25	
	Проєктування архітектури застосунку	01.06.25-03.06.25	
	Реалізація головних функцій застосунку	04.06.25-07.06.25	
	Тестування застосунку	08.05.25-10.06.25	
	Оформлення матеріалів кваліфікаційної роботи	11.06.25-12.06.25	
	Подання кваліфікаційної роботи керівникам	13.06.25-14.06.25	
	для попереднього захисту		
	Подання кваліфікаційної роботи на	15.06.25-16.06.25	
	рецензування		

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Юрій КОЛТУН
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 64 с., 19 рис., 1 дод., 14 джерел.

ВІДЖЕТИ, ІНТЕРФЕЙС КОРИСТУВАЧА, ДОСВІД КОРИСТУВАЧА, НАВІГАЦІЯ, АСИНХРОННИЙ КОД, FLAT ARCHITECTURE, ТЕСТУВАННЯ, SQLITE, ПЛАГІНИ, ЛОКАЛЬНЕ СХОВИЩЕ.

Метою кваліфікаційної роботи є розробка мобільного застосунку для ведення власного бюджету у середовищі розробки Visual Studio Code. У ході розробки було використано фреймворк Flutter, мову програмування Dart: його пакети та плагіни, SQLite для зберігання даних.

Результатом є працездатний мобільний застосунок, який надає користувачам можливість додавати транзакції, і на основі цих даних формує щоденні, щотижневі діаграми та звіти за цілий місяць. Крім цього, користувач може встановити ціль накопичення і власноруч "класти гроші в скарбничку". Кодова база проєкту побудована з використанням базових підходів, що забезпечує легку підтримку та розширення функціональності застосунку.

ABSTRACT

Bachelor’s thesis: 64 pages, 19 figures, 1 appendice, 14 sources.

WIDGETS, USER INTERFACE, USER EXPERIENCE, NAVIGATION, ASYNCHRONOUS CODE, FLAT ARCHITECTURE, TESTING, SQLITE, PLAGINS, LOCAL STORAGE.

The major goal of this thesis is the development of a mobile application for personal budget tracking using the Visual Studio Code development environment. Flutter framework and Dart programming language with its packages and plugins were used during the development, as well as SQLite for data storage.

The result is fully functional mobile application that allows users to add transations and, based on this data, generates daily, weekly charts and monthly reports. Besides user can set a savings goal and manually "put money into a piggy bank". The project’s codebase is built using fundamental development approaches, which ensures easy maintenance and extensibility of the application’s functionality.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ОС – операційна система

ШІ – штучний інтелект

AI – штучний інтелект (англ., Artificial Intelligence)

DB – база даних (англ., Database)

GPS – глобальна система позиціонування (англ., Global Positioning System)

GPT – мовна модель, розроблена OpenAI (англ., Generative Pre-trained Transformer)

IOS – операційна система iPhone від Apple (англ., iPhone Operating System)

IT – інформаційні технології (англ., Information Technology)

OS – операційна система (англ., Operating System)

SPR – принцип єдиної відповідальності (англ., Single Responsibility Principle)

SQL – мова структурованих запитів (англ., Structured Query Language)

ВСТУП

Програмні продукти, спрямовані на оптимізацію фінансового стану користувача, набувають все більшої актуальності в час стрімкого розвитку технологій та цифровізації повсякденного життя. Одним із таких напрямків є створення застосунків для ведення особистого бюджету [1], які дозволяють користувачам ефективно управляти власними фінансами, планувати бюджет та досягати фінансових цілей.

Сьогодні на ринку існує велика кількість рішень у сфері управління особистими фінансами [2]. Найпопулярніші програми надають користувачам функції для автоматичного запису транзакцій, створення звітів та аналітики, а також встановлення фінансових цілей. Світові тенденції вказують на інтеграцію цих продуктів із банківськими послугами, використанням штучного інтелекту для персоналізованих рекомендацій та впровадження трендових елементів для підвищення зацікавленості користувачів.

Актуальність роботи полягає в необхідності створення застосунку, який би поєднував функції фінансового обліку з можливістю накопичення коштів для досягнення власних цілей. У сучасних економічних умовах, коли раціональне управління фінансами стає все більш важливим фактором для забезпечення добробуту споживачів, подібний інструмент може бути корисним для широкої аудиторії.

Метою кваліфікаційної роботи є розробка інтуїтивного та функціонального мобільного застосунку для слідкування за особистим бюджетом з функцією "скарбничка", що дозволить користувачам не лише слідкувати за доходами та витратами, але й ефективно планувати та накопичувати кошти. Сфера застосування такого продукту охоплює як приватне використання, так і потенційну інтеграцію з бухгалтерськими платформами для зручного використання компаніями.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність теми

Ведення обліку за власними фінансами бере початок ще з давніх часів, адже як і зараз контроль над власними ресурсами грав ключову роль у забезпеченні добробуту. Перші форми обліку мали застосування не тільки на рівні країни, а й у повсякденному житті людини. Наприклад, в стародавньому Єгипті писарі фіксували обсяги зерна в коморах [3], а в стародавньому Римі заможні громадяни користувалися "codices accepti et expensi" (у перекладі з латинської – "реєстр отриманого і витраченого") [4] – книгою, до якої вносили записи про свої доходи та витрати.

Традиція накопичення не є винятком, й також має дуже давні корені. Найперші практики заощадження виходили далеко за межі накопичення грошей. Вони зосереджувалися на збереженні життєво важливих ресурсів, щоб забезпечити виживання під час нестачі. З часом, люди почали накопичувати коштовності: метали, каміння чи інші речі, які мали велику цінність, а з появою грошей і їх, аби забезпечити собі та своїй родині безтурботне майбутнє, або для досягнення певної мети. Зберігалось це все у скарбничках – ємностях, які були виготовлені в основному з глини, каміння, кераміки, бронзи або міді.

Стрімкий розвиток технологій, ринкової економіки та суспільства в цілому впровадило цілу низку інноваційних товарів та послуг, які кардинально змінили та більшою мірою покращили життя людей. Водночас це стало причиною підвищення важливості обліку фінансів. Якщо в минулому потреби більшості населення не перевищували базових життєвих потреб – їжі, одягу, житла, то зараз цей діапазон значно розширився, як, подарунки, подорожі, гаджети, прикраси та велика кількість інших видів послуг, які не зупиняються з’являтися майже кожен новий день. Все це розмаїття продуктів створює тиск на бюджет середньостатистичних

споживачів. Отже, слідкування за власними витратами є надійним рішенням для запобігання здійснення необдуманих витрат та забезпечення більш стабільного рівня життя.

Відповідно, ці зміни також вплинули на ситуацію із накопиченням коштів. Якщо раніше люди робили це з більш довгостроковою метою: для майбутнього дітей, на випадки кризових ситуацій, на чорний день, то сьогодні це частіше робиться для задоволення потреб короткострокових: купівля рухомого або нерухомого майна, придбання товарів чи послуг, які значно перевищують поточні фінансові можливості людини.

Паралельно такому стрімкому розвитку людство споконвіку стикається з кризами: економічними, політичними, територіальними та іншими. Поточна геополітична ситуація, сформована за останні століття, зробила наслідки таких катаклізмів більш масштабними. Через глобальну мережу взаємопов’язаних політичних та економічних відносин навіть малий прояв нестабільності чи кризи будь-де впливатиме на загальний стан всього світу. Цей важливий фактор змушує людей слідкувати та вести облік своїх фінансів, розпоряджатися ними розумно. Можливо, сьогодні люди, як ніколи раніше, повинні думати про завтрашній день і планувати свої витрати так, щоб забезпечити собі стабільний рівень життя в майбутньому. Це стосується також і задоволення своїх потреб. Через волатильність ринку сучасному середньостатистичному споживачу краще накопичувати кошти, аніж необдумано витрачати на всі потреби, що допоможе уникнути фінансово невигідних витрат.

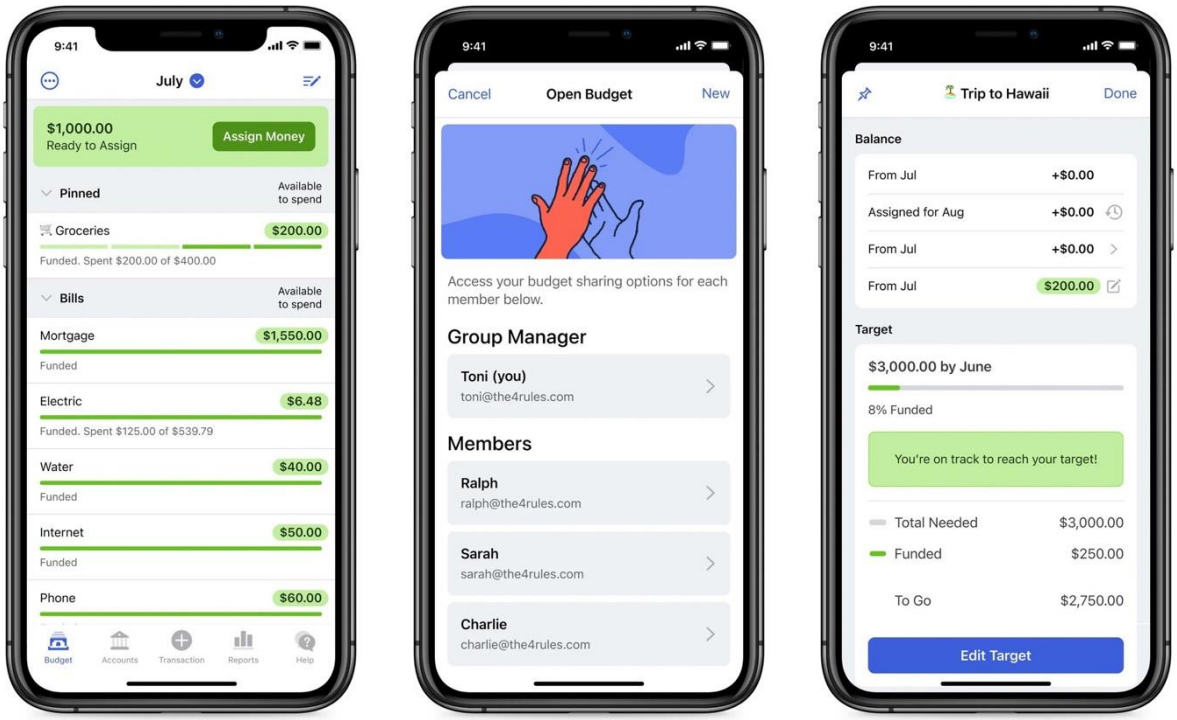
Сьогоднішні технології пропонують такі рішення, як онлайн-банкінг, фінансові трекери та інші [5], проте більшість з них не враховують споживачів, які використовують готівку. Саме тому створення застосунків для управління бюджетом та накопиченнями, які орієнтовано на таку аудиторію, є дуже актуальним у сучасну цифрову еру, що дозволить кожному легко та ефективно розподіляти доходи, уникати фінансових труднощів та досягати поставлених цілей.

1.2 Аналіз існуючих аналогів

На сьогоднішній день існує багато сервісів, для управління власними фінансами, які дозволяють слідкувати за доходами та витратами. Вони пропонують широкий спектр можливостей, які задовольняють потреби користувачів. Незважаючи на таке функціональне розмаїття, кожне з таких рішень має свої певні недоліки, для виявлення яких потрібно провести аналіз.

1.2.1 Мобільний застосунок "YNAB"

YNAB (You Need A Budget) (рисунок 1.1) – це популярний застосунок, доступний для всіх популярних платформ: IOS, Android, Windows та macOS. Він базується на системі "кожен долар має свою роботу", тобто кожен отриманий дохід розподіляється між категоріями витрат або заощаджень, що дозволяє уникнути необдуманого витрачання грошей.



