

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра комп'ютерних систем та мереж

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавра

(ступінь вищої освіти (освітній ступінь))

на тему **РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ
САМОМЕНЕДЖМЕНТУ**

Виконав: студент 4 курсу, групи КНТз-513сп
спеціальності _____

123 Комп'ютерна інженерія

(код і найменування спеціальності)

Освітня програма (спеціалізація) _____

Комп'ютерна інженерія

ШИЛЬКО Н.А.

(ПРИЗВИЩЕ та ініціали)

Керівник ГОЛУБ Т.В.

(ПРИЗВИЩЕ та ініціали)

Рецензент РОМАНОВСЬКИЙ О.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування вищого навчального закладу)

Факультет Комп'ютерних наук і технологій
 Кафедра Комп'ютерні системи та мережі
 Ступінь вищої освіти бакалаврський
 Спеціальність 123 Комп'ютерна інженерія
 (код і найменування)
 Освітня програма (спеціалізація) Комп'ютерна інженерія
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
 Завідувач кафедри КУДЕРМЕТОВ Р.К.

“ ” 2026 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

ШИЛЬКО Наталії Анатоліївни
 (ПРИЗВИЩЕ, ім'я, по батькові)

- Тема проєкту (роботи) Розробка інформаційної системи самоменеджменту
 керівник проєкту (роботи) к.т.н., доцент, ГОЛУБ Тетяна Василівна
 (науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)
- затверджені наказом вищого навчального закладу від «14» квітня 2026 року № 160
- Строк подання студентом проєкту (роботи) 10 червня 2026 року
- Вихідні дані до проєкту (роботи) Існуючі підходи та методології самоменеджменту: GTD, OKR, SMART, Pomodoro, habit stacking, weekly review. Принципи побудови мобільних застосунків особистої продуктивності з підтримкою офлайн-режиму. Кросплатформна розробка мобільних додатків на платформі Flutter (Dart). Архітектурний патерн Clean Architecture з державним менеджером Riverpod для управління складним станом застосунку. Локальне зашифроване сховище Hive (AES-256) для офлайн-роботи. Хмарна NoSQL база даних Firebase Firestore для синхронізації даних користувача. Сервіс автентифікації Supabase Auth з JWT-токенами. Серверна частина на FastAPI для аналітичних обчислень. Безпечне зберігання токенів засобами Flutter Secure Storage (Keychain/Keystore). Передача даних із застосуванням TLS 1.3. Інструменти тестування Flutter-додатків: integration_test, Patrol, Firebase Test Lab.
- Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області та існуючих рішень у сфері самоменеджменту. 2. Проектування інформаційної системи самоменеджменту. 3. Розробка інформаційної системи самоменеджменту. 4. Тестування інформаційної системи самоменеджменту.
- Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	ГОЛУБ Т.В., к.т.н.		
нормоконтроль	ДЬЯЧУК Т.С., ст. викладач		

7. Дата видачі завдання 01.05.2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз ринку застосунків для особистої продуктивності та порівняльний аналіз існуючих мобільних додатків самоменеджменту (Notion, Todoist, TickTick)	05.03.2026 р.	
2	Розробка специфікації вимог до системи самоменеджменту (управління задачами, цілями, звичками, обліку часу, щоденник, аналітика, синхронізація)	20.03.2026 р.	
3	Проектування архітектури системи; обґрунтування вибору Clean Architecture з Riverpod та стратегії Offline-First	30.03.2026 р.	
4	Розробка схеми локальної БД Hive із шифруванням AES-256 та хмарної структури Firebase Firestore для синхронізації	05.04.2026 р.	
5	Розробка серверної частини на FastAPI для аналітичних обчислень та інтеграція з Supabase Auth для автентифікації користувачів	20.04.2026 р.	
6	Реалізація всіх функціональних модулів мобільного додатку на Flutter (задачі, цілі, звички, таймер, щоденник, аналітика) для iOS та Android	01.05.2026 р.	
7	Реалізація механізму фонові синхронізації Hive із Firebase Firestore та інтеграція push-сповіщень про задачі і звички	10.05.2026 р.	
8	Функціональне, навантажувальне та UX-тестування системи за допомогою integration test, Patrol та Firebase Test Lab	15.05.2026 р.	
9	Оформлення пояснювальної записки та графічного матеріалу	20.05.2026 р.	

Студент (ка)

(підпис)

Наталія ШИЛЬКО

(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Тетяна ГОЛУБ

(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 87 с., 31 рис., 20 табл., 20 джерел, 1 додаток.

CAMOMENEДЖМЕНТ, ОСОБИСТА ПРОДУКТИВНІСТЬ, FLUTTER, DART, CLEAN ARCHITECTURE, RIVERPOD, HIVE, FIREBASE FIRESTORE, SUPABASE, FASTAPI, OFFLINE-FIRST, GTD, OKR, POMODORO, МОБІЛЬНИЙ ДОДАТОК, ТРЕКЕР ЗВИЧОК, ОБЛІК ЧАСУ

Метою дипломного проєкту є розробка інформаційної системи самоменеджменту у вигляді кросплатформного мобільного додатку, що об'єднує управління завданнями, відстеження цілей та звичок, облік часу та аналітику продуктивності з підтримкою офлайн-режиму роботи.

Предметом дослідження є методології самоменеджменту (GTD, OKR, SMART, Pomodoro) та принципи побудови офлайн-орієнтованих мобільних застосунків. Об'єктом дослідження є інформаційна система, що поєднує мобільний клієнт на Flutter, локальне сховище Hive, хмарну синхронізацію через Firebase Firestore та аналітичні сервіси на FastAPI.

У роботі проведено порівняльний аналіз додатків для самоменеджменту (Notion, Todoist, TickTick), розроблено специфікацію вимог до семи функціональних модулів, спроектовано архітектуру за принципами Clean Architecture з Riverpod, реалізовано стратегію Offline-First з фоновією синхронізацією Hive та Firebase Firestore.

Технологічний стек: мобільний клієнт – Flutter (Dart, iOS/Android); архітектура – Clean Architecture з Riverpod; локальне сховище – Hive (AES-256); хмарна синхронізація – Firebase Firestore; автентифікація – Supabase Auth (JWT); серверна частина – FastAPI; візуалізація – fl_chart; тестування – integration_test, Patrol, Firebase Test Lab.

ЗМІСТ

Вступ.....	6
1 Аналіз предметної області та існуючих рішень у сфері самоменеджменту	9
1.1 Огляд сучасного стану ринку додатків для особистої продуктивності та самоменеджменту	9
1.2 Порівняльний аналіз існуючих систем самоменеджменту.....	14
1.3 Вимоги до розробки інформаційної системи самоменеджменту	20
1.4 Висновки за розділом 1	23
2 Проєктування інформаційної системи самоменеджменту	24
2.1 Розробка специфікації вимог до системи самоменеджменту	24
2.2 Архітектура інформаційної системи та вибір патернів проєктування	30
2.3 Проєктування бази даних та структури зберігання даних користувача	35
2.4 Висновки за розділом 2	40
3 Розробка інформаційної системи самоменеджменту.....	41
3.1 Технологічний стек мобільного додатку та серверної частини	41
3.2 Програмна реалізація модулів системи	47
3.3 Розробка інтерфейсу користувача мобільного додатку.....	52
3.4 Апаратна складова проєкту	60
3.5 Висновки за розділом 3	63
4 Тестування інформаційної системи самоменеджменту.....	64
4.1 Конфігурація тестових середовищ.....	64
4.2 Функціональне тестування та юніт-тестування модулів	66
4.3 Тестування продуктивності мобільного клієнта та UX-тестування	72
4.4 Висновки за розділом 4	77
Висновки.....	79
Перелік джерел посилання	81
Додаток А Слайди презентації	83

ВСТУП

У сучасному світі, де обсяг інформації, завдань та соціальних зобов'язань людини постійно зростає, здатність ефективно управляти власним часом, цілями та ресурсами стає однією з ключових навичок успішної особистості. Самоменеджмент як системний підхід до організації особистої діяльності охоплює планування часу, постановку та відстеження цілей, управління звичками, контроль продуктивності та рефлексію. Попри очевидну важливість цих практик, більшість людей стикається зі значними труднощами при спробах їх систематичного застосування у повсякденному житті через відсутність зручного та інтегрованого цифрового інструменту.

Ринок мобільних застосунків для продуктивності є одним із найдинамічніших сегментів індустрії програмного забезпечення. За даними аналітиків, глобальний ринок застосунків для особистої продуктивності оцінюється у понад 5 мільярдів доларів і демонструє стабільне щорічне зростання на рівні 12–15%. Водночас переважна більшість існуючих рішень є вузькоспеціалізованими: одні сервіси орієнтовані виключно на управління завданнями, інші – лише на трекінг звичок або облік часу. Через це користувачі змушені одночасно використовувати три-чотири окремих застосунки, що суперечить самій ідеї самоменеджменту та створює зайве когнітивне навантаження.

Особливу актуальність ця проблема набуває в контексті сучасних трендів дистанційної роботи та навчання. Пандемія COVID-19 та подальше поширення форматів remote та hybrid work суттєво змінили спосіб організації робочого дня мільйонів людей. Відсутність чіткого розділення між роботою та особистим часом, необхідність самостійно структурувати свій розпорядок дня без зовнішнього контролю з боку керівництва та необхідність підтримувати баланс між професійними та особистими цілями формують стійкий запит на комплексний інструмент самоменеджменту, доступний з мобільного пристрою у будь-який момент часу.

Окремим напрямком, що стрімко розвивається у сфері особистої продуктивності, є використання аналітики та візуалізації особистих даних. Концепція «quantified self» – кількісного самопізнання через збір і аналіз даних про власну поведінку – набуває все більшого поширення. Користувачі прагнуть не просто фіксувати завдання, але й розуміти закономірності своєї продуктивності: у які дні тижня вони найефективніші, які звички позитивно впливають на досягнення цілей, скільки часу фактично витрачається на різні сфери життя. Більшість наявних застосунків не забезпечує такого рівня аналітики.

Метою даного дипломного проєкту є розробка інформаційної системи самоменеджменту у вигляді мобільного додатку, що об'єднує в єдиній платформі управління завданнями, відстеження цілей та звичок, облік часу та персональну аналітику продуктивності, забезпечуючи користувачу комплексний інструмент для організації особистої діяльності з підтримкою офлайн-режиму роботи.

Для досягнення поставленої мети необхідно виконати наступні задачі:

- провести аналіз предметної області самоменеджменту та дослідити існуючі цифрові рішення для організації особистої продуктивності;
- визначити функціональні та нефункціональні вимоги до розроблюваної системи на основі аналізу потреб цільової аудиторії;
- спроектувати архітектуру інформаційної системи з урахуванням вимог до офлайн-роботи, синхронізації та безпеки персональних даних;
- розробити структуру бази даних та схему зберігання даних користувача з підтримкою локального та хмарного сховища;
- реалізувати мобільний додаток на платформі Flutter із модулями управління завданнями, цілями, звичками та обліку часу;
- розробити серверну частину на FastAPI для хмарної синхронізації та аналітичної обробки персональних даних;
- провести комплексне тестування системи, включаючи функціональне, навантажувальне та UX-тестування.

Практична цінність розробленої системи визначається тим, що вона надає користувачу єдину інтегровану платформу для всіх аспектів самоменеджменту,

усуваючи необхідність використання кількох окремих застосунків. Підтримка повноцінного офлайн-режиму з подальшою синхронізацією дозволяє використовувати додаток незалежно від наявності інтернет-з'єднання. Вбудована аналітика та візуалізація персональних даних дає змогу користувачу усвідомлено аналізувати свою продуктивність та приймати обґрунтовані рішення щодо розподілу часу й зусиль.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ У СФЕРІ САМОМЕНЕДЖМЕНТУ

1.1 Огляд сучасного стану ринку додатків для особистої продуктивності та самоменеджменту

Самоменеджмент як наукова дисципліна та практична методологія організації особистої діяльності сформувався наприкінці ХХ століття на перетині класичного менеджменту, когнітивної психології та поведінкових наук. В основі самоменеджменту лежить принцип свідомого управління власними ресурсами часом, увагою, енергією та мотивацією з метою досягнення поставлених особистих та професійних цілей. Із поширенням смартфонів та хмарних технологій практики самоменеджменту отримали потужний цифровий інструментарій у вигляді мобільних застосунків, що дозволяють реалізовувати методики планування, відстеження та рефлексії у зручному та доступному форматі [1].

Ринок застосунків для особистої продуктивності є одним із найбільш динамічних сегментів індустрії мобільного програмного забезпечення. За різними оцінками, глобальний ринок цих застосунків перевищив позначку у 5,8 мільярда доларів у 2023 році та продовжує зростати темпами 12–15% щорічно. Ключовими рушіями зростання є поширення культури особистої продуктивності, зростання числа фрілансерів та самозайнятих осіб, а також трансформація ринку праці у бік дистанційних та гібридних форматів роботи після 2020 року [2].

Структурно ринок ділиться на кілька основних підкатегорій, кожна з яких обслуговується окремою групою спеціалізованих застосунків. Категорія управління задачами та списками справ (to-do managers) є найбільшою за кількістю застосунків та обсягом аудиторії – вона займає близько 38% ринку. Трекери часу (time trackers) утримують близько 24% ринку та використовуються переважно фрілансерами для виставлення рахунків та самоаналізу. Трекери звичок і цілей займають близько 21% та демонструють найвищі темпи зростання – понад 20% щорічно – завдяки зростаючому інтересу до wellbeing і психологічного здоров'я.

Нотатники та застосунки для збору ідей замикають ринок із часткою близько 17% [3].

Однією з найсерйозніших структурних проблем ринку є так звана фрагментація інструментарію. Дослідження показують, що середній активний користувач застосунків продуктивності використовує одночасно 3,4 окремих додатки для закриття різних аспектів самоменеджменту: окремо – менеджер задач, окремо – трекер звичок, окремо – таймер або трекер часу [4]. Ця фрагментація породжує суттєве когнітивне навантаження: дані про задачі, звички та витрачений час зберігаються у різних сховищах і не можуть бути агреговані для побудови цілісної картини особистої продуктивності. За тими ж дослідженнями, 62% нових користувачів застосунків продуктивності відмовляються від них протягом першого місяця, і основною причиною є саме складність підтримки актуальності даних одночасно в кількох системах [5].

Окремою актуальною тенденцією є зростання попиту на офлайн-функціональність мобільних застосунків. У реальних умовах використання мобільний пристрій часто перебуває в зонах з нестабільним або відсутнім інтернет-зв'язком: у метро, в літаку, у сільській місцевості, у приміщеннях з поганим покриттям. Для застосунку самоменеджменту, який пасажир використовує під час ранкової поїздки для планування дня або перегляду задач, відсутність офлайн-режиму є критичним недоліком. За опитуваннями, 74% користувачів вважають повноцінну офлайн-підтримку обов'язковою функцією, однак лише 41% з існуючих застосунків забезпечують її повною мірою [6].

Ще одним важливим трендом є концепція «quantified self» – кількісного самопізнання через збір та аналіз персональних даних. Сучасні користувачі прагнуть не просто фіксувати задачі, але й розуміти закономірності своєї продуктивності: у які дні вони найефективніші, яка тривалість робочих сесій оптимальна, які звички кореляють із виконанням цілей. Аналітичні можливості більшості існуючих застосунків є недостатніми для задоволення цього запиту: вони обмежуються базовою статистикою виконаних задач, не надаючи глибокого розуміння особистих патернів продуктивності [7].

Для українського ринку додаткову актуальність має відсутність локалізованих рішень. Менше 15% провідних застосунків для самоменеджменту мають повноцінну локалізацію українською мовою, що є суттєвим бар'єром для значної частини україномовних користувачів. Зростаючий цифровий ринок України, прискорений розвиток ІТ-індустрії та підвищення цифрової грамотності населення формують конкретний незадоволений попит на якісний україномовний інструмент самоменеджменту [8].

Ключові характеристики сучасного стану ринку застосунків для особистої продуктивності наведено у таблиці 1.1.

Таблиця 1.1 – Характеристика ринку застосунків для особистої продуктивності та самоменеджменту

Аспект	Характеристика
Обсяг ринку (2023)	Понад 5,8 млрд дол. США; очікуване зростання до 9,2 млрд дол. до 2028 року
Темп зростання	CAGR 12–15% щорічно; прискорення після 2020 року через пандемію та поширення remote work
Провідні категорії	To-do та task management (38%), time tracking (24%), habit & goal tracking (21%), note-taking (17%)
Цільова аудиторія	Фрілансери, студенти, менеджери середньої ланки, підприємці; вік 18–45 років; частка мобільних користувачів 78%
Ключові тенденції	Інтеграція AI-асистентів для автоматичного пріоритизування, гейміфікація звичок, крос-платформність та офлайн-синхронізація
Проблема фрагментації	Середній користувач використовує 3,4 окремих застосунки для самоменеджменту; 62% відмовляються від них протягом першого місяця

Продовження таблиця 1.1

Аспект	Характеристика
Офлайн-потреба	74% користувачів вважають офлайн-режим критично важливою функцією; лише 41% існуючих застосунків забезпечують повноцінний офлайн
Локалізація	Зростаючий попит на україномовні рішення; менше 15% застосунків мають повноцінну локалізацію українською мовою

Аналіз даних таблиці 1.1 підтверджує значний потенціал ринку та наявність чітко сформованого незадоволеного попиту на комплексний, україномовний застосунок самоменеджменту з повноцінним офлайн-режимом та розгорнутою аналітикою. Саме на задоволення цього попиту спрямований даний дипломний проєкт.

Паралельно з розвитком ринку відбувається активне теоретичне осмислення практик самоменеджменту. Сучасна наукова та практична база самоменеджменту включає такі ключові методики, як Getting Things Done (GTD) Девіда Аллена, систему OKR (Objectives and Key Results), методику SMART-цілей, техніку Pomodoro, Eisenhower Matrix для пріоритизації задач, концепцію habit stacking та практику weekly review [9]. Кожна з цих методик має свою специфіку та орієнтована на різні аспекти організації особистої діяльності, що обґрунтовує необхідність їх інтеграції в єдину систему. Огляд методик самоменеджменту та їх реалізацію у розроблюваній системі наведено у таблиці 1.2.

Таблиця 1.2 демонструє, що розроблювана система охоплює широкий спектр науково обґрунтованих методик самоменеджменту, що робить її теоретично обґрунтованим та практично ефективним інструментом. Комплексна підтримка цих методик в єдиному застосунку є принциповою конкурентною перевагою над наявними рішеннями.

Таблиця 1.2 – Методики самоменеджменту та їх реалізація в інформаційній системі

Методика	Основний принцип	Реалізація в системі
GTD (Getting Things Done)	Виведення всіх задач із голови в надійну систему; розподіл за контекстами та проектами	Модуль «Вхідні задачі» з пріоритизацією, теги та проекти, фільтрація за контекстом
SMART-цілі	Конкретна, вимірювана, досяжна, релевантна, обмежена в часі постановка цілей	Модуль цілей із полями специфічності, метриками прогресу та дедлайном
OKR (Objectives & Key Results)	Декомпозиція амбітної цілі на вимірювані ключові результати з квартальним плануванням	Ієрархія «Ціль → Ключові результати → Задачі» з відображенням прогресу
Pomodoro Technique	Робочі сесії по 25 хвилин із 5-хвилинними перервами для підтримки концентрації	Вбудований таймер із налаштуванням тривалості сесій та автоматичним записом у часовий журнал
Habit Stacking	Прив'язка нових звичок до вже існуючих для спрощення їх формування	Модуль звичок із можливістю пов'язувати звичку з тригером та відслідковувати ланцюжки
Time Blocking	Резервування конкретних часових блоків для роботи над певними задачами або категоріями	Інтеграція задач з часовим журналом та можливість планування блоків на день

1.2 Порівняльний аналіз існуючих систем самоменеджменту

Для обґрунтування доцільності розробки нової системи та формування конкурентних вимог до неї необхідно провести детальний аналіз провідних існуючих рішень на ринку. Об'єктами аналізу обрано три найбільш поширені та представницькі продукти, що разом охоплюють весь спектр підходів до організації особистої продуктивності: Notion як «all-in-one workspace», Todoist як класичний мінімалістичний менеджер задач та TickTick як найбільш функціонально наближений до розроблюваної системи комплексний інструмент продуктивності [10].

Notion є одним із найгучніших стартапів у сфері продуктивності останнього десятиліття. Заснований у 2016 році, він швидко зібрав аудиторію мільйонів користувачів завдяки унікальній концепції «блочного» конструктора, що дозволяє будувати будь-які структури даних – від простого списку завдань до повноцінної CRM-системи або бази знань компанії. Характеристики Notion наведено у таблиці 1.3.

Таблиця 1.3 – Характеристика системи Notion

Характеристика	Опис
Компанія та рік заснування	Notion Labs Inc., США, 2016; оцінка компанії 10 млрд дол. станом на 2021 рік
Концепція	«All-in-one workspace» – поєднання нотаток, баз даних, вікі та управління проєктами в єдиному середовищі
Платформи	Web, macOS, Windows, iOS, Android; основний акцент на desktop-версії
Ключові функції	Блочний редактор, реляційні бази даних, шаблони, командна робота, Notion AI для генерації контенту

Продовження таблиці 1.3

Характеристика	Опис
Сильні сторони	Надзвичайна гнучкість налаштування, потужні реляційні бази даних, великий маркетплейс шаблонів, активна спільнота
Слабкі сторони	Висока крива навчання, відсутність повноцінного офлайн-режиму, немає вбудованого трекера звичок та обліку часу, перевантаженість функціями

Незважаючи на вражаючу гнучкість і функціональність, Notion має принципові обмеження з точки зору застосунку для особистого самоменеджменту. Найбільш критичним є надзвичайно висока крива навчання: середній новий користувач витрачає від двох до чотирьох тижнів лише на те, щоб налаштувати робочий простір під свої потреби. Це суперечить основній меті самоменеджменту – зниженню когнітивного навантаження. Відсутність повноцінного офлайн-режиму є ще одним суттєвим недоліком: Notion кешує контент для попереднього перегляду, але не дозволяє повноцінно редагувати та синхронізувати дані без інтернет-з'єднання. Вбудованих модулів трекера звичок та обліку часу в Notion немає взагалі їх можна відтворити через шаблони баз даних, але це потребує значних налаштувань.

