Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка кросплатформного мобільного інтерфейсу для управління особистими фінансами»

Виконав: студент 4 курсу
групи 2ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Закерничний І.О.
(прізвище та ініціали)
Керівник:д.т.н., проф. каф. ПЗ Піщинська Л.Б.
(прізвище та ініціали)
Рецензент: Черняк О.І.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О.Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
« 21 » березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Закерничному Ігорю Олександровичу

1. Тема роботи – «Розробка кросплатформного мобільного інтерфейсу для управління особистими фінансами»

Керівник роботи: Ліщинська Людмила Броніславівна, д.т.н., проф. кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи 2 червня 2025.

3. Вихідні дані до роботи: тип програми – мобільний застосунок;; засоби організації даних програми – фреймворки React Native, MongoDB, Node.js; вхідні дані: дані користувача, дані транзакцій, план бюджету; вихідні дані: графіки, індекс фінансового здоров'я, записи про транзакції; середовище розробки – Visual Studio Code; мова програмування – JavaScript, програмне забезпечення для симуляції мобільних пристроїв – Expo Go.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз стану розвитку застосунків для ведення особистих фінансів; розробка моделі та алгоритмів застосунку; розробка застосунку; тестування застосунку; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд; актуальність теми; мета,

об'єкт та предмет дослідження; задачі дослідження, наукова новизна, практична цінність одержаних результатів; порівняльний аналіз аналогів; використані технології; удосконалення методу «фінансового здоров'я» користувача; розробка моделі застосунку; тестування застосунку; висновки; апробація результатів роботи і публікації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Ліщинська Л.Б., д.т.н., проф. кафедри ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану розвитку застосунків для ведення особистих фінансів	25.03.2025 - 06.04.2025	вик
2	Розробка моделі та алгоритмів застосунку	07.04.2025 - 20.04.2025	вик
3	Варіантний аналіз та вибір мови програмування	21.04.2025 - 27.04.2025	вик
4	Розробка застосунку	28.04.2025 - 18.05.2025	вик
5	Тестування застосунку	19.05.2025 - 25.05.2025	вик
6	Оформлення матеріалів до захисту БКР	26.05.2025- 30.05.2025	вик

Студент Закерничний І.О.
(підпис) (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи Ліщинська Л.Б.
(підпис) (прізвище та

ініціали)

АНОТАЦІЯ

УДК 004:005.9

Закерничний І.О. Розробка кросплатформного мобільного інтерфейсу для управління особистими фінансами : бакалаврська кваліфікаційна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця : ВНТУ, 2025. 101 с.

На укр. мові. Бібліогр. : 28 назв.; рис.: 35; табл.: 10.

У бакалаврській кваліфікаційній роботі було визначено доцільність та мету розробки, яка полягає у покращенні процесу ведення особистих фінансів шляхом розробки спеціалізованого програмного забезпечення, що дозволяє керувати інформацією про доходи та витрати, формувати бюджет та відстежувати його виконання. Для досягнення поставленої мети було сформульовано задачі дослідження, проведено аналіз стану питання розробки застосунків для ведення особистих фінансів, визначено сучасні підходи та методи до реалізації подібних систем.

У ході роботи було розроблено модель застосунку; розроблено алгоритми управління фінансами та формування бюджету.

Програмне забезпечення розроблено із використанням середовища розробки Visual Studio Code та мови програмування JavaScript. У процесі розробки використаний фреймворк React Native. Під час тестування було перевірено працездатність основного функціоналу системи та підтверджено коректність реалізованих методів.

У результаті виконання роботи розроблено мобільний кросплатформний застосунок, що дозволяє керувати особистими фінансами, записувати витрати та доходи, планувати бюджет.

Ключові слова: мобільний застосунок, фінанси, JavaScript, React Native, бюджет.

ANNOTATION

UDC 004:005.9

Zakernychny I.O. Development of a cross-platform mobile interface for personal finance management: bachelor's qualification work in the specialty 121 Software Engineering, educational program - Software Engineering. Vinnytsia: VNTU, 2025. 101p.

In Ukrainian. Bibliography: 28 titles; fig.: 35; tab.: 10.

The bachelor's qualification work determined the feasibility and purpose of the development, which is to improve the process of managing personal finances by developing specialized software that allows you to manage information about income and expenses, form a budget and track its implementation. To achieve the goal, the research tasks were formulated, an analysis of the state of the issue of developing applications for managing personal finances was conducted, modern approaches and methods for implementing such systems were identified.

In the course of the work, an application model was developed; developed algorithms for financial management and budgeting.

The software was developed using the Visual Studio Code development environment and the JavaScript programming language. The React Native framework was used in the development process. During testing, the functionality of the main functionality of the system was checked and the correctness of the implemented methods was confirmed.

As a result of the work, a mobile cross-platform application was developed that allows you to manage personal finances, record expenses and income, and plan a budget.

Keywords: mobile application, finance, JavaScript, React Native, budget.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Розвиток цифрових технологій і широке проникнення мобільних пристроїв у всі сфери життя суттєво вплинули на способи управління особистими фінансами. У сучасному суспільстві фінансова грамотність і вміння ефективно керувати власним бюджетом є важливими умовами для досягнення економічної стабільності та особистого успіху. Популярність мобільних застосунків для фінансового обліку постійно зростає, адже користувачі прагнуть отримати доступний, простий та зрозумілий інструмент для контролю доходів і витрат, фінансового планування, заощадження та інвестування коштів [1, 2].

Незважаючи на значний прогрес у цій галузі, критичний аналіз існуючих мобільних застосунків дозволяє виявити низку суттєвих недоліків і прогалин. Більшість доступних на ринку рішень забезпечують лише базові функції, такі як простий облік доходів і витрат, формування бюджетів та елементарну візуалізацію фінансових даних. Проте далеко не всі застосунки пропонують поглиблений аналіз фінансових звичок, динамічні прогнози майбутніх витрат і доходів або ж адаптивні персоналізовані рекомендації з урахуванням змін у фінансовій поведінці користувача [3, 4]. Через це багато користувачів стикаються з проблемою, коли застосунок не відповідає їхнім реальним потребам, і змушені використовувати кілька додатків одночасно, що знижує загальну ефективність та зручність.

Іншою серйозною проблемою є недостатня інтеграція мобільних застосунків із банківськими системами. Хоча провідні банки поступово відкривають API для доступу до фінансових даних клієнтів, цей процес і досі є неповноцінним і непослідовним, особливо у регіональних банках і країнах із менш розвиненою цифровою інфраструктурою. Це призводить до необхідності ручного введення транзакцій, що підвищує ймовірність помилок, знижує зручність користування і збільшує витрати часу користувачів [5, 6].

Також варто зазначити проблему забезпечення безпеки і конфіденційності фінансової інформації. Попри поширення сучасних методів захисту, таких як

багатофакторна аутентифікація, шифрування та регулярні аудити безпеки, не всі розробники належно дотримуються цих стандартів. Недостатній рівень захисту створює ризики компрометації фінансових даних користувачів, що може мати серйозні наслідки як для індивідуальних користувачів, так і для репутації самих застосунків [7, 8].

Отже, актуальною є розробка застосунку для ведення особистих фінансів.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є покращення процесу ведення особистих фінансів шляхом розробки спеціалізованого програмного забезпечення, що дозволяє керувати інформацією про доходи та витрати, формувати бюджет та відстежувати його виконання.

Основними задачами дослідження є:

- провести аналіз стану питання розробки застосунків для ведення особистих фінансів;
- дослідити існуючі аналоги та провести їх порівняльний аналіз;
- розробити модель застосунку для ведення особистих фінансів;
- розробити метод «фінансового здоров’я» користувача;
- розробити алгоритми роботи застосунку;
- розробити модулі застосунку для ведення особистих фінансів;
- провести тестування розробленого застосунку.

Об'єкт дослідження – процеси управління особистими фінансами.

Предмет дослідження – методи та засоби управління особистими фінансами.

Методи дослідження. У процесі досліджень використовувались такі методи дослідження: аналіз аналогів та порівняння їх з власною розробкою для визначення та постановки задач розробки; теорія користувацького досвіду та принципів UX/UI дизайну; теорія кросплатформної розробки для створення та оптимізації мобільних додатків; елементи математичної статистики для побудови статистичних графіків у застосунку; комп’ютерне моделювання для аналізу та перевірки отриманих

теоретичних положень.

Наукова новизна отриманих результатів.

1. Подальшого розвитку отримав метод «фінансового здоров’я» користувача, який враховує обсяг резервного фонду, рівень диверсифікації джерел доходу та ступінь досягнення поставлених фінансових цілей, що дозволяє отримати більш об'єктивну та комплексну оцінку фінансового стану через єдиний інтегральний індекс за шкалою від 0 до 100, що спрощує розуміння загальної фінансової ситуації користувача та допомагає виявляти потенційні ризики на ранніх стадіях.

2. Подальшого розвитку набула модель системи, яка, на відміну від існуючих, реалізована за трирівневою модульною архітектурою з інтегрованими алгоритмами фінансової аналітики та диференційованим підходом до безпеки даних, що дозволяє забезпечити автоматичне генерування персоналізованих рекомендацій з повним контролем приватності даних при збереженні можливостей хмарного масштабування.

Практична цінність отриманих результатів. Практична цінність розробки полягає у можливості використання застосунку для обліку особистих доходів, витрат, формування бюджету.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто. У наукові роботі, опублікованій у співавторстві, автору належать такі результати: особливості мобільних застосунків для ведення особистих фінансів, перспективи розвитку мобільного застосунку для ведення особистих фінансів.

Апробація матеріалів. Описані у бакалаврській кваліфікаційній роботі положення доповідалися на міжнародній науково-практичній конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)».

Публікації. За тематикою дослідження опубліковано 1 наукову працю у збірнику матеріалів конференції [9].

1 АНАЛІЗ СТАНУ РОЗВИТКУ ЗАСТОСУНКІВ ДЛЯ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ

1.1 Особливості предметної галузі застосунків для управління особистими фінансами

Розробка мобільних застосунків для ведення особистих фінансів є актуальним напрямом у сфері інформаційних технологій, що зумовлено зростаючою потребою користувачів у ефективному управлінні власними фінансами [10]. На сьогоднішній день існує велика кількість мобільних додатків, які пропонують базові функції, такі як облік доходів та витрат, бюджетування, автоматичну категоризацію транзакцій, візуалізацію статистики витрат тощо [11]. Попри широкий вибір доступних рішень, більшість із них мають певні обмеження щодо функціональності, персоналізації та прогнозування фінансової поведінки користувачів, що свідчить про значні перспективи вдосконалення наявних продуктів і розвитку нових, більш інноваційних підходів.

Одним із ключових аспектів розвитку таких застосунків є інтеграція з банківськими системами через відкриті API, проте процес інтеграції є непростим завданням, адже розробники стикаються з такими викликами, як забезпечення високого рівня безпеки транзакцій, стабільності взаємодії з різними банківськими API, а також відповідність вимогам законодавчих та регуляторних норм, особливо щодо захисту персональних і фінансових даних [12].

Значний потенціал для розвитку мають алгоритми аналізу фінансових даних, що базуються на штучному інтелекті та машинному навчанні [13]. Вже сьогодні деякі застосунки пропонують користувачам поглиблений аналіз фінансових звичок, прогнозування майбутніх витрат і доходів, а також персоналізовані поради з оптимізації бюджету [14]. Втім, ефективність таких рішень суттєво залежить від якості та кількості доступних даних, правильності алгоритмів та рівня довіри користувачів до рекомендацій системи, тому значна увага приділяється не лише технічному вдосконаленню алгоритмів, а й забезпеченню прозорості та зрозумілості результатів аналізу для кінцевого користувача [15].

Отже, актуальною є розробка нового мобільного застосунку для ведення особистих фінансів.

1.2 Порівняльний аналіз аналогів

На ринку застосунку для ведення особистих фінансів є чимало готових рішень. Розглянемо такі застосунку, як: Mint, You Need a Budget, Personal Capital.

Mint [16] є одним із найпопулярніших персональних фінансових застосунків на ринку, що пропонує комплексне рішення для відстеження та оптимізації особистих фінансів, який пропонує функціонал ведення бюджету, проте виходить за межі простого встановлення лімітів витрат, надсилає користувачам сповіщення про незвичні витрати, наближення до бюджетних лімітів та надає персоналізовані поради щодо заощаджень.

Інтерфейс Mint наведено на рисунку 1.1.

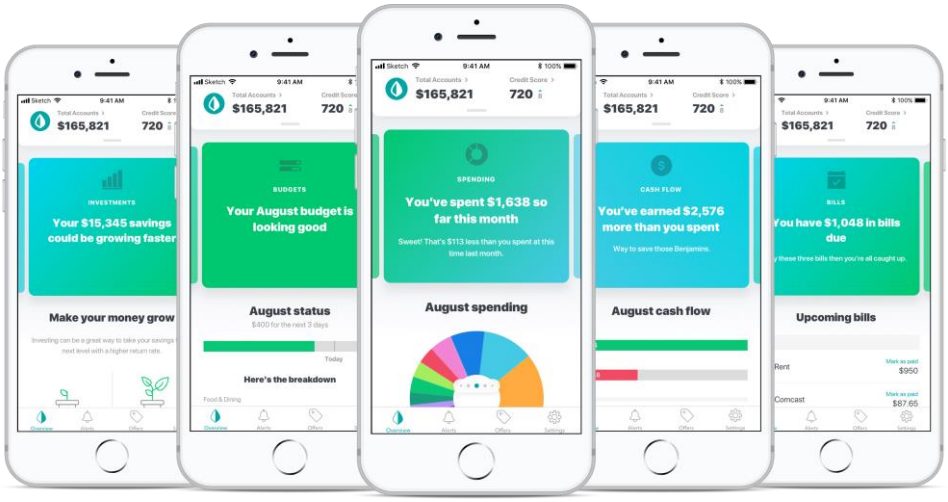


Рисунок 1.1 — Інтерфейс Mint

You Need A Budget [17] фокусується на комплексній оцінці фінансового стану, проактивному бюджетуванні та формуванні здорових фінансових звичок. Застосунок допомагає користувачам розподіляти кошти на різні категорії витрат ще до того, як вони будуть витрачені. You Need A Budget надає детальні звіти про витрати та заощадження.

Інтерфейс You Need A Budget наведено на рисунку 1.2.

Рисунок 1.2 — Інтерфейс You Need A Budget

Personal Capital [18] є фінансовою платформою, що поєднує функції відстеження щоденних витрат із інструментами для управління інвестиціями та пенсійного планування, пропонує аналітичні інструменти для оцінки продуктивності інвестицій, аналізу комісій інвестиційних фондів та планування виходу на пенсію. Застосунок також має функції для щоденного ведення бюджету, але розглядає це як частину ширшої фінансової стратегії.

Інтерфейс Personal Capital наведено на рисунку 1.3.

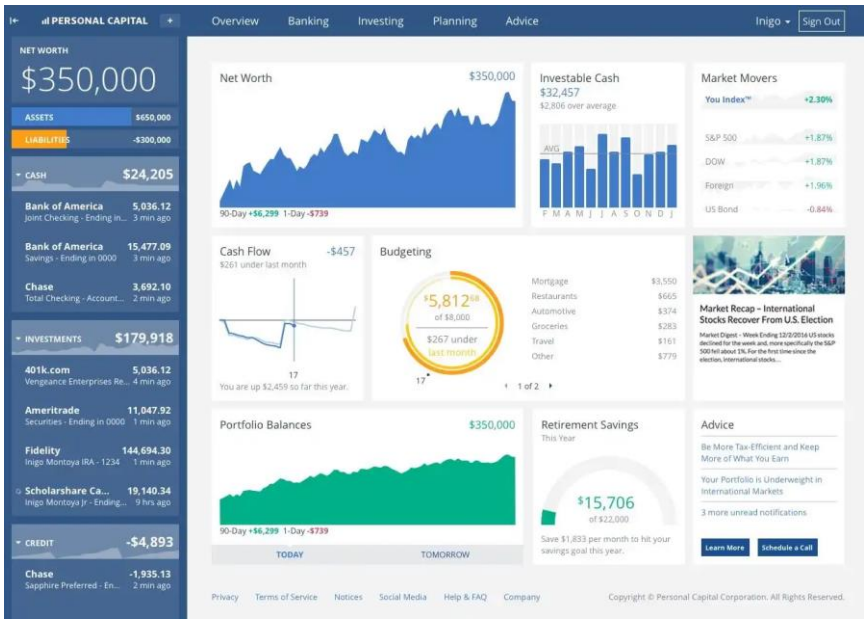


Рисунок 1.3 — Інтерфейс Personal Capital

Для порівняння характеристик цих застосунків із власною розробкою, було

сформовано таблицю 1.1.

Таблиця 1.1 — Порівняльний аналіз аналогів

Характеристика	Personal Capital	You Need A Budget (YNAB)	Mint	Власна розробка
Відстеження витрат	+	+	+	+
Автоматична категоризація транзакцій	+	+	+	+
Встановлення бюджету	+	-	+	+
Індекс фінансового стану	-	-	-	+
Персональні рекомендації	+	-	-	+
Встановлення цілей	+	+	+	+
Сумарний бал	5	3	4	6

Таким чином, власна розробка є актуальною, оскільки її сумарний бал вищий за аналоги, що означає те, що власна розробка має переваги над існуючими рішеннями, а саме: містить персоналізовані рекомендації, та розраховує індекс фінансового стану користувача, чого немає у більшості аналогів.

1.3 Аналіз методів розв’язання задачі

Розробка мобільного застосунку для ведення особистих фінансів може здійснюватися за допомогою кількох принципово різних методологічних підходів, кожен з яких має свої переваги та обмеження.

Метод нативної розробки передбачає створення окремих версій застосунку для кожної платформи, забезпечуючи цим максимальну продуктивність, безперешкодний доступ до всіх апаратних функцій пристрою та оптимальну інтеграцію з кожною з відповідних операційних систем [19]. Застосунки, що розробляються з використанням цього методу, мають вищий рівень безпеки та швидкодії, однак це потребує більших затрат на розробку.

У методі кросплатформеної розробки використовуються фреймворки, такі як React Native, Flutter або Xamarin, що дозволяють створити єдиний код для бізнес-логіки для усіх платформ, завдяки чому суттєво скорочується час розробки та

спрощується підтримка застосунку [20]. Однак, такі застосунки вони можуть мати обмеження з точки зору продуктивності, особливо при складних обчисленнях, та доступу до деяких специфічних функцій пристрою.

Гібридний підхід використовує веб-технології (HTML, CSS, JavaScript) в поєднанні з нативними контейнерами, такими як Apache Cordova або Ionic [21]. Застосунки, розроблені цим методом, по суті є веб-застосунками, вбудованими в нативну оболонку. Такий підхід дозволяє значно прискорити розробку, особливо якщо команда має досвід роботи з веб-технологіями. Однак гібридні застосунки зазвичай поступаються нативним у швидкодії та користувацькому досвіді, що може бути критичним для фінансового програмного забезпечення.

Порівняльну характеристику описаних методів наведено у таблиці 1.2.

Таблиця 1.2 – Порівняльна характеристика методів розробки

Критерій	Нативна розробка	Кросплатформена розробка	Гібридний підхід
Продуктивність	Висока	Середня	Низька
Доступ до апаратних функцій	Високий	Середній	Низький
Інтеграція з ОС	Висока	Середня	Низька
Рівень безпеки	Високий	Середній	Низький
Швидкість розробки	Низька	Висока	Висока
Витрати на розробку	Високі	Середні	Низькі
Складність підтримки	Висока	Низька	Низька
Користувацький досвід	Високий	Середній	Низький
Ефективність для фінансових застосунків	Висока	Висока	Середня
Потреба в спеціалізованих знаннях	Висока	Середня	Низька

Проаналізувавши особливості кожного підходу, для розробки мобільного застосунку з ведення особистих фінансів було обрано кросплатформну розробку з використанням React Native, оскільки React Native являє собою оптимальне рішення для розробки фінансового застосунку завдяки унікальному поєднанню продуктивності, гнучкості та ефективності розробки.

Архітектура React Native дозволяє досягти високої продуктивності завдяки виконанню коду у окремому потоці та використанню нативних компонентів інтерфейсу. Для застосунку для ведення особистих фінансів це означає швидку візуалізацію складних графіків, діаграм та аналітичних даних без затримок. Крім цього, React Native надає можливості інтеграції з нативним кодом через нативні модулі, що дозволяє використовувати специфічні для платформи функції безпеки, що важливо для захисту фінансової інформації користувачів. Також, React Native підтримує принцип «write once, run anywhere», що дозволяє підтримувати єдину кодову базу для iOS та Android, одночасно забезпечуючи дотримання дизайн-стандартів кожної платформи.

Таким чином, кросплатформна розробка з використанням React Native представляє оптимальне рішення для створення мобільного застосунку з ведення особистих фінансів, поєднуючи ефективність розробки та можливості забезпечення необхідного рівня безпеки для фінансових операцій.

1.4 Постановка задач дослідження

У результаті аналізу методів розв’язання поставленої задачі, аналіз стану застосунків для ведення особистих фінансів, порівняння аналогів було сформовано такі функціональні вимоги:

1.Аутентифікація та управління обліковим записом:

- Користувач може зареєструватися за допомогою імені, email і пароля (мін. 6 символів).
- Користувач може увійти в систему, отримавши JWT-токен для доступу до захищених ресурсів.
- Після входу користувач може переглянути свої дані (GET /api/auth/user) і змінити пароль у будь-який момент.

2.Управління транзакціями:

- Додавання нової транзакції з параметрами: сума, дата, категорія (дохід/витрата), опис (POST /api/transactions).

- Перегляд списку транзакцій із можливістю фільтрації за датами, типом (дохід/витрата) та категорією (GET /api/transactions).
- Пагінація результатів та відображення загальної кількості записів.
- Редагування (PUT /api/transactions/:id) та видалення (DELETE /api/transactions/:id) окремих транзакцій.
- Отримання статистики транзакцій: загальна сума доходів/витрат, баланс, розподіл по категоріях (GET /api/transactions/stats).

3.Управління бюджетами:

- Створення бюджету на місяць із списком категорій і лімітів по кожній.
- Перегляд всіх бюджетів користувача, з фільтром за роком.
- Перегляд детального бюджету за конкретний місяць.
- Оновлення бюджету (зміна лімітів, назв категорій).
- Видалення бюджету.
- Автоматичний підрахунок сум витрат по категоріях на основі транзакцій користувача та відображення прогресу.

4.Фінансове здоров'я:

- Розрахунок індексу фінансового здоров'я на основі останніх 3 місяців транзакцій:
  - Співвідношення дохід/витрати.
  - Різноманітність джерел доходу.
  - Виконання фінансових цілей.
  - Формування загального скору (0–100).
- Збереження кожного розрахунку з датою і можливість переглядати історію змін.

5.Мобільний Інтерфейс (Expo/React Native)

- Екрани:
  - Login / Register — форми автентифікації;
  - Dashboard — зведена інформація про баланс, бюджет та індекс здоров'я;
  - Transactions — список транзакцій з фільтрами;

- Add Transaction — форма додавання;
- Budget / Create Budget — перегляд та створення бюджетів;
- Financial Health — графік індексу за часом;
- Advice — перелік персоналізованих порад;
- Settings — екран налаштувань.

- Зберігання JWT-токена та інших налаштувань у AsyncStorage.
- Використання axios для взаємодії з REST API та обробки помилок/стану завантаження.

Окрім функціональних, визначено також нефункціональні вимоги:

1.Безпека та конфіденційність:

- Всі API-запити мають йти по HTTPS.
- Використання JWT для авторизації, bcrypt для хешування паролів.
- Захист від CSRF/CORS налаштований через helmet і cors.
- Вхідна валідація даних (express-validator) на сервері.

2.Продуктивність

- Час відповіді API має не перевищувати 200 мс при середньому навантаженні.
- Плавний UI із часом рендерінгу екранів до 100 мс (без «підвисань» при навігації).