а) б) в)

Рисунок 1.1 – Скріншоти застосунку: а) головний екран; б) синхронізація між користувачами та пристроями; в) інтерфейс застосунку

Застосунок має функцію синхронізації з банками, яка автоматично імпортує транзакції, що значно спрощує ведення бюджету. Це економить час користувачам та мінімізує ризик помилок, а синхронізація між пристроями робить його зручним для сімейних бюджетів (рисунок 1.1 б). Крім того, візуальна реалізація застосунку багатофункціональна та інформативна (рисунок 1.1 в), що дозволяє аналізувати фінансові звички користувача та вдосконалювати підхід до управління бюджетом.

YNAB дотримується високих стандартів безпеки, що гарантує захист фінансової інформації користувачів від кібератак. Проте, застосунок має і недоліки. По-перше, сервіс є платним, що робить його менш привабливим для тих, хто шукає економне рішення. По-друге, незважаючи на ефективність їхньої методології, нові користувачі можуть відчувати складнощі з тим, як правильно адаптуватися до такої специфічної дисципліни бюджетування. Також відсутність офлайн-режиму може бути проблемою, незважаючи на те, що зараз майже всюди є доступ до мережі.

1.2.2 Мобільний застосунок "Expensify"

Expensify – це застосунок для управління витратами, який автоматизує їх відстеження та створення звітів (рисунок 1.2). Така автоматизація звітності корисна, наприклад, при подачі податкових декларацій. Застосунок підтримує функцію синхронізації з банками, дозволяє створювати кастомні категорії витрат та правила для автоматичного сортування. До особливостей сервісу можна віднести:

- сканування чеків: технологія SmartScan (рисунок 1.2 а) дозволяє фотографувати чеки, а програма автоматично розпізнає інформацію (сума, дата, місце);

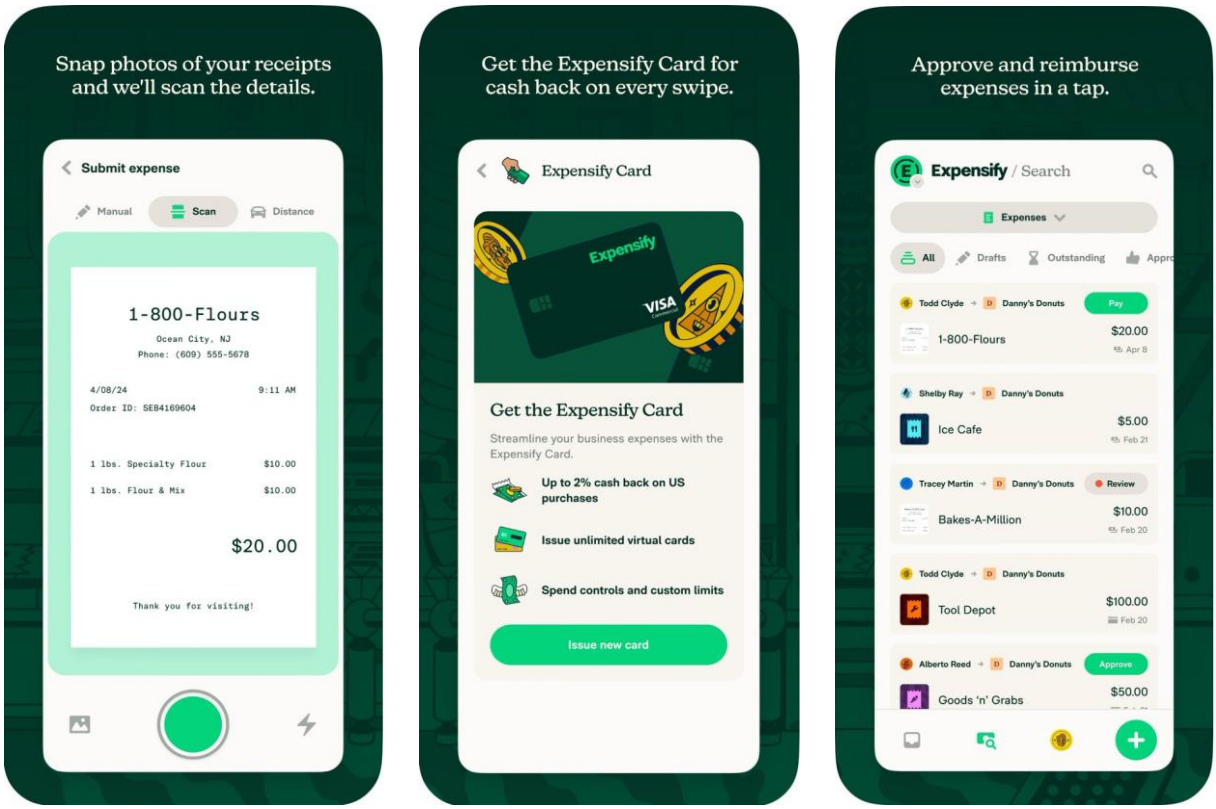
- можливість отримання власної кеш-бек картки (Expensify card) (рисунок 1.2 б);

- інтеграція з бухгалтерськими системами (QuickBooks, Xero) та

платформами для керування проєктами (Slack);

- підтримка командної роботи, що робить його ідеальним для компаній, які відстежують витрати співробітників;

- автоматичні нагадування: користувач отримуватиме повідомлення про необхідність оновлення фінансових даних.



а) б) в)

Рисунок 1.2 – Скріншоти застосунку: а) технологія SmartScan;
б) Expensify Card; в) інтерфейс застосунку

Попри все це, можна виділити такі недоліки:

- безкоштовна версія функціонально обмежена: в залежності від типу підписки, ціна становить від 5 до 10 доларів на місяць;

- інтерфейс може здатися заплутаним (рисунок 1.2 в) для тих, хто використовує застосунок вперше;

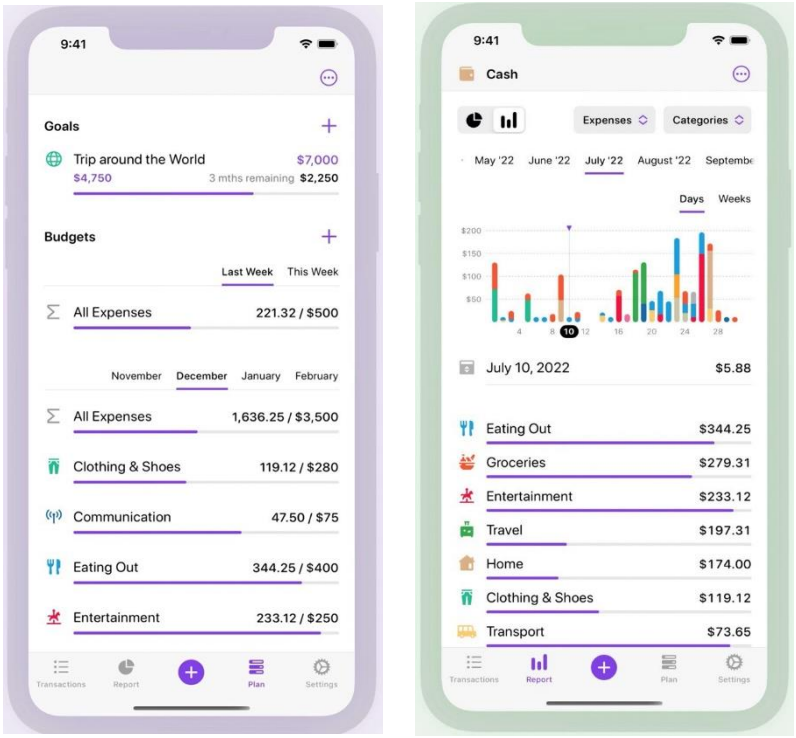
- функція SmartScan зручна, але іноді неправильно розпізнає дані з чеків, тому необхідно контролювати цей процес;

- без мережі обмежений доступ до функцій.

1.2.3 Мобільний застосунок "Spending Tracker – Money Flow"

Мобільний застосунок "Spending Tracker – Money Flow" підходить для тих, хто шукає простий та зручний варіант для відстеження особистих фінансів. Основними перевагами перед конкурентами є:

- зрозумілий, інтуїтивно організований та легкий у використанні навіть для новачків інтерфейс (рисунок 1.3 а);
- базовий функціонал доступний безкоштовно;
- присутня візуалізація (графіки, діаграми, звіти) (рисунок 1.3 б), яка допомагає легко та швидко аналізувати фінансовий стан;
- користувач має офлайн-доступ і застосунок працює без підключення до мережі Інтернет, що робить його доступним будь-де;
- застосунок не вимагає створення облікового запису, що є також важливим фактором для користувачів. Завдяки цьому підвищується конфіденційність та зручність його використання.



а) б)

Рисунок 1.3 – Скріншоти застосунку: а) інтерфейс застосунку; б) візуальні інструменти (діаграма)

Однак, як і попередні застосунки, цей також має певні недоліки:

- відсутність синхронізації з банками: усі транзакції потрібно вводити вручну;
- застосунок призначений виключно для особистого використання, тому він не підійде для спільного управління бюджетом, наприклад, сімейним;
- немає можливості автоматично додавати витрати чи доходи, що повторюються;
- при втраті пристрою, дані також можуть бути втрачені, але у платній версії є можливість резервного копіювання.

1.2.4 Мобільний застосунок "MyMoney Boost"

MyMoney Boost – мобільний застосунок, який є корисним інструментом для управління фінансами (рисунок 1.4).

а) б)

Рисунок 1.4 – Скріншоти застосунку: а) інтерфейс застосунку; б) звіти

Переваги застосунку:

- інтуїтивно зрозумілий інтерфейс (рисунок 1.4 а);
- застосунок надає детальні звіти, що допомагають аналізувати фінансові потоки (рисунок 1.4 б);
- користувачі можуть вести облік в різних валютах, що є корисним для мандрівників або тих, хто має доходи в іноземній валюті;
- застосунок надає можливість користувачам створювати власні категорії для більш детального відстеження витрат.

Недоліками застосунку є:

- відсутність автоматичної синхронізації з банками, що може бути незручним для користувачів;
- не підтримує інтеграцію з популярними фінансовими платформами.

1.3 Аналіз нефункціональних вимог користувача

Сучасні користувачі мають високі вимоги до продуктів IT-сфери, зокрема мобільних застосунків, які включають високу функціональність, естетичну привабливість та разом із цим простоту. Головною ціллю для гарантованого успіху проєкту є забезпечення комфортного досвіду користувача [6], який дозволить з легкістю слідкувати за фінансами та заощадженнями. Виходячи з цього, можна виділити ключові аспекти, які визначають нефункціональні вимоги користувачів.

1.3.1 Простий та привабливий дизайн

На даному етапі головну роль відіграють простота та лаконічність. Інтерфейс застосунку, що розробляється, має бути зрозумілим з першого погляду, без зайвих деталей, що можуть відволікати. Для реалізації такого дизайну найкращим рішенням буде використання сучасних підходів до візуального оформлення: мінімалізм, плавна анімація, гармонійна та ніжна кольорова палітра зроблять взаємодію із застосунком комфортною та

інтуїтивною. Також треба врахувати, щоб кнопки та текстові елементи були достатньо великими, що дозволить користуватися продуктом на пристроях із різними розмірами екранів та задовольнить вимоги різних категорій користувачів.

1.3.2 Зрозумілий та логічний інтерфейс

Користувачі завжди очікують, що основні функції (додавання доходів та витрат чи нових цілей накопичування, перегляд залишку бюджету та скарбнички) будуть доступні на головному екрані або у кілька натискань. Для цього треба вистроїти логічну послідовність сторінок, меню та кнопок, розробити зрозумілу навігацію, а назви розділів придумати чіткими та простими. Структура застосунку повинна бути інтуїтивною, що сприятиме швидкому освоєнню навіть новим користувачам.

Крім того, діаграми, які відображатимуть фінансову активність за певний період повинні бути зрозумілими, щоб користувач міг легко визначити, що вони означають і яку інформацію містять, також важливо забезпечити чітке позначення кольорів на них. Наприклад, зелений для позначення доходів, а червоний – витрат. Щомісячна звітність, сформована у вигляді кільцевої діаграми, має відображати в кожному сегменті відповідний відсоток, який він займає. Не зайвим буде сформувати легенду до цієї діаграми.