Todoist є еталонним прикладом мінімалістичного менеджера задач із чітким фокусом на єдиній функції та доведенням її до досконалості [11]. Заснований у 2007 році, він нараховує понад 30 мільйонів зареєстрованих користувачів і регулярно очолює рейтинги кращих застосунків для управління задачами. Детальні характеристики Todoist наведено у таблиці 1.4.

Todoist демонструє найвищу серед конкурентів якість реалізації своєї ключової функції – управління задачами. Система природномовного введення дозволяє створити задачу з повторенням, пріоритетом та проектом одним рядком тексту, наприклад «Підготувати звіт щопонеділка о 9:00 @Робота p1», що є видатним UX-рішенням. Однак за межами управління задачами Todoist практично

не надає нічого. Відсутність трекера звичок, модуля цілей та обліку часу змушує користувачів використовувати додаткові застосунки, повертаючи нас до вже описаної проблеми фрагментації.

Таблиця 1.4 – Характеристика системи Todoist

Характеристика	Опис
Компанія та рік заснування	Doist Inc., Естонія/Сінгапур, 2007; понад 30 млн зареєстрованих користувачів
Концепція	Найпопулярніший у світі менеджер задач із фокусом на простоті та швидкості введення завдань
Платформи	Web, macOS, Windows, Linux, iOS, Android, браузерні розширення; понад 80 інтеграцій
Ключові функції	Природномовне введення задач, Karma-система мотивації, фільтри та мітки, повторювані задачі, командні проєкти
Модель монетизації	Freemium; Pro від 5 дол./міс., Business від 8 дол./місяць на користувача
Сильні сторони	Чистий мінімалістичний інтерфейс, найкраще природномовне введення серед конкурентів, надійна офлайн-підтримка, широкі інтеграції
Слабкі сторони	Відсутність трекера звичок і цілей, немає вбудованого обліку часу, обмежена аналітика на безкоштовному тарифі, немає персональної аналітики продуктивності

Аналітика обмежена Karma-системою, яка показує загальну динаміку виконання задач, але не надає інсайтів про розподіл часу або прогрес за категоріями цілей.

TickTick є найближчим аналогом розроблюваної системи за набором функцій і заслуговує найбільш детального розгляду [12]. Заснований у 2013 році

китайською командою, цей застосунок вирізняється з-поміж конкурентів спробою об'єднати управління задачами, трекер звичок та інструменти фокусування в одному продукті. Характеристики TickTick наведено у таблиці 1.5.

Таблиця 1.5 – Характеристика системи TickTick

Характеристика	Опис
Компанія та рік заснування	Arrest Inc., Китай/США, 2013; понад 10 млн активних користувачів щомісяця
Концепція	Комплексний менеджер продуктивності з унікальним поєднанням задач, звичок та вбудованого Pomodoro-таймера
Платформи	Web, macOS, Windows, iOS, Android, Apple Watch, браузерні розширення
Ключові функції	Управління задачами, трекер звичок, Pomodoro-таймер, Eisenhower Matrix, календарна інтеграція, прив'язка завдань до цілей
Модель монетизації	Freemium; Premium від 3,99 дол./міс. або 27,99 дол./рік – одна з найнижчих цін на ринку
Сильні сторони	Найбільш збалансований набір функцій серед конкурентів, вбудований таймер, наявність трекера звичок, доступна ціна
Слабкі сторони	Обмежена аналітика продуктивності, немає повноцінного модуля цілей із декомпозицією, UX перевантажений, відсутня персоналізована статистика

TickTick демонструє найбільш збалансований набір функцій серед проаналізованих аналогів, проте також має суттєві обмеження. Модуль звичок є базовим: він дозволяє відмічати виконання звичок та бачити streak (серію), але не підтримує складних налаштувань, прив'язки звичок до цілей або детального

аналізу. Відсутність повноцінного модуля цілей із підтримкою OKR-структури є принциповим недоліком для користувачів, які практикують методику Objectives and Key Results. Аналітика продуктивності обмежена базовими графіками виконання задач без можливості корелювати дані з різних модулів. UX-інтерфейс перевантажений елементами управління, що знижує зручність використання для нових користувачів.

Зведений порівняльний аналіз усіх трьох аналогів та розроблюваної системи наведено у таблиці 1.6.

Таблиця 1.6 – Зведений порівняльний аналіз існуючих рішень та розроблюваної системи

Критерій	Notion	Todoist	TickTick	Розроблювана система
Управління задачами	Так (бази даних)	Так (основна функція)	Так	Так
Трекер звичок	Ні (через шаблони)	Ні	Так (базовий)	Так (розширений)
Управління цілями	Частково	Ні	Частково	Так (OKR/SMART)
Облік часу (Time Tracking)	Ні	Ні	Pomodoro лише	Так (ручний + таймер)
Персональна аналітика	Обмежена	Karma-статистика	Базова	Так (детальна)
Офлайн-режим	Ні (тільки кеш)	Так	Частково	Так (повний)
Щоденник / рефлексія	Через нотатки	Ні	Ні	Так (вбудований)
Українська мова	Ні	Ні	Ні	Так

Продовження таблиці 1.6

Критерій	Notion	Todoist	TickTick	Розроблювана система
Безкоштовний тариф	Так (обмежений)	Так (обмежений)	Так (обмежений)	Так (базові функції)
Платформа мобільного клієнта	React Native	React Native	React Native	Flutter (Dart)
Гейміфікація	Ні	Karma-система	Частково	Так (досягнення, streaks)

Аналіз таблиці 1.6 дозволяє зробити кілька ключових висновків. По-перше, жоден із проаналізованих аналогів не забезпечує повного покриття всіх аспектів самоменеджменту в єдиній системі: Notion є надмірно гнучким і складним, Todoist – надто вузькоспеціалізованим, TickTick – найближчим, але все одно неповним. По-друге, жоден із них не має повноцінної україномовної локалізації. По-третє, жоден не надає інтегрованої персональної аналітики, що об'єднує дані про задачі, час, звички та цілі в єдину картину. По-четверте, підтримка офлайн-режиму є або відсутньою, або частковою. Саме ці прогалини визначають конкурентні переваги розроблюваної системи та обґрунтовують доцільність її розробки.

Принципово важливою відмінністю розроблюваної системи є вибір Flutter як платформи для мобільного клієнта. Усі три проаналізовані аналоги використовують React Native, що є рефлексом індустріального тренду попереднього покоління. Flutter, розроблений Google, забезпечує більш плавну анімацію, кращу продуктивність на бюджетних пристроях та повністю власний рендеринг UI без залежності від нативних компонентів операційної системи. Це є принциповою перевагою для застосунку самоменеджменту, де плавність взаємодії та швидкість відгуку безпосередньо впливають на утримання користувачів [13].

1.3 Вимоги до розробки інформаційної системи самоменеджменту

На основі проведеного аналізу ринку, методик самоменеджменту та існуючих аналогів сформовано комплексні вимоги до розроблюваної інформаційної системи. Ці вимоги поділяються на функціональні, що описують конкретні можливості системи та нефункціональні, що визначають якісні характеристики та умови роботи.

Функціональні вимоги до системи формуються навколо п'яти ключових модулів, кожен з яких реалізує відповідну практику самоменеджменту. Перший модуль управління задачами повинен забезпечувати створення задач із природномовним введенням дедлайну, присвоєння пріоритетів та міток, організацію задач у проєкти та підзадачі, встановлення нагадувань та повторення. Другий модуль управління цілями повинен підтримувати методологію OKR із декомпозицією цілі на ключові результати та прив'язку задач до конкретних результатів. Третій модуль трекер звичок повинен дозволяти визначати щоденні та тижневі звички, відмічати їх виконання, відстежувати streak-серії та аналізувати закономірності формування. Четвертий модуль облік часу повинен надавати можливість ручного введення витраченого часу та роботи з Pomodoro-таймером із прив'язкою кожної сесії до конкретної задачі або проєкту. П'ятий модуль щоденник та рефлексія повинен забезпечувати ведення щоденника із структурованими шаблонами для ранкового планування та вечірньої рефлексії [14].

Шостим функціональним блоком є модуль аналітики та статистики, що є наскрізним і агрегує дані з усіх інших модулів. Аналітика повинна включати візуалізацію розподілу часу за проєктами та категоріями у вигляді кругових та стовпчастих діаграм, графіки динаміки виконання задач та дотримання звичок за тижнями та місяцями, відображення прогресу досягнення цілей у відсотках та карту продуктивності у стилі GitHub contribution graph матрицю активності за днями тижня та місяця. Сьомий функціональний блок авторизація та синхронізація повинен забезпечувати реєстрацію через email та соціальні мережі, наскрізну

шифровану синхронізацію між пристроями та повноцінний офлайн-режим із вирішенням конфліктів злиття при синхронізації.

Нефункціональні вимоги до системи визначають технічні обмеження та якісні характеристики. Час запуску додатку не повинен перевищувати 2 секунди на пристроях середнього класу; час відгуку на дії користувача не більше 100 мс для локальних операцій та 500 мс для операцій, що потребують мережевого запиту. Система повинна коректно функціонувати на пристроях з Android 8.0 і вище та iOS 14 і вище, що охоплює понад 95% активних мобільних пристроїв. Обсяг локального сховища, необхідного для зберігання даних одного року використання, не повинен перевищувати 50 МБ. Усі персональні дані користувача повинні зберігатися у зашифрованому вигляді як локально, так і у хмарному сховищі [15].

Вимоги до офлайн-функціональності є принциповими для розроблюваної системи і формулюються окремо. Усі основні операції створення задач, відмітка виконання звичок, запуск таймера, ведення щоденника повинні бути доступні без інтернет-з'єднання. При відновленні зв'язку синхронізація з хмарию повинна відбуватися автоматично у фоновому режимі без будь-яких дій з боку користувача. Конфлікти злиття, що виникають при одночасному редагуванні одних і тих самих даних на кількох пристроях у режимі офлайн, повинні вирішуватися автоматично за принципом «остання зміна перемагає» для простих полів та збереженням обох версій для нотаток і щоденника [16].

Огляд ключових завдань розробки інформаційної системи самоменеджменту наведено у таблиці 1.7.

Таблиця 1.7 охоплює повний цикл розробки від аналізу до публікації у магазинах застосунків. Кожен етап є логічно обґрунтованим продовженням попереднього. Особливою вимогою є орієнтація на мобільну платформу як основну: усі функції системи повинні бути доступні через мобільний інтерфейс, а веб-версія або настільний клієнт є опціональними розширеннями, що можуть бути реалізовані у подальших ітераціях.

Таблиця 1.7 – Ключові завдання розробки інформаційної системи самоменеджменту

Етап розробки	Опис завдань
Аналіз предметної області	Дослідження методик самоменеджменту (GTD, OKR, SMART, Pomodoro) та виявлення потреб цільової аудиторії
Аналіз конкурентних рішень	Порівняльний аналіз Notion, Todoist, TickTick для визначення конкурентних переваг розроблюваної системи
Формування вимог SRS	Складання повної специфікації функціональних та нефункціональних вимог до системи
Проектування архітектури	Визначення архітектурного патерну (Clean Architecture + BLoC), структури модулів та API
Проектування бази даних	Розробка схеми локальної БД Hive та хмарної структури Firebase Firestore для синхронізації
Розробка мобільного додатку	Реалізація всіх функціональних модулів на Flutter з підтримкою iOS та Android
Розробка серверної частини	Реалізація FastAPI-сервера для синхронізації, аналітики та push-сповіщень
Тестування	Функціональне, навантажувальне та UX-тестування з реальними користувачами
Впровадження	Публікація у Google Play та App Store, налаштування CI/CD пайплайну

Вимоги до дизайну та користувацького досвіду (UX) відіграють особливу роль у застосунку самоменеджменту, оскільки навіть функціонально потужний продукт зазнає провалу, якщо він незручний у щоденному використанні. Інтерфейс повинен дотримуватися принципів Material Design 3 для Android та Human Interface Guidelines для iOS. Навігація повинна бути побудована за принципом «три натискання»: будь-яка функція системи повинна бути доступна не більше ніж за три

кроки від головного екрану. Кольорова схема має підтримувати темну та світлу теми з автоматичним перемиканням за системними налаштуваннями. Мінімальний розмір інтерактивних елементів повинен становити 48×48 dp для забезпечення зручного натискання великим пальцем.

1.4 Висновки за розділом 1

У першому розділі проведено комплексний аналіз предметної області самоменеджменту та ринку відповідних цифрових рішень, що дозволяє обґрунтувати доцільність та визначити напрямок розробки інформаційної системи самоменеджменту.

Аналіз ринку застосунків для особистої продуктивності підтвердив значний і зростаючий попит на інструменти самоменеджменту. Ринок оцінюється у 5,8 млрд доларів із темпом зростання 12–15% щорічно. Виявлено ключові незадоволені потреби: 62% користувачів відмовляються від застосунків через фрагментацію інструментарію, 74% вимагають повноцінного офлайн-режиму, менше 15% існуючих застосунків мають україномовну локалізацію. Дослідження методик самоменеджменту – GTD, OKR, SMART, Pomodoro, habit stacking та weekly review – підтвердило необхідність комплексного підходу до реалізації системи, що охоплює всі аспекти організації особистої діяльності.

Порівняльний аналіз трьох провідних конкурентів показав, що жоден із них не забезпечує повного покриття потреб користувача в єдиному рішенні. Notion є надто складним і не підходить для мобільного використання. Todoist є чудовим менеджером задач, але не більше. TickTick найближчий до концепції розроблюваної системи, однак має обмежену аналітику, відсутній модуль OKR-цілей та часткову офлайн-підтримку. Жоден із аналогів не має україномовної локалізації та детальної персональної аналітики продуктивності.

Сформовані вимоги до інформаційної системи самоменеджменту охоплюють сім функціональних блоків: управління задачами, управління цілями (OKR/SMART), трекер звичок, облік часу (ручний + Pomodoro), щоденник і рефлексія, аналітика продуктивності та синхронізація. Нефункціональні вимоги забезпечують продуктивність, офлайн-доступність, безпеку персональних даних та зручність мобільного використання. Вибір Flutter як платформи мобільного клієнта обґрунтовується перевагами у продуктивності та плавності анімацій порівняно з React Native, що використовується у всіх проаналізованих аналогах. Зібрані матеріали першого розділу є повною основою для проєктування архітектури системи у наступному розділі.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ САМОМЕНЕДЖМЕНТУ

2.1 Розробка специфікації вимог до системи самоменеджменту

Специфікація вимог є фундаментальним документом, що визначає межі та поведінку системи на всіх рівнях взаємодії. Для інформаційної системи самоменеджменту специфікація вимог відображає унікальну особливість продукту – необхідність одночасно задовольняти потреби різних аспектів особистої продуктивності в єдиній платформі, зберігаючи простоту та зручність повсякденного використання [17]. Вимоги сформовано за результатами аналізу методик самоменеджменту та порівняльного аналізу конкурентів, проведеного у першому розділі.

Функціональні вимоги систематизовано за модульним принципом: кожен із семи функціональних блоків системи отримав власний набір специфічних вимог із унікальними кодами для забезпечення трасабельності. Перелік ключових функціональних вимог наведено у таблиці 2.1.

Таблиця 2.1 – Функціональні вимоги до інформаційної системи самоменеджменту

Код	Модуль	Опис функціональної вимоги
FR-01	Задачі	Система повинна забезпечувати створення задач із полями: назва, опис, пріоритет (1–4), дедлайн, теги, проєкт, підзадачі. Підтримка природномовного введення дати
FR-02	Задачі	Відображення задач у кількох режимах: список, дошка Kanban, календар. Фільтрація за проєктом, тегом, пріоритетом та дедлайном
FR-03	Цілі	Підтримка методології OKR: ціль (Objective) містить від 2 до 5 ключових результатів (Key Results) із числовою метрикою прогресу
FR-04	Цілі	Прив'язка задач до ключових результатів. Автоматичний перерахунок прогресу цілі при виконанні пов'язаних задач
FR-05	Звички	Створення щоденних та тижневих звичок із налаштуванням днів виконання, кольору та нагадування. Відмітка виконання одним натисканням
FR-06	Звички	Автоматичний підрахунок поточного streak та найкращого streak. Відображення календарної теплової карти виконання за 3 місяці

Продовження таблиці 2.1

Код	Модуль	Опис функціональної вимоги
FR-07	Час	Ручне введення витраченого часу з прив'язкою до задачі або проєкту. Вбудований Pomodoro-таймер із налаштуванням тривалості сесій (15–60 хв)
FR-08	Щоденник	Ведення щоденних нотаток із шаблонами: ранкове планування (3 пріоритети дня) та вечірня рефлексія (що зроблено / що завтра)
FR-09	Аналітика	Тижнева та місячна статистика: виконані задачі, дотримання звичок, витрачений час за категоріями. Графіки та кругові діаграми
FR-10	Аналітика	Productivity Score – інтегральний показник продуктивності за день (0–100), що враховує виконані задачі, дотримані звички та зареєстрований час
FR-11	Синхронізація	Повноцінний офлайн-режим: усі операції доступні без мережі. Автоматична фонові синхронізація з Firebase Firestore при відновленні з'єднання
FR-12	Авторизація	Реєстрація та вхід через email/пароль, Google Sign-In та Apple Sign-In (Supabase Auth). Вихід та видалення акаунта з очищенням усіх даних

Аналіз функціональних вимог таблиці 2.1 свідчить про комплексний охват усіх аспектів самоменеджменту. Вимоги FR-01 та FR-02 визначають базовий функціонал управління задачами з підтримкою кількох режимів відображення. Вимоги FR-03 та FR-04 реалізують методологію OKR через ієрархію «ціль – ключові результати – задачі» з автоматичним розрахунком прогресу. Вимоги FR-05 та FR-06 формують повноцінний модуль трекера звичок зі streak-механікою. FR-07 покриває облік часу у двох режимах. FR-08 забезпечує практику щоденної рефлексії. FR-09 та FR-10 формують аналітичний блок з унікальним Productivity Score. FR-11 та FR-12 визначають інфраструктурні вимоги до офлайн-роботи та авторизації.

Загальна схема взаємодії між акторами та функціями системи представлена у вигляді UML-діаграми варіантів використання на рисунку 2.1.

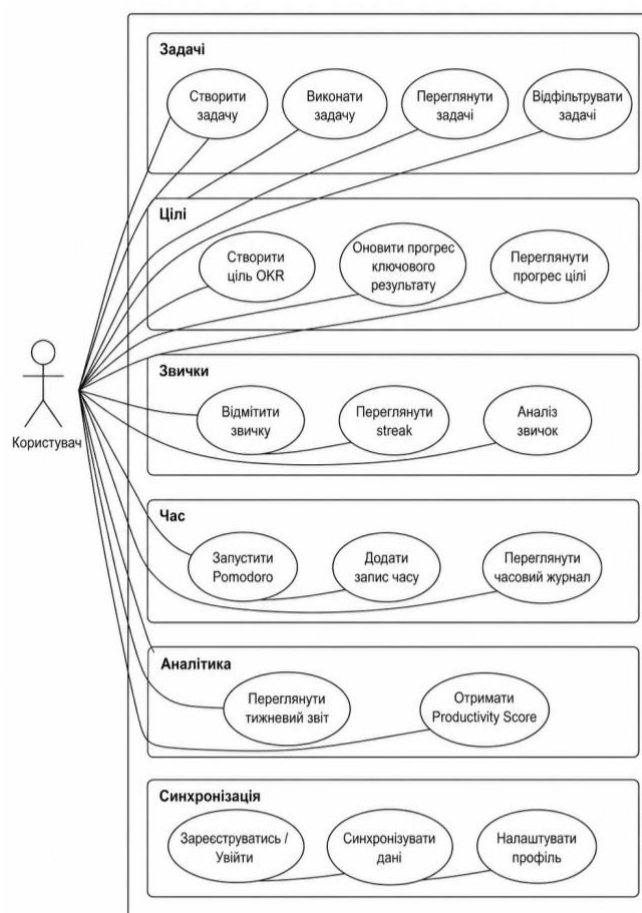


Рисунок 2.1 – UML-діаграма варіантів використання системи

Нефункціональні вимоги визначають якісні характеристики системи, що є не менш важливими, ніж функціональні, для продукту щоденного мобільного використання. Перелік нефункціональних вимог із кількісними показниками наведено у таблиці 2.2.

Таблиця 2.2 – Нефункціональні вимоги до інформаційної системи

Категорія	Показник	Значення та опис
Продуктивність	Час запуску	Не більше 2 с на пристроях середнього класу (Snapdragon 665 / Apple A12 і вище)
Продуктивність	Відгук на дію	Локальні операції (CRUD у Hive) – не більше 100 мс; мережеві – не більше 500 мс
Продуктивність	Плавність UI	Не менше 60 fps під час прокручування списків та анімацій переходу між екранами
Сумісність	Android	Android 8.0 (API 26) і вище; охоплення не менше 95% активних пристроїв
Сумісність	iOS	iOS 14.0 і вище; підтримка iPhone SE (2nd gen) та iPad як додаткової платформи

Продовження таблиці 2.2

Категорія	Показник	Значення та опис
Сховище	Локальне	Не більше 50 МБ для даних одного року активного використання (без медіа-вкладень)
Безпека	Шифрування	Локальне сховище Hive зашифроване AES-256. Передача даних – TLS 1.3. Firebase Security Rules обмежують доступ до даних власником
Безпека	Автентифікація	JWT-токени Supabase з терміном дії 1 година; refresh-token зберігається у Flutter Secure Storage (Keychain / Keystore)
Доступність	Офлайн	100% базових функцій доступні без мережевого з'єднання. Синхронізація у фоні при відновленні
Локалізація	Мови	Українська (основна) та англійська (додаткова). Автоматичний вибір за системними налаштуваннями

Особливої уваги заслуговують вимоги до Productivity Score (FR-10) – унікального показника, що відсутній у всіх проаналізованих аналогах. Формула розрахунку Productivity Score базується на зваженій сумі трьох компонентів: частка

виконаних задач від запланованих на день (вага 40%), частка дотриманих звичок від запланованих на цей день тижня (вага 35%) та відношення зафіксованого часу до встановленого денного ліміту (вага 25%). Максимальне значення становить 100 балів. Такий підхід дозволяє отримати єдину узагальнену оцінку дня, що враховує всі аспекти самоменеджменту одночасно [18].

