ЛЬВІВСЬКИЙ ТОРГОВЕЛЬНО-ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Факультет міжнародних економічних відносин та інформаційних технологій

Кафедра комп'ютерних наук, прикладної та вищої математики

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ВІДСТЕЖЕННЯ ПРОДУКТИВНОСТІ

Виконав: здобувач вищої освіти першого (бакалаврського) рівня
ОПП "Комп'ютерні науки"
спеціальності 122 "Комп'ютерні науки"
денної форми навчання

_____ Мандзюк Станіслав Валерійович

(підпис)

Науковий керівник

_____ доц.Зацерковний Ростислав Григорович

(підпис)

Львів – 2026

ЛЬВІВСЬКИЙ ТОРГОВЕЛЬНО-ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
Факультет міжнародних економічних відносин та інформаційних технологій
Кафедра комп'ютерних наук, прикладної та вищої математики
Перший (бакалаврський) рівень вищої освіти
Спеціальність 122 "Комп’ютерні науки"
Освітньо-професійна програма "Комп’ютерні науки"

"ЗАТВЕРДЖУЮ"
Завідувач кафедри комп'ютерних наук,
прикладної та вищої математики
_____ З. Г. Новосад
"___" _____ 20___ р.

Завдання
до кваліфікаційної роботи

Мандзюк Станіслав Валерійович

(прізвище, ім’я, по батькові)

1. Тема роботи: Розробка мобільного застосунку для відстеження продуктивності.
Керівник роботи Зацерковний Ростислав Григорович, доктор філософії,
затверджена наказом по Університету ***№ 991/03 від 06 жовтня 2025 р.***

2. Термін здачі здобувачем вищої освіти роботи 05 червня 2026 р.

3. Вхідні дані до роботи: література, документація, матеріали та сайти Internet, що пов’язані з розробкою мобільних застосунків для відстеження продуктивності.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці): 1) дослідження предметної області мобільних застосунків для відстеження продуктивності та аналіз існуючих рішень; 2) аналіз засобів реалізації програмного забезпечення й обґрунтування технологічного стеку; 3) розробка та реалізація мобільного застосунку.

5. Перелік графічного матеріалу: рисунки, діаграми та фрагменти програмного коду відповідно до змісту роботи.

6.Консультанти розділів кваліфікаційної роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	Завдання прийняв

7. Дата видачі завдання: "___" _____ 2026 р.

Календарний план

№	Найменування етапів виконання кваліфікаційної роботи	Терміни виконання етапів	Примітка
1.	Отримання завдання й аналіз інформаційних джерел за темою роботи	до 11.12.2025	виконано
2.	Дослідження предметної області та оцінка існуючих рішень	до 13.01.2025	виконано
3.	Аналіз засобів реалізації програмного забезпечення	до 07.02.2026	виконано
4.	Проєктування архітектури та структури бази даних застосунку	до 01.04.2026	виконано
5.	Реалізація функціональних модулів застосунку	до 15.04.2026	виконано
6.	Розробка інтерфейсу користувача	до 22.04.2026	виконано
7.	Тестування розробленого застосунку	до 16.05.2026	виконано
8.	Оформлення кваліфікаційної роботи. Подання роботи на перевірку керівнику	До 20.05.2026	виконано
9.	Подання роботи для перевірки на рівень запозичень	до 27.05.2026	виконано
10.	Подання кваліфікаційної роботи на кафедру	до 5.06.2026	виконано
11.	Захист кваліфікаційної роботи на засіданні екзаменаційної комісії	05.06.2026-06.06.2026	виконано

Здобувач вищої освіти _____ Мандзюк Станіслав Валерійович

Керівник роботи _____ д.філ. Зацерковний Р. Г.

Реферат

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел з 30 найменувань та додатків. Основний зміст роботи викладено на 40 сторінках і містить 7 рисунків та 1 таблицю.

Об’єктом дослідження є процес управління особистою продуктивністю користувача із застосуванням сучасних мобільних технологій.

Предметом дослідження є методи, програмні засоби та технології розробки мобільного застосунку для відстеження продуктивності, аналізу активності користувача та управління завданнями.

Метою дослідження є проєктування та розробка мобільного застосунку для відстеження власної продуктивності, який забезпечує ефективне планування завдань, моніторинг активності користувача та формування аналітичних звітів.

У першому розділі проведено дослідження предметної області мобільних застосунків для відстеження продуктивності, проаналізовано існуючі програмні рішення та визначено основні вимоги до системи.

У другому розділі виконано аналіз засобів реалізації програмного забезпечення, обґрунтовано вибір технологічного стеку, мови програмування Kotlin, бази даних SQLite та середовища розробки Android Studio.

У третьому розділі розглянуто архітектуру мобільного застосунку, особливості взаємодії з мобільним пристроєм, реалізацію функціональних модулів системи та результати тестування розробленого програмного забезпечення.

Ключові слова: МОБІЛЬНИЙ ЗАСТОСУНОК, ПРОДУКТИВНІСТЬ, KOTLIN, ANDROID STUDIO, SQLITE, УПРАВЛІННЯ ЗАВДАННЯМИ, АНАЛІТИКА, ІНТЕРФЕЙС КОРИСТУВАЧА, МОБІЛЬНА РОЗРОБКА, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

Abstract

The qualification thesis consists of an introduction, three chapters, conclusions, a list of 30 references and appendices. The main content of the thesis is presented on 40 pages and includes 7 figures and 1 table.

The object of the research is the process of personal productivity management using modern mobile technologies.

The subject of the research is methods, software tools and technologies for developing a mobile application for productivity tracking, user activity analysis and task management.

The aim of the research is to design and develop a mobile application for personal productivity tracking that provides efficient task planning, user activity monitoring and analytical reporting.

The first chapter investigates the subject area of productivity tracking mobile applications, analyzes existing software solutions and defines the main system requirements.

The second chapter presents an analysis of software development tools, substantiates the choice of the technological stack, Kotlin programming language, SQLite database and Android Studio development environment.

The third chapter describes the architecture of the mobile application, interaction with mobile devices, implementation of functional modules and testing results of the developed software product.

Keywords: MOBILE APPLICATION, PRODUCTIVITY, KOTLIN, ANDROID STUDIO, SQLITE, TASK MANAGEMENT, ANALYTICS, USER INTERFACE, MOBILE DEVELOPMENT, SOFTWARE ENGINEERING.

ЗМІСТ

ВСТУП *7*

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ МОБІЛЬНИХ ЗАСТОСУНКІВ ДЛЯ ВІДСТЕЖЕННЯ ПРОДУКТИВНОСТІ *10*

1.1 Особливості галузі **10**

1.2 Аналіз існуючих мобільних застосунків для відстеження продуктивності **13**

1.3 Вимоги до мобільного застосунку для відстеження продуктивності **16**

РОЗДІЛ 2. АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ *19*

2.1 Вибір мови програмування та фреймворків **19**

2.2 Мова програмування Kotlin **21**

2.3 SQLite **23**

2.4 Вибір середовища розробки **27**

3.1 Архітектура мобільного застосунку **30**

3.2 Системні вимоги та взаємодія з пристроєм **33**

3.3 Модуль збору даних **37**

3.4 Модуль обробки та аналізу даних **40**

3.5 Інтерфейс користувача та взаємодія **42**

ВИСНОВКИ *45*

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 48

Додаток А *50*

ВСТУП

Сучасний етап розвитку інформаційних технологій характеризується стрімким поширенням мобільних рішень, цифрових сервісів та інтелектуальних систем, що спрямовані на оптимізацію діяльності людини. Одним із ключових напрямів цифрової трансформації є розробка мобільних застосунків, які забезпечують автоматизацію повсякденних процесів, підвищення ефективності використання часу та формування продуктивних поведінкових моделей. У цих умовах особливого значення набуває створення ефективних програмних систем, здатних забезпечувати моніторинг, аналіз і покращення особистої продуктивності користувача в реальному часі.

Зростаюча складність сучасного способу життя, велика кількість задач і інформаційних потоків формують потребу у впровадженні спеціалізованих інструментів, які дозволяють людині ефективно керувати власною діяльністю. Мобільні застосунки для відстеження продуктивності виступають універсальними рішеннями, що поєднують функції планування, контролю виконання завдань, аналізу активності та формування аналітичних звітів. Водночас підвищення вимог до таких систем зумовлює необхідність глибокої інтеграції програмних компонентів, алгоритмів аналізу даних і інтерфейсів взаємодії з користувачем.[4]

Особливої актуальності набуває питання якості взаємодії між користувачем і мобільним застосунком, оскільки саме від цього залежить рівень залученості, ефективність використання системи та досягнення поставлених цілей. Наявність недоліків у проєктуванні програмного забезпечення, таких як складний інтерфейс, низька швидкодія, відсутність персоналізації або обмежена функціональність, може суттєво знижувати ефективність застосунку та мотивацію користувача до його використання.