Кожну ціль накопичення також варто представити у вигляді діаграми, яка демонструє, скільки вже вдалося накопичити і скільки ще не вистачає до досягнення мети, що дозволить користувачам візуалізувати свій прогрес та підтримувати мотивацію.

1.3.3 Комфорт у використанні

При огляді з точки зору комфорту у використанні особливу увагу слід приділити наступним аспектам:

- швидкість роботи: застосунок має швидко реагувати на дії користувача без затримок;

- лаконічність та чистота інтерфейсу: відсутність зайвих елементів, що можуть відволікати;

- простота подачі інформації: відсутність перевантаження даними. Вони повинні подаватися структуровано, щоб користувач міг легко знайти потрібну йому інформацію.

Адаптивність інтерфейсу, стабільна робота та відповідність очікуванням користувачів забезпечить комфортне використання застосунку та покращить досвід користувача.

1.3.4 Головна сторінка та перший досвід із застосунком

Домашня сторінка відіграє роль "обличчя" продукту, тому вона повинна бути не тільки функціональною, а й естетично привабливою. Тут доцільно буде розмістити найважливішу інформацію, таку як поточний баланс, загальні витрати та дохід за певний період, наприклад, сьогодні та вчора, а також швидкий доступ до функцій, що використовуються найчастіше. Також розумним буде передбачити функцію додавання імені користувача, яке буде виводитись на головному екрані, а також його редагування та за потреби видалення. Людям зазвичай приємно бачити чи чути своє ім’я. Як зазначав Дейл Карнегі у своїй книзі "Як здобувати друзів і впливати на людей" [7], ім’я є найсолодшим та найважливішим звуком для кожної людини. Такий підхід сприятиме створенню більш "дружніх" відносин між користувачем та продуктом, що сформує позитивне враження та підвищить лояльність користувачів до застосунку.

Для формування приємнішого досвіду можна також додати "Splash screen" – екран, який користувач бачить при кожному новому запуску застосунку, своєрідна "візитна картка". Найчастіше на ньому розміщують логотип, іконку, мотивуючий слоган, а також якісні графічні елементи, що відповідають тематиці: гаманець, монети, картки, скарбнички. Таким чином

користувач матиме вражаючий перший досвід, що підкреслить високий рівень стилю застосунку.

Використання яскравих та гармонійно поєднаних кольорів, стилістично підібраних графічних елементів на головній сторінці утримає увагу користувача та створить позитивне враження.

Отже, мобільний застосунок для слідкування за бюджетом повинен бути не лише функціональним, а ще й комфортним та привабливим у використанні. Цього можна досягти завдяки зрозумілому дизайну, чіткій структурі та адаптивності, що задовольнить потреби широкої аудиторії користувачів та допоможе їм ефективно управляти власними фінансами та досягати своїх цілей.

1.3.5 Принцип кросплатформності

Окремо варто наголосити на важливості кросплатформності. Сучасний користувач мобільними пристроями очікує мати доступ до застосунку незалежно від операційної системи свого смартфону. Реалізація кросплатформності дозволить забезпечити однаковий рівень зручності й функціональності як на ОС Android, так і на IOS, а також розширить аудиторію продукту, його доступність та конкурентоспроможність.

1.4 Побажання користувача до функціональних вимог застосунку

Користувачам важливо мати змогу зручно відстежувати свої транзакції, аналізувати доходи та витрати за певні періоди, а також керувати цілями накопичення. Тож, можна виділити основні функції, яких очікують користувачі.

1.4.1 Облік доходів та витрат

Додавання та зберігання транзакцій є найголовнішою функцією застосунку, тому вона повинна бути інтуїтивно зрозумілою та швидкою для

використання, що дозволить користувачам швидко вносити інформацію. Для кожної транзакції необхідно врахувати:

- суму транзакції: текстове поле, яке дозволяє вводити цифри та знаки: наприклад, знаки "," або ".";
- тип транзакції: випадаюче меню, де можливо обрати категорію із запропонованих варіантів;
- назву транзакції: текстове поле, де користувач вводить назву за власним бажанням;
- іконку та колір її фону: малюнок, який обирається за бажанням користувача, щоб зробити процес більш інтерактивним та цікавим;
- дату транзакції: дата-пікер, у якому автоматично встановлена сьогоднішня дата, яку можна змінити за допомогою інтерактивного вибору.

Після додавання, бажано, щоб транзакція виводилась одразу на головний екран, щоб користувач одразу бачив результат функції додавання. Також варто передбачити перегляд всіх транзакцій за певною категорією, що забезпечить користувачу зручний перегляд транзакцій певного типу. Крім цього, корисною буде можливість видаляти окрему транзакцію на випадок, якщо вона була помилково створена чи виникла помилка в даних.

1.4.2 Формування та побудова діаграм

Для аналізу доходів та витрат розумно використовувати візуальні інструменти, такі як діаграми, що охоплюють різні часові інтервали (наприклад, щоденний або щотижневий аналіз), кожна з яких виконуватиме певну мету для інформування користувача.

На діаграмах щоденної активності вертикальна вісь може показувати суму витрат або накопичень, а горизонтальна – поділяти інформацію за днями тижня. Для швидкої оцінки співвідношення цих показників, такі діаграми представляють у формі стовпчиків різних кольорів для доходів та витрат. Як правило, вони оновлюються щотижня, щоб користувач міг аналізувати свої фінансові потоки в короткостроковій перспективі.

Тижневі діаграми зазвичай охоплюють кілька тижнів, наприклад чотири, для тенденції фінансової активності користувача за місяць. Горизонтальна вісь таких діаграм може показувати часові межі кожного тижня, в рамках яких будуть групуватися дані для відображення. Після закінчення періоду діаграма автоматично оновлюється для подальшого використання.

1.4.3 Формування звітності

Звіт у більшості випадків формується щомісяця та показує загальну суму доходів та частку кожної категорії витрат від цієї суми. Для того щоб користувач візуально оцінив на що саме він частіше витрачає на місяць, у звіт також додають кільцеву діаграму. Шляхом зафарбовування певного сегменту на ній зображується, яку частку з усього доходу за місяць займає певна категорія витрат та частка що залишилась. Крім цього, зазвичай, на кожному сегменті слід зазначати, який відсоток він займає. Таким чином, аналіз фінансової активності за місяць буде простим, зрозумілим та інформативним для кожного користувача.

1.4.4 Функціональність скарбнички

Для цілей накопичення необхідно враховувати такі дані:

- сума накопичення;
- назва цілі;
- іконка та колір її фону;
- дата створення.

Процес створення цілі накопичення майже однаковий із процесом додавання транзакції. Різниця полягає лише в тому, що не треба вказувати категорію. Після створення вона одразу додається до окремого екрану "Скарбничка", доступ до якого, зазвичай, є прямо з головного екрану, що дозволяє користувачеві швидко переключатись між частиною транзакцій та

накопичень.

При натисканні на ціль відкривається діалогове вікно, яке містить кільцеву діаграму та поле для введення суми з кнопкою для підтвердження "депозиту". Діаграма служить для візуального відображення прогресу користувача. Як і діаграма для звіту, вона відображає відсоткове значення сегментів: накопичено та скільки ще залишилось. Нижче варто розташувати легенду (для розуміння значення сегментів).

У випадку, якщо користувач спробує ввести суму більшу за те, що планується накопичити, система має попередити користувача відповідним повідомленням про неможливість додавання. При досягненні цілі накопичення в діалоговому вікні має відобразитись повідомлення про успішне завершення цілі разом з поточною дату. Також слід надати користувачу можливість видаляти ціль, якщо вона більше не актуальна або була досягнута.

1.4.5 Очищення даних

Для більшої продуктивності застосунку та розумного використання пам’яті пристрою користувача, застосунок має лише дані за певний період (наприклад, три місяці), дані старіші за цей період мають видалятися автоматично. Крім того, є сенс реалізувати функцію повного очищення даних, яка передбачає видалення всієї інформації про транзакції та цілі накопичення, якщо користувач вирішить почати облік з нуля.

Функціональні вимоги до застосунку для управління бюджетом визначають його здатність забезпечувати користувачів інструментами для ефективного обліку фінансів. Ведення транзакцій, побудова наочних діаграм, формування звітності та управління скарбничкою створюють зручний та потужний інструмент для досягнення фінансових цілей. Крім того, функції очищення даних та гнучке управління транзакціями забезпечують комфортність роботи із застосунком.

1.5 Постановка задачі кваліфікаційної роботи

В ході проведення попереднього аналізу існуючих застосунків виникла необхідність у розробці кросплатформного мобільного застосунку, який матиме наступний набір можливостей:

- додавання імені користувача;
- додавання транзакцій;
- додавання цілей накопичення;
- формування діаграм щоденної та щотижневої активності;
- формування щомісячного звіту;
- перегляд всіх транзакцій за певними категоріями;
- досягання мети накопичення;
- видалення певної транзакції чи цілі;
- повне очищення даних.

Для невеликих проєктів з використанням Flutter, які не потребують масштабних функцій, типовою є Flat Architecture, перевагами якої є:

- простота реалізації: кожен екран та функціональність виділені в окремі файли, що робить код базово організованим;

- легка налагоджуваність: виправляти й відстежувати помилки простіше, оскільки кожен екран ізольований;

- чітка відповідальність модулів: завдяки принципу Single Responsibility Principle (SPR) кожен файл відповідає за певну частину застосунку, що робить код модульним та більш логічним;

- швидкість розробки: займає менше часу, оскільки не потрібно впроваджувати складніші архітектури;

- зручність для невеликих проєктів.

На даному етапі Flat Architecture забезпечує оптимальний баланс між простотою та швидкістю розробки, що дозволяє сфокусуватися на функціональності та досвіді користувача, а не на складних архітектурних патернах та робить її ідеальним інструментом для реалізації даного проєкту.

2 АНАЛІЗ ТЕХНОЛОГІЙ, ЩО ВИКОРИСТОВУЮТЬСЯ ПРИ РОЗРОБЦІ МОБІЛЬНОГО ЗАСТОСУНКУ

2.1 Мова програмування Dart

Dart – сучасна мова програмування, яка була створена компанією Google у 2011 році, орієнтована на розробку високопродуктивних та кросплатформних застосунків. Мова в основному використовується з фреймворком Flutter, який став відомим завдяки зручності розробки візуально привабливих інтерфейсів.

Для того, щоб привернути увагу розробників, Dart має ряд особливостей, таких як:

- зрозумілий синтаксис, схожий на Java, JavaScript, що робить його легким для вивчення тим, хто вже знайомий з цими мовами [8];

- підтримка статичної типізації, що допомагає уникати помилок на етапі компіляції, а також там, де зручно, дозволяє використовувати динамічну;

- компіляція Just-In-Time та Ahead-Of-Time: перша використовується для швидкого перезавантаження коду під час розробки, а друга дозволяє компілювати застосунок у високорівневий рідний код для максимальної швидкості виконання;

- підтримка асинхронного програмування для ефективної роботи із запитами до серверів або обробкою великих обсягів даних;

- система пакетів, що дозволяє швидко та легко підключати зовнішні бібліотеки та модулі через платформу pub.dev;

- підтримка кросплатформності, що дозволяє створювати застосунки з єдиним кодом для IOS, Android, десктопів та вебплатформ.

Перелічені особливості роблять мову Dart ідеальним вибором для розробки сучасного застосунку, який матиме привабливий, інтуїтивний дизайн разом з високою функціональністю та підтримкою роботи на багатьох

сучасних та популярних платформах. Крім того, Dart відзначається високою швидкістю розробки, що також є вагомим фактором при виборі мови програмування.

2.2 Фреймворк Flutter

Flutter – це популярний відкритий фреймворк, розроблений компанією Google для створення кросплатформних застосунків. Він використовує мову програмування Dart і дозволяє створювати сучасні, візуально привабливі та високопродуктивні інтерфейси.

