Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

РОЗРОБКА IOS-ЗАСТОСУНКУ ДЛЯ ОРГАНІЗАЦІЇ ТА МЕНЕДЖМЕНТУ ПОБУТОВИХ ПОТРЕБ

Виконав: студент 4 курсу, групи ЗКН-21б
спеціальності 122 Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Андрій ЗАХАРЕВИЧ
(прізвище та ініціали)
Керівник: асистент
Єгор СІЛАГІН
(прізвище та ініціали)
« 04 » червня 2025 р.
Рецензент: к. т. н., доц. каф. АІІТ
Олексій КОЗАЧКО
(прізвище та ініціали)
« 05 » червня 2025 р.

Допущено до захисту
Зав. кафедри проф., д.т.н. Андрій ЯРОВИЙ
« 06 » червня 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо - професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
« 05 » травня 2025 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Захаревичу Андрію Миколайовичу

1. Тема роботи: Розробка IOS-застосунку для організації та менеджменту побутових потреб.

Керівник роботи: Сілагін Єгор Олексійович, ассистент кафедри КН.
затверджені наказом вищого навчального закладу « 20 » березня 2025 року №97.

2. Термін подання студентом роботи « 30 » травня 2025 року.

3. Вихідні дані до роботи : мова програмування – об'єктно-орієнтована, мінімалістичний GUI, ОС – IOS.

4. Зміст текстової частини (перелік питань, які потрібно розробити):
Обґрунтування доцільності розробки застосунку для організації та менеджменту побутових потреб; Проектування організації та менеджменту побутових потреб; Програмна реалізація IOS-застосунку для організації та менеджменту побутових потреб; Тестування розробленої програми.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
Структура програмного модуля постоптимального аналізу задач лінійного програмування; Схема алгоритму роботи IOS-застосунку для організації та менеджменту побутових потреб; UML-діаграма класів IOS-застосунку для організації та менеджменту побутових потреб; Приклад роботи програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдан прийн
1-3	Єгор СІЛАГІН асистент кафедри КН		

7. Дата видачі завдання « 05 » травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Аналіз предметної області IOS-застосунку для організації та менеджменту побутових потреб	05.05.2025	07.05.2025	
2	Розробка IOS-застосунку для організації та менеджменту побутових потреб	08.05.2025	11.05.2025	
3	Програмна реалізація IOS-застосунку для організації та менеджменту побутових потреб.	12.05.2025	29.05.2025	
4	Розробка інструкції користувача	30.05.2025	01.05.2025	
5	Оформлення матеріалів до захисту БКР та розробка презентації	02.06.2025	04.06.2025	

Студент _____ **Андрій ЗАХАРЕВИ**
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ **Єгор СІЛАГІ**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 49 сторінки формату А4, які включають 14 рисунків і 5 таблиць. У списку використаних джерел зазначено 17 найменування.

У загальній частині роботи на основі аналізу існуючих технічних рішень, методів та технологій доведена доцільність розробки IOS-застосунку для організації та менеджменту побутових потреб. Розроблено алгоритм роботи, структура програмного модуля та UML діаграми класів, які спрощують розробку та розуміння структури програмного забезпечення. Програмний продукт розроблений в інтегрованому середовищі Xcode.

Розроблено програмний продукт та проведено його тестування. Результати тестування розробки вказують на те, що основні функції IOS-застосунку для організації та менеджменту побутових потреб реалізовано.

Ключові слова: IOS-застосунок, організація та менеджмент, побутові потреби.

ABSTRACT

The bachelor's thesis consists of 49 A4 pages, including 14 figures and 5 tables. The list of references contains 17 titles.

In the general part of the work, based on the analysis of existing technical solutions, methods and technologies, the feasibility of developing an IOS application for organizing and managing household needs is proved. The algorithm of work, the structure of the program module and UML class diagrams have been developed, which simplify the development and understanding of the software structure. The software product is developed in the integrated Xcode environment.

The software product is developed and tested. The results of development testing indicate that the main functions of the IOS application for organizing and managing household needs have been implemented.

Keywords: IOS application, organization and management, household needs.

ЗМІСТ

ВСТУП

Актуальність дослідження.

В сучасному світі людина щодня отримує та витрачає цінний ресурс – час. Планування дня, чи то тижня, фінансових витрат, догляду за домашніми тварин, чи рослинами, це та буденність, за якою нам потрібно слідкувати, і без належного розподілу часу та коштів на виконання цих справ, ми будемо позичати ресурси, в майбутніх себе. Тому все частіше люди використовують застосунки на смартфонах, для організації побуту, планування часу, та контролю над витратами, саме вони дозволяють структурувати всі справи.

Наразі існує велика кількість застосунків для планування дня, наприклад «Diary: Daily, Weekly Planner», який дозволяє вести щоденник з планування, «Todoist», що дає можливість створювати списки з задачами, які потрібно виконати. До переліку застосунків, які допомагають керувати фінансовими витратами входять «monobudget», в функціоналі якого є категоризація витрат, та можливість планування бюджету, «Plant App: Plant Identifier», корисна програма, для тих хто піклується про свої рослини. Проте такі аспекти як прибирання оселі, чи догляд за тваринами та рослинами, часто відходять на задній план, або ж про них забувають взагалі.

Предмет дослідження – програмні засоби для організації особистого менеджменту, контролю за витратами, та догляду за побутом.

Об’єкт дослідження – процеси організації особистого менеджменту, контролю за витратами, та догляду за побутом.

Мета дослідження – розширення функціональних можливостей програмного забезпечення для організації та менеджменту побутових потреб.

Задачі дослідження:

1. Провести аналіз предметної області, існуючих методів, алгоритмів та інструментів для організації та менеджменту побутових потреб.

2. Розробка алгоритму функціонування застосунку для організації та менеджменту побутових потреб.

3. Програмна реалізація застосунку для організації та менеджменту побутових потреб за допомогою мови програмування Swift.

4. Тестування застосунку для організації та менеджменту побутових потреб.

5. Розробка інструкції користувача застосунку для організації та менеджменту побутових потреб.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ IOS-ЗАСТОСУНОК ДЛЯ ОРГАНІЗАЦІЇ ТА МЕНЕДЖМЕНТУ ПОБУТОВИХ ПОТРЕБ

1.1 Аналіз методів організації та менеджменту побутових потреб

Для покращення ефективності організації та менеджменту побутових потреб, було проведено аналіз різноманітних методик, які допомагають контролювати планування дня, розподіляти побутові справи у розкладі, розставляти пріорітетність для виконання різних задач.

Використання тайм-менеджменту (управління часом), спрямоване на свідомий контроль над часом, що витрачається для виконання певної діяльності, метою якого є підвищення продуктивності та ефективності в організації та менеджменті побуту.

Методи класничного тайм-менеджменту можна поділити на кілька категорій:

1 Піраміда Франкліна – це метод пріоритизації справ, що ґрунтується на загальних життєвих цінностях та принципах. Піраміда ґрунтується на кількох рівнях, що відповідають певній ланці в житті:

- Перший рівень – життєві цінності, він є основою піраміди, сюди відноситься все що є найважливіше в житті: сім’я, здоров’я, розвиток, саме цей рівень визначає життєві орієнтири людини.
- Другий рівень – глобальна ціль, відображає загальну мету життя людини, змушує замислитися над запитанням: «Навіщо я живу?».
- Третій рівень – генеральний план, ланка, яка уособлює в собі плани на майбутні 10 – 15 – 20 років, цілі які людина хоче досягнути в різних сферах протягом тривалого часу.
- Четвертий рівень – довгострокові цілі, те, що людина планує виконати за наступні рік-два.
- П’ятий рівень – місячні або тижневі плани, планування важливих завдань та справ, що призведуть до реалізації довгострокових цілей.

- Шостий рівень – щоденні справи, останній рівень піраміди, у якому розглядаються конкретні дії на день, вони повинні випливати з планів кожного з попередніх рівнів, та бути ефективними.

2 Матриця Стівена Кові – покращений метод матриці Ейзенхауера, який допомагає розподіляти завдання за важливістю та терміновістю, задля підвищення ефективності виконання завдань [1].

Таблиця 1.1 - Розподіл квадрантів матриці

	Термінове	**Нетермінове**
Важливе	Кризові ситуації Термінові завдання Негайні проблеми	Стратегічне планування Розвиток, навчання, стосунки Профілактика
Неважливе	Втрата зосередження Чужі пріорітети Вигадані термінові справи	Розваги, соцмережі Прокрастинація Виконання побутових завдань при наявних робочих

В даній таблиці зображений розподіл за «квадрантами», приклад опису:

- Перший квадрант (важливе та термінове) – справи, що потребують якнайшвидшого вирішення.
- Другий квадрант (важливе та нетермінове) – зона стратегічного планування, саморозвитку.
- Третій квадрант (неважливе та термінове) – дозволяє створити ілюзію зайнятості, насправді не приносячи вагомих результатів.
- Четвертий квадрант (неважливе та нетермінове) – зона якої слід уникати, в ній легко втрачається час, і підвищується прокрастинація.

Важливою складовою менеджменту побутових потреб є управління фінансами, особливо контроль персольнальних витрат. Це дозволяє розподіляти ресурси та планувати майбутні фанансові цілі / витрати. До методів управління фінансами належать [2]:

- Фіксація та категоризація витрат – основний процес для відслідковування кожної витрати чи оплати послуги.
- Аналіз фінансових даних – процес використання інструментів аналітики для отримання глибшого розуміння фінансового стану, виявлення тенденцій та планування фінансових витрат. Можлива візуалізація даних за допомогою графіків, діаграм, та таблиць, які допомагають легко представити складні фінансові показники.
- Планування бюджету та фінансових цілей – створення бюджету, фінансових цілей, та планування майбутніх витрат чи збережень.
- Використання штучного інтелекту та машинного навчання – сучасні методи автоматизованого аналізу фінансових активностей дозволяють визначити шлях до фінансової цілі, та класифікувати доцільність витрат.

В організації та менеджменті побутових потреб, ключову роль відіграє менеджмент завдань, він дозволяє керувати паралельним виконанням множин завдань, з використанням наступних методів [3]:

- Kanban – фокусується на візуалізації робочого процесу, обмеженні кількості роботи на одночасне виконання, складається з дошки зі стовпцями «Зробити», «Виконувані завдання», «Завершені завдання».
- Scrum – підхід, що передбачає роботу з розбивкою на короткі спринти, що дозволяє пріоритизувати виконання завдань.
- Waterfall – це каскадна модель, чітко структурований план виконання завдання, що включає в себе визначення вимог, проектування, реалізацію, перевірку, та обслуговування створеного продукту (в більшості випадків використовується при виконання IT-завдань).

Таблиця 1.2 - Порівняння методів менеджменту

Метод	**Гнучкість**	**Структура**
Kanban	Середня	Проста
Scrum	Висока	Чітка
Waterfall	Низька	Дуже чітка

В даній таблиці наведено порівняння методології для менеджменту завдань, з розподіленням за їх гнучкістю та структурою.

1.2 Аналіз програмних засобів організації та менеджменту побутових потреб

На ринку наразі представлена велика кількість програмного забезпечення, що призначене для організації та менеджменту часу, завдань, фінансів, Аналіз існуючих рішень дозволяє визначити їх сильні сторони та наявні недоліки:

- Trello – дозволяє створювати дошки та картки для візуалізації завдань, створювати картки, встановлювати терміни виконання, використовується методологія Kanban.
- Asana – комплексне рішення для створення завдань, підзавдань, їх делегування.
- Todoist – застосунок для створення завдань.
- Microsoft To Do – застосунок для створення завдань (рис 1.1.).
- EmpMonitor – застосунок орієнтований на моніторинг активності та дотримання правил, створює діаграми з порівнянням витрати часу на задання.

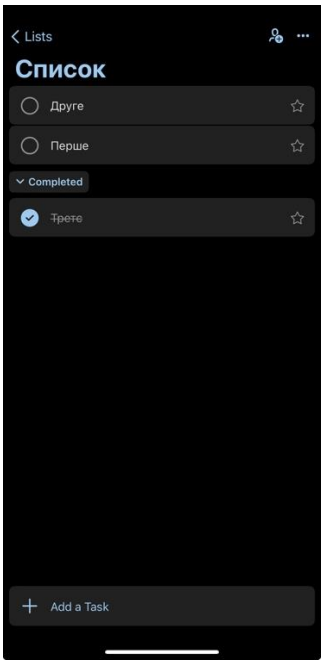

Рисунок 1.1 - Вигляд інтерфейсу застосунку «Microsoft To Do»

На зображеному рисунку видно головний інтерфейс застосунку Microsoft To Do. Головна задача – створити іменований список, і вже в ньому створювати окремі завдання, що мають бути виконані [4].

Загалом, аналіз програмних засобів, що наявні на ринку дозволив нам визначити, що існуючі рішення часто мають обмежений функціонал, або розраховані на середні чи великі організації, через що потребують додаткової плати за користування сервісом.

- Expense Tracker – MonAi – мінімалістичний застосунок для відстеження витрат, має функцію відстеження витрат, класифікації транзакцій, та створення бюджету.
- Money Keeper: Expenses Tracker – застосунок, що дозволяє додавати витрати за допомогою календаря, на певний день місяця, у висновку він наводить діаграму витрат за календарний місяць.
- Monobudget – дозволяє проводити бюджетування, та створювати звіти по витратах (рис 1.2).

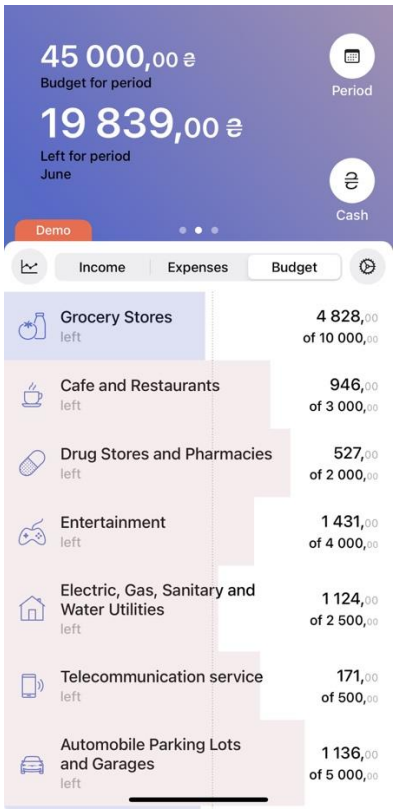

Рисунок 1.2 - Вигляд інтерфейсу застосунку «monobudget»

Дані застосунки мають функції створення бюджетів, відстеження витрат, класифікування транзакцій, але можуть мати обмежені функції звітування, або навпаки, надто перевантажені звіти, та бракує інтеграції з різними аспектами життя. Деякі з них є дорогими для окремих користувачів, через наявність плати у вигляді щомісячної підписки, тому навіть незважаючи на їх відомість, вони можуть мати певні мінуси.

Розглядаючи такі специфічні побутові потреби як догляд за кімнатними рослинами, чи домашніми тваринами, можна знайти кілька застосунків, які використовують різні підходи до надання рекомендацій – аналіз користувацького досвіду, системний підхід, задіяння штучного інтелекту, однак це все може бути загорнуто у модель користування за підпискою, що обмежує коло користувачів, які можуть цим застосунком користуватися [5].

- Plant App: Plant Identifier – застосунок для догляду та контролю кімнатних рослин, що дозволяє ідентифікувати ботанічну назву рослини, та створити нагадування про її полив, підкормку та догляд (рис 1.3).
- Pet-Care – застосунок для догляду за домашніми улюбленцями, дозволяє створити кратку улюбленця, додати фото, нагадування про годівлю, прогулянки, та лікування.

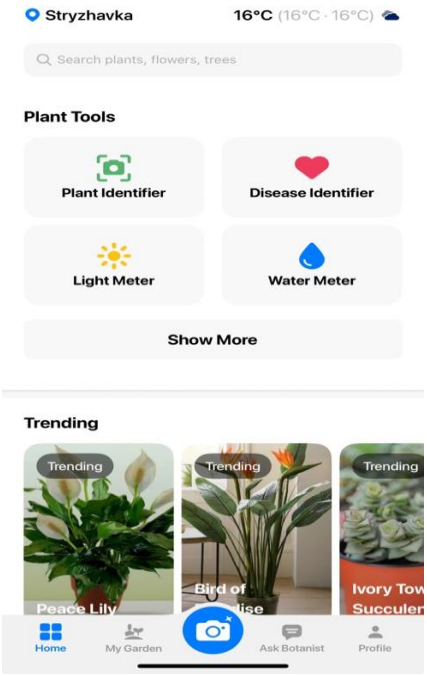

Рисунок 1.3 – Вигляд інтерфейсу «Plant App: Plant Identifier»

Загалом, проаналізувавши існуючі програмні засоби для управління часом, завданнями та фінансами, можна дійти висновку, що багато з них є спеціалізованими, мають обмежений функціонал, або не забезпечують достатнього рівня персоналізації, інтеграції та автоматизації, необхідних для комплексного менеджменту побутових потреб.

1.3 Постановка задачі для організації та менеджменту побутових потреб

Після проведення аналізу різноманітних методів організації та менеджменту побутових потреб, було виокремлено певні недоліки в наявних на ринку застосунках.

Основним недоліком є відсутність універсального, зручного та персоналізованого застосунку для комплексного управління побутовими справами, до яких входять тайм-менеджмент, менеджмент завдань та контроль фінансів, в одному місці. Існуючі рішення є переважно вузькоспеціалізованими, через що користувач вимушений використовувати кілька застосунків, з різними інтерфейсами, що є незручним, та дублює наявну інформацію.

Більшість існуючих рішень для менеджменту побутових справ не враховують індивідуальних особливостей користувачів, а саме не мають персоналізації в застосунку, відсутністю сповіщень про виконання завдань, або застосунок має малу точність прогнозування та рекомендації.

Для автоматизації рутинних справ, таких як облік витрат, відстеження фінансових операції, автоматична категоризація витрат, наявні застосунки можуть вимагати значних зусиль та часу. Крім того, вони можуть мати обмежений функціонал, високу вартість щомісячної підписки, або використовувати застарілі методи та алгоритми, що не забезпечує необхідної швидкодії, точності та надійності.

Постановка задачі, має на меті проведення аналізу існуючих на ринку застосунків, з оцінкою їх сильних та слабких сторін, визначення сценарію використання користувачем, шляхом виділення ключових функцій. Спроектувати структуру застосунку, розбивши на логічні блоки, реалізувати функціональні модулі (додавання справ, редагування справ, встановлення тегів, та дедлайнів). Інтегрувати систему сповіщень, яка буде синхронізуватися з дедлайнами завдань, й оповіщувати користувача про них. Забезпечити безпечне локальне зберігання даних на пристрої.

1.4 Особливості застосування та перспективи розвитку програмного засобу для організації та менеджменту побутових потреб

Розроблений застосунок для організації та менеджменту побутових потреб матиме певні особливості та перспективи розвитку, завдяки можливостям платформи IOS та її інтеграції сучасних технологій.

Особливості застосування:

- Застосунок завжди під рукою, що забезпечує швидку можливість фіксації інформації, незалежно від зв’язку та місцезнаходження.
- Тісна інтеграція з апаратним забезпеченням IOS, та інструментарієм розробника забезпечує високий рівень сумісності всіх компонентів.
- Централізоване керування, організація та менеджмент усіх справ в одному місці.
- Візуалізація даних шляхом відображення діаграм виконання завдань, тощо.
- Застосунок пропонує персоналізовані рекомендації з виконання справ.

Існує кілька перспектив розвитку застосунку:

- Інтеграція штучного інтелекту та машинного навчання для автоматизації аналізу даних, прогнозування та надання персоналізованих рекомендацій, надання рекомендацій з оптимізації бюджету, чи оптимального планування завдань з урахуванням дедлайнів, пріоритетів та навантаження.
- Інтеграція голосового помічника Siri, для можливості усної постановки завдання, контролю прогресу виконання, тощо.
- Додавання функції інтеграції таймеру для фокусування на виконанні справи, створення аналітики про виконання кількості справ за певний період.
- Розширення можливостей фінансової аналітики шляхом створення детальних звітів, порівняння періодів, та аналізу за категоріями.