Актуальність даного дослідження зумовлена необхідністю розробки сучасного мобільного застосунку для відстеження власної продуктивності, який забезпечує високу ефективність, надійність і гнучкість у процесі управління особистими задачами. Важливим аспектом є також можливість масштабування

функціоналу, інтеграції з іншими цифровими сервісами та забезпечення інтуїтивно зрозумілої взаємодії з користувачем.

Об’єктом дослідження є процес розробки мобільного застосунку для відстеження продуктивності користувача в умовах сучасного цифрового середовища.

Предметом дослідження виступає архітектура та функціональні компоненти мобільного застосунку, що включають програмні модулі планування задач, аналітики продуктивності, системи збереження даних, а також алгоритми взаємодії між складовими системи.

Метою кваліфікаційної роботи є проєктування та розробка мобільного застосунку для відстеження власної продуктивності, який забезпечує ефективне виконання поставлених задач, адаптивність до індивідуальних потреб користувача та стабільну роботу в режимі реального часу.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1) проаналізувати предметну область систем управління продуктивністю та існуючі підходи до їх реалізації;
2) дослідити сучасні технології та платформи, що використовуються у розробці мобільних застосунків;
3) сформулювати функціональні та нефункціональні вимоги до застосунку;
4) обґрунтувати вибір архітектури системи та програмного забезпечення;
5) розробити структурну модель взаємодії компонентів мобільного застосунку;
6) реалізувати програмне забезпечення для управління задачами та збору даних про активність користувача;
7) забезпечити обробку даних у реальному часі та формування аналітичних показників;
8) провести тестування розробленого застосунку на надійність, продуктивність та відповідність заданим вимогам.

Теоретичне значення дослідження полягає в узагальненні підходів до проєктування мобільних застосунків у сфері підвищення продуктивності, а також у формуванні принципів інтеграції аналітичних і функціональних компонентів в єдину ефективну систему.

Практичне значення отриманих результатів полягає у створенні мобільного застосунку, який може бути використаний для планування задач, моніторингу активності та підвищення особистої ефективності користувача, забезпечуючи оптимізацію використання часу та зменшення впливу людського фактору.

Апробація результатів дослідження здійснювалася шляхом комплексного тестування розробленого мобільного застосунку в умовах, наближених до реального використання. Перевірка ефективності системи проводилася на основі типових сценаріїв використання, що включають створення задач, відстеження їх виконання, аналіз активності та формування звітів. Особлива увага приділялася оцінці стабільності роботи застосунку, швидкодії обробки даних, а також точності аналітичних результатів. Додатково було проведено серію експериментів для визначення рівня зручності використання та адаптивності системи до різних сценаріїв.

Структура роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків і списку використаних джерел.

У першому розділі здійснено аналіз предметної області мобільних застосунків для відстеження продуктивності, визначено ключові особливості та вимоги до системи.

У другому розділі обґрунтовано вибір технологій та розроблено архітектуру застосунку.

У третьому розділі описано процес реалізації та результати тестування системи.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ МОБІЛЬНИХ ЗАСТОСУНКІВ ДЛЯ ВІДСТЕЖЕННЯ ПРОДУКТИВНОСТІ

1.1 Особливості галузі

Сфера мобільних застосунків для відстеження продуктивності є однією з ключових складових сучасної цифрової інфраструктури, яка забезпечує оптимізацію діяльності користувачів, підвищення ефективності використання часу та покращення якості виконання повсякденних задач. Такі застосунки виступають універсальними інструментами для управління особистою ефективністю – від планування задач і контролю дедлайнів до аналізу поведінкових патернів і формування звітів про продуктивність, що зумовлює зростаючі вимоги до їх функціональності, зручності та адаптивності.[4]

Програмні комплекси, що лежать в основі мобільних застосунків продуктивності, охоплюють широкий спектр компонентів і технологій, спрямованих на забезпечення ефективної взаємодії між користувачем і цифровим середовищем. До таких компонентів належать інтерфейси користувача (UI), логіка обробки даних, системи збереження інформації, аналітичні модулі, а також алгоритми персоналізації. Кожен із цих елементів має свої специфічні вимоги, які визначають загальну ефективність функціонування застосунку.

Особливістю сучасних мобільних застосунків [8] є необхідність забезпечення стабільної роботи в умовах постійної зміни контексту використання. Наприклад, користувач може взаємодіяти із застосунком у різний час доби, за різних умов навантаження, використовуючи різні сценарії – від швидкого додавання задачі до глибокого аналізу власної продуктивності. У таких умовах система повинна забезпечувати швидку реакцію, коректне збереження даних та інтуїтивну взаємодію.

Реалізація мобільних застосунків для відстеження продуктивності здійснюється з використанням різних технологічних підходів, вибір яких залежить від складності системи, необхідного функціоналу та вимог до продуктивності. Наприклад, прості застосунки можуть використовувати

локальні бази даних і мінімальний набір функцій, тоді як більш складні рішення включають аналітичні модулі, синхронізацію з хмарними сервісами та елементи штучного інтелекту.

Рівень розвитку мобільних застосунків значною мірою залежить від загального стану технологічного середовища та цифрової культури користувачів. Зростання попиту на інструменти саморозвитку, тайм-менеджменту та підвищення ефективності стимулює створення нових рішень, водночас конкуренція на ринку змушує розробників постійно вдосконалювати функціональність і користувацький досвід.

Крім того, існують відмінності у рівні використання таких застосунків у різних регіонах. У технологічно розвинених країнах активно впроваджуються інноваційні рішення з використанням аналітики даних і штучного інтелекту, тоді як у менш розвинених регіонах впровадження може стримуватися через обмежений доступ до сучасних технологій або низький рівень цифрової грамотності.

Таким чином, розробка мобільних застосунків для відстеження продуктивності є складним багатокомпонентним процесом, що потребує врахування технічних, поведінкових та організаційних аспектів. Ефективна реалізація таких систем дозволяє підвищити рівень самоконтролю користувача, оптимізувати використання часу та зменшити вплив зовнішніх відволікаючих факторів.

З технологічної точки зору галузь активно розвивається завдяки впровадженню сучасних цифрових інструментів та інноваційних підходів. Використання мобільних платформ, хмарних сервісів і API дозволяє забезпечити безперервну синхронізацію даних і доступ до інформації з різних пристроїв. Крім того, застосування засобів аналітики дає змогу оцінювати ефективність діяльності користувача та формувати рекомендації.

У перспективі очікується зростання ролі інтелектуальних технологій, зокрема алгоритмів машинного навчання, які здатні аналізувати поведінку користувача, прогнозувати рівень навантаження та пропонувати оптимальні

сценарії планування. Це відкриває можливості для створення більш адаптивних і персоналізованих застосунків.

Сучасні тенденції розвитку пов’язані з інтеграцією різних технологій, таких як хмарні обчислення, великі дані та Інтернет речей (IoT). Це дозволяє створювати екосистеми, у яких мобільний застосунок взаємодіє з іншими сервісами – календарями, трекерами активності, корпоративними системами.

Важливою тенденцією є також орієнтація на безпеку та конфіденційність даних. Оскільки такі застосунки обробляють персональну інформацію (плани, звички, активність), необхідно забезпечити захист даних і контроль доступу до них.

Значну роль відіграє стандартизація та використання сучасних підходів до розробки, що дозволяє забезпечити сумісність компонентів і спростити інтеграцію систем.

Разом із тим, галузь стикається з рядом викликів. До них належать:

– складність розробки інтуїтивного інтерфейсу;

– необхідність обробки даних у реальному часі;

– забезпечення стабільності роботи застосунку;

– підтримка різних платформ і пристроїв.

Ще одним викликом є утримання користувача. Навіть функціонально потужний застосунок може бути неефективним, якщо він не мотивує користувача до регулярного використання.

Таким чином, незважаючи на активний розвиток і значні можливості, розробка мобільних застосунків для відстеження продуктивності залишається складним завданням, що потребує комплексного підходу.

Зі зростанням попиту на цифрові інструменти саморозвитку розробники змушені приділяти більше уваги UX/UI, аналітиці та персоналізації. Хоча створення таких застосунків потребує значних ресурсів, ці інвестиції виправдані підвищенням ефективності користувачів.

У сукупності це формує галузь як один із перспективних напрямів розвитку інформаційних технологій.