Основною перевагою фреймворку є можливість створювати програми з однією кодовою базою, які виглядають та працюють однаково на всіх платформах, а саме: IOS, Android, вебплатформи, Windows, macOS, Linux та навіть вбудовані системи. Також Flutter підтримує створення анімаційних та кастомних інтерфейсів завдяки компонентам:

- Widgets, із яких складаються всі елементи інтерфейсу. Є також можливість створення повністю власних віджетів для більшого контролю над зовнішнім виглядом;
- Material Design – набір стилів та компонентів, які забезпечують сучасний і зручний інтерфейс;
- Cupertino – компоненти, що відповідають стилю IOS.

Крім того фреймворк використовує власний графічний механізм для рендерингу інтерфейсів, що надає повний контроль над зовнішнім виглядом та поведінкою елементів.

До переваг Flutter також можна віднести підтримку функції Hot reload (гаряче перезавантаження), яка дозволяє миттєво бачити результати зміни в коді без необхідності перезапускати програму, що значно прискорює процес розробки та налагодження [9]. Окрім технічної досконалості, Flutter отримує активну підтримку від Google, що забезпечує його постійний розвиток та оновлення. Велика спільнота розробників, багата документація та доступ до

численних бібліотек та навчальних ресурсів роблять цей фреймворк практичним як для початківців, так і для досвідчених програмістів. Універсальність, простота використання та підтримка Google роблять Flutter також ідеальним вибором для тих, хто хоче створювати високоякісні міжплатформні програми з мінімальними витратами часу та зусиль.

2.3 Середовище розробки Visual Studio Code

Visual Studio Code (VS Code) – це ефективний та простий у використанні редактор коду, що підтримує велику кількість розширень, що робить його надзвичайно гнучким для різних типів проєктів [10]. Основні переваги VS Code включають:

- простий інтерфейс: легко налаштовується під потреби розробника, крім того містить вбудовану консоль, яка дозволяє виконувати команди безпосередньо в редакторі;
- гнучкість та розширюваність: широкий вибір розширень для підтримки різних мов програмування та фреймворків, можливість налаштування сценаріїв запуску та налагодження через файл launch.json;
- продуктивність: інструменти для рефакторингу та автозаповнення коду, швидке завантаження та стабільна робота навіть з великими проєктами;
- інтеграція з іншими інструментами: легка інтеграція з Azure, GitHub та іншими сервісами;
- підтримка спільної розробки: інструменти для командної роботи, включаючи Live Share у VS Code.

Для підтримки мови програмування Dart у VS Code встановлено відповідне розширення Dart, яке забезпечує ефективне використання коду, що значно підвищує продуктивність роботи. Це розширення дозволяє розробникам повною мірою використовувати можливості Dart прямо в VS Code, одночасно отримуючи доступ до всіх інструментів та функцій, доступних у середовищі редактора.

2.4 Інструменти тестування мобільних застосунків

Android Emulator та Xcode Simulator є ключовими інструментами для тестування мобільних застосунків, які дають розробникам можливість імітувати їх роботу без використання реальних пристроїв.

Android Emulator, розроблений Google, дозволяє відтворювати роботу операційної системи Android на комп'ютері [11] та пропонує широкі можливості тестування, такі як налаштування різних версій операційних систем, екрани з різною роздільною здатністю та моделювання роботи з датчиками, такими як GPS, акселерометр або камера. Емулятор інтегрується з Android Studio та забезпечує зручний інтерфейс для налагодження та аналізу застосунків, що робить його незамінним у процесі розробки.

Симулятор Xcode призначений для тестування програм на пристроях Apple є одним із інструментів, який доступний через середовище розробки Xcode, що дозволяє тестувати програми для таких операційних систем як iOS, watchOS, tvOS та iPadOS, надаючи розробникам доступ до віртуальних пристроїв, які точно імітують реальні пристрої [12]. Незважаючи на те, що симулятор не підтримує повний набір функцій, доступних на реальному пристрої, наприклад Bluetooth або тестування продуктивності апаратного забезпечення, він ідеально підходить для тестування інтерфейсу, поведінки програми та основних функцій.

Емулятор Android та симулятор Xcode – це ефективні рішення, які допомагають розробникам зберігати час та ресурси, шляхом швидкого знаходження помилки та оптимізування продуктивності програми на етапі тестування.

2.5 Система контролю версій Git

Система контролю версій Git використовується для відстеження змін у коді та спільної роботи над проєктами [13]. Вона зберігає історію всіх змін разом з описом, автором та датою, дозволяє працювати над різними версіями

файлів паралельно та легко об’єднувати ці зміни за допомогою гілкування.

В даному проєкті було використано Git та хмарний сервіс для зберігання Git-репозиторіїв Github, який також надає функції для перегляду змін та управління проєктами. Репозиторій проєкту знаходився на Github, що забезпечувало зручний доступ до коду.

При додаванні змін (нові функції чи виправлення) створювалися окремі гілки, які після перевірки об’єднувалися (merge) у спільну гілку develop. Зазвичай, таку схему використовують, щоб інші розробники могли перевірити запропоновані зміни та в разі знаходження помилок надати коментарі. Проте, у випадку, коли над проєктом працює одна людина, слід скористатися допомогою ШІ-асистента, наприклад, OpenAI ChatGPT, який здатний аналізувати зміни в коді, надавати рекомендації для покращення коду, а також виявляти потенційні проблеми. Використання такого асистенту забезпечить високу якість коду та спростить процес розробки незважаючи на те, яка кількість розробників працює над проєктом.

Остаточна версія проєкту після завершення розробки та тестування була перенесена до головної гілки main. Зазвичай, ця гілка містить стабільну версію продукту, готову до релізу.

2.6 Інструмент управління базами даних SQLite

DB Browser for SQLite – це відкрита та зручна утиліта для роботи з базами даних SQLite, розроблена для спрощення створення, перегляду, редагування та управління SQLite-базами даних.

Особливостями даної утиліти є:

- простий та зрозумілий інтерфейс, що дозволяє працювати з базою даних, не використовуючи складні термінальні команди;

- можливість створення таблиць, редагування існуючих таблиць та їх полів та налаштування первинних ключів, унікальних значень та індексів;

- зручний перегляд даних у табличній формі;

- підтримка виконання SQL-запитів та їх збереження до історії

виконаних запитів;

- можливість перегляду журналу змін;

- імпорт даних та експорт таблиць чи бази даних у різних форматах;

- можливість створення резервних копій для безпечного використання;

- велика кількість інструментів для аналізу структури бази даних;

- доступна для Windows, macOS, Linux, а SQLite взагалі не вимагає серверної інфраструктури.

Ці фактори роблять дану утиліту чудовим інструментом для швидкого та простого управління базами даних SQLite.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ЗАСТОСУНКУ

3.1 Огляд структури проєкту

У проєкті застосовано підхід Flat Architecture, який полягає у спрощеному розподілі програмного коду на окремі, але взаємопов'язані компоненти з чітким визначенням відповідальностей. На рисунку 3.1 наведена структура проєкту.

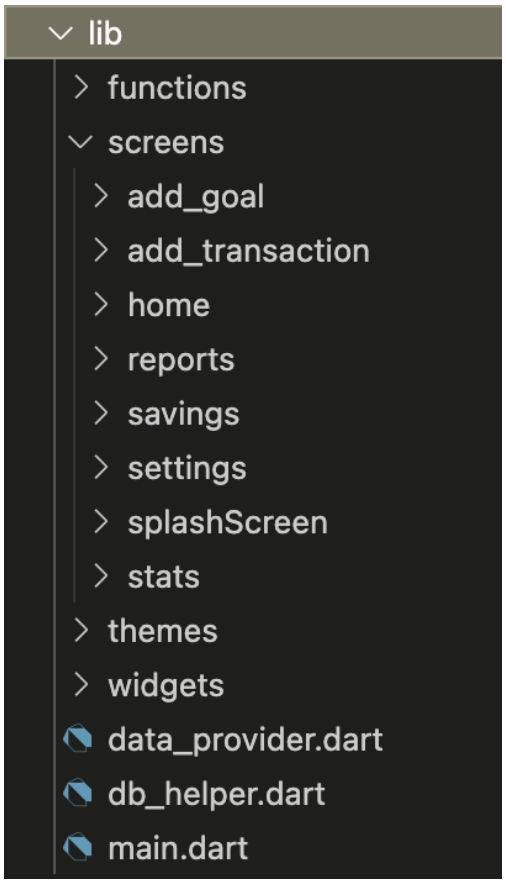


Рисунок 3.1 – Структура проєкту

3.2 Каталог functions

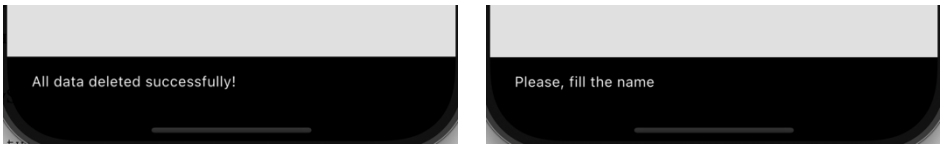
Каталог functions містить файли з функціями, які використовуються повторно в різних частинах проєкту, що дозволяє уникнути дублювання коду та зробити його більш організованим, а також зробити код файлів, де вони

використовуються, чистішим та легким для підтримки. Наприклад, файл show_message.dart містить функцію showMessage (лістинг 3.1), яка використовується для виведення повідомлення у вигляді SnackBar, компонент який з'являється внизу екрана, щоб привернути увагу користувача без необхідності переривати його взаємодію з інтерфейсом. Завдяки цій функції відображаються короткі повідомлення: інформування користувача про успішне виконання якоїсь дії (рисунок 3.2 а) або виникнення помилки (рисунок 3.2 б).

Для виклику цієї функції потрібно передати два параметри: context – контекст, який необхідний для отримання доступу до теми і налаштувань інтерфейсу та message – текст повідомлення, яке буде відображено.

Лістинг 3.1 – Функція showMessage

```
void showMessage(BuildContext context, String message) {
  ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(
        content: CustomTextWidget(
      text: message,
      styleType: TextStyleType.defaultStyle,
      color: Theme.of(context).colorScheme.onPrimary,
    )),  );  }
```



а) б)

Рисунок 3.2 – Приклад виклику функції showMessage:
а) успішна операція; б) помилка

3.2 Каталог screens

Каталог screens містить підкаталоги, в яких розташовані файли для кожного екрану застосунку. Їх всього 5, а саме: головний екран, екрани накопичень, налаштувань, звітів та статистики.

3.2.1 Підкаталоги add_goal та add_transaction

Підкаталоги add_goal та add_transaction відповідають за відображення екранів для створення та подальшого збереження нової цілі та транзакції відповідно.

Для відображення полів на екрані використовується віджет TextFormField, який надає багато можливостей для кастомізації текстових полів. Наприклад, для відображення поля суми встановлені обмеження, які дозволяють вводити лише цифри та символи для розділення десяткових частин (крапка або кома). Віджет налаштований з використанням TextInputFormatter для фільтрації введення, що дозволяє лише допустимі символи (лістинг 3.2).

Лістинг 3.2 – Реалізація поля для введення суми

```
TextFormField(
        maxLength: 9,
        controller: amountController,
        decoration: InputDecoration(
          filled: true,
          fillColor: Theme.of(context).colorScheme.onPrimary,
          prefixIcon: Icon(CupertinoIcons.money_euro,
          color: Theme.of(context).colorScheme.outlineVariant,),
          border: OutlineInputBorder(
          borderRadius: BorderRadius.circular(30),
          borderSide: BorderSide.none),  ),
         keyboardType:
         const TextInputType.numberWithOptions(decimal: true),
         inputFormatters: [
         FilteringTextInputFormatter.allow(
         RegExp(r'^\d+([.,]\d{0,2})?')),],  ),
```

А ось поле для відображення дати автоматично заповнюється поточною датою, і це значення не можна змінити завдяки параметру readOnly, якому присвоєно значення true. Ініціалізація значення дати здійснюється в методі initState (лістинг 3.3). Для форматування дати в потрібному вигляді (наприклад, в проєкті це dd/MM/yyyy) використовується плагін intl, який відповідає за форматування дати та часу в застосунку.