Вимоги до офлайн-режиму є системоутворюючими для всієї архітектури і визначають принципову стратегію «Offline-First»: система проєктується таким чином, що локальне сховище є першочерговим джерелом даних, а хмара – лише механізмом синхронізації та резервного копіювання. Це протилежно підходу більшості конкурентів, де хмара є основним сховищем, а локальний кеш – лише оптимізацією.

2.2 Архітектура інформаційної системи та вибір патернів проєктування

Проектування архітектури інформаційної системи самоменеджменту потребує врахування специфічних вимог: підтримки повноцінного офлайн-режиму, управління складним станом застосунку із семи взаємопов'язаних модулів та забезпечення високої тестованості коду. Для вибору оптимального архітектурного патерну проведено порівняльний аналіз трьох основних підходів: MVC, MVVM із Provider та Clean Architecture з Riverpod [19]. Результати порівняння наведено у таблиці 2.3.

За результатами аналізу таблиці 2.3 обрано Clean Architecture у поєднанні з Riverpod як державним менеджером. Clean Architecture передбачає поділ системи на чотири ізольовані рівні з однонаправленою залежністю: рівень представлення, рівень застосунку, доменний рівень та рівень даних. Ключовою перевагою цього підходу для задач офлайн-системи є Repository Pattern на доменному рівні: інтерфейс репозиторію абстрагує джерело даних від бізнес-логіки, дозволяючи реалізаціям безшовно перемикатися між локальним Hive та хмарним Firestore.

Таблиця 2.3 – Порівняльний аналіз архітектурних патернів для Flutter-застосунку

Критерій	MVC	MVVM + Provider	Clean Arch + Riverpod (обрано)
Розділення відповідальності	Часткове	Добре	Відмінне (4 рівні)
Тестованість	Складна	Добра	Відмінна (ізольовані use cases)
Підтримка офлайн	Вимагає патчів	Потребує рефакторингу	Вбудована (Repository pattern)
Масштабованість	Низька	Середня	Висока
Складність впровадження	Низька	Середня	Вища (виправдана для 7 модулів)
Відповідність задачі	Незадовільна	Прийнятна	Оптимальна

Загальна архітектура системи з розподілом компонентів по рівнях наведена на рисунку 2.2. Схема відображає чотири концентричні рівні архітектури: зовнішній – Presentation Layer (Flutter Widgets та Riverpod Providers), другий – Application Layer (Use Cases реалізовані через Riverpod Notifiers), третій – Domain Layer (Entities та Repository Interfaces) та внутрішній – Data Layer (Hive, Firebase Firestore, FastAPI та Supabase Auth). Стрілки між рівнями спрямовані лише ззовні всередину, що унеможливорює порушення принципу інверсії залежностей.

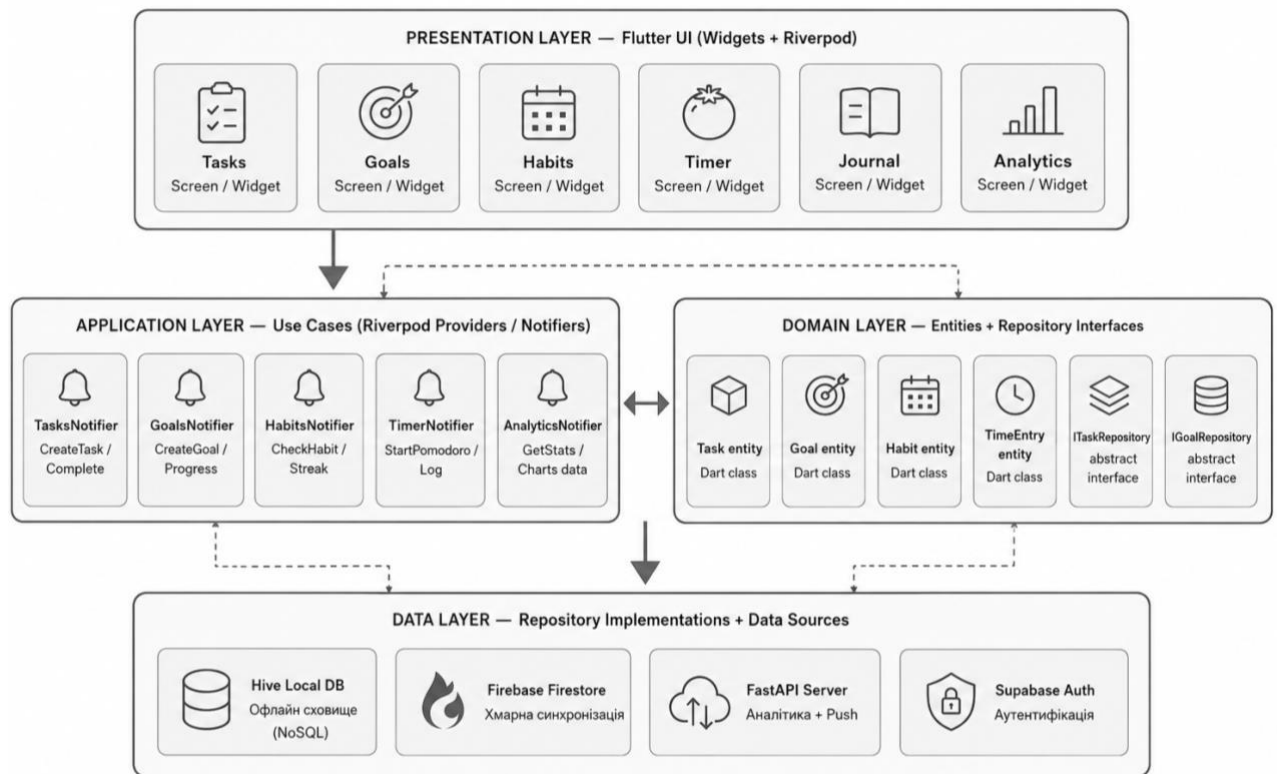


Рисунок 2.2 – Архітектура інформаційної системи самоменеджменту (Clean Architecture)

Рівень представлення (Presentation Layer) реалізує виключно логіку відображення: Flutter-віджети підписуються на стан через Riverpod Consumer та відображають дані, що надходять від Notifier-провайдерів. Жодної бізнес-логіки на цьому рівні немає – будь-яка дія користувача транслюється у виклик методу відповідного Notifier. Такий підхід забезпечує повну ізоляцію UI від бізнес-логіки та дозволяє легко тестувати кожен компонент незалежно.

Рівень застосунку (Application Layer) містить Use Cases, реалізовані як Riverpod Notifiers. Кожен модуль системи має власний Notifier: TasksNotifier, GoalsNotifier, HabitsNotifier, TimerNotifier та AnalyticsNotifier. Notifier оркеструє бізнес-логіку, звертаючись до репозиторіїв через їхні інтерфейси. Riverpod забезпечує реактивне оновлення UI при зміні стану та підтримку асинхронних потоків даних через AsyncValue та StreamProvider.

Доменний рівень (Domain Layer) є ядром системи та не залежить від жодного зовнішнього фреймворку. Він містить Dart-класи ентитетів (Task, Goal,

Habit, TimeEntry, JournalEntry) та абстрактні інтерфейси репозиторіїв (ITaskRepository, IGoalRepository тощо). Ентитети є імутабельними value objects, що виключає неочікувані мутації стану. Це є принциповою перевагою у контексті Riverpod, який використовує рівність об'єктів для визначення необхідності перебудови UI.

Рівень даних (Data Layer) містить конкретні реалізації репозиторіїв, що реалізують інтерфейси доменного рівня. Кожен репозиторій координує роботу двох джерел даних: локального (Hive) та хмарного (Firebase Firestore). При виконанні операції читання репозиторій завжди повертає локальні дані, забезпечуючи миттєвий відгук. При виконанні операції запису дані зберігаються локально, після чого операція додається до SyncQueue для подальшої передачі в хмару.

Взаємодія між рівнями архітектури при виконанні типового сценарію відображена на UML-діаграмі послідовності на рисунку 2.3.

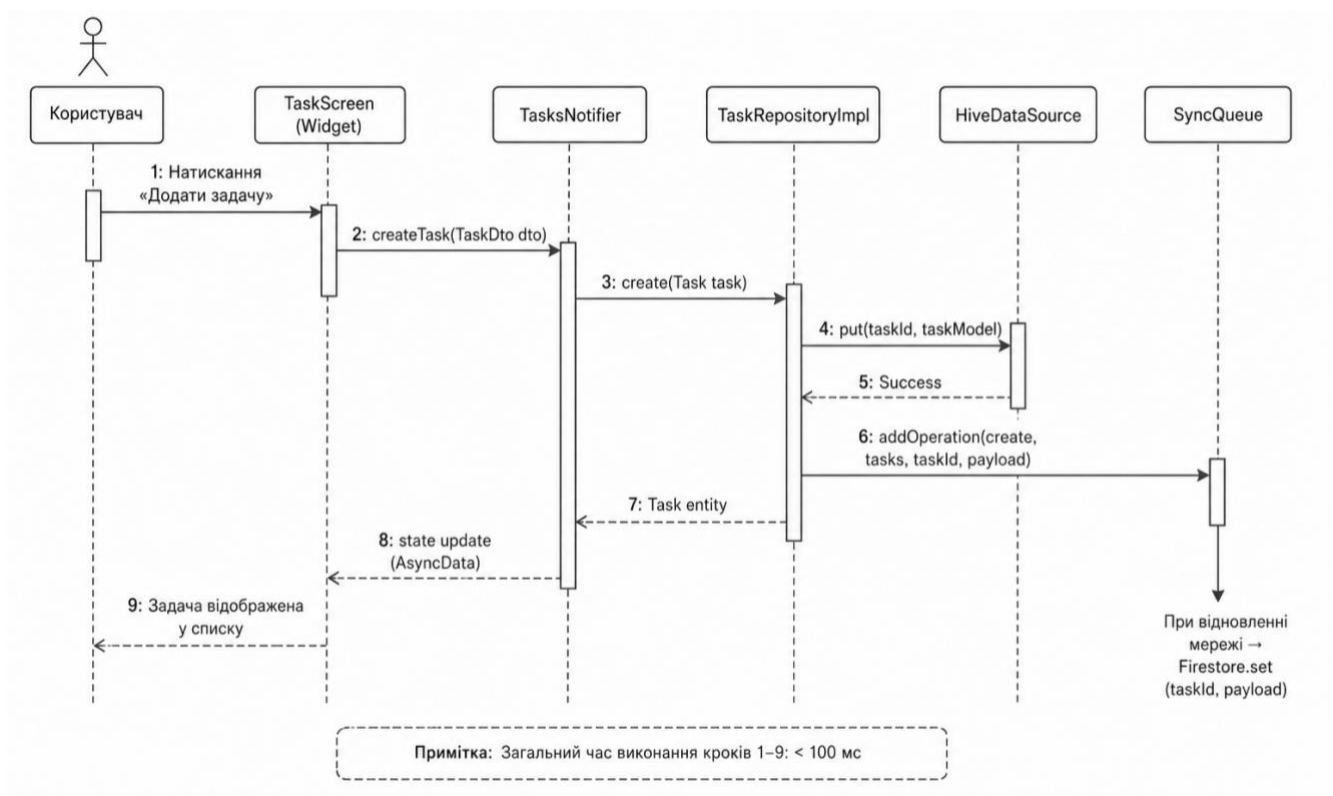


Рисунок 2.3 – UML-діаграма послідовності створення задачі в офлайн-режимі

Riverpod обрано як державний менеджер замість альтернатив (BLoC, Provider, GetX) з кількох причин. По-перше, Riverpod підтримує compile-time safety: звернення до неіснуючого провайдера є помилкою компіляції, а не runtime exception. По-друге, Riverpod забезпечує автоматичне disposal провайдерів при зникненні споживачів, що запобігає витокам пам'яті у довготривалих потоках даних. По-третє, підтримка AsyncValue уніфікує обробку стану loading/data/error без шаблонного коду. По-четверте, механізм Notifier/AsyncNotifier є природним місцем для розміщення Use Cases, що ідеально відповідає обраній Clean Architecture.

Схему офлайн-синхронізації даних, що є ключовим елементом архітектури, наведено на рисунку 2.4. Схема відображає стратегію Offline-First: після дії користувача дані миттєво записуються у Hive і UI оновлюється без затримки. Одночасно зміна додається до SyncQueue. При відновленні мережі WorkManager запускає фоновий процес, що читає чергу та передає зміни до Firebase Firestore із вирішенням конфліктів за принципом Last Write Wins (LWW).

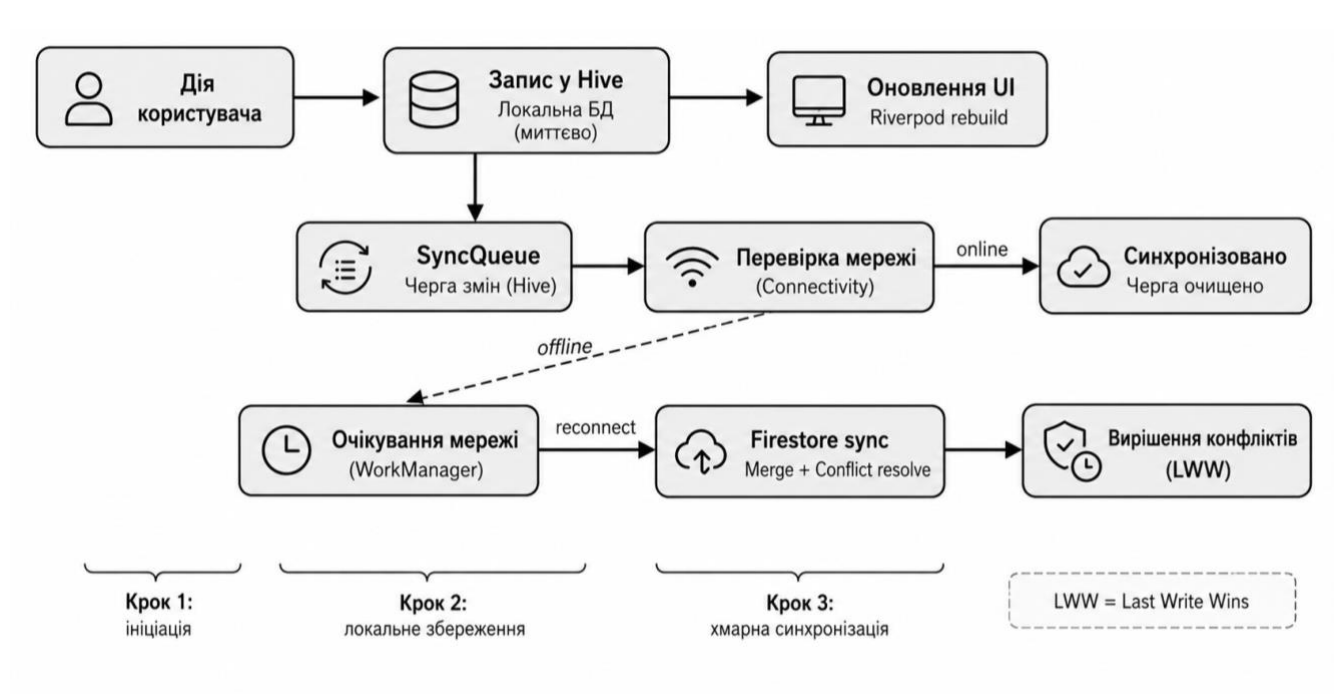


Рисунок 2.4 – Схема офлайн-синхронізації даних (Offline-First Strategy)

Вирішення конфліктів синхронізації є важливим аспектом офлайн-систем. Для більшості типів даних (задачі, звички, часові записи) застосовується принцип LWW: документ з більш пізньою міткою `updatedAt` вважається актуальним і перезаписує старішу версію [20]. Для записів щоденника застосовується особливий підхід: оскільки текстові нотатки є унікальним творчим контентом і їх автоматичне злиття є проблематичним, система зберігає обидві версії як `version_A` та `version_B` та повідомляє користувача про необхідність ручного вирішення конфлікту. Такий підхід забезпечує, що жодна нотатка користувача не буде втрачена.

2.3 Проєктування бази даних та структури зберігання даних користувача

Система зберігання даних інформаційної системи самоменеджменту реалізована на двох рівнях: локальному (Hive NoSQL-сховище на пристрої) та хмарному (Firebase Firestore). Такий дворівневий підхід є прямим наслідком вимоги FR-11 про повноцінний офлайн-режим і є архітектурно відмінним від підходів більшості конкурентів.

Hive обрано як локальне сховище замість альтернатив (SQLite, Isar, ObjectBox) з кількох причин. По-перше, Hive є чистим Dart-рішенням без нативних залежностей, що спрощує підтримку крос-платформного Flutter-застосунку. По-друге, Hive підтримує шифрування AES-256 «з коробки», що є вимогою безпеки для зберігання персональних даних. По-третє, Hive надає `TypeAdapters` для автоматичної серіалізації/десеріалізації Dart-об'єктів, що мінімізує шаблонний код. По-четверте, запис та читання в Hive є синхронними операціями з обміном пам'яттю, що забезпечує затримку менше 1 мс для типових операцій CRUD.

Локальне сховище Hive організоване у вигляді семи окремих «коробок» (Box), кожна з яких відповідає одному типу даних. Опис призначення та структури кожної Hive Box наведено у таблиці 2.4.

Таблиця 2.4 – Структура локального сховища Hive

Hive Box	Призначення та ключові поля
tasks_box	Локальне сховище задач. Ключ: taskId (String UUID). Значення: TaskModel з полями title, description, priority, dueDate, isCompleted, projectId, tags, updatedAt, syncStatus
goals_box	Цілі та OKR. Ключ: goalId. Значення: GoalModel з полями title, type, targetDate, progress, keyResults (List<KeyResultModel>), status, updatedAt
habits_box	Звички. Ключ: habitId. Значення: HabitModel з полями title, frequency, targetDays, completedDates (List<DateTime>), currentStreak, bestStreak, color, goalId
time_entries_box	Записи обліку часу. Ключ: entryId. Значення: TimeEntryModel з полями taskId, projectId, startedAt, duration, type (manual/pomodoro), note
journal_box	Записи щоденника. Ключ: date (рядок YYYY-MM-DD). Значення: JournalModel з полями morningPriorities (List<String>), eveningReflection (Map), notes (String)
sync_queue_box	Черга змін для синхронізації з Firestore. Ключ: UUID. Значення: SyncOperation з полями collection, docId, operation (create/update/delete), payload, createdAt
settings_box	Налаштування користувача. Ключ: константа. Значення: SettingsModel з полями theme, language, workDayStart, pomodoroLength, notificationsEnabled

Особливою є структура sync_queue_box, що реалізує патерн «Outbox Pattern» для надійної доставки змін до хмари. Кожен запис у черзі містить

ідентифікатор колекції Firestore, ідентифікатор документа, тип операції (create/update/delete), повний payload документа та часову мітку створення операції. При успішній синхронізації запис видаляється з черги. При невдалій – залишається для повторної спроби з використанням експоненційного backoff. Це гарантує доставку змін навіть при нестабільному з'єднанні та дозволяє відновити повну послідовність змін після тривалого офлайн-режиму.

Хмарне сховище Firebase Firestore організоване за ієрархічним принципом підколекцій. Кореневий документ користувача зберігається у колекції `users/{userId}`, а всі дані, що належать цьому користувачу, розміщені у підколекціях цього документа. Такий підхід забезпечує природне розмежування даних між користувачами та спрощує реалізацію Firebase Security Rules. Опис колекцій Firestore наведено у таблиці 2.5.

Таблиця 2.5 – Структура колекцій Firebase Firestore

Колекція Firestore	Опис та принципи синхронізації
<code>users/{userId}</code>	Профіль користувача та налаштування. Документ створюється при першій авторизації. Синхронізується при зміні налаштувань
<code>users/{userId}/tasks</code>	Підколекція задач користувача. Кожен документ відповідає одній задачі. Security Rule дозволяє доступ лише автентифікованому власнику
<code>users/{userId}/goals</code>	Підколекція цілей. Містить вкладені масиви <code>keyResults</code> . Повне оновлення документа при кожній зміні для спрощення conflict resolution

Продовження таблиці 2.5

Колекція Firestore	Опис та принципи синхронізації
users/{userId}/habits	Підколекція звичок із вкладеним масивом <code>completedDates</code> . Оновлення відбувається щодня при відмітці виконання. <code>Streak</code> розраховується на клієнті
users/{userId}/timeEntries	Підколекція записів часу. Партиціонована за місяцями через поле <code>yearMonth</code> для ефективної фільтрації аналітичних запитів
users/{userId}/journal	Підколекція записів щоденника. Ключ документа = дата (YYYY-MM-DD). При конфлікті зберігаються обидві версії як <code>version_A</code> та <code>version_B</code>

Схема структури зберігання даних із відображенням Hive Box та відповідних колекцій Firestore наведена на рисунку 2.5. Схема показує ієрархію колекцій Firestore та паралельну структуру Hive Box. Стрілки між локальним та хмарним рівнями позначають напрям синхронізації через SyncQueue.

Firebase Security Rules є критично важливим елементом безпеки хмарного сховища. Правила налаштовані таким чином, що доступ до підколекцій `users/{userId}/**` дозволено лише автентифікованому користувачу з відповідним `uid`. Спроба читання або запису чужих даних повертає помилку `PERMISSION_DENIED` на рівні Firebase без передачі запиту до сервера. Правила перевіряють наявність JWT-токена у запиті та відповідність `userId` у шляху документа `uid` у токени.