3.Надійність та доступність:

- Сервер повинен мати доступність ≥ 99,9 % в робочий час.
- Обробка всіх неочікуваних помилок централізовано через middleware.
- Регулярне резервне копіювання бази даних (MongoDB).

4.Масштабованість

- Модульна архітектура: можливість горизонтального розширення бекенду (кластеризація Node.js).
- Структура REST API дозволяє додавати нові сервіси без зміни існуючих маршрутів.

5.Портативність та сумісність:

•Мобільний додаток – кросплатформний (Android, iOS, Web через Expo).

•Підтримка мінімальних версій ОС: Android 8.0+, iOS 13+.

•Бекенд сумісний з MongoDB 4.4+.

6.Юзабіліті:

•Інтуїтивно зрозумілий інтерфейс: чіткі форми, валідація на клієнті, сповіщення про помилки.

•Дотримання принципів доступності (контрастність, розмір шрифту, елементи керування).

7.Логування та моніторинг:

•Серверне логування використовуючи morgan; можливість підключити APM (NewRelic, Datadog).

•На стороні клієнта – централізований відлов помилок і надсилання до сервісу аналітики.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

Проаналізовано стан питання розробки застосунків для ведення особистих фінансів. Було визначено, що на сьогодні існує значна кількість застосунків для управління особистих фінансів, однак більшість із них мають обмеження щодо функціональності, персоналізації та прогнозування фінансової поведінки користувачів, через що актуальною є розробка застосунку для управління особистими фінансами;

Проведено дослідження існуючих аналогів, а саме: Mint, Personal Capital, You Need A Budget, проведено порівняльний аналіз із власною розробкою шляхом побудови порівняльної таблиці їх характеристик, в результаті чого було доведено актуальність власної розробки, та її переваги над аналогами, до яких належить: наявність персоналізованих рекомендацій та розрахунок індексу фінансового стану користувача.

2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ ЗАСТОСУНКУ

2.1 Аналіз інформаційного забезпечення

Основу інформаційного забезпечення застосунку для обліку фінансів складають дані про користувачів, транзакції, бюджети, показники фінансового здоров'я, персоналізовані поради та налаштування системи [22].

Дані про користувача включають в себе електронну адресу та ім’я користувача, а також пароль. Ця інформація необхідна для створення профілю користувача, за яким закріплюються усі інші дані користувача у базі даних. Це дозволяє отримувати доступ до усіх даних з будь-якого пристрою, для цього необхідно лише авторизуватися, увівши свою електронну адресу та пароль.

Дані про бюджет включають в себе планову загальну суму витрат за обраний період — тиждень, місяць чи рік. Важливою особливістю є механізм автоматичного оновлення фактичних витрат за категоріями на основі здійснених транзакцій, що забезпечує постійну актуальність даних бюджету, що створює можливість для ефективного відстеження ступеня дотримання фінансового плану та своєчасного виявлення перевитрат у окремих категоріях.

Модель фінансового здоров'я надає оцінку фінансового стану користувача шляхом агрегування даних з усіх компонентів застосунку та обчислення таких ключових показників, як: співвідношення доходів до витрат, наявність подушки безпеки, диверсифікація джерел доходу та прогрес у досягненні фінансових цілей. Кожен показник представлений парою значень, а саме абсолютним значенням та оцінкою за шкалою від 0 до 100. Такий підхід дозволяє відстежувати динаміку змін фінансового стану з часом та виявляти проблемні аспекти, що потребують уваги.

Для зберігання даних застосунок використовує технологію AsyncStorage для локального збереження інформації на пристрої користувача. Такий підхід має свої переваги, зокрема можливість автономної роботи без підключення до мережі та захист персональних фінансових даних, однак має обмеження щодо обсягу збережених даних та не дозволяє синхронізації між кількома пристроями.

Важливим аспектом інформаційного забезпечення є механізми валідації та

обробки даних. У застосунку реалізовано систему перевірки коректності введених даних на рівні компонентів інтерфейсу та на рівні Redux-thunks перед збереженням у сховище. Це дозволяє запобігти введенню некоректних даних та забезпечити цілісність інформаційної бази. Додатково реалізовано функції форматування фінансових даних для їх коректного відображення в інтерфейсі з урахуванням обраної користувачем валюти.

Таким чином, розроблене інформаційне забезпечення застосунку дозволяє додавати дані про фінанси, бюджет, отримувати аналіз та оцінку поточного фінансового стану користувача, реєструватися та авторизуватися у застосунку.

2.2 Удосконалення методу «фінансового здоров'я» користувача

Метод «фінансового здоров'я» користувача надає оцінку фінансового стану користувача на основі багатьох параметрів, яка визначається шляхом аналізу такі ключових показників:

- Обсяг резервного фонду (подушка безпеки).
- Рівень диверсифікації джерел доходу.
- Ступінь досягнення фінансових цілей.

На виході метод формує єдиний інтегральний індекс у діапазоні 0…100, що відображає загальну фінансову стійкість та допомагає вчасно ідентифікувати потенційні ризики.

Алгоритм роботи:

1. Збір даних:

– Агрегуються всі транзакції (доходи та витрати) за обраний період.

– Витягуються налаштовані користувачем фінансові цілі та поточний баланс резервного фонду.

2. Обчислення базових метрик:

2.1. Співвідношення загального доходу до загальних витрат.

2.2. Тривалість «покриття» витрат наявними заощадженнями (кількість місяців).

2.3. Кількість унікальних джерел доходу (мера диверсифікації).

2.4. Відсоток виконання кожної фінансової цілі відносно запланованих термінів.

3. Формування зваженого індексу:

– Кожна метрика отримує вагу за узгодженими бізнес-правилами.

– Розраховується зважене середнє значення та масштабування у шкалу 0..100.

4. Генерація звіту та рекомендацій:

– Формується деталізований звіт із візуалізацією всіх складових індексу.

– Додаються поради щодо підвищення резервного фонду, оптимізації витрат, диверсифікації доходів та корекції фінансових цілей.

5. Збереження результатів: індекс та супровідні дані зберігаються в базі для історії та подальшого моніторингу.

Отже, метод «фінансового здоров'я» пропонує інструмент для самоаналізу, перетворює поняття фінансового благополуччя на конкретні, вимірювані показники, що спонукає користувачів приймати більш обґрунтовані фінансові рішення, орієнтуючись на об'єктивні дані.

2.3 Розробка моделі застосунку

Застосунок для обліку особистих фінансів складається з взаємопов'язаних компонентів, що забезпечують увесь функціонал для додавання та управління інформацією про особисті фінанси.

Для реалізації інтерфейсу використовується бібліотека React Native Paper, яка надає готові компоненти з дизайном Material Design.

Навігація між екранами реалізована через React Navigation, що пропонує гнучкий підхід до організації переходів. У застосунку використано комбінацію навігації для авторизації та модальних вікон, а також навігації з вкладками для основних розділів додатку.

Візуалізація даних здійснюється за допомогою React Native Chart Kit, що дозволяє створювати інформативні графіки та діаграми для аналізу фінансових показників. Бібліотека забезпечує лінійні графіки для відстеження динаміки, кругові діаграми для відображення розподілу витрат та інші типи візуалізацій.

Управління станом застосунку централізовано через Redux в поєднанні з Redux Toolkit, що спрощує роботу з глобальним станом. Архітектура стану розділена на окремі «слайси», кожен з яких відповідає за певний функціональний блок: автентифікацію, транзакції, бюджети, показники фінансового здоров'я, поради та налаштування.

Модель застосунку наведена на рисунку 2.1.

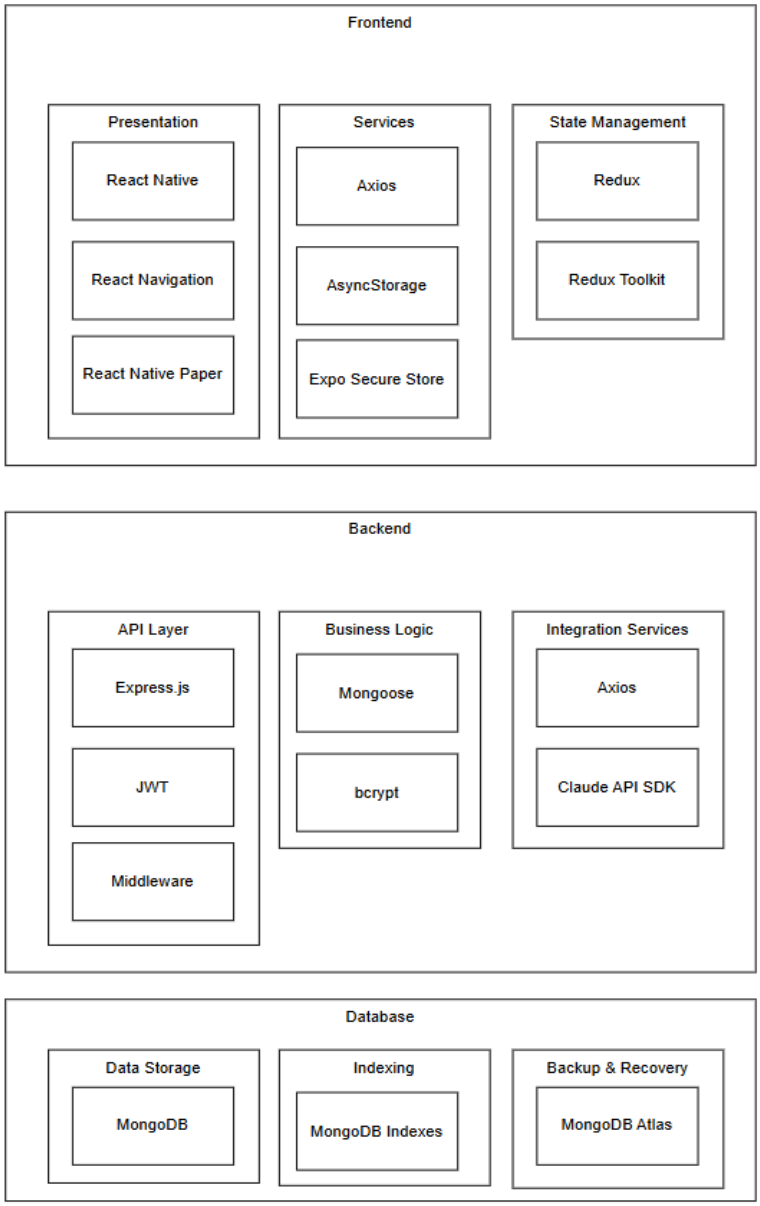


Рисунок 2.1 — Модель застосунку

Асинхронні операції, такі як імітація запитів до серверу та робота з локальним сховищем, реалізовані через Redux Thunk. Ця технологія дозволяє виконувати складні послідовності дій з оновленням стану на різних етапах

виконання.

Бібліотека React Redux використовується для ефективного зв'язування компонентів React з Redux-сховищем, застосовуючи підхід з хуками useSelector та useDispatch, що спрощує доступ до даних та виконання дій.

Для зберігання даних на пристрої користувача застосовано AsyncStorage для загальних даних, таких як транзакції, бюджети, історія фінансового здоров'я та налаштування, і Expo SecureStore для зберігання чутливої інформації, зокрема даних користувача для автентифікації.

Система автентифікації включає локальну реєстрацію та вхід користувачів. Для забезпечення безпеки паролів використовується бібліотека bcryptjs, що реалізує сучасні алгоритми хешування.

Система фінансового здоров'я використовує математичні алгоритми для обчислення та нормалізації різних фінансових показників. Ця частина використовує спеціалізовані функції в Redux Thunk для аналізу транзакцій та бюджетів.

Для забезпечення коректного відображення фінансових даних та валідації введених користувачем значень використовуються спеціалізовані утиліти, а система валідації реалізує логіку перевірки введених даних, забезпечуючи коректність інформації в транзакціях, бюджетах та інших частинах застосунку.

Розроблена модель застосунку дозволяє реалізувати увесь необхідний функціонал для мобільного застосунку для управління особистими фінансами.

2.4 Розробка алгоритмів роботи застосунку

Алгоритм додавання транзакції дозволяє додавання інформації про витрати та доходи користувача. Користувач вказує такі дані, як сума витрат чи доходів, вибір виду транзакції, категорії, дати витрат або отримання доходів, після чого інформація зберігається у базу даних. Блок-схема цього алгоритму наведена на рисунку 2.2.

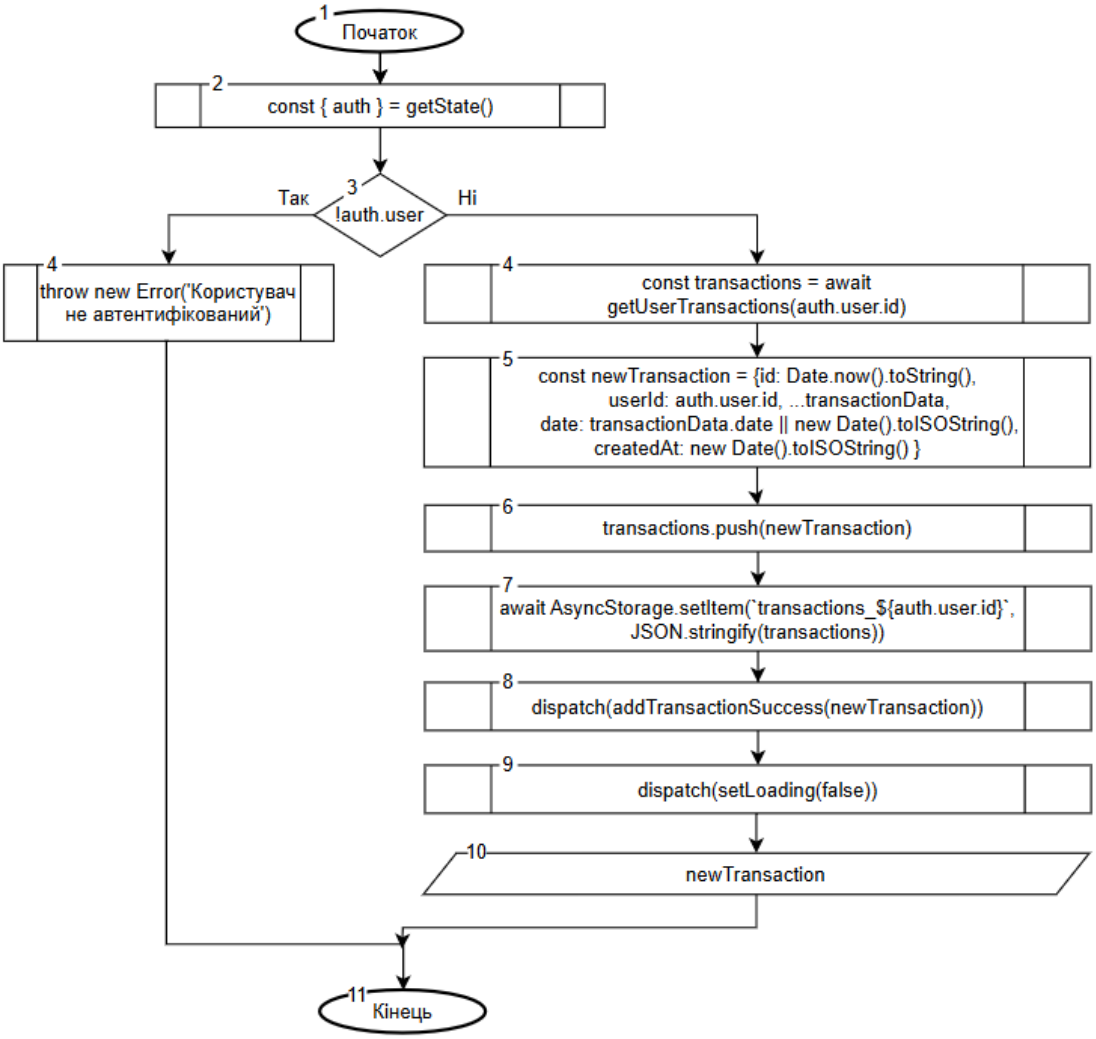


Рисунок 2.2 — Блок-схема алгоритму додавання транзакції

Алгоритм управління фінансовими транзакціями є центральним елементом застосунку, що забезпечує створення, редагування, видалення та категоризацію фінансових операцій. Процес розпочинається з отримання даних про нову транзакцію, включаючи суму, тип, категорію та дату. Після валідації введених даних створюється новий запис, який зберігається у локальному сховищі. Алгоритм також включає методи для отримання транзакцій з використанням фільтрації, сортування та пагінації, що дозволяє ефективно працювати з великими обсягами даних. Особливу увагу приділено синхронізації даних про транзакції з іншими функціональними компонентами системи, зокрема з модулем бюджетування, де дані про витрати використовуються для відстеження виконання бюджету.

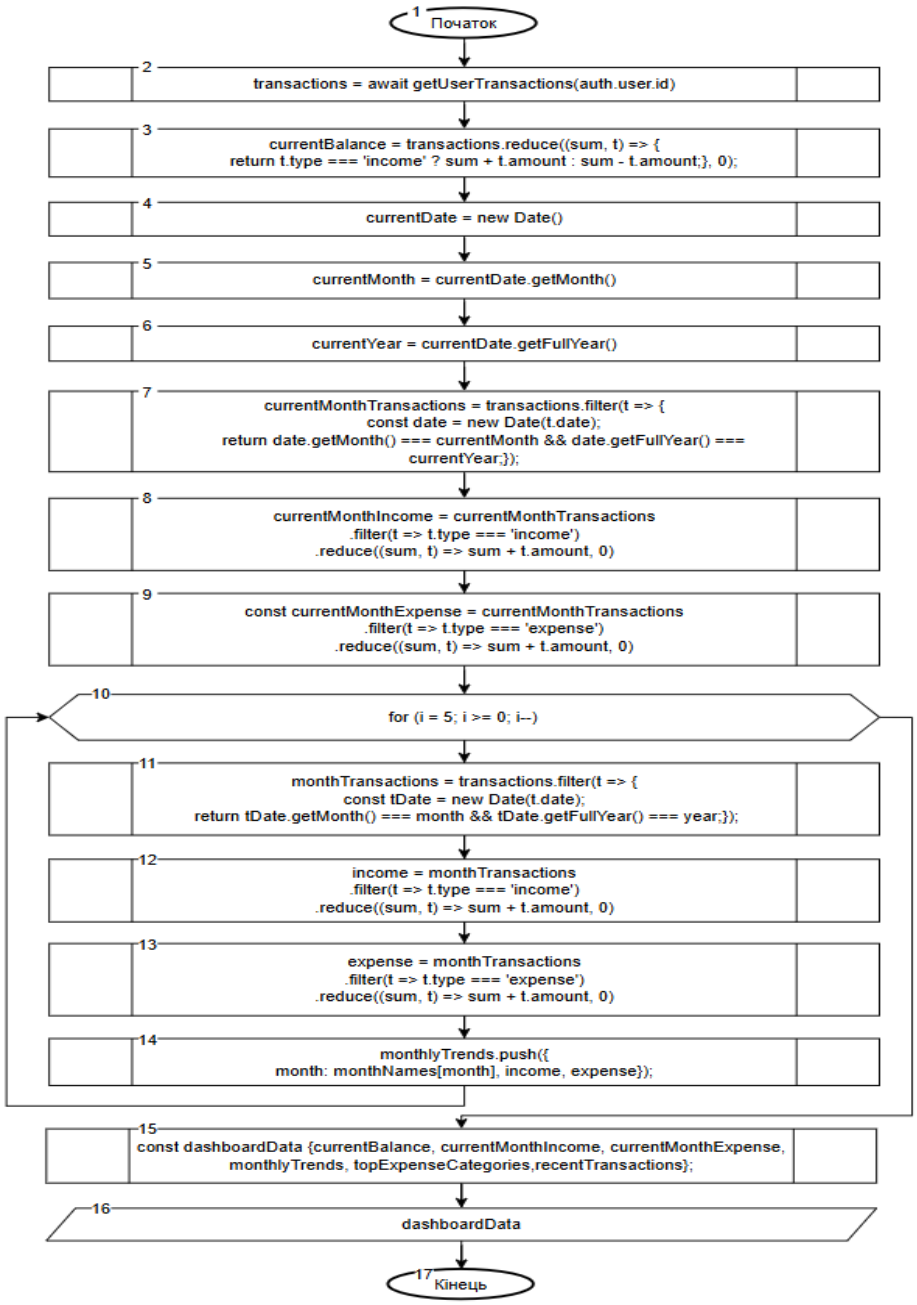


Рисунок 2.3 — Блок-схема алгоритму управління фінансовими транзакціями

Алгоритм формування та контролю бюджету забезпечує планування фінансів користувача на місячній основі. Основний процес включає створення бюджету з розподілом коштів за категоріями витрат та подальше відстеження фактичних витрат відносно запланованих. Алгоритм підтримує динамічне оновлення даних про виконання бюджету на основі здійснених транзакцій, автоматично перераховуючи відсоток виконання при додаванні нових витрат. Для зручності аналізу система визначає категорії з перевищенням бюджету та ті, що

наближаються до ліміту. Важливою функцією алгоритму є обчислення прогресу виконання бюджету з урахуванням часової компоненти, що дозволяє оцінити, чи відповідають поточні витрати плановому графіку. Блок-схема цього алгоритму наведена на рисунку 2.4.

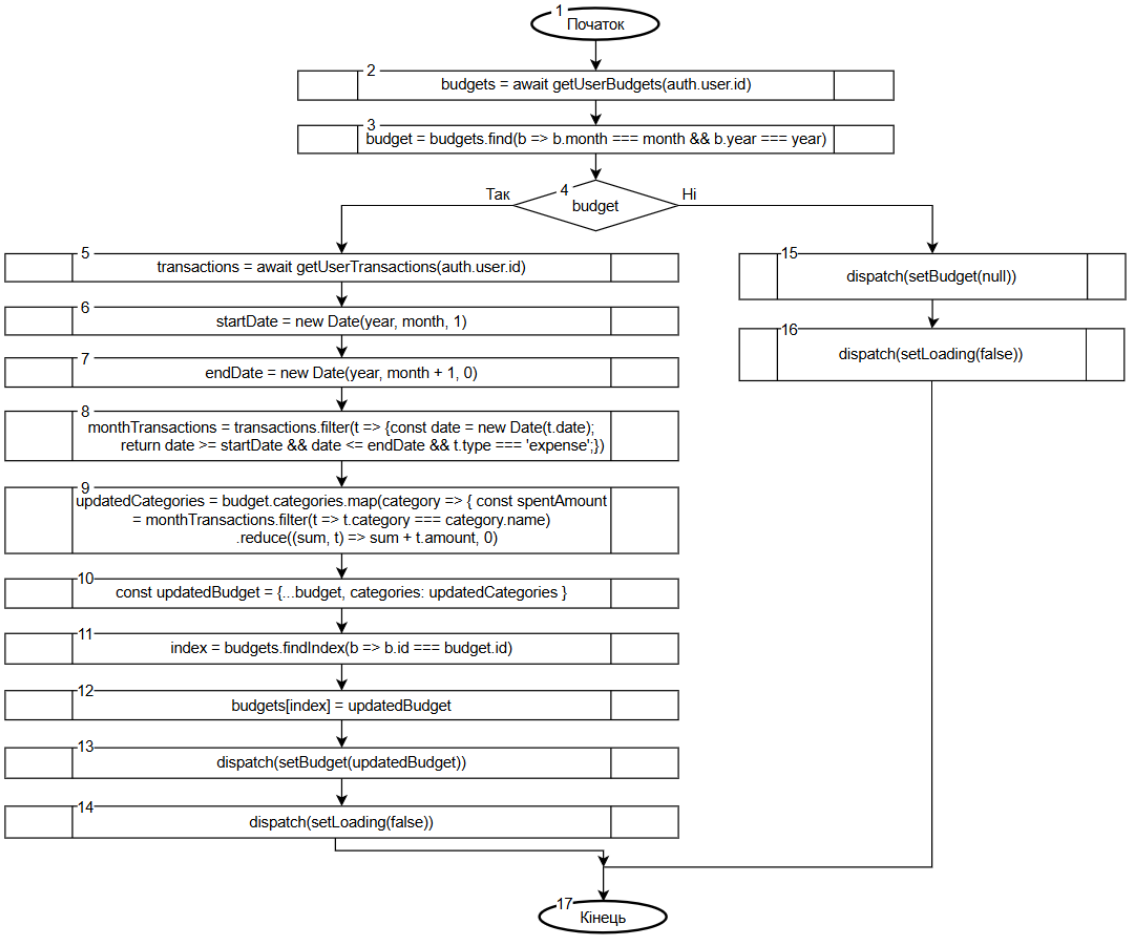


Рисунок 2.4 - Блок-схема алгоритму управління бюджетом

Алгоритм оцінки фінансового здоров'я реалізує комплексний аналіз фінансового стану користувача на основі кількох ключових показників. Спочатку виконується збір даних про транзакції за останні три місяці. Далі обчислюються чотири основні метрики: співвідношення доходів до витрат, наявність подушки безпеки (кількість місяців, на які вистачить заощаджень), диверсифікація джерел доходу та прогрес у досягненні фінансових цілей. Кожен показник нормалізується за шкалою від 0 до 100, після чого обчислюється загальний індекс фінансового здоров'я як середнє арифметичне цих показників. Отримані дані зберігаються в

історії, що дозволяє відстежувати динаміку фінансового стану з часом. Блок-схема алгоритму оцінки фінансового здоров’я наведена на рисунку 2.5.

