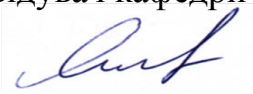
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут комп'ютерних наук та управління проектами
Кафедра інформаційних управляючих систем та технологій

"Допущений до захисту"

Завідувач кафедри ІУСТ



к.т.н, доц. Михелєв І.Л.

"____" _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти "бакалавр"

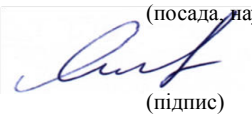
на тему: "Розробка інформаційної системи планування бюджету і контроля витрат фізичної особи"

Виконав: студент групи 4141

(підпис) *Сорочан А.В.*

Керівник роботи:

_____
(посада, науковий ступінь вчене звання)



(підпис) *Михелєв І.Л.*

м. Миколаїв – 2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут комп'ютерних наук та управління проектами

Кафедра інформаційних управляючих систем та технологій

Спеціальність 122 "Комп'ютерні науки"

Освітня програма "Комп'ютерні науки"

"ЗАТВЕРДЖУЮ"Гарант
освітньої програми

к.т.н, доц. Гайда А.Ю.

"____" _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
на здобуття ступеня вищої освіти "бакалавр"

Студенту Сорочану Артему Валерійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи:
Розробка інформаційної системи планування бюджету і контроля витрат фізичної особи

ерівник роботи: Михелєв Ігор Леонідович

Затверджені наказом ректора № 295-уч від "27" березня 2025 року.

2. Термін подання роботи: 20 червня 2025 р.

3. Вихідні дані до проекту (роботи) ДСТУ щодо обробки інформації, літературні джерела, технічна документація на існуючі аналоги систем.

4. Перелік питань, що належать до розробки (найменування розділів)

- Титульний аркуш, завдання на дипломну роботу, анотація (українською, англійською), зміст, перелік умовних позначень, символів, одиниць та термінів.
- Вступ
- Аналіз предметної області і постановка задачі
- Визначення вимог до системи
- Моделювання предметної області
- Вибір архітектури

- Проектування структури бази даних
- Моделювання
- Проектування інтерфейсу користувача
- Вибір інструментів розробки
- Реалізація та тестування системи
- Безпека та захист персональних даних
- Охорона праці та безпека у надзвичайних ситуаціях
- Висновки
- Список використаної літератури
- Додатки

5. Перелік презентаційних матеріалів:

5.1 Архітектура системи

5.2 Інтерфейс головних екранів користувача

5.3 Інтерфейс екранів додавання витрат і бюджета

5.4 Схема роботи БД і авторизації

5.5 Схема захисту БД

5.6 Умови праці на робочому місці

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Михелєв І.Л.	03.02.2025	22.02.2025
2	Михелєв І.Л.	22.02.2025	21.03.2025
3	Михелєв І.Л.	21.03.2025	25.04.2025
4	Михелєв І.Л.	25.04.2025	07.06.2025

7. Дата видачі завдання 03 лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів роботи	Примітка

1	Аналіз предметної галузі	13.02.2025	
2	Постановка задачі	15.02.2025	
3	Розробка концепції	22.02.2025	
4	Розробка проєкту ІС	21.03.2025	
5	Реалізація проєкту	25.04.2025	
6	Розробка питань з охорони праці	07.06.2025	
7	Розробка графічних матеріалів	12.06.2025	
8	Подання роботи на перевірку на плагіат	10.06.2025	
9	Подання роботи рецензенту	15.06.2025	
10	Подання роботи на захист	20.06.2025	

Студент  *Сорочан А.В.*

(підпис) (ПІБ)

Керівник роботи  *Михелєв І.Л.*

(підпис) (ПІБ)

АНОТАЦІЯ

У цій дипломній роботі представлено розробку мобільного веб застосунку, який надає просунуті функції користувачам у сфері логування витрат і контролю власного бюджету фізичним обличчям з фокусом на простоту використання для кожного.

Дипломна робота виконана на 64 сторінках та містить: 30 рисунків та 15 використаних джерел.

Робота оформлена українською мовою.

Ключові слова: інформаційна система, контроль витрат, планування бюджету, кросплатформена розробка, хмарні технології, безпека даних.

ANNOTATION

This work presents the development of a mobile web application that provides advanced features to users in expense logging and budget control by individuals with a focus on ease of use for everyone.
The work is completed on 64 pages and contains: 30 pictures and 15 sources used.
The work is written in Ukrainian.

Keywords: information system, spending control, budget planning, cross-platform development, cloud technologies, data security.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ.

MVP (Мінімально життєздатний продукт) – це найпростіша версія продукту, яку потрібно створити, щоб відправити його на ринок;

UML - або Уніфікована мова моделювання, - це стандартизована мова візуального моделювання, що використовується для створення діаграм, що відображають структуру та поведінку програмних систем;

БД - база даних;

UX - скорочення від User Experience (Користувацький досвід), стосується загальних відчуттів та сприйняття людини щодо взаємодії з продуктом, системою чи послугою;

UI - скорочення від User Interface (Користувацький інтерфейс) – це спосіб взаємодії користувача з пристроєм, програмою або веб-сайтом;

GDPR - Загальний регламент про захист даних – це регламент Європейського Союзу (ЄС) щодо конфіденційності інформації та захисту даних;

Android - одна з найпопулярніших мобільних операційних систем (ОС), розроблена компанією Google;

iOS - одна з найпопулярніших мобільних операційних систем (ОС), розроблена компанією Apple;

Web-додаток - розподілений додаток, у якому клієнтом є браузер, а сервером — веб сервер;

Server-less - (Безсерверні обчислення) це хмарний підхід, де розробники створюють та запускають додатки без керування серверами; Постачальники хмарних технологій займаються інфраструктурою, дозволяючи розробникам зосередитися на коді та логіці додатків;

SDK - розшифровується як Software Development Kit (Комплект розробки програмного забезпечення). Це набір інструментів та ресурсів, які допомагають розробникам створювати програмне забезпечення для певної платформи, мови програмування або системи;

ПЗВ - Пристрій захисного відключення.

ВСТУП

Актуальність теми

Багато людей в сучасному світі мають труднощі у плануванні особистих фінансів. Судячи з останнього часу, розвиток цифрових технологій дає можливості для розробки досить ефективних рішень у сфері особистого фінансового обліку та контролю витрат. Мобільні девайси, які фактично заполонили життя кожної людини, мають великий вплив на людей.
Хоча і мобільний банкінг є дуже популярним рішенням, воно не враховує різні джерела витрат, такі як покупки за готівку, або ситуація коли користувач має декілька банків, і дані між ними ніяк не синхронізуються.

Тому виходячи з цього, розробка інформаційної системи для планування бюджеті і контроля витрати з використанням сучасних технологій на сьогоднішній день є актуальним і практично значущим завданням.

Мета і завдання роботи

Ця випускна кваліфікаційна робота має мету створення мобільної інформаційної системи, яка надає можливість користувачу:

- Логувати витрати за категоріями
- Встановлювати бюджет на місяць(і)
- Переглядати витрати списком або на інтеракційному графіку
- Слідкувати за планомірним витрачанням коштів в межах бюджету на графіку.

Завдання:

- Проаналізувати подібні існуючі системи
- Зробити опис вимог до програмного забезпечення
- Розробити архітектуру програми
- Створити прототип і провести його тестування
- Впевнитись у простоті використання, навіть для початкових користувачів.
- Розробляти систему як MVP (мінімально життєздатний продукт).

Об’єкт і предмет дослідження

В якості об’єкта дослідження робота враховує процес планування та контролю особистих витрат фізичної особи.

Предметом є інформаційна система, яка реалізує спеціальні процеси через мобільний інтерфейс застосунку, що працював би на всіх популярних мобільних платформах (Android, iOS, Web).

Методологія

- Аналіз вимог
- Моделювання архітектури (UML-діаграми)
- Розробка інтерфейсу застосунка (UX/UI)
- Зберігання даних (Firebase Firestore)
- Реалізація авторизації (Firebase Auth)
- Оцінка ресурсів що потребуються для розробки

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області

Сучасні інформаційні системи управління особистими фінансами стають все більш і більш популярними. Вони дозволяють користувачам зручно контролювати свої витрати, планувати свій бюджет і приймати кращі фінансові рішення. Поміж популярний рішень на ринку можна виокремити такі мобільні додатки як Monefy, Spendee, YNAB.

Основні функції таких систем є:

- Додавання витрат із зазначенням суми, категорії і короткого опису.
- Створення і контроль місячного бюджету.
- Візуалізація фінансових даних у вигляді графіків.
- Авторизация користувача і збереження даних у хмарі.

Згідно з цим аналізом, виявилось що значна частина існуючих рішень є багатофункціональною, але деякі надто складні для звичайних користувачів, або мають свої обмеження безкоштовної версії. Отже розроблення власної адаптивної системи, що орієнтована на базові потреби користувача і простоту використання є потрібним.

Окремо, був проведен аналіз існуючих банківських додатків (Privat24), і виявлено брак можливості зручного логування витрат що здійснюються поза банківською системою.

Додатково варто зазначити, що наявність аналогів серед мобільних додатків є ознакою здорової конкуренції, яка сприяє інновація та якості. У ринкових умовах, конкуренція спонукає розробників шукати нові підходи, вдосконалювати інтерфейс, підвищувати зручність і функції свої рішень. Завдяки цьому, користувачі отримують можливість вибору поміж різних концепцій і реалізацій однієї задачі, що стимулює постійний розвиток галузі [1].

1.2 Визначення вимог до системи

Функціональні вимоги

- Реєстрація та авторизація користувача (пошта і пароль)
- Додавання витрати (сума, назва, категорія).
- Створення місячного бюджету.
- Виведення графіків витрат і бюджету.
- Скидання пароля через поштову адресу
- Видалення аккаунту користувачем (згідно з вимогами захисту прав користувачів GDPR [2])

Чому є необхідність надати користувачу можливість видалити свої дані?
Це важливо ознака етичного підходу до обробки інформації. У сучасному цифровому середовищі, де конфіденційність та безпека особистих даних мають особливу цінність, права на “забуття” є базовим принципом, закріпленим у міжнародних стандартах захисту персональних даних. Для користувача це створює відчуття контролю над власною інформацією та значно підвищує рівень довіри до додатку. Це відповідає принципам прозорості та відповідальності з боку розробника.

Нефункціональні вимоги:

- Кросплатформеність (Android/iOS/Web).
- Інтуїтивно зрозумілий інтерфейс.
- Реалізація за допомогою сучасного фреймворку Flutter.
- Збереження даних у хмарному сховищу Firebase Firestore.
- Безпечна авторизація користувачів через хмарний сервіс Firebase Auth.
- Розробка у паттерні MVP (мінімально життєздатний продукт).

Поточна ситуація на ринку, налаштована на реалізацію систем на різних платформах. Різні користувачі мають різні навички і звички. Для деяких зручно використовувати специфічні додатки своєї операційної системи (Android/iOS), для інших використання браузера є невід’ємною частиною повсякденного

життя [3].

Розробка системи для кожної платформи окремо використовуючи нативні рішення було б набагато більше затратним, з точки зору часу, зусиль а також перспектив підтримки система у майбутньому, оскільки це потребує знання одразу декількох мов програмування і патернів розробки.

Саме тому кросплатформеність (робота на різних платформах) є однією з ключових нефункціональних вимог.

Чому саме MVP? Це концепція розробки продукту, яка передбачає створення його найпростішої версії з базовим функціоналом, достатнім для вирішення основної “проблеми” користувача. Основна мета - не створити ідеальний або завершений продукт, а якнайшвидше вийти на ринок, щоб перевірити, чи дійсно продукт цікавий і корисний для потенційних користувачів.

Такий підхід дозволяє зекономити час, ресурси та зусилля, уникаючи розробки непотрібних функцій.

Процес створення починається з глибокого розуміння проблеми, яку потрібно вирішити. Команда аналізує потреби цільової аудиторії, вивчає ринок щодо того, як їх продукт може задовольнити потреби середньостатистичного користувача.

1.3 Постановка задачі

У межах дипломної роботи необхідно реалізувати інформаційну систему у вигляді мобільного веб-додатку.
Основні компоненти системи:

- Клієнтська частина на Flutter, що забезпечує інтерфейс користувача.
- Серверна частина реалізована як інтеграція з хмарними сервісами Firebase (Auth, Firestore).
- Інформаційна структура зберігання даних витрат, категорій і бюджетів.

З орієнтиру на те що система має бути розроблена як MVP, використання таких технологій як Flutter та Firebase є доречним і необхідним.