Сучасний ринок мобільних застосунків продуктивності характеризується високою конкуренцією, де ключовими факторами є не лише функціональність, але й зручність, швидкодія та адаптивність.

Таким чином, розробка мобільних застосунків для відстеження продуктивності є важливою складовою цифрової трансформації, спрямованою на підвищення ефективності діяльності людини.

1.2 Аналіз існуючих мобільних застосунків для відстеження продуктивності

Існуючі мобільні застосунки для відстеження продуктивності відіграють ключову роль у формуванні сучасних підходів до управління особистою ефективністю, оскільки забезпечують автоматизацію процесів планування, моніторингу активності та аналізу виконання задач. У контексті розробки таких систем вони виступають базовою основою для реалізації функціоналу, спрямованого на підвищення продуктивності користувача, оптимізацію використання часу та формування ефективних поведінкових моделей.[1]

З огляду на зростання кількості задач у повсякденному житті та необхідність ефективного управління часом, використання спеціалізованих мобільних застосунків стає необхідною умовою підвищення особистої продуктивності. Такі системи дозволяють інтегрувати різні аспекти діяльності користувача – планування, виконання, аналіз – в єдину інформаційну структуру.

Сучасні платформи для управління задачами забезпечують реалізацію функцій створення, редагування та контролю виконання задач, а також підтримують встановлення пріоритетів, дедлайнів і нагадувань. Вони дозволяють автоматизувати процес організації діяльності користувача та забезпечують гнучкість налаштування відповідно до індивідуальних потреб.

У свою чергу, застосунки з розширеними аналітичними можливостями дозволяють здійснювати збір і аналіз даних про активність користувача. Це включає оцінку витраченого часу, аналіз продуктивності протягом дня, тижня або місяця, а також формування візуальних звітів. Отримані дані можуть

використовуватися для оптимізації планування, підвищення ефективності виконання задач і формування нових звичок.

Окрему категорію становлять застосунки, що використовують елементи гейміфікації. Вони спрямовані на підвищення мотивації користувача шляхом впровадження систем досягнень, балів, рівнів або віртуальних винагород. Такий підхід дозволяє підвищити рівень залученості користувача та сприяє формуванню регулярності використання застосунку.

Крім того, важливим аспектом є інтеграція мобільних застосунків із зовнішніми сервісами, такими як календарі, нотатки або корпоративні системи управління задачами. Це дозволяє створювати єдине інформаційне середовище та забезпечує безперервність роботи користувача на різних пристроях.

Разом із тим, впровадження мобільних застосунків для відстеження продуктивності супроводжується рядом викликів. До них належать складність розробки інтуїтивного інтерфейсу, необхідність забезпечення високої швидкодії, а також потреба у створенні ефективних алгоритмів обробки даних. Окремою проблемою є забезпечення персоналізації, оскільки різні користувачі мають різні підходи до планування та виконання задач.

Таким чином, мобільні застосунки для відстеження продуктивності є важливим інструментом підвищення ефективності діяльності користувача. Їх використання дозволяє систематизувати задачі, оптимізувати використання часу та формувати ефективні поведінкові стратегії.

Таблиця 1. Найпопулярніші сервіси та їх плюси

Назва системи	**Функціо-нальність**	**Призначення**	**Вартість**	**Локалізація**	**Недоліки**
Todoist	Управління задачами, нагадування	Особиста продуктивність	Частково безкоштовно	Часткова	Обмежена аналітика
Notion	Планування, нотатки, бази даних	Комплексне управління	Частково безкоштовно	Часткова	Складність освоєння

Назва системи	**Функціо-нальність**	**Призначення**	**Вартість**	**Локалізація**	**Недоліки**
TickTick	Тайм-менеджмент, трекінг задач	Щоденне планування	Частково безкоштовно	Часткова	Частина функцій платні
Habitica	Гейміфікація продуктивності	Формування звичок	Безкоштовно	Часткова	Не підходить для всіх

Серед ключових труднощів у використанні таких застосунків варто виділити залежність від користувацької дисципліни. Навіть найбільш функціональні системи не гарантують підвищення продуктивності без регулярного використання та відповідального підходу до планування.

Крім того, технічні недоліки, такі як затримки в роботі, помилки синхронізації або некоректне збереження даних, можуть негативно впливати на ефективність застосунку та викликати втрату довіри користувача.

Окрему увагу слід приділити питанням безпеки, оскільки такі застосунки зберігають персональні дані користувача, включаючи плани, задачі та поведінкові патерни. Порушення безпеки може призвести до витоку інформації, що робить необхідним використання сучасних методів захисту даних.

Разом із тим, розвиток технологій відкриває нові можливості для вдосконалення таких систем. Зокрема, використання штучного інтелекту дозволяє автоматизувати процес планування, аналізувати продуктивність і формувати персоналізовані рекомендації.

Технології хмарних сервісів сприяють синхронізації даних між пристроями та забезпечують доступ до інформації з будь-якого місця. Це є особливо важливим для користувачів, які працюють у різних середовищах або використовують кілька пристроїв.

Перспективним напрямом є також використання аналітики великих даних, що дозволяє глибше досліджувати поведінку користувачів і вдосконалювати функціональність застосунків.

Таким чином, аналіз існуючих мобільних застосунків показує, що вони є важливою складовою сучасних цифрових інструментів продуктивності. Водночас їх ефективне використання потребує врахування технічних, поведінкових і безпекових аспектів. Комплексний підхід до розробки таких систем дозволяє створювати ефективні, зручні та адаптивні рішення.

1.3 Вимоги до мобільного застосунку для відстеження продуктивності

Мобільний застосунок для відстеження продуктивності повинен відповідати сукупності функціональних і нефункціональних вимог, що забезпечують ефективне, стабільне та безпечне виконання поставлених задач у різних умовах використання. Однією з ключових вимог є забезпечення коректної взаємодії між користувачем і програмною системою. Це передбачає інтуїтивно зрозумілий інтерфейс, логічну структуру навігації, а також швидкий доступ до основних функцій застосунку. Від цього безпосередньо залежить ефективність використання системи та рівень залученості користувача.

Важливим аспектом є адаптивність застосунку до різних сценаріїв використання. Система повинна забезпечувати стабільну роботу незалежно від рівня навантаження, типу пристрою чи умов використання. Наприклад, користувач може взаємодіяти із застосунком у режимі багатозадачності, в умовах обмеженого доступу до мережі або під час швидкого додавання інформації. У таких випадках застосунок повинен гарантувати коректне збереження даних і відсутність затримок.[3]

Оптимізація процесів обробки інформації є ще однією критичною вимогою. Система повинна забезпечувати мінімальні затримки при виконанні операцій, таких як створення задач, оновлення статусів або формування аналітичних звітів. Це вимагає використання ефективних алгоритмів обробки даних, оптимізованої архітектури та сучасних технологій розробки.

Додатково застосунок повинен підтримувати можливість гнучкого налаштування параметрів роботи. Це включає зміну інтерфейсу, налаштування нагадувань, вибір режимів планування, а також персоналізацію аналітичних

показників. Такий підхід дозволяє адаптувати систему до індивідуальних потреб користувача без необхідності змін у програмному коді.

Ще однією важливою вимогою є наявність механізмів моніторингу та аналізу активності користувача. Застосунок повинен забезпечувати збір даних про виконання задач, витрачений час і рівень продуктивності, а також формування звітів і візуалізацію результатів. Це дозволяє користувачу оцінювати свою ефективність і приймати обґрунтовані рішення щодо оптимізації діяльності.

Значну роль відіграє забезпечення безпеки даних. Оскільки застосунок працює з персональною інформацією, необхідно передбачити механізми захисту від несанкціонованого доступу, шифрування даних та резервного копіювання. Це дозволяє забезпечити конфіденційність і цілісність інформації користувача.

Мобільний застосунок повинен бути адаптованим до інтеграції з іншими цифровими сервісами. Це передбачає підтримку синхронізації з календарями, нотатками, хмарними сховищами та іншими платформами. Такий підхід забезпечує розширення функціональності та створення єдиного інформаційного середовища.

Для забезпечення ефективної роботи в умовах зростаючого навантаження система повинна мати масштабовану архітектуру. Це означає можливість додавання нових функцій, розширення обсягу даних і інтеграції додаткових модулів без значного впливу на продуктивність. Важливим є також забезпечення енергоефективності, особливо для мобільних пристроїв.

Окрему увагу слід приділити зручності взаємодії з користувачем. Інтерфейс повинен бути інтуїтивним, мінімалістичним і водночас функціональним, забезпечуючи швидке виконання основних дій. Відображення ключових показників продуктивності у вигляді графіків, діаграм і статистики сприяє кращому розумінню власної ефективності.