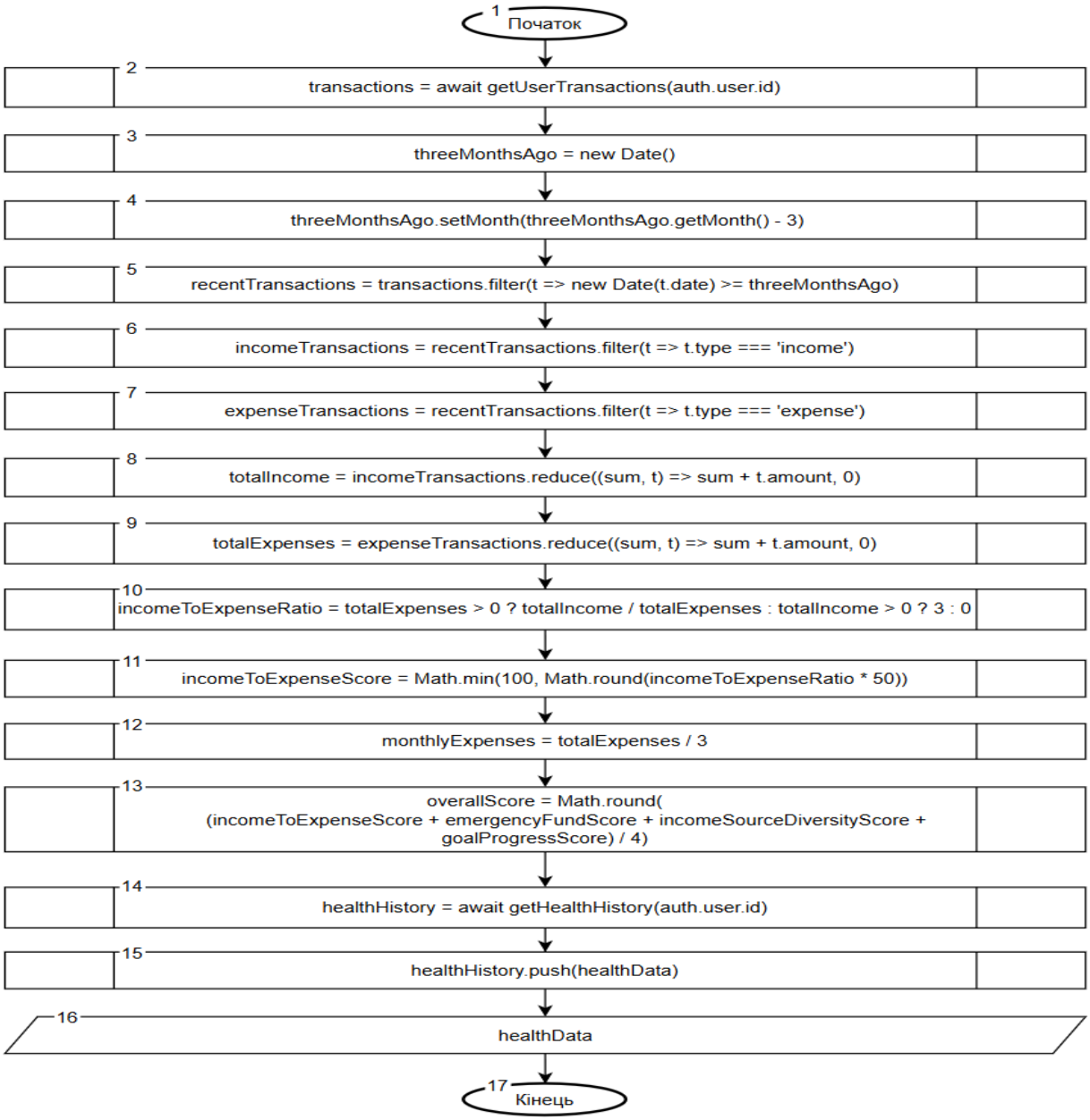


Рисунок 2.5 — Блок-схема алгоритму розрахунку індексу фінансового здоров’я

Розроблені алгоритми забезпечують основний функціонал застосунку для управління особистих фінансів. Вони дозволяють додавання, керування транзакціями та бюджетом, отримувати інформацію про «фінансове здоров’я» користувача.

Розроблено діаграму класів серверної частини системи. Вона відповідає за

обробку даних користувачів, транзакцій, бюджету, розрахунку індексу фінансового здоров'я, а також збереження цих даних. Діаграма класів серверної частини системи наведена на рисунку 2.6.

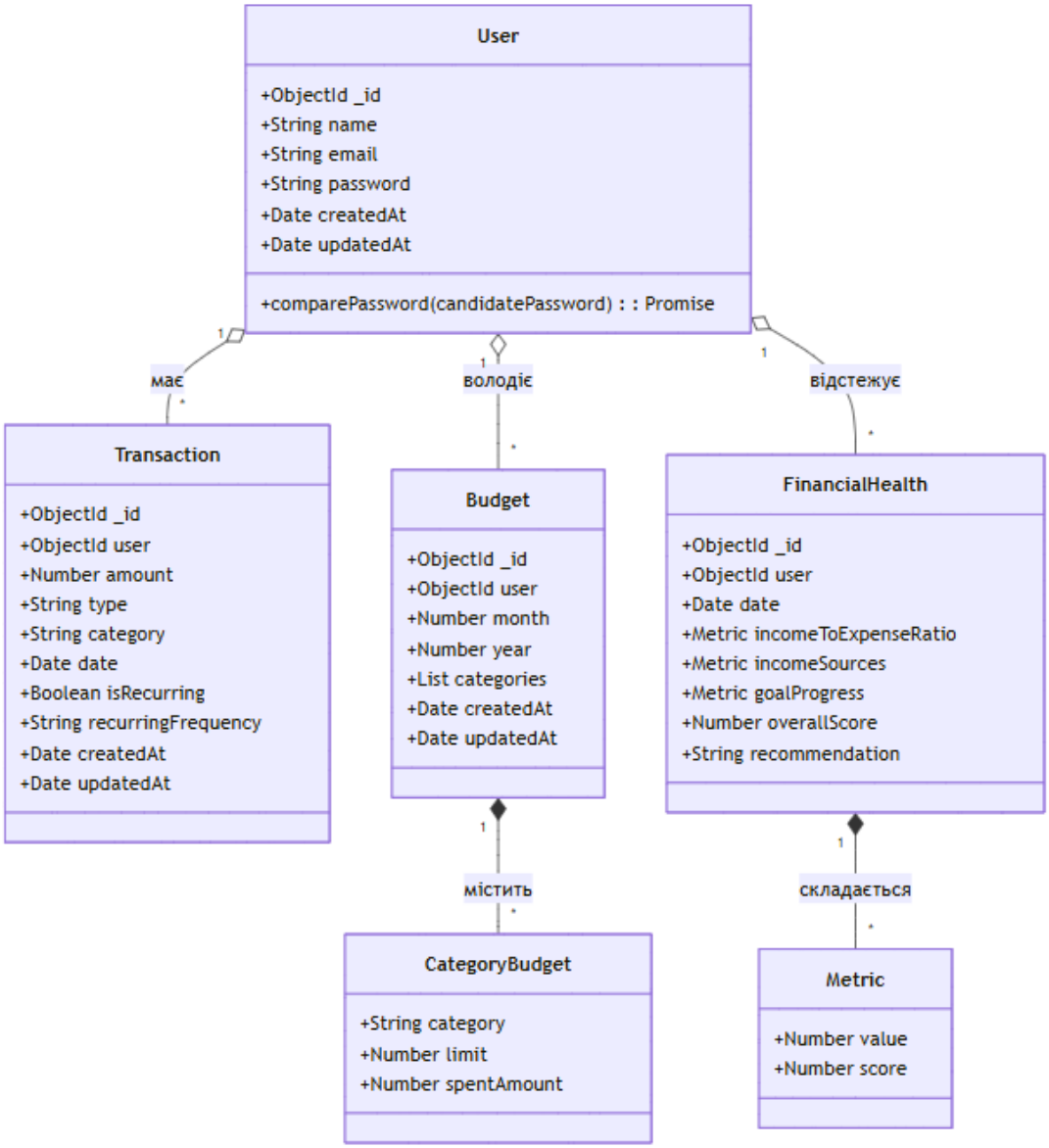


Рисунок 2.6 - Діаграма класів бекенду

Клас User моделює основного користувача системи. Він зберігає персональні дані, облікові атрибути й забезпечує механізм перевірки пароля та генерації токена.

Поля класу:

- _id: ObjectId — унікальний ідентифікатор користувача в базі.

- name: String — ім'я або псевдонім.

- email: String — електронна пошта (унікальне поле).

- password: String — хешований пароль.

- createdAt: Date — дата реєстрації.

- updatedAt: Date — дата останнього оновлення облікового запису.

Функції:

- comparePassword(candidatePassword: String): Promise<Boolean> — порівнює переданий пароль із збереженим хешем.

- (можливі ще допоміжні методи, наприклад, для генерації JWT).

Клас Transaction відповідає за збереження кожного фінансового запису — витрати чи надходження. Також містить інформацію про категорію, дату та параметри рекурентності.

Поля класу:

- _id: ObjectId — ідентифікатор транзакції.

- user: ObjectId — посилання на власника (User).

- amount: Number — сума операції.

- type: String — тип ("income" або "expense").

- category: String — категорія витрати/доходу.

- date: Date — дата проведення.

- isRecurring: Boolean — чи є повторюваною.

- recurringFrequency: String — інтервал повторення (щомісяця, щотижня тощо).

- createdAt: Date — дата створення запису.

- updatedAt: Date — дата останнього оновлення.

Функції:

- save(): Promise<Transaction> — зберігає або оновлює запис у базі.

Клас Budget моделює план бюджету користувача на конкретний місяць і рік. Складається зі списку категорій, кожна з яких має ліміт і вже використані кошти.

Поля класу:

- _id: ObjectId — ідентифікатор бюджету.

- user: ObjectId — посилання на власника (User).

- month: Number — номер місяця (1–12).

- year: Number — рік.

- categories: List<CategoryBudget> — перелік бюджетних категорій.

- createdAt: Date — дата створення.

- updatedAt: Date — дата останнього оновлення.

Функції:

- save(): Promise<Budget> — збереження/оновлення бюджету.

- calculateSpent(): Number — обчислює суму всіх витрат за категоріями.

- addCategory(category: CategoryBudget): void — додає нову категорію до бюджету.

Клас CategoryBudget - допоміжний клас для поділу бюджету на категорії. Кожна категорія містить налаштований ліміт та відслідковує вже витрачену суму.

Поля класу:

- category: String — назва категорії (наприклад, "Food", "Transport").

- limit: Number — встановлений ліміт витрат.

- spentAmount: Number — сума вже витрачених коштів.

Методи:

- updateSpent(amount: Number): void — додає до spentAmount нову витрату.

- isExceeded(): Boolean — повертає true, якщо spentAmount перевищує limit.

5. Клас FinancialHealth зберігає щоденні чи щомісячні показники “фінансового здоров’я” користувача, включно з комплексними метриками і рекомендаціями.

Поля:

_id: ObjectId — ідентифікатор запису.

user: ObjectId — власник показників.

date: Date — дата вимірювання.

incomeToExpenseRatio: Metric — співвідношення доходів до витрат.

incomeSources: Metric — різноманітність джерел доходу.

goalProgress: Metric — прогрес до фінансової мети.

overallScore: Number — зведений бал здоров’я.

recommendation: String — текстова порада.

Функції:

- calculateOverallScore(): Number — агрегує метрики в єдиний бал.

- generateRecommendation(): String — формує текстову пораду на основі показників.

Клас Metric - універсальний контейнер для кожного окремого показника: містить реальне значення та нормалізований бал.

Поля класу:

- value: Number — фактичне числове значення метрики (наприклад, коефіцієнт).

- score: Number — приведений до шкали 0–100 оцінковий бал.

Функції:

- normalize(min: Number, max: Number): void — обчислює score на підставі value, min і max.

- isHealthy(threshold: Number): Boolean — перевіряє, чи score вище заданого порогу.

Розроблено діаграму класів клієнтської частини системи. Вона включає в себе класи для реєстрації, авторизації, додавання транзакцій, створення бюджету, налаштування, перегляду списку транзакцій, індексу фінансового здоров’я, порад щодо формування бюджету.

Ці класи дозволяють користувачу зручно взаємодіяти з функціоналом застосунку.

Діаграма станів клієнтської частини системи наведена на рисунку 2.7.

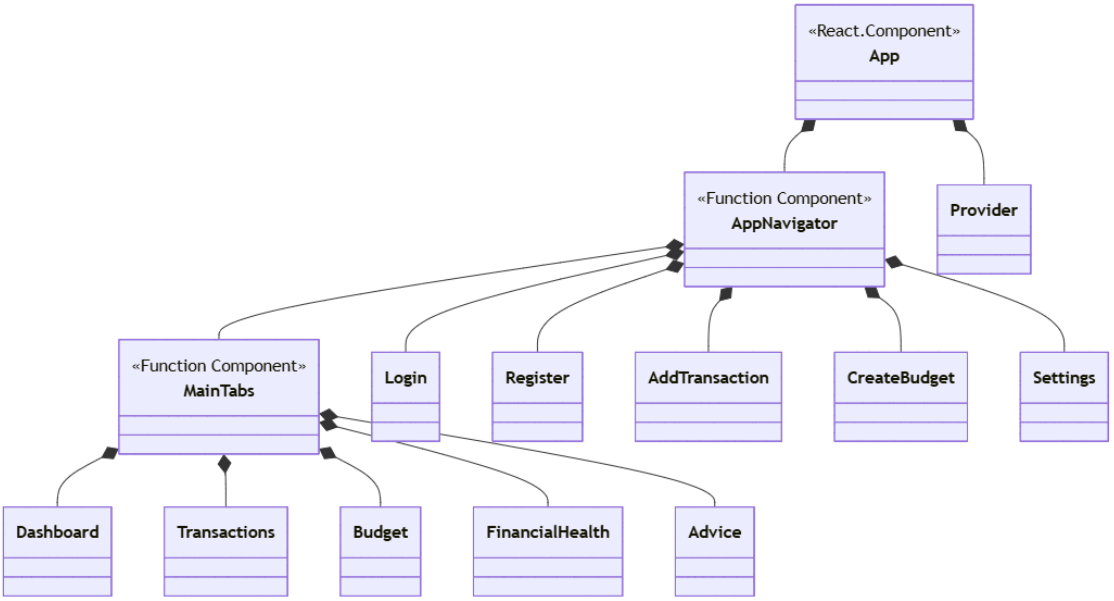


Рисунок 2.7 - Діаграма класів клієнтської частини системи

Таким чином, було розроблено алгоритми роботи застосунку для ведення особистих фінансів та описано діаграми класів, що відповідають за клієнтську та серверну частини застосунку, описано призначення та вміст кожного з класів.

2.5 Висновки

Шляхом побудови діаграми компонентів застосунку було вирішено задачу з розробки моделі застосунку для ведення особистих фінансів, яка складається з взаємопов'язаних компонентів, що забезпечують увесь функціонал для додавання та управління інформацією про особисті фінанси;

Удосконалено метод «фінансового здоров’я» користувача, який враховує рівень диверсифікації джерел доходу, запас коштів та виконання поставлених фінансових цілей, на основі чого розраховує індекс, що визначає рівень фінансової стабільності користувача.

Було побудовано діаграми класів та алгоритмів роботи застосунку, чим було вирішено завдання з розробки діаграм та алгоритмів роботи застосунку. Розроблені діаграми та алгоритми роботи дозволяють додавати транзакції, керувати транзакціями, управляти бюджетом, розраховувати індекс «фінансового здоров’я».

3 РОЗРОБКА ЗАСТОСУНКУ

3.1 Варіантний аналіз та вибір мови програмування

JavaScript [23] — мова програмування, що в основному використовується для веброзробки. Головною перевагою JavaScript є гнучкість та динамічність, завдяки чому код легко адаптується під різноманітні завдання і дозволяє розробникам швидко створювати прототипи та реалізовувати функціонал.

JavaScript має слабку типізацію, що дозволяє зменшити час написання коду, але при цьому міститься ризик появи помилок при роботі коду, що супроводжується при цьому складністю виявлення причин проблеми. Екосистема JavaScript включає в себе численні бібліотеки та фреймворки, такі як React, Angular, Vue.js для фронтенду, і Express.js для бекенду, що суттєво розширюють можливості використання мови для вирішення різноманітних завдань.

Kotlin [24] — це сучасна мова програмування, розроблена компанією JetBrains, яка здобула значну популярність завдяки тому, що стала офіційною мовою для розробки під платформу Android. Вона вирізняється лаконічністю, безпечністю та чітким синтаксисом, що дозволяє знизити кількість помилок на етапі компіляції. Kotlin повністю сумісна з Java, що дає змогу поступово мігрувати на неї проєкти, написані на Java, без значних витрат ресурсів.

Мова Kotlin пропонує високий рівень типобезпеки та підтримку функціонального програмування, що робить її зручною для створення складних додатків. Вона також активно застосовується у багатоплатформній розробці завдяки Kotlin Multiplatform (KMP), що дозволяє використовувати один код для різних платформ (Android, iOS, веб, десктоп). Крім того, завдяки зручним конструкціям, таким як data-класи, корутини і розширення функцій, Kotlin суттєво спрощує написання та підтримку чистого, продуктивного коду.

Python [25] — це мова програмування загального призначення, що вирізняється простим, зрозумілим і легким для освоєння синтаксисом. Вона використовується у різних галузях, включаючи веброзробку, аналіз даних, машинне навчання, штучний інтелект, автоматизацію та наукові дослідження. Завдяки

широкій підтримці бібліотек (наприклад, NumPy, Pandas, TensorFlow) Python є основною мовою для роботи з даними та проведення різних видів аналітичних досліджень.

Особливо цінною рисою Python є його читабельність та простота, що сприяє швидкому навчанню та ефективній роботі навіть для початківців. Водночас, незважаючи на простоту, Python має високі показники продуктивності в розробці складних систем завдяки великій спільноті та наявності стабільних фреймворків, таких як Django, Flask, FastAPI для веброзробки. Python також широко використовується для написання скриптів автоматизації завдяки швидкості написання коду та зручній інтеграції з іншими технологіями.

Таблиця 3.1 — Порівняльний аналіз мов програмування

Характеристика	Kotlin	JavaScript	Python
Розвинута екосистема фінансових бібліотек	-	+	+
Мобільна розробка	+	+	-
Інтеграція з SQL базами даних	+	+	+
Асинхронна обробка транзакцій	+	+	+
API банків і платіжних систем	+	+	+
Експорт/імпорт фінансових форматів даних	-	+	+
Інтеграція з аналітичними інструментами	-	+	+
Сумарний бал	4	7	6

Виходячи з результатів порівняльного аналізу, найкращим вибором для розробки мобільного застосунку для ведення особистих фінансів є JavaScript. Тому було прийнято рішення розробити застосунок з використанням цієї мови програмування.

3.2 Розробка бази даних

Таблиця users слугує центральним сховищем інформації про зареєстрованих користувачів системи. Кожен запис містить унікальний ідентифікатор _id, який

забезпечує однозначне посилання на користувача в інших таблицях (через зовнішні ключі або ObjectId). Поле email є унікальним і використовується для аутентифікації та відновлення пароля; його унікальність контролюється на рівні бази даних через індекс. Хешований пароль зберігається в полі password за сучасними стандартами безпеки, а відкритий доступ до цього поля заборонено.

Поля name і createdAt відіграють допоміжну роль: перше — для відображення імені користувача в інтерфейсі, друге — для аудиту і статистики реєстрацій. Опціональне поле financialGoals, яке є масивом об’єктів, дозволяє зберігати декілька фінансових цілей користувача в одному документі. Кожна ціль описується назвою (title), цільовою сумою (targetAmount), поточним станом накопичення (currentAmount) та дедлайном (deadline). Така структура спрощує читання і оновлення цілей без необхідності переходу в інші таблиці.

Поля таблиці users наведено у таблиці 3.2.

Таблиця 3.2 — Поля таблиці users

Поле	Тип	Обов’язкове	Опис
_id	ObjectId	так	Унікальний ідентифікатор документу
name	String	так	Ім’я користувача
email	String	так	Електронна пошта (унікальна)
password	String	так	Хешований пароль
financialGoals	Array of Objects	ні	Перелік фінансових цілей користувача
title	String	так	Назва цілі
targetAmount	Number	так	Бажана сума
currentAmount	Number	ні	Накопичена наразі сума
deadline	Date	ні	Кінцевий термін досягнення цілі
category	String	ні	Категорія (за потреби)
createdAt	Date	ні	Дата створення запису

Таблиця transactions фіксує всі фінансові операції користувачів — доходи й витрати. Кожна транзакція пов’язана з користувачем через поле user (ObjectId із таблиці users). Поле date дозволяє сортувати та фільтрувати записи за часом; за

замовчуванням встановлюється поточна дата і час транзакції.

Поле amount зберігає числове значення суми операції, а type (enum "income" або "expense") визначає природу транзакції. Категоризація включає обов’язкове поле category (наприклад, “Їжа”, “Транспорт”) та опційне subcategory (наприклад, “Ресторан”, “Автобус”). Додаткове текстове поле description дає змогу зберігати пояснення або коментарі. Для періодичних витрат передбачені поля isRecurring та recurringFrequency, які дозволяють автоматично створювати копії транзакцій за розкладом.

Індекси на полях (user, date), (user, type) та (user, category) оптимізують запити до історії операцій конкретного користувача, а також статистичні агрегати (наприклад, підсумок витрат за категоріями або за період). Це вкрай важливо для швидкої генерації звітів і висновків у режимі реального часу. Поля таблиці transactions наведено у таблиці 3.3.

Таблиця 3.3 — Поля таблиці transactions

Поле	Тип	Обов’язкове	Опис
_id	ObjectId	так	Унікальний ідентифікатор транзакції
user	ObjectId	так	Посилання на власника транзакції
amount	Number	так	Сума транзакції
type	String	так	Тип транзакції
category	String	так	Головна категорія
subcategory	String	ні	Підкатегорія
description	String	ні	Опис транзакції
date	Date	ні	Дата й час транзакції
isRecurring	Boolean	ні	Чи є транзакція періодичною

Таблиця budgets відповідає за планування фінансових лімітів на місяці. Для кожного користувача (user) і конкретної пари місяць–рік (month, year) зберігається документ із запланованими сумами за категоріями. Поле categories є масивом об’єктів, де кожен об’єкт визначає назву категорії (name), заплановану суму (plannedAmount) та фактичні витрати в рамках бюджету (spentAmount).