- Розробка кросплатформенного застосунку для MacOS, шляхом використання тієї ж бази даних, через тісну підтримку IOS застосунків на системі MacOS.

1.5 Висновки до розділу 1

Проведений аналіз предметної області організації та менеджменту побутових потреб дозволив окреслити ключові теоритичні та практичні аспекти. Організація та менеджмент побутових потреб розглядається к ефективний інструмент для підвищення власної ефективності, управління часом та завданнями, які об’єднанні в одному середовищі, з’ясовано, що на ринку існують вузькоспеціалізовані рішення, що змушує користувачів встановлювати по декілька застосунків, для власного менеджменту справ, але в умовах звичайного користувача, застосунок в якому об’єднано керування різними аспектами життя, дозволяє оптимізувати час на виконання побутових потреб, розглянуто переваги розробки на Swift.

2 ПРОЕКТУВАННЯ IOS-ЗАСТОСУНКУ ДЛЯ ОРГАНІЗАЦІЇ ТА МЕНЕДЖМЕНТУ ПОБУТОВИХ ПОТРЕБ

2.1 Розробка структури програмного застосунку

Проектування структури застосунку є важливим етапом у створенні будь-якої інформаційної системи, оскільки саме на ньому визначаються її принципи та ключові компоненти. Архітектура програмного забезпечення стосується загальної структури та організації системи, включаючи принципи проектування, шаблони, та рішення, які рухають її розробку. Правильно розроблена структура забезпечує масштабованість, гнучкість , продуктивність та безпеку застосунку.

Для забезпечення ефективності, масштабованості, та зручності підтримки програмного забезпечення застосовується розбиття на модулі, вони виконують чітко визначені функції, мають мінімум зв’язків між собою, уникаючи використання глобальних змінних, це зменшує залежності, та спрощує модифікацію частин системи. Завдяки чітко виначеним вхідним та вихідним параметрам кожного модуля, отримується передбачувана поведінка, що зменшує кількість помилок та невдач.

UML включає в себе різні типи діаграм, які служать для візуалізації різних елементів системи, та їх взаємодії, наприклад:

- Діаграма класів – моделює структуру системи, вказуючи класи, їх атрибути, методи, та зв’язки між ними.
- Діаграма компонентів – відображає компоненти системи, та їхні інтерфейси, а також залежності між ними.
- Діаграма розгортання – графічно зображає архітектуру системи, показуючи розміщення апаратного та програмного забезпечення.
- Діаграма станів – відображає життєвий цикл об’єктів, показуючи можливі стани та переходи між ними.
- Діаграма активності – описує послідовність дій користувача під час взаємодії з застосунком.

Важливою складовою структури застосунку є організація зберігання даних, часто використовується нормалізація таблиць, це забезпечує структуроване, ефективне та надійне зберігання даних, уникаючи дублювання, та забезпечуючи їх цілісність, також вибір бази даних напряму впливає на швидкодію системи.

Проектування інтерфейсу користувача має включати в себе простоту, інтуїтивність, та адаптивність. Інтерфейс має бути зрозумілим користувачеві, має працювати з різними розмірами екрану, реагувати на режим світлової теми, змінюватися залежно від положення пристрою, візуалізувати за допомогою діаграм чи таблиць інформацію.

Структуру можна скласти з наступних модулів:

- Модуль керування фінансами, що дозволяє проводити дії з транзакціями, встановлювати бюджету, створювати звіти, тощо.
- Модуль керування завданнями, який дозволяє користувачеві проводити дії з завданнями, , створення, редагування, додавання дедлайнів.
- Модуль побутових справ, дозволяє додавати активності, догляд за тваринами та рослинами, нагадування, тощо.

На рисунку 2.1 представлено архітектуру застосунку для організації та менеджменту побутових потреб:

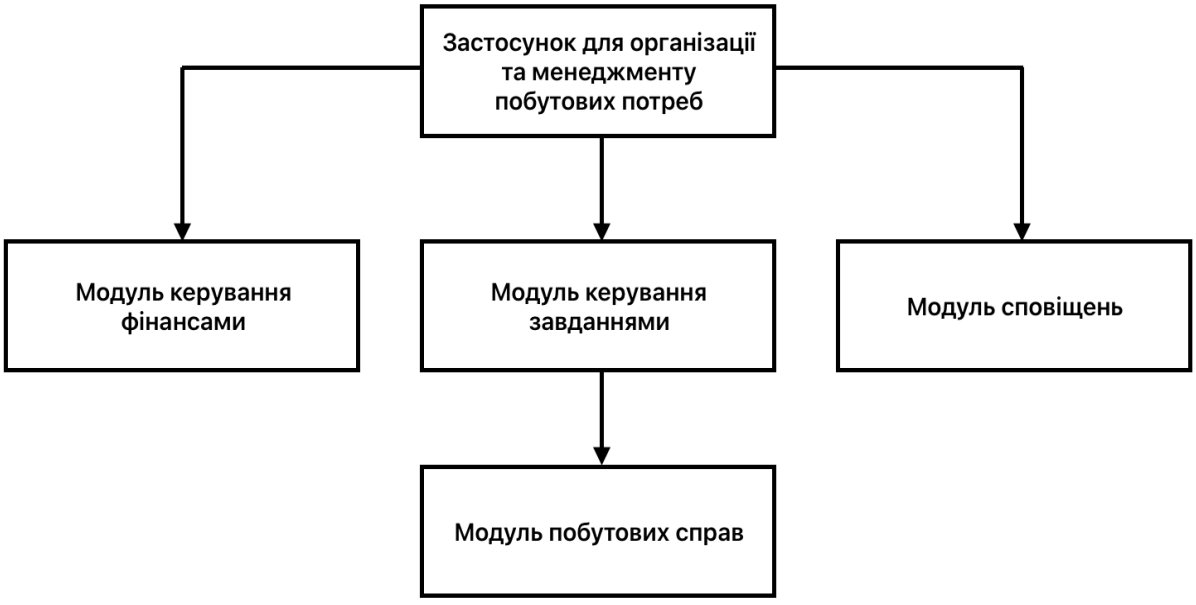

Рисунок 2.1 – Архітектура застосунку

На рисунку представлено архітектуру, розроблену для організації та менеджменту побутових потреб, вона чітко розподіляється на модулі, кожен з яких виконує певну функцію, та взаємодіє з іншими компонентами системи.

Головним елементом тут є модуль керування завданнями, він дозволяє взаємодіяти користувачеві з застосунком, створювати, видаляти, змінювати завдання, налаштовувати побутові справи. Завдяки своїй зрозумілості та простоті користувач може швидко відслідковувати процес виконання завдань.

Модуль керування завданнями взаємодіє з модулем побутових справ, це дозволяє інтегрувати завдання щодо виконання побутових справ ефективніше, й допомагає спростити роботу модуля при великих обсягах даних.

Модуль керування фінансами працює окремо від модуля керування завданнями, це дозволяє знизити навантаження на базу даних, коли ведеться облік транзакцій, чи витрат, та взаємодія даних з виокремленою базою даних пришвидшує швидкодію системи.

Модуль сповіщень генерується менеджером сповіщень, який обробляє інформацію про дедлайни завдань, облік фінансів, побутові потреби. Можливість надсилання сповіщень при досягнені певного часу до закінчення виконання завдання, чи за іншим фактором дозволяє користувачеві бути завжди усвідомленим в порядку виконання справ.

Застосунок буде розроблений з використанням об’єктно орієнтованого програмування, що дозволить створити логічну структуру системи, а SwiftUI дозволить інтерфейсу адаптуватися до змін в реальному часі. Це дозволить розбити на кілька підмодулів, які в свою чергу дозволять гнучкіше налаштувати систему.

Завдяки змінам інтерфейсу в реальному часі, залежно від певних умов, є можливість автоматично змінювати наприклад шкалу прогресу виконання завдань, при відмічуванні завдання як виконане, чи створенні нового завдання.

2.2 Розробка алгоритму роботи системи

Розробка алгоритму роботи системи є основним етапом створення будь-якого застосунку, саме на цьому етапі визначається логіка функціонування, послідовність дій, та взаємодія компонентів. Це передбачає деталізацію поставлених завдань, обґрунтування вибору методів та моделей розробки, та візуалізацію процесів за допомогою діаграм. Ефективно розроблений алгоритм є запорукою точного, персоналізованого та зручного функціонування системи (рис 2.2).

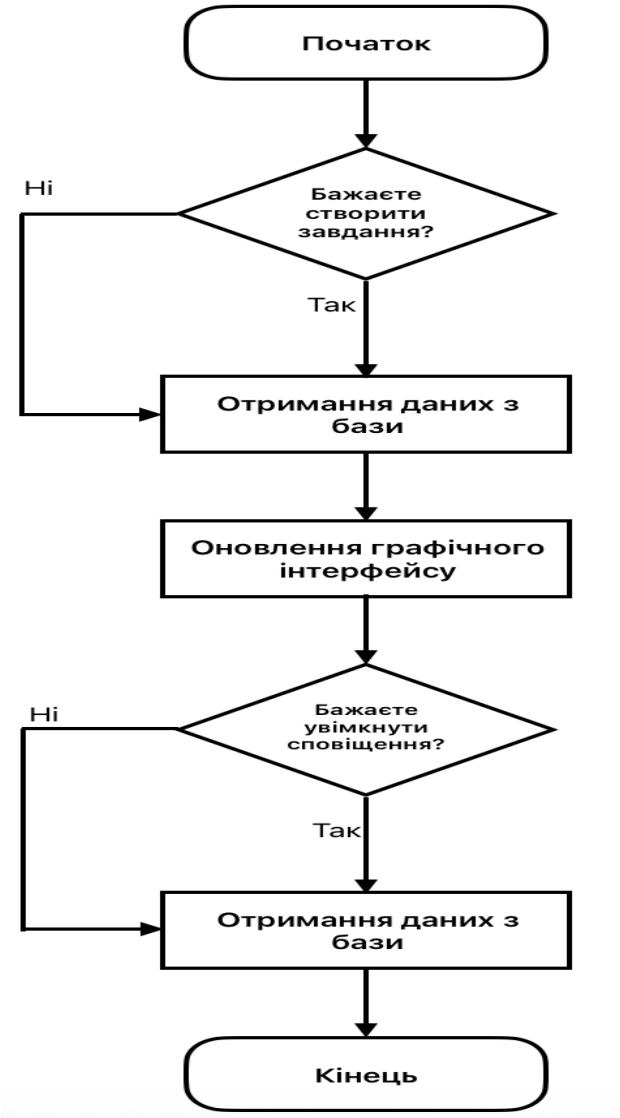

Рисунок 2.2 – Алгоритм роботи застосунку

Проектування алгоритмів включає в себе розробку логіки кожної дії, та визначення послідовності операції, які потрібні для досягнення поставленої мети. Процес розробки алгоритму включає в себе:

- Визначення цілей та предметної області, щоб зрозуміти проблеми та питання, які має вирішити застосунок.
- Аналіз вхідних даних, для категоризації за типом даних.
- Вибір математичних моделей та методів штучного інтелекту.
- Передбачення механізму для виявлення та обробки помилок системи.
- Проектування алгоритму з поглядом на майбутню масштабованість.
- Опис послідовності кроків виконання системою, представлених у вигляді блок-схем чи діаграм.

Візуалізація алгоритму створюється за допомогою діаграм уніфікованої мови моделювання (UML), вони допомагають зрозуміти конструкцію, архітектуру та логіку роботи застосунку, наприклад [6]:

- ER (Entity-Relationship) – модель, дозволяє візуалізувати структуру даних з зв’язками між ними, для проектування бази даних.
- Діаграми класів – візуалізація статичної структури системи, з зображенням класів, атрибутів, та зв’язків.
- Діаграма прецедентів – дозволяє виокремити основні сценарії взаємодії користувача з системою.
- Діаграма діяльності – моделює послідовність дій, потоку керування в процесах.
- Діаграма послідовності – моделює взаємодію об’єктів системи в часі, та обміну повідомлень між ними, до досягнення певного результату.
- Структурні схеми – зображують призначення та взаємодію основних модулів програми між собою, та з користувачем.

Застосунок для організації та менеджменту побуту є універсальною системою, що поєднує в собі функції планування, фінансового контролю, та нагадування про побутові справи, з можливістю встановлення пріоритетів.

Централізоване керування побутовими справами, та персоналізовані списки завдань включають в себе:

- Організацію та пріоритизацію завдань, використовуючи алгоритм організації часу, що базується на Матриці Стівена Кові (постановка пріоритетності: термінові, важливі, не термінові, не важливі), та категоризацію списку справ.
- Моніторинг виконання завдань з урахуванням дедлайнів, що дозволяє системі постійно моніторити статус виконання завдань, а алгоритм, що аналізує термін їх виконання, надає рекомендації шляхом надсилання сповіщень користувачеві.
- Фінансовий контроль, відстеження витрат, їх категоризація, це може відбуватися вручну, та автоматично, виявлення аномалії в транзакціях, прогнозування витрат, планування бюджету, та персоналізовані рекомендації для користувача, залежно від його рівня накопичень та витрат.
- Нагадування за побутові справи, які в себе включають догляд за будинком, рослинами, тваринами, транспортом, адаптацію до звичок користувача, що дозволяє пропонувати певні справи, якщо їх не було внесено до списку.

2.3 Розробка UML-діаграми класів

Уніфікована мова моделювання (UML) – стандарт для візуалізації, специфікації, побудови, та документування програмних систем, вона дозволяє зобразити різноманітні системи, тапредметні області невеликою діаграмою.

UML-діаграма класів – це статичне відображення структури моделів в рамках об’єктно-орієнтованого програмування, їх призначення – відобразити статичну структуру моделі, класи, інтерфейси, об’єкти, зв’язки. UML – діаграма створюється для оптимізації процеру розробки програмних систем, що дозволяє

збільшити ефективність та якість застосунку, знизивши при цьому витрату часу на розробку.

UML-діаграма класів та компонентів дозволяє розробнику:

- Чітко розуміти архітектуру системи, її основні класи, атрибути, методи та зв’язки.
- Логічно розподілити функціонал, розбивши систему на кілька модулів, які можуть розроблятися та тестуватися незалежно.
- Прогнозувати майбутні розширення чи зміни в функціоналі.
- Полегшити створення документації системи, що охоплює її структуру, поведінку, та зв’язки.

Діаграма класів представляє собою набір статичних та декларативних елементів моделі, вона дає найбільш розгорнуте уявлення про взаємозвя’зки, функціональність, та інформацію про окремі класи. Наприклад:

- Клас - об’єднує сутності зі спільними атрибутами та методами, кожен клас має власне ім’я, атрибут, та метод.
- Атрибут – описує характеристики та дані сутності.
- Метод – функція чи дія, яку може виконати клас.
- Зв’язок – показує взаємодію класів: один до одного, один до багатьох, багато до багатьох, , асоціативний зв’язок, агрегаційний зв’язок, композиційний зв’язок, спадкування, залежність.

До принципів розробки UML входять:

- Модульність – дозволяє розбити схему на модулі, що полегшує розробку та тестування застосунку.
- Документація та аналіз, що дозволяють полегшити виявлення проблем.
- Моделювання системи на різних етапах життєвого циклу програми.
- Використання структурних та поведінкових діаграм.
- Гнучкість для моделювання різних типів систем, та адаптації до потреб у разі розширення функціоналу.
- Стандартизована структура поведінки системи.

Для створення структурованої реалізації застосунку, було створено UML-дігаграму класів (рис. 2.3). Завдяки ній можна продемострувати зв’язки між ключовими елементами системи, та виокремити їх методи взаємодії, ознайомитися з логікою побудови застосунку.

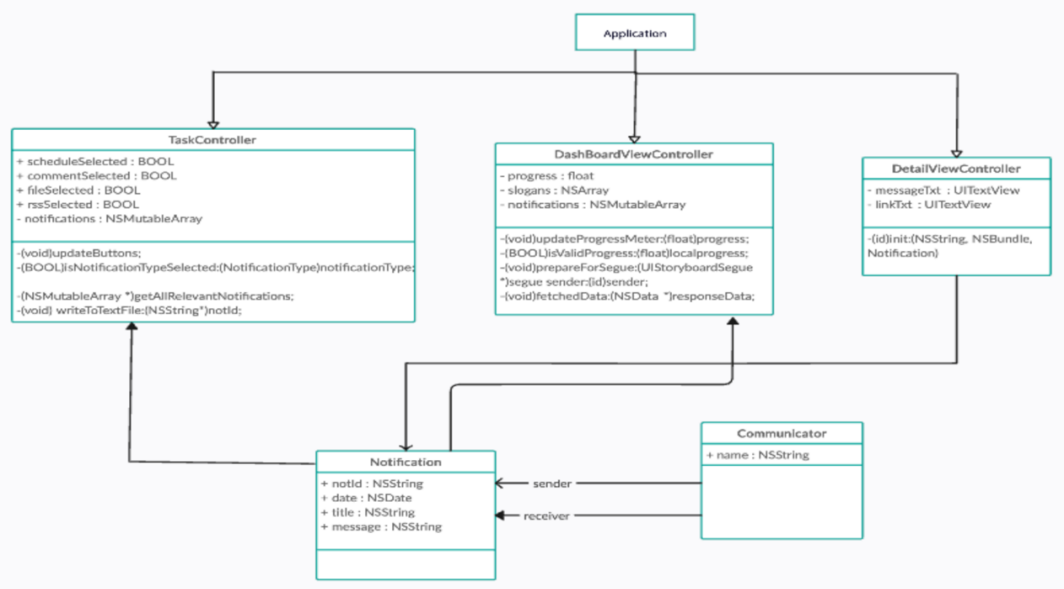

Рисунок 2.3 – UML-діаграма класів застосунку

На рисунку 2.3, в центрі структури знаходиться головний елемент, DashBoardViewController, який відповідає за відображення прогресу виконання завдань у графічному інтерфейсі користувача, він має поле progress, що відображає ступінь виконання завдань, та масиви slogans та notifications. Метод updateProgressMeter оновлює покажчик прогресу, а метод isValidProgress перевіряє його на кореектну роботу, метоод prepareForSegue готує дані перед переходом між екранами.

Коли користувач вводить інформацію, використовується клас TaskController, який відповідає за вибір типу завдання, з яким працює користувач, він має логічні прапорці, такі як розклад (дедлайн), коментар, файл, він також містить масив зі сповіщеннями notifications, та надає метод updateButtons, який оновлює стан інтерфейсу при додаванні чи редагуванні завдань. Метод

isNotificationTypeSelected, перевіряє чи вибрана опція з додатковим сповіщенням користувача про статус виконання завдання, а getAllRelevantNotifications повертає відфільтровані повідомлення, що відносяться до певного типу.

Клас DetailViewController відповідає за відображення окремих повідомлень.

Клас Notification є моделлю сповіщення, що включає в себе ідентифікатор, дату сповіщення, заголовок, та основний текст повідомлення.

Клас Communicator дозволяє ідентифікувати яке з завдань надіслало сповіщення про дедлайн, тощо.

Загалом, використання UML-діаграм класів є фундаментальним етапом у проектуванні сучасних інформаційних систем, оскільки вони забезпечують чітке, стандартизоване та візуальне представлення архітектури системи, що є критично важливим для ефективної розробки та підтримки програмного забезпечення.

2.4 Висновки до розділу 2

В даному розділі було розглянуто процес проектування IOS-застосунку для організації та менеджменту побутових потреб, було визначено структуру майбутньої системи, алгоритм її функціонування, та побудовано UML-діаграму класів, що забезпечує наочне бачення архітектури та логіки роботи застосунку.