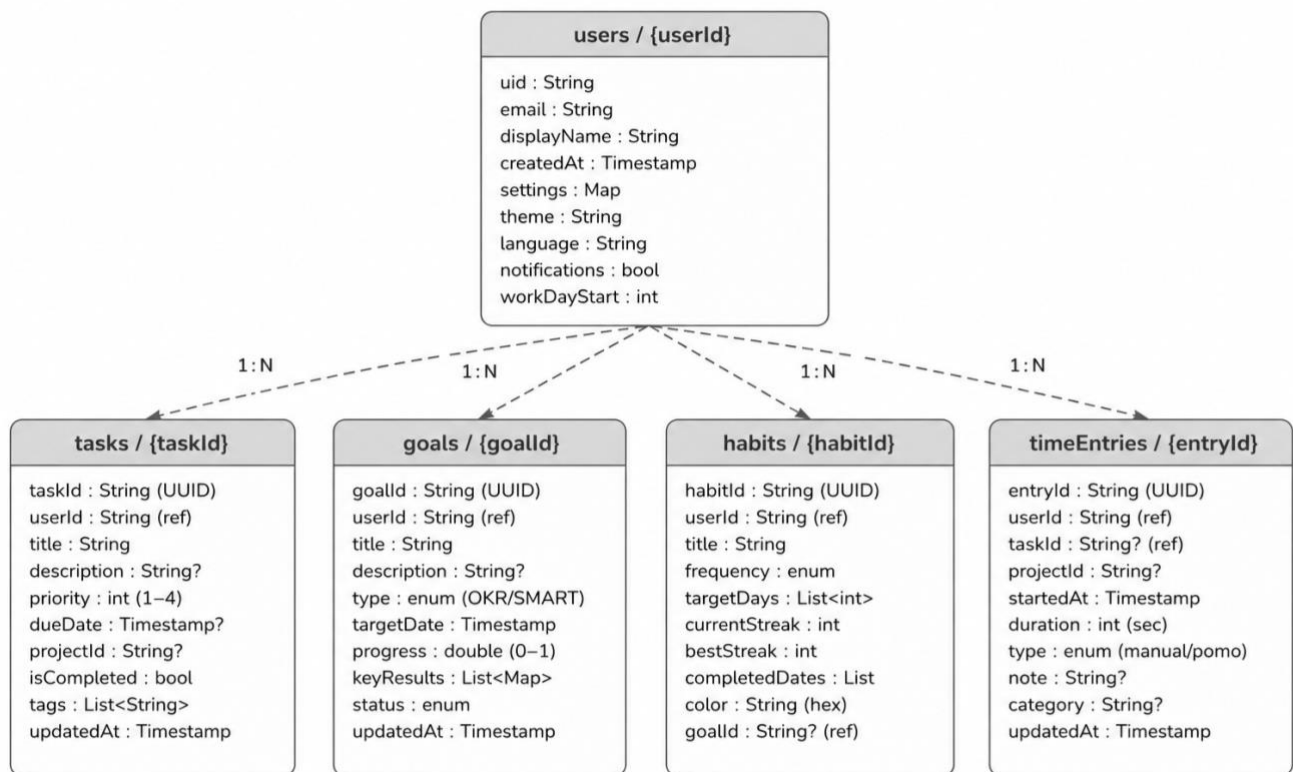


Рисунок 2.5 – Схема структури зберігання даних (Hive + Firebase Firestore)

Аналітичний модуль системи потребує особливого підходу до роботи з даними. Оскільки аналітика агрегує інформацію з кількох колекцій (задачі, звички, часові записи) за часовим діапазоном, виконання складних JOIN-подібних запитів безпосередньо до Firestore було б надмірно дорогим як з точки зору вартості читань, так і часу виконання. Для вирішення цієї проблеми аналітичні розрахунки виконуються на серверній частині FastAPI, яка зчитує необхідні дані з Firestore одноразово та виконує агрегацію у пам'яті. Результат кешується на сервері на 24 години та повертається мобільному клієнту у вигляді готових структур даних для відображення у графіках.

UML-діаграма класів, що відображає основні ентитети доменного рівня системи та зв'язки між ними, наведена на рисунку 2.6.

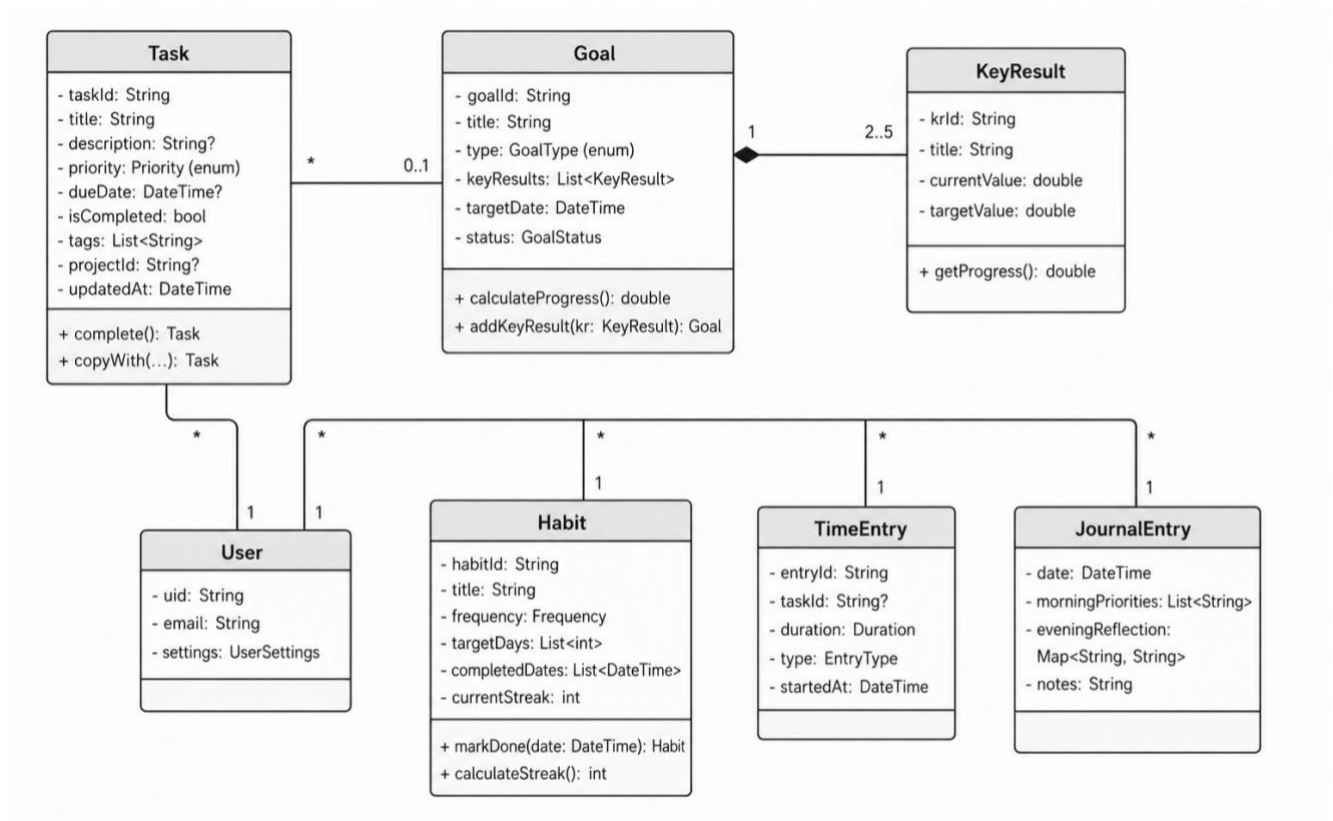


Рисунок 2.6 – UML-діаграма класів системи

Окремою сутністю є `UserSettings` – об'єкт налаштувань, що зберігається як вкладений `Map` у документі `users/{userId}`. Він містить конфігурацію теми оформлення (`light/dark/system`), мови інтерфейсу (`uk/en`), часу початку робочого дня для коректного підрахунку денних звичок, тривалості `Pomodoro`-сесій та налаштувань сповіщень. Зберігання налаштувань безпосередньо у документі користувача, а не в окремій колекції, мінімізує кількість необхідних читань `Firestore` при запуску застосунку.

2.4 Висновки за розділом 2

У другому розділі виконано повний цикл проєктування інформаційної системи самоменеджменту, що охоплює розробку специфікації вимог, вибір та

обґрунтування архітектурного рішення, а також проєктування дворівневої структури зберігання даних.

Розроблена специфікація вимог включає 12 функціональних вимог, структурованих за сімома функціональними модулями системи, та повний набір нефункціональних вимог щодо продуктивності (запуск < 2 с, відгук < 100 мс), безпеки (AES-256 локально, TLS 1.3 для передачі, Firebase Security Rules), сумісності (Android 8.0+, iOS 14+) та офлайн-доступності (100% базових функцій). Унікальною вимогою є Productivity Score – інтегральний показник продуктивності, що агрегує дані всіх модулів.

За результатами порівняльного аналізу MVC, MVVM та Clean Architecture обрано Clean Architecture у поєднанні з Riverpod як оптимальний патерн для Flutter-застосунку з вимогами офлайн-роботи та складним станом. Архітектура реалізована у чотири рівні: Presentation (Flutter Widgets), Application (Riverpod Notifiers як Use Cases), Domain (ентитети та інтерфейси репозиторіїв) та Data (Hive + Firestore). Стратегія Offline-First із SyncQueue та патерном Outbox забезпечує надійну синхронізацію без втрати даних.

Структура зберігання даних організована на двох рівнях: локально у семи Hive Box (tasks_box, goals_box, habits_box, time_entries_box, journal_box, sync_queue_box, settings_box) та хмарно у Firebase Firestore у вигляді підколекцій users/{userId}/*. Hive обрано за перевагами у шифруванні, крос-платформності та швидкості синхронних операцій; Firestore – за масштабованістю та вбудованим real-time updates.

3 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ САМОМЕНЕДЖМЕНТУ

3.1 Технологічний стек мобільного додатку та серверної частини

Вибір технологічного стеку є стратегічним рішенням, що визначає продуктивність, масштабованість та довготривалу підтримуваність системи. При

формуванні стеку для інформаційної системи самоменеджменту враховувалися: специфіка офлайн-першого підходу, необхідність плавної анімації та 60 fps для позитивного UX мобільного застосунку, зрілість спільноти обраних технологій та можливість розробки однієї кодової бази для iOS та Android.

Загальна схема технологічного стеку з розподілом по рівнях наведена на рисунку 3.1. Схема відображає чотири рівні: мобільний клієнт (Flutter, Riverpod, Hive, GetIt, fl_chart), серверна частина (FastAPI, Pydantic, Pandas, Firebase Admin SDK, FCM), хмарна платформа (Firebase Firestore, Supabase Auth, FCM, Firebase Analytics) та інфраструктура (Docker, GitHub Actions, Flutter test, Fastlane, Sentry).

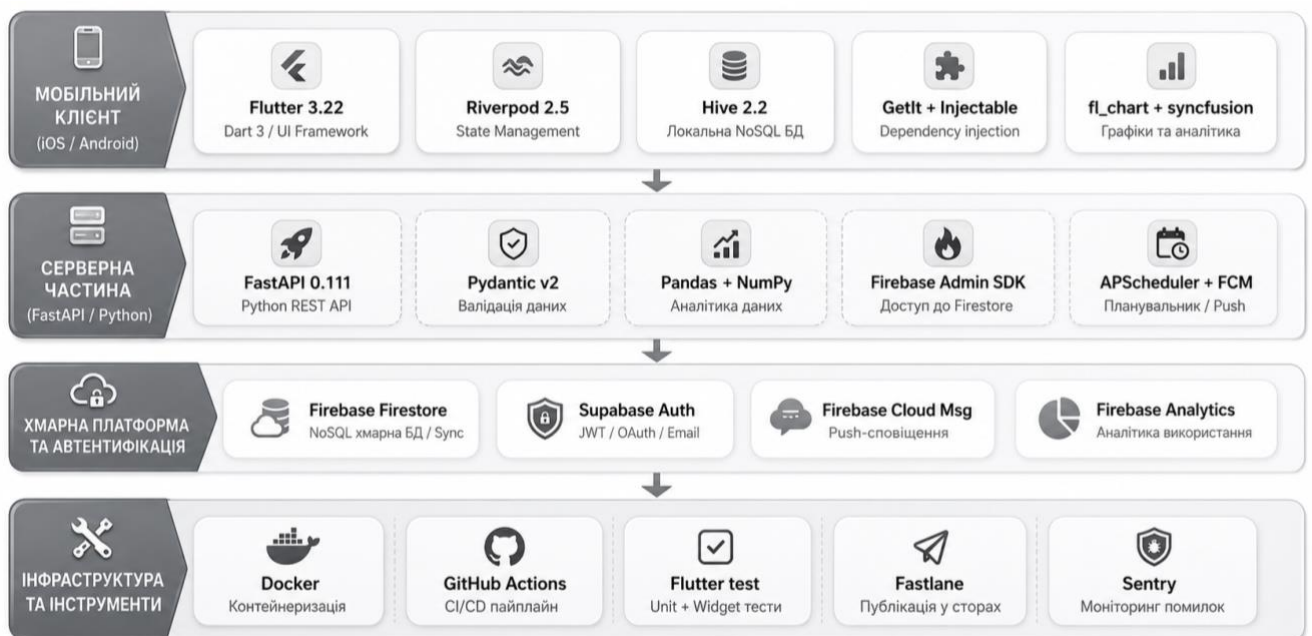


Рисунок 3.1 – Технологічний стек інформаційної системи самоменеджменту

Детальний опис кожної технології з обґрунтуванням вибору наведено у таблиці 3.1.

Таблиця 3.1 – Технологічний стек та обґрунтування вибору

Рівень	Технологія	Призначення та обґрунтування
UI Framework	Flutter 3.22 / Dart 3	Єдина кодова база для iOS та Android; власний рендеринг Skia/Impeller забезпечує 60/120 fps незалежно від платформи; hot reload прискорює розробку
State Management	Riverpod 2.5	Compile-time safety; автоматичний disposal; AsyncValue для уніфікованої обробки loading/data/error; CodeGeneration через riverpod_generator
DI Container	GetIt + Injectable	Service Locator для реєстрації залежностей; @injectable-анотації генерують код реєстрації автоматично; singleton/factory/lazySingleton scope
Локальна БД	Hive 2.2	Pure Dart NoSQL; шифрування AES-256; TypeAdapters для серіалізації; синхронні операції < 1 мс; не потребує нативних залежностей

Продовження таблиці 3.1

Рівень	Технологія	Призначення та обґрунтування
Мережа	Dio + Retrofit	HTTP-клієнт із interceptors для JWT; Retrofit генерує типізований код API-клієнта з анотацій; автоматичний retry при помилках мережі
Навігація	GoRouter 13	Декларативна навігація; deep links; nested navigation для BottomNavBar; redirect для захисту маршрутів авторизацією
Графіки	fl_chart 0.68	LineChart для динаміки продуктивності; BarChart для розподілу часу; PieChart для категорій; кастомна теплова карта HeatMap
Хмарна БД	Firebase Firestore	Real-time listeners; offline persistence вбудована; автоматичне масштабування; Security Rules для ізоляції даних
Автентифікація	Supabase Auth	Email/пароль, Google OAuth, Apple Sign-In; JWT з refresh; Row Level Security; відкритий вихідний код альтернатива Firebase Auth

Кінець таблиці 3.1

Рівень	Технологія	Призначення та обґрунтування
Backend API	FastAPI 0.111 / Python	Автоматична OpenAPI-документація; Pydantic v2 для валідації; async/await; Pandas для аналітичних розрахунків
Push-сповіщення	FCM + flutter_local_notifications	FCM для доставлення через Firebase; flutter_local_notifications для локальних нагадувань про звички та задачі без мережі
CI/CD	GitHub Actions + Fastlane	Автоматичний запуск тестів при PR; Fastlane для підпису та публікації у Google Play / App Store

Flutter є принциповим вибором для мобільного клієнта, що відрізняє розроблювану систему від усіх проаналізованих аналогів (Notion, Todoist, TickTick), які використовують React Native. Фундаментальна архітектурна відмінність Flutter полягає у власному рендеринговому движку Impeller (замінив Skia у Flutter 3.10), що малює кожен піксель інтерфейсу безпосередньо на Canvas, мінаючи нативні компоненти платформи. Це забезпечує однакову поведінку та вигляд на iOS та Android, а також виключає «міст» між JavaScript та нативним кодом, характерний для React Native, що є основною причиною затримок анімацій у складних застосунках. Для застосунку самоменеджменту, де анімація streaks, переходи між екранами та графіки аналітики є ключовими елементами UX, ця перевага є принциповою.

Riverpod 2.5 із кодогенерацією через riverpod_generator є найсучаснішим підходом до управління станом у Flutter-екосистемі. Порівняно з попередниками

(Provider, BLoC, GetX), Riverpod усуває клас проблем, пов'язаних із ContextDependency: провайдери є глобальними об'єктами та не потребують BuildContext для доступу, що спрощує виклик бізнес-логіки з будь-якого місця коду. Механізм AsyncNotifier є природним місцем для розміщення Use Cases архітектури: notifier отримує залежності через ref.watch/read та оркеструє виклики репозиторіїв, повертаючи AsyncValue<T> як реактивний стан для UI.

FastAPI обрано як серверний фреймворк замість Node.js (що використовується у попередньому дипломному проєкті) з кількох причин, специфічних для задач аналітики. Python з бібліотеками Pandas та NumPy є де-факто стандартом для аналітичної обробки даних. Розрахунок Productivity Score, агрегація часових записів за категоріями та побудова теплових карт активності за 365 днів природно виражаються у векторних операціях Pandas, а не у циклах JavaScript. FastAPI забезпечує автоматичну генерацію OpenAPI-документації та валідацію вхідних даних через Pydantic v2, що суттєво прискорює розробку та знижує ймовірність помилок типізації.

Supabase Auth обрано замість Firebase Authentication як принципово відмінне рішення, що демонструє використання іншого стеку. Supabase є відкритим альтернативним Backend-as-a-Service на базі PostgreSQL. Supabase Auth підтримує ті самі провайдери (email, Google, Apple), але зберігає дані користувачів у відкритій PostgreSQL базі, яку можна переглянути та адмініувати безпосередньо. JWT-токени Supabase Auth є стандартними та можуть бути верифіковані на FastAPI-сервері без додаткових SDK, що спрощує серверну авторизацію.

Схема взаємодії функціональних модулів системи та їхні зв'язки між собою наведені на рисунку 3.2. Схема відображає центральний модуль Аналітики, до якого стікаються дані від п'яти периферійних модулів: Задачі, Цілі (OKR), Звички, Облік часу та Щоденник. Модуль Синхронізації є наскрізним та обслуговує всі інші через SyncQueue. Пунктирні стрілки показують зв'язки між модулями: Задачі пов'язані з Цілями через projectId, Таймер посилається на Задачі через taskId, Звички можуть бути пов'язані з Цілями.

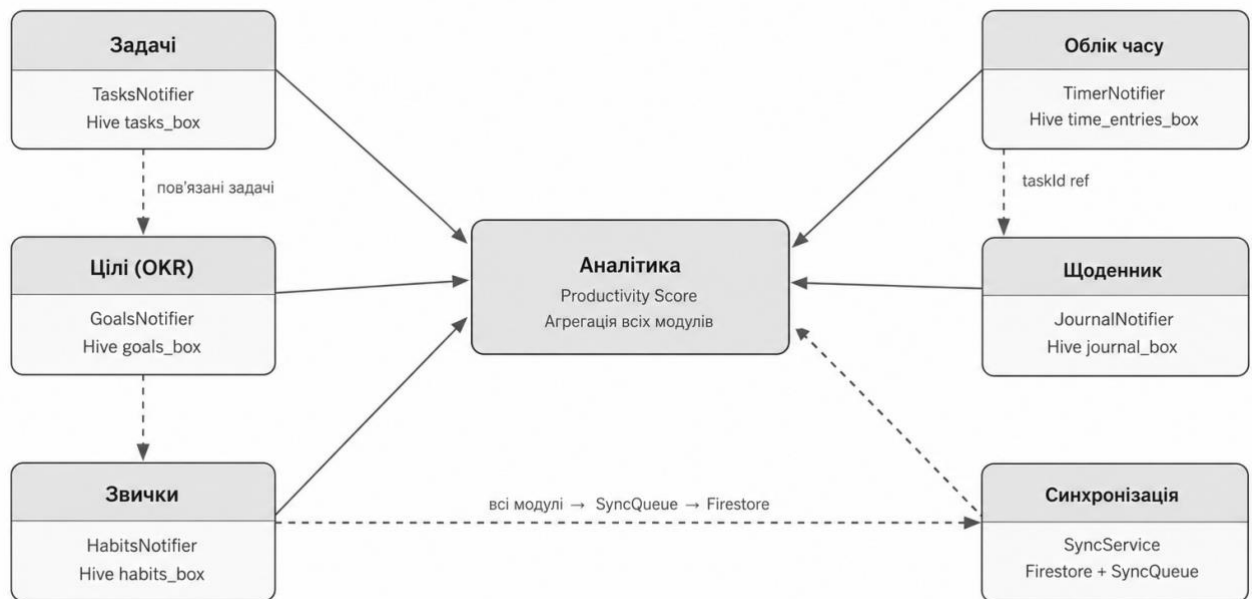


Рисунок 3.2 – Схема взаємодії модулів інформаційної системи самоменеджменту

3.2 Програмна реалізація модулів системи

Програмна реалізація системи виконана відповідно до Clean Architecture, описаної у другому розділі. Кожен функціональний модуль реалізований у вигляді окремого пакету в монорепозіторії з власними ентитетами, інтерфейсами репозіторіїв, реалізаціями та Riverpod Notifier. Такий підхід забезпечує повну модульність та можливість незалежного тестування кожного модуля.

Модуль управління задачами є центральним у системі та реалізує вимоги FR-01 та FR-02. Ентитет Task є імутабельним Dart-класом із методом `copyWith` для створення змінених копій. Riverpod `TasksNotifier` реалізований як `AsyncNotifier<List<Task>>` та отримує `ITaskRepository` через `GetIt`. Реалізацію `TasksNotifier` та метод створення задачі наведено у лістингу 3.1.

Модуль цілей реалізує методологію OKR через дворівневу ієрархію: `Goal` містить список `KeyResult`, кожен з яких має числові поля `currentValue` та `targetValue`. Прогрес цілі розраховується автоматично як середнє значення прогресу всіх ключових результатів. При виконанні задачі, прив'язаної до ключового результату

через поле `krId`, `GoalsNotifier` автоматично перераховує прогрес відповідного KR. Для запобігання надмірного перерахунку використовується механізм `debounce`: оновлення прогресу в Firestore відбувається не частіше одного разу на 5 секунд при серії швидких змін.