Унікальний композитний індекс по (user, month, year) запобігає дублюванню бюджетів на один і той же період для одного користувача. Також присутні допоміжні поля createdAt та updatedAt для відстеження термінів створення та змін бюджету. Така архітектура дозволяє легко отримувати бюджетні дані за вибраний період, порівнювати фактичні витрати з планом і формувати користувацькі оповіщення (наприклад, «ви перевищили ліміт по їжі»). Поля таблиці budgets наведено у таблиці 3.4.

Таблиця 3.4 — Поля таблиці budgets

Поле	Тип	Обов’язкове	Опис
_id	ObjectId	так	Унікальний ідентифікатор документу
user	ObjectId → users	так	Власник бюджету
month	Number	так	Місяць (0–11)
year	Number	так	Рік (≥ 2000)
categories	Array of Objects	так	Перелік категорій у бюджеті
name	String	так	Назва категорії
plannedAmount	Number	так	Запланована сума
spentAmount	Number	так	Фактично витрачена сума
createdAt	Date	ні	Дата створення запису
updatedAt	Date	ні	Дата останнього оновлення

Таблиця financialhealths призначена для збереження результатів аналізу «фінансового здоров’я» користувача, який обчислюється на основі сукупності ключових показників. Кожен документ містить дату розрахунку (date) та об’єктно-числові пари для таких метрик, як співвідношення доходів до витрат (incomeToExpenseRatio), розмір аварійного фонду в місяцях (emergencyFund), відношення боргів до доходів (debtToIncomeRatio), різноманітність джерел доходу (incomeSourceDiversity) та прогрес у досягненні фінансових цілей (goalProgress).

Кожна метрика зберігається як об’єкт з полями value (змістовне числове значення) і score (нормалізований показник, наприклад від 0 до 100). Поле

overallScore підсумовує всі індикатори в єдину оцінку фінансового стану. Опційне текстове поле recommendation містить конкретні поради щодо поліпшення фінансового здоров’я. Така модель дозволяє відтворювати історію змін ключових фінансових індикаторів, будувати дашборди з трендами та автоматично генерувати персоналізовані рекомендації. Поля таблиці financialhealths наведено у таблиці 3.5.

Таблиця 3.5 — Поля таблиці financialhealths

Поле	Тип	Обов’язкове	Опис
_id	ObjectId	так	Унікальний ідентифікатор документу
user	ObjectId → users	так	Власник запису
date	Date	ні	Дата розрахунку
incomeToExpenseRatio	Object	ні	Співвідношення доходів до витрат
emergencyFund	Object	ні	Місяці покриття витрат фондом
debtToIncomeRatio	Object	ні	Відношення боргів до доходів
incomeSourceDiversity	Object	ні	Кількість джерел доходу
goalProgress	Object	ні	Процент виконання фінансових цілей
overallScore	Number	так	Загальний індекс фінансового здоров’я (0–100)
recommendation	String	ні	Рекомендації на основі аналізу

Ця база даних забезпечує повноцінну підтримку всіх функціональних вимог застосунку: від керування користувачами та обліку транзакцій до планування бюджетів і оцінки фінансового здоров’я. Завдяки чіткій структурі та індексуванню таблиць забезпечується висока продуктивність запитів і надійність зберігання інформації.

3.3 Розробка модуля фінансового здоров’я

Головною задачею модуля фінансового здоров'я є комплексна оцінка фінансового стану користувача на основі аналізу кількох ключових фінансових показників. Для реалізації цієї задачі було створено клас HealthCalculator, що приймає в себе в якості параметра історію транзакцій користувача. При створенні і запуску класу автоматично виконується аналіз транзакцій, розраховуються фінансові показники та формується загальний індекс фінансового здоров'я. Функція розрахунку фінансового здоров’я наведена на рисунку 3.1.

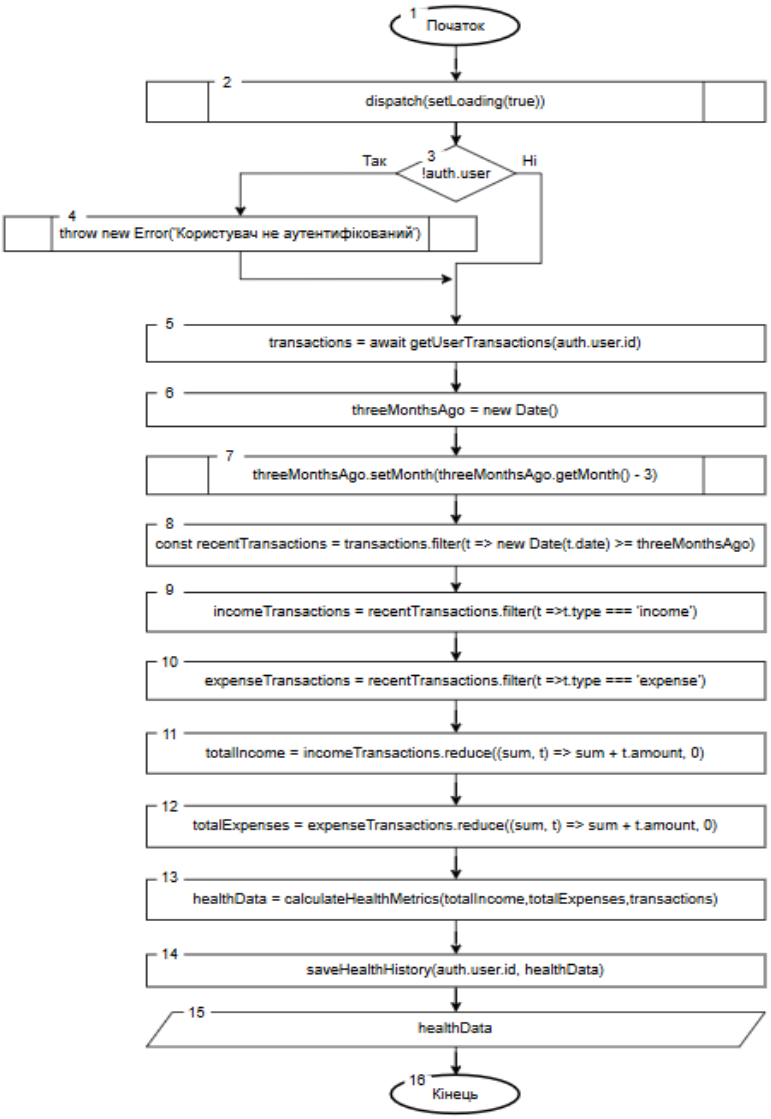


Рисунок 3.1 – Функція розрахунку фінансового здоров'я

Крім того, було розроблено окрему функцію для обчислення основних

показників фінансового здоров'я, а саме: співвідношення доходів до витрат, подушки безпеки, диверсифікації джерел доходу та прогресу в досягненні фінансових цілей (рис. 3.2). Ця функція нормалізує кожен із показників за шкалою від 0 до 100 та обчислює загальний індекс здоров'я як середнє значення цих показників.

Крім того, було розроблено окрему функцію для обчислення основних показників фінансового здоров'я, а саме: співвідношення доходів до витрат, подушки безпеки, диверсифікації джерел доходу та прогресу в досягненні фінансових цілей (рис. 3.2). Ця функція нормалізує кожен із показників за шкалою від 0 до 100 та обчислює загальний індекс здоров'я як середнє значення цих показників.

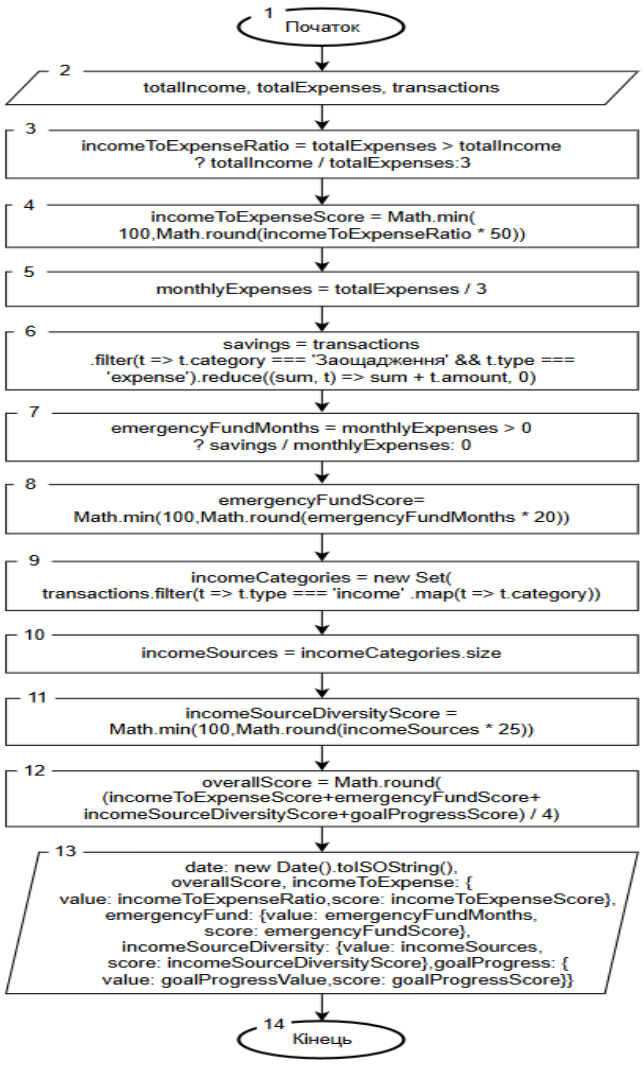


Рисунок 3.2 – Функція обчислення показників фінансового здоров'я

Отже, розроблений модуль фінансового здоров'я реалізує відповідний метод, що дає кількісну оцінку фінансовому стану користувача.

3.4 Розробка модуля обліку фінансових транзакцій

Головною задачею модуля обліку фінансових транзакцій є забезпечення користувачів інструментами для додавання, редагування та управління записами про доходи та витрати. Для реалізації цієї задачі було створено клас TransactionManager, що включає функціонал роботи з транзакціями різних типів та категорій. При створенні і запуску класу автоматично ініціалізується взаємодія з локальним сховищем даних та завантажуються існуючі транзакції користувача (рис. 3.3).

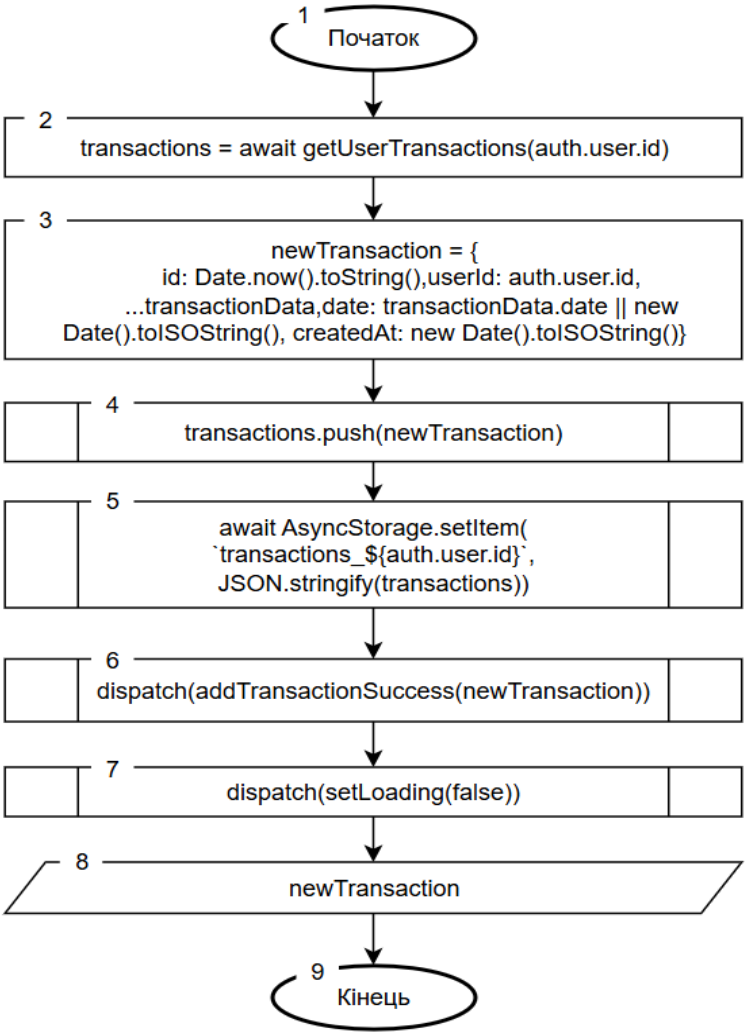


Рисунок 3.3 – Функція додавання фінансової транзакції

Крім того, було розроблено окремий компонент для інтерфейсу додавання нових транзакцій, який забезпечує зручне введення даних про доходи та витрати (рис. 3.4). Цей компонент надає користувачу гнучкий інтерфейс для вибору типу транзакції, категорії, введення суми та додаткової інформації.

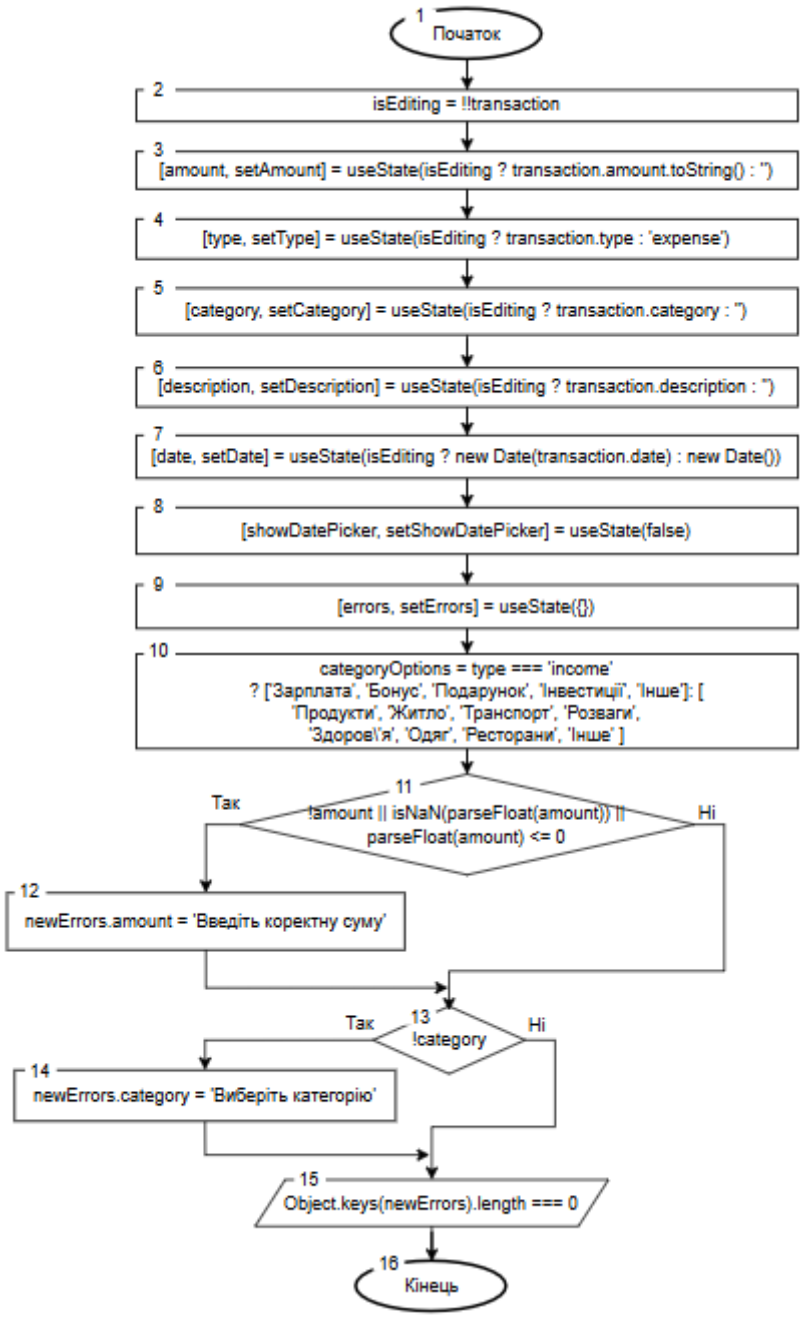


Рисунок 3.4 – Компонент форми додавання/редагування транзакції

Наступним кроком є реалізація функціоналу для відображення списку транзакцій та управління ними. Для цього було розроблено компонент

TransactionList, який отримує дані про транзакції з Redux-сховища та відображає їх у вигляді списку з можливістю фільтрації, сортування та пошуку (рис. 3.5). Кожна транзакція відображається як окремий елемент з інформацією про суму, категорію, дату та тип, а також надається можливість переходу до редагування або видалення транзакції.

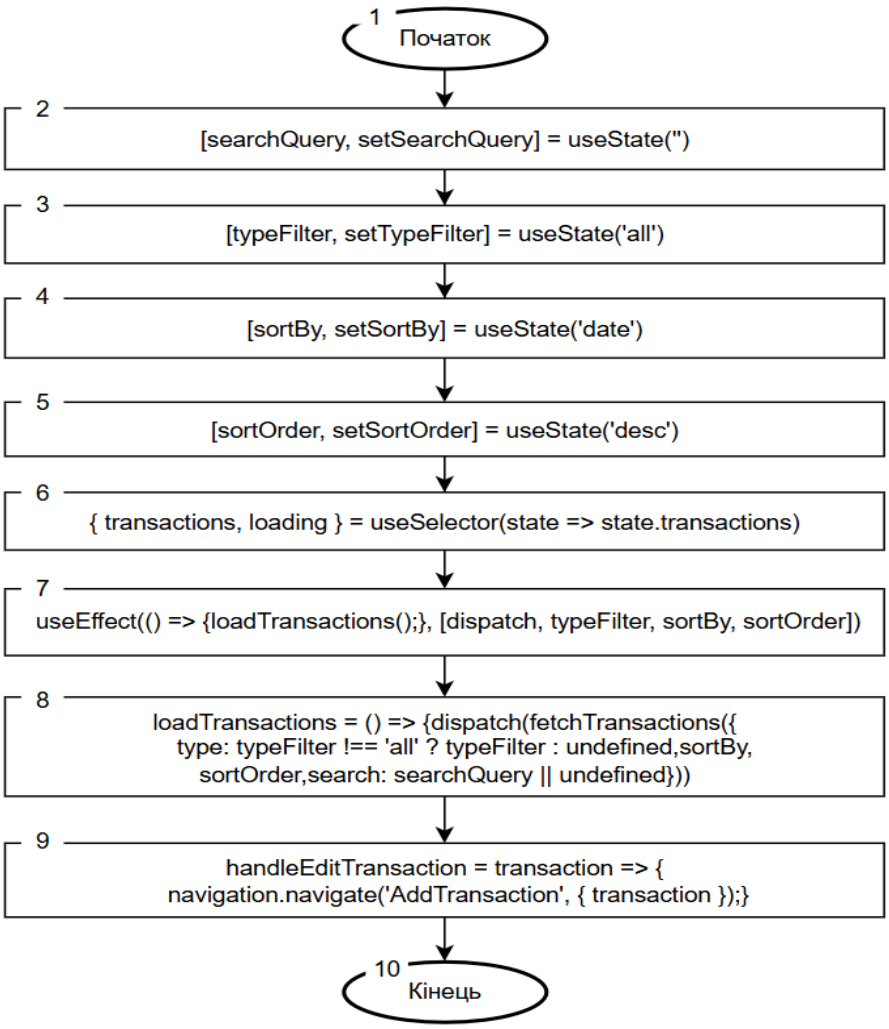


Рисунок 3.5 – Компонент списку транзакцій

Модуль обліку фінансових транзакцій є основним у застосунку, оскільки забезпечує основний функціонал для відстеження фінансових потоків користувача, дозволяє гнучко категоризувати доходи та витрати, що створює основу для подальшого аналізу фінансових даних та формування фінансових звітів, які допомагають користувачу краще розуміти власні фінансові патерни та приймати

обґрунтовані рішення щодо управління особистими фінансами.

3.5 Розробка інтерфейсу користувача

Головний екран застосунку відкриває користувачеві зведену інформацію про фінанси: у лівому верхньому блоці відображається поточний баланс із розбивкою на доходи й витрати за місяць. Праворуч — картка «Фінансове здоров’я», яка на початковому етапі показує знак питання і пропонує натиснути для розрахунку: це спонукає користувача до взаємодії. Нижче розташовано графік «Тенденції доходів і витрат» із відмітками по місяцях, що дозволяє швидко побачити динаміку за півроку. Інтуїтивна навігація знизу, у вигляді таббару, дає доступ до інших розділів. Інтерфейс головного екрану застосунку наведений на рисунку 3.6.

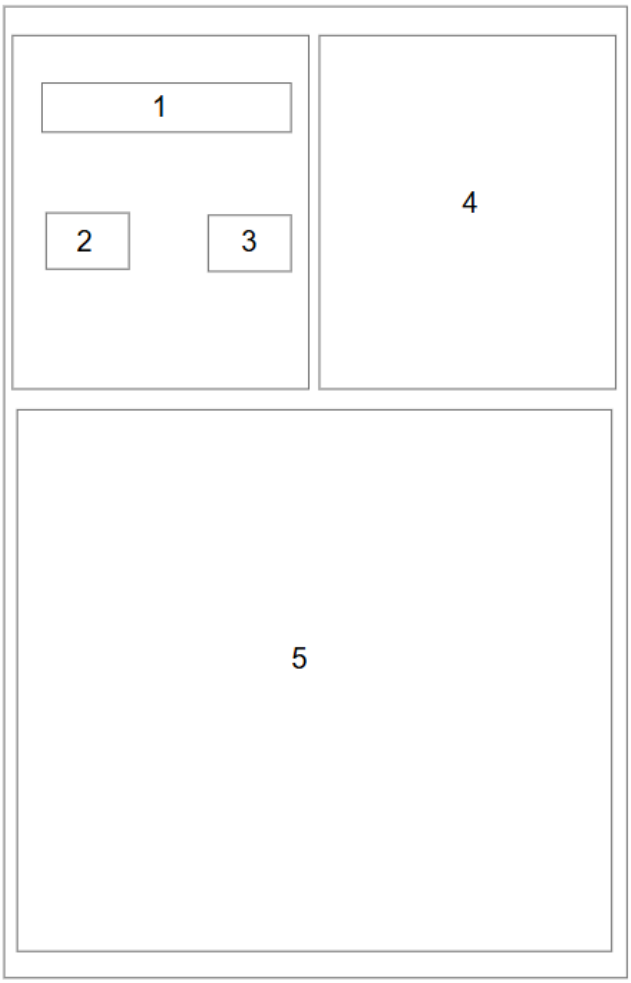


Рисунок 3.6 — Структурна схема головного екрану застосунку

Складові елементи головного екрану застосунку:

1. Поточний баланс.
2. Доходи.
3. Витрати.
4. Індекс фінансового здоров’я.
5. Графік тенденцій доходів та витрат.

На екрані додавання транзакції основна увага приділена формі: зверху — заголовок «Нова транзакція» та радіокнопки для вибору типу (дохід / витрата). Далі поле введення суми підкреслює важливість цього параметра. Секцію з категоріями реалізовано у вигляді об’єктних кнопок із закругленими рамками та чіткими підписами, що полегшує вибір. Структурна схема інтерфейсу екрану додавання транзакції наведена на рисунку 3.7.