Створення логічної структури застосунку із розділенням на модулі – керування завданнями, фінансами, побутовими справами, та система сповіщень, це ті аспекти, що забезпечують покращений користувацький досвід, такий підхід допомагає знизити кількість залежностей, тип самим підвищивши стійкість до критичних помилок, при внесенні змін до окремих модулів.

Алгоритм роботи системи було реалізовано на основі принципів користувацької зручності, а саме на логіці додавання, редагування, та пріоритизація завдань, контролю фінансів, завдяки сучасним принципам тайм-менеджменту було сформовано персоналізований користувацький досвід. Добре спроектований алгоритм забезпечує надійне функціонування застосунку.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ДЛЯ ОРГАНІЗАЦІЇ ТА МЕНЕДЖМЕНТУ ПОБУТОВИХ ПОТРЕБ

3.1 Обґрунтування вибору мови програмування, бібліотек та підходу до реалізації

Вибір мови програмування, середовища розробки, бібліотек, та підходу до реалізації є найважливішим етапом розробки застосунку, вони впливають на продуктивність системи, її масштабованість, легкість підтримки, швидкість та ефективність процесу розробки застосунку. Сучасні застосунки потребують обробки даних у реальному часі, взаємодії з базами даних, тому важливо врахувати не лише технічні характеристики пристрою користувача, а й можливість інтеграції додаткових елементів.

Критерії вибору технології для розробки застосунку:

- Продуктивність та швидкодія, для забезпечення обробки великих обсягів даних.
- Масштабованість системи, здатність адаптуватися до зростаючих потреб користувача, без суттєвих змін архітектури застосунку.
- Наявність інструментів розробки, фреймворків, та бібліотек.
- Зручність розробки та підтримки застосунку завдяки зрозумілому синтаксису, якісній документації, та підтримці об’єктно-орієнтованого програмування.
- Наявність спільноти розробників для обміну досвідом.
- Безпечність системи, при роботі з конфіденційними даними.
- Кросплатформенність, можливість розгорнути окрім мобільної версії застосунку, ще версію на стаціонарний комп’ютер, тощо.

Вибір мови програмування (таб 3.1) [7]:

- C# - об’єктно орієнтована мова програмування, широко застосовується на персональних комп’ютерах, має багату екосистему, багатопоточність, автоматичне керування пам’яттю, статичну типізацію.

- JavaScript / TypeScript – об’єктно орієнтовані мови програмування, кросплатформенні, динамічна та статична, мають багато фреймворків, дуже популярні, та мають малий поріг входу.
- Python – високорівнева мова, відома своєю розповсюдженістю, проста, має широкий спектр застосувань, та має легкий синтаксис.
- Java – суворо типізована об’єктно орієнтована мова програмування, кросплатформенна, має автоматичне керування пам’яттю, надійна, безпечна, має багато бібліотек, та велику спільноту розробників.
- Swift / SwiftUI – підтримує функціональне та об’єктно орієнтоване програмування, має статичну типізацію, автоматичне керування пам’яттю, інтегрована лише з пристроями на платформі Apple.

Таблиця 3.1 – Порівняння мов програмування

Мова програмування	Типізація	Парадигма	Кросплатформенність
C#	Статична	Об’єктно орієнтована	Переважно Windows
JavaScript / TypeScript	Динамічна / Статична	Об’єктно орієнтована	Кросплатформенна
Python	Динамічна	Багатопарадигмова	Кросплатформенна
Java	Статична	Об’єктно орієнтована	Кросплатформенна
Swift / SwiftUI	Статична	Об’єктно орієнтована та функціональна	Тільки екосистема Apple

Вибір середовища розробки:

- Microsoft Visual Studio / Visual Studio Code – підтримують багато мов програмування, мають зручні інтерфейси, легкі в налаштуванні, та забезпечують легку розробку проектів.
- Xcode – має зручний інтерфейс, вбудований симулятор пристроїв, інтелектуальну підсвітку синтаксису, та автодоповнення, інструменти для тестування, контролю версій, та миттєвого прев'ю інтерфейсу.

Методи реалізації архітектури системи (таб. 3.2) [8]:

- Клієнт-серверна архітектура, що розподілена між клієнтом (інтерфейс), та сервером (логіка, база даних), дозволяє легко масштабуватися, убезпечує дані, дозволяє незалежно розробляти та оновлювати систему, ефективна обробка даних.
- Мікросервісна архітектура, розподіляє застосунок на маленькі автономні сервіси, кожен з яких відповідає за певну функцію, і має окрему базу даних, це полегшує швидкість розгортання, але ускладнює управління, та тестування застосунку.
- Монолітна архітектура – всі компоненти розробляють та розгортаються як одне ціле, це полегшує розробку та тестування на початку, але ускладнює підтримку застосунку вподальшому.
- Модульна архітектура – розбиття застосунку на менші, незалежні модулі, кожен з яких обробляє певну частину більшої програми, це дозволяє легше тестувати застосунок.
- Model-View-ViewModel (MVVM) – розділяє відповідальності між моделлю (дані та логіка), інтерфейсом користувача, та обробкою подій з взаємодією між моделлю та інтерфейсом (ViewModel).

Таблиця 3.2 – Порівняння архітектур розробки

Архітектура	Опис	Переваги	Недоліки
Клієнт-серверна	Розподілена на клієнтську частину з інтерфейсом, та серверну з логікою та базою даних	Масштабованість, безпека даних, незалежна розробка, ефективна обробка записів	Потрібне стабільне з’єднання, складна синхронізація змін
Мікросервісна	Поділена на окремі сервіси, кожен з яких виконує одну фінкцію, і має власну базу даних	Швидке розгортання, гнучкість масштабування, незалежність модулів	Висока складність керування, та взаємодії між сервісами
Монолітна	Усі компоненти системи реалізовані як одне ціле	Проста на початковому етапі, зручне тестування	Ускладненість під час масштабування
Модульна	Складається з окремих модулів, які відповідають за певні функції, й можуть бути оновлені окремо	Легке тестування, повторне використання модулів	Потребує продуманого проектування залежностей між модулями

Продовження таблиці 3.2 – Порівняння архітектур розробки

Архітектура	Опис	Переваги	Недоліки
Model-View-ViewModel (MVVM)	Розділяє програму на дані й логіку, інтерфейс, та посередника, що обробляє взаємодію між ними	Чітке розділення відповідальності, покращене тестування логіки	Надмірна складність для простих задач

В даній таблиці зображено порівняння архітектурних моделей, залежно від потреб та масштабів проекту.

Вибір бази даних:

- Реляційна база даних – часто використовується для зберігання структурованих даних.
- Ієрархічна база даних – дані організовані у вигляді дерева, що дозволяє швидко отримати доступ до даних, але має низьку масштабованість.

3.2 Розробка програмного застосунку

Застосунок розбито на велику кількість під-модулів для оптимізації роботи, та розробки, також завдяки SwiftUI, маємо ефективне оновлення інтерфейсу користувача в режимі реального часу, при внесенні будь-яких змін [9].

Структура проекту складається з 16 файлів.

MainView.swift – головний файл запуску програми.

ContentView.swift – графічний інтерфейс застосунку.

TaskModel.swift, TaskRowView.swift, EditTaskView.swift, AnalyticsView.swift, AddTaskView.swift – файли, що відповідають за екран з списком завдань.

FinanceView.swift, TransactionChartView.swift, TransactionModel.swift, PaymentBreakdownView.swift, AddOrEditTransactionView.swift – файли, що відповідають за екран з менеджментом фінансів.

HouseHoldModel.swift, HouseholdListView.swift, AddEditPlantView.swift, AddEditPetView.swift – файли, що відповідають за екран з побутом.

Розглянемо файл «MainView.swift», який є серцем застосунку, та забезпечує взаємодію користувача з модулями системи організації та менеджменту побутових потреб:

```
NavigationView {
  VStack {
    HStack {
      Text("Мій список справ")
        .font(.largeTitle)
        .padding(.top)
      Spacer()
      Button(action: {
        showingStats = true
      }) {
        Image(systemName: "chart.pie")
          .font(.title2)
          .padding(8)
      }
    }
    .padding(.horizontal)
```

Основний інтерфейс реалізовано задопомогою контейнера NavigationView, який дає можливіть переходу між вікнами, забезпечуючи правильне розташування елементів, списків, кнопок.

```
    ScrollView {
      ForEach($tasks) { $task in
        NavigationLink(destination: EditTaskView(task: $task)) {
          TaskRowView(task: $task)
            .padding(.vertical, 5)
        }
      }
```

```
            .padding()
        }
        HStack {
            Button("ФІНАНСИ") {
                showingFinance = true
            }
            .font(.headline)
            .frame(maxWidth: .infinity)
            .padding()
            .background(Color.blue.opacity(0.2))
            .cornerRadius(12)
            Button(action: {
                showingAddTask = true
            }) {
                Image(systemName: "plus")
                    .font(.system(size: 24))
                    .frame(width: 50, height: 50)
                    .background(Color.green)
                    .foregroundColor(.white)
                    .clipShape(Circle())
                    .shadow(radius: 5)
            }
            .padding(.horizontal)
            Button("ПОБУТ") {
                showingHousehold = true
            }
            .font(.headline)
            .frame(maxWidth: .infinity)
            .padding()
            .background(Color.orange.opacity(0.2))
            .cornerRadius(12)
        }
        .padding()
    }
    .sheet(isPresented: $showingAddTask) {
        AddTaskView(tasks: $tasks)
    }
    .sheet(isPresented: $showingStats) {
```

```
            AnalyticsView(tasks: tasks)
        }
        .sheet(isPresented: $showingFinance) {
            FinanceView()
        }
        .sheet(isPresented: $showingHousehold) {
            HouseholdListView(pets: $pets, plants: $plants)
        }
        .onAppear {
            loadTasks()
            loadHousehold()
            UNUserNotificationCenter.current().requestAuthorization(options:
[.alert, .sound, .badge]) { granted, _ in
                if granted {
                    print("□ Дозвіл на сповіщення надано")
                }
            }
        }
        .onChange(of: tasks) { _ in
            saveTasks()
        }
        .onChange(of: pets) { _ in
            saveHousehold()
        }
        .onChange(of: plants) { _ in
            saveHousehold()
        }
    }
```

Завдяки вбудованим методам створення сповіщень, в цьому блоці можемо створити перевірку на дозвіл надсилання сповіщень, та вибрати дату й час, коли це сповіщення повинне бути доставлене [10].

```
    NavigationView {
        Form {
            Section(header: Text("Назва завдання")) {
                TextField("Введіть назву", text: $title)
            }
```

```
                Section(header: Text("Теги")) {
                    TextField("Напр. 'прання, фінанси'", text: $tags)
                }
                Section(header: Text("Дедлайн")) {
                    DatePicker("Оберіть дату і час", selection: $deadline, displayedComponents: [.date, .hourAndMinute])
                }
                Section {
                    Toggle(isOn: $hasNotification) {
                        Text("Надіслати сповіщення")
                    }
                }
            }
            .navigationTitle("Нове завдання")
            .navigationBarItems(trailing: Button("Зберегти") {
                if !title.isEmpty {
                    let newTask = Task(title: title, tags: tags, deadline: deadline, hasNotification: hasNotification, isCompleted: false)
                    tasks.append(newTask)
                    if hasNotification {
                        scheduleNotification(for: newTask)
                    }
                    presentationMode.wrappedValue.dismiss()
                }
            })
        }
    }
```

У цьому блоці відображатимемо створення завдань, вибір назви завдання, створення тегів, вибір дати за допомогою компонента DatePicker, що дозволяє вибрати точну дату й час виконання завдання, та налаштування сповіщення користувачеві, завдяки перемикачеві.

```
Text("Виконано \(completedCount) із \(totalCount) справ")
    .font(.headline)
Text("\(Int(completionPercentage))%")
    .font(.title)
    .padding()
```

```
            .frame(width: 100, height: 50)
            .background(percentageColor.opacity(0.2))
            .foregroundColor(.primary)
            .clipShape(Capsule())
        if completionPercentage == 100 {
            Text(motivationalMessage)
                .font(.headline)
                .foregroundColor(.green)
                .transition(.opacity)
        }
```

У розділі зі статистикою виконання завдань, реалізовано графічне відображення, яке оновлюється динамічно, в залежності від прогресу користувача, і при досягненні певного відсотку виконання завдань, демонструє користувачеві мовиваційне повідомлення. Приклад мотиваційного повідомлення:

```
var motivationalMessage: String {
    let emojis = ["□", "□", "□", "□", "□", "□", "□"]
    return "Супер, Ти молодець! " + (emojis.randomElement() ?? "□")
}
```

Перейдемо до наступного екрану, з фінансами, де користувачеві доступна створення транзакцій, з його подальшим редагуванням, та відслідковування співвідношення витрат карткового до готівкового [11].

Основні змінні стану, які забезпечують збереження усіх транзакцій, та модальні вікна, що мають бути показані користувачеві:

```
@State private var transactions: [Transaction] = []
@State private var showAddTransaction = false
@State private var showBreakdown = false
@State private var showChart = false
@State private var selectedTransaction: Transaction?
```

Основний блок обрахування, та зміни балансу дозволяє формувати місячний баланс, відслідковувати скільки внесено, витрачено, та залишилося

коштів у користувача, проведення розділу на картоковий та готівковий розрахунок за необхідності, та створення окремого графіку для відслідковування помісячних витрат [12]:

```
Text("Баланс за місяць:")
   .font(.headline)
Text("Отримано: \(income, specifier: "%.2f") грн")
Text("Витрачено: \(expense, specifier: "%.2f") грн")
Text("Залишок: \((income - expense), specifier: "%.2f") грн")
   .fontWeight(.bold)
```

Перейдемо до наступного екрану з побутом, на ньому користувач може додати домашніх тварин, їх імена, дати народження, та щеплення, також користувач може додати рослини, їх назву, дату останнього поливу, та останнього підживлення.

```
if !pets.isEmpty {
   Section("Тварини") {
      ForEach(pets) { pet in
         Button(action: { selectedPet = pet }) {
            VStack(alignment: .leading) {
               Text(pet.name).font(.headline)
               Text("Народилася: \(pet.birthDate, style: .date)")
                  .font(.caption)
            }
         }
      }
      .onDelete { pets.remove(atOffsets: $0) }
   }
}
```

Блок, що дозволяє виводити список тварин, кожен запис в якому представлений як кнопка, яка при натисканні відкриє форму для редагування selectedPet, та аналогічний блок для домашніх рослин який відкриває форму редагування через selectedPlant [13].

```
if !plants.isEmpty {
```

```
        Section("Рослини") {
            ForEach(plants) { plant in
                Button(action: { selectedPlant = plant }) {
                    VStack(alignment: .leading) {
                        Text(plant.name).font(.headline)
                        Text("Останній полив: \(plant.lastWatered, style: .date)")
                            .font(.caption)
                    }
                }
            }
            .onDelete { plants.remove(atOffsets: $0) }
        }
    }
```

Готовий застосунок гарантує повний функціонал, для створення та відслідковування виконання завдань, догляду за домашніми тваринами та рослинами, що дозволяє тримати стан їх здоров’я завжди на гарному рівні, а також облік фінансів, із зручним візуальним інтерфейсом. Застосунок може злегкістю масштабуватися, та можуть бути додані нові функції.

3.3 Тестування роботи програмного застосунку

Після реалізації основного функціоналу застосунку, було проведено тестування, перевірка коректності створення завдань, їх збереження в пам’яті, можливість редагування завдань, зміни дедлайну, додання тегів. Тестування охоплювало як і створення декількох завдань, так і кількох десятків завдань, це дозволило переконатися у стійкості алгоритму, та здатності програми зберігати та обробляти велику кількість різних типів вхідних даних, тим самим забезпечуючи правильне функціонування застосунку [14].

Після запуску застосунку ОМРР, користувача зустрічає вхідна форма, де користувач може бачити список своїх справ (рис 3.1).

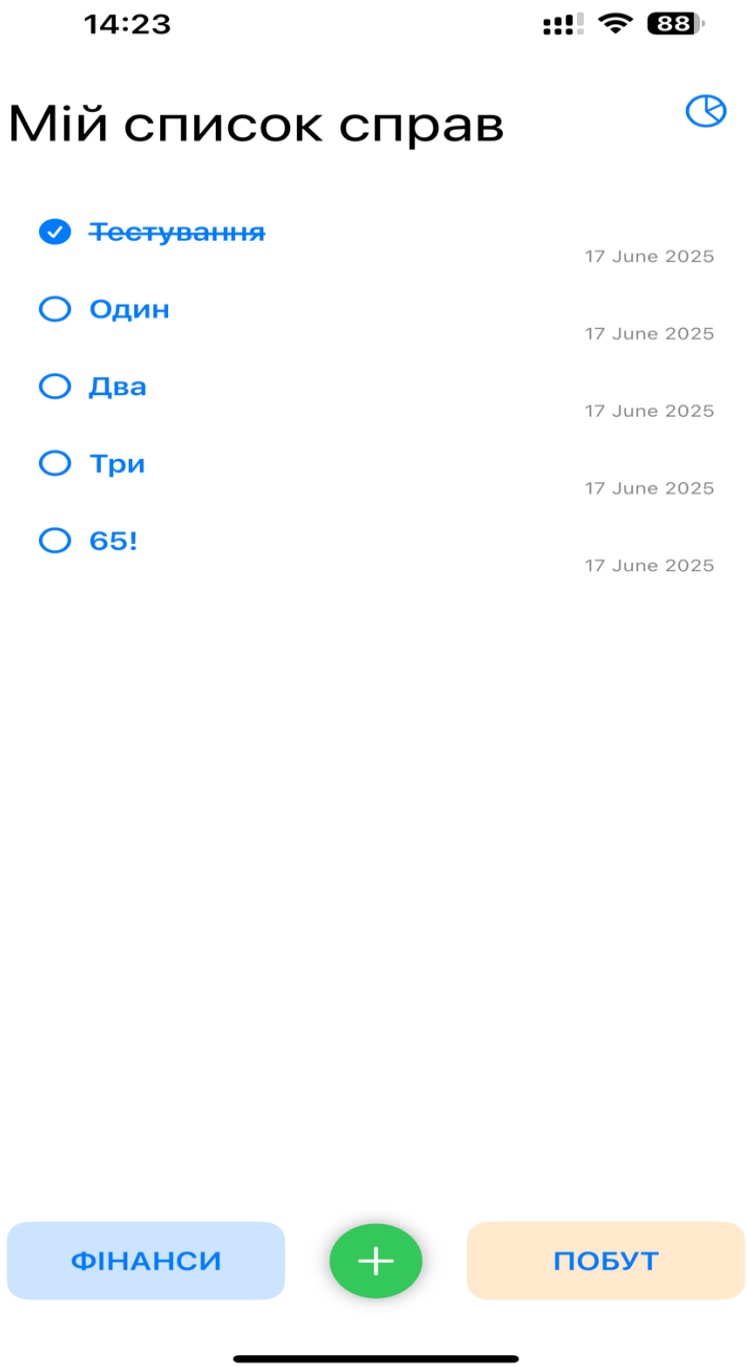

Рисунок 3.1 – Головне меню застосунку

На рисунку 3.2 зображено вхідні дані для створення завдання в застосунку, в ньому наведено назву, тег, час виконання, та наявність сповіщення.