Flutter — це фреймворк з відкритим кодом для створення красивих, нативно скомпільованих, багатоплатформних застосунків з єдиної кодової бази. Однією з ключових переваг (і причиною роботи на різних платформах) є власний графічний рушій, який дозволяє створювати високопродуктивні та плавні інтерферси, без необхідності використання нативних компонентів.
Flutter підтримує підхід “hot reload”, що значно прискорює розробку, тестуванні та ітерації над інтерфейсом, дозволяючи оновлювати частини кода без значних затримок у режимі розробки [4].
Хоча Flutter є розробкою величезної компанії Google, значну підтримку фрейморку надає open-source контрибуція, що значить що будь-який досвідчений розробник може запросити і внести свої зміни у цю еко-систему.

Рис. 1.3.1 Фреймворк Flutter

Основною мовою програмування у Flutter є Dart, яка поєднує у собі елементи об’єктно-орієнтованого та функціонального підходів. Була створена також компанією Google у 2011 році [5].

Вона поєднує в собі елементи Java, C#, JavaScript та інших популярних мов, забезпечуючи знайомий синтаксис та гнучкість у стилі програмування. Dart підтримує як компільований, так і інтерпретований підходи:

- для мобільних платформ використовується Ahead-of-Time (AOT) компіляція, яка дозволяє досягати ефективної швидкодії
- під час розробки застосовується Just-in-Time (JIT) компіляція, яка забезпечує “hot-reload” - миттєве оновлення інтерфейсу без повного перезавантежжня програми.

Dart має строгу типізацію, але дозволяє і частково працювати з динамічними типами, що робить його збалансованим для швидкої розробки та підтримки великих проєктів.

```dart
if (year >= 2001) {
  print('21st century');
} else if (year >= 1901) {
  print('20th century');
}

for (final object in flybyObjects) {
  print(object);
}

for (int month = 1; month <= 12; month++) {
  print(month);
}

while (year < 2016) {
  year += 1;
}
```

Рис 1.3.2 Приклад синтаксису на мові Dart.

Firebase — це платформа для розробки мобільних додатків, яка допомагає створювати, вдосконалювати та розвивати ваш додаток [6].

Вона включає набір хмарних server-less сервісів, які дозволяють прискорити розробку додатків, пере використовуючи готові рішення, наприклад сервіс авторизації Firebase Auth та хмарного сховища Firestore.

Хоча пропозиція на ринку хмарним платформ є досить велика, і включає в себе таких технологічних гігантів як Amazon AWS Amplify, Microsoft Azure та інших, вони не мають такої сумісної інтеграції і засобів розробки як це зробила компанія Google.

Firebase та Flutter є невід'ємними "компаньонами" у розробці мобільних додатків, оскільки їх засновники надають перевагу інтеграції цих технологій між собою.

Рис. 1.3.3 Платформа хмарних сервісів Firebase.

1.4 Моделювання предметної області

Схеми використання системи користувачем позначені рисунками 1-3.

- Підсистема авторизації

- Підсистема витрат
- Підсистема бюджета

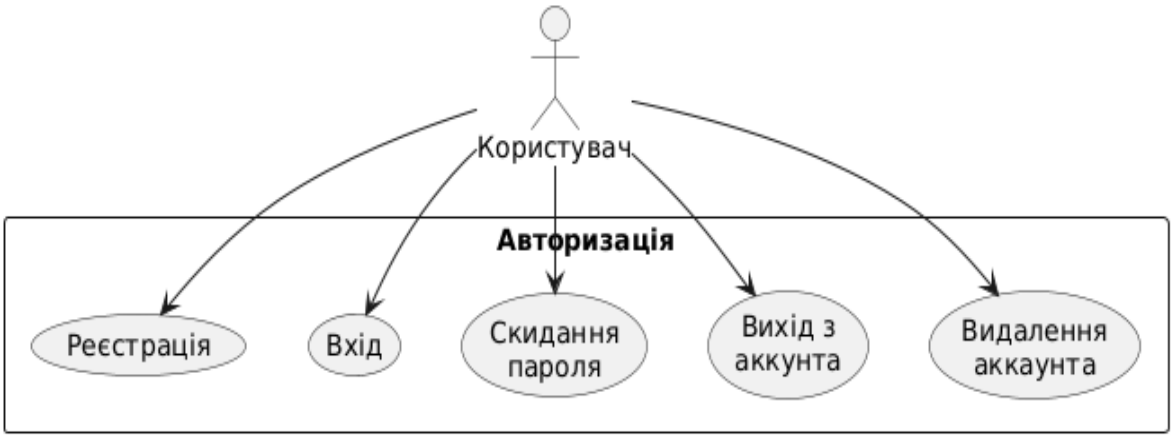


Рис. 1.4.1 Підсистема авторизації

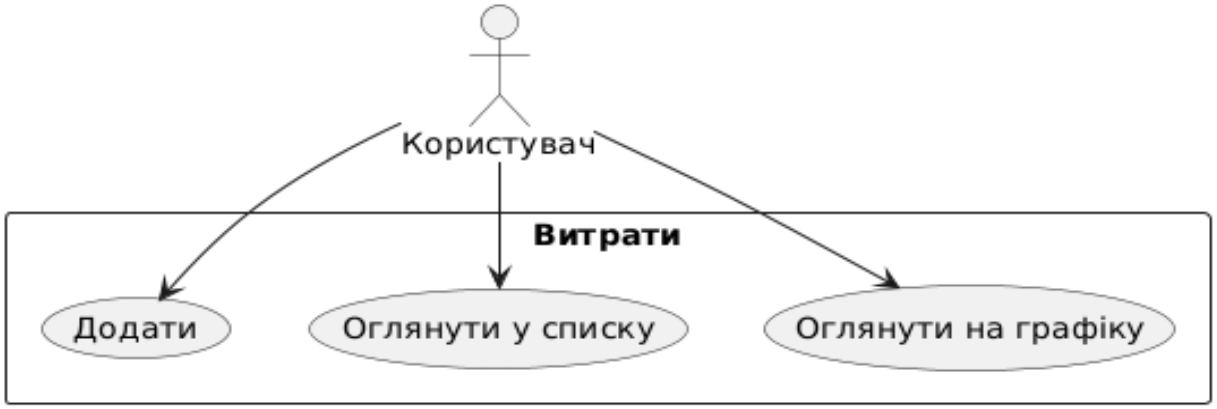


Рис. 1.4.2 Підсистема витрат

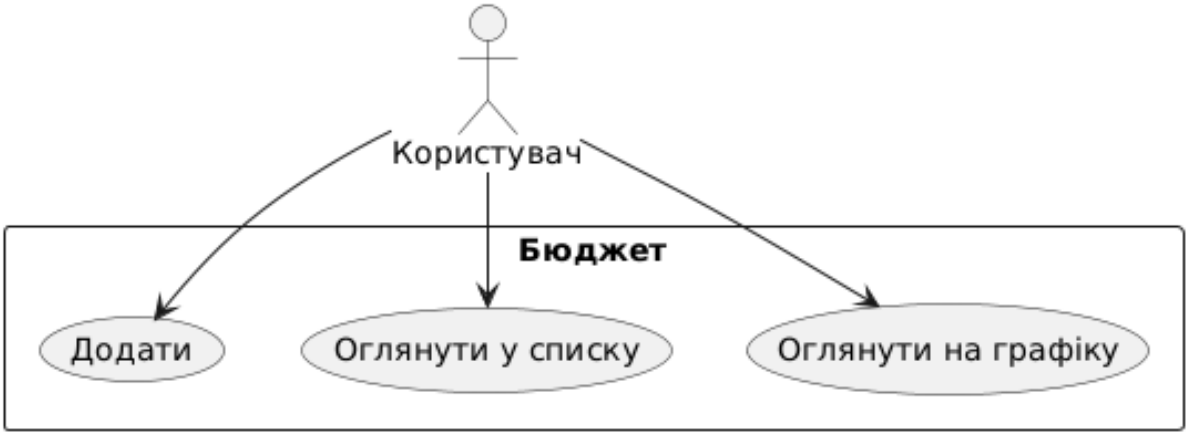


Рис 1.4.3 Підсистема бюджета

1.5 Висновки до розділу 1

За результатами аналізу було визначено основні вимоги до інформаційної системи, і було вивчено особливості існуючих аналогів. Сформульовано задачі, що мають бути вирішені запропонованою системою. Проінформовано щодо технологій що мають бути застосовані у системі і в чому їх вигода. У наступному розділі, буде виконано проектування архітектури і структури бази даних.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Вибір архітектури програмного забезпечення.

Система проекту є клієнт-серверною. Клієнт реалізується у вигляді мобільного застосунку, який створений з використанням сучасного фреймворку Flutter. Серверна частина реалізується за допомогою сервісів хмари Firebase, зокрема:

- Firebase Authentication - для реєстрації/логіну/скидання пароля.
- Cloud Firestore - для зберігання даних витрат, бюджету користувача.

Це оптимальний вибір, для системи що має бути розроблена як MVP, бо значна частина серверного коду вже буде готова і має бути використана лише як інтеграція з клієнтом (мобільним додатком) і немає потреби у створенні і розгортанні власного сервера і бази даних (і згідно усіх інших зв’язаних речей).

2.2 Проектування структури бази даних

Зберігання даних реалізовано у **NoSQL** базі - Firebase Firestore.

NoSQL-бази даних – це нереляційні системи, які зберігають дані інакше, ніж традиційні реляційні бази даних. Вони пропонують гнучкість у моделюванні даних, підтримують різні формати, такі як бази даних ключ-значення, документи та графи, і чудово справляються з обробкою великих неструктурованих наборів даних [7].

Поява NoSQL було обумовлена обмеженнями класичних реляційних СУБД у сфері:

- масштабування - SQL-бази масштабуються переважно вертикально (за рахунок потужнішого сервера), тоді як NoSQL - горизонтально (додаванням нових вузлів).
- Гнучкість в роботі зі структурами даних - у багатьох застосунках структура даних змінюється динамічно, тому фіксовані схеми у SQL створюють додаткові труднощі

- Продуктивності при високому навантаженні - у випадку великої кількості читань/записів SQL-бази потребують складної оптимізації.

Firebase Firestore - це один з хмарним сервісів Firebase, що дозволяє використовувати обгортку у вигляді власного **SDK**, для зручної інтеграції між клієнт-додатком та сервером [8].

У Firestore основною одиницею зберігання даних є документ, який зберігається в колекціях. Система організована у вигляді вкладених структур, які нагадуються JSON або дерево.

Колекція (Collection)

- це контейнер для документів
- колекції не містять інших колекцій напряму, але документи можуть містити підколекції.

Документ (Document)

- це одиниця зберігання даних (аналог запису)
- містить пари ключ-значення
- кожен документ має унікальний ID в межах своєї колекції.
- документи можуть містити підколекції, що дозволяє створювати вкладені структури.

Структура бази даних передбачає такі колекції:

- spendings;
- budgets.

На рисунку 2.2.1 зображена UML схема розташування колекцій і документів бази даних.

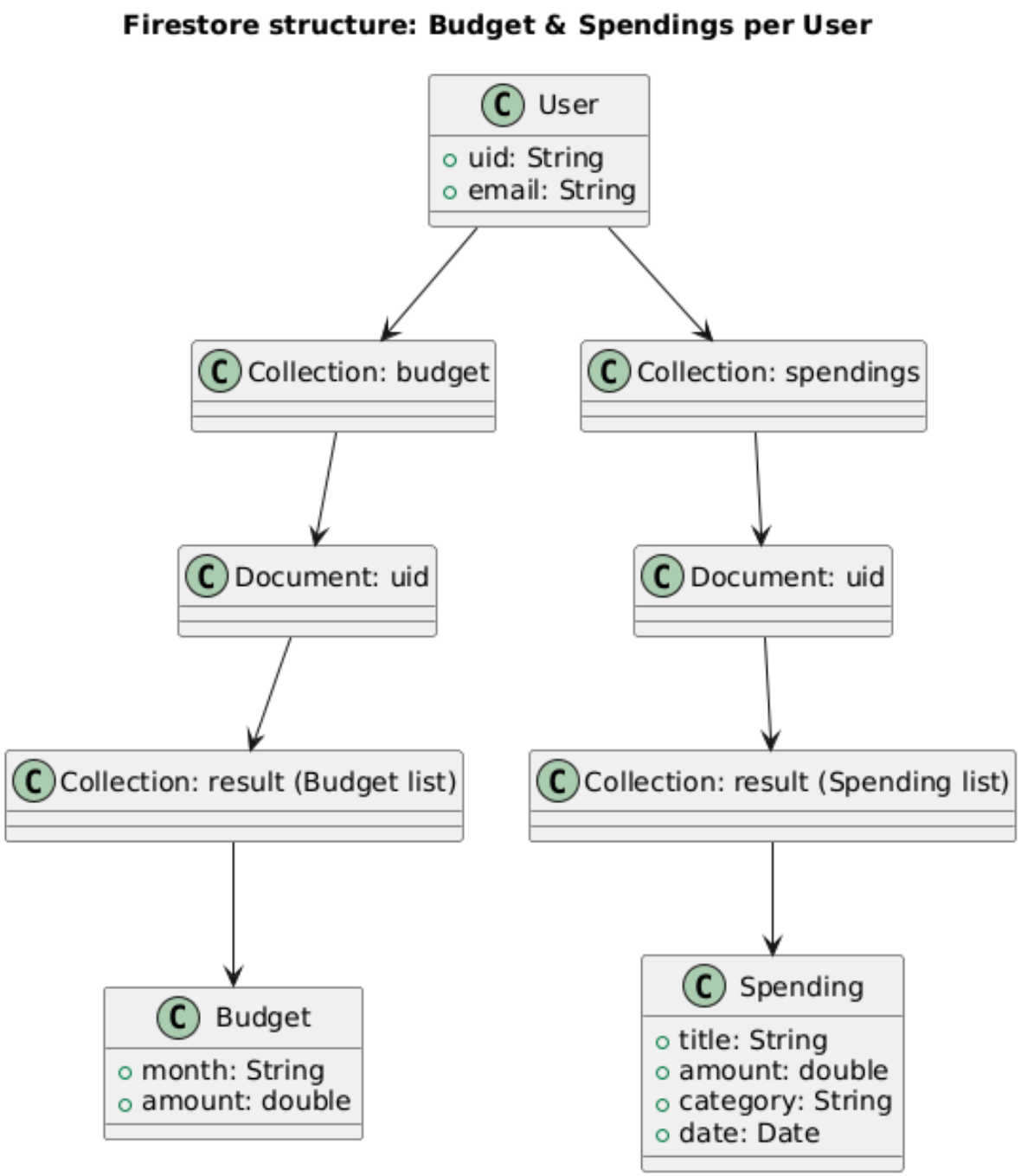


Рис. 2.2.1. Колекції і документи в БД Firestore.