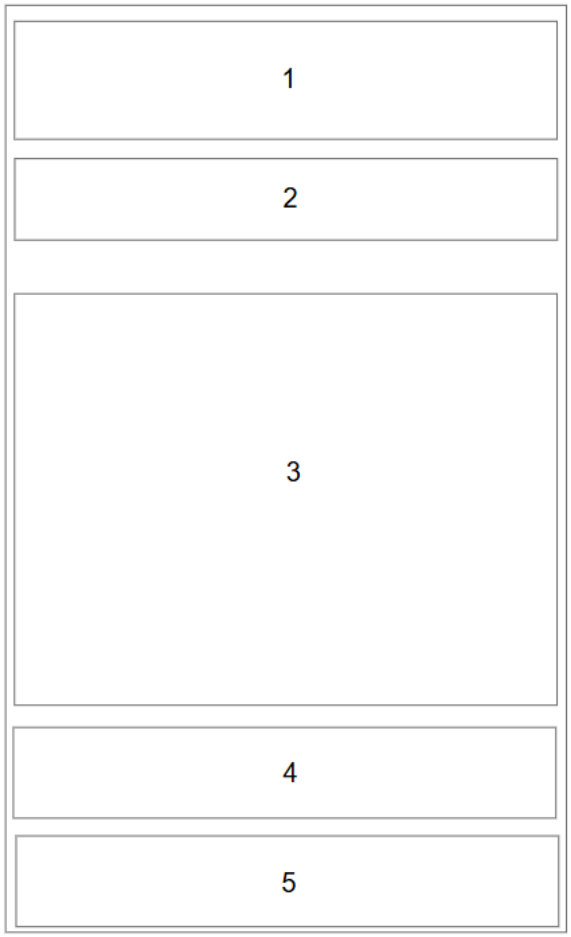


Рисунок 3.7 — Структурна схема інтерфейсу екрану додавання транзакцій

Складові елементи екрану додавання транзакцій:

1. Вибір типу транзакції (дохід чи витрата).

2. Сума транзакції.

3. Категорія.

4. Підкатегорія.

5. Опис.

На екрані створення бюджету розміщено список категорій із двома полями в кожному рядку: «Назва категорії» (текстове поле з автопідстановкою поточної категорії) і «Сума (грн)» (числове поле), а також іконка смітника для видалення непотрібних рядків. Кнопка «+ Додати категорію» під списком дозволяє вставити новий рядок для іншої статті витрат.

Структурна схема екрану створення бюджету наведена на рисунку 3.8.

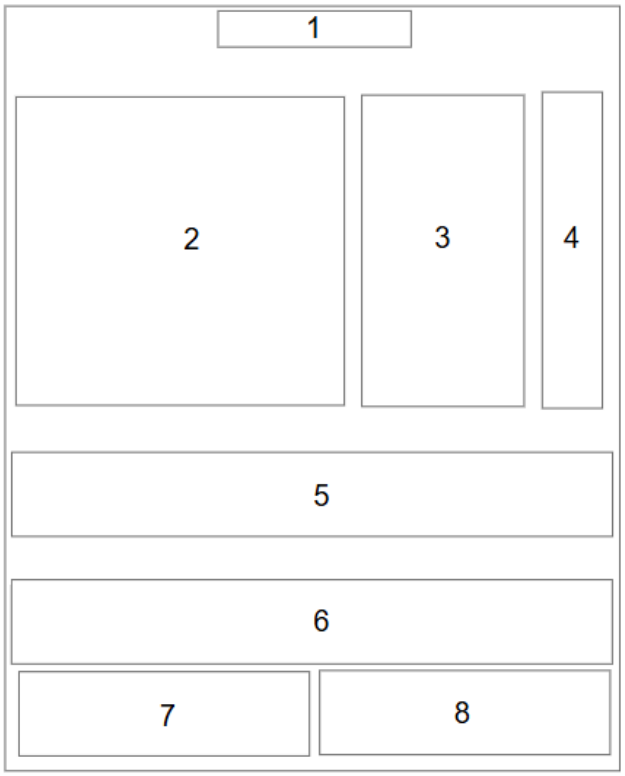


Рисунок 3.8 — Структурна схема інтерфейсу вікна додавання бюджету

Складові елементи вікна додавання бюджету:

1. Місяць, на який створюється бюджет.

2. Список категорій бюджету.

3. Бюджети по кожній категорії.

4. Кнопки «Видалити категорію».

5. Кнопка «Додати категорію».

6. Загальна сума.

7. Кнопка «Скасувати».

8. Кнопка «Зберегти».

Екран фінансового здоров’я користувача містить розрахований індекс, що визначає рівень фінансового здоров’я користувача, та показники, що на нього у впливають. У верхній частині екрану демонструється поточний індекс. Під ним — короткий опис метрик, які пояснюють цей індекс та впливають на нього. Для кожної з них відведено окремий рядок із графічною шкалою та значеннями, що дозволяє оцінити сильні та слабкі сторони власного фінансового стану.

Структурна схема інтерфейсу екрану фінансового здоров’я користувача наведена на рисунку 3.9.

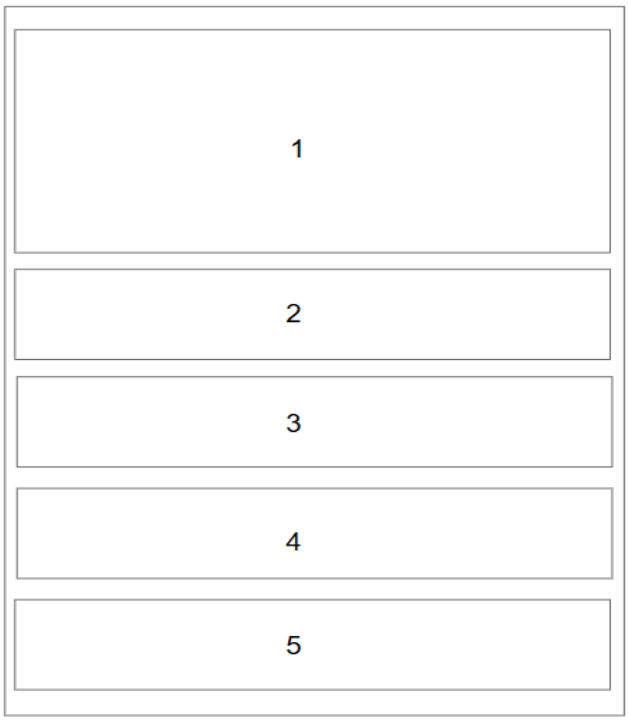


Рисунок 3.9 — Структурна схема інтерфейсу екрану фінансового здоров’я користувача

Складові елементи екрану фінансового здоров’я користувача:

1. Індекс фінансового здоров’я користувача.

2. Співвідношення доходів до витрат.

3. Подушка безпеки.

4. Диверсифікація доходів.

5. Прогрес фінансових цілей.

Екран реєстрації містить чотири поля вводу: «Ім’я», «Електронна пошта», «Пароль» та «Повторіть пароль». У полях паролю реалізовано іконку «око» для перемикання видимості тексту. Якщо виникає помилка (наприклад, невірний формат email або паролі не співпадають), під полями показується червоний текст з повідомленням «Невірний email або пароль». Внизу цих полів йде кнопка «Зареєструватися», нижче якого знаходиться кнопка авторизації для уже зареєстрованих користувачів.

Структурна схема інтерфейсу екрану реєстрації наведена на рисунку 3.10.

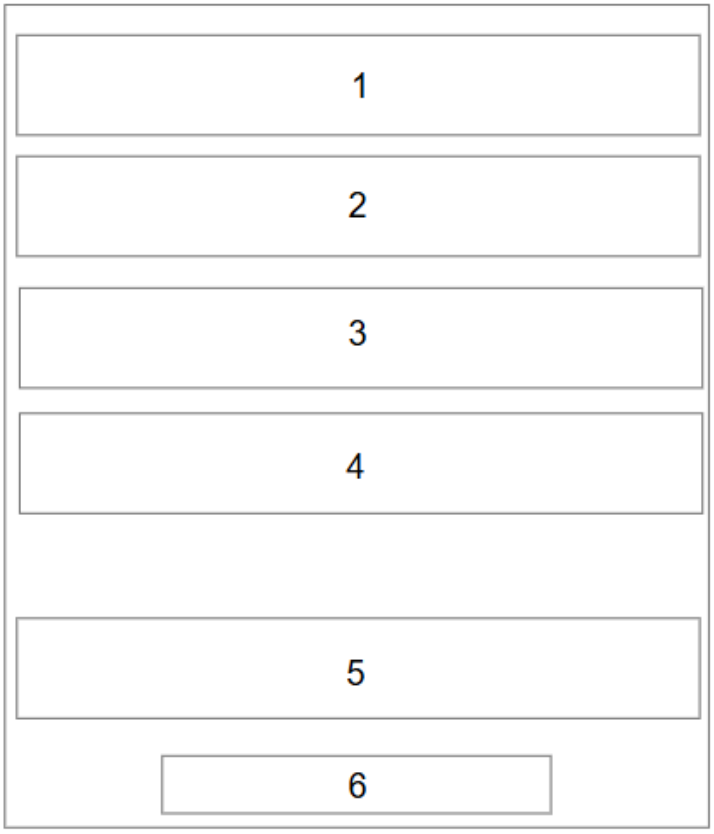


Рисунок 3.10 — Структурна схема інтерфейссу екрану реєстрації

Складові екрану реєстрації:

1. Ім’я користувача.

2. Електронна адреса.

3. Пароль.

4. Повтор пароля.

5. Кнопка «Зареєструватися».

6. Кнопка «Увійти».

Усі екрани відповідають основним завданням користувача на різних етапах взаємодії з додатком: від реєстрації та входу в обліковий запис до ведення й аналізу фінансових операцій. Форма реєстрації збирає базові дані (ім’я, email, пароль) і повідомляє про помилки при їх введенні, після чого користувач потрапляє на головний екран із загальним балансом та швидким доступом до розрахунку індексу фінансового здоров’я. Екран додавання транзакції дозволяє ввести суму, вибрати категорію й за потреби додати підкатегорію та опис, що робить процес обліку операцій простим і послідовним.

Розділ бюджету дозволяє задати план витрат за категоріями з автоматичним підрахунком загальної суми, а сторінка фінансового здоров’я надає детальні показники та рекомендації на основі ключових метрик (співвідношення доходів і витрат, «подушка безпеки» тощо). Кожен екран сфокусований на конкретній дії — створенні облікових даних, введенні операцій, формуванні бюджету чи аналізі фінансового стану.

3.6 Висновки

Шляхом реалізації розроблених алгоритмів та інтерфейсу користувача було вирішено задачу з розробки модулів застосунку для ведення особистих фінансів. Описано основні функції, що використовуються при роботі цих модулів. Реалізовані модулі дозволяють керувати інформацією про транзакції, вести особистий бюджет, отримувати розрахунок індексу «фінансового здоров’я» на основі введених даних про особисті фінанси та їх використання.

4 ТЕСТУВАННЯ ЗАСТОСУНКУ

4.1 Аналіз методів тестування

Тестування застосунків для обліку особистих фінансів є важливим процесом, оскільки ці програми працюють із чутливими фінансовими даними користувачів. Якісне тестування забезпечує точність розрахунків, безпеку даних користувача. Застосунки такого типу мають специфічні вимоги до функціональності, включаючи відстеження витрат та доходів, категоризацію транзакцій, формування звітів та візуалізацію фінансової інформації.

Функціональне тестування є основним підходом при тестуванні застосунків для ведення особистих фінансів [26]. Воно передбачає перевірку відповідності функцій програми заявленим вимогам, оскільки важливо перевірити коректність обчислень балансу, правильність категоризації транзакцій, точність генерування звітів та функціонування нагадувань про платежі. Функціональне тестування дозволяє виявити помилки в бізнес-логіці, які можуть призвести до некоректних фінансових даних.

Тестування інтерфейсу користувача спрямоване на перевірку зручності використання та візуальної цілісності застосунку [27]. Для фінансового застосунку особливо важливо, щоб користувач міг легко додавати транзакції, переглядати звіти та налаштовувати бюджети. UI-тестування допомагає виявити проблеми з навігацією, невідповідності в дизайні та інші фактори, які можуть ускладнити використання програми.

Ручне тестування є ефективним методом ручного тестування, який дозволяє одночасно досліджувати систему, проектувати тести та виконувати їх. Цей підхід дозволяє тестувальнику виявити неочевидні проблеми, які можуть виникнути при нестандартних сценаріях використання. Для застосунку ведення особистих фінансів таке тестування допомагає імітувати реальну поведінку користувача та виявити проблеми, які могли бути пропущені при формальному тестуванні.

На основі аналізу різних методів ручного тестування, було обрано метод ручного тестування. Цей метод дозволяє гнучко реагувати на знайдені проблеми,

імітувати реальні сценарії використання та глибоко досліджувати взаємодію між різними функціями застосунку. Завдяки своїй адаптивності, ручне тестування дає можливість виявити критичні помилки в бізнес-логіці, проблеми з інтерфейсом та базові проблеми безпеки, що є ключовими аспектами для фінансового застосунку.

4.2 Тестування застосунку для ведення особистих фінансів

Тестування застосунку розпочинається із вікна реєстрації, оскільки необхідно створити профіль користувача, за допомогою якого можна буде отримувати доступ до даних про фінанси з різних пристроїв. Створимо новий аккаунт, заповнивши відповідну форму, як на рисунку 4.1.

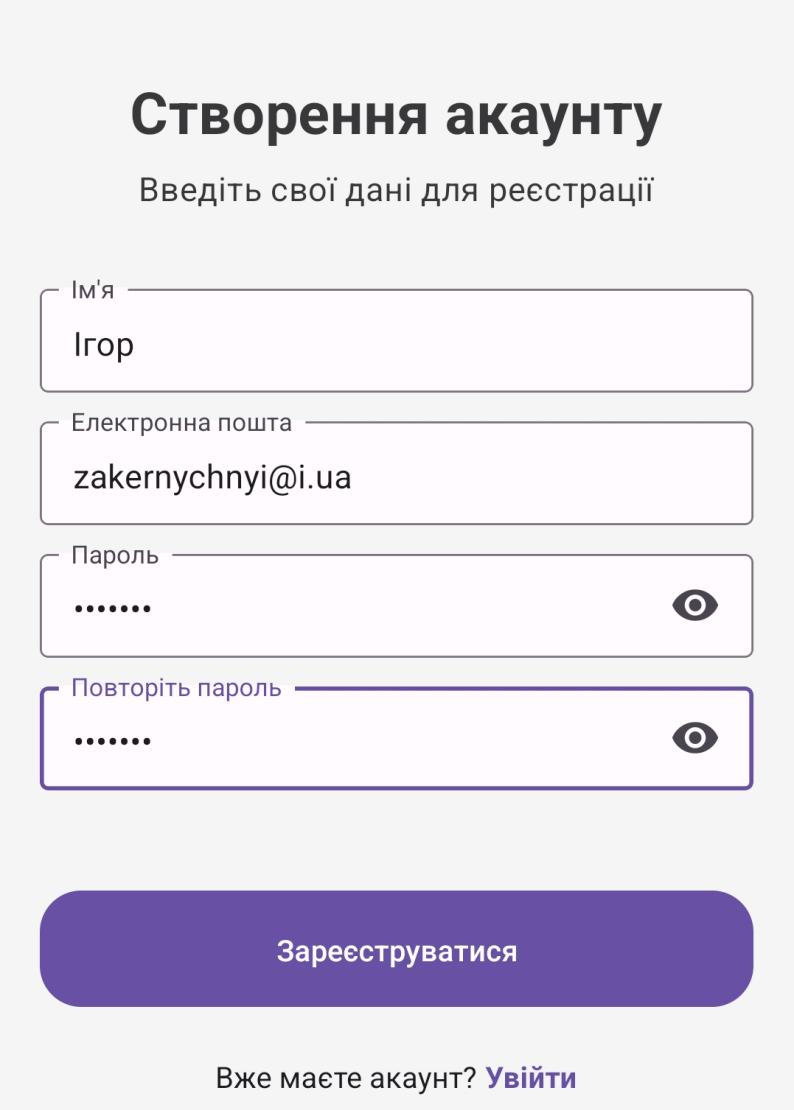


Рисунок 4.1 — Створення аккаунту користувача

Після цього відбувається перехід на головний екран застосунку (рисунок 4.2). Цей екран містить інформацію про поточний баланс, фінансове здоров’я, тенденції доходів і витрат у вигляді графіків, розподіл витрат.

Рисунок 4.2 — Початковий стан головного екрану застосунку

Додамо нову транзакцію. Для цього на головному екрані оберемо відповідну

опцію, після чого перейдемо до вікна додавання транзакцій. Для цього необхідно обрати тип транзакції (дохід чи витрата), суму транзакції, обрати категорію витрат. Також за бажанням можна додати інформацію про підкатегорію витрат, та додати опис. Необхідно також вказати дату додавання транзакції. Також можна вказати, регулярна транзакція чи одноразова. Екран додавання транзакції наведеноа на рисунку 4.3.

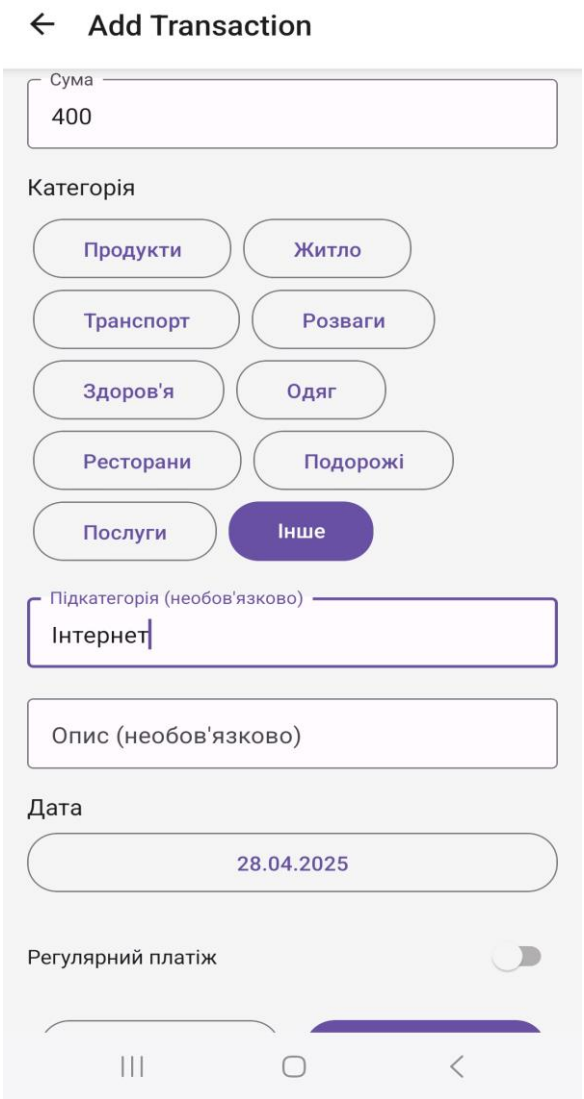


Рисунок 4.3 — Додавання транзакції

Після додавання транзакції, вона відображається у списку транзакцій (рисунок 4.4).

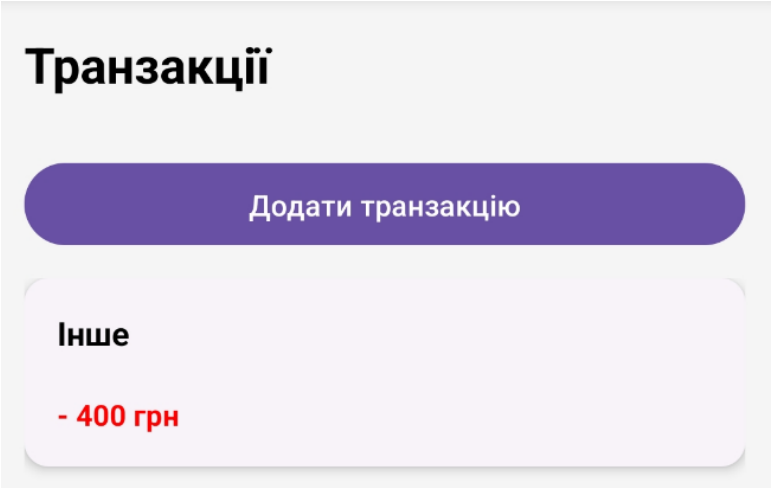


Рисунок 4.4 — Додана транзакція у списку транзакцій

Додамо ще кілька транзакцій аналогічним чином, після чого перейдемо назад на головний екран застосунку. Можемо побачити зміни (рисунок 4.5).

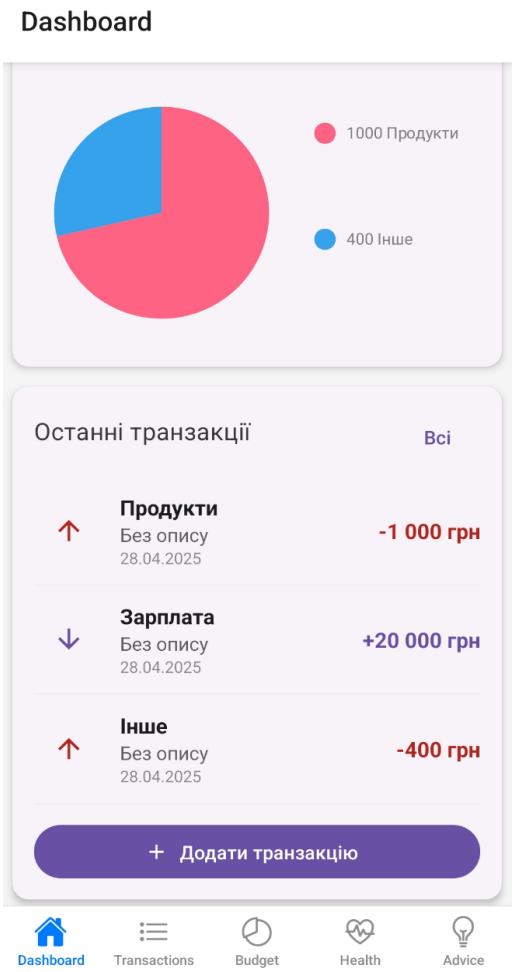


Рисунок 4.5 — Останні транзакції у головному екрані застосунку

У головному екрані застосунку відображається кругова діаграма розподілу

витрат за категоріями. Крім цього формується графік динаміки витрат та змінюється інформація про баланс коштів (рисунок 4.6).