Важливою функцією застосунку є також збір і аналіз історичних даних. Формування довгострокових звітів дозволяє виявляти тенденції, оцінювати

динаміку продуктивності та визначати слабкі місця. Це створює основу для постійного вдосконалення системи та підвищення її ефективності.

Таким чином, мобільний застосунок для відстеження продуктивності повинен відповідати широкому спектру вимог, що охоплюють технічні, функціональні та безпекові аспекти. Лише комплексний підхід до їх реалізації дозволяє створити надійну, ефективну та зручну систему, здатну забезпечити підвищення продуктивності користувача.

РОЗДІЛ 2. АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вибір мови програмування та фреймворків

Розробка мобільного застосунку для відстеження власної продуктивності передбачає створення ефективного, надійного та продуктивного програмного забезпечення, яке забезпечує обробку даних користувача, управління задачами, формування аналітики та взаємодію з інтерфейсом у режимі реального часу. Така система повинна гарантувати швидке реагування на дії користувача, стабільну роботу застосунку та коректне збереження інформації незалежно від умов використання.

Програмна складова мобільного застосунку дозволяє автоматизувати ключові процеси, зокрема:

- створення, редагування та видалення задач;
- планування діяльності з урахуванням дедлайнів і пріоритетів;
- збір і обробку даних про активність користувача;
- аналіз продуктивності та формування звітів;
- візуалізацію результатів у зручному форматі;
- синхронізацію даних між пристроями.

Одним із найважливіших етапів розробки є вибір мови програмування, бібліотек і фреймворків, що безпосередньо впливає на:

- швидкість розробки та підтримки застосунку;
- стабільність і надійність роботи системи;
- продуктивність обробки даних;
- можливість інтеграції з іншими сервісами;
- масштабованість і гнучкість архітектури.

На відміну від класичних інформаційних систем, мобільний застосунок для продуктивності повинен:

- забезпечувати швидку реакцію інтерфейсу без затримок;
- підтримувати роботу в офлайн-режимі;
- гарантувати цілісність даних при збої або перезапуску;

• забезпечувати зручність взаємодії для користувача;

• мати механізми обробки помилок і відновлення даних.

У межах даного розділу розглядаються сучасні мови програмування та технології, що використовуються для розробки мобільних застосунків, з урахуванням їх продуктивності, зручності використання та можливостей інтеграції.

Однією з ключових мов програмування є Python, який активно використовується для реалізації серверної логіки, аналітики даних і обробки інформації. Завдяки простому синтаксису та великій кількості бібліотек Python є ефективним інструментом для створення алгоритмів аналізу продуктивності, обробки статистики та формування рекомендацій.

Ще однією важливою технологією є JavaScript, який широко застосовується для створення інтерфейсів мобільних застосунків. У поєднанні з такими фреймворками, як React Native або Vue Native, він дозволяє створювати кросплатформені рішення з високим рівнем продуктивності та зручності використання.

Для організації серверної частини та обміну даними часто використовується Node.js. Його асинхронна архітектура забезпечує ефективну обробку запитів і дозволяє організувати взаємодію між мобільним застосунком і сервером у реальному часі.

У випадках, коли необхідна висока надійність і масштабованість, можуть використовуватися такі технології, як Java або C#. Вони застосовуються для створення серверних компонентів, API та систем збереження даних.

Для розробки інтерфейсу користувача використовуються сучасні підходи до UX/UI-дизайну, що реалізуються за допомогою JavaScript-фреймворків. Це дозволяє створювати зручні, адаптивні та візуально привабливі інтерфейси.

Окрему роль відіграють інструменти аналітики та обробки даних, які дозволяють реалізувати функціонал відстеження продуктивності. Це можуть бути бібліотеки для роботи з даними, статистичні модулі або інструменти машинного навчання.[7]

Таким чином, вибір мови програмування та фреймворків залежить від вимог до функціональності, продуктивності та масштабованості застосунку.

Оптимальний технологічний стек може включати:

1. Python – для обробки даних і аналітики;
2. JavaScript (React Native) – для мобільного інтерфейсу;
3. Node.js – для серверної частини;
4. SQLite / PostgreSQL – для збереження даних;
5. API – для інтеграції з зовнішніми сервісами.

Отже, правильний вибір технологічного стеку є критично важливим для створення ефективного мобільного застосунку для відстеження продуктивності, оскільки він визначає швидкодію, стабільність і зручність використання системи.

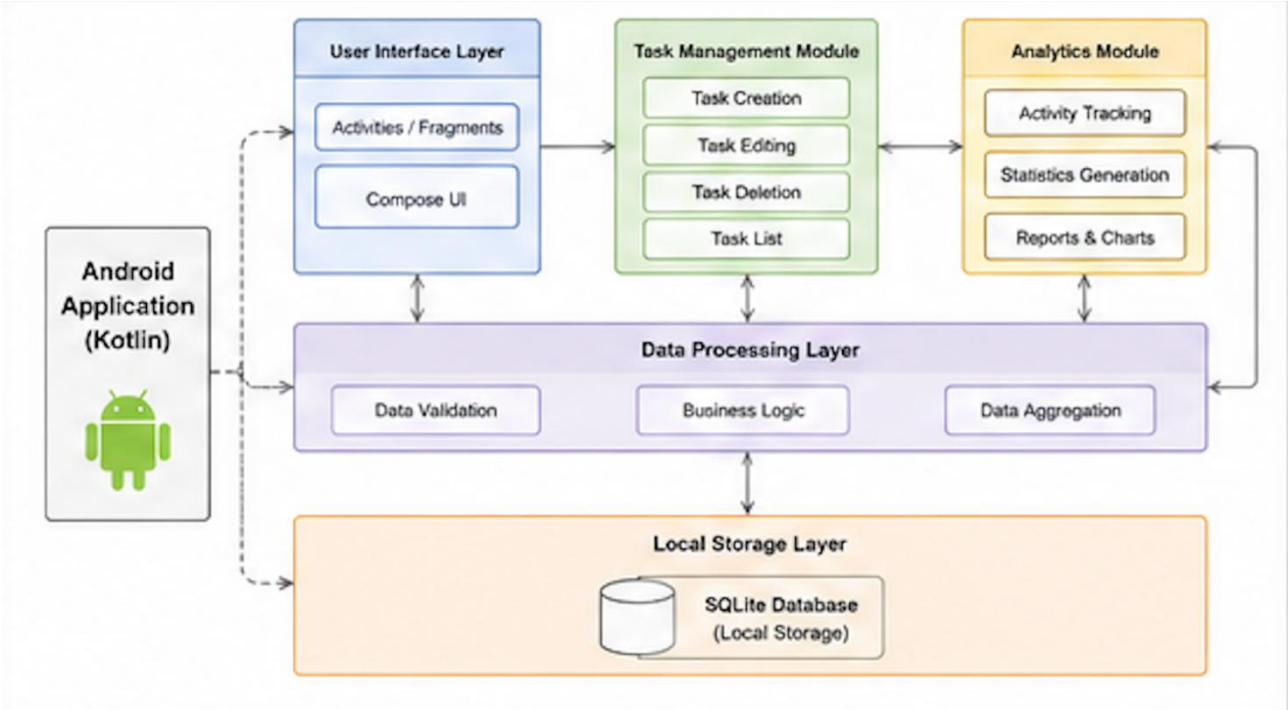


Рис.2.1. Загальна архітектура мобільного застосунку

2.2 Мова програмування Kotlin

Kotlin є сучасною статично типізованою мовою програмування, яка широко використовується для розробки мобільних застосунків, зокрема на платформі Android [9, 10]. Вона була розроблена компанією JetBrains та офіційно підтримується Google [11] як основна мова для створення Android-

застосунків. Завдяки своїй лаконічності, безпечності та високій продуктивності Kotlin став одним із ключових інструментів у розробці сучасних мобільних рішень.

Однією з головних переваг Kotlin є його сумісність із Java, що дозволяє використовувати існуючі бібліотеки та інфраструктуру Android без необхідності повного переписування коду. Крім того, Kotlin підтримує сучасні парадигми програмування, включаючи об’єктно-орієнтований та функціональний підходи, що забезпечує гнучкість у проєктуванні архітектури застосунку.

У контексті розробки мобільного застосунку для відстеження продуктивності Kotlin відіграє ключову роль у реалізації основної логіки системи, взаємодії з інтерфейсом користувача та обробки даних. Він дозволяє ефективно реалізувати такі функціональні можливості, як:

- створення та управління задачами;
- обробка подій користувача;
- реалізація системи нагадувань;
- збереження та оновлення даних;
- інтеграція з базами даних і зовнішніми сервісами.