2.3 Моделювання

UML-діаграма основних класів на рисунку 4 включає:

- User (uid, email)
- Spending (amount, category, title, date)

- Budget (month, amount)

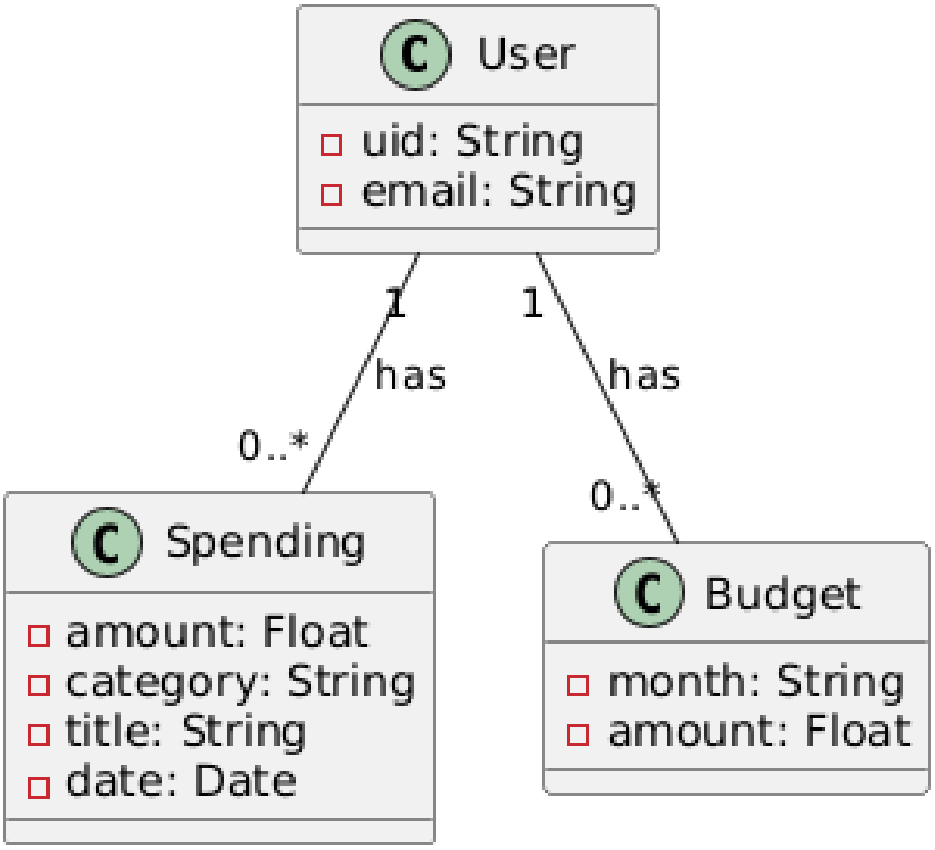


Рис. 2.3.1

2.4 Проектування інтерфейсу користувача

Принципи UX/UI:

- Мінімалізм та зручність
- Колірна схема з акцентом на категорії витрат
- Графіки витрат/бюджету - інформативні, інтерактивні.

Система використовує сучасну дизайн систему **Material Design**.

Material Design - це адаптивна система інструкцій, компонентів та інструментів, що підтримують найкращі практики дизайну користувацького інтерфейсу [9].

Ця система грунтується на фізичних властивостях матеріалу - наприклад карти, тіні, світло, глибина.

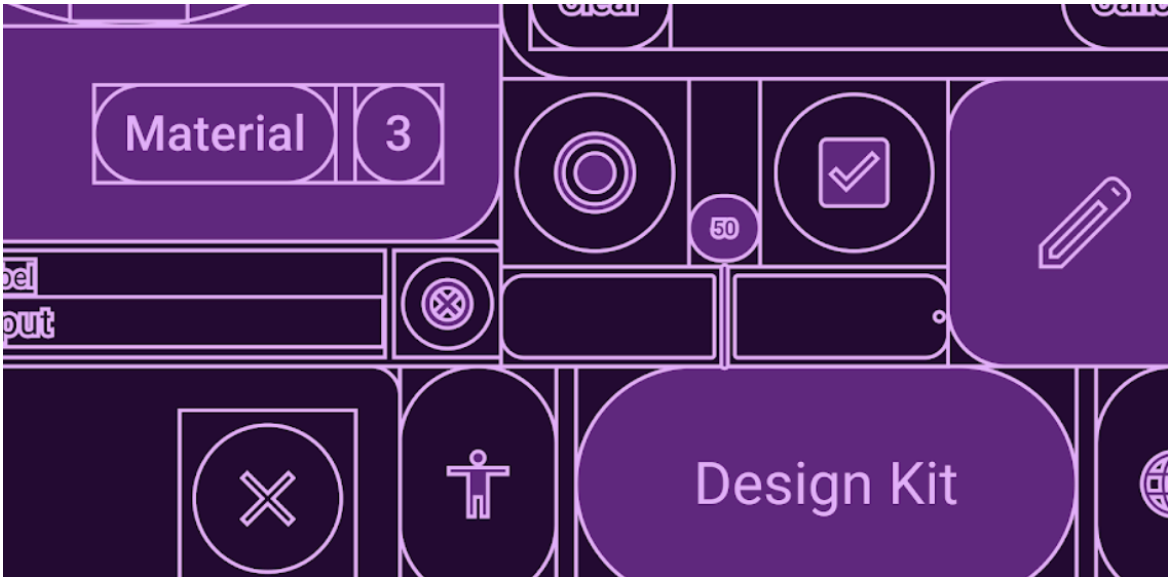


Рис. 2.4.1 Material design

Основні схеми використані в системі позначені наступними рисунками.

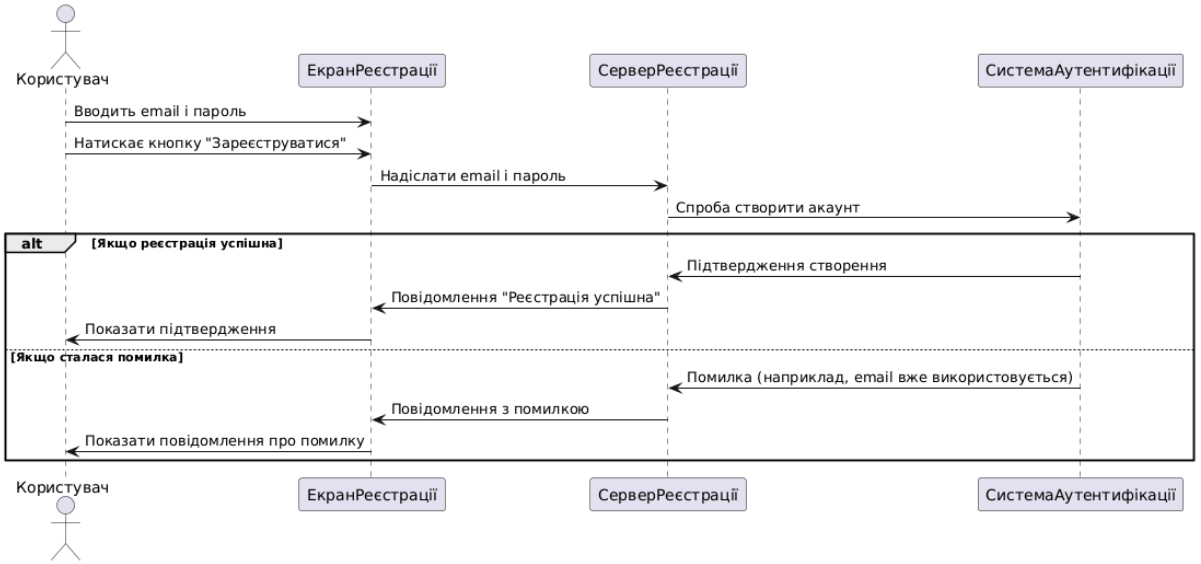


Рис. 2.4.1. Реєстрація і вхід

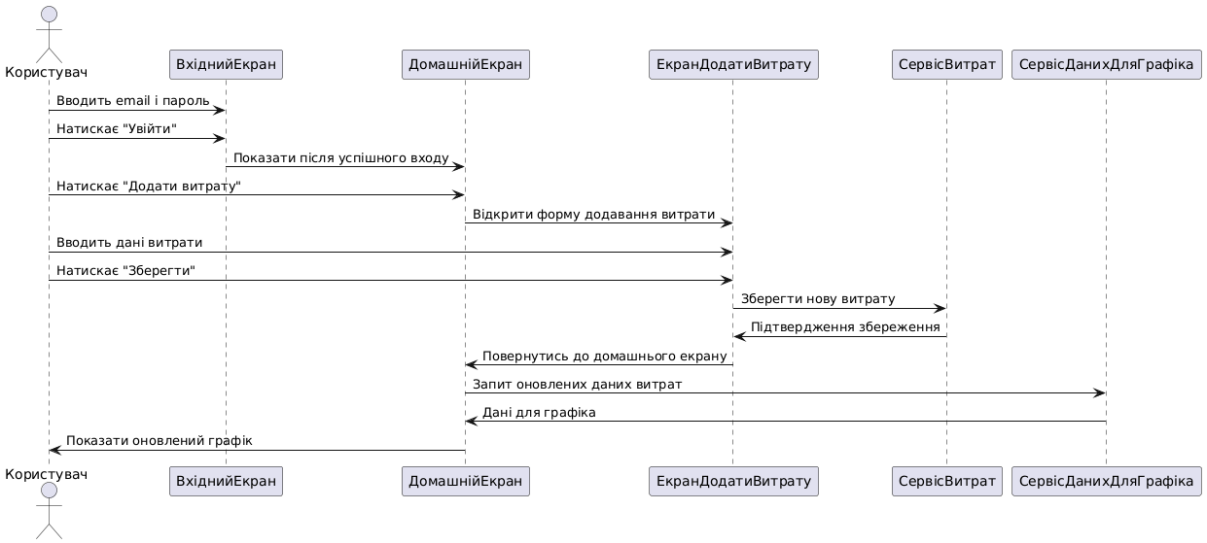


Рис. 2.4.2. Додавання витрати

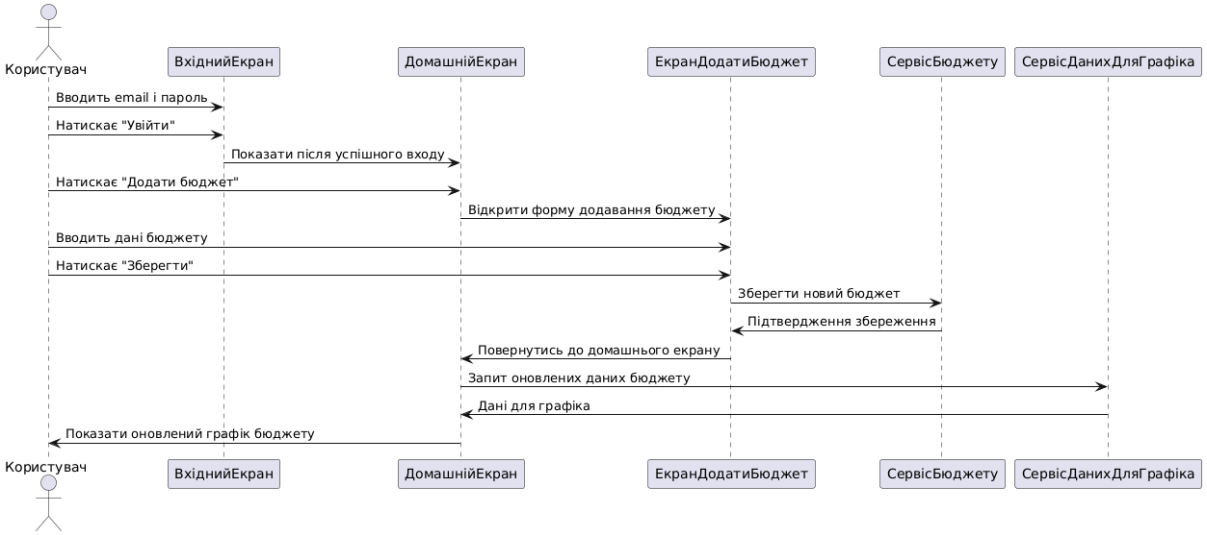


Рис. 2.4.3. Додавання бюджету

2.5 Вибір інструментів розробки

В якості інструментів розробки було вирішено використовувати наступні рішення:

- Flutter - кросплатформений фреймворк. У порівнянні з іншими рішеннями, Flutter забезпечує і високу продуктивність і якісний інтерфейс, і економію часу.
- Material - сучасна дизайн система.
- Dart - сучасна мова програмування

- Firebase - хмарний сервіс.
- Git / GitHub - система контролю версій і хостингу репозиторія.
- Android Studio - інтегроване середовище розробки.
- Figma - онлайн інструмент для дизайну інтерфейсів.
- bloc - бібліотека для кооперації інтерфейса і бізнес-логіки.

Чому поміж різних архітектурних патернів був обран саме BLoC? Він був створений для забезпечення чіткого поділу між презентаційним інтерфейсом користувача та бізнес-логікою застосунку. Його головна мета - зробити додаток масштабованим, передбачуваним і зручним для тестування та підтримки. Усі дії в додатку запускаються як події, які передаються у компонент. Потім він він реагує на ці події, обробляє бізнес-логіку (включаючи інтеграцію з БД) та випускає нові стани, на які інтерфейс може підписатись і відображати.

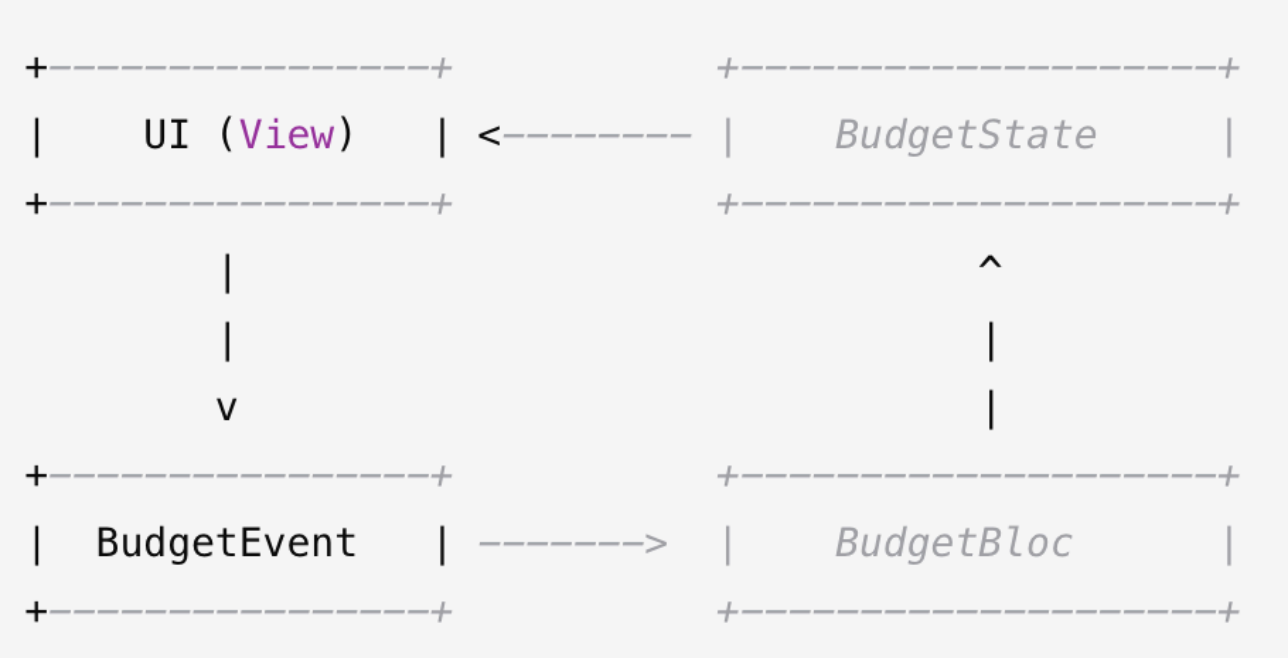


Рис. 2.5.1 Схема роботи інтерфейсу, івентів, стану та бізнес-логіки.

2.6 Висновки до розділу 2

Було опрацьовано і спроектовано архітектуру інформаційної системи, структуру бази даних, логіку роботи компонентів і багатьох елементів інтерфейсу користувача. Обрані технології і підходи дозволяють забезпечити адаптивність, масштабованість та зручність для кінцевого користувача системи. Зазначено схеми баз даних та основних шляхів якими користувач може користуватись у системі.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Розробка мобільної інформаційної системи було здійснена за допомгою фреймворку Flutter і використанням Firebase як back-end платформи. Усі функціональні компоненти було розроблено відповідно до технічних вимог у Розділі 1.

Нижче наведені скріншоти і опис актуального мобільного застосунку.

3.1.1 Реєстрація та авторизація

- Вибір поміж режимами реєстрацію та входа, поля для введення email та пароля для реєстрації користувача або поновного входу, відновлення пароля.

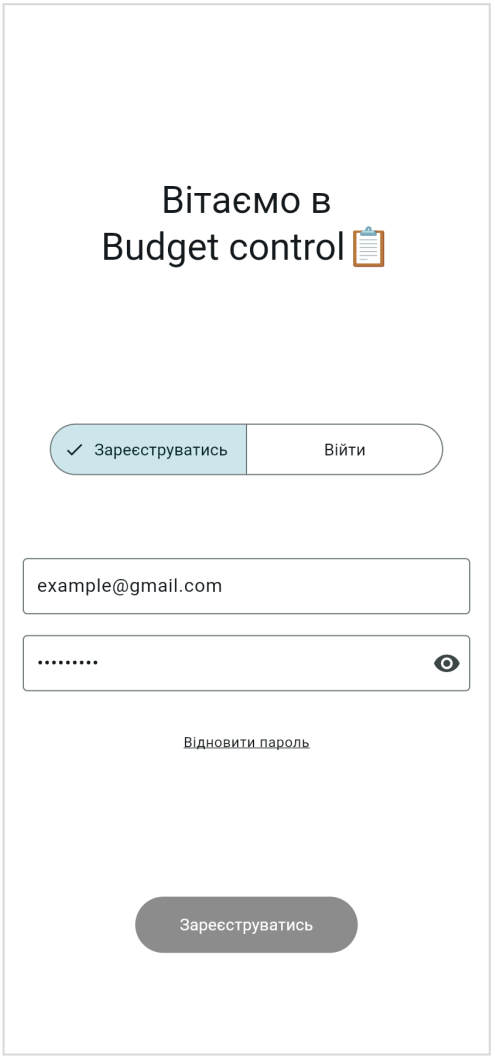


Рис. 3.1.1. Реєстрація та вхід

3.1.2 Оглядання витрат

- Графік витрат по днях
- Лист витрат з повною інформацією щодо залогованої інформації від юзера.
- Call-to-action кнопка (головна кнопка що спонукає користувача натиснути її) - “Додати витрату”

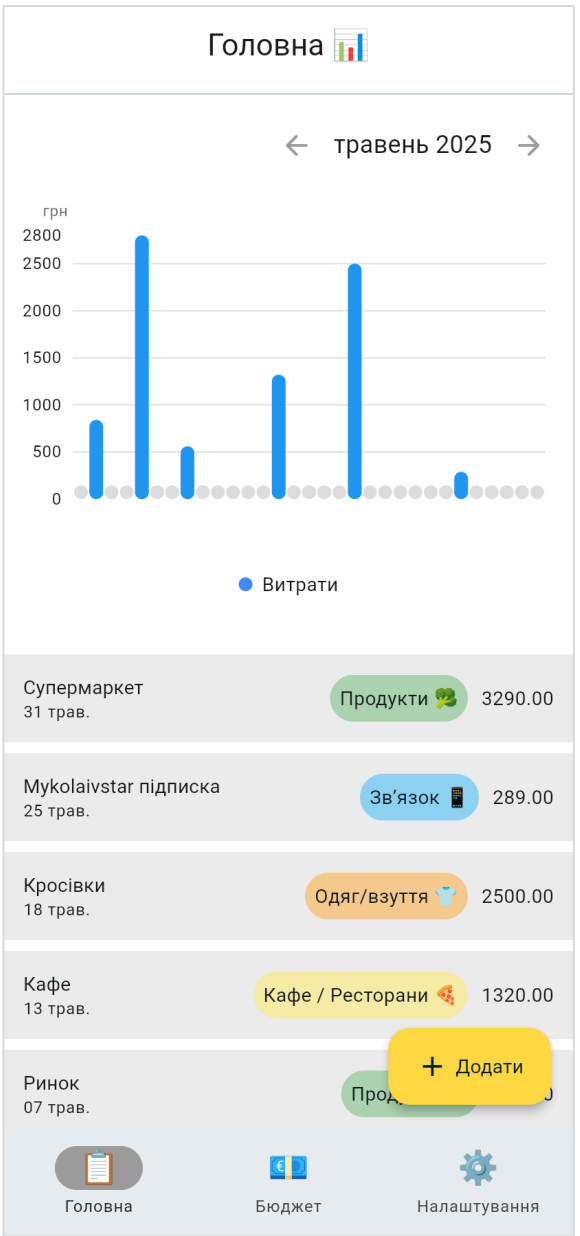


Рис. 3.1.2. Графік та лист витрат

3.1.3 Додавання витрат

Форма для додавання витрати містить поля:

- Коротка назва
- Сума
- Категорія (разом з спеціальним кольором і стікером задля зрозумілого інтерфейсу)

Дані зберігаються у колекції spendings в Firestore.

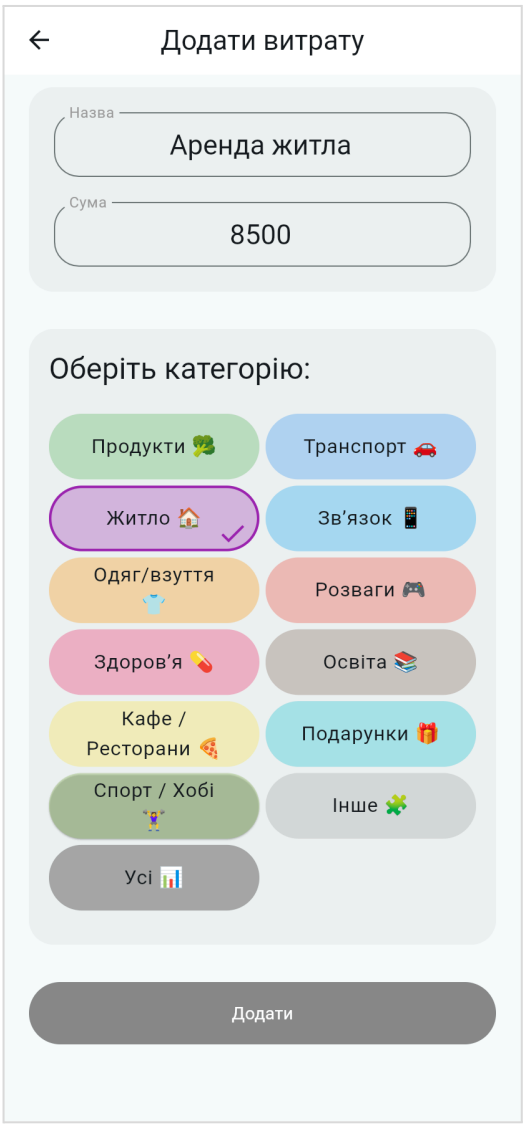


Рис. 3.1.3.1. Додавання витрати

Користувач має можливість видалити витрату, натиснувши на неї.