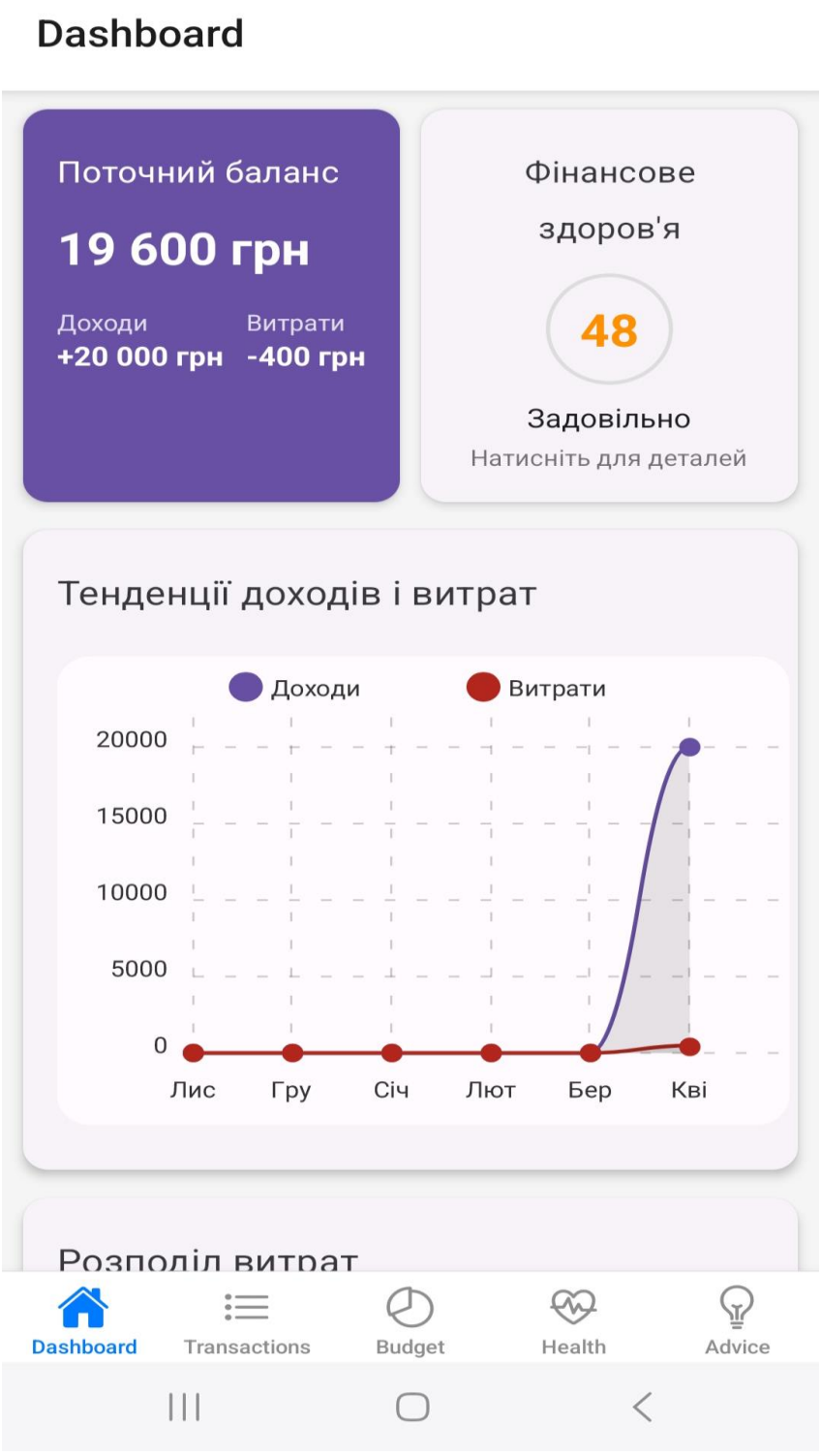


Рисунок 4.6 — Зміна стану головного екрану застосунку

На рисунку 4.7 наведено екран фінансового здоров’я. Він містить власне

індекс фінансового здоров’я, історію зміни, та ключові показники, що враховуються при розрахунку цього індексу — диверсифікація доходів, прогрес фінансових цілей, подушка безпеки, співвідношення доходів до витрат.

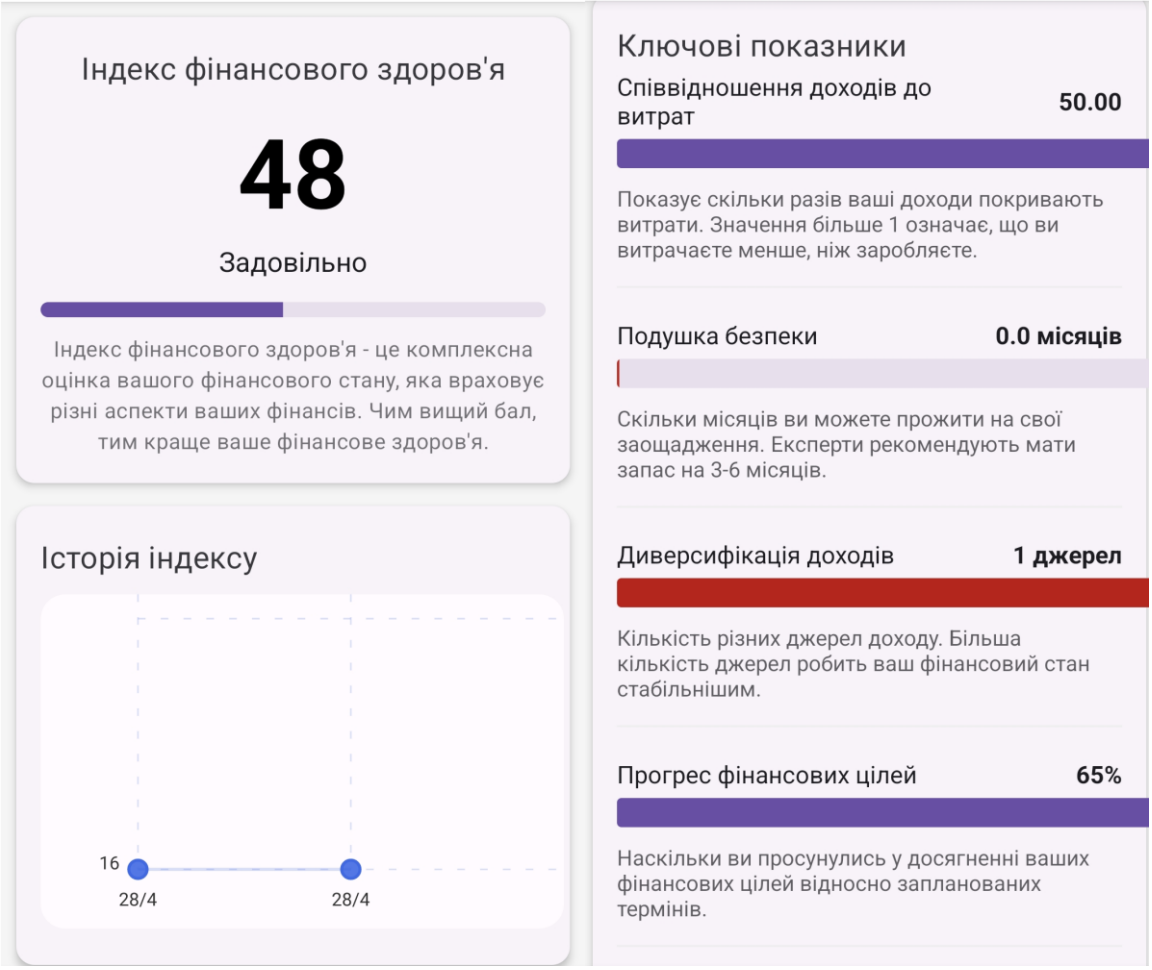


Рисунок 4.7 — Екран індексу фінансового здоров’я

На рисунку 4.8 наведено екран створення бюджету. Він дозволяє планувати бюджет на місяць, розподіляючи майбутні витрати по категоріям, що потім буде використовуватися для відслідкування виконання бюджету та виявлення можливих перевищень, що буде впливати на індекс фінансового здоров’я.

Користувач додає або редагує категорії витрат: для кожної вказує назву та відповідну суму в національній валюті. Система автоматично підраховує загальну суму, оновлюючи її у режимі реального часу. По-друге, доступна можливість видалити будь-яку категорію одним дотиком, що дає гнучкість під час коригування бюджету. Після завершення налаштування користувач може зберегти бюджет або

скасувати внесені зміни за допомогою явних кнопок у нижній частині екрана.

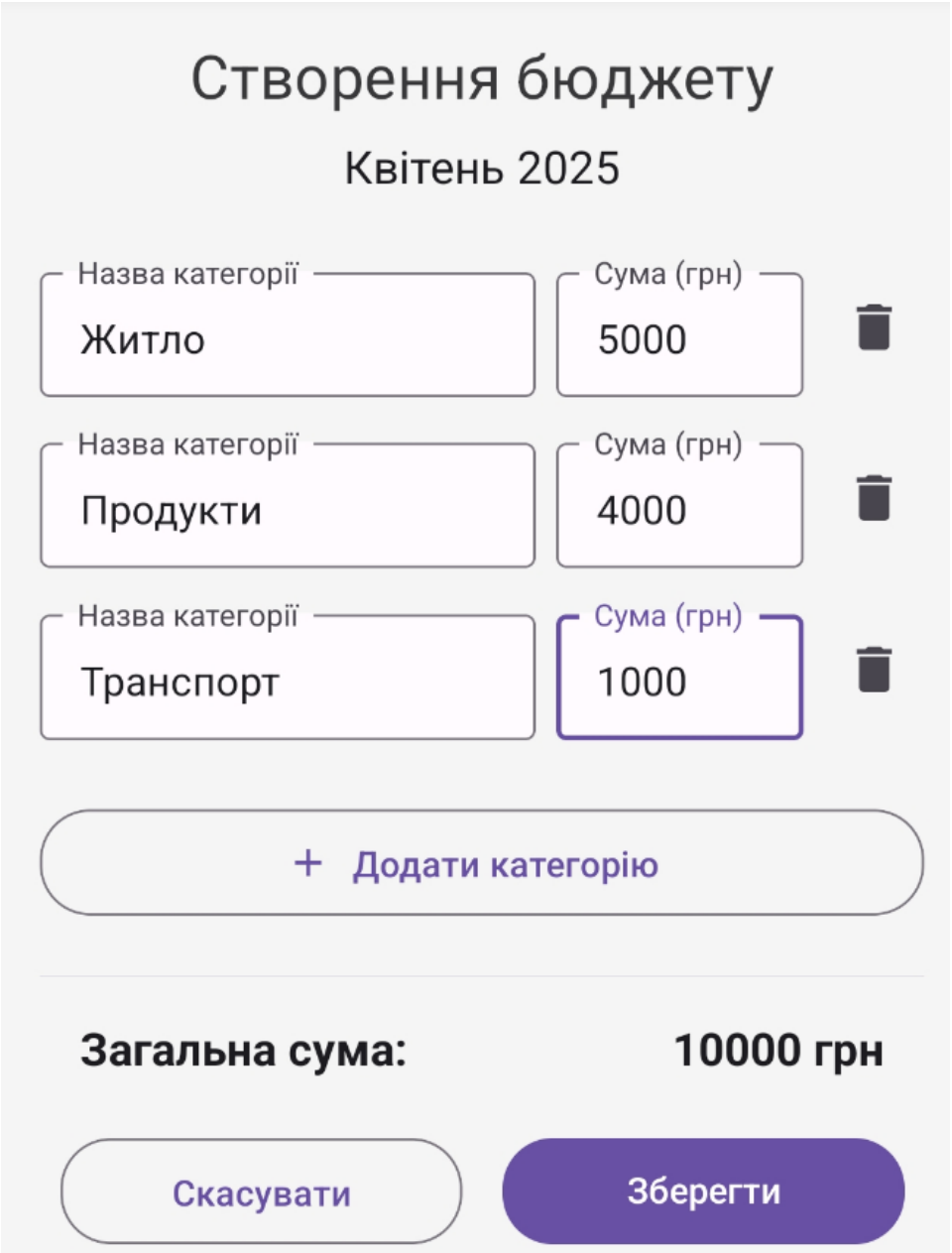


Рисунок 4.8 — Екран створення бюджету

На рисунку 4.9 наведено діалогове вікно додавання категорії бюджету. Воно містить 2 поля для введення назви та планової суми витрат на місяць. Додана категорія відображається поряд з іншими у плані бюджету.

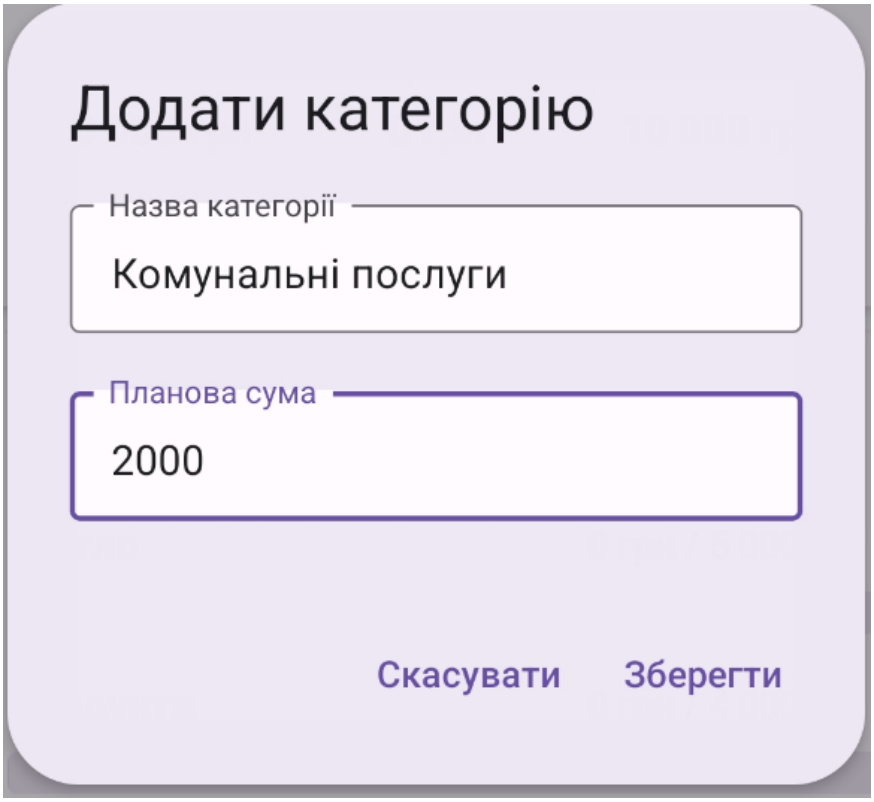


Рисунок 4.9 — Діалогове вікно додавання категорії

Таким чином, було проведено тестування застосунку для ведення особистих фінансів. Продемонстровано розроблений функціонал, перевірено його роботу. Розроблений застосунок дозволяє вести облік витрат та доходів, отримувати графіки для візуалізації даних, отримувати індекс фінансового здоров’я, отже виконує поставлені на початку розробки завдання.

4.3 Висновки

Шляхом використання ручного методу тестування та демонстрації можливостей застосунку було вирішено завдання з тестування розробленого застосунку. Розроблений застосунок дозволяє вести облік витрат та доходів, отримувати графіки для візуалізації даних, отримувати індекс фінансового здоров’я.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було розроблено мобільну систему для ведення особистих фінансів. Для розробки використовувалося середовище розробки Visual Studio Code. Бакалаврську кваліфікаційну роботу реалізовано згідно методичних вказівок [28].

Проаналізовано стан питання розробки застосунків для ведення особистих фінансів. Було визначено, що на сьогодні існує значна кількість застосунків для управління особистих фінансів, однак більшість із них мають обмеження щодо функціональності, персоналізації та прогнозування фінансової поведінки користувачів, через що актуальною є розробка застосунку для управління особистими фінансами;

Проведено дослідження існуючих аналогів, а саме: Mint, Personal Capital, You Need A Budget, проведено порівняльний аналіз із власною розробкою шляхом побудови порівняльної таблиці їх характеристик, в результаті чого було доведено актуальність власної розробки, та її переваги над аналогами, до яких належить: наявність персоналізованих рекомендацій та розрахунок індексу фінансового стану користувача.

Шляхом побудови діаграми компонентів застосунку було вирішено задачу з розробки моделі застосунку для ведення особистих фінансів, яка складається з взаємопов'язаних компонентів, що забезпечують увесь функціонал для додавання та управління інформацією про особисті фінанси;

Удосконалено метод «фінансового здоров’я» користувача, який враховує рівень диверсифікації джерел доходу, запас коштів та виконання поставлених фінансових цілей, на основі чого розраховує індекс, що визначає рівень фінансової стабільності користувача.

Було побудовано діаграми класів та алгоритмів роботи застосунку, чим було вирішено завдання з розробки діаграм та алгоритмів роботи застосунку. Розроблені діаграми та алгоритми роботи дозволяють додавати транзакції, керувати транзакціями, управляти бюджетом, розраховувати індекс «фінансового здоров’я».

Шляхом реалізації розроблених алгоритмів та інтерфейсу користувача було вирішено задачу з розробки модулів застосунку для ведення особистих фінансів. Описано основні функції, що використовуються при роботі цих модулів. Реалізовані модулі дозволяють керувати інформацією про транзакції, вести особистий бюджет, отримувати розрахунок індексу «фінансового здоров'я» на основі введених даних про особисті фінанси та їх використання.

Шляхом використання ручного методу тестування та демонстрації можливостей застосунку було вирішено завдання з тестування розробленого застосунку. Розроблений застосунок дозволяє вести облік витрат та доходів, отримувати графіки для візуалізації даних, отримувати індекс фінансового здоров'я.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Національний банк України. Фінанси у смартфоні – мобільні застосунки для планування бюджету : веб-сайт. URL: https://harazd.bank.gov.ua/article/finansove-planuvanna/finansovi-cili/finansi-u-smartfoni-mobilni-zastosunki-dla-planuvanna-budzetu (дата звернення: 25.03.2025).

2. Bazilik Media. 10 застосунків і програм для роботи з грошима: веб-сайт. URL: https://bazilik.media/10-zastosunkiv-i-prohram-dlia-roboty-z-hroshyma/ (дата звернення: 25.03.2025).

3. WoMo.ua. Топ-9 застосунків для контролю особистих фінансів: веб-сайт. URL: https://womo.ua/top-10-dodatkiv-dlya-kontrolyu-osobistih-finansiv/ (дата звернення: 26.03.2025).

4. ProIT. Топ-5 найкращих застосунків для особистих фінансів: веб-сайт. URL: https://proit.ua/top-5-naikrashchikh-zastosunkiv-dlia-osobistikh-finansiv/ (дата звернення: 26.03.2025).

5. Laba.ua. 6 додатків, які допоможуть дотягти до зарплати та навіть накопичити: веб-сайт. URL: https://laba.ua/blog/3181-6-prilozheniy-kotorye-pomogut-dotyanut-do-zarplaty (дата звернення: 27.03.2025).

6. Економічна правда. Мобільні застосунки для планування бюджету та контролю витрат: веб-сайт. URL: https://epravda.com.ua/publications/2023/02/22/697326/ (дата звернення: 28.03.2025).

7. Marketing-UA. Плануємо бюджет: ТОП найкращих мобільних додатків для особистих фінансів: веб-сайт. URL: https://www.marketing-ua.com/article/planuyemo-byudzhet-top-najkrashhih-mobilnih-dodatkiv-dlya-osobistih-finansiv/ (дата звернення: 29.03.2025).

8. КПІ ім. Ігоря Сікорського. Розробка мобільного застосунку для ведення особистих фінансів: веб-сайт. URL: https://ela.kpi.ua/bitstream/123456789/28727/1/Rusanov_bakalavr.docx (дата звернення: 29.03.2025).

9. Закерничний І.О. Особливості мобільного застосунку для ведення

особистих фінансів/ І.О. Закерничний, Л.Б. Ліщинська // Матеріали Міжнародної науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми,перспективи (МН-2025)». – Вінниця: ВНТУ, 2025. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/viewFile/25194/20784

10. eVNUIR. Дослідження мобільних застосунків для менеджменту фінансів: веб-сайт. URL: https://www.evnuir.vnu.edu.ua/bitstream/123456789/24403/1/rasik_2024.pdf (дата звернення: 30.03.2025).

11. Рубрика. П'ять українських додатків для контролю витрат.: веб-сайт. URL: https://rubryka.com/article/dlya-kontrolyu-vytrat/ (дата звернення: 30.03.2025).

12. Мінфін. 10 зручних додатків для контролю особистого бюджету: веб-сайт. URL: https://minfin.com.ua/ua/2019/07/24/38514719/ (дата звернення: 31.03.2025).

13. Banker.ua. Фінансова грамотність у вашій кишені: огляд мобільних додатків для управління особистими фінансами: веб-сайт. URL: https://banker.ua/uk/projects/oglyad-dodatkiv-dlya-upravlinnya-osobistimi-finansami/ (дата звернення: 01.04.2025).

14. AVADA-MEDIA. Розробка мобільних додатків на IOS та Android на замовлення: веб-сайт. URL: https://avada-media.ua/services/razrabotka-mobilnyh-prilozhenij/ (дата звернення: 02.04.2025).

15. Speka. Топ найкращих застосунків для особистих фінансів та обліку грошей: веб-сайт. URL: https://speka.media/top-naikrashhix-zastosunkiv-dlya-osobistix-finansiv-ta-obliku-grosei-982k86 (дата звернення: 03.04.2025).

16. Mint: веб-сайт. URL: https://mint.intuit.com/ (дата звернення: 04.04.2025).

17. You Need A Budget: веб-сайт. URL: https://www.ynab.com/ (дата звернення: 04.04.2025).

18. Personal Capital: веб-сайт. URL: https://home.personalcapital.com/ (дата звернення: 05.04.2025).

19. What is a native app?: веб-сайт. URL: https://www.techtarget.com/searchsoftwarequality/definition/native-application-native-app (дата звернення: 06.04.2025).

20. React Native: веб-сайт. URL: https://reactnative.dev/ (дата звернення: 06.04.2025).

21. Apache Cordova: веб-сайт. URL: https://cordova.apache.org/ (дата звернення: 06.04.2025).

22. TORNO A., WERTH O., NICKERSON R.C., BREITNER M.H., MUNTERMANN J. More than Mobile Banking – A Taxonomy-based Analysis of Mobile Personal Finance Applications : матеріали 25-ї Міжнародної конференції з інформаційних систем Тихоокеанської Азії (Pacific Asia Conference on Information Systems, PACIS ’21) : Virtual Event / Dubai, UAE, 12–14 липня 2021. С. 179.

23. Addy Osmani. Learning JavaScript Design Patterns: A JavaScript and React Developer's Guide. Farnham: O’Reilly Media, 2023. 296с.

24. Pierre-Olivier Laurence. Programming Android with Kotlin. Achieving Structured Concurrency with Coroutines. Farnham: O’Reilly Media, 2021. 336с.

25. П. Беррі. Head First. Python. Farnham: O’Reilly Media, 2021. 624с.

26. Gayathri Mohan. Full Stack Testing: A Practical Guide for Delivering High Quality Software. Farnham: O’Reilly Media, 2022. 405с.

27. Dario Kondratiuk. UI Testing with Puppeteer: Implement end-to-end testing and browser automation using JavaScript and Node.js. Birmingham: Packt Publishing, 2021. 316с.

28. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. О. Н. Романюк, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 52с.

ДОДАТКИ

Додаток А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«25» березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка кросплатформного мобільного інтерфейсу для управління особистими фінансами»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
д.т.н., проф.каф. ПЗ Ліщинська Л.Б.
«24» березня 2025 р.

Виконав:
студент гр. 2ПІ-21б Закерничний І.О.
«24» березня 2025 р.

м. Вінниця – 2025 рік

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка кросплатформного мобільного інтерфейсу для управління особистими фінансами».

Галузь застосування – мобільні технології.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від 20 березня 2025 року про закріплення тем БКР.

3. Мета та призначення розробки.

Метою роботи є покращення процесу ведення особистих фінансів шляхом розробки спеціалізованого програмного забезпечення, що дозволяє керувати інформацією про доходи та витрати, формувати бюджет та відстежувати його виконання.

4. Вихідні дані для проведення НДР

1. Фінанси у смартфоні – мобільні застосунки для планування бюджету. [Електронний ресурс] – Режим доступу: https://harazd.bank.gov.ua/article/finansove-planuvanna/finansovi-cili/finansi-u-smartfoni-mobilni-zastosunki-dla-planuvanna-budzetu

2. React Native [Електронний ресурс] – Режим доступу: https://reactnative.dev/

3. Addy Osmani. Learning JavaScript Design Patterns: A JavaScript and React Developer's Guide. Farnham: O’Reilly Media, 2023. 296c.

4. Gayathri Mohan. Full Stack Testing: A Practical Guide for Delivering High Quality Software. Farnham: O’Reilly Media, 2022. 405c.

5. Технічні вимоги

Вхідні дані — дані користувача, дані транзакцій, план бюджету.

Вихідні дані — графіки, індекс фінансового здоров’я, записи про транзакції.

6. Конструктивні вимоги.

Інтерфейс програми поовинен відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

–пояснювальна записка до бакалаврської кваліфікаційної роботи;

–технічне завдання;

–лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану розвитку застосунків для ведення особистих фінансів	25.03.2025 - 06.04.2025
2	Розробка моделі та алгоритмів застосунку	07.04.2025 - 20.04.2025
3	Варіантний аналіз та вибір мови програмування	21.04.2025 - 27.04.2025
4	Розробка застосунку	28.04.2025 - 18.05.2025
5	Тестування застосунку	19.05.2025 - 25.05.2025
6	Оформлення матеріалів до захисту БКР	26.05.2025 - 30.05.2025

10. Порядок контролю та прийняття.