Лістинг 3.3 – Метод initState

```
@override
  void initState() {
    dateController.text =
DateFormat('dd/MM/yyyy').format(DateTime.now());
    super.initState();
  }
```

Також цікавим прикладом використання даного віджету є поле для завдвння кольору фону іконки за допомогою плагіна flutter_colorpicker (лістинг 3.4), який дозволяє користувачам обирати кольори з різних палітр.

Лістинг 3.4 – Використання плагіну flutter_colorpicker

```
ColorPicker(
  pickerColor:
      designColorDB,
  onColorChanged:
      (value) {
    setState(() {
      designColorDB =
          value;
    });
  },),
```

Для того щоб зберегти та додати ціль накопичення або транзакцію треба натиснути на кнопку «Save», яка є віджетом TextButton в параметрі onPressed. В цьому параметрі вказано метод, що викликає функцію збереження в базу даних (лістинг 3.5).

Лістинг 3.5 – Реалізація кнопки «Save»

```
TextButton(
  onPressed: saveGoal,
  style: TextButton.styleFrom(
      backgroundColor: Theme.of(context).colorScheme.onSurface,
      shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(12))),
  child: CustomTextWidget(
    text: 'Save',
    styleType: TextStyleType.defaultStyle,
    color: Theme.of(context).colorScheme.onPrimary,
    size: 22,
  ),),
```

3.2.2 Підкаталог home

Підкаталог home містить два файли: home_screen.dart та main_screen.dart, які реалізують відображення головного екрану.

Файл home_screen.dart відповідає за реалізацію елементів інтерфейсу, таких як AppBar (рисунок 3.3 а) та BottomNavigationBar (рисунок 3.3 б), які є фіксованими. У AppBar відображається напис «Welcome!» та ім’я користувача (якщо він додав його) (лістинг 3.6) та іконка, що дозволяє перейти на екран налаштувань (лістинг 3.7). BottomNavigationBar слугує для реалізації навігації (лістинг 3.8) між головним екраном, на якому відображаються останні транзакції та баланс користувача та екраном скарбнички.

Лістинг 3.6 – Привітання користувача в AppBar

```
Row(
  children: [
    Stack(),
    const SizedBox(
      width: 8,
    ),
    Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        const CustomTextWidget(
            text: 'Welcome!', styleType: TextStyleType.h2),
        CustomTextWidget(
            text: dataProvider.profileName,
            styleType: TextStyleType.h1),
    ],    ),  ],  ),
```

Лістинг 3.7 – Перехід до екрану налаштувань

```
IconButton(
  onPressed: () {
    Navigator.push(
      context,
      MaterialPageRoute(
          builder: (context) => const SettingsScreen()),
    );   },
  icon: const Icon(CupertinoIcons.settings),
)
```

Лістинг 3.8 – Елемент інтерфейсу BottomNavigationBar

```
BottomNavigationBar(
  onTap: (value) {
    setState(() {
      index = value;
    });  },
  backgroundColor: Theme.of(context).colorScheme.onPrimary,
  showSelectedLabels: false,
  showUnselectedLabels: false,
  items: [
    BottomNavigationBarItem(
        icon: Icon(
          CupertinoIcons.graph_square_fill,
          color: index == 0 ? selectedItem : unselectedItem,
        ),
        label: 'Main'),
    BottomNavigationBarItem(
        icon: FaIcon(
          FontAwesomeIcons.vault,
          size: 20,
          color: index == 1 ? selectedItem : unselectedItem,
        ),
        label: 'Savings')
  ]),
```

У параметрі body визначається, який екран відображати залежно від обраного індексу (лістинг 3.9).

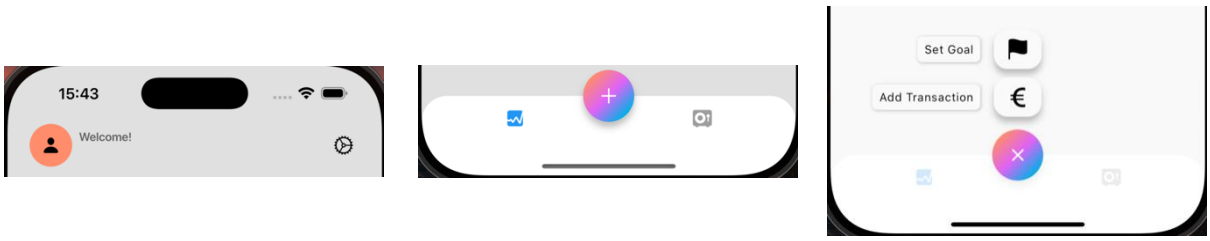
Лістинг 3.9 – Параметр body

```
body: () {
  switch (index) {
    case 0:
      return const MainScreen();
    case 1:
      return const SavingsScreen();
  }
}(),
```

Крім того у BottomNavigationBar реалізовано кнопку для додавання транзакцій та цілей накопичення (лістинг 3.10). При натисканні на неї відкривається меню, яке дозволяє обрати одну з опцій: додати транзакцію або ціль накопичення (рисунок 3.3 в), після чого відкривається відповідний екран. Це реалізовано за допомогою плагіна flutter_speed_dial, який надає зручні можливості для створення таких випадаючих меню з кількома діями.

Лістинг 3.10 – Кнопка для додавання транзакцій або цілей

```
SpeedDial(
  icon: CupertinoIcons.add,
  activeIcon: Icons.close_rounded,
  gradient: LinearGradient(colors: [
    Theme.of(context).colorScheme.tertiary,
    Theme.of(context).colorScheme.secondary,
    Theme.of(context).colorScheme.primary
  ], transform: const GradientRotation(pi / 4)),
  gradientBoxShape: BoxShape.circle,
  buttonSize: const Size(60, 60),
  spacing: 10,
  children: [
    SpeedDialChild(
      child: const Icon(CupertinoIcons.money_euro,
        size: 40,      ),
      //backgroundColor: ,
      label: 'Add Transaction',
      onTap: () {
        Navigator.push(
          context,
          MaterialPageRoute(builder: (context) => const
AddTransaction()),        );      },     ),
    SpeedDialChild(
      child: const Icon(
        CupertinoIcons.flag_fill,
        size: 30,      ),
      //backgroundColor: ,
      label: 'Set Goal',
      onTap: () {
        Navigator.push(context,
          MaterialPageRoute(builder: (context) => const
AddGoal()),        );       },    ),  ],   ),
```



а) б) в)

Рисунок 3.3 – Інтерфейс головного екрану: а) AppBar;
б) BottomNavigationBar; в) кнопка додавання транзакцій та цілей

Файл main_screen.dart відповідає за відображення головного екрану. У верхній частині екрану відображається баланс у вигляді елемента, який

оформлено у вигляді кредитної картки. Його створено за допомогою класичних віджетів для побудови інтерфейсу (лістинг 3.11): Container (контейнер), Row (рядок), Column (стовпець), а також він має градієнтне забарвлення (рисунок 3.4).

Лістинг 3.11 – Елемент "картка" на головному екрані

```
Container(
  width: MediaQuery.of(context).size.width,
  height: MediaQuery.of(context).size.width / 2,
  decoration: BoxDecoration(
      gradient: LinearGradient(colors: [
        Theme.of(context).colorScheme.primary,
        Theme.of(context).colorScheme.secondary,
        Theme.of(context).colorScheme.tertiary
      ], transform: const GradientRotation(pi / 4)),
      borderRadius: BorderRadius.circular(25),
      boxShadow: [
        BoxShadow(
            blurRadius: 5, color: Theme.of(context)
                .colorScheme
                .outline, //Colors.grey.shade500,
            offset: const Offset(0, 5)),      ]),
  child: Column()
```

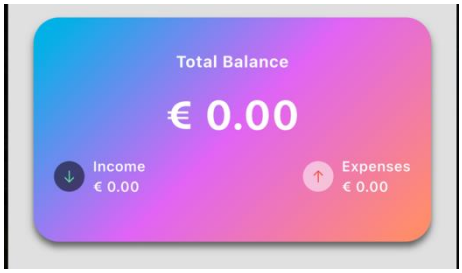


Рисунок 3.4 – Вигляд елементу "картка" на головному екрані

Нижче розташований список транзакцій, який було створено "учора" та "сьогодні". Для його побудови використовується ListView.builder, що забезпечує динамічне завантаження списку (лістинг 3.12).

Лістинг 3.12 – Реалізація списку транзакцій за допомогою ListView.builder

```
ListView.builder(
  itemCount: filteredTransactions.length,
  itemBuilder: (context, int i) {
```

```
    final transaction =
        parseTransaction(filteredTransactions[i]);
    return CustomTAshowWidget(transaction: transaction, i: i);
  },
```

Також над списком реалізовано кнопку для переходу на екран зі статистикою та звітами. Для її створення використано віджет GestureDetector, який дозволяє обробляти натискання користувача (лістинг 3.13).

Лістинг 3.13 – Кнопка "View All"

```
GestureDetector(
  onTap: () {
    Navigator.push(
      context,
      MaterialPageRoute(
          builder: (context) => const StatsScreen()),
    );
  },
  child: const CustomTextWidget(
      text: 'View All', styleType: TextStyleType.h2),)
```

3.2.3 Підкаталог reports

Підкаталог reports містить два файли: reports.dart, який відповідає за реалізацію екрана звітності та report_chart.dart, який містить саму діаграму звітності. Файл report_chart.dart містить функції для: ініціалізації місяця звіту, розрахунку загальної суми витрат за місяць у кожній категорії, формування сегментів діаграми та присвоєння їм кольорів, побудови легенди (де кожен сегмент позначає певний тип витрат із відповідною сумою), створення та відображення самої діаграми (лістинг 3.14).

Лістинг 3.14 – Функції файлу report_chart.dart

```
List<DateTime> _initializeReportMonths() {
  final dataProvider = context.read<DataProvider>();
  final uniqueMonths = dataProvider.transactions
      .map((transaction) {
        final date = _dateFormat.parse(transaction['date']);
        return DateTime(date.year, date.month);      })
      .toSet()
```

```
      .toList();
  uniqueMonths.sort((a, b) => a.compareTo(b));
  return uniqueMonths;}
Map<String, num> _calculateCategorySums(DateTime month) {
  final dataProvider = context.read<DataProvider>();
  final monthlyTransactions =
dataProvider.transactions.where((transaction) {
    final date = _dateFormat.parse(transaction['date']);
    return date.year == month.year && date.month == month.month;
});
  final Map<String, num> sums = {
    'Income': 0,
    'Living': 0,
    'Entertainment': 0,
    'Others': 0,  };
  for (var transaction in monthlyTransactions) {
    final category = transaction['category'] ?? 'Others';
    sums[category] = sums[category]! + transaction['amount'];  }
  return sums;}
List<PieChartSectionData> _calculateSections(Map<String, num>
sums) { Color _getCategoryColor(String category) {
Widget _buildLegend(Map<String, num> sums) {
Widget _buildPieChartWithLegend(BuildContext context, DateTime
month) {
void _updateReportMonths() {
```

Для створення діаграми використовується плагін fl_chart, який пропонує широкий вибір графіків та діаграм. Для проєкту було обрано кільцеву діаграму, оскільки вона забезпечує наочне відображення витрат за категоріями. Крім того, для перегляду звітів за попередні місяці використовується PageView.builder, який дозволяє перегортати сторінки з минулими звітами (лістинг 3.15).

Лістинг 3.15 – PageView.builder

```
PageView.builder(
  controller: _pageController,
  itemCount: reportMonths.length,
  itemBuilder: (context, index) {
    final month = reportMonths[index];
    return _buildPieChartWithLegend(context, month);  },),
```

При початку нового місяця поточний звіт зміщується назад, а найстаріший із трьох збережених автоматично видаляється за допомогою функції _updateReportMonths (лістинг 3.16).