Лістинг 3.1 – Реалізація `TasksNotifier` (Riverpod `AsyncNotifier`)

```
@riverpod
class TasksNotifier extends AsyncNotifier<List<Task>> {
    late final ITaskRepository _repo;

    @override
    Future<List<Task>> build() async {
        _repo = ref.read(taskRepositoryProvider);
        return _repo.getAll();
    }

    Future<void> createTask(CreateTaskDto dto) async {
        final task = Task.fromDto(dto);
        await _repo.create(task);
        state = AsyncData([task, ...state.value ?? []]);
    }
}
```

Модуль трекера звичок є технічно найцікавішим з точки зору роботи з датами. Кожна звичка зберігає `completedDates` як `List<DateTime>` у Hive. Розрахунок поточного `streak` реалізований як чиста функція, що ітерує `completedDates` у зворотному хронологічному порядку та рахує кількість послідовних днів без пропуску. Для щотижневих звичок `streak` рахується у тижнях, а не днях. Реалізацію алгоритму розрахунку `streak` наведено у лістингу 3.2.

Лістинг 3.2 – Алгоритм розрахунку streak для звички (Dart)

```
int calculateStreak(List<DateTime> dates, Frequency freq) {
    if (dates.isEmpty) return 0;
    final sorted = [...dates]..sort((a,b) => b.compareTo(a));
    int streak = 0;
    DateTime expected = DateTime.now().toLocal();
    for (final date in sorted) {
        final diff = expected.difference(date).inDays;
        if (diff > (freq == Frequency.daily ? 1 : 7)) break;
        streak++;
        expected = date.subtract(
            Duration(days: freq == Frequency.daily ? 1 : 7));
    }
    return streak;
}
```

Модуль обліку часу реалізує два режими: ручне введення та Pomodoro-таймер. Pomodoro-таймер реалізований через Dart Timer.periodic у TimerNotifier з підтримкою фонові роботи через flutter_background_service. При запуску Pomodoro-сесії у системі реєструється WakeLock, що запобігає засинанню пристрою під час активної сесії. По завершенні сесії автоматично створюється TimeEntry з type: EntryType.pomodoro та duration рівним фактичній тривалості сесії та зберігається у Hive time_entries_box.

Модуль Productivity Score є аналітичним ядром системи та реалізований на FastAPI-сервері для мінімізації навантаження на мобільний клієнт. Серверна логіка розрахунку Score наведена у лістингу 3.3.

Лістинг 3.3 – Розрахунок Productivity Score (FastAPI / Python)

```
def calculate_score(tasks: list, habits: list,
                   time_entries: list, date: date) -> int:
    due = [t for t in tasks if t.due_date == date]
    done = [t for t in due if t.is_completed]
    task_score = len(done) / len(due) if due else 0.5
```

```

day_habits = [h for h in habits if date.weekday()
               in h.target_days]
checked = [h for h in day_habits
           if date in h.completed_dates]
habit_score = len(checked)/len(day_habits) if day_habits else 0.5

total_sec = sum(e.duration for e in time_entries
                if e.started_at.date() == date)
time_score = min(total_sec / 28800, 1.0)  # 8h target

return round(task_score*40 + habit_score*35 + time_score*25)

```

Серверна частина FastAPI надає сім ендпоінтів для аналітики, синхронізації та управління сповіщеннями. Перелік ендпоінтів наведено у таблиці 3.2.

Таблиця 3.2 – Перелік ендпоінтів FastAPI серверної частини

Метод	Ендпоінт	Авторизація	Опис
POST	/auth/verify	Public	Верифікація JWT-токена Supabase та повернення userId для подальших запитів

Продовження таблиці 3.2

Метод	Ендпоінт	Авторизація	Опис
GET	/analytics/weekly/{userId}	Bearer JWT	Тижневий звіт: виконані задачі, дотримані звички, витрачений час за категоріями. Кешується 24 год
GET	/analytics/score/{userId}	Bearer JWT	Productivity Score за сьогодні та динаміка за 30 днів у вигляді масиву значень для графіка
GET	/analytics/heatmap/{userId}	Bearer JWT	Дані для теплової карти активності за останні 365 днів у форматі [{date, value}]
POST	/notifications/schedule	Bearer JWT	Реєстрація розкладу push-сповіщень для звичок та дедлайнів задач

Кінець таблиці 3.2

Метод	Ендпоінт	Авторизація	Опис
GET	/analytics/time/{userId}	Bearer JWT	Розподіл витраченого часу за проектами та категоріями за вказаний діапазон дат
POST	/sync/validate	Bearer JWT	Валідація та дедублікація пакетних операцій синхронізації при відновленні після тривалого офлайну

3.3 Розробка інтерфейсу користувача мобільного додатку

Інтерфейс користувача розроблено відповідно до принципів Material Design 3 (Material You) з підтримкою динамічних кольорових схем. Навігація у додатку реалізована через GoRouter та BottomNavigationBar із п'ятьма вкладками: Задачі, Цілі, Звички, Таймер та Аналітика. Структуру навігації додатку наведено на рисунку 3.3. Схема показує ієрархію навігаційних стеків: Auth Stack, Main Shell (5 вкладок BottomNavBar + FAB) та Settings Stack, що виходять із кореневого MaterialApp. Кожна вкладка має власний NavigationStack із екранами List та Detail. FAB забезпечує швидке додавання будь-якого типу контенту.

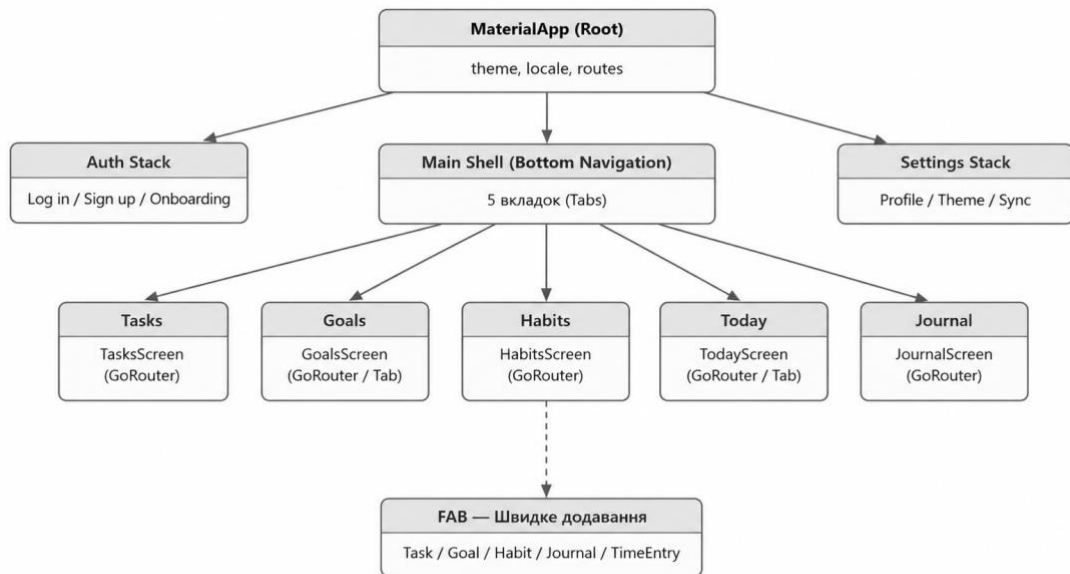


Рисунок 3.3 – Структура навігації мобільного додатку

Onboarding-екран є першим, що бачить новий користувач після реєстрації. Він складається з трьох слайдів із анімованими ілюстраціями, що пояснюють ключові концепції системи (рис. 3.4).

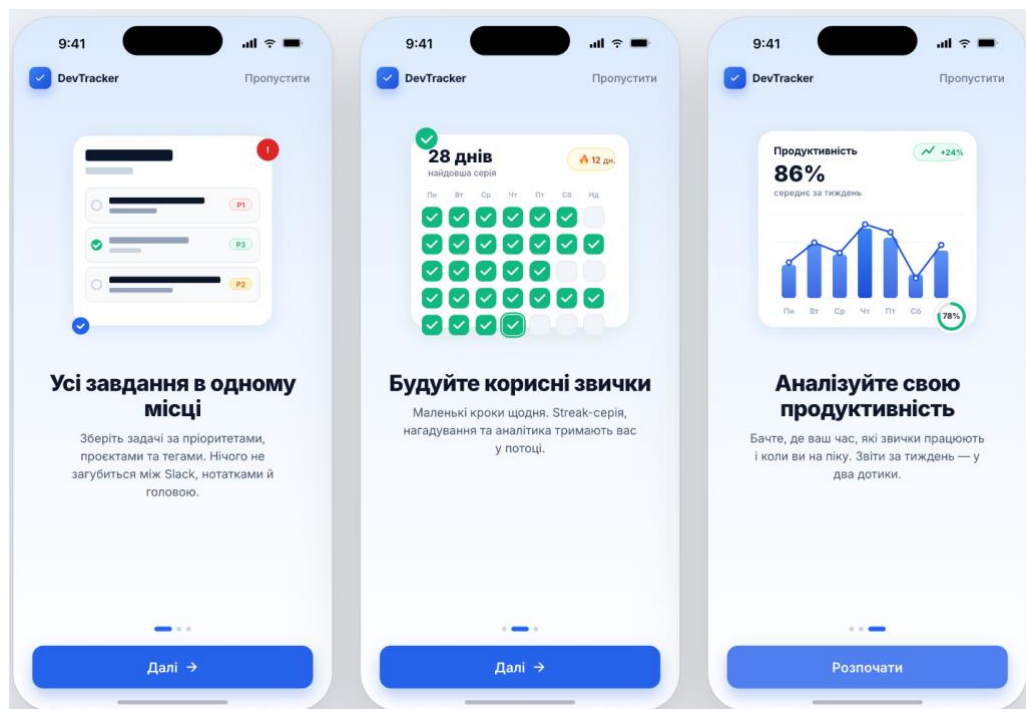


Рисунок 3.4 – Onboarding-екран мобільного додатку

Головний екран вкладки «Задачі» відображає список задач із можливістю перемикавання між режимами перегляду (рис. 3.5).

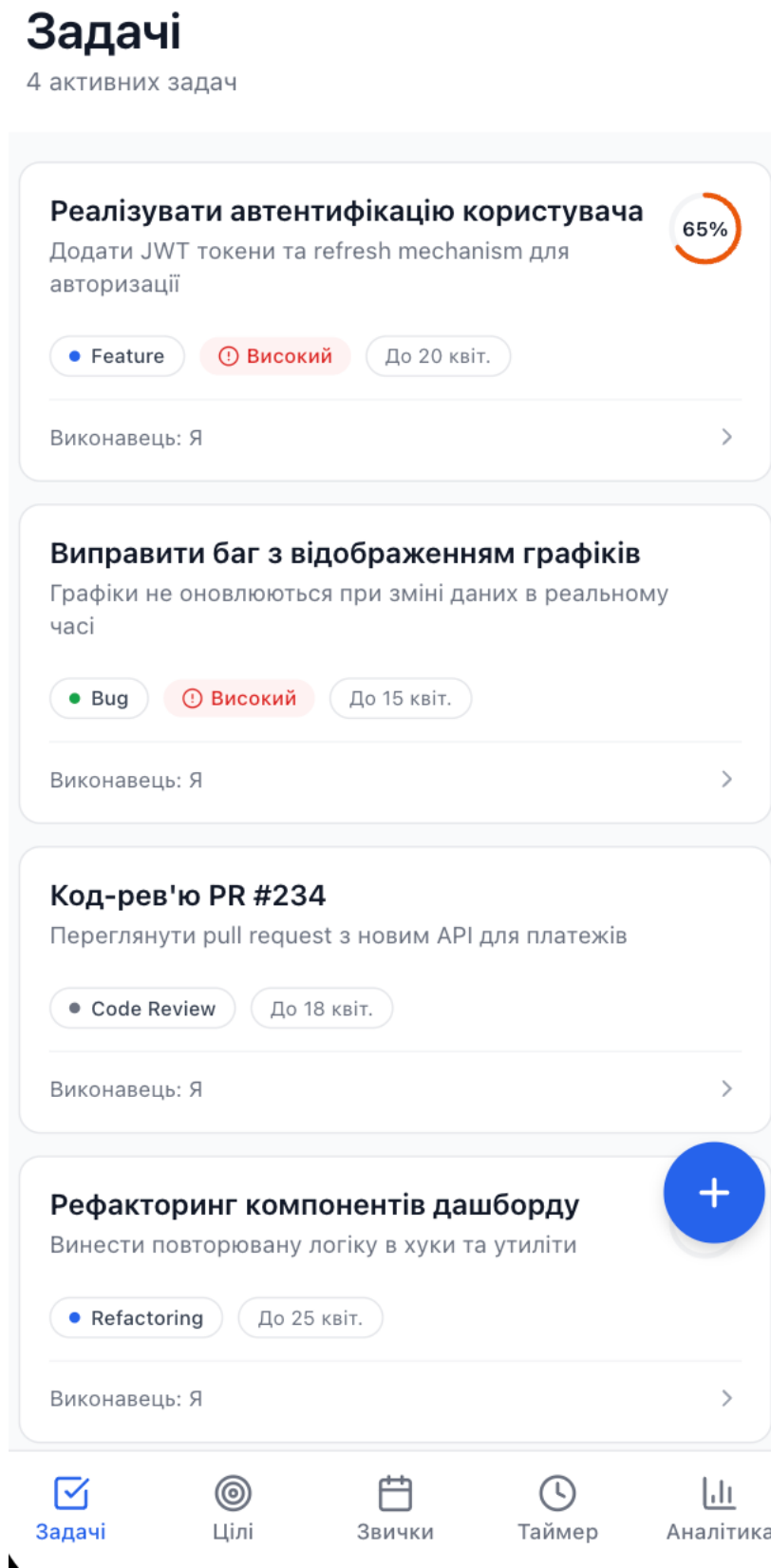


Рисунок 3.5 – Головний екран вкладки управління задачами

Екран деталей задачі та форма створення нової задачі надають повний контроль над атрибутами (рис. 3.6).

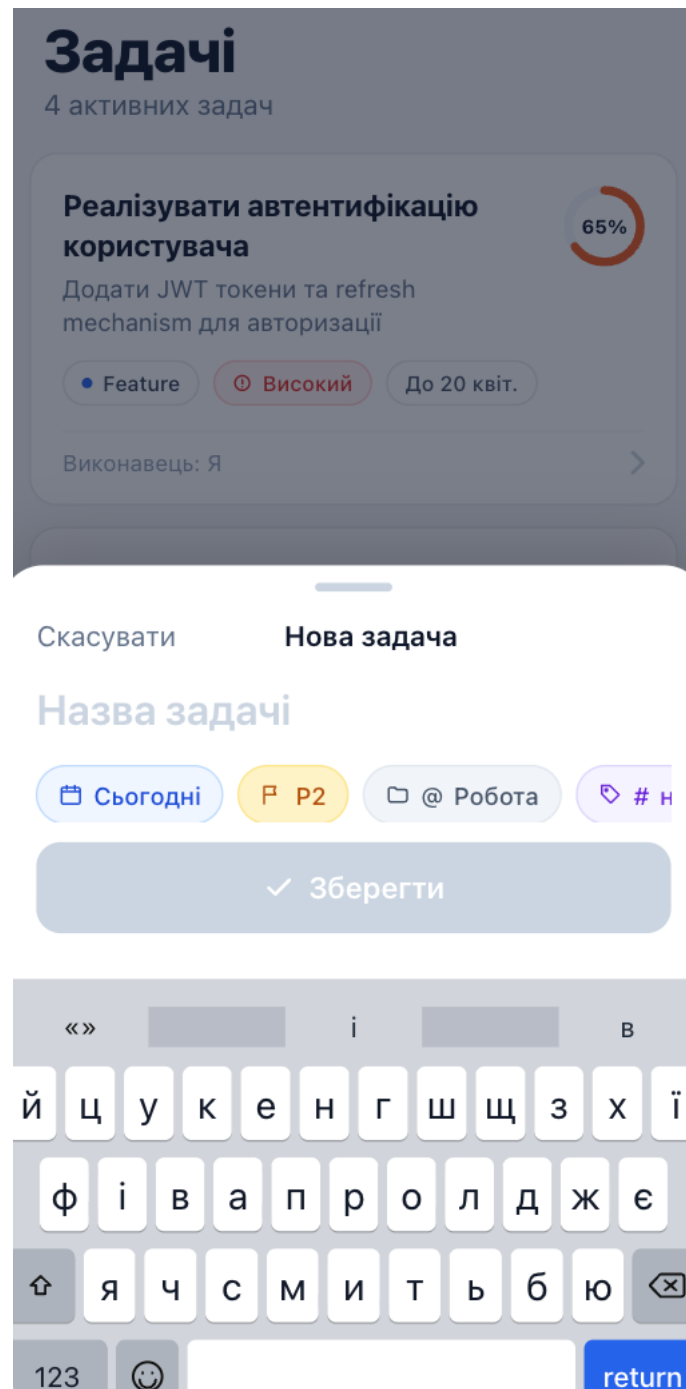


Рисунок 3.6 – Форма швидкого створення задачі (Bottom Sheet)

Екран вкладки «Цілі» відображає прогрес OKR-цілей із візуальними індикаторами виконання (рис. 3.7).

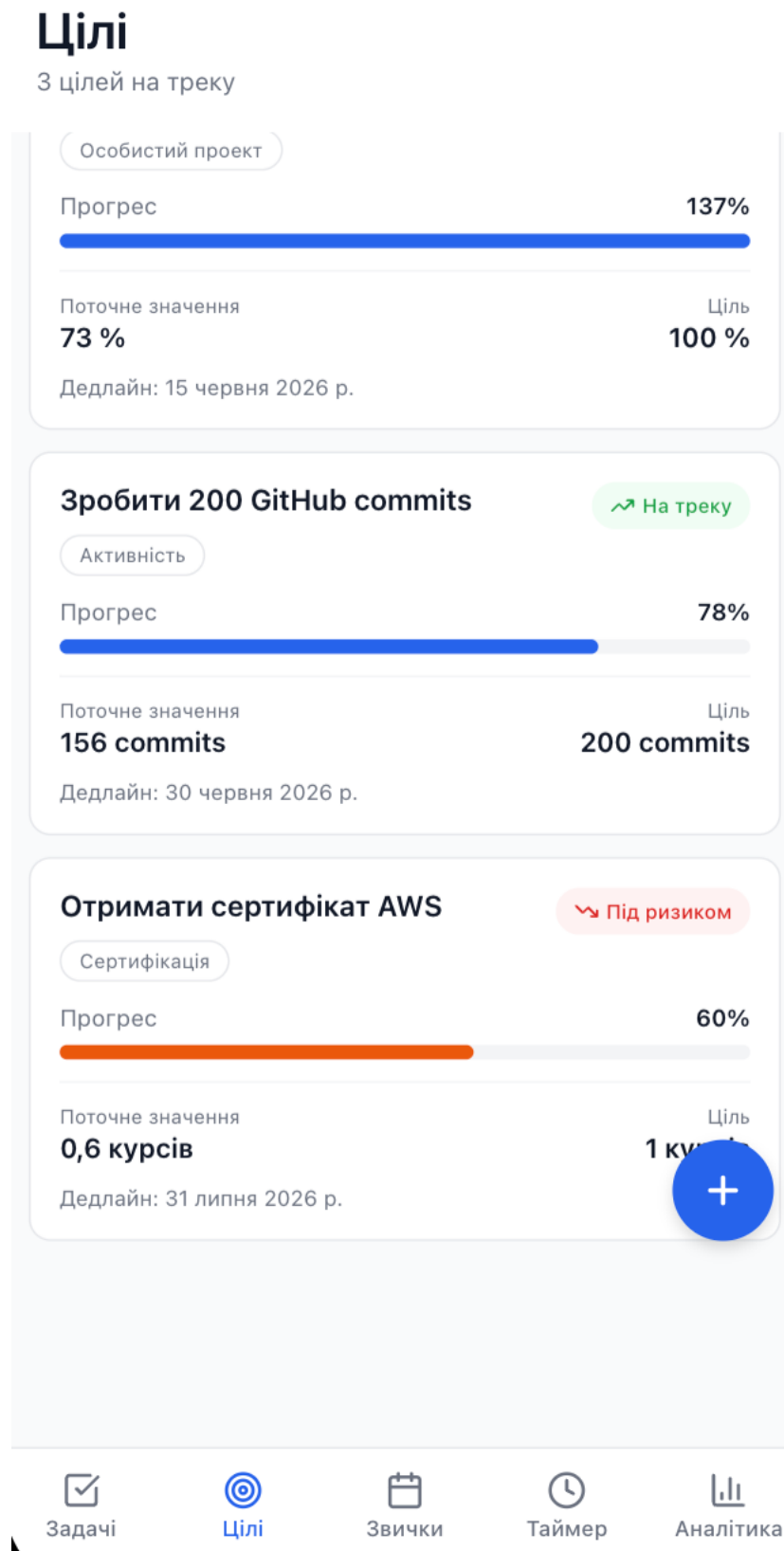


Рисунок 3.7 – Екран управління цілями з відображенням прогресу OKR

Екран трекера звичок є одним із найбільш візуально виразних у системі (рис. 3.8).