Особливу роль у Kotlin відіграє підтримка корутин (coroutines), які дозволяють реалізувати асинхронну обробку даних без складних конструкцій. Це є критично важливим для мобільних застосунків, оскільки дозволяє виконувати фонові операції (наприклад, збереження даних або отримання інформації з мережі) без блокування інтерфейсу користувача.

Kotlin також забезпечує високий рівень безпеки коду завдяки вбудованим механізмам захисту від типових помилок, таких як null-pointer exception. Це дозволяє підвищити стабільність застосунку та зменшити кількість критичних збоїв під час роботи.

Для розробки інтерфейсу користувача Kotlin активно використовується разом із сучасними інструментами, такими як Jetpack Compose [12], що дозволяє створювати декларативні UI-компоненти. Це значно спрощує процес

розробки інтерфейсу та забезпечує його адаптивність до різних розмірів екранів і пристроїв.

Kotlin також добре інтегрується з архітектурними підходами, такими як MVVM (Model-View-ViewModel) [5, 29], що дозволяє розділити логіку застосунку на окремі компоненти. Це підвищує читабельність коду, спрощує тестування та забезпечує масштабованість системи.

У процесі розробки мобільного застосунку для відстеження продуктивності Kotlin може використовуватися для реалізації таких функцій:

• обробка взаємодії користувача з інтерфейсом;

• управління станом застосунку;

• робота з локальною базою даних;

• реалізація логіки аналітики продуктивності;

• інтеграція з API та хмарними сервісами.

Завдяки підтримці сучасних бібліотек і фреймворків Kotlin дозволяє реалізувати складні сценарії обробки даних, включаючи аналіз активності користувача, формування статистики та оптимізацію процесів планування.

Крім того, Kotlin забезпечує швидке розгортання застосунку та зручну підтримку, що є важливим фактором у процесі розробки. Його використання дозволяє скоротити обсяг коду, зменшити кількість помилок і підвищити загальну ефективність розробки.

Середовище розробки, яке зазвичай використовується для роботи з Kotlin, забезпечує повний набір інструментів для створення, тестування та налагодження застосунків. Це дозволяє ефективно контролювати процес розробки та забезпечувати високу якість програмного продукту.

Таким чином, використання Kotlin у розробці мобільного застосунку для відстеження продуктивності є доцільним і обґрунтованим рішенням. Він забезпечує високу продуктивність, гнучкість архітектури та зручність розробки, що є критично важливим для створення сучасного мобільного застосунку.

2.3 SQLite

SQLite – це вбудована реляційна система керування базами даних, яка широко застосовується у розробці мобільних застосунків [13], зокрема для платформ Android та iOS. Вона є легким, компактним і ефективним рішенням для зберігання структурованих даних без необхідності розгортання окремого серверного середовища. Основною особливістю SQLite є її інтеграція безпосередньо у програмний код застосунку, що дозволяє значно спростити архітектуру системи та забезпечити автономність роботи.

У контексті розробки мобільного застосунку для відстеження власної продуктивності SQLite відіграє ключову роль як основний інструмент для локального збереження даних користувача. Застосунок повинен зберігати значний обсяг інформації, пов’язаної з активністю користувача, виконаними завданнями, часовими витратами та результатами аналізу продуктивності. Використання SQLite дозволяє ефективно організувати цей процес без залежності від постійного підключення до Інтернету.

Однією з основних переваг SQLite є висока швидкість виконання операцій читання та запису даних у межах локального середовища. Це є критично важливим для мобільного застосунку, оскільки забезпечує миттєвий доступ до інформації, швидке оновлення списків завдань та оперативне відображення статистики продуктивності користувача.

SQLite дозволяє реалізувати структуроване збереження даних із використанням таблиць, що включають записи про:

- список задач користувача;
- статус виконання (виконано, в процесі, відкладено);
- часові мітки (дата створення, дедлайн, час виконання);
- категорії або пріоритети задач;
- показники продуктивності (кількість виконаних задач, тривалість роботи, ефективність);
- історію активності користувача.

Завдяки підтримці SQL-запитів SQLite забезпечує можливість гнучкої вибірки даних, що дозволяє реалізувати функціонал аналітики продуктивності.

Наприклад, застосунок може формувати звіти про виконані задачі за певний період, визначати найбільш продуктивні дні або аналізувати динаміку ефективності користувача.

Інтеграція SQLite з мовою програмування Kotlin здійснюється за допомогою стандартних бібліотек Android або через сучасні абстракції, такі як Room. Це дозволяє спростити роботу з базою даних, зменшити кількість помилок і підвищити якість програмного коду.

Особливістю SQLite є її автономність, що дозволяє застосунку функціонувати навіть без доступу до мережі. Це важливо для користувачів, які можуть працювати з додатком у будь-яких умовах, зокрема без стабільного Інтернет-з'єднання. Усі дані зберігаються локально на пристрої та можуть бути синхронізовані з хмарними сервісами за потреби.

SQLite також підтримує транзакції, що забезпечує цілісність даних навіть у разі збоїв або переривання роботи застосунку. Це є важливим аспектом для систем, які працюють із персональними даними користувача та повинні гарантувати їх збереження.

Таблиця 2.1

Переваги використання SQLite у мобільному застосунку

Критерій	**SQLite**
Тип бази даних	Вбудована, файлова
Потреба в сервері	Відсутня
Швидкість роботи	Висока для локальних операцій
Налаштування	Мінімальне
Підтримка транзакцій	Так
Сфера застосування	Мобільні застосунки, автономні системи, персональні трекери

Важливою перевагою SQLite є її компактність і низькі вимоги до ресурсів пристрою. Це дозволяє використовувати її навіть на смартфонах із обмеженими

технічними характеристиками без негативного впливу на продуктивність застосунку.

SQLite може використовуватися для збереження таких типів даних:

• інформація про задачі та підзадачі;

• статистика продуктивності користувача;

• журнали активності;

• налаштування застосунку;

• результати аналітики та рекомендації.

Завдяки можливості швидкої обробки даних SQLite дозволяє реалізувати інтерактивні елементи застосунку, такі як дашборди продуктивності, графіки виконання задач і аналітичні панелі. Це підвищує залученість користувача та сприяє більш ефективному управлінню часом.

Хоча SQLite не призначена для високонавантажених багатокористувацьких систем, вона повністю відповідає вимогам персонального мобільного застосунку. У разі необхідності масштабування система може бути доповнена віддаленою базою даних або хмарною синхронізацією.

Таким чином, використання SQLite у мобільному застосунку для відстеження власної продуктивності є ефективним рішенням, що забезпечує швидкий доступ до даних, автономність роботи, простоту інтеграції та

достатню продуктивність для реалізації всіх ключових функцій застосунку.

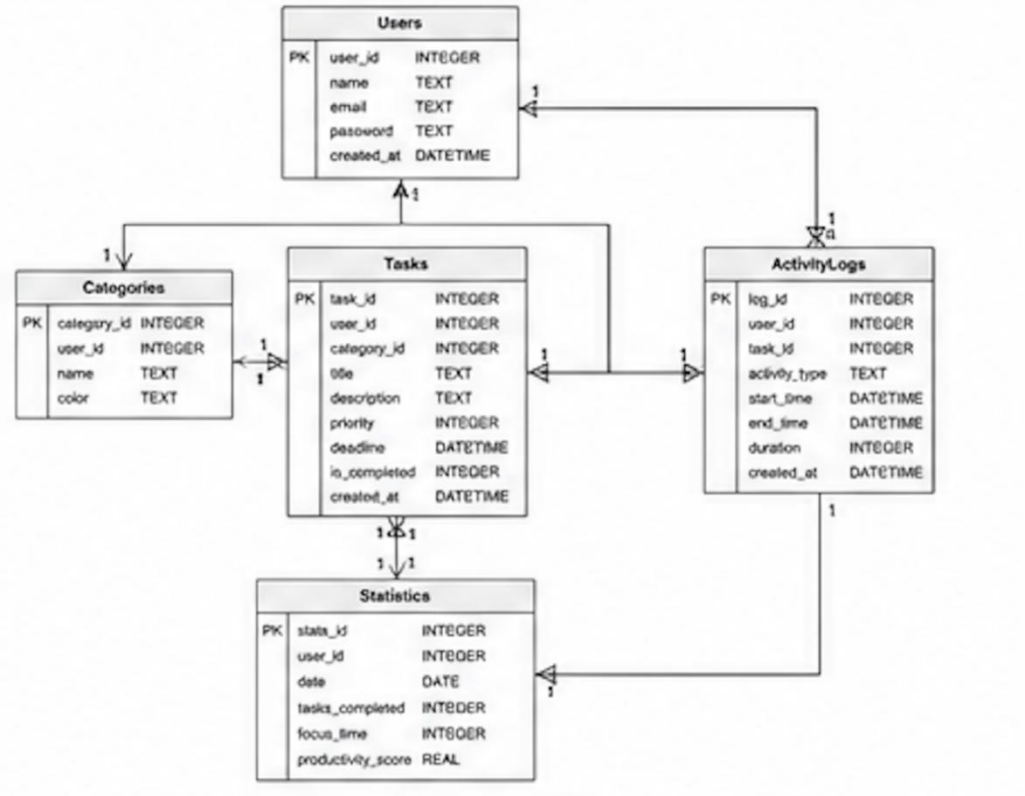


Рис.2.2. ER-діаграма бази даних

2.4 Вибір середовища розробки

Вибір середовища розробки є важливим етапом у створенні мобільного застосунку для відстеження власної продуктивності, оскільки саме від нього залежить ефективність процесу програмування, тестування та подальшої підтримки програмного продукту. Сучасні інтегровані середовища розробки (IDE) забезпечують розробнику широкий набір інструментів, що дозволяють прискорити створення застосунку, зменшити кількість помилок та підвищити загальну якість коду.