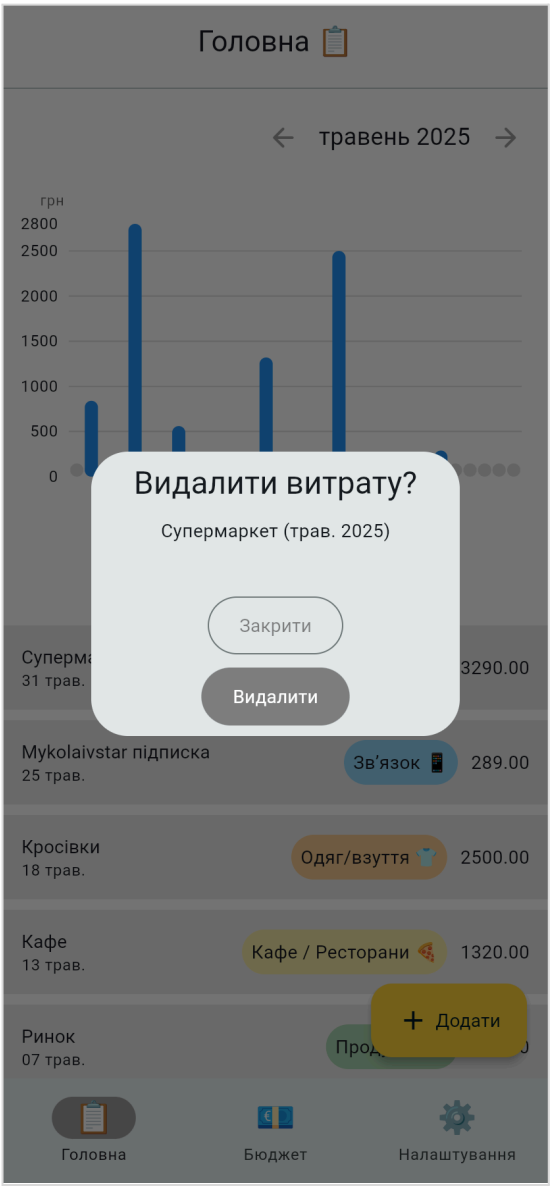


Рисунок 3.1.3.2. Видалення витрати.

3.1.4 Бюджетування

Користувач може вказати бюджет на поточний місяць вказавши:

- назву
- суму
- дати (кінець опціонально),

Інформація зберігається у колекції budgets.

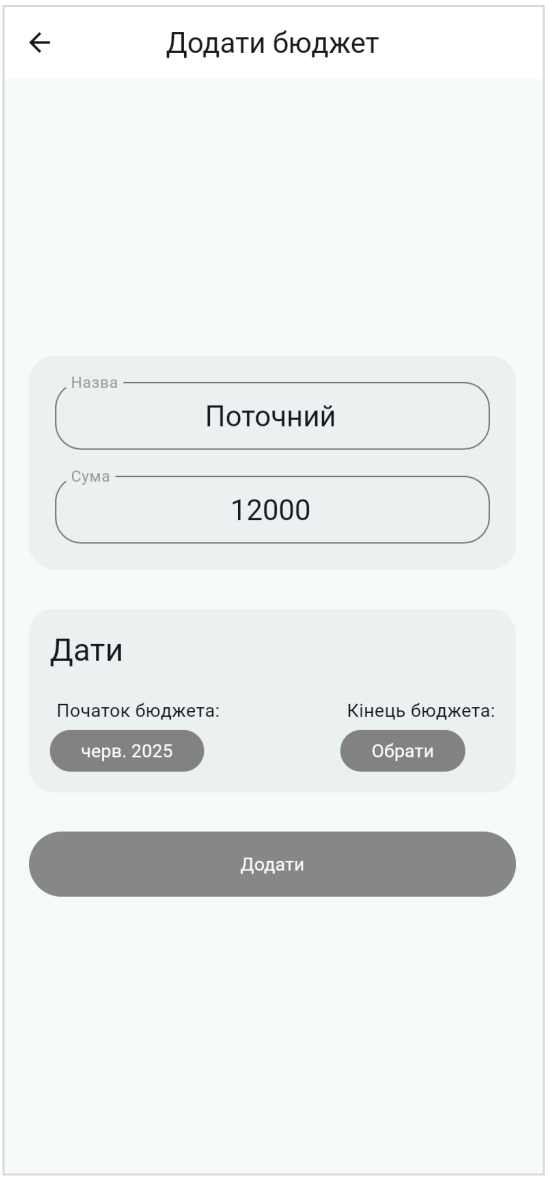


Рис. 3.1.4.1. Додавання бюджету

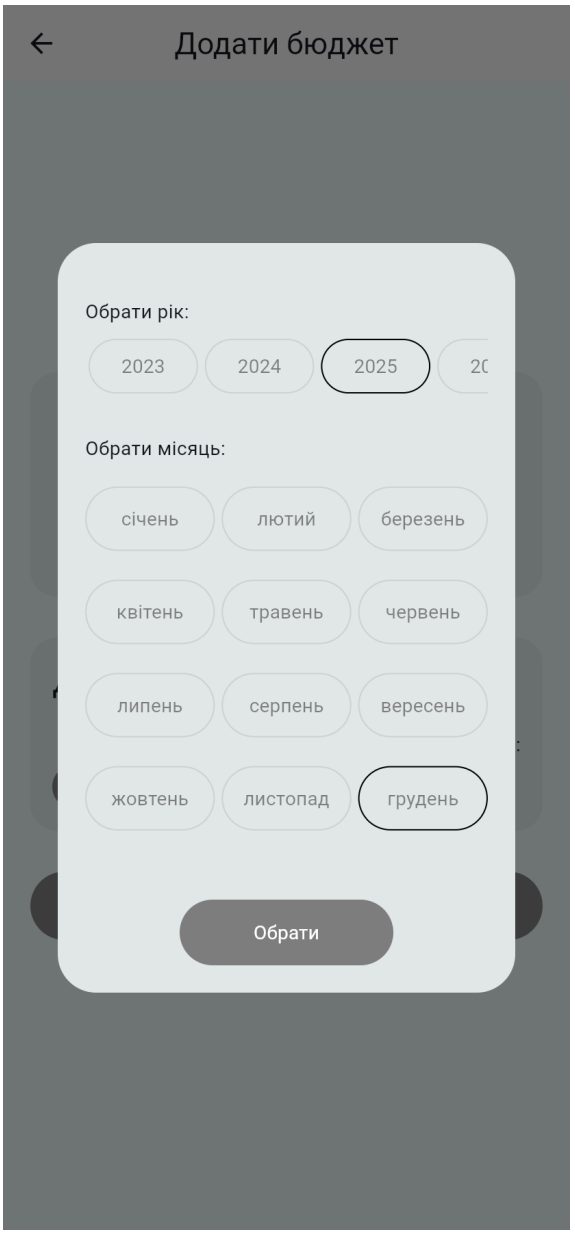


Рис. 3.1.4.2. Вибір дат для бюджету.

3.1.5 Огляд бюджету

Вивід графіка порівнює бюджет із сумарними витратами по днях, де сіра лінія зазначає запланований бюджет (середнє арифметичне), а синя те скільки користувач витратив за цей період. Тобто користувач має можливість наглядно бачити наскільки витримується бюджет.

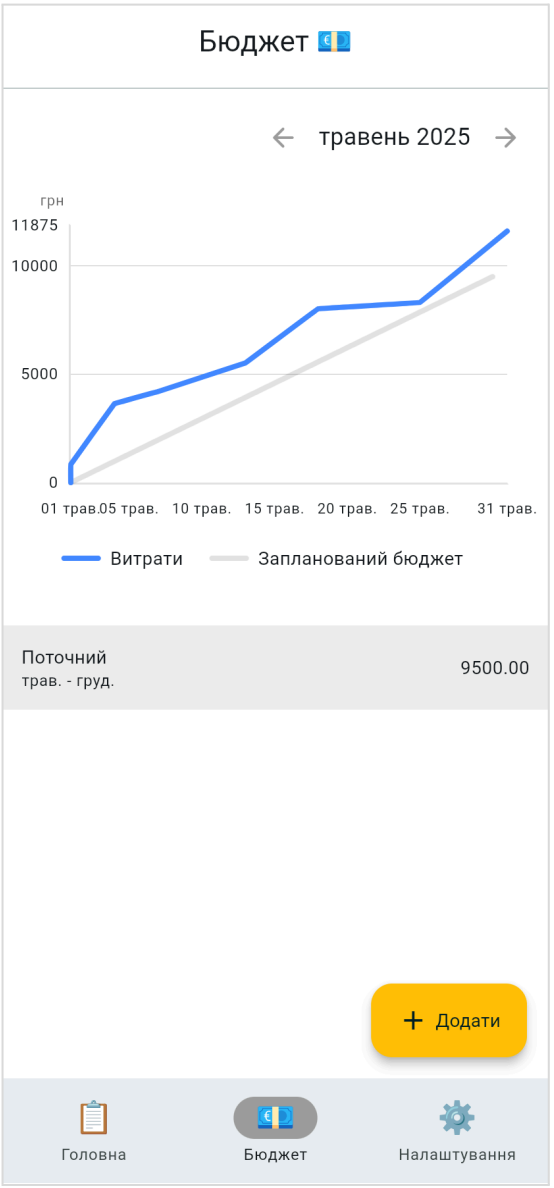


Рис. 3.1.5. Оглядання бюджету на графіку.

3.1.6 Графіки витрат і бюджету

Використано open-source бібліотеку fl_chart для візуалізації витрат і бюджета користувача. Ця бібліотека надає можливість мінімізувати час і ресурси що були б потрібні у разі написання власного рішення малювання графіків.

Рис. 3.1.6.1. Лінійний графік.

Рис. 3.1.6.2. Стовпчиковий графік.

3.1.6 Екран налаштувань

Екран налаштувань має три опції для користувача:

- зкинути пароль
- вийти
- видалити аккаунт

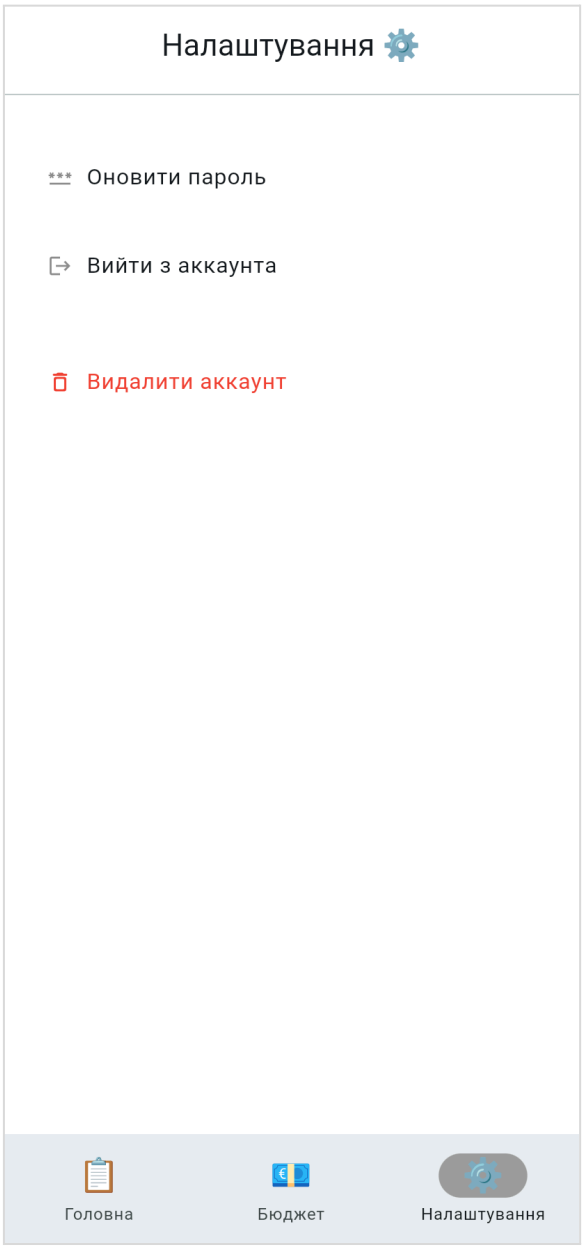


Рис. 3.1.6.1.