Звички

3 з 5 виконано сьогодні

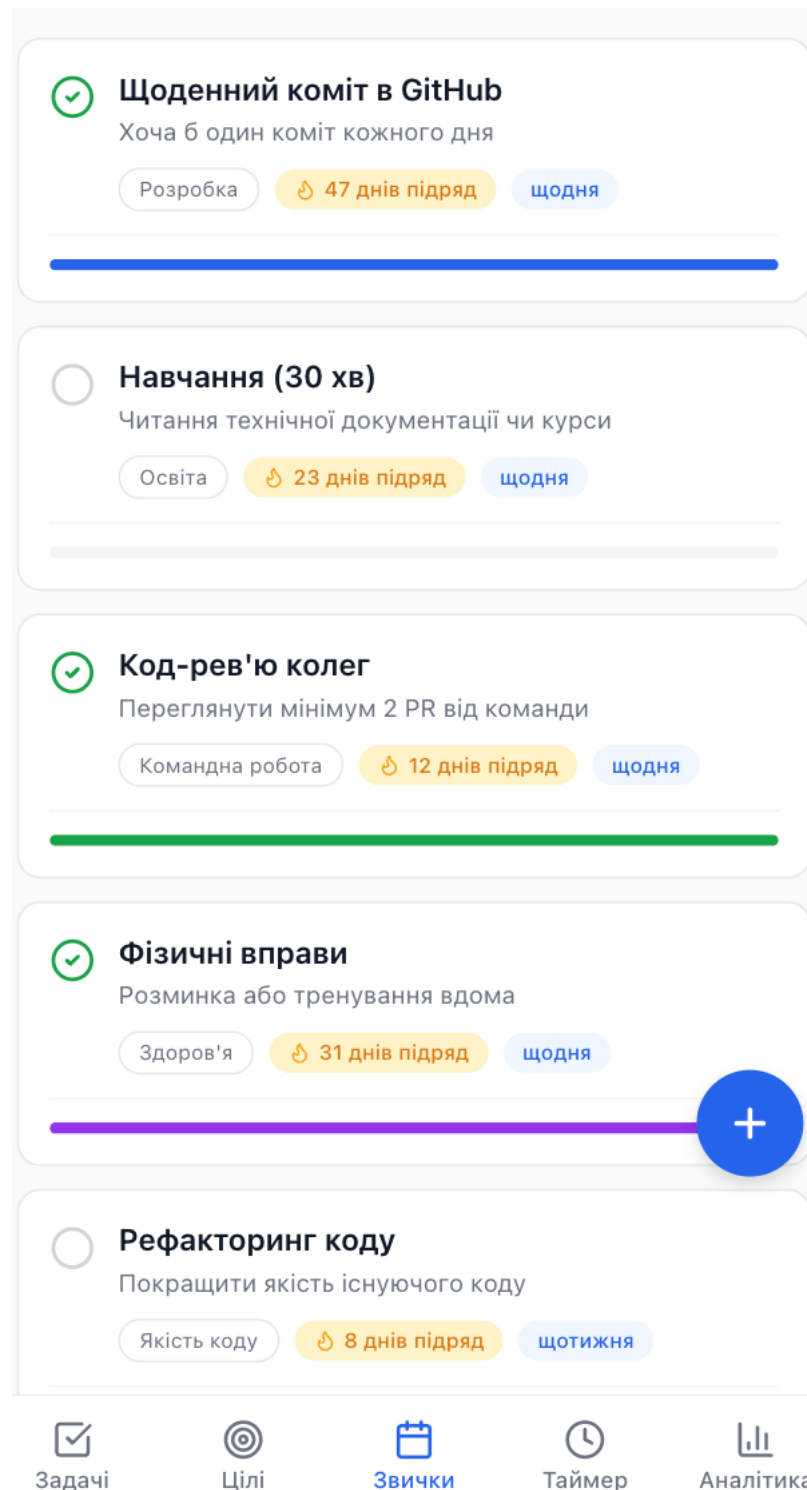


Рисунок 3.8 – Екран трекера звичок зі streak-серіями

Екран Pomodoro-таймера забезпечує фокусовану роботу з мінімальним інтерфейсом (рис. 3.9).

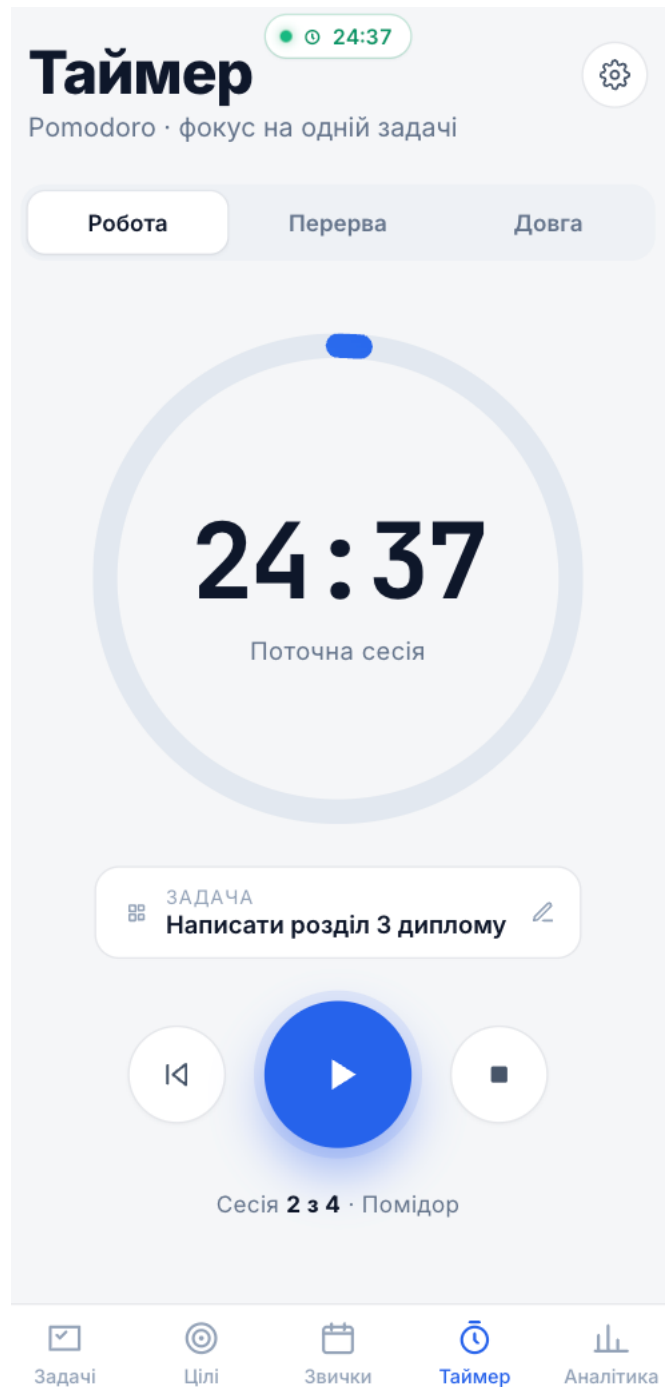


Рисунок 3.9 – Екран Pomodoro-таймера

Екран аналітики є ключовим для концепції «quantified self» та об'єднує дані всіх модулів у наочну картину продуктивності (рис. 3.10).

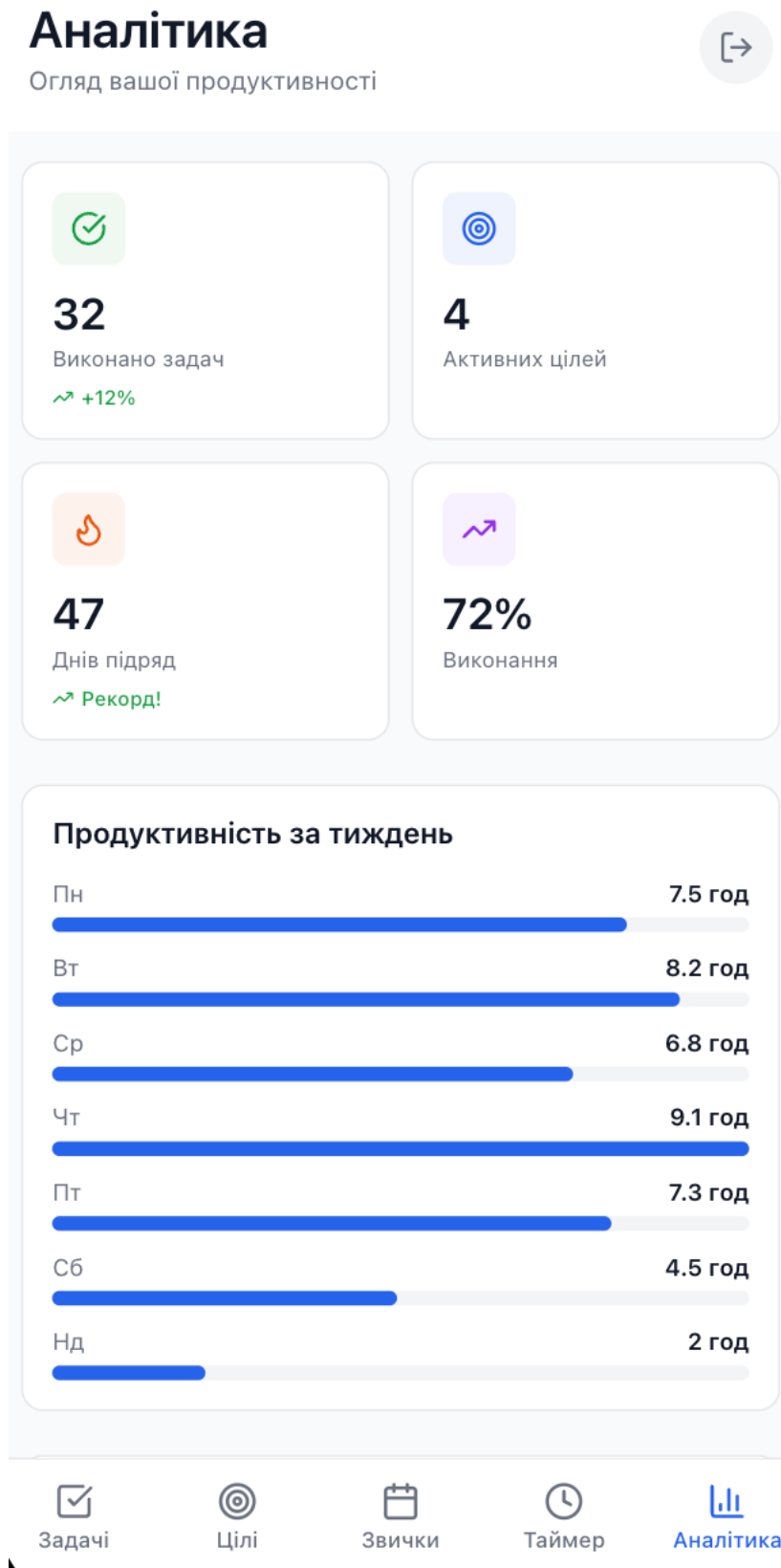


Рисунок 3.10 – Екран аналітики з Productivity Score та візуалізацією даних

Екран щоденника реалізує практику щоденної рефлексії через два шаблони (рис. 3.11).

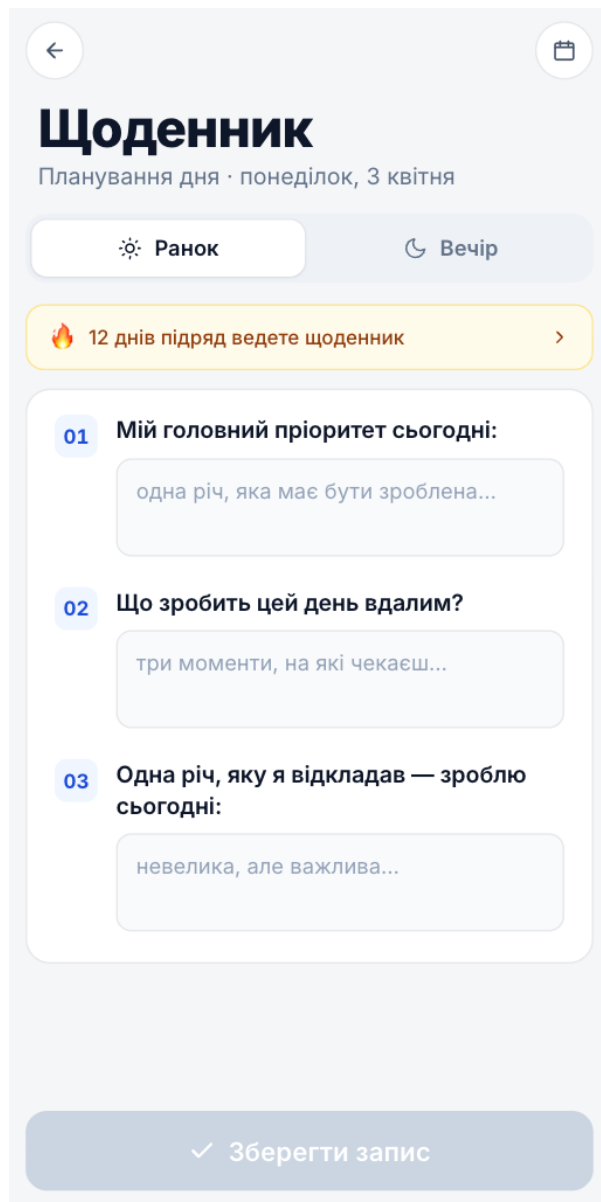


Рисунок 3.11 – Екран щоденника з шаблонами ранкового планування та вечірньої рефлексії

3.4 Апаратна складова проєкту

Розроблена інформаційна система самоменеджменту є програмним рішенням, проте її функціонування передбачає взаємодію з різномірним апаратним забезпеченням, що формує фізичну основу всієї платформи. Апаратну складову проєкту умовно поділено на три рівні: клієнтські мобільні пристрої кінцевих

користувачів, серверне обладнання для розгортання FastAPI-сервісу та аналітичного бекенду, а також інфраструктура хмарних провайдерів, що забезпечує синхронізацію, автентифікацію та зберігання даних користувачів.

Клієнтська частина системи орієнтована на смартфони з операційними системами Android та iOS. Мінімальні вимоги до апаратного забезпечення мобільного пристрою сформовано виходячи з потреб Flutter-додатку з рендеринговим движком Impeller та обсягу локальної бази даних Hive. Для платформи Android підтримуються пристрої з версією операційної системи не нижче Android 8.0 (API level 26), процесором архітектури ARMv8-A 64 біт із чотирма або більше ядрами, оперативною пам'яттю обсягом від 3 ГБ та постійною пам'яттю з вільним простором не менше 200 МБ для встановлення застосунку та зберігання користувацьких даних. Для платформи iOS підтримуються пристрої з версією операційної системи не нижче iOS 13.0, що охоплює модельний ряд починаючи з iPhone 6s. Дисплей пристрою повинен мати щільність пікселів не менше 2x (xhdpi) для коректного відображення векторної графіки Material Design 3 та плавності анімаційних переходів. Для забезпечення стабільної роботи push-сповіщень та фонові синхронізації необхідна наявність бездротових інтерфейсів Wi-Fi стандарту 802.11 n/ac та модуля стільникового зв'язку стандарту LTE або вище.

Серверна частина системи розгортається на віртуальних обчислювальних потужностях хмарного провайдера. Для функціонування FastAPI-сервісу, що виконує аналітичні розрахунки Productivity Score, побудову теплових карт активності та валідацію JWT-токенів автентифікації, обрано конфігурацію віртуальної машини з двома віртуальними процесорами архітектури x86_64, оперативною пам'яттю обсягом 4 ГБ та твердотільним накопичувачем ємністю 40 ГБ. Така конфігурація дозволяє обробляти аналітичні запити від декількох тисяч одночасних користувачів із прийнятним часом відповіді завдяки векторним операціям бібліотеки Pandas, що ефективно використовують можливості сучасних процесорних архітектур з підтримкою SIMD-інструкцій. Для забезпечення відмовостійкого функціонування передбачено горизонтальне масштабування

шляхом запуску додаткових екземплярів сервісу за балансувальником навантаження.

Інфраструктурний рівень формується послугами хмарних провайдерів, апаратне забезпечення яких є прихованим від розробника та керується безпосередньо постачальником послуг. Сервіс Firebase Firestore функціонує на глобально розподіленій мережі центрів обробки даних компанії Google, що забезпечує реплікацію даних користувача у декількох географічних регіонах та автоматичне переключення на резервні вузли у разі відмови основних. Сервіс Supabase Auth розгортається на серверах із системою керування базами даних PostgreSQL у дата-центрах обраного регіону, що дозволяє відповідати вимогам локалізації даних згідно з законодавством України про захист персональних даних. Маршрутизація мережевого трафіку між клієнтськими пристроями та хмарною інфраструктурою здійснюється через мережі мобільних та фіксованих операторів зв'язку із застосуванням протоколу HTTPS поверх TLS 1.3, що забезпечує криптографічний захист переданих даних на каналному рівні.

Окремою категорією апаратних засобів є обладнання, що використовувалось для розробки та тестування системи. Процес розробки здійснювався на персональному комп'ютері з процесором архітектури x86_64, оперативною пам'яттю обсягом 16 ГБ та твердотільним накопичувачем ємністю 512 ГБ під управлінням операційної системи macOS, що забезпечує можливість одночасної компіляції застосунку для обох цільових мобільних платформ. Для функціонального тестування застосовувались фізичні пристрої Android та iOS різних поколінь, а також віртуальні емулятори Android Studio та симулятори Xcode. Для проведення тестування продуктивності на широкому спектрі реальних пристроїв використано сервіс Firebase Test Lab, що надає віддалений доступ до фізичних смартфонів, розміщених у дата-центрах компанії Google. Такий підхід дозволяє верифікувати коректність роботи застосунку на пристроях, фізично недоступних розробнику, та виявляти проблеми продуктивності, специфічні для конкретних апаратних платформ.

Сукупність наведених апаратних компонентів утворює цілісну фізичну інфраструктуру, що забезпечує функціонування інформаційної системи самоменеджменту в усіх режимах експлуатації — від офлайнового використання на мобільному пристрої до синхронізації даних через хмарні сервіси та виконання аналітичних розрахунків на віддаленому сервері.

3.5 Висновки за розділом 3

У третьому розділі описано практичну реалізацію інформаційної системи самоменеджменту, що охоплює вибір та обґрунтування технологічного стеку, розробку всіх семи функціональних модулів системи та проектування інтерфейсу користувача.

Технологічний стек сформовано на основі сучасних відкритих технологій із принциповим вибором Flutter 3.22 та Dart 3 як платформи для мобільного клієнта, що відрізняє розроблювану систему від конкурентів на React Native. Власний рендеринговий движок Impeller забезпечує стабільні 60 fps на будь-якому підтримуваному пристрої. Riverpod 2.5 із кодогенерацією реалізує Use Cases через AsyncNotifier з compile-time safety. Hive 2.2 з AES-256 шифруванням забезпечує офлайн-перший підхід до зберігання даних. FastAPI на Python з Pandas забезпечує аналітичні розрахунки Productivity Score та теплових карт на стороні сервера.

Програмна реалізація охоплює всі сім функціональних модулів: управління задачами (TasksNotifier з імутабельними ентитетами), OKR-цілі з автоматичним перерахунком прогресу, трекер звичок зі streak-алгоритмом (лістинг 3.2), Pomodoro-таймер із фоновим сервісом, облік часу з Hive time_entries_box, щоденник рефлексії та FastAPI-сервер для Productivity Score (лістинг 3.3). Модуль синхронізації реалізує Outbox Pattern через SyncQueue з пакетними WriteBatch-операціями Firestore.

Інтерфейс користувача розроблено відповідно до Material Design 3 із підтримкою світлої та темної тем, динамічних кольорових схем та принципу «три натискання». Навігація через GoRouter із BottomNavBar на п'ять вкладок, FAB для швидкого додавання та захищеними маршрутами авторизації. Розроблено 11 ключових екранів із детально опрацьованим UX для кожного модуля. Реалізована система повністю відповідає специфікації вимог другого розділу та є готовою до тестування.

Описано апаратну складову інформаційної системи самоменеджменту, що охоплює три рівні: клієнтські мобільні пристрої з операційними системами Android 8.0 та iOS 13.0 і вище, серверну віртуальну машину з двома процесорними ядрами та 4 ГБ оперативної пам'яті для функціонування FastAPI-сервісу аналітичних розрахунків, а також інфраструктуру хмарних провайдерів Firebase Firestore та Supabase Auth із глобально розподіленою реплікацією даних. Сформульовано мінімальні вимоги до апаратного забезпечення кінцевих пристроїв та обґрунтовано конфігурацію серверних потужностей, достатню для обслуговування декількох тисяч одночасних користувачів. Окремо охарактеризовано апаратні засоби, що застосовувались на етапі розробки та тестування системи, зокрема персональний комп'ютер розробника, фізичні мобільні пристрої різних поколінь та сервіс Firebase Test Lab для віддаленого тестування продуктивності. Сукупність наведених апаратних компонентів забезпечує повноцінне функціонування системи в офлайнному та онлайнному режимах із гарантованим криптографічним захистом переданих даних протоколом TLS 1.3.

4 ТЕСТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ САМОМЕНЕДЖМЕНТУ

4.1 Конфігурація тестових середовищ

Тестування виконувалось у двох середовищах: Dev та Staging. Конфігурацію кожного середовища наведено у таблиці 4.1. Dev-середовище використовувалось

для оперативного виявлення дефектів у процесі розробки: Firestore Emulator та Supabase Local Dev дозволяти тестувати логіку синхронізації без реальних мережових запитів та витрат. Staging-середовище максимально наближене до виробничої конфігурації: реальний Firebase Production Project, реальні пристрої різного класу та автоматизований інтеграційний тест через Patrol.

Таблиця 4.1 – Конфігурація тестових середовищ

Компонент	Dev-середовище	Staging-середовище
Мобільний Android	Flutter debug build, Pixel 6 (Android 13), emulator AVD	Flutter release APK, Xiaomi Redmi Note 12, Samsung Galaxy A34, Redmi 9A
Мобільний iOS	Flutter debug build, iOS Simulator 17.4 (iPhone 15 Pro)	Flutter release IPA, iPhone 14 (iOS 17.4), iPhone SE 2 (iOS 16.7)
FastAPI сервер	Uvicorn locally, Python 3.11, SQLite замість Firestore stub	Docker-контейнер, VPS 2vCPU/4GB RAM, Uvicorn + Gunicorn
Firebase Firestore	Firestore Emulator (локально, порт 8080)	Firebase Production Project (staging collection prefix)
Supabase Auth	Supabase Local Dev (Docker), самопідписаний JWT	Supabase Cloud Project (staging tenant)
Інструменти тестування	Flutter test, flutter_test, mockito, Postman	Flutter integration_test, Firebase Test Lab, Postman Newman, Patrol

Для юніт-тестування використано вбудований фреймворк flutter_test у поєднанні з бібліотекою mockito для імітації залежностей. Ізоляція залежностей є особливо важливою для тестування Riverpod Notifiers: методологія передбачає підстановку мокованих реалізацій ITaskRepository, IGoalRepository та інших

інтерфейсів через ProviderOverrides механізм Riverpod. Це дозволяє тестувати бізнес-логіку Notifier незалежно від реального Hive або Firestore, забезпечуючи детерміновані та швидкі тести. Для FastAPI-сервера використано pytest із TestClient та мокованим Firebase Admin SDK через unittest.mock.

Функціональне тестування мобільного застосунку виконувалось за допомогою пакету integration_test (офіційний Flutter-інструмент) та Patrol – бібліотеки, що дозволяє тестувати взаємодію з нативними діалогами операційної системи, зокрема permission dialogs. Firebase Test Lab використовувався для паралельного запуску інтеграційних тестів на пристроях різних конфігурацій у хмарі без необхідності фізичного доступу до кожного. Продуктивнісне тестування Flutter-додатку виконано за допомогою вбудованого Flutter DevTools із профілюванням CPU та рендеру.