Лістинг 3.16 – Функція оновлення місяців звіту

```
void _updateReportMonths() {
  final now = DateTime.now();
  final currentMonth = DateTime(now.year, now.month);
  setState(() {
    // Додаємо поточний місяць, якщо його ще нема
    if (!reportMonths.contains(currentMonth)) {
      reportMonths.add(currentMonth);
    }
    // Видаляємо найстаріший місяць, якщо їх більше 3
    if (reportMonths.length > 3) {
      reportMonths.removeAt(0);
    }  }); }
```

3.2.4 Підкаталог savings

Підкаталог savings містить два файли: savings.dart та savings_chart.dart. Перший файл відповідає за відображення списку цілей накопичення. Він реалізований аналогічно списку транзакцій за допомогою ListView.builder (лістинг 3.17), але має відмінне оформлення (рисунок 3.5). Кожен елемент списку містить кнопку у вигляді іконки "стрілка вправо", натискання на яку відкриває діалогове вікно з детальною інформацією про ціль.

Лістинг 3.17 – ListView.builder для цілей накопичення

```
ListView.builder(
  itemCount: dataProvider.goals.length,
  itemBuilder: (context, int i) {
    final filteredTransactions =
        dataProvider.goals.toList();
    final goal = filteredTransactions[i];
    final amount = "${goal['amount'].toStringAsFixed(2)}";
    String iconName = goal['icon'];
    String iconPath = 'assetsg/$iconName.png';
    final color = Color(int.parse(goal['color']));
    ...
),
```

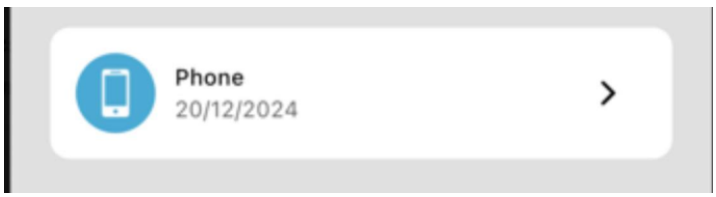


Рисунок 3.5 – Дизайн цілі накопичення

Другий файл підкаталогу, який відповідає за вміст діалогового вікна, містить реалізацію діаграми прогресу досягнення цілі (за дизайном вона схожа на діаграму звітів).

У нижній частині діалогового вікна розташоване поле TextFormField для введення суми, яку користувач хоче "покласти" на накопичення. Поруч знаходиться кнопка "Put", яка реалізована через TextButton. Перед виконанням дії анонімна функція в параметрі onPressed перевіряє, чи введена сума не перевищує необхідну для досягнення цілі. Якщо перевищує – виводиться відповідне повідомлення, якщо ні – сума додається до вже накопиченої, після чого оновлюються дані (лістинг 3.18).

Лістинг 3.18 – Кнопка "Put"

```
TextButton(
  onPressed: () {
    double enteredAmount = double.tryParse(amountController.text
            .trim()
            .replaceAll(',', '.')) ??
        0.0;
    if (enteredAmount + collected >
        double.parse(widget.amount.replaceAll(',', '.'))) {
      showMessage(context, 'The amount is too big');
      return;
    }
    setState(() {
      collected += enteredAmount;
      _saveCollected();
      amountController.clear();
    });
  },
  style: TextButton.styleFrom(
      backgroundColor: Theme.of(context).colorScheme.onSurface,
      shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(12))),
  child: const CustomTextWidget(
    text: 'Put',
    styleType: TextStyleType.defaultStyle,
    size: 18,
  ),),
```

Для збереження поточного прогресу використовується SharedPreferences (лістинг 3.19) – механізм локального сховища для

збереження простих даних у вигляді пар "ключ-значення". Дані зберігаються безпосередньо в пам'яті пристрою, що дозволяє застосунку працювати без підключення до інтернету.

В даному випадку метод записує зібрану суму до відповідного ключа в локальному сховищі та при завершенні цілі – зберігається ще й поточна дата.

Лістинг 3.19 – Функція збереження даних через SharedPreferences

```
Future<void> _saveCollected() async {
final prefs = await SharedPreferences.getInstance();
final collectedKey = 'collected_${widget.id}';
final dateKey = 'currentDate_${widget.id}';
await prefs.setDouble(collectedKey, collected);
if (collected == double.tryParse(widget.amount.replaceAll(',',
'.'))) {
  String currentDate =
DateFormat('dd/MM/yyyy').format(DateTime.now());
  await prefs.setString(dateKey, currentDate);
  _loadCollected();
}}
```

Після досягнення цілі накопичення на місці діаграми з'являється відповідне повідомлення, а саму ціль можна видалити змахом вліво (рисунок 3.6). Це викликає функцію видалення даних як із списку, так і з SharedPreferences (лістинг 3.20).

Рисунок 3.6 – Видалення цілі змахом вліво

Лістинг 3.20 – Видалення цілі накопичення

```
Dismissible(
  key: Key(goal['id'].toString()),
  direction: DismissDirection.endToStart,
  onDismissed: (direction) {
    dataProvider.deleteElementG(i);
    dataProvider.clearCollected(goal['id']);
  },
```

3.2.5 Підкаталог settings

Підкаталог settings містить файл, у якому реалізовано екран налаштувань. Він надає користувачеві можливість:

- додавати, оновлювати та видаляти ім'я: користувач може вказати або змінити своє ім'я, а також видалити його (рисунок 3.7);
- змінювати тему застосунку (доступні світла та темна теми). Перемикання між ними реалізоване через віджет TextButton (кнопка-текст) (лістинг 3.21);
- очищувати всі дані: кнопка, яка дозволяє видалити всі транзакції та цілі накопичення, що повертає додаток до початкового стану (лістинг 3.22).

Рисунок 3.7 – Діалогове вікно для управління іменем користувача

Лістинг 3.21 – Зміна теми застосунку

```
TextButton(
  onPressed: () {
    dataProvider.toggleTheme();
  },
  style: TextButton.styleFrom(
    backgroundColor: dataProvider.isDark
        ? Theme.of(context).colorScheme.outline
        : Theme.of(context).colorScheme.onPrimary,
    shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(12)),
  ),
  child: CustomTextWidget(
    text: dataProvider.isDark ? 'Dark Theme' : 'Light Theme',
    styleType: TextStyleType.h3,
  ), ),
```

Лістинг 3.22 – Очищення даних

```
TextButton(
  onPressed: () {
    DataErasure.deleteAllData(context);
  },
  style: TextButton.styleFrom(
    backgroundColor: Theme.of(context).colorScheme.onPrimary,
    shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(12)),
  ),
  child: Row(
    children: [
      Icon(
        CupertinoIcons.delete,
        size: 20,
        color: Theme.of(context).colorScheme.onSurface,
      ),
      const SizedBox(width: 5),
      const CustomTextWidget(
        text: 'Delete all data',
        styleType: TextStyleType.h3,
      ),    ],  ),  ),
```

Наведені вище функції, забезпечують персоналізацію інтерфейсу та управління збереженими даними.

3.2.6 Підкаталог stats

Підкаталог stats містить три файли:

- stats.dart: реалізує екран статистики, де користувач бачить діаграму щоденної фінансової активності. А завдяки використанню віджета GestureDetector, при подвійному натисканні на неї знизу з'являється ще одна діаграма, яка відображає тижневу активність. Вона зберігає дані за останні чотири тижні, що дозволяє користувачу аналізувати прибутки та витрати в рамках місяця (лістинг 3.23). Нижче діаграм розташовані списки-пункти, які дозволяють переглянути деталізовану інформацію по кожному типу транзакцій, а також відкрити щомісячний звіт (рисунок 3.8);

- файли day_chart.dart та week_chart.dart відповідають за побудову відповідних діаграм. Для їх реалізації використовується плагін fl_chart, який

надає широкі можливості для візуалізації даних. Дані зберігаються за допомогою SharedPreferences, що дозволяє зберігати та оновлювати статистику без необхідності підключення до зовнішнього сервера.

Для кожного з цих файлів було написано такі функції:

- завантаження даних при вході користувачем в даний екран;
- оновлення даних на екрані;
- збереження даних локально, в пам’яті пристрою;
- видалення даних після закінчення терміну (неділя або тиждень).

Лістинг 3.23 – Відображення діаграм

```
GestureDetector(
  onDoubleTap: () {
    setState(() {
      isExpanded = !isExpanded;
    });  },
  child: ChartContainerWidget(
    isExpanded: isExpanded,
    isTop: true,
    child: const MyDayChart(),
  ),),
if (isExpanded)
  ChartContainerWidget(
    isExpanded: isExpanded,
    isTop: false,
    child: const WeekChart(),  ),
```

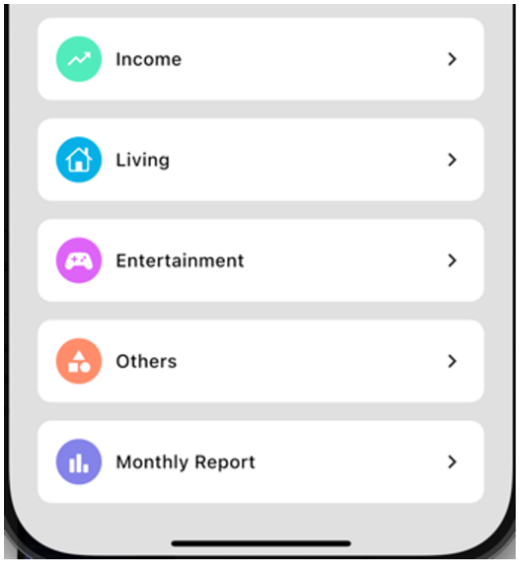


Рисунок 3.8 – Списки-пункти екрану stats

3.3 Файл data_provider.dart

Файл data_provider.dart відповідає за управління даними в застосунку. Завдяки використанню плагіна Provider, який забезпечує зручний механізм для управління станами, можна легко та швидко оновлювати дані у всьому інтерфейсі застосунку. Наприклад, коли до бази даних додаються або видаляються дані, зміни автоматично відображаються у відповідних елементах інтерфейсу без необхідності вручну оновлювати екран (лістинг 3.14). Також використання Provider дозволяє зберігати чистоту коду.

У цьому файлі реалізовано зміну теми застосунку (лістинг 3.24), завантаження транзакцій з бази даних та їх оновлення при додаванні нових або видаленні існуючих, а також управління цілями накопичення, включаючи їх отримання, оновлення та видалення (лістинг 3.25). Оскільки дані зберігаються локально, передбачена функція очищення старих транзакцій, що допомагає підтримувати продуктивність застосунку на високому рівні.

Лістинг 3.24 – Функції для зміни теми застосунку

```
bool isDark = false;
  void toggleTheme() async {
    isDark = !isDark; // Переключаем тему
    await _saveTheme();
    notifyListeners();
  }
  Future<void> _saveTheme() async {
    final prefs = await SharedPreferences.getInstance();
    await prefs.setBool('isDark', isDark);
  }
  Future<void> loadTheme() async {
    final prefs = await SharedPreferences.getInstance();
    isDark = prefs.getBool('isDark') ?? false;
    notifyListeners();
  }
```

Лістинг 3.25 – Функції управління цілями накопичення

```
Future<void> loadGoals() async {
    goals = await AppDatabase.instance.getGoals();
    notifyListeners();
  }
```

```
Future<void> deleteElementG(int i) async {
    if (i >= 0 && i < goals.length) {
      int id = goals[i]['id'];
      await AppDatabase.instance.deleteRecord(id);
      await loadGoals();
    }  }
```

3.4 Файл db_helper.dart

Файл db_helper.dart містить клас AppDatabase, який відповідає за роботу з базою даних застосунку. У цьому класі реалізовані функції ініціалізації бази даних (лістинг 3.26), створення таблиць (лістинг 3.27), а також функції для додавання, видалення та управління даними (лістинг 3.28).