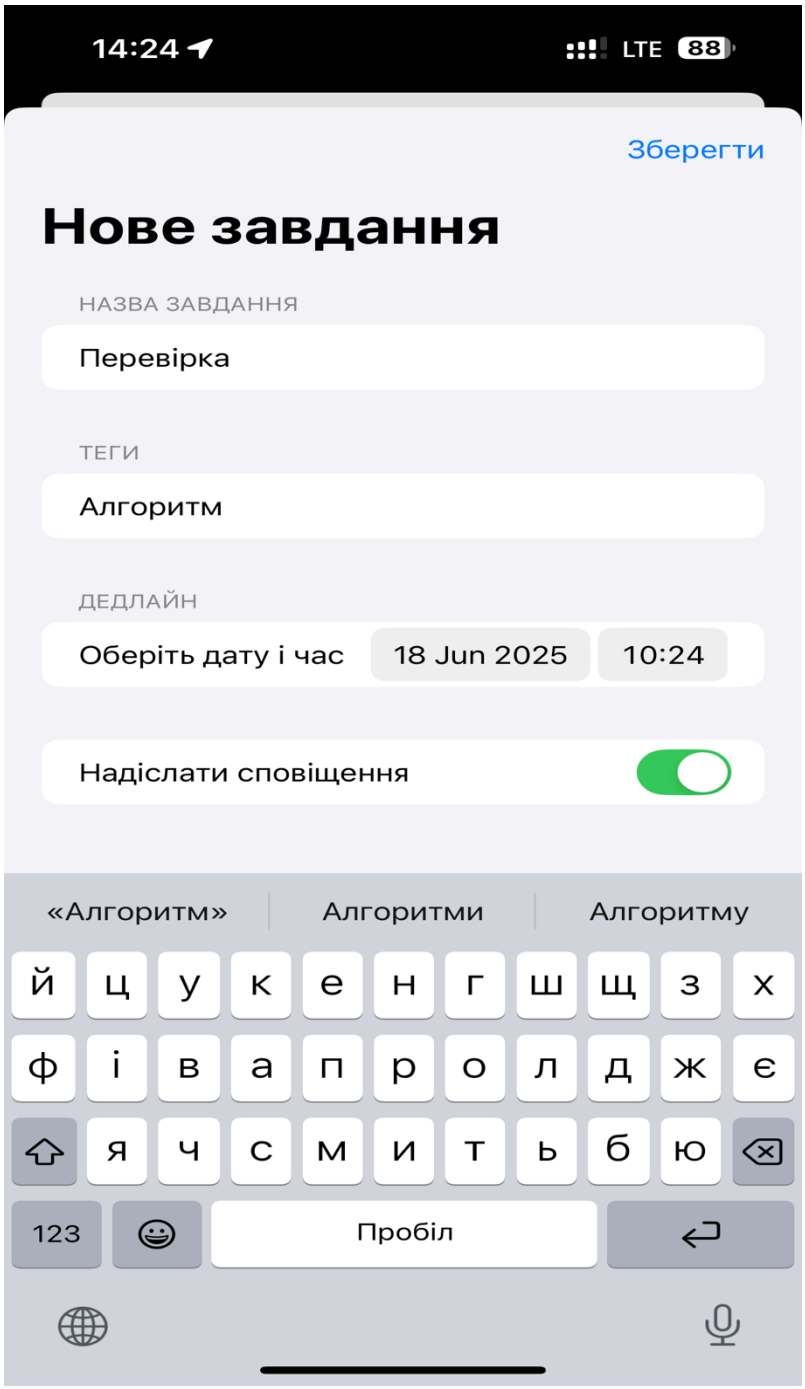

Рисунок 3.2 – Меню створення нового завдання

На рисунку 3.3 зображений результат виконання функції зі створення нового завдання.

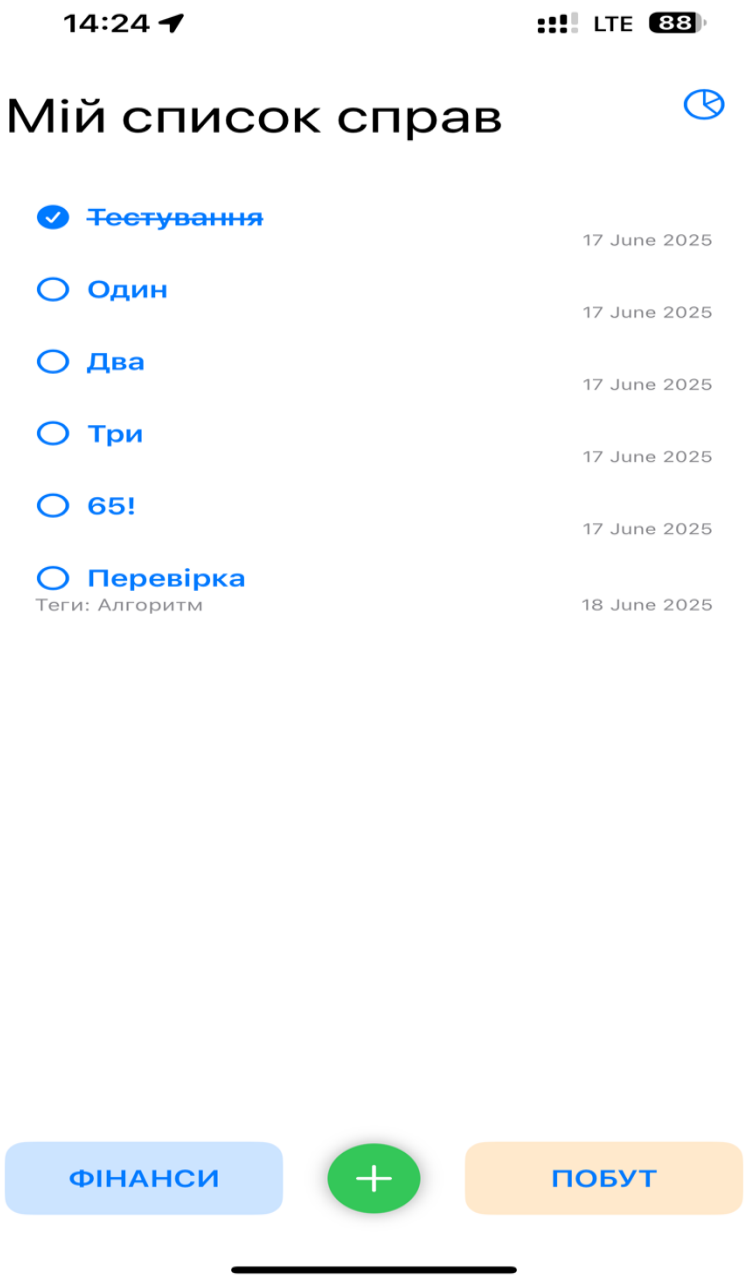

Рисунок 3.3. – Результат створення нового завдання

На рисунку 3.4 зображений модуль фінансів, який використовується для внесення витрат та доходів, та перевірки за типом витрат.

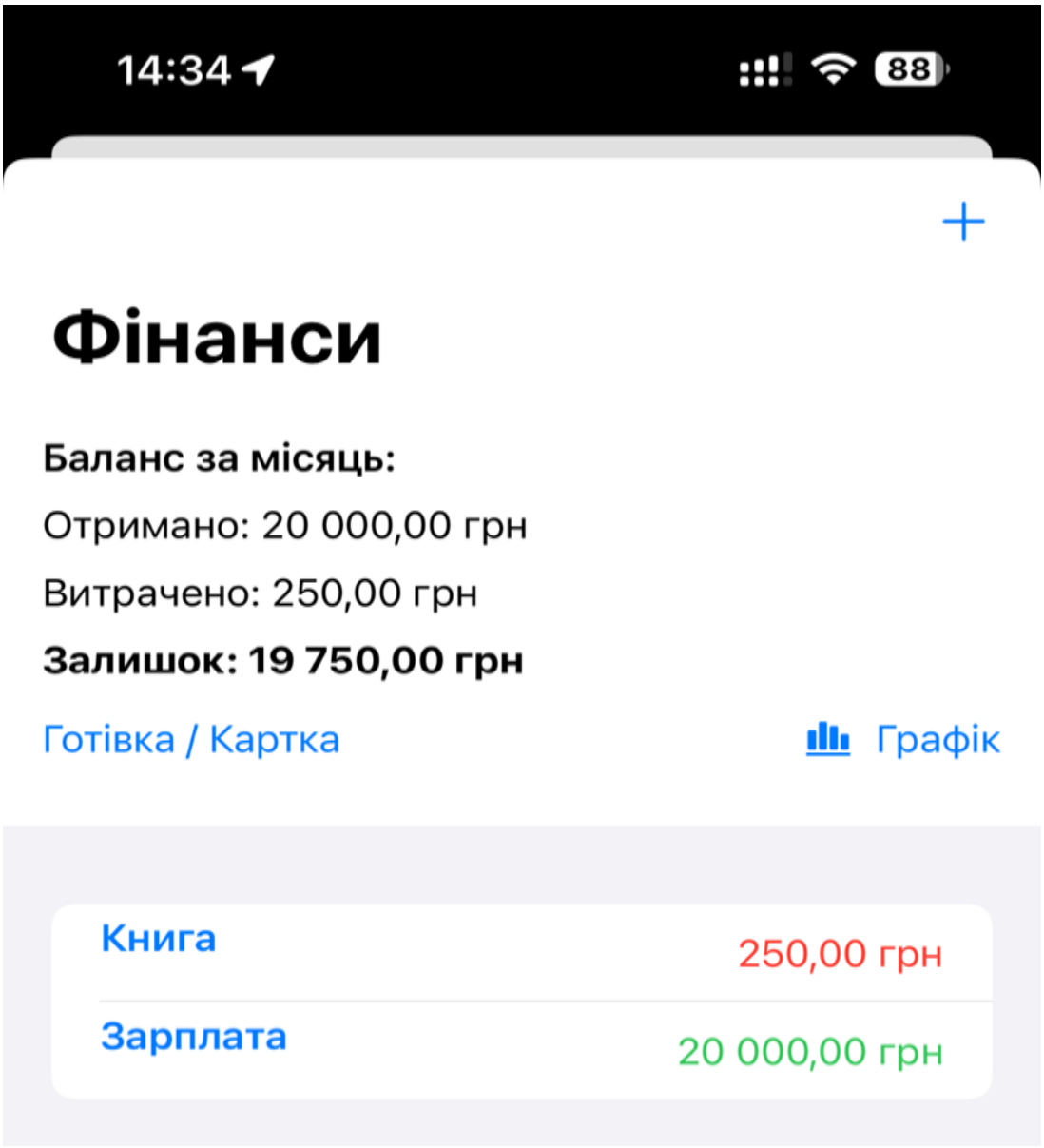

Рисунок 3.4 – Вікно з ведення фінансового обліку

На рисунку 3.5 зображене вікно зі створення нової транзакції, яку було використано для перевірки достовірності роботи модуля

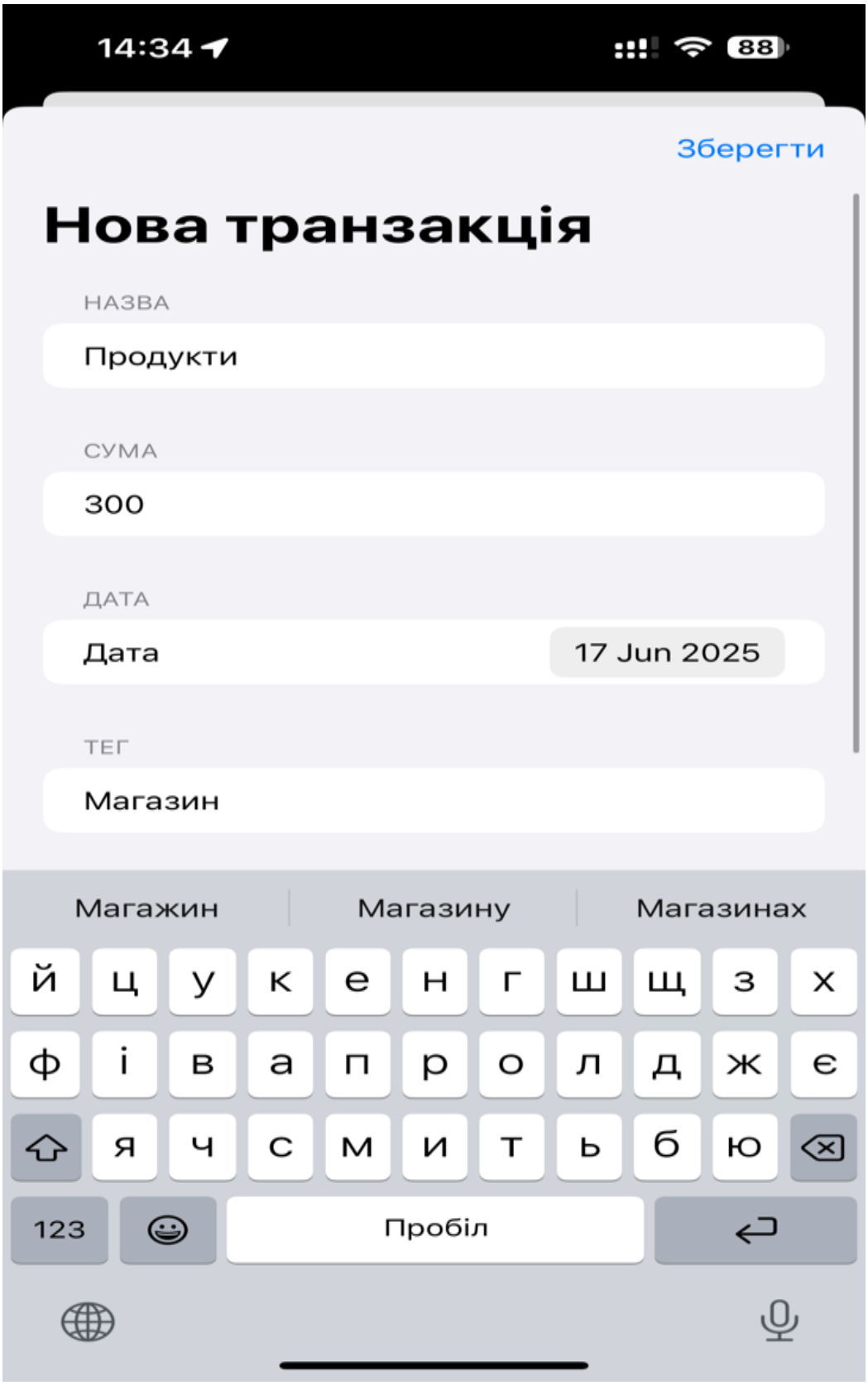

Рисунок 3.5 – Вікно створення нової транзакції

На рисунку 3.6 зображений результат виконання функції зі створення нових фінансових витрат та доходів.

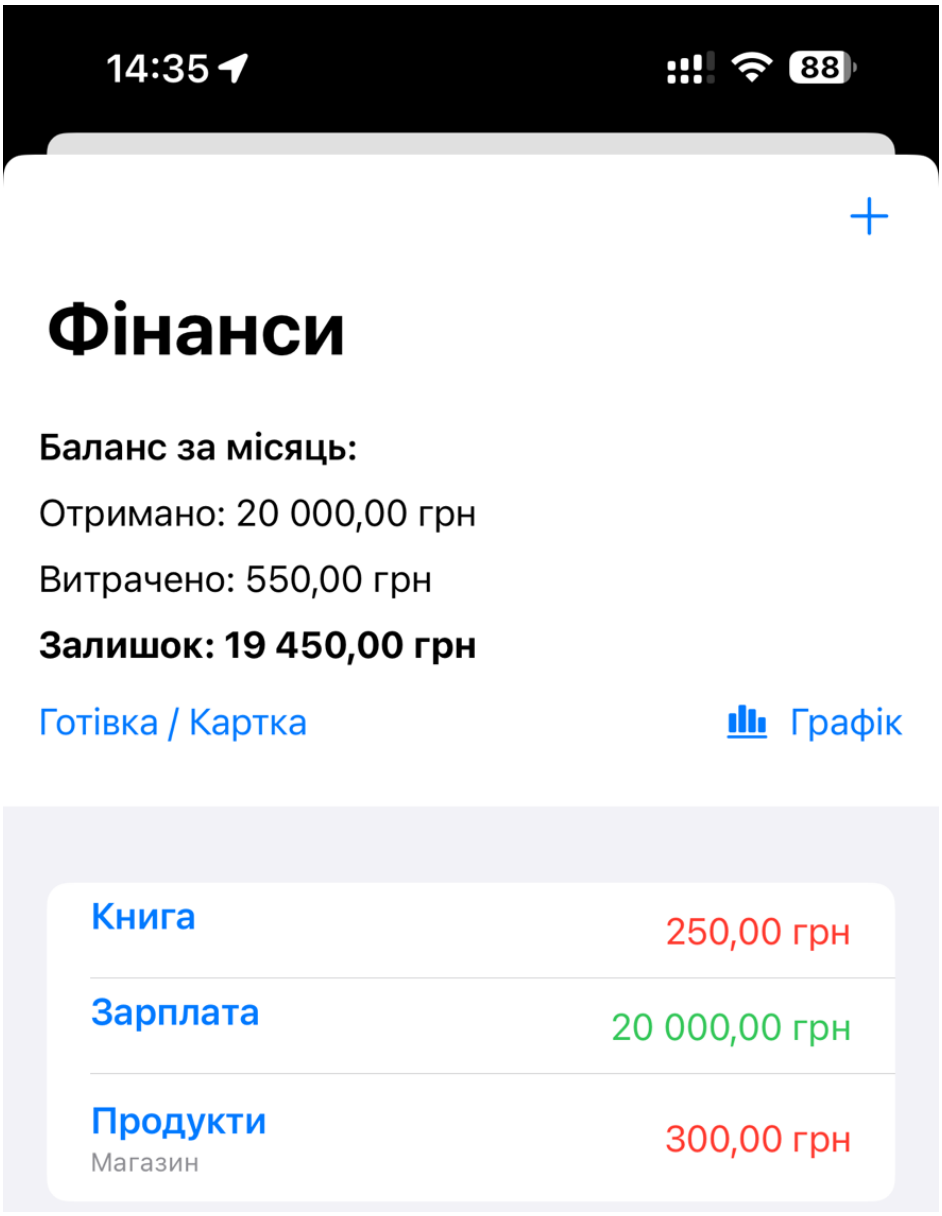

Рисунок 3.6 – Результат виконання функції обліку фінансів

Третій модуль, що стосується побуту, дозволяє користувачеві додати своїх домашніх улюбленців, та домашні рослини, і відслідковувати їх стан.

Для тварин можна відслідковувати їхні щеплення, для рослин дату їх поливу та підживлення. На рисунку 3.7 зображено процес створення картки тварини.