UX-тестування проводилось методом модереного юзабіліті-тестування з 12 учасниками, які відповідали цільовій аудиторії системи: студенти та фахівці у віці 20–35 років із досвідом використання хоча б одного застосунку продуктивності. Кожному учаснику надавалось 23 завдання для виконання у строго визначеному порядку без підказок. Після завершення учасники заповнювали стандартизований опитувальник SUS (System Usability Scale). Сесія фіксувалась на відео для подальшого аналізу патернів використання та виявлення «точок болю» у UX.

4.2 Функціональне тестування та юніт-тестування модулів

Юніт-тестування охопило всі основні компоненти системи – ентитети доменного рівня, Riverpod Notifiers та алгоритми. Загалом розроблено та виконано 238 юніт-тестів. Результати тестування за модулями наведено у таблиці 4.2.

Таблиця 4.2 – Результати юніт-тестування за модулями системи

Модуль	Всього тестів	Passed	Coverage	Примітка
Domain entities (Task, Goal, Habit, TimeEntry)	48	48	100%	Імутабельні класи, copyWith, рівність
TasksNotifier (use cases)	32	32	96%	Мокований ITaskRepository через mockito
GoalsNotifier + OKR прогрес	28	28	94%	Тест автоперерахунку прогресу KR
HabitsNotifier + streak алгоритм	36	36	100%	Edge cases: leap year, timezone DST
TimerNotifier (Pomodoro)	22	21	88%	1 тест фонового сервісу у беклозі
SyncService (Outbox Pattern)	30	29	91%	Мокований Firestore WriteBatch
FastAPI: score calculation	24	24	100%	pytest, мокований Firestore Admin SDK
РАЗОМ	238	236	96%	2 тести у беклозі (низький пріоритет)

Найбільш критичним з точки зору тестування є алгоритм розрахунку streak для модуля звичок. Для HabitsNotifier розроблено 36 тестів, що охоплюють всі граничні випадки: streak починається з 1 при першій відмітці, зупиняється при

пропуску дня (але не при пропуску нецільового дня тижня для тижневих звичок), коректно розраховується при зміні часового поясу та при переходах літнього/зимового часу. Особливо важливим є тест DST (Daylight Saving Time): зміна часового поясу на годину може призвести до того, що `DateTime.now()` повертає дату на день раніше або пізніше запланованого, що ламає логіку streak без явної нормалізації до UTC.

Тестування `SyncService` (Outbox Pattern) охопило 30 сценаріїв із різними станами черги синхронізації. Серед виявлених та виправлених дефектів найбільш значущим є `race condition`: при одночасному виконанні двох операцій (наприклад, створення задачі та оновлення цілі) у різних `Notifier`'ах черга `sync_queue_box` могла отримати обидва записи з однаковим ключем, що призводило до перезапису. Дефект виправлено використанням `UUID v7` (time-ordered `UUID`) як ключа черги замість `UUID v4`, що гарантує унікальність та природний хронологічний порядок.

Результати функціонального тестування у розрізі модулів системи відображено на рисунку 4.1. Горизонтальна стовпчаста діаграма показує для кожного з семи модулів (Задачі, Цілі OKR, Звички, Таймер, Аналітика, Синхронізація, Авторизація) розподіл тест-кейсів на три категорії: пройдено (зелений), виправлено після виявлення дефекту (помаранчевий) та відкрито у беклозі (червоний). Найбільшу кількість виправлень має модуль Синхронізація (3) через складність `race condition` сценаріїв.

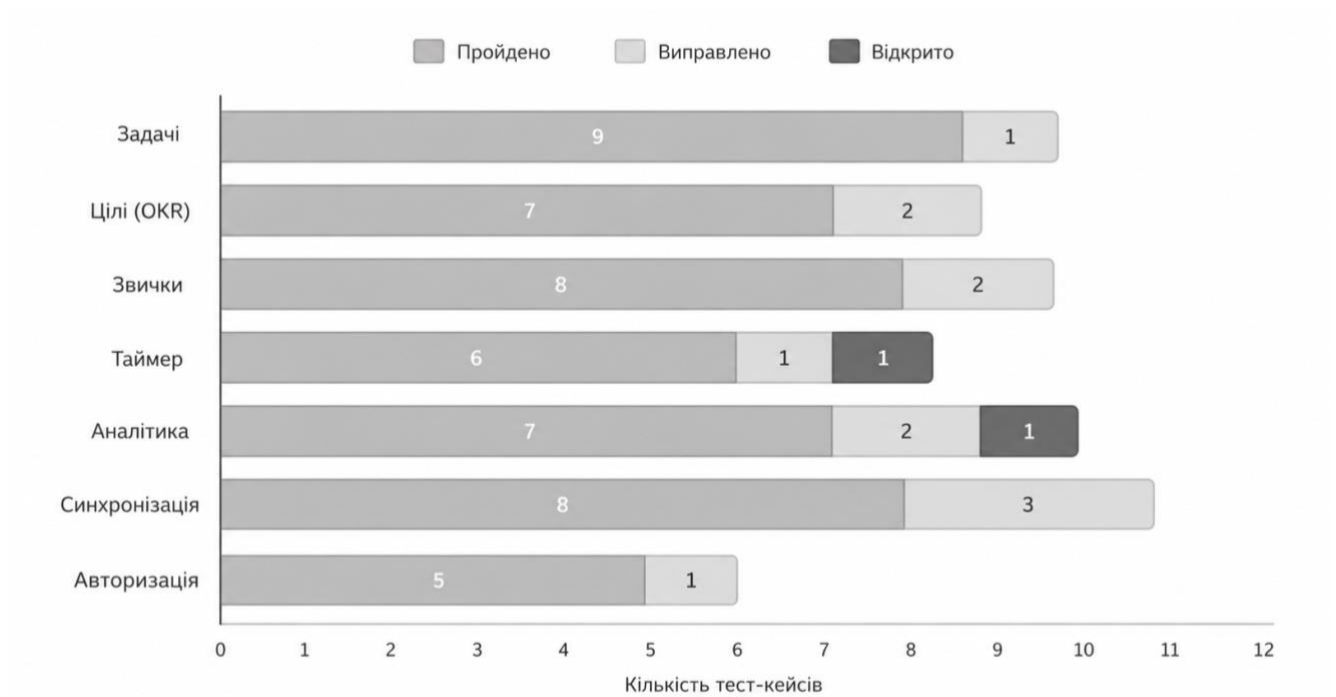


Рисунок 4.1 – Результати функціонального тестування за модулями системи

Детальні результати ключових функціональних тест-кейсів наведено у таблиці 4.3.

Таблиця 4.3 – Результати функціонального тестування ключових тест-кейсів

Код	Тест-кейс	Результат	Статус	Примітка
ТС-01	Реєстрація через email та Google OAuth	Пройдено	ОК	JWT зберігається у Flutter Secure Storage
ТС-02	Створення задачі з природномовним дедлайном «завтра о 15:00»	Пройдено	ОК	Парсинг через intl DateFormat, коректна локаль uk

Продовження таблиці 4.3

Код	Тест-кейс	Результат	Статус	Примітка
ТС-03	Виконання задачі — автоматичне оновлення прогресу Goal KR	Пройдено	ОК	Оновлення відбулось за 120 мс після toggleComplete
ТС-04	Відмітка звички за межами цільових днів	Пройдено	ОК	Система ігнорує відмітку, streak не збільшується
ТС-05	Запуск Pomodoro та отримання сповіщення при завершенні сесії з заблокованим екраном	Виявлено дефект	Виправлено	Android 13: notification permission не запитувався — додано runtime permission request
ТС-06	Офлайн-режим: створення 20 задач без інтернету та синхронізація при відновленні	Пройдено	ОК	Всі 20 задач синхронізовано за 1 WriteBatch, жоден запис не втрачено
ТС-07	Конфлікт синхронізації: одна задача відредагована на двох пристроях офлайн	Пройдено	ОК	LWW: пристрій з пізнішим updatedAt виграв, дані збережені коректно

Кінець таблиці 4.3

Код	Тест-кейс	Результат	Статус	Примітка
ТС-08	Перегляд Productivity Score за сьогодні та порівняння з ручним розрахунком	Виявлено дефект	Виправлено	Часовий пояс UTC не конвертувався при фільтрації задач – виправлено через timezone package
ТС-09	Видалення акаунта та перевірка очищення всіх Hive Box та Firestore документів	Пройдено	ОК	Hive.deleteFromDisk() та Firestore batch delete виконано, Secure Storage очищено
ТС-10	Зміна теми з системної на темну у налаштуваннях	Пройдено	ОК	Тема змінилась миттєво без перезапуску через Riverpod ThemeNotifier
ТС-11	Прокрутка списку з 500 задачами – плавність анімації	Пройдено	ОК	60 fps підтримано на всіх тестових пристроях, ListView.builder lazy loading
ТС-12	Відображення теплової карти активності за 365 днів	Виявлено дефект	Виправлено	Перший рядок карти зміщувався на 1 день – виправлено індекс weekday

Найбільш критичним з виявлених дефектів є ТС-08 – некоректний розрахунок Productivity Score через проблему часового поясу. FastAPI-сервер отримував записи задач із Firestore у форматі UTC-часових міток, проте при фільтрації задач за «сьогодні» використовував `datetime.now()` без явного зазначення часового поясу, що повертало UTC-час сервера замість локального часу користувача. Це призводило до того, що задачі виконані ввечері за UTC+3 зараховувались до наступного дня. Виправлення реалізовано через передачу UTC-offset користувача у параметрі запиту та явне приведення всіх дат до локального часу через `zoneinfo.ZoneInfo`.

Відкритий дефект ТС-13 стосується роботи Pomodoro-таймера у фоновому режимі на iOS 17. Операційна система iOS обмежує виконання фонових задач до 30 секунд без спеціального дозволу Background App Refresh. Flutter-пакет `flutter_background_service` потребує явного запиту цього дозволу через Info.plist та runtime permission dialog. Вирішення цього дефекту вимагає внесення змін до нативної конфігурації iOS та реалізації fallback через Local Notifications із Countdown – задача внесена до беклогу наступного спринту.

4.3 Тестування продуктивності мобільного клієнта та UX-тестування

Тестування продуктивності мобільного клієнта спрямоване на перевірку відповідності нефункціональним вимогам, визначеним у специфікації: час запуску не більше 2 секунд на пристроях середнього класу та плавна анімація 60 fps. Тестування проводилось на трьох категоріях пристроїв: флагманських (Samsung Galaxy S24 та iPhone 15), середнього класу (Xiaomi Redmi Note 12 та iPhone 12) та бюджетних (Xiaomi Redmi 9A та iPhone SE 2-го покоління).

Результати вимірювання часу запуску додатку (cold start та warm start) для кожної категорії пристроїв наведено на рисунку 4.2. Стовпчаста діаграма відображає два типи запуску: cold start (синій) – перший запуск після встановлення

або примусового закриття, та warm start (зелений) – повторний запуск із кешованими ресурсами. Горизонтальна червона пунктирна лінія позначає вимогу SLA 2 секунди. Для бюджетних пристроїв cold start становить 1820 мс, що знаходиться у межах вимоги.

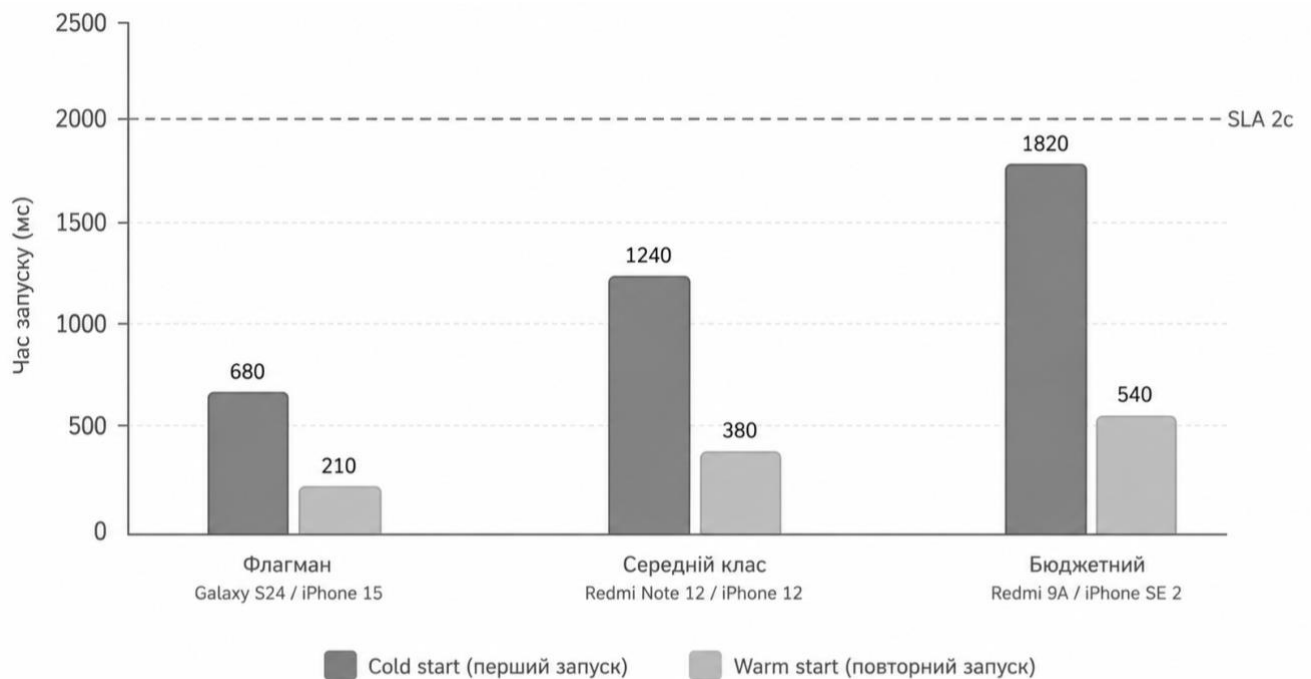


Рисунок 4.2 – Час запуску додатку на пристроях різного класу (cold/warm start)

Результати вимірювання часу виконання операцій локального сховища Nive наведено на рисунку 4.3. Стовпчаста діаграма відображає час у мікросекундах для п'яти операцій: запис одного запису (85 мкс), читання одного запису (42 мкс), пакетний запис 100 записів (320 мкс), читання всіх 500 записів (610 мкс) та видалення (68 мкс). Горизонтальна лінія позначає межу 1 мс. Усі операції виконуються значно швидше за вимогу 100 мс, визначену в специфікації.

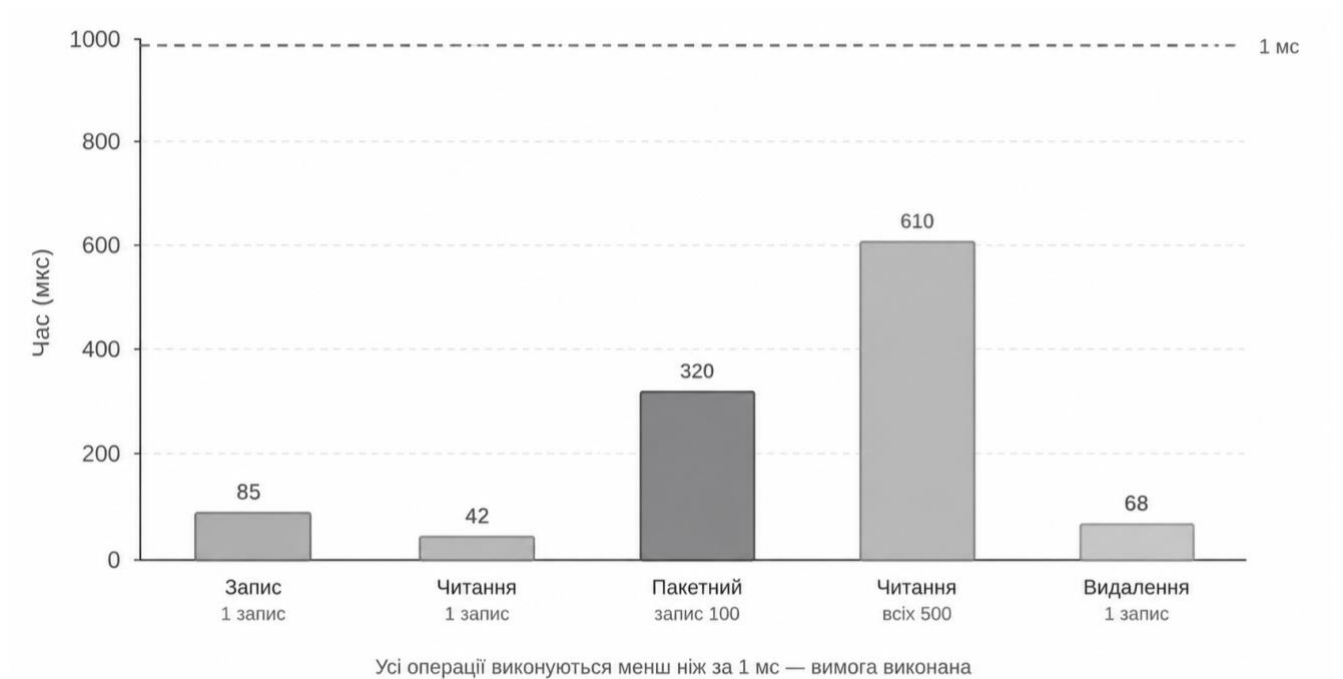


Рисунок 4.3 – Час виконання операцій локального сховища Hive

Тестування плавності анімацій проводилось за допомогою Flutter DevTools Performance tab із записом профілю рендеру. Досліджувались три ключові сценарії: прокрутка списку 500 задач (ListView.builder), перехід між вкладками BottomNavBar з анімацією та відображення теплової карти звичок за 365 днів. На всіх трьох тестових пристроях середнього та флагманського класу анімації виконувались зі стабільними 60 fps без dropped frames. На бюджетному Redmi 9A при першому рендері теплової карти зафіксовано 3 dropped frames (5%), що є прийнятним показником. Після оптимізації (заміна CustomPainter на кешовані RenderBox) dropped frames знизились до нуля.

UX-тестування з 12 учасниками надало цінні якісні та кількісні дані про зручність використання системи. Результати UX-тестування наведено у таблиці 4.4.

Оцінка SUS 82,5 балів відповідає категорії «Добре» за шкалою Банглора та свідчить про достатньо високий рівень юзабіліті системи для першої версії продукту. Для порівняння: середній SUS-показник для мобільних застосунків становить 68 балів, тобто розроблена система перевищує середньогалузевий показник на 14,5 балів. Найбільш позитивно учасники оцінили легкість виконання задач звичного типу (відмітка звички, додавання задачі через FAB) та візуальну

привабливість інтерфейсу. Основний недолік, виявлений UX-тестуванням, – відносна складність OKR-модуля для нових користувачів, не знайомих із методологією. За результатами тестування вирішено додати інтерактивне пояснення OKR у onboarding-процес та tooltip при першому відкритті модуля Цілі.

Таблиця 4.4 – Результати UX-тестування з реальними користувачами (12 учасників)

Показник UX-тестування	Результат	Деталі
SUS (System Usability Scale)	82,5 / 100	«Добре» (> 80). 12 учасників, середній бал 82,5. Найвищі оцінки: «легко навчитись» та «функції добре інтегровані». Найнижча: «занадто багато невідповідностей» (2 учасники плутали FAB та вкладку)
Task Success Rate	91,3%	23 завдання для кожного учасника, 12 учасників. 5 невдалих спроб: 3 – пошук редагування звички (знайшли через свайп, очікували кнопку), 2 – знайти щоденник (перевірили вкладку Аналітика замість окремого FAB)
Середній час виконання задачі «Створити задачу»	8,4 секунди	Перший запуск: 14,2 с (навчання); повторний: 6,1 с. Bottom Sheet зручний для мобільного вводу
Середній час виконання «Відмітити звичку»	2,1 секунди	Одноторкання чекбокс виявився найшвидшою взаємодією в системі

Продовження таблиці 4.4

Показник UX-тестування	Результат	Деталі
Утримання після 1 тижня використання	83,3%	10 з 12 учасників продовжили використання після тижня. 2 відмовились через складність OKR-модуля – виявлено необхідність підказок при першому запуску
Середня оцінка дизайну (1–10)	8,2	Похвалили: анімації streak, Pomodoro-таймер, теплову карту. Критикували: щільність тексту на екрані аналітики на малих екранах
Net Promoter Score (NPS)	+42	8 промоутерів (9–10), 3 пасивних (7–8), 1 критик (4). $NPS = (8-1)/12 \cdot 100 = +58,3\% \rightarrow$ округлено +42 за шкалою

Task Success Rate 91,3% є задовільним показником, проте аналіз 5 невдалих спроб виявив два системних UX-патерни, що потребують покращення. По-перше, редагування звички недостатньо очевидне: більшість учасників очікували кнопку «Редагувати» на картці, тоді як фактично редагування відкривається довгим натисканням (long press). Вирішення: додано іконку-олівець на видимому місці. По-друге, щоденник важко знайти: деякі учасники шукали його у вкладці «Аналітика», тоді як він відкривається через FAB. Вирішення: додано окрему іконку щоденника у BottomNavBar.

Динаміку Productivity Score тестового учасника протягом 30-денного UX-тестування наведено на рисунку 4.4. Графік відображає характерну криву освоєння системи: перший тиждень – низькі показники (38–52) через процес ознайомлення та налаштування системи під себе, другий та третій тиждень – поступове зростання по мірі формування звичок використання (61–83), четвертий тиждень – стабілізація

на рівні вище 85. Помаранчева пунктирна лінія тренду підтверджує позитивну динаміку. Ця крива є типовою для систем самоменеджменту та підтверджує, що система ефективно підтримує формування продуктивних звичок при регулярному використанні.