Для реалізації мобільного застосунку доцільно використовувати середовище розробки Android Studio [11, 30], яке є офіційним інструментом для створення Android-додатків і підтримує мову програмування Kotlin. Android Studio надає повний набір функцій для розробки, включаючи редагування коду, компіляцію, тестування та налагодження застосунків.

Однією з ключових переваг Android Studio є інтелектуальна система автодоповнення коду, яка дозволяє прискорити процес розробки та зменшити

кількість синтаксичних і логічних помилок. Це особливо важливо при створенні складних логічних модулів, що відповідають за обробку даних продуктивності користувача, роботу з базою даних та формування аналітики.

Вбудовані інструменти налагодження дозволяють детально аналізувати роботу застосунку в реальному часі. Розробник може відстежувати виконання коду, перевіряти значення змінних, а також знаходити та виправляти помилки у функціоналі, що безпосередньо впливає на стабільність роботи системи відстеження продуктивності.

Android Studio також підтримує інтеграцію з системами контролю версій, такими як Git, що дозволяє ефективно керувати змінами у коді, працювати в команді та зберігати історію розробки проєкту. Це є важливим аспектом при створенні масштабованих мобільних застосунків.

Окремою перевагою середовища є підтримка емуляторів пристроїв, які дозволяють тестувати застосунок на різних конфігураціях смартфонів без необхідності використання фізичних пристроїв. Це забезпечує перевірку адаптивності інтерфейсу та стабільності роботи застосунку в різних умовах.

У контексті мобільного застосунку для відстеження продуктивності Android Studio дозволяє реалізувати такі ключові процеси:

- розробку інтерфейсу користувача (UI/UX);
- інтеграцію з локальною базою даних SQLite;
- реалізацію логіки обробки задач і тайм-менеджменту;
- створення системи аналітики продуктивності;
- тестування та оптимізацію роботи застосунку.

Завдяки підтримці сучасних фреймворків, зокрема Jetpack Compose, розробка інтерфейсу користувача стає більш гнучкою та ефективною. Jetpack Compose дозволяє створювати декларативні UI-компоненти, що спрощує структуру коду та підвищує його читабельність.

Також Android Studio підтримує інтеграцію з різними бібліотеками для аналітики, баз даних і візуалізації даних, що є важливим для реалізації функціоналу відстеження продуктивності. Це дозволяє створювати графіки,

статистику та дашборди, які допомагають користувачу аналізувати свою ефективність.

Підтримка Gradle як системи збірки забезпечує автоматизацію процесу компіляції та керування залежностями. Це дозволяє легко додавати нові бібліотеки, оновлювати компоненти та підтримувати стабільність проєкту.

Таким чином, використання Android Studio як основного середовища розробки для мобільного застосунку відстеження продуктивності є обґрунтованим і ефективним рішенням. Воно забезпечує повний цикл розробки, високу продуктивність роботи та можливість створення сучасного, масштабованого і стабільного мобільного застосунку.

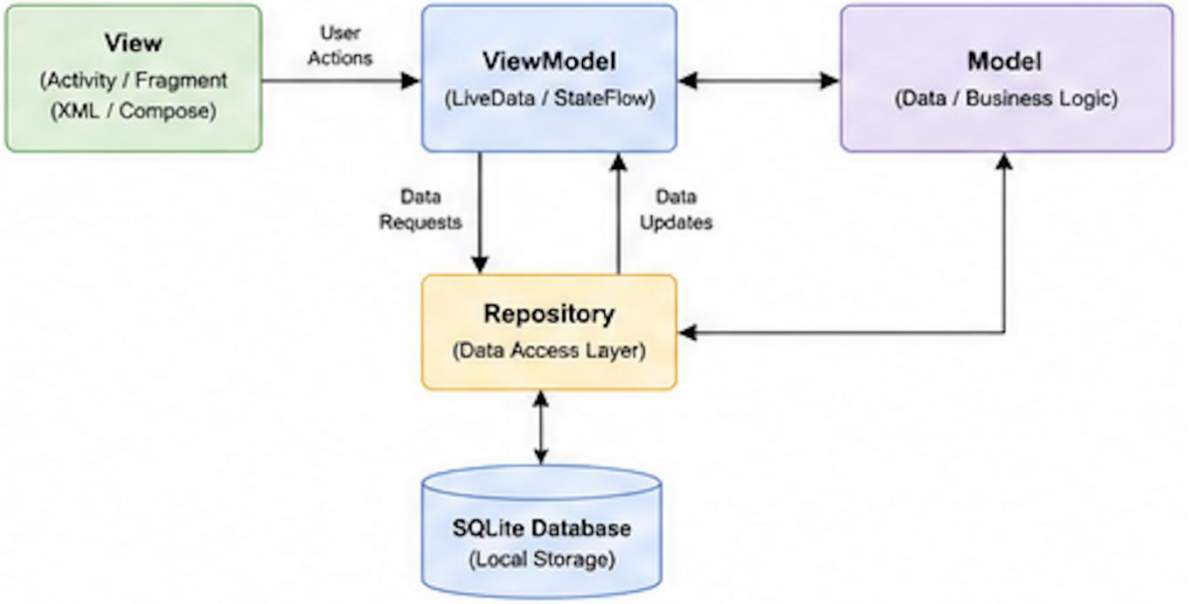


Рис.2.3. Архітектура MVVM

РОЗДІЛ 3. РОЗРОБКА ТА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

3.1 Архітектура мобільного застосунку

Програма, яку ми пропонуємо, включає в себе ергономічний дизайн, який не лише виглядає естетично привабливо, але й значно сприяє простоті навігації. Завдяки добре продуманому та ретельно організованому інтерфейсу, користувачі можуть швидко та без проблем знайти всю необхідну інформацію або функціонал, що їх цікавить.

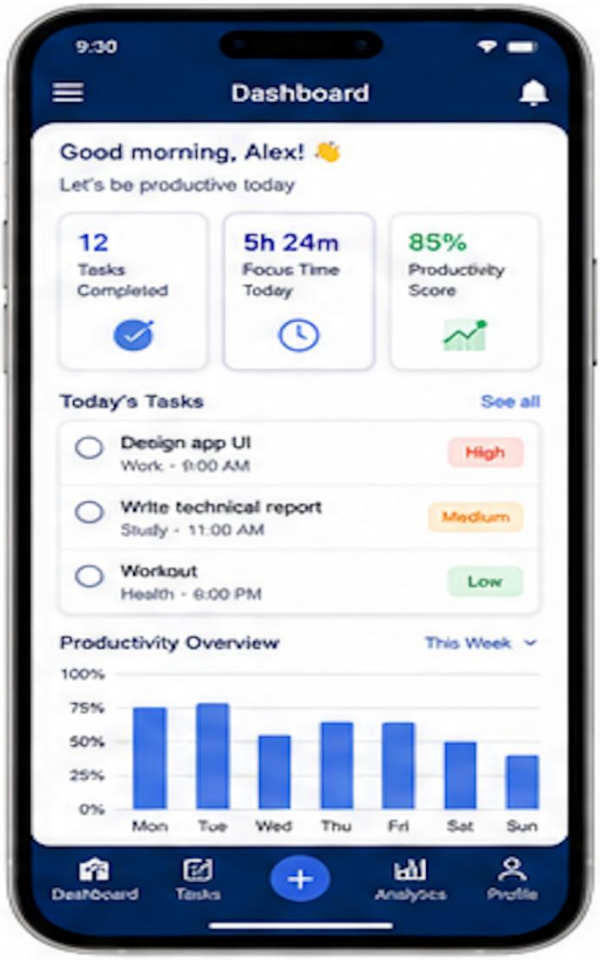


Рис.3.1. Інтерфейс головного екрану

Увага до дрібних, але надзвичайно важливих нюансів в дизайні, таких як кольорова гамма, вибір шрифтів та оптимальне розташування ключових елементів інтерфейсу, забезпечує не лише естетичну візуальну привабливість, але й високу зручність користування. Важливо також, щоб користувачі мали можливість налаштовувати інтерфейс відповідно до своїх особистих уподобань і вимог, що значно підвищує їх загальну задоволеність від використання

програми. Вони прагнуть, щоб цей інструмент був не лише функціональним і практичним, а також приємним для повсякденного використання та ефективної роботи.