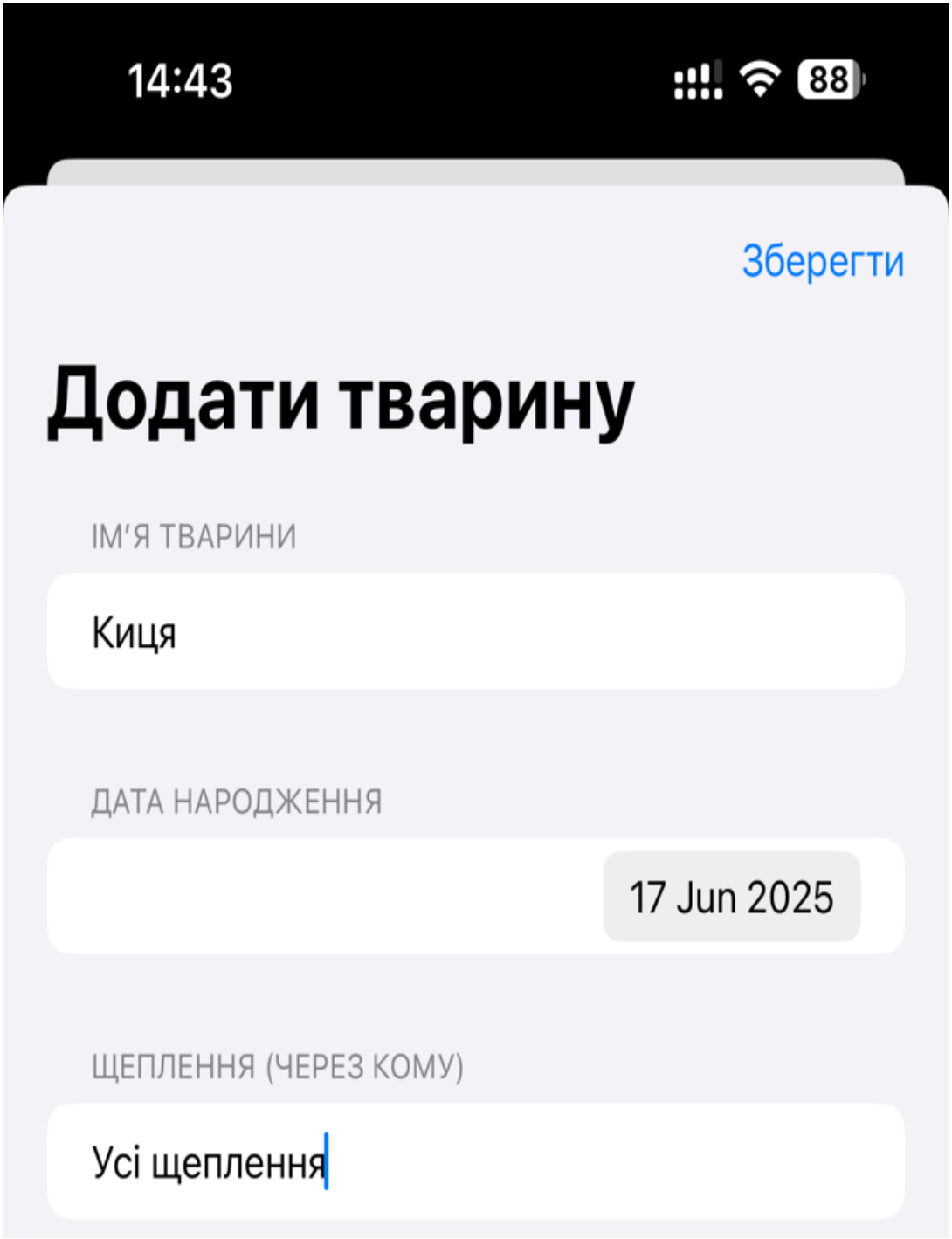

Рисунок 3.7 – Створення картки тварини

На рисунку 3.8 зображено результат виконання функції з додавання картки тварини.

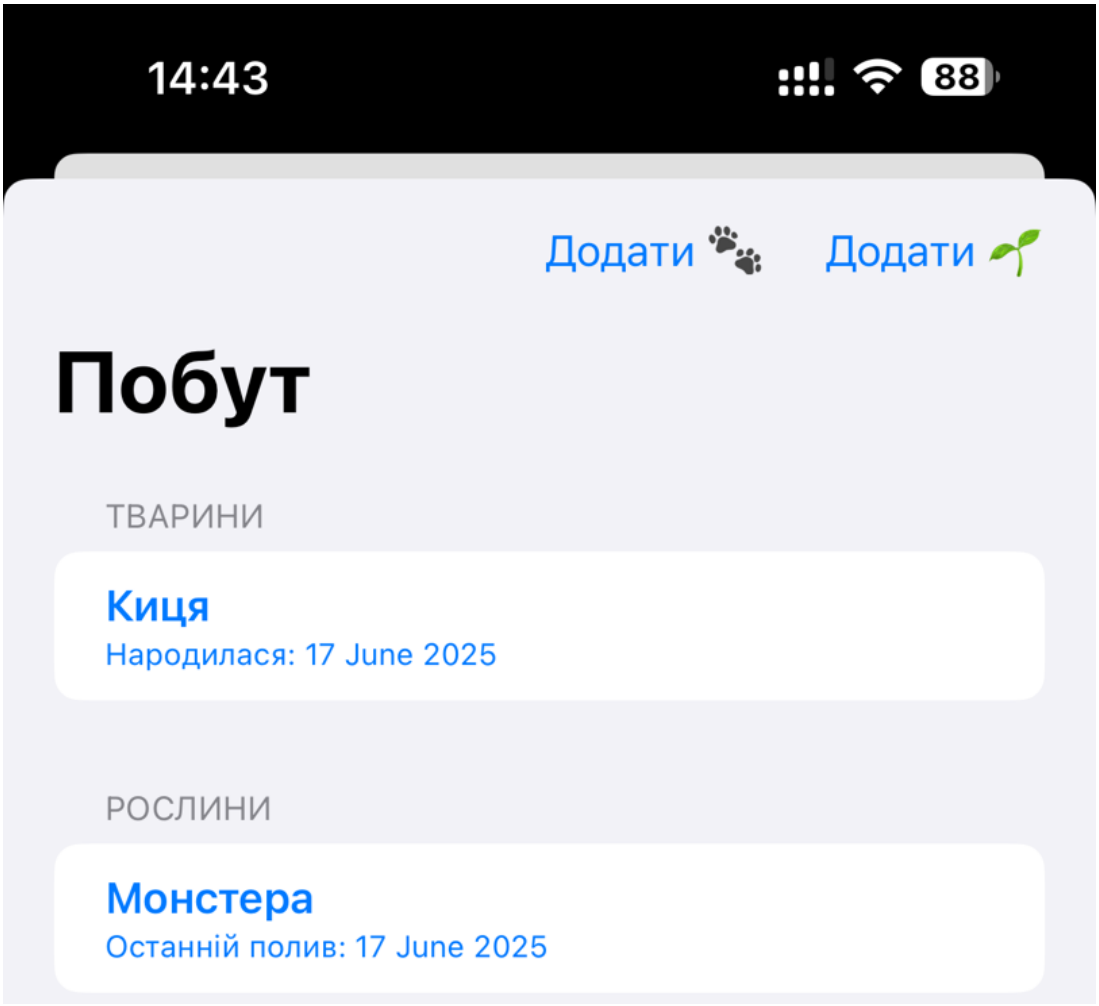

Рисунок 3.8 – Результат виконання функції

3.4 Аналіз результатів тестування

У процесі проектування та створення застосунку для організації та менеджменту побутових потреб, важливо порівняти його функціональність з існуючими програмними рішеннями. Для порівняння було розглянуто кілька популярних рішень (таб 3.1).

Таблиця 3.3 – порівняння програмних рішень на ринку

Назва	Методологія	Ключові функції
Trello	Kanban	Дошки, картки, дедлайни, теги, інтеграції
Asans	Agile / Kanban	Завдання, підзавдання, календар, таймлайн, шаблони
Microsoft To Do	To-Do	Списки, підзадачі, нагадування
OMPP	To-Do	Списки, завдання, теги, дедлайни, облік фінансів, догляд за побутом

Переваги розробленого застосунку [15]:

- Має зручний та мінімалістичний інтерфейс.
- Забезпечує оптимальний режим відслідковування виконання завдань.
- Можна використовувати офлайн, що є перевагою, порівняно з деякими популярними рішеннями.
- Підвищена продуктивність, яка допомагає швидко проводити дії з даними.

Недоліки розробленого застосунку:

- Для певного прошарку користувачів, обмежений функціонал, навідміну від багатовартісних продуктів, які є вузькоспеціалізованими застосунками.

3.5 Висновки до розділу 3

В даному розділі було проведено ґрунтовний вибір мови програмування, та архітектури підходу, що стало основою для реалізації застосунку. Після аналізу наявних мов для розробки, було вибрано Swift та SwiftUI, які завдяки своєму потужному середовищу, та розвиненій мові, дозволяють з мінімальною кількістю сторонніх бібліотек створити застосунок, з підтримкою додавання, редагування, видалення елементів, проведення швидкої аналітики виконання завдань. Завдяки динамічному оновленню інтерфейсу, покращується користувацький досвід, та через наявність сповіщень в системі, користувач має змогу гнучко, та швидше виконувати завдання.

З технічної сторони, застосунок розроблений з використанням модульної архітектури, що забезпечує можливість масштабування програми в майбутньому без вагомих зусиль, та дозволяє легше проводити тестування окремих функцій, без ризику зламати систему [16].

Розроблений застосунок було порівняно з існуючими на ринку рішеннями, виявлено що цей застосунок має переваги в певному напрямку, а саме всеохопленість побутових справ в одному застосунку, що дозволяє пришвидшити організацію та менеджмент справ.

ВИСНОВКИ

Поставлені перед бакалаврською кваліфікаційною роботою задачу було виконано в повному обсязі, а саме:

Провести аналіз предметної області, існуючих методів, алгоритмів та інструментів для організації та менеджменту побутових потреб.

Розробка алгоритму функціонування застосунку для організації та менеджменту побутових потреб.

Програмна реалізація застосунку для організації та менеджменту побутових потреб за допомогою мови програмування Swift.

Тестування застосунку для організації та менеджменту побутових потреб.

Розробка інструкції користувача застосунку для організації та менеджменту побутових потреб.

Для розробки була використана мова програмування Swift, було розроблено структуру IOS-застосунку для організації та менеджменту побутових потреб, створено UML-діаграму класів, схему алгоритму роботи. Розроблений застосунок після тестування й порівняння з існуючими рішеннями показав себе успішно, отримавши 4 переваги, що суттєво підвищує користувацький досвід від використання застосунку.

Мета роботи розширення функціональних можливостей застосунку, була досягнена за рахунок уніфікації системи менеджменту побуту, догляду за домашніми тваринами та рослинами, облік фінансів, що дозволяє користувачеві швидко вести облік витрат та доходів [17].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 4 квадранти тайм-менеджменту [Електронний ресурс] // Ігор Ріпа. – 2024. – Режим доступу: https://pg-group.online/4-kvadranty-tajm-menedzhmentu/ – Назва з екрана.
2. Scrum vs Agile vs Kanban: що обрати? [Електронний ресурс] // EPAM. – 2023. – Режим доступу: https://campus.epam.ua/ua/blog/577 - Назва з екрана.
3. Microsoft To Do [Електронний ресурс] // Microsoft. – 2025. – Режим доступу: https://www.microsoft.com/uk-ua/microsoft-365/microsoft-to-do-list-app - Назва з екрана.
4. БАЗОВІ ПРАВИЛА ДОГЛЯДУ ЗА КІМНАТНИМИ КВІТАМИ [Електронний ресурс] // LoraShen. – 2024. – Режим доступу: https://lorashen.ua/publication/basic-rules-for-caring-for-indoor-flowers/?srsltid=AfmBOooHrPhQQDJAZRtt3_llnMGt7VeibdeRstXShPlE_QHgVaNMnsH_ - Назва з екрана.
5. Фінанси, гроші та кредит: теорія та практика. Навчальний посібник рекомендовано МОН України / Коваленко Д.І. – Київ: Центр навчальної літератури (ЦНЛ), 2019. – 578 с.
6. What is a UML diagram? [Електронний ресурс] // Miro. – 2025. – Режим доступу: https://miro.com/diagramming/what-is-a-uml-diagram/ - Назва з екрана.
7. DOU. Рейтинг мов програмування [Електронний ресурс] // DOU. – 2025. – Режим доступу: https://dou.ua/lenta/articles/language-rating-2025/ - Назва з екрана.
8. Математичні методи дослідження операцій. Лінійне програмування. Частина 1 : навчальний посібник / Яровий А. А., Ваховська Л. М., Крилик Л. В. – Вінниця : ВНТУ, 2020. – 86 с.
9. Winston W. L. Operations Research: Applications and Algorithms. — 4th ed. — Cengage Learning, 2004. — 1432 с.

10. Apple Inc. The Swift Programming Language (Swift 5.7) [Електронний ресурс] // Apple Inc. – 2014. – Режим доступу: https://books.apple.com/us/book/the-swift-programming-language-swift-5-7/id881256329 – Назва з екрана.
11. Apple Inc. The Swift Programming Language (6.1) [Електронний ресурс] // Apple Inc. – 2023. – Режим доступу: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/ – Назва з екрана.
12. Apple Inc. SwiftUI Documentation [Електронний ресурс] // Apple Inc. – 2025. – Режим доступу: https://developer.apple.com/documentation/swiftui/ – Назва з екрана.
13. freeCodeCamp.org. Swift Programming Tutorial – Full Course for Beginners [Електронний ресурс] // freeCodeCamp.org. – 2022. – Режим доступу: https://www.youtube.com/watch?v=8Xg7E9shq0U – Назва з екрана.
14. Sean Allen. SwiftUI Fundamentals | FULL COURSE | Beginner Friendly [Електронний ресурс] // Sean Allen. – 2023. – Режим доступу: https://www.youtube.com/watch?v=b1oC7sLIgpI – Назва з екрана.
15. Swift Pros and Cons: Building an iOS App [Електронний ресурс] – Режим доступу до ресурсу: https://www.netguru.com/blog/building-ios-app-with-swift
16. Xcode IDE Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/xcode/
17. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп’ютерні науки» [Електронний ресурс] / уклад.: А. А. Яровий, О. В. Сілагін, І. В. Богач. – Вінниця : ВНТУ, 2023. – (PDF, 54 с.).

ДОДАТОК А (ОБОВ'ЯЗКОВИЙ)
ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка IOS-застосунку для організації та менеджменту побутових потреб

Тип роботи: бакалаврська кваліфікаційна робота

(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук

(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 0,41 %

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:

Яровий А.А., зав. каф. КН _____ _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Сілагін Є.О.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Захаревич А.М.
(підпис) (прізвище, ініціали)

ДОДАТОК Б (ОБОВ'ЯЗКОВИЙ)
ЛІСТИНГ ПРОГРАМИ

```
# MainView.swift
import SwiftUI
import UserNotifications

struct MainView: View {
    @State private var tasks: [Task] = []
    @State private var showingAddTask = false
    @State private var showingStats = false
    @State private var showingFinance = false
    @State private var showingHousehold = false

    @State private var pets: [Pet] = []
    @State private var plants: [Plant] = []

    var body: some View {
        NavigationView {
            VStack {
                HStack {
                    Text("Мій список справ")
                        .font(.largeTitle)
                        .padding(.top)
                    Spacer()
                    Button(action: {
                        showingStats = true
                    }) {
                        Image(systemName: "chart.pie")
                            .font(.title2)
```

```
            .padding(8)
    }
}
.padding(.horizontal)

ScrollView {
    ForEach($tasks) { $task in
        NavigationLink(destination: EditTaskView(task: $task)) {
            TaskRowView(task: $task)
                .padding(.vertical, 5)
        }
    }
    .padding()
}

HStack {
    Button("ФІНАНСИ") {
        showingFinance = true
    }
    .font(.headline)
    .frame(maxWidth: .infinity)
    .padding()
    .background(Color.blue.opacity(0.2))
    .cornerRadius(12)

    Button(action: {
        showingAddTask = true
    }) {
        Image(systemName: "plus")
```

```
                .font(.system(size: 24))
                .frame(width: 50, height: 50)
                .background(Color.green)
                .foregroundColor(.white)
                .clipShape(Circle())
                .shadow(radius: 5)
        }
        .padding(.horizontal)

        Button("ПОБУТ") {
            showingHousehold = true
        }
        .font(.headline)
        .frame(maxWidth: .infinity)
        .padding()
        .background(Color.orange.opacity(0.2))
        .cornerRadius(12)
    }
    .padding()
}
.sheet(isPresented: $showingAddTask) {
    AddTaskView(tasks: $tasks)
}
.sheet(isPresented: $showingStats) {
    AnalyticsView(tasks: tasks)
}
.sheet(isPresented: $showingFinance) {
    FinanceView()
}
```

```
            .sheet(isPresented: $showingHousehold) {
                HouseholdListView(pets: $pets, plants: $plants)
            }
            .onAppear {
                loadTasks()
                loadHousehold()
                UNUserNotificationCenter.current().requestAuthorization(options:
[.alert, .sound, .badge]) { granted, _ in
                    if granted {
                        print("□ Дозвіл на сповіщення надано")
                    }
                }
            }
            .onChange(of: tasks) { _ in
                saveTasks()
            }
            .onChange(of: pets) { _ in
                saveHousehold()
            }
            .onChange(of: plants) { _ in
                saveHousehold()
            }
        }
    }

    func saveTasks() {
        if let encoded = try? JSONEncoder().encode(tasks) {
            UserDefaults.standard.set(encoded, forKey: "tasks")
        }
```

```
    }

    func loadTasks() {
        if let data = UserDefaults.standard.data(forKey: "tasks"),
           let decoded = try? JSONDecoder().decode([Task].self, from: data) {
            tasks = decoded
        }
    }

    func saveHousehold() {
        let data = HouseholdStorage(pets: pets, plants: plants)
        if let encoded = try? JSONEncoder().encode(data) {
            UserDefaults.standard.set(encoded, forKey: "household")
        }
    }

    func loadHousehold() {
        if let data = UserDefaults.standard.data(forKey: "household"),
           let decoded = try? JSONDecoder().decode(HouseholdStorage.self, from:
data) {
            pets = decoded.pets
            plants = decoded.plants
        }
    }

    struct HouseholdStorage: Codable {
        var pets: [Pet]
        var plants: [Plant]
    }
```

```
}
```

#Файл ContentView.swift

```
import SwiftUI

@main
struct DomManagerApp: App {
    var body: some Scene {
        WindowGroup {
            MainView()
        }
    }
}
```

#Файл AddTaskView.swift

```
import SwiftUI
import UserNotifications

struct AddTaskView: View {
    @Environment(\.presentationMode) var presentationMode
    @Binding var tasks: [Task]

    @State private var title = ""
    @State private var tags = ""
    @State private var deadline = Date()
```

```
@State private var hasNotification = false

var body: some View {
    NavigationView {
        Form {
            Section(header: Text("Назва завдання")) {
                TextField("Введіть назву", text: $title)
            }

            Section(header: Text("Теги")) {
                TextField("Напр. 'прання, фінанси'", text: $tags)
            }

            Section(header: Text("Дедлайн")) {
                DatePicker("Оберіть дату і час", selection: $deadline, displayedComponents: [.date, .hourAndMinute])
            }

            Section {
                Toggle(isOn: $hasNotification) {
                    Text("Надіслати сповіщення")
                }
            }
        }
        .navigationTitle("Нове завдання")
        .navigationBarItems(trailing: Button("Зберегти") {
            if !title.isEmpty {
                let newTask = Task(title: title, tags: tags, deadline: deadline, hasNotification: hasNotification, isCompleted: false)
```

```
                tasks.append(newTask)
                if hasNotification {
                    scheduleNotification(for: newTask)
                }
                presentationMode.wrappedValue.dismiss()
            }
        })
    }
}

func scheduleNotification(for task: Task) {
    let content = UNMutableNotificationContent()
    content.title = "Нагадування"
    content.body = task.title
    content.sound = .default

    let triggerDate = Calendar.current.dateComponents([.year, .month, .day, .hour, .minute], from: task.deadline)
    let trigger = UNCalendarNotificationTrigger(dateMatching: triggerDate, repeats: false)

    let request = UNNotificationRequest(identifier: task.id.uuidString, content: content, trigger: trigger)
    UNUserNotificationCenter.current().add(request)
}
}
```

```
#Файл EditTaskView.swift
import SwiftUI
import UserNotifications

struct EditTaskView: View {
    @Environment(\.presentationMode) var presentationMode
    @Binding var task: Task

    var body: some View {
        NavigationView {
            Form {
                Section(header: Text("Назва завдання")) {
                    TextField("Назва", text: $task.title)
                }

                Section(header: Text("Теги")) {
                    TextField("Теги", text: $task.tags)
                }

                Section(header: Text("Дедлайн")) {
                    DatePicker("Оберіть дату і час", selection: $task.deadline, displayedComponents: [.date, .hourAndMinute])
                }

                Section {
                    Toggle("Надіслати сповіщення", isOn: $task.hasNotification)
                }
            }
            .navigationTitle("Редагування")
```

```
            .navigationBarItems(trailing: Button("Зберегти") {
                if task.hasNotification {
                    scheduleNotification(for: task)
                }
                presentationMode.wrappedValue.dismiss()
            })
        }
    }

    func scheduleNotification(for task: Task) {
        let content = UNMutableNotificationContent()
        content.title = "Нагадування"
        content.body = task.title
        content.sound = .default

        let triggerDate = Calendar.current.dateComponents([.year, .month, .day, .hour, .minute], from: task.deadline)
        let trigger = UNCalendarNotificationTrigger(dateMatching: triggerDate, repeats: false)

        let request = UNNotificationRequest(identifier: task.id.uuidString, content: content, trigger: trigger)
        UNUserNotificationCenter.current().add(request)
    }
}
```