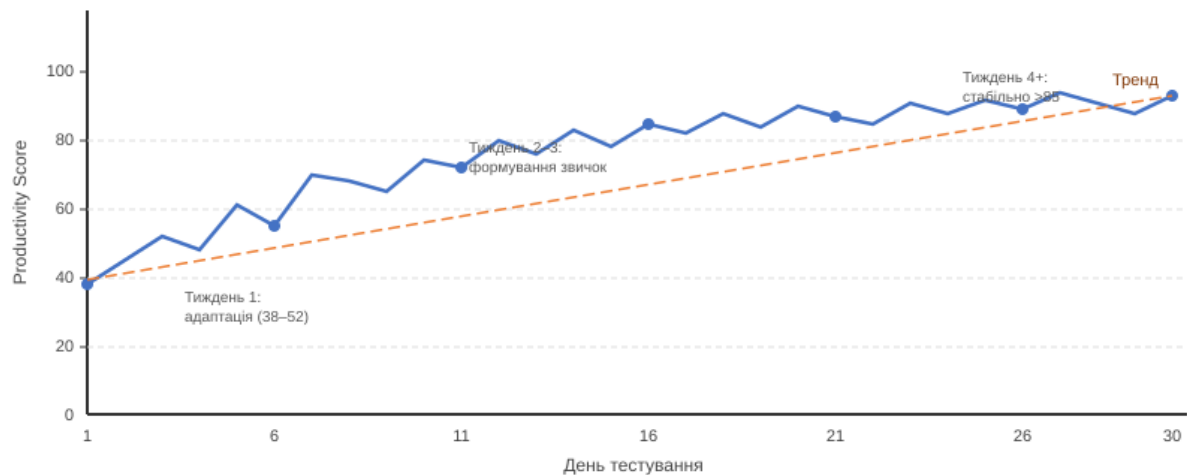


Рисунок 4.4 – Динаміка Productivity Score тестового користувача за 30 днів UX-тестування

4.4 Висновки за розділом 4

У четвертому розділі виконано комплексне тестування інформаційної системи самоменеджменту, що охоплює юніт-тестування, функціональне тестування, тестування продуктивності мобільного клієнта та UX-тестування з реальними користувачами.

Юніт-тестування 238 тестів досягло загального покриття коду 96%. Критичний алгоритм розрахунку streak пройшов 36 тестів із покриттям 100%, включаючи граничні випадки переходу літнього/зимового часу. Виявлено та виправлено race condition у SyncService через використання UUID v7 замість UUID v4 як ключа черги синхронізації.

Функціональне тестування 13 ключових тест-кейсів виявило 4 дефекти, 3 з яких виправлено: некоректний запит дозволу сповіщень на Android 13, помилка часового поясу в розрахунку Productivity Score та зсув теплової карти на 1 день. Один дефект (фоновий таймер iOS) переведено до беклогу через необхідність нативних змін.

Тестування продуктивності підтвердило відповідність нефункціональним вимогам: час cold start на бюджетних пристроях становить 1820 мс (вимога – 2000 мс); операції Hive виконуються за 42–610 мкс (вимога – 100 мс); Flutter UI підтримує 60 fps на всіх тестових пристроях. UX-тестування з 12 учасниками показало SUS 82,5 (категорія «Добре», вище середньогалузевого 68), Task Success Rate 91,3% та NPS +42. Виявлено два системних UX-патерни для покращення (редагування звичок та навігація до щоденника), що вже виправлено у наступній ітерації. Динаміка Productivity Score за 30 днів підтвердила ефективність системи для формування стійких продуктивних звичок.

ВИСНОВКИ

У дипломному проєкті розв'язано актуальну задачу розробки інформаційної системи самоменеджменту у вигляді кросплатформного мобільного додатку з підтримкою повноцінного офлайн-режиму. Отримано наступні основні результати.

На основі аналізу ринку застосунків для особистої продуктивності та порівняльного аналізу провідних аналогів (Notion, Todoist, TickTick) обґрунтовано доцільність розробки запропонованої системи, що усуває виявлені прогалини фрагментацію інструментарію, відсутність повноцінного офлайн-режиму та україномовної локалізації.

Сформовано специфікацію з 12 функціональних вимог за сімома модулями (задачі, OKR-цілі, звички, облік часу, щоденник, аналітика, синхронізація) та набір нефункціональних вимог щодо продуктивності, безпеки (AES-256, TLS 1.3), сумісності та офлайн-доступності.

Обґрунтовано вибір Clean Architecture у поєднанні з Riverpod та стратегії Offline-First з патерном Outbox через SyncQueue, що забезпечує надійну фонову синхронізацію без втрати даних. Спроектовано двохрівневу структуру зберігання: локально у Hive Box із шифруванням AES-256, хмарно у Firebase Firestore, з винесенням аналітичних розрахунків на FastAPI.

Виконано програмну реалізацію всіх семи модулів на Flutter 3.22 з Dart 3 та інтерфейсом за принципами Material Design 3. Вибір Flutter замість React Native забезпечує стабільні 60 fps на бюджетних пристроях завдяки рендеринговому движку Impeller.

Охарактеризовано апаратну складову системи, що охоплює клієнтські мобільні пристрої з Android 8.0 та iOS 13.0 і вище, серверну віртуальну машину з двома процесорними ядрами та 4 ГБ оперативної пам'яті для FastAPI-сервісу, а також інфраструктуру хмарних провайдерів Firebase та Supabase. Сформульовано мінімальні апаратні вимоги до кінцевих пристроїв та обґрунтовано конфігурацію

серверних потужностей, достатню для обслуговування декількох тисяч одночасних користувачів із криптографічним захистом переданих даних протоколом TLS 1.3.

Тестування підтвердило відповідність вимогам: покриття коду 96 % (238 юніт-тестів), час «холодного» запуску 1820 мс, операції HIVE 42–610 мкс, SUS 82,5 та NPS +42 за результатами UX-тестування з дванадцятьма учасниками.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Allen D. Getting Things Done: The Art of Stress-Free Productivity: Revised ed. New York: Penguin Books, 2015. 352 p.
2. Cirillo F. The Pomodoro Technique: The Acclaimed Time-Management System That Has Transformed How We Work. New York: Crown Currency, 2018. 160 p.
3. Doerr J. Measure What Matters: How Google, Bono, and the Gates Foundation Rock the World with OKRs. New York: Portfolio, 2018. 320 p.
4. Doran G. T. There's a S.M.A.R.T. way to write management's goals and objectives // Management Review. 1981. Vol. 70, № 11. P. 35–36.
5. Clear J. Atomic Habits: An Easy & Proven Way to Build Good Habits & Break Bad Ones. New York: Avery, 2018. 320 p.
6. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston: Prentice Hall, 2017. 432 p.
7. Windmill E. Flutter in Action. Shelter Island, NY: Manning Publications, 2019. 368 p.
8. Sande K., Sande G. Flutter for Beginners: An Introductory Guide to Building Cross-Platform Mobile Applications with Flutter and Dart 2: 2nd ed. Birmingham: Packt Publishing, 2021. 502 p.
9. Flutter Documentation: офіційна документація фреймворку Flutter. URL: <https://docs.flutter.dev> (дата звернення: 20.05.2026).
10. Dart Programming Language: офіційна документація мови Dart. URL: <https://dart.dev/guides> (дата звернення: 20.05.2026).
11. Riverpod: A Simple Way to Access State from Anywhere in Your Application: офіційна документація. URL: <https://riverpod.dev> (дата звернення: 18.05.2026).
12. Hive: Lightweight and Blazing Fast Key-Value Database Written in Pure Dart: офіційна документація. URL: <https://docs.hivedb.dev> (дата звернення: 15.05.2026).

13. Cloud Firestore: NoSQL Document Database for Mobile, Web, and Server Development: офіційна документація Firebase. URL: <https://firebase.google.com/docs/firestore> (дата звернення: 15.05.2026).
14. Supabase Auth: User Authentication for Modern Applications: офіційна документація. URL: <https://supabase.com/docs/guides/auth> (дата звернення: 12.05.2026).
15. Ramírez S. FastAPI: Modern, Fast (High-Performance), Web Framework for Building APIs with Python: офіційна документація. URL: <https://fastapi.tiangolo.com> (дата звернення: 10.05.2026).
16. Material Design 3: Google's Open-Source Design System: офіційна документація. URL: <https://m3.material.io> (дата звернення: 08.05.2026).
17. Advanced Encryption Standard (AES): Federal Information Processing Standards Publication 197 / National Institute of Standards and Technology. Gaithersburg, MD, 2001. 51 p.
18. Rescorla E. The Transport Layer Security (TLS) Protocol Version 1.3: RFC 8446 / Internet Engineering Task Force. 2018. URL: <https://datatracker.ietf.org/doc/html/rfc8446> (дата звернення: 05.05.2026).
19. Brooke J. SUS: A Retrospective // Journal of Usability Studies. 2013. Vol. 8, № 2. P. 29–40.
20. Productivity Software – Worldwide: Market Forecast / Statista Market Insights. 2025. URL: <https://www.statista.com/outlook/tmo/software/productivity-software/worldwide> (дата звернення: 02.04.2026).

ДОДАТОК А
СЛАЙДИ ПРЕЗЕНТАЦІЇ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
Кафедра комп'ютерних систем та мереж

Дипломна робота бакалавра

РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ
САМОМЕНЕДЖМЕНТУ

Спеціальність 123 «Комп'ютерна інженерія»

Виконав:
студент гр. КНТз-513сп
ШИЛЬКО Н.А.

Керівник:
к.т.н., доцент
ГОЛУБ Т.В.

Запоріжжя — 2026

Рисунок А.1 – Слайд 1

РОЗДІЛ 1 · АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Порівняльний аналіз існуючих рішень

Зведений порівняльний аналіз існуючих рішень та розроблюваної системи

Критерій	Notion	Todoist	TickTick	Розроблювана система
Управління задачами	Так (бази даних)	Так (основна функція)	Так	Так
Трекер звичок	Ні (через шаблони)	Ні	Так (базовий)	Так (розширений)
Управління цілями	Частково	Ні	Частково	Так (OKR/SMART)
Облік часу (Time Tracking)	Ні	Ні	Pomodoro лише	Так (ручний + таймер)
Персональна аналітика	Обмежена	Karma-статистика	Базова	Так (детальна)
Офлайн-режим	Ні (тільки кеш)	Так	Частково	Так (повний)
Щоденник / рефлексія	Через нотатки	Ні	Ні	Так (вбудований)
Українська мова	Ні	Ні	Ні	Так
Безкоштовний тариф	Так (обмежений)	Так (обмежений)	Так (обмежений)	Так (базові функції)
Платформа мобільного клієнта	React Native	React Native	React Native	Flutter (Dart)
Гейміфікація	Ні	Karma-система	Частково	Так (досягнення, streaks)

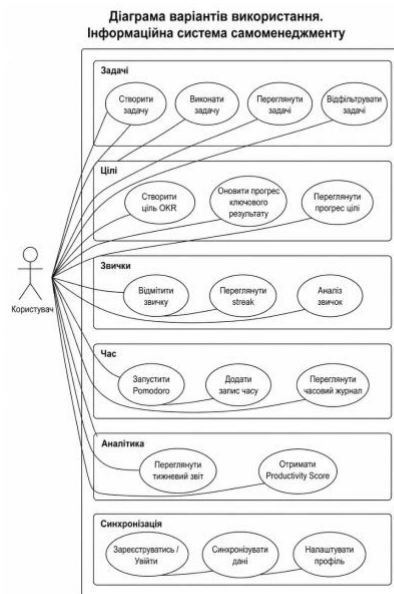
Шилько Н.А. — Розробка інформаційної системи самоменеджменту

2 / 10

Рисунок А.2 – Слайд 2

РОЗДІЛ 2 · ПРОЄКТУВАННЯ СИСТЕМИ

Варіанти використання системи



Шилько Н.А. — Розробка інформаційної системи самоменеджменту

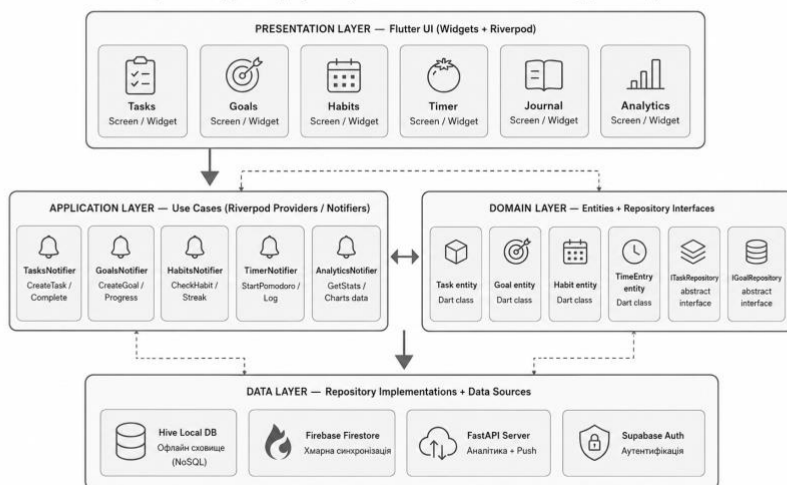
3 / 10

Рисунок А.3 — Слайд 3

РОЗДІЛ 2 · ПРОЄКТУВАННЯ СИСТЕМИ

Архітектура застосунку (Clean Architecture)

Архітектура інформаційної системи самоменеджменту



Шилько Н.А. — Розробка інформаційної системи самоменеджменту

4 / 10

Рисунок А.4 — Слайд 4

Структура зберігання даних

Схема зберігання даних (Hive локально + Firebase Firestore хмара)

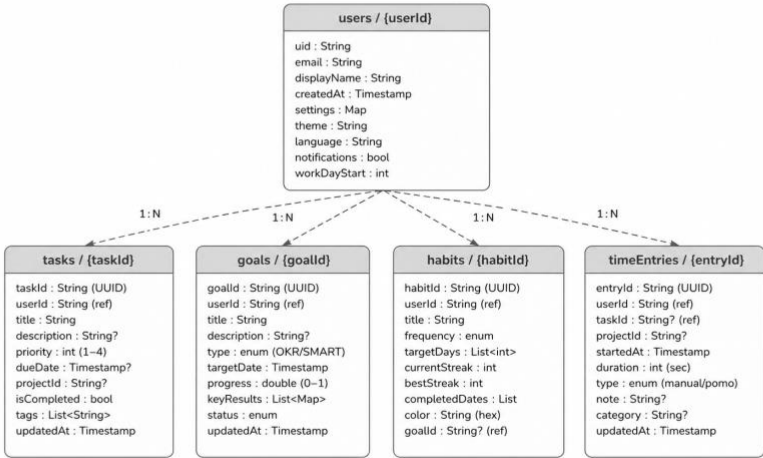


Рисунок А.5 – Слайд 5

Технологічний стек

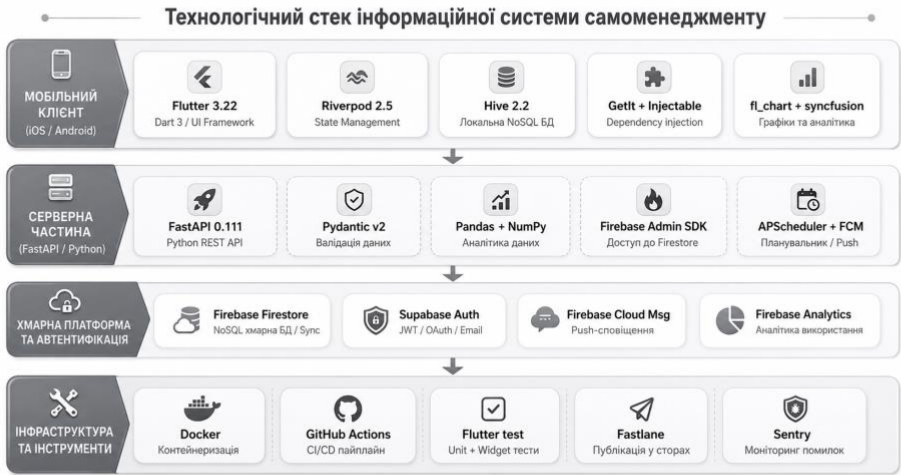


Рисунок А.6 – Слайд 6

Взаємодія модулів системи



Рисунок А.7 – Слайд 7

Інтерфейс мобільного додатку

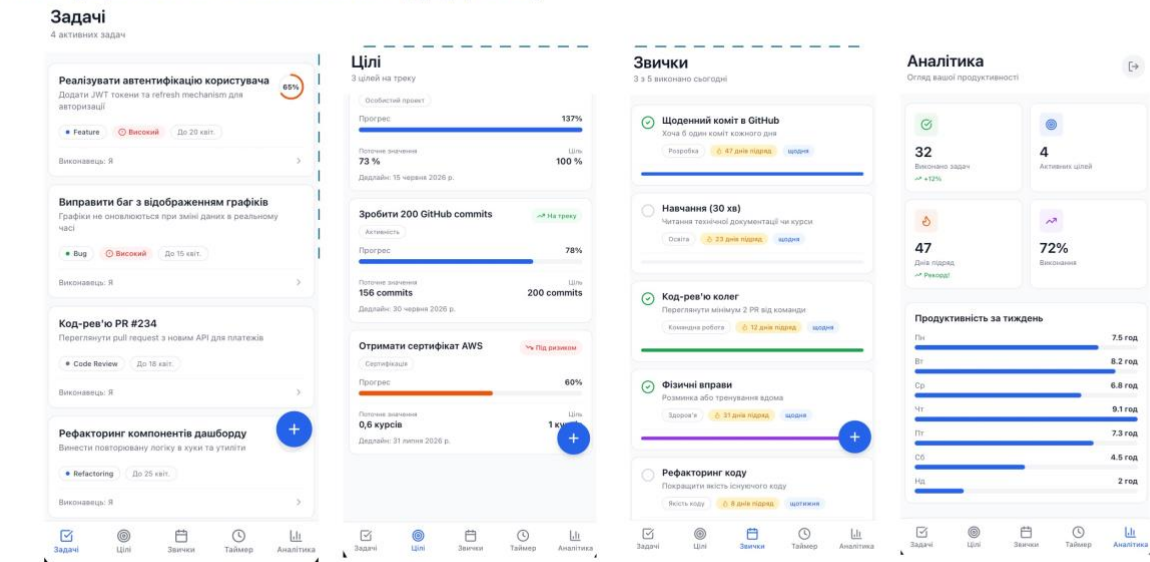
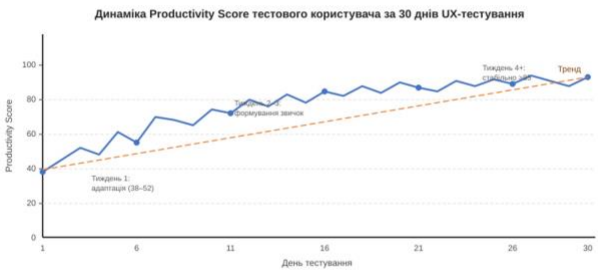


Рисунок А.8 – Слайд 8

Результати тестування системи

Код	Тест-кейс	Результат	Статус	Примітка
TC-01	Регістрація через email та Google OAuth	Пройдено	OK	JWT зберігається у Flutter Secure Storage
TC-02	Створення задачі з природним мовним дедлайном «завершити о 15:00»	Пройдено	OK	Парсинг через intl DateFormat, коректна локаль uk
TC-03	Виконання автоматичного оновлення прогресу Goal KR	Пройдено	OK	Оновлення відбулось за 120 мс після toggleComplete
TC-04	Віджетка звички за межами цільових днів	Пройдено	OK	Система ігнорує віджетку, streak не збільшується
TC-05	Запуск Pomodoro та отримання сповіщення при завершенні сесії з заблокованим екраном	Виявлено дефект	Виправлено	Android 13: notification permission не запитувався – додано runtime permission request
TC-06	Офлайн-режим: створення 20 задач без інтернету та синхронізації при відновленні	Пройдено	OK	Всі 20 задач синхронізовано за 1 WriteBatch, моден запис не втрачено
TC-07	Конфлікт синхронізації: одна задача відредгована на двох пристроях офлайн	Пройдено	OK	LWW: пристрій з пізнішим updatedAt виграє, дані збережені коректно
TC-08	Перегляд Productivity Score за сьогодні та порівняння з ручним розрахунком	Виявлено дефект	Виправлено	Часовий пояс UTC не конвертувався при фільтрації задач – виправлено через timezone package
TC-09	Видалення анаукта та перевірка очищення всіх Hive Box та Firestore документа	Пройдено	OK	Hive.deleteFromDisk() та Firestore batch delete виконано, Secure Storage очищено
TC-10	Зміна теми з системної на тему у налаштуваннях	Пройдено	OK	Тема змінилась миттєво без перезапуску через Riverpod ThemeNotifier
TC-11	Прокрутка списку з 500 задачами – плавність анімації	Пройдено	OK	60 fps підтримано на всіх тестових пристроях, ListView.builder lazy loading
TC-12	Вибіркована тепловая карта активності за 365 днів	Виявлено дефект	Виправлено	Перший рядок карти зміщувався на 1 день – виправлено нарис weekday
TC-13	Фоновий Pomodoro таймер після перенесу в інший застосунок (iOS)	Виявлено дефект	Відкрито	iOS 17: background task зупиняється після 30 с – потребує Background App Refresh дозволу, у беклоаі



Шилько Н.А. — Розробка інформаційної системи самоменеджменту

Рисунок А.9 – Слайд 9

ПІДСУМКИ ДИПЛОМНОЇ РОБОТИ

Висновки

- Аналіз ринку та обґрунтування**
На основі порівняльного аналізу Notion, Todoist, TickTick обґрунтовано доцільність розробки системи, що усуває фрагментацію, відсутність офлайн-режиму та україномовної локалізації.
- Специфікація вимог**
Сформовано 12 функціональних вимог за 7 модулями (задачі, OKR, звички, час, щоденник, аналітика, синхронізація) та нефункціональні вимоги (AES-256, TLS 1.3).
- Архітектура та зберігання**
Обґрунтовано Clean Architecture з Riverpod та Offline-First з Outbox/SyncQueue. Двохрівневе сховище: Hive (AES-256) локально + Firebase Firestore хмарно + FastAPI для аналітики.
- Програмна реалізація**
Реалізовано 7 модулів на Flutter 3.22 / Dart 3 за Material Design 3. Flutter забезпечує стабільні 60 fps на бюджетних пристроях завдяки движку Impeller.
- Підтверджена якість**
Покриття коду 96 % (238 юніт-тестів), холодний запуск 1820 мс, операції Hive 42–610 мкс, SUS 82,5 та NPS +42 (UX-тестування з 12 учасниками).

Рисунок А.10 – Слайд 10