Для підвищення продуктивності є надзвичайно важливим забезпечення інтеграції з різноманітними зовнішніми сервісами та платформами, такими як соціальні мережі, електронна пошта, файлообмінники і різноманітні API. Це значно розширює загальні можливості застосунку, роблячи його ще більш зручним і універсальним. Споживачі отримують можливість мати легкий і швидкий доступ до інструментів, які вони уже використовують у повсякденному житті, що істотно спрощує обмін інформацією між різними платформами, що є безперечно зручним і практичним. Зокрема, можливість синхронізації даних у реальному режимі з іншими пристроями, такими як смартфони, планшети або комп'ютери, додає зручності, дозволяючи безперервно слідкувати за змінами у продуктивності, планах, завданнях та інших важливих параметрах і показниках. Це, в свою чергу, робить процес роботи більш систематизованим, організованим і ефективним, що, безумовно, підвищує зручність у використанні програми. Важливим аспектом нашої програми також є наявність різноманітних навчальних ресурсів, які безпосередньо допомагають користувачам отримати глибоке уявлення про те, як максимально ефективно використовувати весь функціонал застосунку. Це може бути у формі відеоуроків, різноманітної документації, інтерактивних підказок або ж навіть вебінарів, які заохочують користувачів до активного використання всіх можливостей програми.

Як результат, користувачі можуть набагато швидше освоювати інтерфейс та функціонал програми, що зрештою веде до суттєвого підвищення загальної задоволеності від роботи з продуктом. Вони отримують не тільки ефективний інструмент, але й супутні знання для його корисного та продуктивного використання.

Тестування та відстеження зворотного зв'язку користувачів є надзвичайно важливими елементами, які сприяють безперервному вдосконаленню програми.

Залучення потенційних користувачів до процесу розробки через різноманітні опитування, питання, фокус-групи та дочасні версії продукту допомагає виявити проблеми та потреби, які могли бути упущені під час початкового етапу створення програми.

Ця двостороння комунікація між розробниками та користувачами забезпечує формування відчуття спільноти, команди, що, в свою чергу, підвищує лояльність до продукту та зацікавленість у його розвитку на всіх етапах. Крім того, у сучасному мобільному застосунку інтегруються новітні рішення для адаптації до різних платформ, типів пристроїв та умов використання, що робить його надзвичайно універсальним.

Це передбачає не лише адаптацію дизайну до різних розмірів екранів, але й оптимізацію під різні типи зв'язків, такі як Wi-Fi, 4G, 5G, що забезпечує стабільну та швидку роботу програми в умовах різноманітних мереж. Це, безумовно, є критично важливим для користувачів, які потребують надійного доступу до інформації в будь-який час та в будь-якому місці, без зайвих затримок або труднощів. Не менш важливим є врахування аспектів екологічної відповідальності та усвідомленого широкого впливу на навколишнє середовище.

Наприклад, розробка може включати використання енергозберігаючих технологій, що сприяє суттєвому зменшенню споживання енергії під час роботи програми. Це стає особливо актуальним у сучасному світі технологій, де питання екології, сталого розвитку та енергоефективності стають дедалі більш значущими в суспільстві, яке усвідомлює свої обов'язки перед природою та навколишнім середовищем.

Наприкінці, основна мета полягає у створенні інноваційного мобільного застосунку, що не лише відповідає вимогам сучасного ринку та технологій, але й є корисним, зручним інструментом, який істотно полегшує повсякденне життя користувачів. Це допомагає їм досягати своїх цілей більш ефективно, швидко і простіше, що в свою чергу забезпечує позитивний досвід

використання. Все це створює атмосферу повного комфорту та задоволення від використання програми щодня, вносячи важливі зміни у їхнє життя.

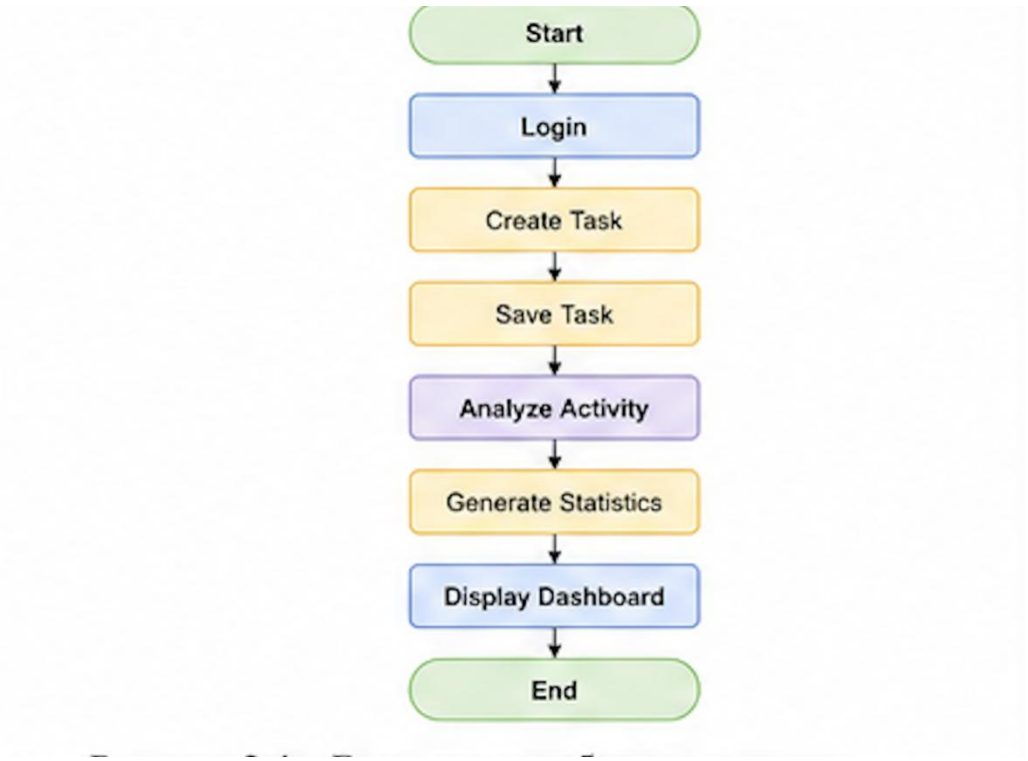


Рис.3.2. Блок-схема роботи застосунку

3.2 Системні вимоги та взаємодія з пристроєм

Системні вимоги та взаємодія з пристроєм є критично важливими складовими, які забезпечують стабільну, надійну та ефективну роботу будь-якого сучасного мобільного застосунку в умовах нашого сьогоднішнього технологічного світу, що постійно змінюється та еволюціонує. Цей динамічний світ характеризується надзвичайно швидко зростаючою конкуренцією, а також величезним збільшенням вимог до швидкості, надійності і загальної якості користувацького враження, яке користувачі очікують від своїх мобільних пристроїв. Особливо це стосується постійно зростаючих потреб користувачів у функціональності, зручності, а також інтуїтивно зрозумілому інтерфейсі, які в свою чергу ставлять нові задачі перед розробниками, оскільки адаптація до цих вимог є критично важливою для успіху продукту.

Тому ретельний вибір відповідних технічних специфікацій, а також постійна підтримка сумісних апаратних ресурсів мають вирішальне значення для розвитку і підтримки сучасного програмного забезпечення, що

безпосередньо впливає на можливість програми залишатися релевантною в конкурентному середовищі. Це дозволяє розробникам точно й об'єктивно збирати актуальні, новітні й достовірні дані про загальну продуктивність користувача, що неабияк впливає на загальну якість роботи програми та її зручність для користувачів, що сприяє створенню позитивного іміджу застосунку на ринку.

Додаток повинен бути максимально адаптований під різноманітні запити й потреби цільової аудиторії, адже це стало однією з основних умов для досягнення успіху продукту на сучасному динамічному ринку, де переможець отримує все, а ті, хто не зможе адаптуватися, ризикують залишитися позаду.