Лістинг 3.26 – Функція ініціалізації бази даних

```
Future<Database> _initDB(String fileName) async {
final directory = await getApplicationDocumentsDirectory();
final path =
join(directory.path, fileName);
return await openDatabase(path, version: 1, onCreate:
_createDB);    }
```

Лістинг 3.27 – Функція створення таблиці в базі даних

```
Future _createDB(Database db, int version) async {
const idType = 'INTEGER PRIMARY KEY AUTOINCREMENT';
const textType = 'TEXT';
const realType = 'REAL';
await db.execute('''
  CREATE TABLE transactions_and_goals (
    id $idType,
    type $textType NOT NULL,
    category $textType,
    title $textType NOT NULL,
    icon $textType NOT NULL,
    color $textType NOT NULL,
    amount $realType NOT NULL,
    date $textType NOT NULL     )
  ''');   }
```

Лістинг 3.28 – Приклади функцій для управління даними

```
Future<List<Map<String, dynamic>>> getTransactions() async {
final db = await database;
return await db.query('transactions_and_goals',
```

```
    where: 'type = ?', whereArgs: ['transaction'], orderBy: 'id
DESC');   }
Future<List<Map<String, dynamic>>> getGoals() async {
final db = await database;
return await db.query('transactions_and_goals',
    where: 'type = ?', whereArgs: ['goal'], orderBy: 'id DESC');
}
Future<void> deleteRecord(int id) async {
final db = await database;
await db.delete(
  'transactions_and_goals',
  where: 'id = ?',
  whereArgs: [id],  ); }
```

Всі функції для роботи з базою даних є асинхронними, тому використовують об’єкт Future. Даний об’єкт вказує на те, що певна дія, зазвичай це тривалі операції як запит до бази даних, завантаження з серверу чи читання файлів, виконується в фоновому режимі, щоб не блокувати основний потік програми та залишити інтерфейс плавним та швидким.

У проєкті використовується SQLite – легка вбудована база даних, яка не потребує окремого серверу для роботи [14]. Це робить її ідеальним вибором для мобільних застосунків, оскільки дані зберігаються локально на пристрої, що забезпечує доступ до них навіть без підключення до мережі.

3.5 Файл main.dart

Файл Main.dart є точкою входу в застосунок, в якому відбувається ініціалізація бази даних, запускається сам застосунок через метод runApp та використовується ChangeNotifierProvider для організації передавання об’єкту DataProvider у весь програму (лістинг 3.29).

Лістинг 3.29 – Функція main

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await AppDatabase.instance.database;
  runApp(ChangeNotifierProvider(
      create: (context) => DataProvider(), child: const
MyApp()));     }
```

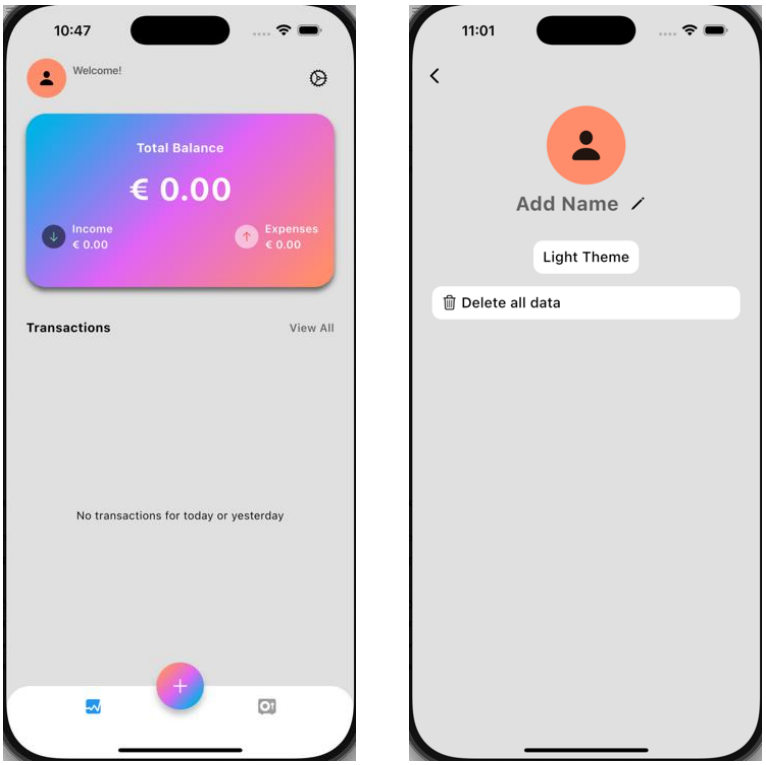
Метод runApp запускає застосунок, передаючи у нього клас MyApp, який використовує метод initState() для початкового завантаження теми через DataProvider, після чого повертає віджет MaterialApp, у якому визначається поточна тема застосунку та встановлюється головний екран, представлений класом HomeScreen (лістинг 3.30).

Лістинг 3.30 – Клас MyApp