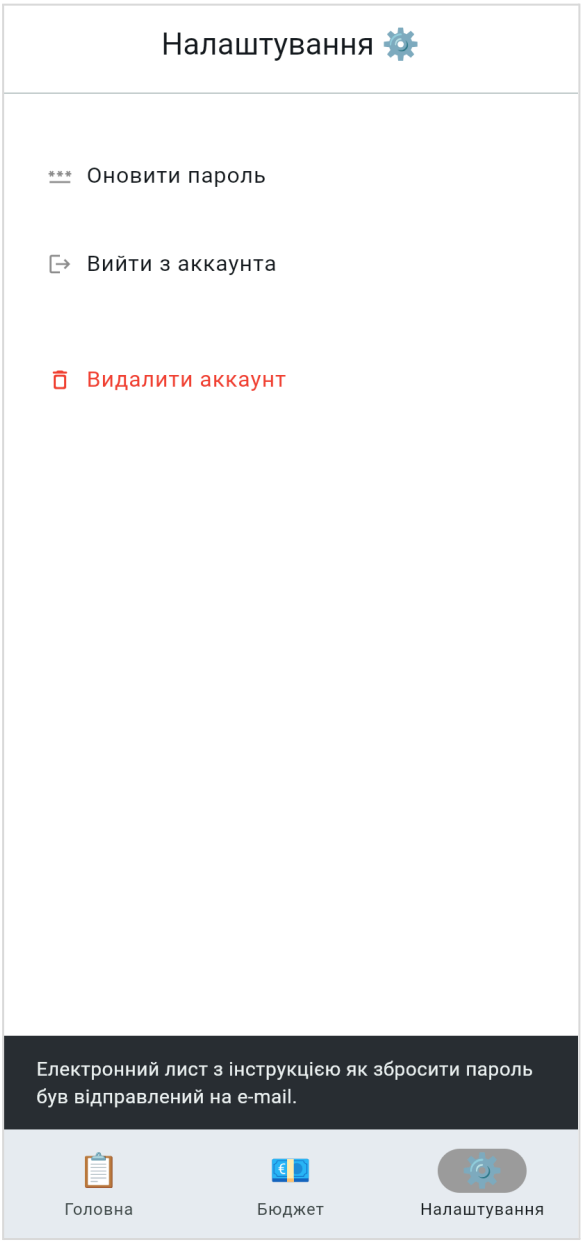


Рис. 3.1.6.2.

При натисканні на кнопку “Оновити пароль”, система авторизації Firebase Auth відправляє лист на електронну пошту користувача, з посиланням на спеціальний сайт авторизації, де згідно з тимчасовим токеном можна змінити пароль. Такий спосіб дозволяє впевнитись що саме власник цього аккаунта має доступ до додатка, і зменшує ризик неавторизованого використання.

При натисканні на кнопку “Вийти за аккаунта”, мобільний додаток видаляє усі локальні дані щодо юзера. Дозволяючи користувачу зайти в інший

додаток, або тимчасово призупинити свою роботу з додатком. Цей метод не видаляє дані юзер з серверу.

При натисканні на кнопку “Видалити аккаунт”, проводиться повне видалення всіх існуючих даних юзера. Така функція є однією з вимог GDPR.

3.2 Специфічні патерни при розробці Flutter додатків.

Як і в інших мовах програмування, Dart надає можливість імплементувати різні підходи, включаючи:

- Інкапсуляцію
- Наслідування
- Поліморфізм
- Абстракцію

Однак є речі, що рідше застосовуються у більшості мовах програмування. Наприклад паттерн “copyWith”. Це поширенний паттерн, який дозволяє створити нову копію об’єкта з певними зміненими полями, не змінюючи оригінальний об’єкт. Таких підхід важливий “незмінному”(immutable) програмуванні, де об’єкти не змінюються напряму, а створюються їхні нові версії з оновленням.

Це важлива частина роботи з менеджером стану додатків. В цьому прикладі система працює за допомогою бібліотеки BLoC (Business Logic Component), що має окрему частину для виконання специфічних завдань, а іншу для зберігання поточного стану додатка. І це один з найголовніших прикладів, чому паттерн “copyWith” має таку важливу роль у розробці мобільного додатка.

```
// Виклик оновлення стану, змінюючи поле spendings
emit(state.copyWith(spendings: spendings));

// Метод copyWith у класі BudgetState
BudgetState copyWith({
  List<Budget>? spendings,
  LoadSpendingsStatus? loadSpendingsStatus,
  AddSpendingStatus? addSpendingStatus,
  RemoveSpendingStatus? removeSpendingStatus,
  String? errorMessage,
}) {
  return BudgetState(
    spendings: spendings ?? this.spendings,
    loadSpendingsStatus: loadSpendingsStatus ?? this.loadSpendingsStatus,
    addSpendingStatus: addSpendingStatus ?? this.addSpendingStatus,
    removeSpendingStatus: removeSpendingStatus ?? this.removeSpendingStatus,
    errorMessage: errorMessage ?? this.errorMessage,
  );
}
```

Рис. 3.2 Приклад використання паттерна copyWith.

3.3 Тестування і результати реалізації

Після реалізації основних функцій проведені основні ручне тестування основних функцій додатка.

- Логіка перевірки введених даних
- Функції роботи з БД (CRUD операції)
- Генерація графіків.
- Взаємодія інтерфейса з бізнес логікою і базою.
- Обробка помилок

В результаті розробки створено функціональний прототип адаптивного веб застосунку з можливостями описаних у попередніх розділах.

3.4 Безпека та захист персональних даних

Розробка інформаційної системи, яка працює з персональними (фінансовими) даними користувачів, вимагає особливої уваги до питань

інформаційної безпеки, конфіденційності та цілісності даних.

Для автентифікації користувачів використовується Firebase Authentication, зокрема метод авторизації через електронну пошту і пароль. Цей механізм:

- забезпечує захист доступу до облікових записів, використовуючи перевірку введених даних та систему керування сесіями [10].
- підтримує шифрування паролів на стороні сервера, що робить неможливим їх перехоплення або несанкціонований доступ
- дозволяє відновлювати доступ через скидання пароля, що підвищує надійність використання.

Всі дані користувачів зберігаються у Firebase Cloud Firestore. Забезпечення їх безпеки здійснюється за допомогою:

- Firebase Security Rules, які обмежують доступ до документів тільки авторизованим користувачам
- ізоляції даних між користувачами - кожен користувач має доступ лише до своїх даних
- захищенного SSL-з’єднання між клієнтським додатком та хмарною базою даних, що запобігає перехоплення даних під час передачі [11].

Ниже додано використані декларативні правила доступу, що перевіряються при кожному зверненні до бази даних. Вони базуються на перевірці унікального ідентифікатора користувача (що є частиною автентифікації), згідно з назвою документа і колекцій.

```
rules_version = '2';

service cloud.firestore {
  match /databases/{database}/documents {

    // Витрати
    match /spendings/{userId}/result/{spendingId} {
      allow read, write: if request.auth != null && request.auth.uid == userId;
    }

    // Бюджети
    match /budget/{userId}/result/{budgetId} {
      allow read, write: if request.auth != null && request.auth.uid == userId;
    }
  }
}

```

Рис. 3.4.1. Firebase Firestore Security Rules.

Нижче також наведено UML схему роботи БД з системою авторизації

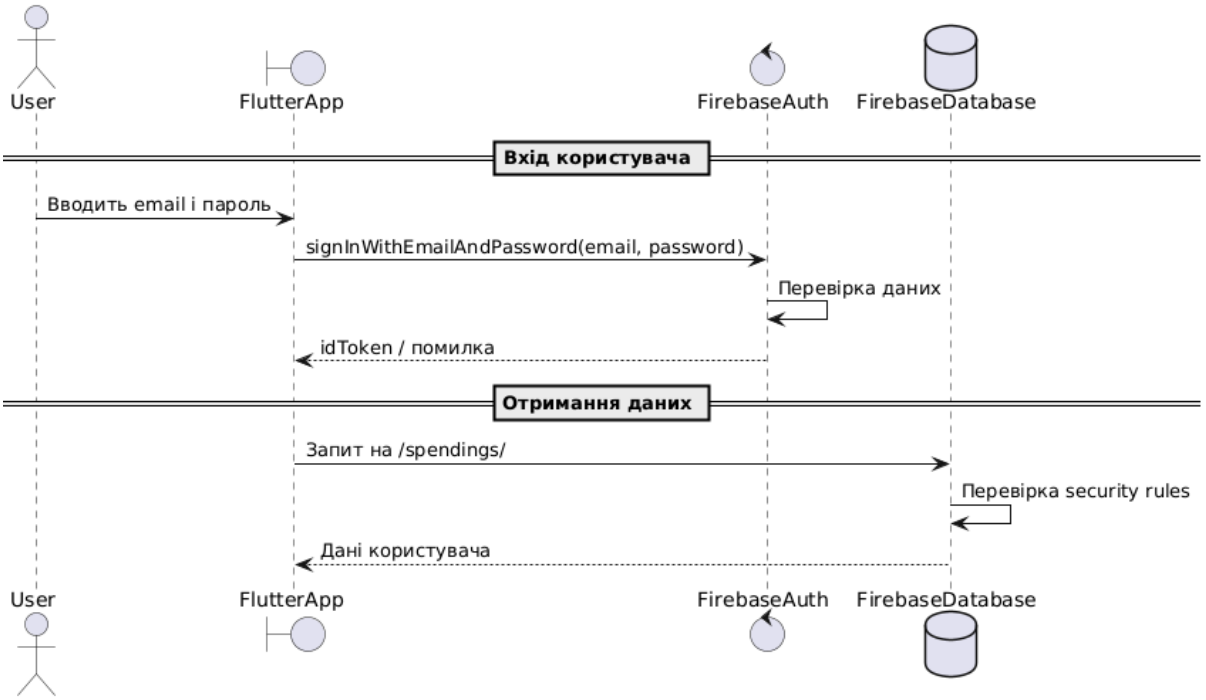


Рис. 3.4.2. UML схема захисту даних користувачів.

Варто зазначити, що невід'ємною частину розробки є перевірка бібліотек з використанням відкритого ісходного коду (Open source), задля забезпечення очікуваємості використання даних.

Також варто зазначити, що велику роль у інформаційних система грає безпеки доступів адміністратор то серверних сервісів. Зазвичай найперший

рівень йде через аккаунт електронної пошти (реєстрація в хмарному сервісі). Тому безпека системи напряму залежить від захисту електронної пошти розробника та адміністратора. Для надійного зберігання облікових даних рекомендується використовувати менеджери паролів (1Password, Authy), де зберігаються email, пароль, резервні коди для двофакторної авторизації та інша службова інформація [12]. Паролі мають бути унікальними й згенерованими автоматично, а не збереженими у браузері чи в месенджерах. Обов’язково треба вмикати двофакторну аутентифікацію для входу в акаунт пошти.

Для роботи, варто використовувати “Принцип найменших привілеїв”. Суть у тому, що кожному користувачу, процесу чи компоненту системи надається тільки той мінімальний рівень доступу, необхідних для виконання його специфічних завдань, щоб мінімізувати ризики помилок або зловживань. Це варто застосовувати як в контексті робочої команди так і окремих сервісів [13].

3.5 Висновки до розділу 3

Система реалізована відповідно до попередніх вимог, а результати тестування підтверджують стабільну роботу основних функцій. Інтерфейс є простий і зручний, а графіки зрозумілі.

Розділ 4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Загальні положення

Забезпечення безпечних умов працює є ключовою складовою професійної діяльності працівника сфери інформаційних технологій. Відповідно до Закону України “Про охорону праці”, кожне робоче місце обов’язково має відповідати санітарно-гігієнічним нормам, вимогам електробезпеки, пожежної безпеки та ергономіки. Особливо це актуально при роботі з різною комп’ютерною технікою, що вимагає дотримання низки нормативних документів, зокрема ДСанПіН 3.3.2.007-98.

4.2 Умови праці на робочому місці

Робоче місце програміста повинно бути організовано відповідно до ергономічних принципів. Висота столу і крісла мають відповідати зросту працівника, а екран монітора розташовуватися на відстані 50-70 см від очера користувача. Освітлення має бути комбінованим: природним та штучним, без прямих відблисків на екрані. Оптимальна температура у приміщення має становити від 24°C до 40°C, а відносна вологість повітря - 40-60%.
Екран монітора повинен розташовуватись на рівні очень або трохи нижче, на відстані від 50 до 70 сантиметрів, з можливістю регулювання нахилу. Стіл та стілець мають бути регульованими по висоті, а крісло з підтримкою поперекового відділу хребта. Освітлення в приміщенні має бути достатнім, не менше 300 лк для загального освітлення, без прямих сонячних променів на екран.

Рис. 4.2 Робоче місце

4.3 Електробезпека

Робота з комп’ютерною технікою вимагає дотримання правил електробезпеки. Всі пристрої, повинні мати надійне заземлення. Використання подовжувачів або трійників може допускатися лише при наявності відповідних сертифікатів. Рекомандується регулярно перевіряти технічних стан обладнання, а також наявність пристроїв захисного відключення (ПЗВ) для запобігання ураження електричним струмом [14].

Працівник повинен проходити інструктаж з електробезпеки та знати порядок дій у разі ураження електричним струмом. У приміщенні бажано мати гумовий килимок в зоні розміщення електрообладнання [15].