Усі етапи бакалаврської кваліфікаційної роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

Додаток Б

Протокол перевірки на наявність текстових запозичень

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Розробка кросплатформного мобільного інтерфейсу для управління особистими фінансами»
Тип роботи: БКР
Підрозділ: кафедра програмного забезпечення, ФІТКІ

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 22 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

□У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

□У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Ліщинська Л.Б. д.т.н., проф.каф.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Закерничний І.О.
(підпис) (прізвище, ініціали)

Додаток В

Лістинг програми

```
const Stack = createStackNavigator();
const Tab = createBottomTabNavigator();

function MainTabs() {
 return (
  <Tab.Navigator
   screenOptions={({ route }) => ({
    tabBarIcon: ({ focused, color, size }) => {
     let iconName;

     if (route.name === 'Dashboard') {
      iconName = focused ? 'home' : 'home-outline';
     } else if (route.name === 'Transactions') {
      iconName = focused ? 'list' : 'list-outline';
     } else if (route.name === 'Budget') {
      iconName = focused ? 'pie-chart' : 'pie-chart-outline';
     } else if (route.name === 'Health') {
      iconName = focused ? 'fitness' : 'fitness-outline';
     } else if (route.name === 'Advice') {
      iconName = focused ? 'bulb' : 'bulb-outline';
     }

     return <Ionicons name={iconName} size={size} color={color} />;
    },
   })}
  >
   <Tab.Screen name="Dashboard" component={Dashboard} />
```

```
      <Tab.Screen name="Transactions" component={Transactions} />
      <Tab.Screen name="Budget" component={Budget} />
      <Tab.Screen name="Health" component={FinancialHealth} />
      <Tab.Screen name="Advice" component={Advice} />
    </Tab.Navigator>
  );
}

function AppNavigator() {
  const [isLoading, setIsLoading] = useState(true);
  const { isAuthenticated } = useSelector(state => state.auth);
  const dispatch = useDispatch();

  useEffect(() => {
    // Перевірка автентифікації при запуску
    const checkAuthentication = async () => {
      await dispatch(checkAuth());
      setIsLoading(false);
    };

    checkAuthentication();
  }, [dispatch]);

  if (isLoading) {
    return (
      <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
        <ActivityIndicator size="large" />
      </View>
    );
  }
```

```
return (
  <NavigationContainer>
    <Stack.Navigator>
      {isAuthenticated ? (
       <>
         <Stack.Screen
          name="Main"
          component={MainTabs}
          options={{ headerShown: false }}
         />
         <Stack.Screen
          name="AddTransaction"
          component={AddTransaction}
          options={{ title: 'Add Transaction' }}
         />
         <Stack.Screen
          name="Settings"
          component={Settings}
          options={{ title: 'Settings' }}
         />
          <Stack.Screen
          name="CreateBudget"
          component={CreateBudget}
          options={{ title: 'Create Budget' }}
/>
       </>
      ) : (
       <>
         <Stack.Screen
```

```
            name="Login"
            component={Login}
            options={{ headerShown: false }}
           />
           <Stack.Screen
            name="Register"
            component={Register}
            options={{ headerShown: false }}
           />
          </>
        )}
      </Stack.Navigator>
    </NavigationContainer>
  );
}

export default function App() {
  return (
    <Provider store={store}>
      <PaperProvider>
        <AppNavigator />
      </PaperProvider>
    </Provider>
  );
}

// src/redux/thunks/adviceThunks.js
import { createAsyncThunk } from '@reduxjs/toolkit';
import { setLoading, setAdviceData, setError } from '../slices/adviceSlice';
import AsyncStorage from '@react-native-async-storage/async-storage';
```

```
// Отримання персоналізованих порад
export const fetchAdvice = createAsyncThunk(
  'advice/fetchAdvice',
  async (_, { dispatch, getState, rejectWithValue }) => {
    try {
      dispatch(setLoading(true));

      const { auth, health, transactions } = getState();
      if (!auth.user) {
        throw new Error('Користувач не автентифікований');
      }

      // Перевірка наявності даних про фінансове здоров'я
      if (!health.healthData) {
        throw new Error('Дані про фінансове здоров\'я відсутні');
      }

      // Отримання категорій витрат
      const topSpendingCategories =
        transactions.dashboardData?.topExpenseCategories ||
        [
          { name: 'Житло', percentage: '36.6' },
          { name: 'Продукти', percentage: '26.8' },
          { name: 'Транспорт', percentage: '18.3' }
        ];

      // Генерація порад на основі показників фінансового здоров'я
      let personalizedAdvice = '';
```

```
// 1. Поради щодо співвідношення доходів до витрат
if (health.healthData.incomeToExpenseRatio.score < 60) {
 personalizedAdvice += `1. Ваше співвідношення доходів до витрат (${health.healthData.incomeToExpenseRatio.value.toFixed(2)}) є недостатнім. Рекомендується збільшити доходи або зменшити витрати, щоб досягти співвідношення не менше 1.5.\n\n`;
}

// 2. Поради щодо подушки безпеки
if (health.healthData.emergencyFund.score < 60) {
 personalizedAdvice += `2. Ваша подушка безпеки (${health.healthData.emergencyFund.value.toFixed(1)} місяців) недостатня для фінансової безпеки. Намагайтеся накопичити суму, що покриває витрати на 3-6 місяців.\n\n`;
}

// 3. Поради щодо диверсифікації доходів
if (health.healthData.incomeSourceDiversity.score < 60) {
 personalizedAdvice += `3. У вас недостатньо диверсифіковані джерела доходу (${health.healthData.incomeSourceDiversity.value}). Розгляньте можливості додаткових джерел, таких як інвестиції, підробіток або пасивний дохід.\n\n`;
}

// 4. Поради щодо категорій витрат
if (topSpendingCategories.length > 0) {
 const topCategory = topSpendingCategories[0];
 personalizedAdvice += `4. Найбільша категорія ваших витрат - "${topCategory.name}" (${topCategory.percentage}%). Розгляньте можливості оптимізації витрат у цій категорії.\n\n`;
}
```

```
// Якщо всі показники добрі, даємо загальні рекомендації
if (personalizedAdvice === '') {
  personalizedAdvice = `Ваші фінансові показники виглядають добре! Продовжуйте дотримуватися поточної фінансової стратегії. Ось кілька загальних рекомендацій для подальшого вдосконалення:

1. Розгляньте можливості інвестування для зростання вашого капіталу.

2. Регулярно переглядайте свій бюджет та коригуйте його відповідно до змін у вашому житті.

3. Зверніть увагу на довгострокові фінансові цілі, такі як пенсійні накопичення.`;
}

// Формування даних порад
const adviceData = {
  personalizedAdvice,
  topSpendingCategories,
  financialHealth: health.healthData
};

// Збереження порад
await AsyncStorage.setItem(`last_advice_${auth.user.id}`, JSON.stringify({
  ...adviceData,
  date: new Date().toISOString()
}));

// Оновлення стану
```

```
            dispatch(setAdviceData(adviceData));
            dispatch(setLoading(false));

            return adviceData;
          } catch (error) {
            const errorMessage = error.message || 'Помилка при отриманні персоналізованих порад';
            dispatch(setError(errorMessage));
            return rejectWithValue(errorMessage);
          }
        }
      );

      // src/redux/thunks/budgetThunks.js
      import { createAsyncThunk } from '@reduxjs/toolkit';
      import { setLoading, setBudget, setBudgetProgress, setError } from '../slices/budgetSlice';
      import AsyncStorage from '@react-native-async-storage/async-storage';

      // Отримання бюджетів користувача з AsyncStorage
      const getUserBudgets = async (userId) => {
        try {
          const budgetsJSON = await AsyncStorage.getItem(`budgets_${userId}`);
          return budgetsJSON ? JSON.parse(budgetsJSON) : [];
        } catch (error) {
          console.error('Помилка отримання бюджетів:', error);
          return [];
        }
      };
```

```
// Створення нового бюджету
export const createBudget = createAsyncThunk(
  'budgets/createBudget',
  async (budgetData, { dispatch, getState, rejectWithValue }) => {
    try {
      dispatch(setLoading(true));

      const { auth } = getState();
      if (!auth.user) {
        throw new Error('Користувач не автентифікований');
      }

      // Отримання існуючих бюджетів
      const budgets = await getUserBudgets(auth.user.id);

      // Перевірка чи бюджет на цей місяць уже існує
      const existingBudget = budgets.find(
        b => b.month === budgetData.month && b.year === budgetData.year
      );

      if (existingBudget) {
        dispatch(setLoading(false));
        return rejectWithValue('Бюджет на цей місяць уже існує');
      }

      // Створення нового бюджету
      const newBudget = {
        id: Date.now().toString(),
        userId: auth.user.id,
        ...budgetData,
```

```
        createdAt: new Date().toISOString()
      };

      // Збереження в списку бюджетів
      budgets.push(newBudget);
      await AsyncStorage.setItem(`budgets_${auth.user.id}`,
JSON.stringify(budgets));

      // Оновлення стану
      dispatch(setBudget(newBudget));
      dispatch(setLoading(false));

      return newBudget;
    } catch (error) {
      console.error('Помилка створення бюджету:', error);
      dispatch(setError(error.message || 'Помилка при створенні бюджету'));
      dispatch(setLoading(false));
      return rejectWithValue(error.message || 'Помилка при створенні бюджету');
    }
  }
);

// Оновлення існуючого бюджету
export const updateBudget = createAsyncThunk(
  'budgets/updateBudget',
  async (budgetData, { dispatch, getState, rejectWithValue }) => {
    try {
      dispatch(setLoading(true));

      const { auth } = getState();
```

```
        if (!auth.user) {
          throw new Error('Користувач не автентифікований');
        }

        // Отримання існуючих бюджетів
        const budgets = await getUserBudgets(auth.user.id);

        // Пошук індексу бюджету, який треба оновити
        const budgetIndex = budgets.findIndex(b => b.id === budgetData.id);

        if (budgetIndex === -1) {
          dispatch(setLoading(false));
          return rejectWithValue('Бюджет не знайдено');
        }

        // Оновлення бюджету
        const updatedBudget = {
          ...budgets[budgetIndex],
          ...budgetData,
          updatedAt: new Date().toISOString()
        };

        // Заміна старого бюджету на оновлений
        budgets[budgetIndex] = updatedBudget;

        // Збереження оновленого списку бюджетів
        await AsyncStorage.setItem(`budgets_${auth.user.id}`,
JSON.stringify(budgets));

        // Оновлення стану
```

```
        dispatch(setBudget(updatedBudget));
        dispatch(setLoading(false));

        return updatedBudget;
      } catch (error) {
        console.error('Помилка оновлення бюджету:', error);
        dispatch(setError(error.message || 'Помилка при оновленні бюджету'));
        dispatch(setLoading(false));
        return rejectWithValue(error.message || 'Помилка при оновленні бюджету');
      }
    }
  );

  // Допоміжна функція для отримання транзакцій користувача
  const getUserTransactions = async (userId) => {
    try {
      const transactionsJSON = await
AsyncStorage.getItem(`transactions_${userId}`);
      return transactionsJSON ? JSON.parse(transactionsJSON) : [];
    } catch (error) {
      console.error('Помилка отримання транзакцій:', error);
      return [];
    }
  };

  // Отримання бюджету за місяць і рік
  export const fetchBudget = createAsyncThunk(
    'budgets/fetchBudget',
    async ({ month, year }, { dispatch, getState, rejectWithValue }) => {
      try {
```

```
dispatch(setLoading(true));

const { auth } = getState();
if (!auth.user) {
  throw new Error('Користувач не автентифікований');
}

// Отримання бюджетів
const budgets = await getUserBudgets(auth.user.id);

// Пошук бюджету
const budget = budgets.find(b => b.month === month && b.year === year);

if (budget) {
  // Отримання транзакцій
  const transactions = await getUserTransactions(auth.user.id);

  // Фільтрація транзакцій за місяцем і роком
  const startDate = new Date(year, month, 1);
  const endDate = new Date(year, month + 1, 0);

  const monthTransactions = transactions.filter(t => {
    const date = new Date(t.date);
    return date >= startDate && date <= endDate && t.type === 'expense';
  });

  // Оновлення витрат за категоріями
  const updatedCategories = budget.categories.map(category => {
    const spentAmount = monthTransactions
      .filter(t => t.category === category.name)
```

```
            .reduce((sum, t) => sum + t.amount, 0);

          return {
            ...category,
            spentAmount
          };
        });

        const updatedBudget = {
          ...budget,
          categories: updatedCategories
        };

        // Оновлення бюджету
        const index = budgets.findIndex(b => b.id === budget.id);
        budgets[index] = updatedBudget;
        await AsyncStorage.setItem(`budgets_${auth.user.id}`,
JSON.stringify(budgets));

        // Оновлення стану
        dispatch(setBudget(updatedBudget));
        dispatch(setLoading(false));

        return updatedBudget;
      } else {
        dispatch(setBudget(null));
        dispatch(setLoading(false));
        return null;
      }
    } catch (error) {
```

```
      const errorMessage = error.message || 'Помилка при отриманні бюджету';
      dispatch(setError(errorMessage));
      return rejectWithValue(errorMessage);
    }
  }
);

// Отримання прогресу бюджету
export const fetchBudgetProgress = createAsyncThunk(
  'budgets/fetchBudgetProgress',
  async (_, { dispatch, getState, rejectWithValue }) => {
    try {
      dispatch(setLoading(true));

      const { auth } = getState();
      if (!auth.user) {
        throw new Error('Користувач не автентифікований');
      }

      // Отримання поточного місяця і року
      const today = new Date();
      const currentMonth = today.getMonth();
      const currentYear = today.getFullYear();

      // Отримання бюджетів
      const budgets = await getUserBudgets(auth.user.id);

      // Пошук поточного бюджету
      const budget = budgets.find(b => b.month === currentMonth && b.year ===
currentYear);
```

```
        if (!budget) {
          throw new Error('Бюджет на поточний місяць не знайдено');
        }

        // Обчислення днів у місяці і днів, що пройшли
        const daysInMonth = new Date(currentYear, currentMonth + 1, 0).getDate();
        const daysPassed = Math.min(today.getDate(), daysInMonth);
        const monthProgress = Math.round((daysPassed / daysInMonth) * 100);

        // Обчислення загальних витрат
        const totalBudgeted = budget.totalPlanned;
        const totalSpent = budget.categories.reduce((sum, cat) => sum +
cat.spentAmount, 0);
        const spendingProgress = Math.round((totalSpent / totalBudgeted) * 100);
        const remainingBudget = totalBudgeted - totalSpent;

        // Визначення статусу
        let status;
        if (spendingProgress <= monthProgress * 0.9) {
          status = 'under budget';
        } else if (spendingProgress >= monthProgress * 1.1) {
          status = 'over budget';
        } else {
          status = 'on track';
        }

        // Категорії, що перевищили бюджет
        const overBudgetCategories = budget.categories
          .filter(cat => cat.spentAmount > cat.plannedAmount)
```

```
            .map(cat => ({
              name: cat.name,
              plannedAmount: cat.plannedAmount,
              spentAmount: cat.spentAmount,
              overBy: cat.spentAmount - cat.plannedAmount,
              percentage: Math.round((cat.spentAmount / cat.plannedAmount) * 100)
            }))
            .sort((a, b) => b.percentage - a.percentage);

          // Категорії, що наближаються до ліміту
          const approachingLimitCategories = budget.categories
            .filter(cat => cat.spentAmount >= cat.plannedAmount * 0.8 &&
cat.spentAmount <= cat.plannedAmount)
            .map(cat => ({
              name: cat.name,
              plannedAmount: cat.plannedAmount,
              spentAmount: cat.spentAmount,
              remainingAmount: cat.plannedAmount - cat.spentAmount,
              percentage: Math.round((cat.spentAmount / cat.plannedAmount) * 100)
            }))
            .sort((a, b) => b.percentage - a.percentage);

          // Формування даних прогресу
          const progressData = {
            month: currentMonth,
            year: currentYear,
            daysInMonth,
            daysPassed,
            monthProgress,
            totalBudgeted,
```

```
            totalSpent,
            spendingProgress,
            remainingBudget,
            status,
            overBudgetCategories,
            approachingLimitCategories
          };

          // Оновлення стану
          dispatch(setBudgetProgress(progressData));
          dispatch(setLoading(false));

          return progressData;
        } catch (error) {
          const errorMessage = error.message || 'Помилка при отриманні прогресу бюджету';
          dispatch(setError(errorMessage));
          return rejectWithValue(errorMessage);
        }
      }
    );

    // src/redux/thunks/healthThunks.js
    import { createAsyncThunk } from '@reduxjs/toolkit';
    import { setLoading, setHealthData, setHealthHistory, setError } from '../slices/healthSlice';
    import AsyncStorage from '@react-native-async-storage/async-storage';

    // Допоміжна функція для отримання транзакцій користувача
    const getUserTransactions = async (userId) => {
```

```
    try {
      const transactionsJSON = await AsyncStorage.getItem(`transactions_${userId}`);
      return transactionsJSON ? JSON.parse(transactionsJSON) : [];
    } catch (error) {
      console.error('Помилка отримання транзакцій:', error);
      return [];
    }
  };

  // Допоміжна функція для отримання історії фінансового здоров'я
  const getHealthHistory = async (userId) => {
    try {
      const historyJSON = await AsyncStorage.getItem(`health_history_${userId}`);
      return historyJSON ? JSON.parse(historyJSON) : [];
    } catch (error) {
      console.error('Помилка отримання історії фінансового здоров\'я:', error);
      return [];
    }
  };

  // Розрахунок індексу фінансового здоров'я
  export const fetchHealthScore = createAsyncThunk(
    'health/fetchHealthScore',
    async (_, { dispatch, getState, rejectWithValue }) => {
      try {
        dispatch(setLoading(true));

        const { auth } = getState();
        if (!auth.user) {
```

```
  throw new Error('Користувач не автентифікований');
 }

 // Отримання транзакцій за останні 3 місяці
 const transactions = await getUserTransactions(auth.user.id);
 const threeMonthsAgo = new Date();
 threeMonthsAgo.setMonth(threeMonthsAgo.getMonth() - 3);

 const recentTransactions = transactions.filter(t => new Date(t.date) >= threeMonthsAgo);

 // Обчислення відношення доходів до витрат
 const incomeTransactions = recentTransactions.filter(t => t.type === 'income');
 const expenseTransactions = recentTransactions.filter(t => t.type === 'expense');

 const totalIncome = incomeTransactions.reduce((sum, t) => sum + t.amount, 0);
 const totalExpenses = expenseTransactions.reduce((sum, t) => sum + t.amount, 0);

 const incomeToExpenseRatio = totalExpenses > 0 ? totalIncome / totalExpenses : totalIncome > 0 ? 3 : 0;
 const incomeToExpenseScore = Math.min(100, Math.round(incomeToExpenseRatio * 50));

 // Оцінка подушки безпеки
 const monthlyExpenses = totalExpenses / 3; // Середні витрати за місяць
 const savings = transactions
  .filter(t => t.category === 'Заощадження' && t.type === 'expense')
```

```
        .reduce((sum, t) => sum + t.amount, 0);

      const emergencyFundMonths = monthlyExpenses > 0 ? savings / monthlyExpenses : savings > 0 ? 3 : 0;
      const emergencyFundScore = Math.min(100, Math.round(emergencyFundMonths * 20));

      // Оцінка різноманітності джерел доходу
      const incomeCategories = new Set(incomeTransactions.map(t => t.category));
      const incomeSources = incomeCategories.size;
      const incomeSourceDiversityScore = Math.min(100, Math.round(incomeSources * 25));

      // Припустимо, що у нас є фінансові цілі (насправді це буде реалізовано в іншому місці)
      const goalProgressValue = 65; // Значення за замовчуванням
      const goalProgressScore = goalProgressValue;

      // Обчислення загального індексу
      const overallScore = Math.round(
        (incomeToExpenseScore + emergencyFundScore + incomeSourceDiversityScore + goalProgressScore) / 4
      );

      // Формування даних про фінансове здоров'я
      const healthData = {
        date: new Date().toISOString(),
        overallScore,
        incomeToExpenseRatio: {
          value: incomeToExpenseRatio,
```

```
            score: incomeToExpenseScore
          },
          emergencyFund: {
            value: emergencyFundMonths,
            score: emergencyFundScore
          },
          incomeSourceDiversity: {
            value: incomeSources,
            score: incomeSourceDiversityScore
          },
          goalProgress: {
            value: goalProgressValue,
            score: goalProgressScore
          }
        };

        // Отримання історії
        const healthHistory = await getHealthHistory(auth.user.id);

        // Додавання нового запису
        healthHistory.push(healthData);

        // Збереження історії
        await AsyncStorage.setItem(`health_history_${auth.user.id}`,
JSON.stringify(healthHistory));

        // Оновлення стану
        dispatch(setHealthData(healthData));
        dispatch(setLoading(false));
```

```
        return healthData;
      } catch (error) {
        const errorMessage = error.message || 'Помилка при розрахунку індексу фінансового здоров\'я';
        dispatch(setError(errorMessage));
        return rejectWithValue(errorMessage);
      }
    }
  );

  // Отримання історії фінансового здоров'я
  export const fetchHealthHistory = createAsyncThunk(
    'health/fetchHealthHistory',
    async (_, { dispatch, getState, rejectWithValue }) => {
      try {
        dispatch(setLoading(true));

        const { auth } = getState();
        if (!auth.user) {
          throw new Error('Користувач не автентифікований');
        }

        // Отримання історії
        const healthHistory = await getHealthHistory(auth.user.id);

        // Сортування за датою (від нової до старої)
        healthHistory.sort((a, b) => new Date(b.date) - new Date(a.date));

        // Оновлення стану
        dispatch(setHealthHistory(healthHistory));
```

```
            dispatch(setLoading(false));

            return healthHistory;
          } catch (error) {
            const errorMessage = error.message || 'Помилка при отриманні історії фінансового здоров\'я';
            dispatch(setError(errorMessage));
            return rejectWithValue(errorMessage);
          }
        }
      );
```

ДОДАТОК Г

Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА КРОСПЛАТФОРМНОГО МОБІЛЬНОГО ІНТЕРФЕЙСУ ДЛЯ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота на тему:
«Розробка кросплатформного мобільного інтерфейсу для управління особистими фінансами»

Автор: студент групи 2ПІ-21б Закерничний І.О.
Науковий керівник: к.т.н., професор каф. ПЗ Ліщинська Л.Б.

Вінниця - 2025

Рисунок Г.1 – Титульний слайд

Актуальність теми

Розвиток цифрових технологій і широке проникнення мобільних пристроїв у всі сфери життя суттєво вплинули на способи управління особистими фінансами. У сучасному суспільстві фінансова грамотність і вміння ефективно керувати власним бюджетом є важливими умовами для досягнення економічної стабільності та особистого успіху. Популярність мобільних застосунків для фінансового обліку постійно зростає, адже користувачі прагнуть отримати доступний, простий та зрозумілий інструмент для контролю доходів і витрат, фінансового планування, заощадження та інвестування коштів.

Незважаючи на значний прогрес у цій галузі, критичний аналіз існуючих мобільних застосунків дозволяє виявити низку суттєвих недоліків і прогалин. Більшість доступних на ринку рішень забезпечують лише базові функції, такі як простий облік доходів і витрат, формування бюджетів та елементарну візуалізацію фінансових даних. Проте далеко не всі застосунки пропонують поглиблений аналіз фінансових звичок, динамічні прогнози майбутніх витрат і доходів або ж адаптивні персоналізовані рекомендації з урахуванням змін у фінансовій поведінці користувача. Через це багато користувачів стикаються з проблемою, коли застосунок не відповідає їхнім реальним потребам, і змушені використовувати кілька додатків одночасно, що знижує загальну ефективність та зручність.