```
class  MyAppState extends State<MyApp> {
  @override
  void initState() {
    super.initState();
    final dataProvider = context.read<DataProvider>();
    dataProvider.loadTheme();  }
  @override
  Widget build(BuildContext context) {
    final dataProvider = context.watch<DataProvider>();
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: "Lifin",
      theme: lightTheme,
      darkTheme: darkTheme,
      themeMode: dataProvider.isDark ? ThemeMode.dark :
ThemeMode.light,
      home: const HomeScreen(),
    );
}   }
```

4 ІНСТРУКЦІЯ КОРИСТУВАЧА

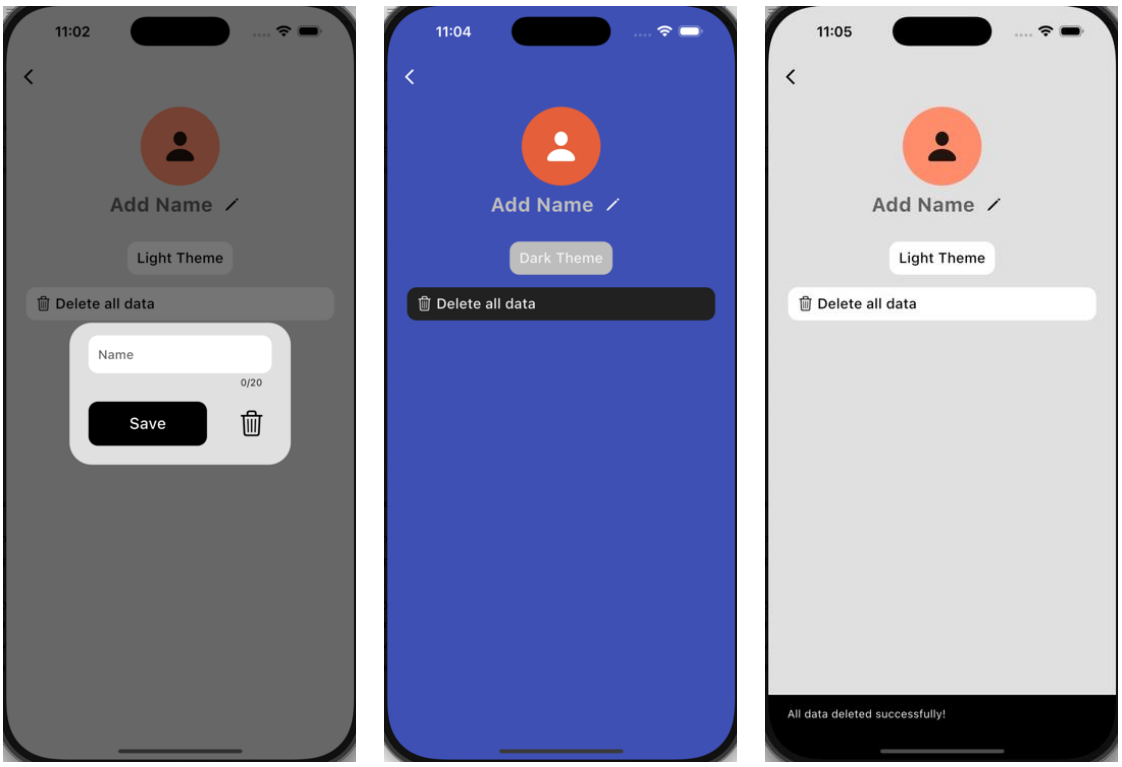
Після запуску застосунку користувач бачить "Головний екран" (рисунок 4.1 а), на якому відображається текст з привітанням, кнопка "Налаштування", поточний баланс, кнопка "View All", список транзакцій та меню навігації (в самому низу). Натискання на іконку шестірні відкриває екран "Налаштування" (рисунок 4.1 б), де можна додати ім’я (рисунок 4.2 а), змінити тему (рисунок 4.2 б) та видалити всі дані (рисунок 4.2 в).



а) б)

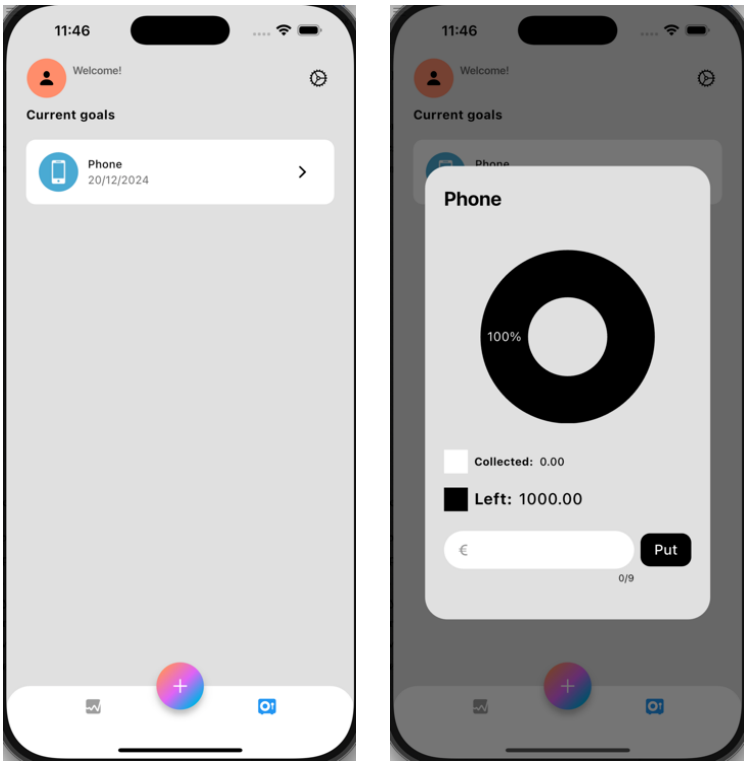
Рисунок 4.1 – Скриншоти застосунку: а) головний екран застосунку; б) екран "Налаштування"

При натисканні на іконку сейфа в меню навігації відкриється екран "Скарбнича" (рисунок 4.3 а), де відображаються поточні цілі. При натисканні на кожну з них буде відкриватись діалогове вікно, де показано стан відповідної цілі (рисунок 4.3 б). Крім того в цьому вікні можна і "покласти" гроші.



а) б) в)

Рисунок 4.2 – Функції екрану "Налаштування": а) додавання імені; б) зміна теми; в) очищення даних

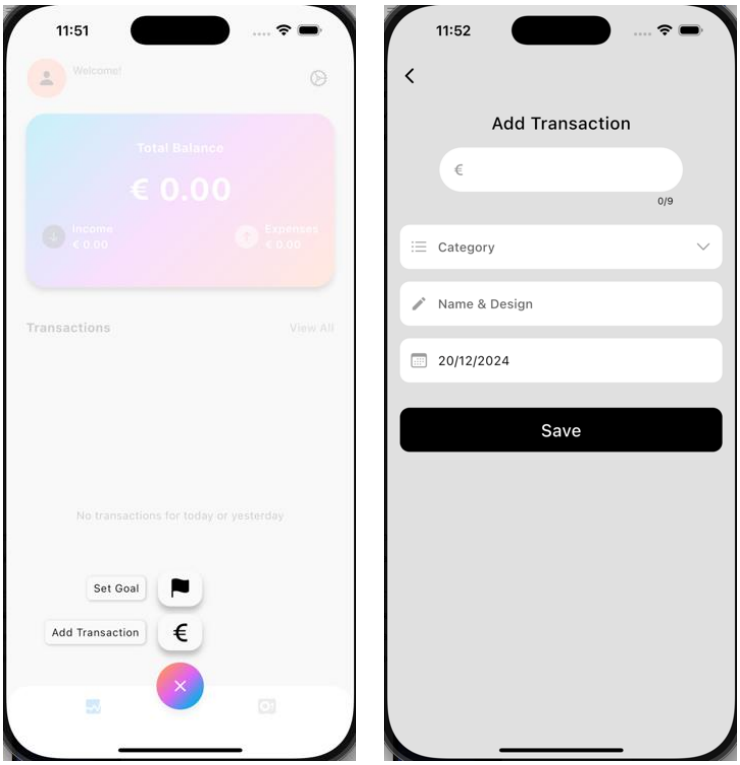


а) б)

Рисунок 4.3 – Екран "Скарбничка":

а) інтерфейс застосунку; б) стан обраної цілі

При натисканні на кнопку "+" на головному екрані, відкриється меню вибору дії: додати транзакцію, поставити ціль (рисунок 4.4 а). Натиснувши на один з варіантів, відкривається відповідний екран (рисунок 4.4 б), на якому відображені поля, які треба заповнити.

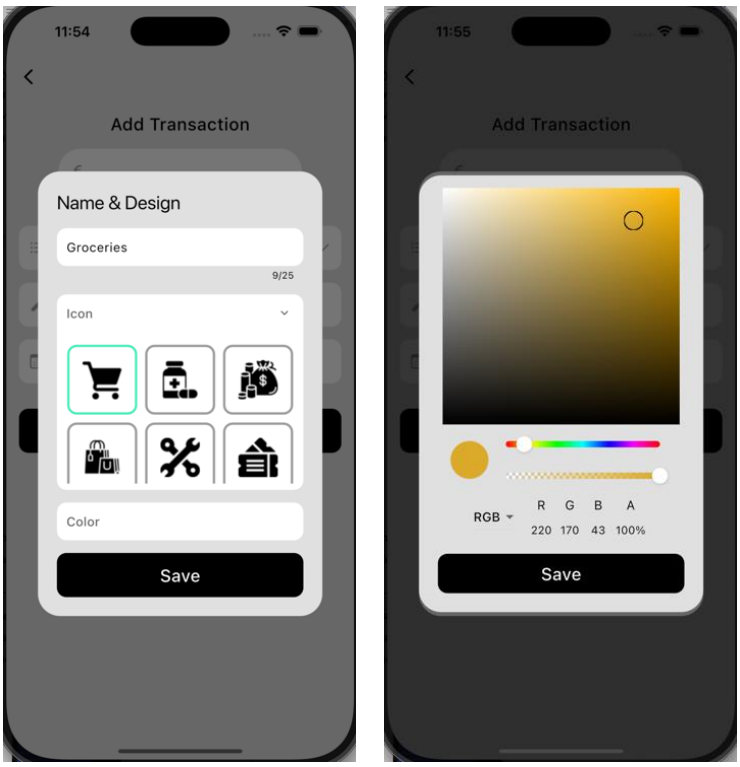


а) б)

Рисунок 4.4 – Дії натискання кнопки "+": а) результат натискання; б) результат вибору однієї з опцій

В верхньому полі треба ввести суму транзакції (дане поле підтримує лише цифри і символи "." та ","). При натисканні на поле "Category" випадає список категорій, одну з яких треба обрати. При натисканні на поле "Name & Design" відкривається діалогове вікно, де треба ввести назву, обрати іконку із запропонованого каталогу (рисунок 4.5 а) та колір (рисунок 4.5 б). Останнє поле завжди відображає поточну дату та служить лише як інформаційне поле, тобто не можна створити транзакцію на іншу дату.

Заповнивши всі поля та натиснувши кнопку "Save" екран закриється та на головному екрані відобразиться транзакція (рисунок 4.6).



а) б)

Рисунок 4.5 – Дії з діалоговим вікном: а) заповнення поля та вибір іконки; б) вибір кольору

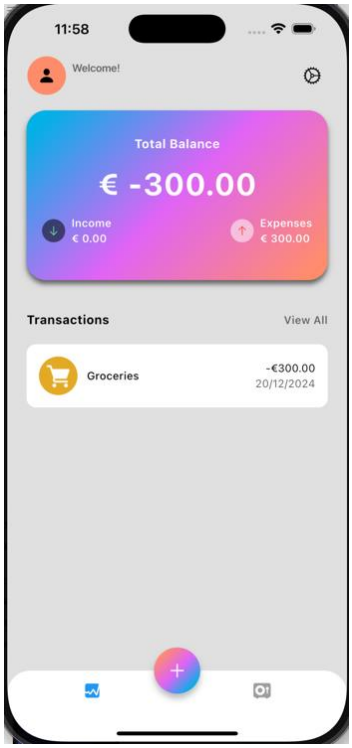
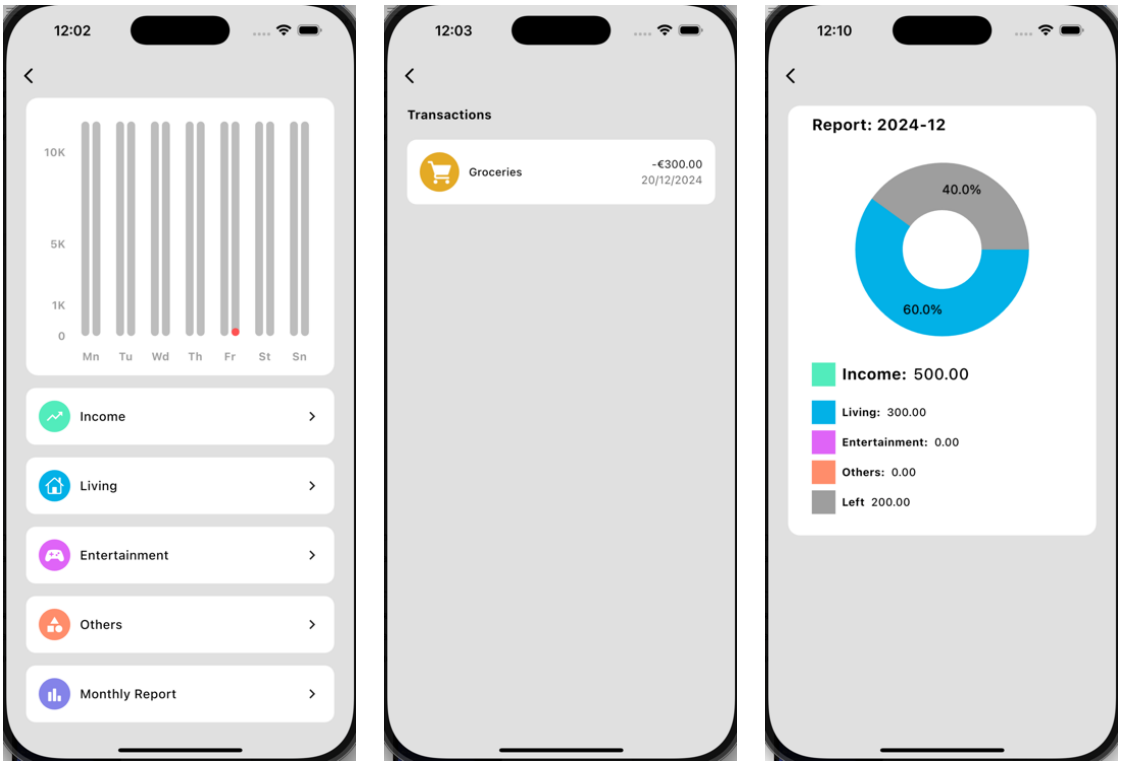


Рисунок 4.6 – Відображення доданої транзакції на головному екрані

При натисканні на кнопку-текст "View All" відкривається сторінка з діаграмою щоденної активності (рисунок 4.7 а) та списком категорій

транзакцій, при натисканні на кожен з них відкривається екран зі списком всіх транзакцій обраної категорії (рисунок 4.7 б). В кінці списку є пункт "Місячний звіт", при натисканні на який відкривається екран із звітом за поточний місяць (рисунок 3.10 в).



а) б) в)

Рисунок 4.7 – Натискання на кнопку "View All": а) екран статистики та звітів; б) екран з транзакціями відповідної категорії; в) екран звітів

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було проведено аналіз існуючих застосунків, який підтвердив необхідність розробки продукту, націленого на аудиторію, яка надає перевагу готівковим розрахункам. Саме це було взято за мету і кінцевий результат даної роботи.

Після вибору відповідних технологій розробки було створено мобільний застосунок для слідкування за власним бюджетом із функцією скарбнички. Застосунок, який було спроєктовано, успішно виконує основну мету – надає користувачам простий засіб для управління фінансами, хоча може поступатися популярним аналогам за функціональністю.

Результат роботи має наразі деякі недоліки: архітектура, яка використана у проєкті, є дещо примітивною, що не повністю відповідає сучасним принципам розробки програмних продуктів. Однак через невеликий масштаб проєкту застосування flat architecture є виправданим та спрощує розробку.

У перспективі розвитку програми можна впровадити такі вдосконалення:

- інтеграція з банківськими рахунками для автоматичного обліку фінансів;
- реалізація функції хмарного збереження даних для підвищення безпеки та зручності;
- створення інтерактивного та автоматизованого функціоналу (рекомендації щодо економії);
- можливість синхронізації з різних пристроїв для сімейного або групового використання.

Ці вдосконалення підвищать конкурентоспроможність застосунку та забезпечать більшу зручність для користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Бояков І.С., Колтун Ю.М., Філімончук Т.В. Мобільний застосунок для ведення бюджету. Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління: тези доповідей п’ятнадцятої міжнародної науково-технічної конференції. Т.2: секція 2. Баку: ІСУ АР; Харків: НТУ «ХПІ»;. Харків: ХНУРЕ; Харків: НАУ «ХАІ»; Жиліна: УмЖ. 2025. с. 32.

2. Best Budgeting Apps Of 2025 https://www.forbes.com/advisor/banking/best-budgeting-apps/

3. Barrow C. The 30 Day MBA in Business Finance: Your Fast Track Guide to Business Success. Publisher: Kogan Page, 2023. 264 p.

4. Smith W., Wayte W., Marindin G. E. A Dictionary of Greek and Roman Antiquities. London: John Murray, Albemarle Street, 1890. 2125 p.

5. Топ-5 фінансових додатків для управління бюджетом. URL: https://ideabank.ua/uk/experts/top-5-finansovykh-dodatkiv-dlya-upravlinnya-byudzhetom.

6. MacCallum K., McDonald J. Non-functional Requirements in Systems Analysis and Design. Cham: Springer, 2016. 156 p.

7. Карнегі Д. Як здобувати друзів і впливати на людей. Київ: Книжковий Клуб «Клуб Сімейного Дозвілля», 2020. 320 с.

8. Leler W. Why Flutter Uses Dart | HackerNoon. HackerNoon – read, write and learn about any technology. URL: https://hackernoon.com/why-flutter-uses-dart-dd635a054ebf

9. Guzzi V., Moore K. D., Katz M., Ngo V. Flutter apprentice: learn to build cross-platform apps. Razeware LLC, 2022. 683 p.

10. Visual Studio Code: Code Editing. Redefined. URL: https://code.visualstudio.com

11. Kushwaha A., Singh H. Android Programming for Beginners: Build in-depth, full-featured Android apps starting from zero programming experience.

Packt Publishing, 2021. 798 p.

12. Mark D., Reischl K., Lehn M. iOS Apprentice: Beginning iOS development with Swift. Razeware LLC, 2021. 670 p.

13. Chacon S., Straub B. Pro Git. Publisher: Apress, 2nd edition. 2014. 440 p.

14. Owens M. The Definitive Guide to SQLite. 2nd ed. Apress, 2010. 368 p.