Рис 4.3 Електробезпека

4.4 Пожежна безпека

У приміщеннях, до розміщується ІТ-обладнання, має бути мінімум один первинний засіб пожежогасіння - вогнегасник (вуглекислотний або порошковий). Також важливо забезпечити вільний доступ до евакуаційних виходів, і розмістити плани евакуації на видимих місцях. Заборонено використовувати пошкоджене обладнання або перевантажувати електромережу.

Рис. 4.4 Пожежна безпека

4.5 Психофізіологічні аспекти та режим праці

Тривала робота за комп’ютерною технікою може призводити до перевтоми, зниження концентрації уваги та проблем із зором. Саме тому дуже важливо дотримуватися режиму праці - робити 5-10 хвилинні перерви кожні 45-60 хвилин. Також рекомендується виконання вправ для очей і легка розминка для профілактики захворювань опорно-рухового апарату.

4.6 Заходи з охорони праці при впровадженні інформаційної системи

При розгортанні програмного забезпечення є необхідність дотримуватись інструкцій з безпечної експлуатації обладнання. Також необхідно мінімізувати ризики, які пов’язані з роботою на висоті (як приклад - прокладка мережевих кабелів), застосовуючи відповідні засоби індивідуального захисту.

4.7 Екологічні аспекти використання ІТ-технологій

Хоча діяльність у сфері ІТ не є безпосередньо шкідливою для довкілля, вона все ж створює опосередковане екологічне навантаження. Це, зокрема:

- витрати електроенергії;
- створення електронних відходів (старі ПК, периферія);
- використання паперу для документації.

Для зменшення екологічного сліду варто:

- впроваджувати енергоощадні технології (сплячий режим, LED-монітори);
- здавати відпрацьовану техніку на утилізацію через сертифіковані компанії;
- зменшити друк, використовуючи цифрові засоби комунікації.

4.8 Висновки до розділу 4.

Проведений аналіз умов праці в сфері інформаційних технологій показав, що хоча робота розробника програмного забезпечення і не пов’язана з якимись виробничими шкідливими чи небезпечними факторами (у класичному розумінні людини), вона має ряд специфічних ризиків, зокрема:

- тривала робота за комп’ютером спричиняє навантаження на зір, хребет, опорно-руховий апарат
- Психоемоційне перевантаження
- Ризики інформаційної безпеки, які мають як технічні, так і правові аспекти
- У випадку дистанційної роботи - відсутність контролю умов праці.
- Екологічні фактори що треба дотримуватись у роботі програміста.
- Специфічні правила роботи з електротехнікою
- Розуміння базових понять щодо пожежної безпеки.

ВИСНОВКИ

У кваліфікаційній роботі виконано повний цикл розробки інформаційної системи планування бюджету і контролю витрат.

- Проведен аналіз предметної області
- Спроектовано структуру бази даних та інтерфейс
- Враховані усі вимоги проєкту
- Реалізовано клієнтську частина на фрейвморці Flutter із використанням - Firebase
- Протестовано основні функції
- Розписана можливі послідовності дій користувача.

Розписано базові практики забезпечення безпеки та захисту адміністрування хмарними сервісами. Роз'яснені важливі моменти, що мають пряме відношення до реаліїв сьогоднення розробників програмного забезпечення.

Проведено порівняння з альтернативними хмарними платформами, такими як AWS Amplify, Microsoft Azure, що показало оптимальність обраної хмарної платформи Firebase.

Система має практичну цінність та може бути використана як основа для повноцінного мобільного веб застосунку. Робота демонструє ефективність використання хмарних сервісів у розробці програмного забезпечення та підкреслює важливість вибору технологічного стеку на ранніх етапах розробки.

Система була розроблена з урахуванням потреби її розробки як "мінімально життєздатний продукт" - що є типовим вимаганням на ринку програмного забезпечення.

Окремий наголос, отримала концепція риночної системи і відкритого конкурування. Було пояснено, що створення схожою за функціональності системи ніколи не може бути зайвим, через те, що кінцевий користувач має альтернативний вибір. І через деякий час це приносить плоди для суспільства - у вигляді розвитку сфери, адже різні розробники починають краще чути

потреби кінцевих користувачів і намагатись поліпшити свій продукт, для здорової конкуренції.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Competitor analysis. URL: https://medium.com/@olex_world/competitor-analysis-c46505495606 (05.04.2025 дата звернення)

2. Загальний регламент захисту даних (GDPR) [Електроний ресурс]. - Режим доступу: https://commission.europa.eu/law/law-topic/data-protection_en/ - Дата звернення: 05.04.2025.

3. Кросплатформена розробка [Електроний ресурс]. URL: https://www.jetbrains.com/help/kotlin-multiplatform-dev/cross-platform-mobile-development.html#the-most-popular-cross-platform-solutions - Дата звернення: 05.04.2025.

4. What is Flutter [Електронний ресурс]. - Режим доступу: https://aws.amazon.com/what-is/flutter/ - Дата звернення: 05.04.2025

5. Офіційна документація Dart. Dart Language Tour [Електроний ресурс]. Режим доступу: https://dart.dev/overview - Дата звернення: 05.04.2025.

6. Firebase. Що таке Firebase? [Електроний ресурс]. - Режим доступу: https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0 - Дата звернення: 05.04.2025.

7. NO-SQL [Електроний ресурс]. URL: https://www.mongodb.com/resources/basics/databases/nosql-explained - Дата звернення: 01.04.2025

8. Firebase Firestore. [Електроний ресурс]. - Режим доступу: https://firebase.google.com/products/firestore#:~:text=Cloud%20Firestore%20is%20a%20NoSQL,web%20apps%20%2D%20at%20global%20scale. - Дата звернення: 05.04.2025.

9. Google Material Design: принципи та компоненти [Електроний ресурс]. - Режим доступу: https://m3.material.io - Дата звернення: 05.04.2025.

10. Firebase. Захист облікових записів через Firebase Authentication [Електроний ресурс]. - Режим доступу: https://firebase.google.com/docs/auth - Дата звернення: 05.04.2025.

11. Firebase. Правила безпеки Firestore [Електроний ресурс]. - Режим доступу: https://firebase.google.com/docs/firestore/security/get-started - Дата звернення: 05.04.2025.

12. Cyber Security: Importance of a Password Manager. [Електроний ресурс]. - Режим доступу: https://medium.com/datadriveninvestor/cyber-security-importance-of-a-password-manager-127375b989f7 - Дата звернення: 05.04.2025.

13. Принцип найменших привілеїв у розподілених системах доступу [Електроний ресурс]. - Режим доступу: https://medium.com/@jaychapel/why-the-principle-of-least-privilege-is-important-for-saas-based-cloud-management-75a51581670c - Дата звернення: 05.04.2025.

14. Пристрої захисного відключення [Електроний ресурс]. - Режим доступу: https://medium.com/@poweroneups/uninterruptible-power-supply-safe-for-essential-electrical-appliance-94e5fabe0505 - Дата звернення: 05.04.2025.

15. Правила електробезпеки при роботі з комп’ютерною технікою [Електроний ресурс]. - Режим доступу: https://www.dell.com/support/kbdoc/en-us/000137973/safety-precautions-when-working-with-electrical-equipment#:~:text=NOTE%3A%20Disconnect%20all%20power%20sources,inside%20your%20computer%2C%20ground%20yourself. - Дата звернення: 05.04.2025.

ДОДАТКИ

Додаток 1. Частина коду авторизації

```
import 'dart:async';

import 'package:budget/error_util.dart';
import 'package:budget/features/auth/models/auth_statuses.dart';
import 'package:budget/features/auth/repository/auth_repository.dart';
import 'package:equatable/equatable.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

part 'package:budget/features/auth/bloc/auth_state.dart';

class AuthCubit extends Cubit<AuthState> {
 final AuthRepository _authRepository;

 StreamSubscription? onUserChanges;
 AuthCubit({required AuthRepository authRepository})
     : _authRepository = authRepository,
       super(const AuthState(user: null));

 Future<void> initializeAuth() async {
   onUserChanges = _authRepository.getStream.listen((user) {
     getUser(user);
   });
 }

 Future<void> getUser(User? user) async {
   try {
     final fUser = user ?? _authRepository.getUser;
     if (fUser != null) {
       emit(state.copyWith(authStatus: AuthStatus.authenticated));
     } else {
       emit(state.copyWith(authStatus: AuthStatus.authenticated));
     }
   } catch (e) {
     emit(state.copyWith(errorMessage: e.toString()));
   }
 }

 Future<void> signUpWithEmailAndPassword({required String email, required String
password}) async {
   try {
     emit(state.copyWith(signStatus: SignStatus.loading));
     final user = await _authRepository.signUpWithEmail(email: email, password:
password);
     emit(state.copyWith(signStatus: SignStatus.loaded, user: user));
   } catch (e, st) {
     recordError(e, st);
```

```
      emit(state.copyWith(signStatus: SignStatus.error, errorMessage:
e.toString()));
    }
  }

  Future<void> signInWithEmailAndPassword({required String email, required String
password}) async {
    try {
      emit(state.copyWith(signStatus: SignStatus.loading));
      final user = await _authRepository.signInWithEmail(email: email, password:
password);
      emit(state.copyWith(signStatus: SignStatus.loaded, user: user));
    } catch (e, st) {
      recordError(e, st);
      emit(state.copyWith(signStatus: SignStatus.error, errorMessage:
e.toString()));
    }
  }

  Future<void> resetPassword() async {
    try {
      emit(state.copyWith(signStatus: SignStatus.loading));
      await _authRepository.resetPassword();
      emit(state.copyWith(signStatus: SignStatus.sentResetPassword));
    } catch (e, st) {
      recordError(e, st);
      emit(state.copyWith(signStatus: SignStatus.error, errorMessage:
e.toString()));
    }
  }

  Future<void> logout() async {
    try {
      emit(state.copyWith(signStatus: SignStatus.loading));
      await _authRepository.logout();
      emit(state.copyWith(signStatus: SignStatus.loaded, authStatus:
AuthStatus.none));
    } catch (e, st) {
      recordError(e, st);
      emit(state.copyWith(signStatus: SignStatus.error, errorMessage:
e.toString()));
    }
  }

  Future<void> deleteAccount() async {
    try {
      emit(state.copyWith(signStatus: SignStatus.loading));
      await _authRepository.deleteAccount();
      emit(state.copyWith(signStatus: SignStatus.loaded, authStatus:
AuthStatus.none));
    } catch (e, st) {
      if (e is FirebaseAuthException && e.code == 'requires-recent-login') {
        emit(state.copyWith(signStatus: SignStatus.reLoginRequired));
      } else {
        recordError(e, st);
```

```
        emit(state.copyWith(signStatus: SignStatus.error, errorMessage:
e.toString()));
      }
    }
  }

  @override
  Future<void> close() {
    onUserChanges?.cancel();
    return super.close();
  }
}
```

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';

class AuthRepository {
  final FirebaseAuth firebaseAuth;

  AuthRepository({required this.firebaseAuth}) {
    firebaseAuth.setPersistence(Persistence.INDEXED_DB);
    firebaseAuth.setLanguageCode('uk');
  }

  Stream<User?> get getStream => firebaseAuth.userChanges();

  User? get getUser => firebaseAuth.currentUser;

  Future<User> signUpWithEmail({required String email, required String password})
async {
    try {
      final userCredential = await
firebaseAuth.createUserWithEmailAndPassword(email: email, password: password);

      return userCredential.user!;
    } catch (e) {
      rethrow;
    }
  }

  Future<User> signInWithEmail({required String email, required String password})
async {
    try {
      final userCredential = await firebaseAuth.signInWithEmailAndPassword(email:
email, password: password);

      return userCredential.user!;
    } catch (e) {
      rethrow;
    }
  }
```

```
 Future<void> resetPassword() async {
   try {
     await firebaseAuth.sendPasswordResetEmail(email:
firebaseAuth.currentUser!.email!);
   } catch (e) {
     rethrow;
   }
 }

 Future<void> logout() async {
   try {
     await firebaseAuth.signOut();
   } catch (e) {
     rethrow;
   }
 }

 Future<void> deleteAccount() async {
   try {
     await
FirebaseFirestore.instance.collection('spendings').doc(firebaseAuth.currentUser!.u
id).delete();
     await
FirebaseFirestore.instance.collection('budget').doc(firebaseAuth.currentUser!.uid)
.delete();
     await firebaseAuth.currentUser!.delete();
   } catch (e) {
     rethrow;
   }
 }
}
```

Додаток 2. Частина кода презентації графіків