#Файл TaskRowView.swift

```
import SwiftUI

struct TaskRowView: View {
    @Binding var task: Task

    var body: some View {
        VStack(alignment: .leading) {
            HStack {
                Image(systemName: task.isCompleted ? "checkmark.circle.fill" : "circle")
                    .onTapGesture {
                        task.isCompleted.toggle()
                    }
                Text(task.title)
                    .strikethrough(task.isCompleted)
                Spacer()
            }
            .font(.headline)

            HStack {
                if !task.tags.isEmpty {
                    Text("Теги: \(task.tags)")
                        .font(.caption)
                        .foregroundColor(.gray)
                }
                Spacer()
                Text(task.deadline, style: .date)
                    .font(.caption2)
                    .foregroundColor(.secondary)
```

```
            }
        }
        .padding(.horizontal)
    }
}
```

#Файл TaskModel.swift

```
import Foundation

struct Task: Identifiable, Codable, Equatable {
    let id: UUID
    var title: String
    var tags: String
    var deadline: Date
    var hasNotification: Bool
    var isCompleted: Bool

    init(id: UUID = UUID(), title: String, tags: String, deadline: Date, hasNotification:
Bool, isCompleted: Bool) {
        self.id = id
        self.title = title
        self.tags = tags
        self.deadline = deadline
        self.hasNotification = hasNotification
        self.isCompleted = isCompleted
    }
}
```

```
#Файл AnalyticsView.swift
import SwiftUI
import Charts

struct AnalyticsView: View {
    var tasks: [Task]

    var completedCount: Int {
        tasks.filter { $0.isCompleted }.count
    }

    var totalCount: Int {
        tasks.count
    }

    var completionPercentage: Double {
        guard totalCount > 0 else { return 0 }
        return (Double(completedCount) / Double(totalCount)) * 100
    }

    var percentageColor: Color {
        switch completionPercentage {
        case 0..<30: return .red
        case 30..<60: return .yellow
        default: return .green
        }
    }
```

```
var motivationalMessage: String {
    let emojis = ["□", "□", "□", "□", "□", "□", "□"]
    return "Супер, Ти молодець! " + (emojis.randomElement() ?? "□")
}

var body: some View {
    VStack(spacing: 24) {
        Text("Статистика")
            .font(.title)
            .bold()

        Chart {
            SectorMark(
                angle: .value("Виконано", completedCount),
                innerRadius: .ratio(0.5),
                angularInset: 1
            )
            .foregroundStyle(.green)

            SectorMark(
                angle: .value("Залишилось", totalCount - completedCount),
                innerRadius: .ratio(0.5),
                angularInset: 1
            )
            .foregroundStyle(.red)
        }
        .frame(height: 260)
        .padding()
```

```
            Text("Виконано \(completedCount) із \(totalCount) справ")
                .font(.headline)

            Text("\(Int(completionPercentage))%")
                .font(.title)
                .padding()
                .frame(width: 100, height: 50)
                .background(percentageColor.opacity(0.2))
                .foregroundColor(.primary)
                .clipShape(Capsule())

            if completionPercentage == 100 {
                Text(motivationalMessage)
                    .font(.headline)
                    .foregroundColor(.green)
                    .transition(.opacity)
            }
        }
        .padding()
    }
}
```

#Файл FinanceView.swift

```
import SwiftUI
import Charts
```

```
struct FinanceView: View {
    @State private var transactions: [Transaction] = []
    @State private var showAddTransaction = false
    @State private var showBreakdown = false
    @State private var showChart = false
    @State private var selectedTransaction: Transaction?

    var body: some View {
        NavigationView {
            VStack(alignment: .leading) {
                let income = monthlyAmount(income: true)
                let expense = monthlyAmount(income: false)

                VStack(alignment: .leading, spacing: 10) {
                    Text("Баланс за місяць:")
                        .font(.headline)
                    Text("Отримано: \(income, specifier: "%.2f") грн")
                    Text("Витрачено: \(expense, specifier: "%.2f") грн")
                    Text("Залишок: \((income - expense), specifier: "%.2f") грн")
                        .fontWeight(.bold)

                    HStack {
                        Button("Готівка / Картка") {
                            showBreakdown.toggle()
                        }

                        Spacer()

                        Button {
```

```
            showChart.toggle()
        } label: {
            Label("Графік", systemImage: "chart.bar.xaxis")
        }
    }
    .padding(.vertical, 5)
}
.padding()

List {
    ForEach(transactions.indices, id: \.self) { i in
        Button {
            selectedTransaction = transactions[i]
        } label: {
            HStack {
                VStack(alignment: .leading) {
                    Text(transactions[i].title)
                        .font(.headline)
                    Text(transactions[i].tag)
                        .font(.caption)
                        .foregroundColor(.gray)
                }
                Spacer()
                Text("\(transactions[i].amount, specifier: "%.2f") грн")
                    .foregroundColor(transactions[i].isIncome ? .green : .red)
            }
        }
    }
    .onDelete { indexSet in
```

```
                transactions.remove(atOffsets: indexSet)
                saveTransactions()
            }
        }
    }
    .navigationTitle("Фінанси")
    .toolbar {
        ToolbarItem(placement: .navigationBarTrailing) {
            Button(action: { showAddTransaction = true }) {
                Image(systemName: "plus")
            }
        }
    }
    .sheet(isPresented: $showAddTransaction) {
        AddOrEditTransactionView(transactions: $transactions)
    }
    .sheet(item: $selectedTransaction) { transaction in
        AddOrEditTransactionView(transactions: $transactions, editingTransaction:
transaction)
    }
    .sheet(isPresented: $showBreakdown) {
        PaymentBreakdownView(transactions: transactions)
    }
    .sheet(isPresented: $showChart) {
        TransactionChartView(transactions: transactions)
    }
    .onAppear(perform: loadTransactions)
  }
}
```

```
    func monthlyAmount(income: Bool) -> Double {
        let calendar = Calendar.current
        let currentMonth = calendar.component(.month, from: Date())
        return transactions.filter {
            calendar.component(.month, from: $0.date) == currentMonth && $0.isIncome
== income
        }.reduce(0) { $0 + $1.amount }
    }

    func saveTransactions() {
        if let encoded = try? JSONEncoder().encode(transactions) {
            UserDefaults.standard.set(encoded, forKey: "transactions")
        }
    }

    func loadTransactions() {
        if let data = UserDefaults.standard.data(forKey: "transactions"),
           let decoded = try? JSONDecoder().decode([Transaction].self, from: data) {
            transactions = decoded
        }
    }
}
```

#Файл TransactionModel.swift

```
import Foundation
```

```
struct Transaction: Identifiable, Codable, Equatable {
    let id: UUID
    var title: String
    var amount: Double
    var date: Date
    var tag: String
    var isIncome: Bool
    var isCash: Bool

    init(id: UUID = UUID(), title: String, amount: Double, date: Date, tag: String, isIncome: Bool, isCash: Bool) {
        self.id = id
        self.title = title
        self.amount = amount
        self.date = date
        self.tag = tag
        self.isIncome = isIncome
        self.isCash = isCash
    }
}
```

#Файл TransactionChartView.swift

```
import SwiftUI
import Charts

struct TransactionChartView: View {
    var transactions: [Transaction]
```

```
    var grouped: [(String, Double)] {
        let groupedDict = Dictionary(grouping: transactions) { transaction in
            let formatter = DateFormatter()
            formatter.dateFormat = "MMM"
            return formatter.string(from: transaction.date)
        }

        return groupedDict.map { (month, transactions) in
            let total = transactions.reduce(0) { $0 + ($1.isIncome ? $1.amount : -$1.amount) }
            return (month, total)
        }.sorted { $0.0 < $1.0 }
    }

    var body: some View {
        VStack {
            Text("Графік витрат/доходів")
                .font(.headline)

            Chart {
                ForEach(grouped, id: \.0) { (month, total) in
                    BarMark(
                        x: .value("Місяць", month),
                        y: .value("Сума", total)
                    )
                    .foregroundStyle(total >= 0 ? .green : .red)
                }
            }
```

```
            .frame(height: 300)
            .padding()
        }
    }
}
```

#Файл AddOrEditTransactionView.swift

```
import SwiftUI

struct AddOrEditTransactionView: View {
    @Environment(\.presentationMode) var presentationMode
    @Binding var transactions: [Transaction]

    var editingTransaction: Transaction?

    @State private var title = ""
    @State private var amount: String = ""
    @State private var date = Date()
    @State private var tag = ""
    @State private var isIncome = false
    @State private var isCash = true

    var body: some View {
        NavigationView {
            Form {
                Section(header: Text("Назва")) {
                    TextField("Наприклад: Зарплата", text: $title)
```

```
            }

            Section(header: Text("Сума")) {
                TextField("0.00", text: $amount)
                    .keyboardType(.decimalPad)
            }

            Section(header: Text("Дата")) {
                DatePicker("Дата", selection: $date, displayedComponents: [.date])
            }

            Section(header: Text("Тег")) {
                TextField("Напр. харчі, транспорт", text: $tag)
            }

            Section(header: Text("Тип транзакції")) {
                Picker("Тип", selection: $isIncome) {
                    Text("Витрата").tag(false)
                    Text("Дохід").tag(true)
                }
                .pickerStyle(SegmentedPickerStyle())
            }

            Section {
                Toggle("Оплата готівкою", isOn: $isCash)
            }
        }
        .navigationTitle(editingTransaction == nil ? "Нова транзакція" :
"Редагування")
```

```
        .navigationBarItems(trailing: Button("Зберегти") {
            guard let value = Double(amount) else { return }

            if let index = transactions.firstIndex(where: { $0.id ==
editingTransaction?.id }) {
                transactions[index].title = title
                transactions[index].amount = value
                transactions[index].date = date
                transactions[index].tag = tag
                transactions[index].isIncome = isIncome
                transactions[index].isCash = isCash
            } else {
                let new = Transaction(title: title, amount: value, date: date, tag: tag,
isIncome: isIncome, isCash: isCash)
                transactions.append(new)
            }

            save()
            presentationMode.wrappedValue.dismiss()
        })
        .onAppear {
            if let editing = editingTransaction {
                title = editing.title
                amount = String(editing.amount)
                date = editing.date
                tag = editing.tag
                isIncome = editing.isIncome
                isCash = editing.isCash
            }
```

```
            }
        }
    }

    func save() {
        if let encoded = try? JSONEncoder().encode(transactions) {
            UserDefaults.standard.set(encoded, forKey: "transactions")
        }
    }
}


#Файл PaymentBreakdownView.swift
import SwiftUI

struct PaymentBreakdownView: View {
    var transactions: [Transaction]

    var body: some View {
        VStack(spacing: 15) {
            Text("Розподіл за способом оплати")
                .font(.title2)

            let cash = transactions.filter { !$0.isIncome && $0.isCash }.reduce(0) { $0 +
$1.amount }
            let card = transactions.filter { !$0.isIncome && !$0.isCash }.reduce(0) { $0 +
$1.amount }

            Text("Готівка: \(cash, specifier: "%.2f") грн")
```

```
            Text("Картка: \(card, specifier: "%.2f") грн")

            Spacer()
        }
        .padding()
    }
}
```

#Файл HouseholdListView.swift

```
import SwiftUI

struct HouseholdListView: View {
    @Binding var pets: [Pet]
    @Binding var plants: [Plant]

    @State private var showAddPet = false
    @State private var showAddPlant = false
    @State private var selectedPet: Pet?
    @State private var selectedPlant: Plant?

    var body: some View {
        NavigationView {
            List {
                if !pets.isEmpty {
                    Section("Тварини") {
                        ForEach(pets) { pet in
                            Button(action: { selectedPet = pet }) {
```

```
                VStack(alignment: .leading) {
                    Text(pet.name).font(.headline)
                    Text("Народилася: \(pet.birthDate, style: .date)")
                        .font(.caption)
                }
            }
        }
        .onDelete { pets.remove(atOffsets: $0) }
    }
  }
  if !plants.isEmpty {
    Section("Рослини") {
        ForEach(plants) { plant in
            Button(action: { selectedPlant = plant }) {
                VStack(alignment: .leading) {
                    Text(plant.name).font(.headline)
                    Text("Останній полив: \(plant.lastWatered, style: .date)")
                        .font(.caption)
                }
            }
        }
        .onDelete { plants.remove(atOffsets: $0) }
    }
  }
}
.navigationTitle("Побут")
.toolbar {
  ToolbarItemGroup(placement: .navigationBarTrailing) {
    Button("Додати □") { showAddPet = true }
```

```
                Button("Додати □") { showAddPlant = true }
            }
        }
        .sheet(isPresented: $showAddPet) {
            AddEditPetView(pets: $pets)
        }
        .sheet(isPresented: $showAddPlant) {
            AddEditPlantView(plants: $plants)
        }
        .sheet(item: $selectedPet) { pet in
            AddEditPetView(pets: $pets, editingPet: pet)
        }
        .sheet(item: $selectedPlant) { plant in
            AddEditPlantView(plants: $plants, editingPlant: plant)
        }
    }
  }
}
```

#Файл HouseHoldModel.swift

```
import Foundation

struct Pet: Identifiable, Codable, Equatable {
  let id: UUID
  var name: String
  var birthDate: Date
  var vaccinations: [String]
```

```
}

struct Plant: Identifiable, Codable, Equatable {
    let id: UUID
    var name: String
    var lastWatered: Date
    var lastFertilized: Date
}

enum HouseholdItem: Identifiable, Codable, Equatable {
    case pet(Pet)
    case plant(Plant)

    var id: UUID {
        switch self {
        case .pet(let p): return p.id
        case .plant(let p): return p.id
        }
    }

    var name: String {
        switch self {
        case .pet(let p): return p.name
        case .plant(let p): return p.name
        }
    }
}
```

```
#Файл AddEditPlantView.swift
import SwiftUI

struct AddEditPlantView: View {
    @Environment(\.presentationMode) var presentationMode
    @Binding var plants: [Plant]
    var editingPlant: Plant?

    @State private var name: String = ""
    @State private var lastWatered: Date = Date()
    @State private var lastFertilized: Date = Date()

    var body: some View {
        NavigationView {
            Form {
                Section("Назва рослини") {
                    TextField("Наприклад: Фікус", text: $name)
                }
                Section("Останній полив") {
                    DatePicker("", selection: $lastWatered, displayedComponents: .date)
                }
                Section("Останнє підживлення") {
                    DatePicker("", selection: $lastFertilized, displayedComponents: .date)
                }
            }
            .navigationTitle(editingPlant == nil ? "Додати рослину" : "Редагувати рослину")
            .navigationBarItems(trailing: Button("Зберегти") {
```

```
            if var pl = editingPlant {
                pl.name = name
                pl.lastWatered = lastWatered
                pl.lastFertilized = lastFertilized
                if let idx = plants.firstIndex(where: { $0.id == pl.id }) {
                    plants[idx] = pl
                }
            } else {
                let new = Plant(id: UUID(), name: name, lastWatered: lastWatered, lastFertilized: lastFertilized)
                plants.append(new)
            }
            presentationMode.wrappedValue.dismiss()
        })
        .onAppear {
            if let plant = editingPlant {
                name = plant.name
                lastWatered = plant.lastWatered
                lastFertilized = plant.lastFertilized
            }
        }
    }
}
```

#Файл AddEditPetView.swift

```
import SwiftUI
```

```
struct AddEditPetView: View {
    @Environment(\.presentationMode) var presentationMode
    @Binding var pets: [Pet]
    var editingPet: Pet?
    @State private var name: String = ""
    @State private var birthDate: Date = Date()
    @State private var vaccinationsText: String = "" // comma-separated
    var body: some View {
        NavigationView {
            Form {
                Section("Ім’я тварини") {
                    TextField("Ім’я", text: $name)
                }
                Section("Дата народження") {
                    DatePicker("", selection: $birthDate, displayedComponents: .date)
                }
                Section("Щеплення (через кому)") {
                    TextField("Ввести щеплення", text: $vaccinationsText)
                }
            }
            .navigationTitle(editingPet == nil ? "Додати тварину" : "Редагувати тварину")
            .navigationBarItems(trailing: Button("Зберегти") {
                let vaccs = vaccinationsText.split(separator: ",").map { $0.trimmingCharacters(in: .whitespaces) }
                if var p = editingPet {
                    p.name = name
                    p.birthDate = birthDate
```

```
                p.vaccinations = vaccs
                if let idx = pets.firstIndex(where: { $0.id == p.id }) {
                    pets[idx] = p
                }
            } else {
                let new = Pet(id: UUID(), name: name, birthDate: birthDate, vaccinations:
vaccs)
                pets.append(new)
            }
            presentationMode.wrappedValue.dismiss()
        })
        .onAppear {
            if let pet = editingPet {
                name = pet.name
                birthDate = pet.birthDate
                vaccinationsText = pet.vaccinations.joined(separator: ", ")
            }
        }
    }
}
```

ДОДАТОК В (ОБОВ'ЯЗКОВИЙ)
ІЛЮСТРАТИВНА ЧАСТИНА

«РОЗРОБКА IOS-ЗАСТОСУНКУ ДЛЯ ОРГАНІЗАЦІЇ ТА МЕНЕДЖМЕНТУ ПОБУТОВИХ ПОТРЕБ»

Виконав: студент 4 курсу, групи ЗКН-21б
спеціальності 122 Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Андрій ЗАХАРЕВИЧ
(прізвище та ініціали)
Керівник: асистент кафедри КН
Єгор СІЛАГІН
(прізвище та ініціали)

3 червня 2025 року

Вінниця ВНТУ – 2025 рік

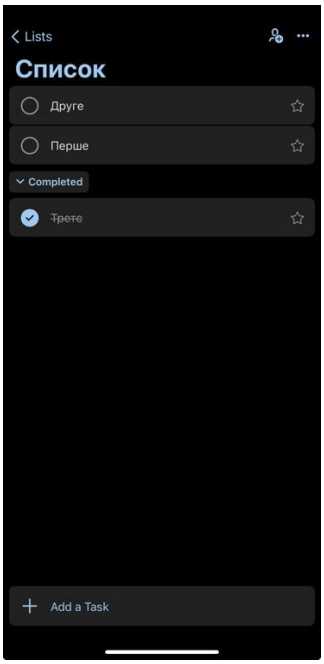

Рисунок В.1 - Вигляд інтерфейсу застосунку «Microsoft To Do»

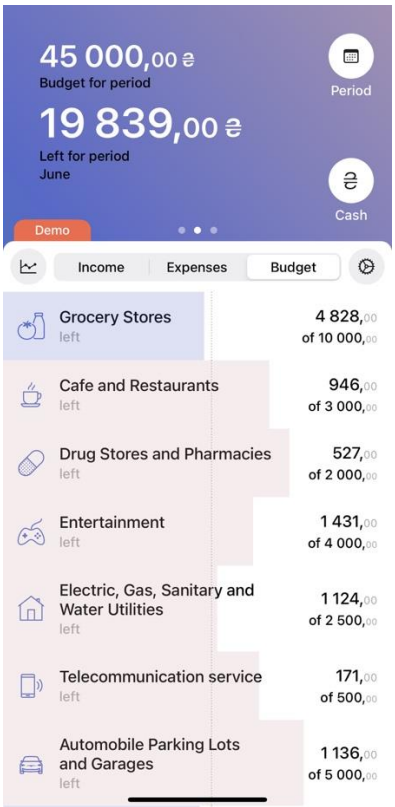

Рисунок В.2 - Вигляд інтерфейсу застосунку «monobudget»