Іншою серйозною проблемою є недостатня інтеграція мобільних застосунків із банківськими системами. Хоча провідні банки поступово відкривають API для доступу до фінансових даних клієнтів, цей процес і досі є неповноцінним і непослідовним, особливо у регіональних банках і країнах із менш розвиненою цифровою інфраструктурою. Це призводить до необхідності ручного введення транзакцій, що підвищує ймовірність помилок, знижує зручність користування і збільшує витрати часу користувачів.

Також варто зазначити проблему забезпечення безпеки і конфіденційності фінансової інформації. Попри поширення сучасних методів захисту, таких як багатофакторна аутентифікація, шифрування та регулярні аудити безпеки, не всі розробники належно дотримуються цих стандартів. Недостатній рівень захисту створює ризики компрометації фінансових даних користувачів, що може мати серйозні наслідки як для індивідуальних користувачів, так і для репутації самих застосунків.

Отже, актуальною є розробка застосунку для ведення особистих фінансів.

Рисунок Г.2 — Актуальність теми

Мета, об’єкт та предмет дослідження

Метою роботи є покращення процесу ведення особистих фінансів шляхом розробки спеціалізованого програмного забезпечення, що дозволяє керувати інформацією про доходи та витрати, формувати бюджет та відстежувати його виконання.

Об'єкт дослідження – процеси управління особистими фінансами.

Предмет дослідження – методи та засоби управління особистими фінансами.

Рисунок Г.3 — Мета, об’єкт та предмет дослідження

Завдання дослідження

Основними задачами дослідження є:

- провести аналіз стану питання розробки застосунків для ведення особистих фінансів;
- дослідити існуючі аналоги та провести їх порівняльний аналіз;
- розробити модель застосунку для ведення особистих фінансів;
- розробити метод «фінансового здоров’я» користувача;
- розробити алгоритми роботи застосунку;
- розробити модулі застосунку для ведення особистих фінансів;
- провести тестування розробленого застосунку.

Рисунок Г.4 — Завдання дослідження

Наукова новизна

1. Подальшого розвитку отримав метод «фінансового здоров’я» користувача, який враховує обсяг резервного фонду, рівень диверсифікації джерел доходу та ступінь досягнення поставлених фінансових цілей, що дозволяє отримати більш об'єктивну та комплексну оцінку фінансового стану через єдиний інтегральний індекс за шкалою від 0 до 100, що спрощує розуміння загальної фінансової ситуації користувача та допомагає виявляти потенційні ризики на ранніх стадіях.

2. Подальшого розвитку набула модель системи, яка, на відміну від існуючих, реалізована за трирівневою модульною архітектурою з інтегрованими алгоритмами фінансової аналітики та диференційованим підходом до безпеки даних, що дозволяє забезпечити автоматичне генерування персоналізованих рекомендацій з повним контролем приватності даних при збереженні можливостей хмарного масштабування.

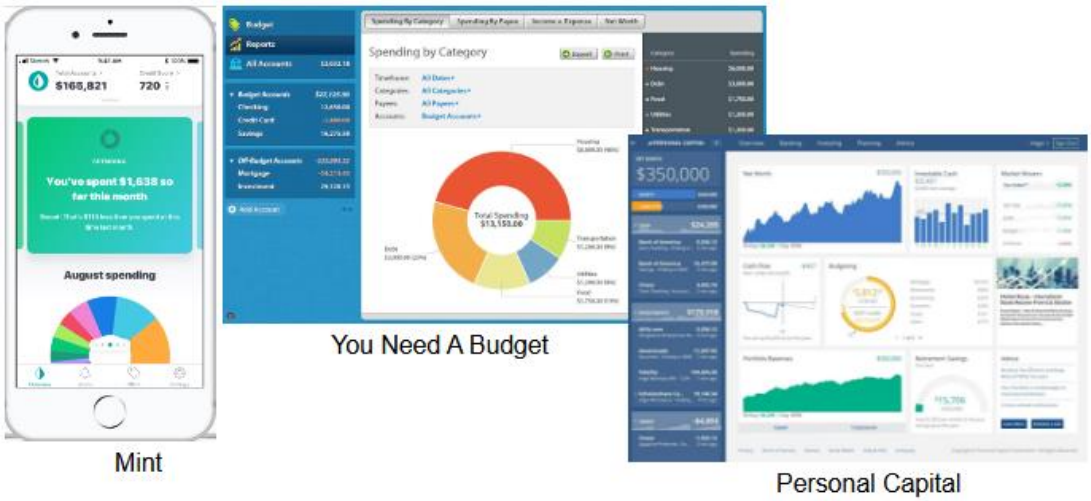
Рисунок Г.5 — Наукова новизна

Практична цінність отриманих результатів

Практична цінність розробки полягає у можливості використання застосунку для обліку особистих доходів, витрат, формування бюджету.

Рисунок Г.6 — Практична цінність отриманих результатів

Аналоги



Mint

You Need A Budget

Personal Capital

Рисунок Г.7 — Аналоги

Порівняльний аналіз аналогів

Характеристика	Personal Capital	You Need A Budget (YNAB)	Mint	Власна розробка
Відстеження витрат	+	+	+	+
Автоматична категоризація транзакцій	+	+	+	+
Встановлення бюджету	+	-	+	+
Індекс фінансового стану	-	-	-	+
Персональні рекомендації	+	-	-	+
Встановлення цілей	+	+	+	+
Сумарний бал	5	3	4	6

Рисунок Г.8 — Порівняльний аналіз аналогів

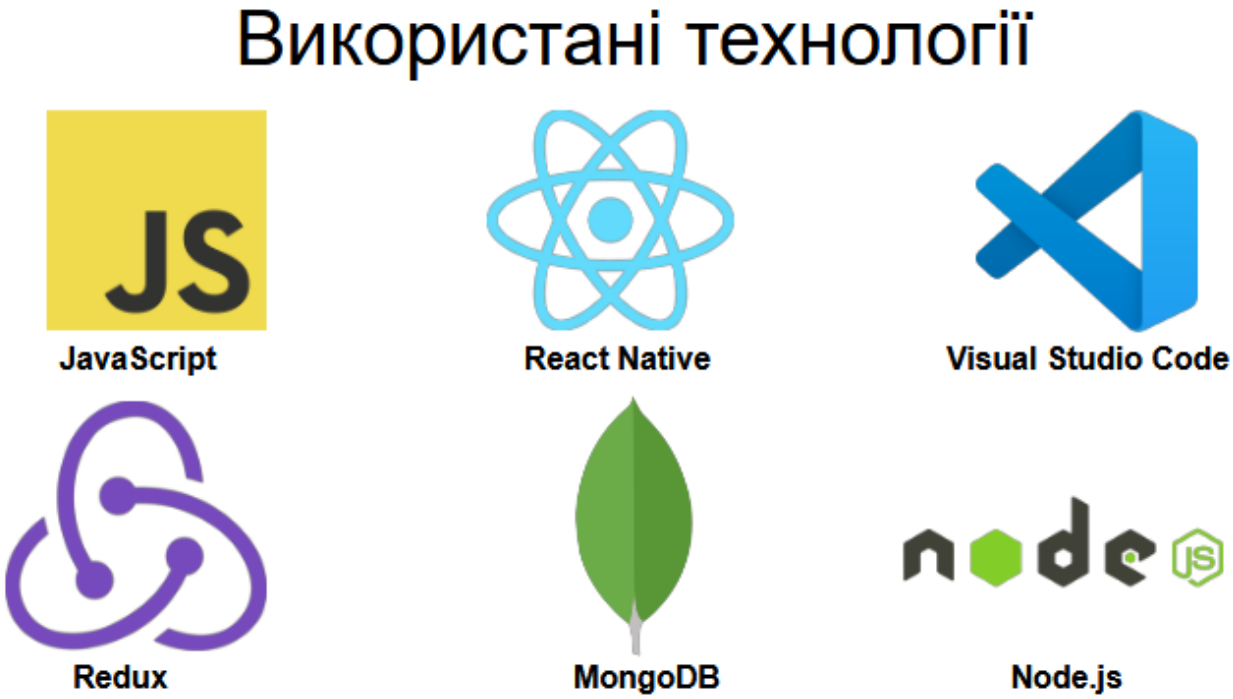


Рисунок Г.9 — Використані технології

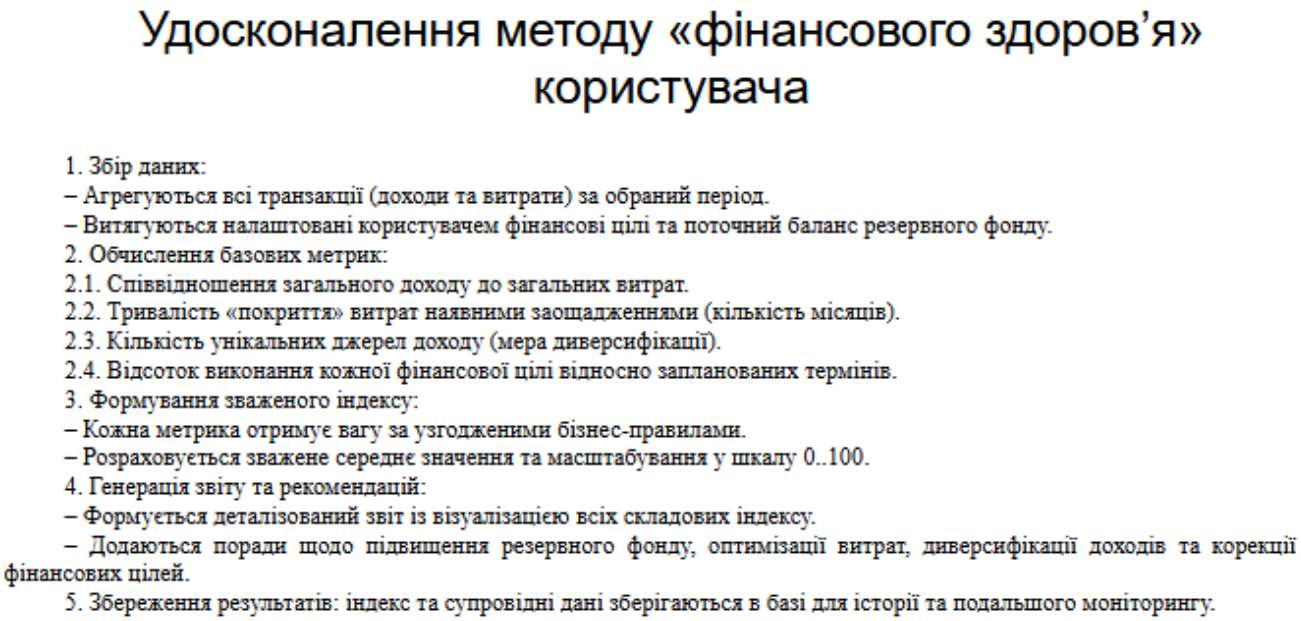


Рисунок Г.10 — Удосконалення методу «фінансового здоров'я» користувача

Удосконалення методу «фінансового здоров’я» користувача

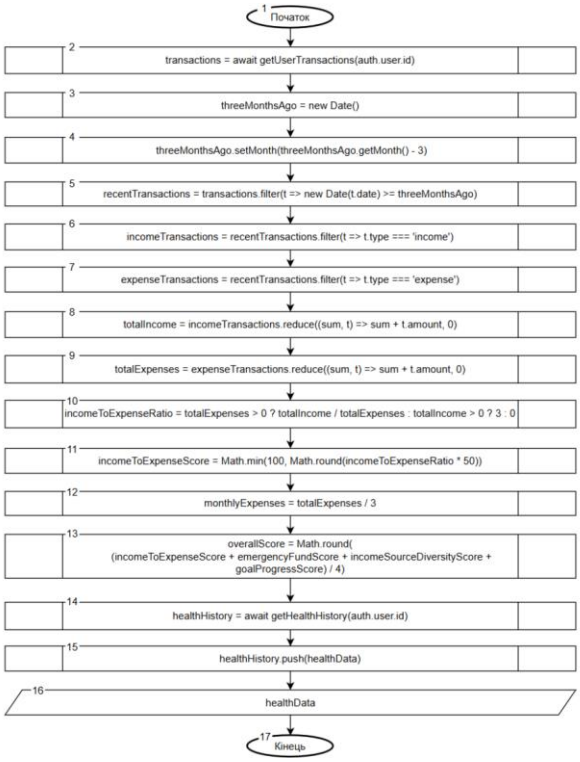


Рисунок Г.11 — Удосконалення методу «фінансового здоров’я» користувача

Розробка моделі застосунку

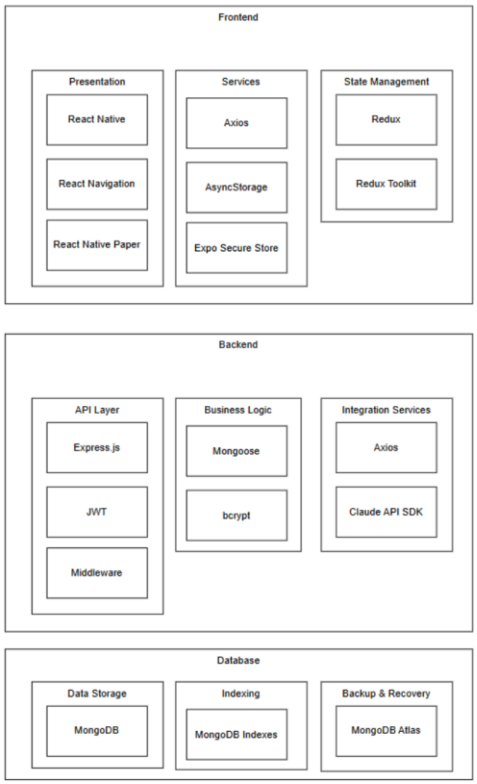


Рисунок Г.12 — Розробка моделі застосунку

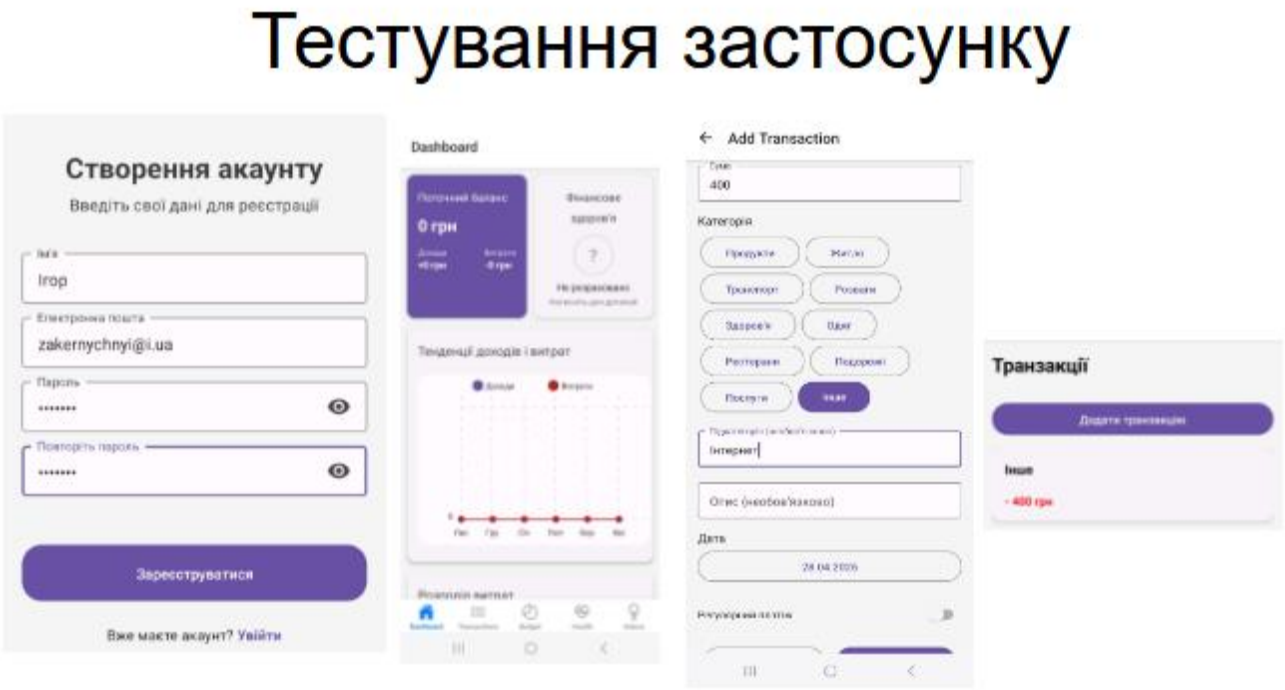


Рисунок Г.13 — Тестування модуля реєстрації та модуля управління транзакціями

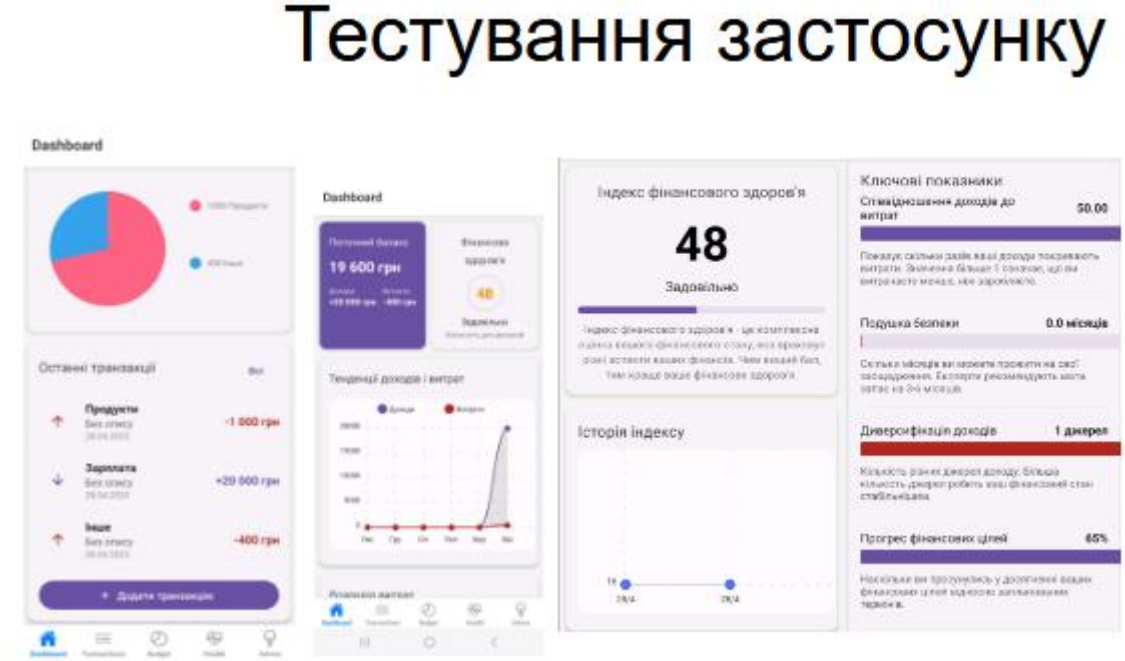


Рисунок Г.14 — Тестування розрахунку індексу фінансового здоров’я

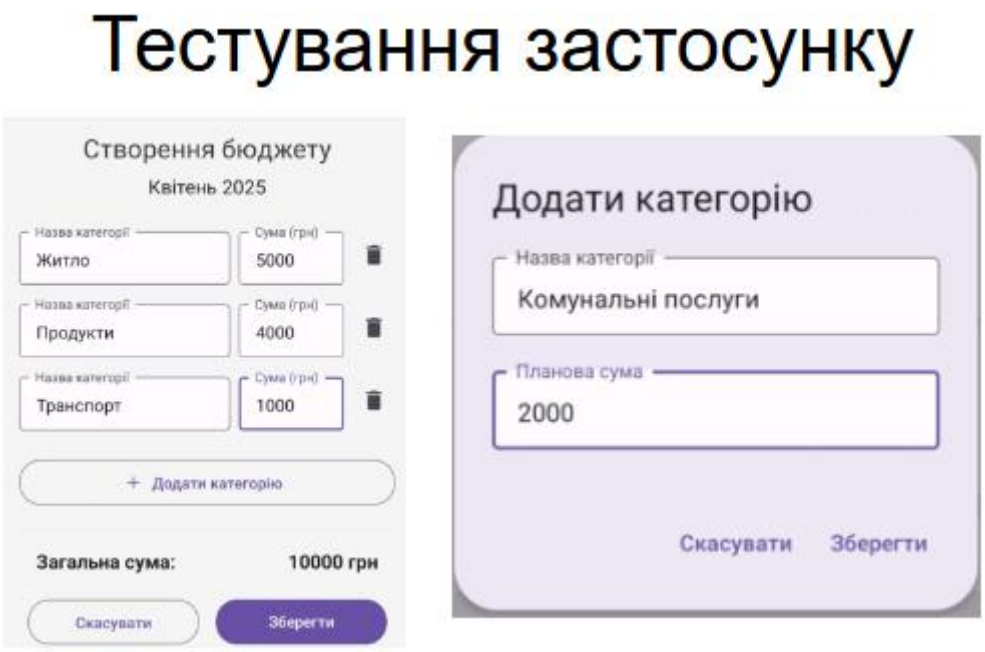


Рисунок Г.15 — Тестування створення бюджету

Висновки

У результаті виконання бакалаврської кваліфікаційної роботи було розроблено мобільну систему для ведення особистих фінансів. Для розробки використовувалося середовище розробки Visual Studio Code. Бакалаврську кваліфікаційну роботу реалізовано згідно методичних вказівок [28].

Проведено аналіз стану питання розробки застосунків для ведення особистих фінансів. Було визначено, що на сьогодні існує значна кількість застосунків для управління особистих фінансів, однак більшість із них мають обмеження щодо функціональності, персоналізації та прогнозування фінансової поведінки користувачів, через що актуальною є розробка застосунку для управління особистими фінансами.

Розглянуті існуючі аналоги: Mint, Personal Capital, You Need A Budget, проведено порівняльний аналіз із власною розробкою, в результаті чого було доведено актуальність власної розробки, та її переваги над аналогами, до яких належить: наявність персоналізованих рекомендацій та розрахунок індексу фінансового стану користувача.

Розроблено модель застосунку для ведення особистих фінансів, яка складається з взаємопов'язаних компонентів, що забезпечують увесь функціонал для додавання та управління інформацією про особисті фінанси.

Рисунок Г.16 — Висновки (частина 1)

Висновки

Удосконалено метод «фінансового здоров'я» користувача, який враховує рівень диверсифікації джерел доходу, запас коштів та виконання поставлених фінансових цілей, на основі чого розраховує індекс, що визначає рівень фінансової стабільності користувача.

Розроблено діаграми та алгоритми роботи застосунку, які дозволяють додавати транзакції, керувати транзакціями, управляти бюджетом, розраховувати індекс «фінансового здоров'я».

Проведено варіантний аналіз мов програмування: JavaScript, Kotlin, Python з метою вибору мови програмування розробки кросплатформного мобільного мобільного інтерфейсу для управління особистими фінансами. У результаті аналізу було обрано мову програмування JavaScript.

Розроблено модулі застосунку для ведення особистих фінансів, описано основні функції, що використовуються при роботі цих модулів.

Проведено тестування розробленого застосунку. Продемонстровано розроблений функціонал, перевірено його роботу. Розроблений застосунок дозволяє вести облік витрат та доходів, отримувати графіки для візуалізації даних, отримувати індекс фінансового здоров'я, отже виконує поставлені на початку розробки завдання.

Рисунок Г.17 — Висновки (частина 2)

Апробація результатів роботи і публікації

Апробація матеріалів. Описані у бакалаврській кваліфікаційній роботі положення доповідалися на міжнародній науково-практичній конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)».

Публікації. Закерничний І.О. Особливості мобільного застосунку для ведення особистих фінансів/ І.О. Закерничний, Л.Б. Ліщинська // Матеріали Міжнародної науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми,перспективи (МН-2025)». – Вінниця: ВНТУ, 2025. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/viewFile/25194/20784

Рисунок Г.18 — Апробація результатів роботи і публікації