```
import 'package:budget/chart_util.dart';
import 'package:budget/common/widgets/agenda_widget.dart';
import 'package:budget/theme.dart';
import 'package:fl_chart/fl_chart.dart';
import 'package:flutter/material.dart';
import 'package:intl/intl.dart';

class BudgetChart extends StatefulWidget {
 final DateTime startDate;
 final DateTime endDate;
 final List<FlSpot> spots;
 final List<FlSpot> spots2;
 final bool isShowingMainData;

 const BudgetChart({
   required this.isShowingMainData,
   required this.spots,
   required this.spots2,
   required this.startDate,
   required this.endDate,
 });

 @override
 State<BudgetChart> createState() => _BudgetChartState();
}

class _BudgetChartState extends State<BudgetChart> {
 @override
 Widget build(BuildContext context) {
   final maxY = ChartUtil.getMaxY(widget.spots);
   return Padding(
     padding: const EdgeInsets.only(top: 10, right: 20),
     child: Column(
       mainAxisSize: MainAxisSize.min,
       crossAxisAlignment: CrossAxisAlignment.start,
       children: [
         Padding(
           padding: const EdgeInsets.only(left: 28, bottom: 10),
           child: Text('грн', style:
context.textTheme.labelSmall.withColor(context.theme.hintColor)),
         ),
         Expanded(
           child: Padding(
             padding: const EdgeInsets.only(right: 10),
             child: LineChart(
               sampleData1(maxY: maxY),
               duration: const Duration(milliseconds: 250),
             ),
           ),
         ),
         const SizedBox(height: 14),
```

```
          Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              AgendaWidget(color: context.theme.colorScheme.tertiary, title:
'Витрати'),
              const SizedBox(width: 20),
              AgendaWidget(color: context.theme.hintColor.withValues(alpha: 0.1),
title: 'Запланований бюджет'),
            ],
          ),
        ],
      ),
    );
  }

  LineChartData sampleData1({required double maxY}) => LineChartData(
        lineTouchData: lineTouchData1,
        gridData: gridData,
        titlesData: titlesData1,
        borderData: borderData,
        lineBarsData: lineBarsData1,
        minX: widget.startDate.day.toDouble(),
        maxX: widget.endDate.day.toDouble(),
        minY: 0.0,
        maxY: maxY,
      );

  LineTouchData get lineTouchData1 => LineTouchData(
        touchTooltipData: LineTouchTooltipData(
          getTooltipColor: (touchedSpot) =>
context.theme.colorScheme.onTertiaryFixed.withValues(alpha: 0.8),
        ),
      );

  FlTitlesData get titlesData1 => FlTitlesData(
        bottomTitles: AxisTitles(
          sideTitles: bottomTitles,
        ),
        rightTitles: const AxisTitles(),
        topTitles: const AxisTitles(),
        leftTitles: AxisTitles(
          sideTitles: leftTitles(),
        ),
      );

  List<LineChartBarData> get lineBarsData1 => [
        lineChartBarData1_1,
        lineChartBarData1_2,
      ];

  Widget leftTitleWidgets(double value, TitleMeta meta) {
    return SideTitleWidget(
      meta: meta,
      child: Text(
        value.toString(),
        style: context.textTheme.bodySmall,
```

```
        textAlign: TextAlign.center,
      ),
    );
  }

  SideTitles leftTitles() => SideTitles(
        getTitlesWidget: leftTitleWidgets,
        showTitles: true,
        reservedSize: 50,
      );

  Widget bottomTitleWidgets(double value, TitleMeta meta) {
    if (value == 1 || value == widget.endDate.day || (value % 5 == 0 && value <
30)) {
      final date = DateTime(widget.startDate.year, widget.startDate.month,
value.toInt());
      return SideTitleWidget(
        meta: meta,
        space: 10,
        child: Text(
          DateFormat('dd MMM').format(date),
          style: context.textTheme.labelSmall,
          textAlign: TextAlign.center,
        ),
      );
    }
    return const SizedBox();
  }

  SideTitles get bottomTitles => SideTitles(
        showTitles: true,
        reservedSize: 32,
        getTitlesWidget: bottomTitleWidgets,
      );

  FlGridData get gridData => FlGridData(
        drawVerticalLine: false,
        getDrawingHorizontalLine: (v) => FlLine(color: context.theme.disabledColor,
strokeWidth: 0.3),
      );

  FlBorderData get borderData => FlBorderData(
        show: true,
        border: Border(
          bottom: BorderSide(color: context.theme.highlightColor, width: 1.5),
          left: BorderSide(color: context.theme.highlightColor, width: 1.5),
          right: const BorderSide(color: Colors.transparent),
          top: const BorderSide(color: Colors.transparent),
        ),
      );

  LineChartBarData get lineChartBarData1_1 => LineChartBarData(
        isCurved: true,
        curveSmoothness: 0.01,
        color: context.theme.hintColor.withValues(alpha: 0.1),
        barWidth: 4,
```

```
        isStrokeCapRound: true,
        dotData: const FlDotData(show: false),
        belowBarData: BarAreaData(),
        spots: widget.spots,
      );
 LineChartBarData get lineChartBarData1_2 => LineChartBarData(
        isCurved: true,
        curveSmoothness: 0.01,
        color: context.theme.colorScheme.tertiary,
        barWidth: 4,
        isStrokeCapRound: true,
        dotData: const FlDotData(show: false),
        belowBarData: BarAreaData(),
        spots: widget.spots2,
      );
}
```

Додаток 3. Додавання витрати

```
import 'package:budget/features/home/bloc/home_cubit.dart';
import 'package:budget/features/home/models/spending.dart';
import 'package:budget/features/home/models/spending_category.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

class AddSpendingScreen extends StatefulWidget {
 const AddSpendingScreen({super.key});

 @override
 State<AddSpendingScreen> createState() => _AddSpendingScreenState();
}

class _AddSpendingScreenState extends State<AddSpendingScreen> {
 final textEditingController = TextEditingController();
 final sumController = TextEditingController();

 SpendingCategory? spendingCategory;
 @override
 void dispose() {
   textEditingController.dispose();
   sumController.dispose();
   super.dispose();
 }

 @override
 Widget build(BuildContext context) {
   final addStatus = context.select((HomeCubit cubit) =>
cubit.state.addSpendingStatus);
   return Scaffold(
     appBar: AppBar(title: const Text('Додати витрату')),
     body: Padding(
       padding: const EdgeInsets.symmetric(vertical: 10, horizontal: 20),
       child: ListView(
```

```
          children: [
            BlocListener<HomeCubit, HomeState>(
              listener: listener,
              child: Container(
                decoration: BoxDecoration(
                  borderRadius: BorderRadius.circular(20),
                  color: Colors.grey.withValues(alpha: 0.1),
                ),
                padding: const EdgeInsets.symmetric(horizontal: 20, vertical: 20),
                child: Column(
                  children: [
                    TextFormField(
                      controller: textEditingController,
                      autofocus: true,
                      keyboardType: TextInputType.text,
                      style: Theme.of(context).textTheme.titleLarge,
                      textAlign: TextAlign.center,
                      decoration: InputDecoration(
                        label: const Text('Назва'),
                        alignLabelWithHint: true,
                        labelStyle: const TextStyle(color: Colors.grey),
                        border: OutlineInputBorder(borderRadius:
BorderRadius.circular(20)),
                        hintText: 'Назва',
                        hintStyle: const TextStyle(color: Colors.grey),
                      ),
                    ),
                    const SizedBox(height: 20),
                    TextFormField(
                      controller: sumController,
                      keyboardType: TextInputType.number,
                      style: Theme.of(context).textTheme.titleLarge,
                      textAlign: TextAlign.center,
                      decoration: InputDecoration(
                        label: const Text('Сума'),
                        labelStyle: const TextStyle(color: Colors.grey),
                        border: OutlineInputBorder(borderRadius:
BorderRadius.circular(20)),
                        hintText: 'Сума',
                        hintStyle: const TextStyle(color: Colors.grey),
                      ),
                    ),
                  ],
                ),
              ),
            ),
            const SizedBox(height: 30),
            Container(
              width: double.infinity,
              decoration: BoxDecoration(
                borderRadius: BorderRadius.circular(20),
                color: Colors.grey.withValues(alpha: 0.1),
              ),
              padding: const EdgeInsets.all(16),
              child: Column(
                mainAxisSize: MainAxisSize.min,
```

```
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Text('Оберіть категорію:', style:
Theme.of(context).textTheme.headlineSmall),
                  const SizedBox(height: 20),
                  SizedBox(
                    width: MediaQuery.sizeOf(context).width,
                    height: 410,
                    child: GridView(
                      gridDelegate: const
SliverGridDelegateWithFixedCrossAxisCount(
                        mainAxisSpacing: 5,
                        crossAxisSpacing: 5,
                        crossAxisCount: 2,
                        childAspectRatio: 3.2,
                      ),
                      children: [
                        ...SpendingCategory.values.map((e) {
                          return SizedBox(
                            height: 60,
                            child: FilledButton(
                              onPressed: () => setState(() => spendingCategory =
e),
                              style: ButtonStyle(
                                side: WidgetStateProperty.all(
                                  BorderSide(color: spendingCategory == e ? e.color
: Colors.transparent, width: 2),
                                ),
                                backgroundColor: WidgetStatePropertyAll(
                                  e.color.withValues(alpha: 0.3),
                                ),
                              ),
                              child: Stack(
                                clipBehavior: Clip.none,
                                alignment: Alignment.center,
                                children: [
                                  Center(
                                    child: Text(
                                      '${e.name} ${e.emoji}',
                                      style: Theme.of(context).textTheme.bodyLarge,
                                      textAlign: TextAlign.center,
                                    ),
                                  ),
                                  if (spendingCategory == e)
                                    Positioned(
                                      right: -15,
                                      bottom: 2,
                                      child: Icon(
                                        Icons.check,
                                        color: e.color,
                                        size: 25,
                                      ),
                                    ),
                                ],
                              ),
                            ),
```

```
                        );
                      }),
                    ],
                  ),
                ),
              ],
            ),
          ),
          const SizedBox(height: 30),
          SizedBox(
            width: double.infinity,
            child: FilledButton(
              onPressed: _onAddSpending,
              child: Padding(
                padding: const EdgeInsets.symmetric(vertical: 15),
                child: addStatus.isLoading ? const CircularProgressIndicator() :
const Text('Додати'),
              ),
            ),
          ),
        ],
      ),
    ),
  );
 }

 void _onAddSpending() {
   context.read<HomeCubit>().addSpending(
         spending: Spending(
           id: '',
           date: DateTime.now(),
           title: textEditingController.text,
           spendingCategory: spendingCategory!,
           sum: num.parse(sumController.text).toDouble(),
         ),
       );
 }

 void listener(BuildContext context, HomeState state) {
   if (state.addSpendingStatus.isLoaded) {
     Navigator.pop(context);
   }
 }
}
```

Додаток 4. Частина коду роботи з тимчасовим станом додатка

```
import 'dart:async';

import 'package:budget/error_util.dart';
import 'package:budget/features/budget/models/budget.dart';
import 'package:budget/features/budget/models/budget_statuses.dart';
import 'package:budget/features/budget/repository/budget_repository.dart';
import 'package:equatable/equatable.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

part 'package:budget/features/budget/bloc/budget_state.dart';

class BudgetCubit extends Cubit<BudgetState> {
 final BudgetRepository _budgetRepository;

 StreamSubscription? spendingsSubscription;

 BudgetCubit({required BudgetRepository budgetRepository})
     : _budgetRepository = budgetRepository,
       super(const BudgetState());

 Future<void> initialize() async {
   try {
     emit(state.copyWith(loadSpendingsStatus: LoadSpendingsStatus.loading));
     await spendingsSubscription?.cancel();
     _budgetRepository.init();
     spendingsSubscription = _budgetRepository.initListeners().listen((spendings)
{
       spendings.sort((a, b) => b.startDate.compareTo(a.startDate));
       emit(state.copyWith(spendings: spendings, loadSpendingsStatus:
LoadSpendingsStatus.loaded));
     });
   } catch (e, st) {
     recordError(e, st);
   }
 }

 Future<void> addBudget({required Budget budget}) async {
   try {
     emit(state.copyWith(addSpendingStatus: AddSpendingStatus.loading));
```

```
      await _budgetRepository.addBudget(spending: budget);
      emit(state.copyWith(addSpendingStatus: AddSpendingStatus.loaded));
    } catch (e, st) {
      emit(state.copyWith(addSpendingStatus: AddSpendingStatus.error));
      recordError(e, st);
    }
  }

  Future<void> removeBudget({required Budget budget}) async {
    try {
      emit(state.copyWith(addSpendingStatus: AddSpendingStatus.loading));

      await _budgetRepository.removeBudget(id: budget.id);
      emit(state.copyWith(addSpendingStatus: AddSpendingStatus.loaded));
    } catch (e, st) {
      emit(state.copyWith(addSpendingStatus: AddSpendingStatus.error));
      recordError(e, st);
    }
  }

  @override
  Future<void> close() {
    spendingsSubscription?.cancel();
    return super.close();
  }
}
```