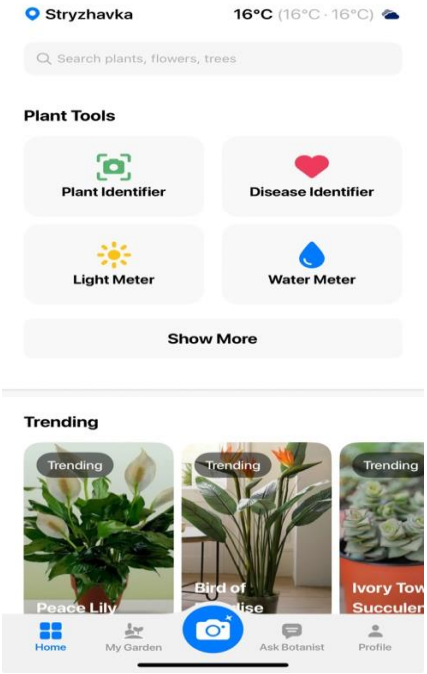

Рисунок В.3 – Вигляд інтерфейсу «Plant App: Plant Identifier»

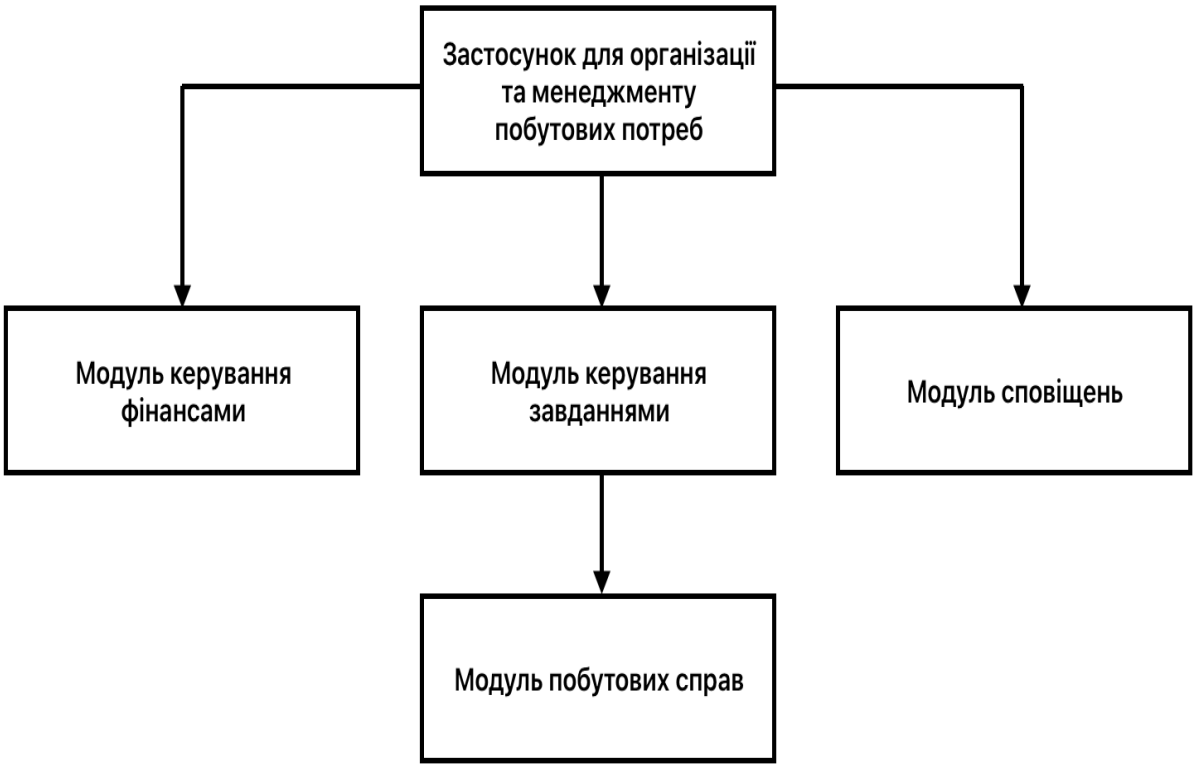

Рисунок В.4 – Архітектура застосунку

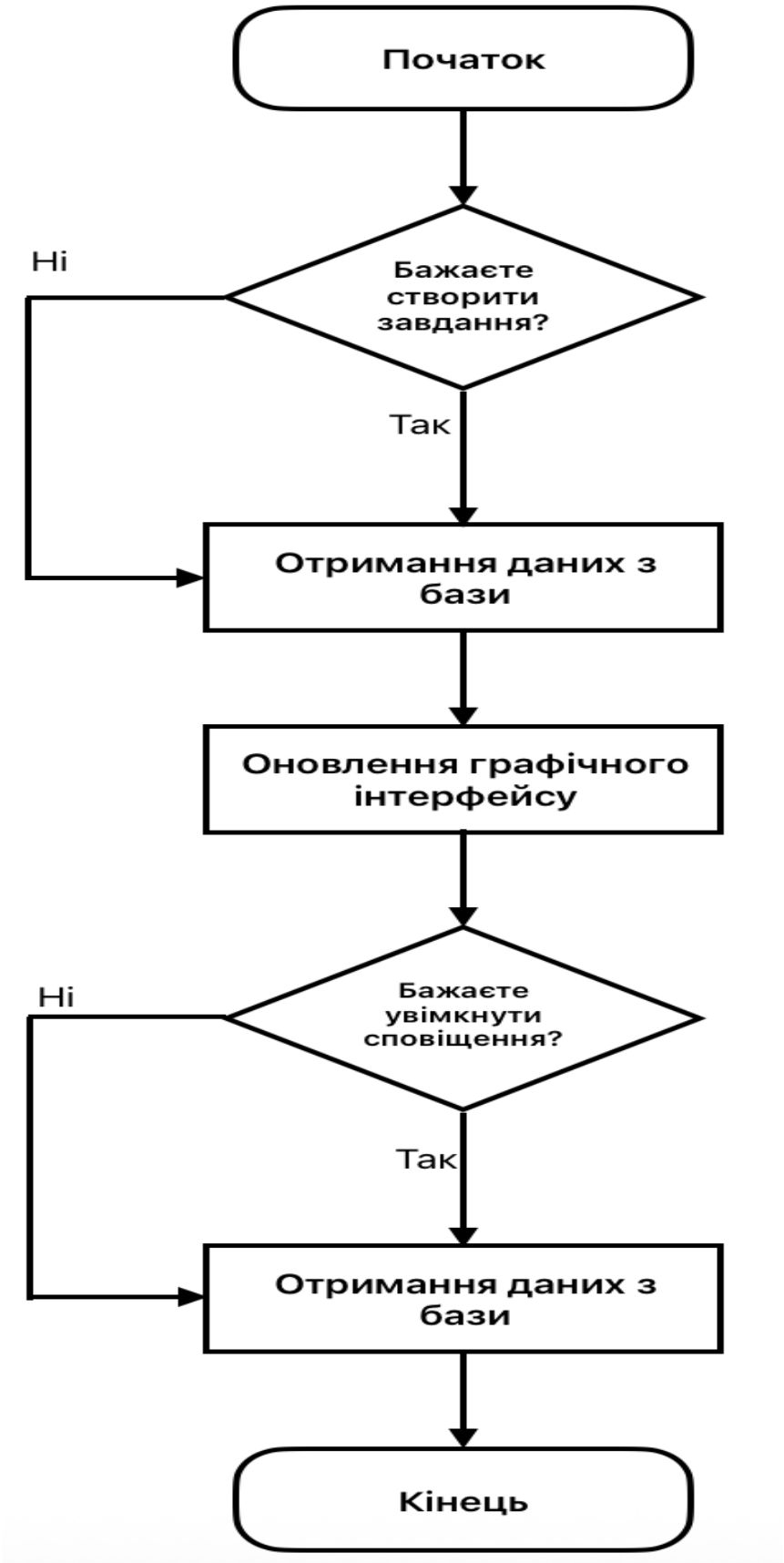

Рисунок В.5 – Алгоритм роботи застосунку

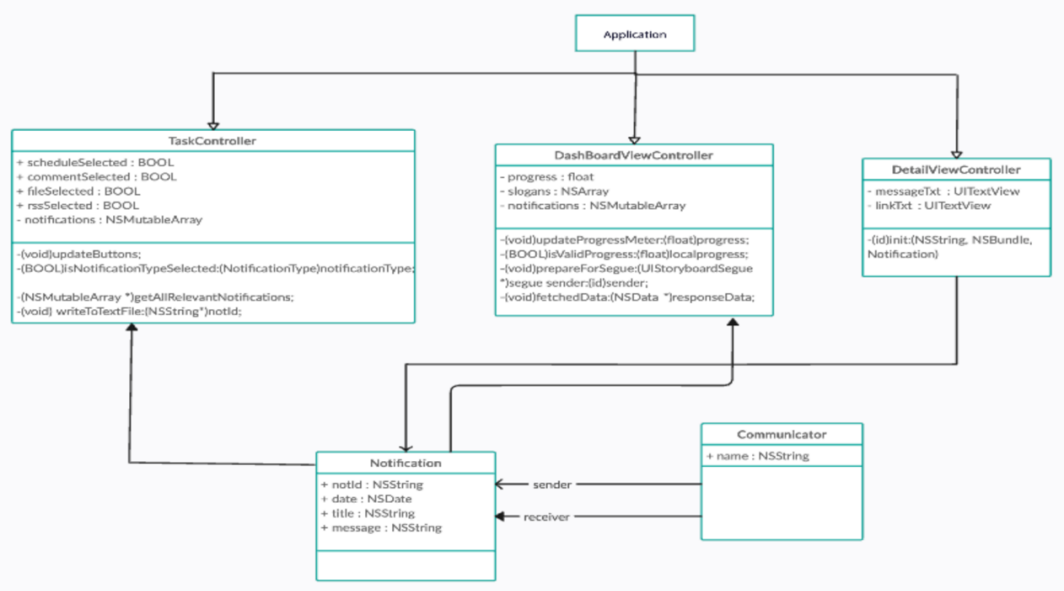

Рисунок В.6 – UML-діаграма класів застосунку

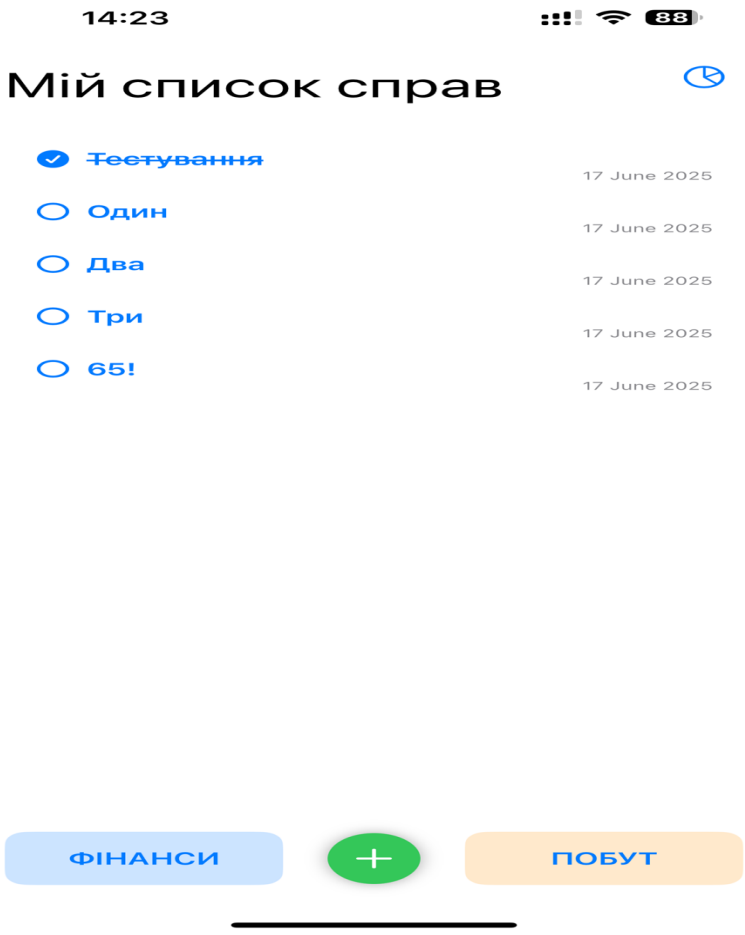

Рисунок В.7 – Головне меню застосунку

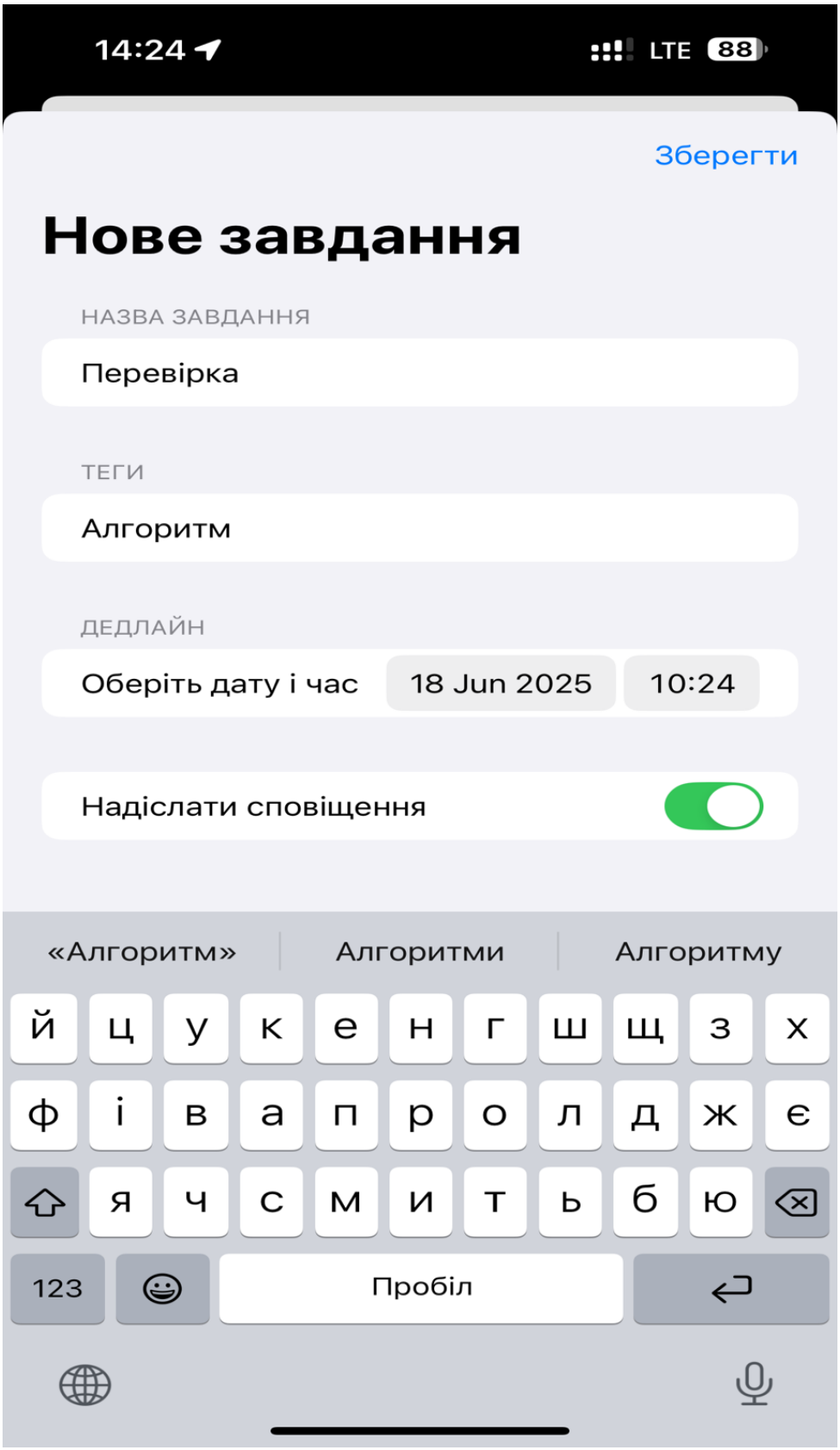

Рисунок В.8 – Меню створення нового завдання

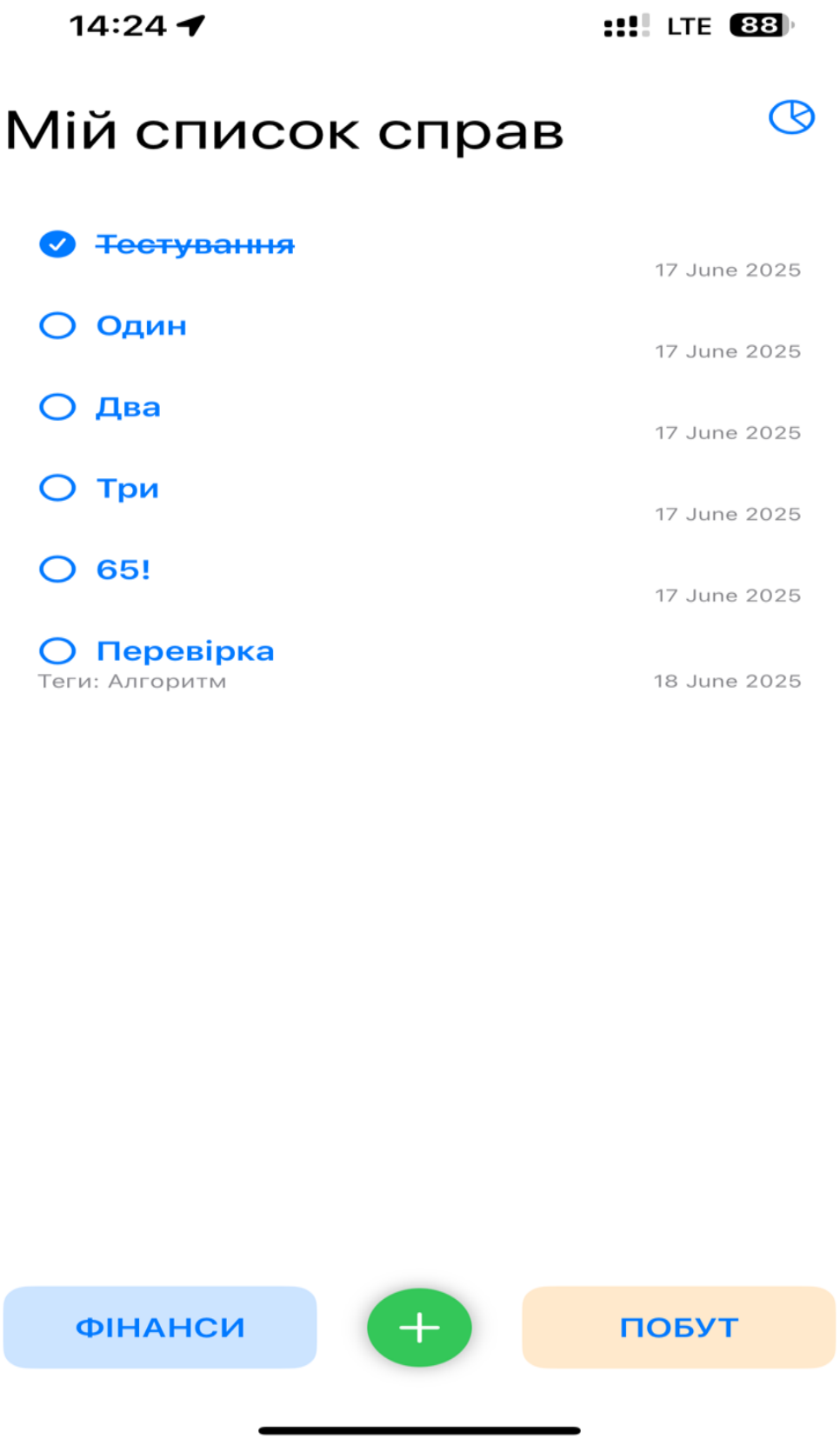

Рисунок В.9. – Результат створення нового завдання

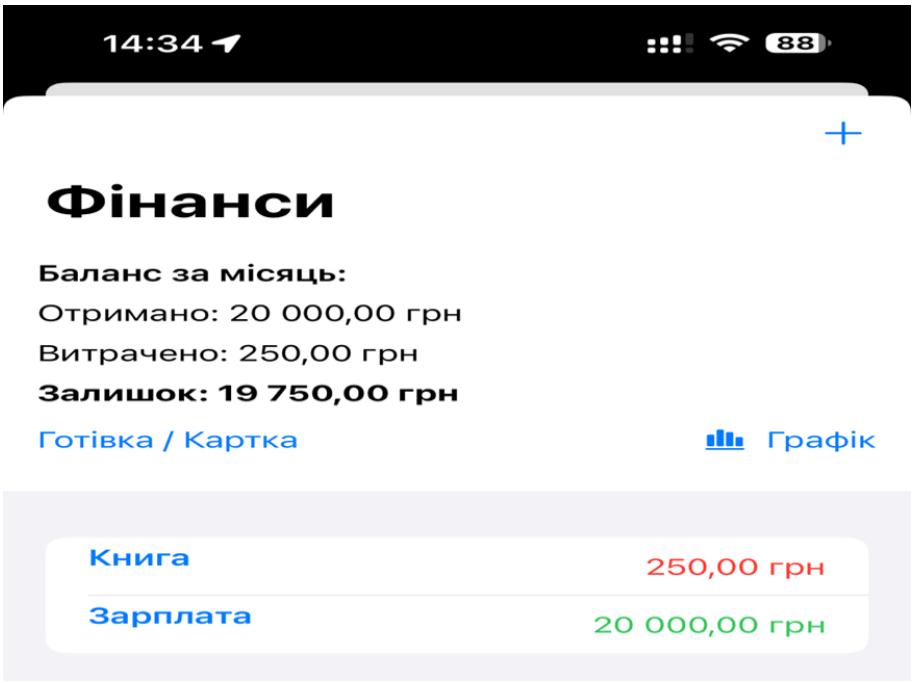

Рисунок В.10 – Вікно з ведення фінансового обліку

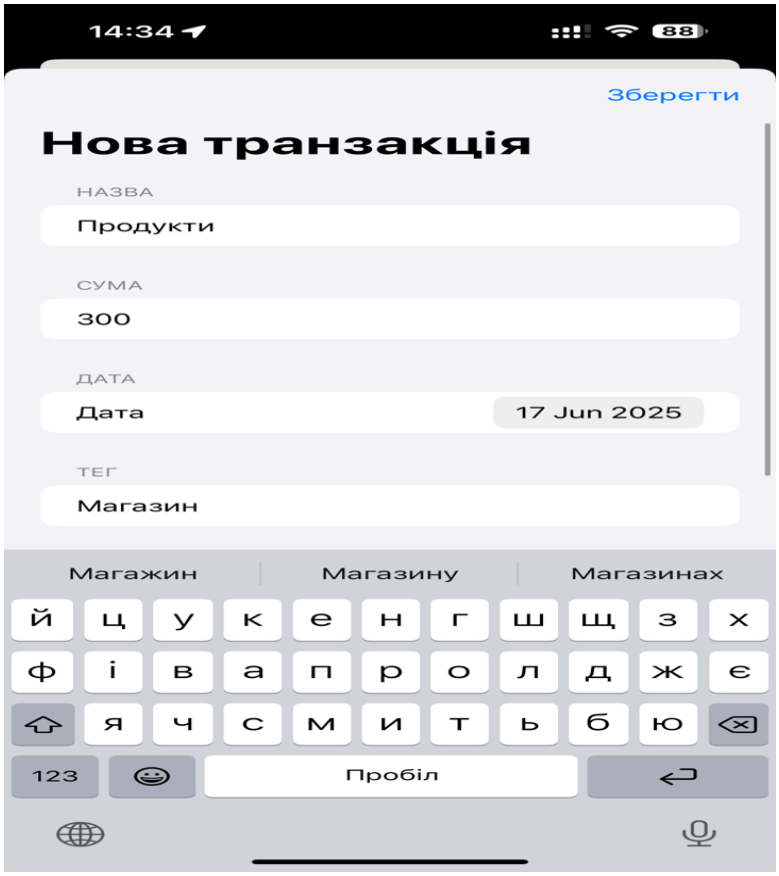

Рисунок В.11 – Вікно створення нової транзакції

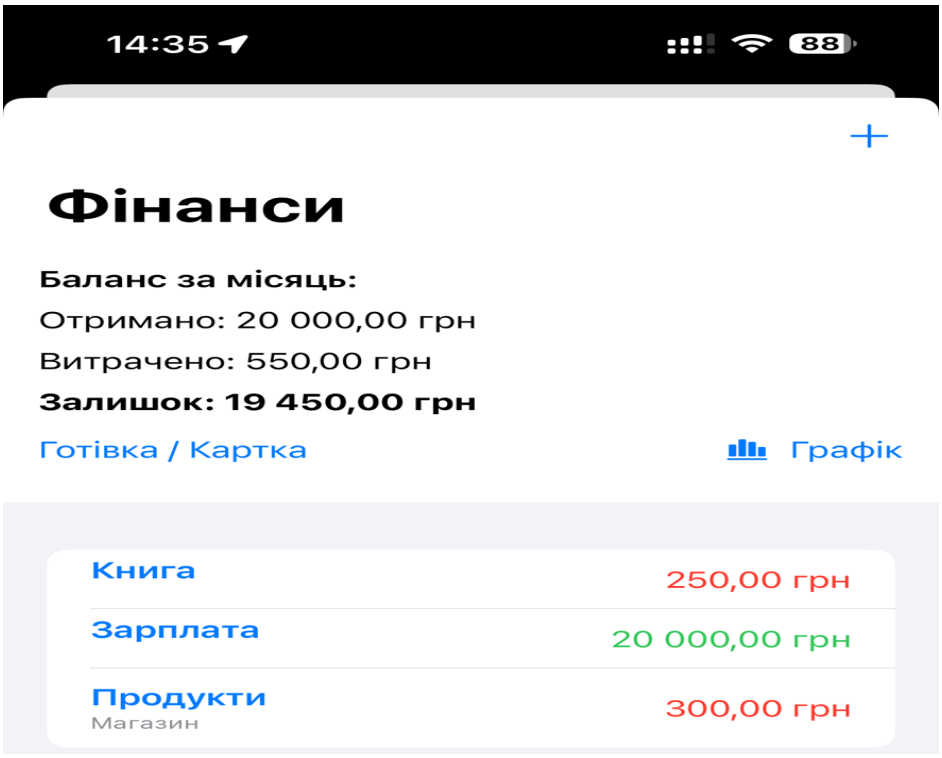

Рисунок В.12 – Результат виконання функції обліку фінансів

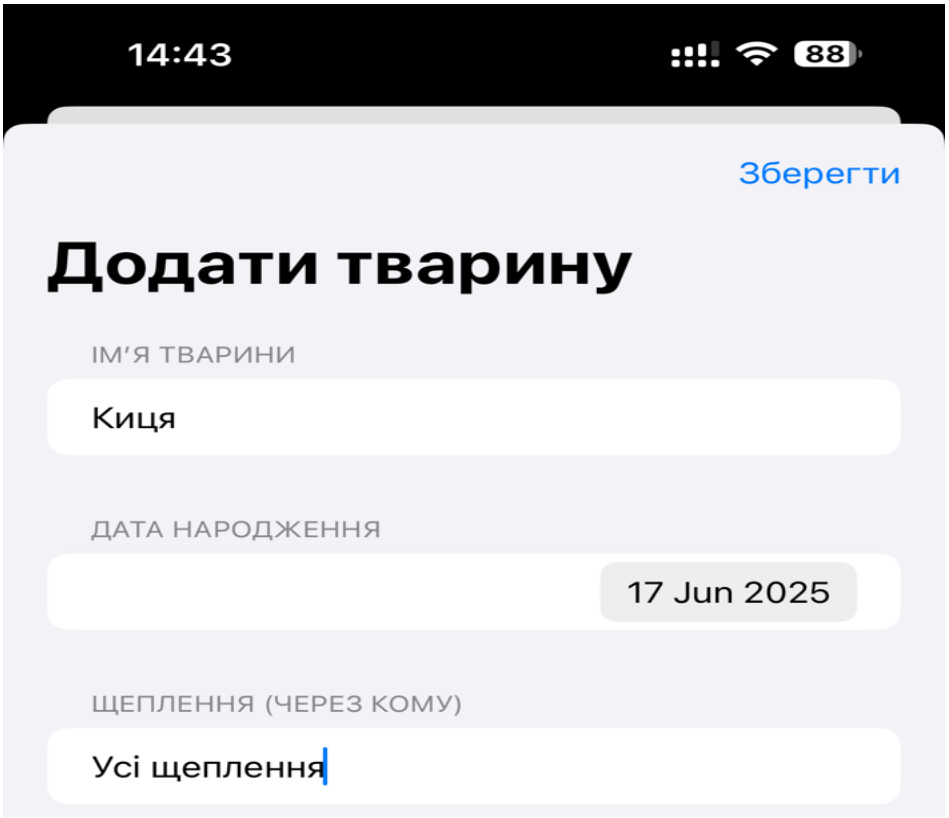

Рисунок В.13 – Створення картки тварини

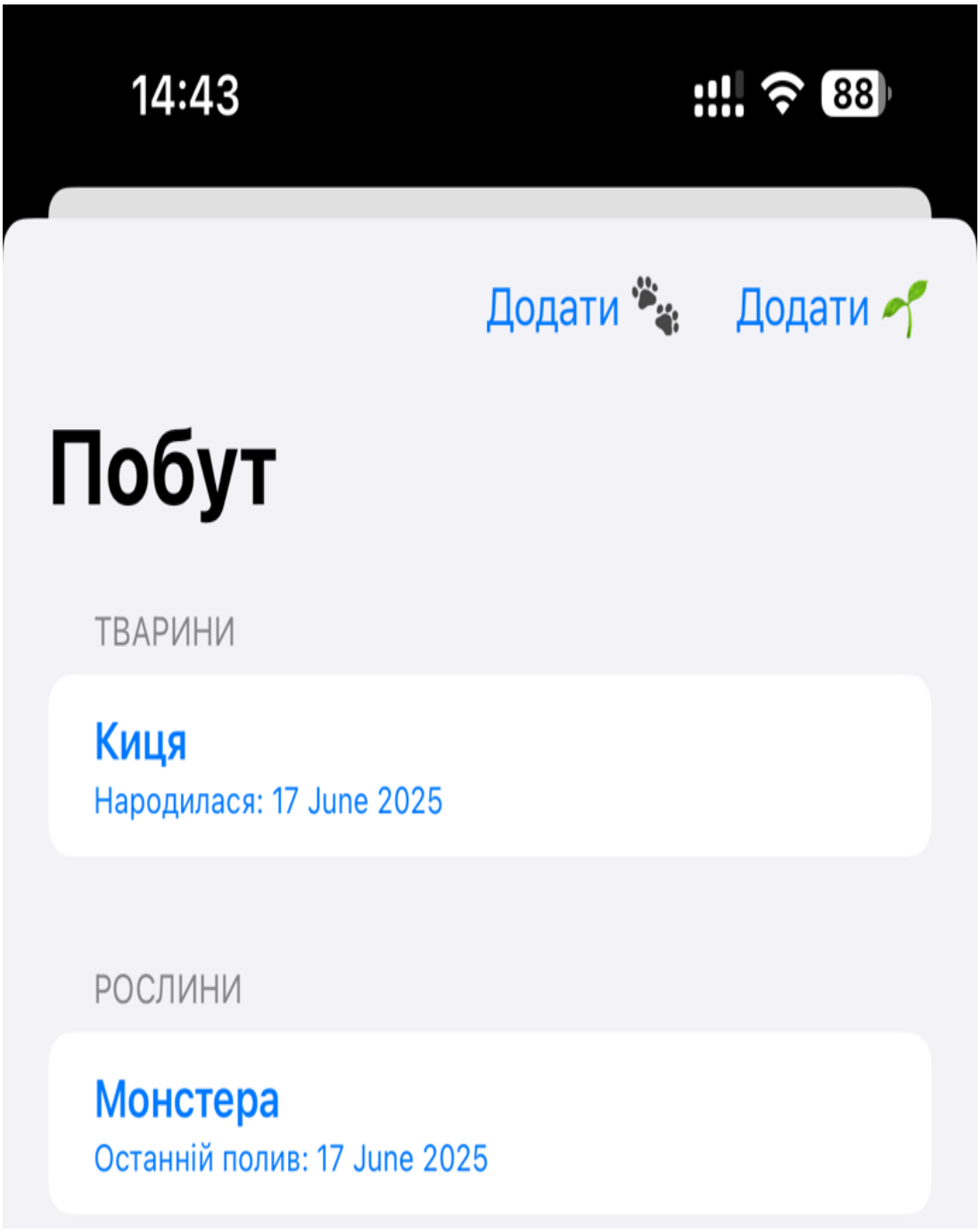

Рисунок В.14 – Результат виконання функції

ДОДАТОК Г (ДОВІДНИКОВИЙ)
ІНСТРУКЦІЯ КОРИСТУВАЧА

1. Завантажити застосунок на смартфон.
2. Відкрити застосунок.
3. На головному екрані натиснути кнопку «+», ввести назву, термін, тег, завдання, та натиснути кнопку зберегти

Посередині екрану буде відображено список з усіма поточними завданнями, праворуч вверху знаходиться кнопка, при натисканні на яку, користувач бачe прогрес виконання усіх завдань.

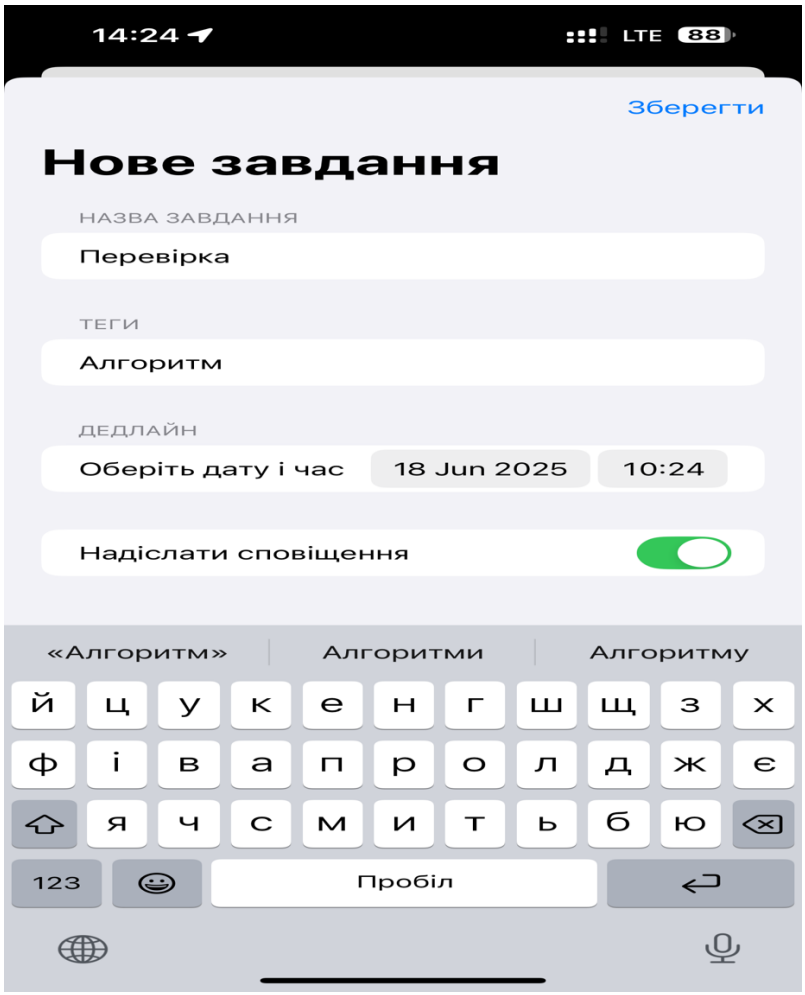

Рисунок Г.1 – Меню створення нового завдання

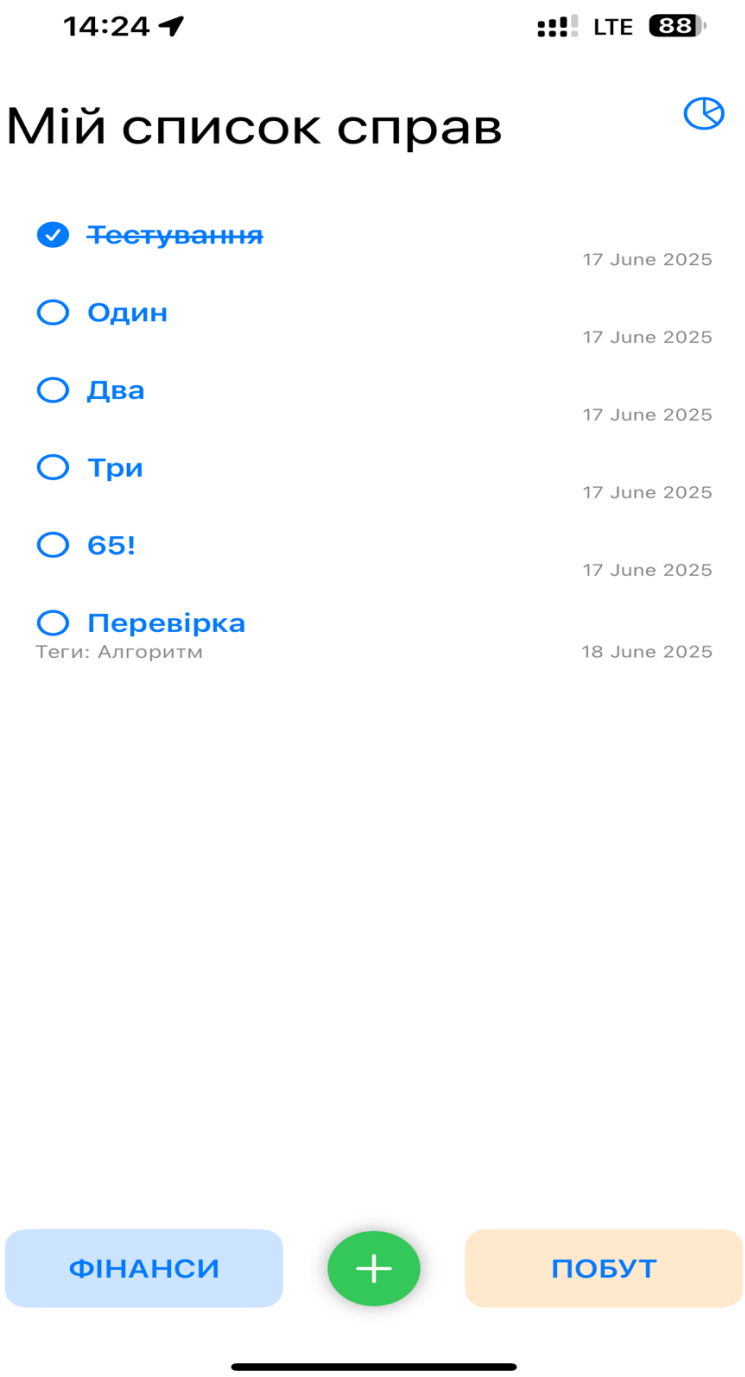

Рисунок Г.2. – Результат створення нового завдання