Основними технічними вимогами, які варто уважно розглянути та ретельно вивчити під час процесу розробки, початкового запуску або встановлення будь-якого застосунку, є операційна система (такі як Android чи iOS), версії цих операційних систем, а також мінімальні специфікації апаратних характеристик для оптимізації роботи. Ці характеристики обов'язково відображаються не лише в потужності процесора, але й в обсязі оперативної пам'яті, розмірі внутрішнього сховища, а також в інших значущих параметрах, які можуть суттєво визначити успішність роботи програми і її функціональність. Це, безумовно, гарантує, що розроблений додаток зможе функціонувати безперервно й стабільно в різних технічних умовах і ситуаціях, а також адміністративно адаптуватися до потреб користувачів, які використовують різноманітні пристрої з різними можливостями.

Таким чином, це забезпечує максимальну доступність для величезного спектра користувачів, котрі використовують найрізноманітніші пристрої, незалежно від їхніх характеристик, можливостей і обмежень, як апаратних, так і програмних. Цей аспект взаємодії не можна недооцінювати, адже він відіграє критично важливу роль у формуванні враження про програму на ринку, особливо в умовах жорсткої конкурентної боротьби за увагу споживачів, яким важливо отримати кращий сервіс.

Взаємодія з пристроями ґрунтується на інтенсивному та продуктивному використанні сучасних API і сенсорів, що дозволяє не лише максимально розширити функціональність застосунків, але й забезпечити легкість адаптації до нових технологій. Це включає в себе прості у використанні акселерометри, гіроскопи, мікрофони та камери, які можуть бути використані для виконання та обробки різноманітних розрахунків і модифікацій, що покращують зручність роботи та упроваджують спеціалізовані функції програми, які спрощують взаємодію користувача з продуктом, роблячи її більш зрозумілою.

Дана технологія дозволяє розробленому застосунку автоматично, в реальному часі, збирати важливі дані про активність користувача, рівень його концентрації, а також інші суттєві параметри, які можуть мати значний вплив на ефективність і зручність використання додатка, підвищуючи його корисність.

Усі зібрані дані можуть використовуватися для персоналізації наданих послуг та покращення загального досвіду взаємодії користувача з програмою, що стає особливо актуальним на ринку, де кожен момент має значення.

У процесі функціонування даний додаток отримує доступ до численних ресурсів і можливостей, шляхом запиту відповідних дозволів у користувача, що є невід’ємною частиною сучасного програмного забезпечення, і сьогодні без належних дозволів важко уявити будь-який цифровий продукт. При цьому він ретельно дотримується специфікацій правил стосовно конфіденційності та безпеки особистих даних користувачів, адже це критично важливо у наш час, коли інформація стала основним активом у бізнесі і технології, і її збереження є пріоритетом.

Управління цими дозволами здійснюється через інтуїтивно зрозумілі та доступні налаштування, які представлені як у самому додатку, так і в системному інтерфейсі, без яких важко уявити нормальну роботу сучасних додатків, що зазвичай передбачає використання різноманітних функцій. Це значно спрощує технічну роботу кінцевого користувача та покращує загальне враження від взаємодії з програмою, роблячи її більш зрозумілою, зручною та

комфортною у використанні, що підвищує ймовірність повторного використання.

Важливо, щоб додаток не лише задовольняв потреби, але й пропонував користувачам приємний досвід, адже це складає основу для підвищення їхньої лояльності до бренду і готовності рекомендувати його іншим. Для досягнення високої точності збору даних і мінімального споживання енергії, використовується оптимізовані алгоритми для взаємодії з сенсорами, а також для систем визначення місцезнаходження, що може значно вплинути на загальну продуктивність і життєздатність програми.

Така увага до деталей є надзвичайно важливою для збереження заряду акумулятора, адже у умовах тривалої роботи в фоновому режимі важливо здійснювати постійний моніторинг і аналіз даних без значних витрат ресурсів. Це, в свою чергу, може суттєво вплинути на тривалість використання програми, адже споживач бажає, щоб його пристрій працював якомога довше, а знадобиться уникати швидкої розрядки пристрою, щоб не створювати неприємностей користувачам і не ускладнювати їхній досвід взаємодії з додатком.

Як наслідок, такі ситуації можуть вплинути на безперебійну роботу програми, і тому важливо враховувати усі ці нюанси під час розробки та оптимізації застосунків, щоб гарантовано забезпечити високу якість всіх послуг в умовах ринку, що постійно розвивається. Зрештою, системні вимоги та механізми взаємодії з пристроєм повинні бути чітко узгоджені та спроектовані між собою, адже від цього залежить стабільність, надійність та безперебійне функціонування програми, важливість якого не можна переоцінити для успішності в користувачів та їхньої прихильності до продукту, що є запорукою стабільного бізнесу. При цьому, під час розробки необхідно акуратно враховувати різноманітність апаратних платформ, системних налаштувань та можливих функцій.

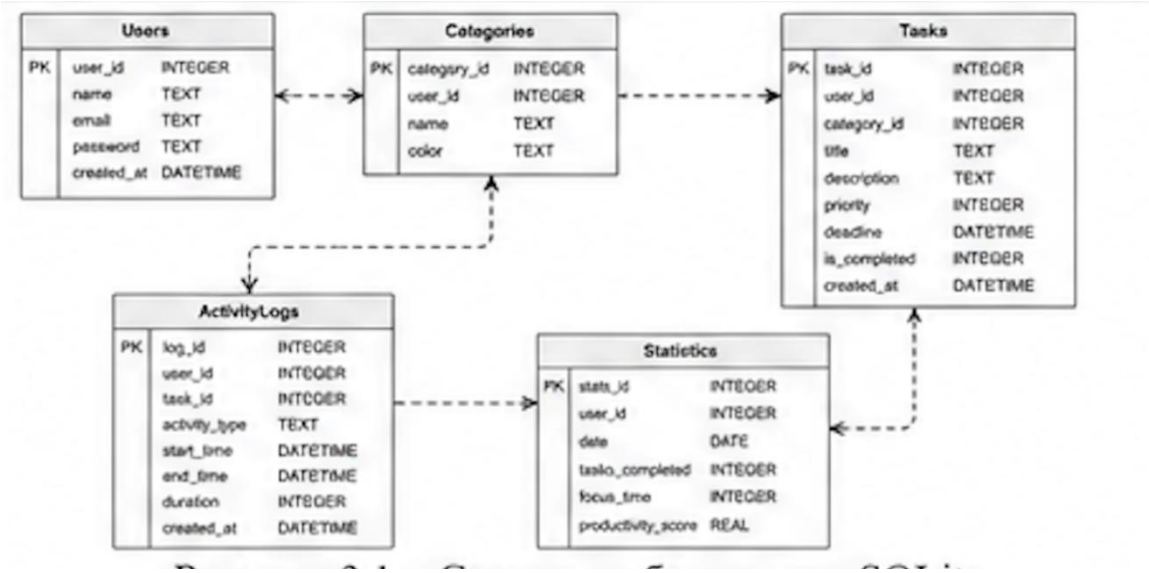


Рис.3.3. Структура бази даних SQLite

3.3 Модуль збору даних

Збереженням споживацького досвіду, що враховує багатогранність емоцій, мотивацій та прагнень кінцевих користувачів, відіграє вирішальну роль у створенні дійсно інноваційного продукту, який здатний змінити уявлення про якість обслуговування та інтерактивність у сучасному світі. Саме завдяки цим важливим аспектам стає можливим виготовлення й удосконалення продуктів, які можуть реально впливати на щоденний життєвий досвід споживачів, зокрема через покращення функціоналу та зручності використання.

З одного боку, модуль збору даних допомагає виявити не лише статистичні дані, але й глибше дослідити, чому користувачі здійснюють вибір на користь певних функцій, чому вони стикаються з труднощами, які напрямки їх вибору ведуть до конкретних дій, і яким чином ці вибори впливають на їхнє ставлення до продукту.

Вивчаючи поведінку та емоції, а також їхнє ставлення до мобільного середовища, ми можемо не лише виявити існуючі проблеми, але й спробувати спрогнозувати наступні кроки, які можуть бути реалізовані в межах даного застосунку. При цьому важливим етапом є формування в самій платформі можливостей для адаптації зовнішнього вигляду, функціоналу та навігації згідно з реальними потребами та побажаннями споживачів. Необхідність в

такий спосіб процесуального підходу полягає ще й у тому, що облік аспектів, які іноді можуть залишатися поза увагою, може мати критичне значення для формування лояльності до продукту.

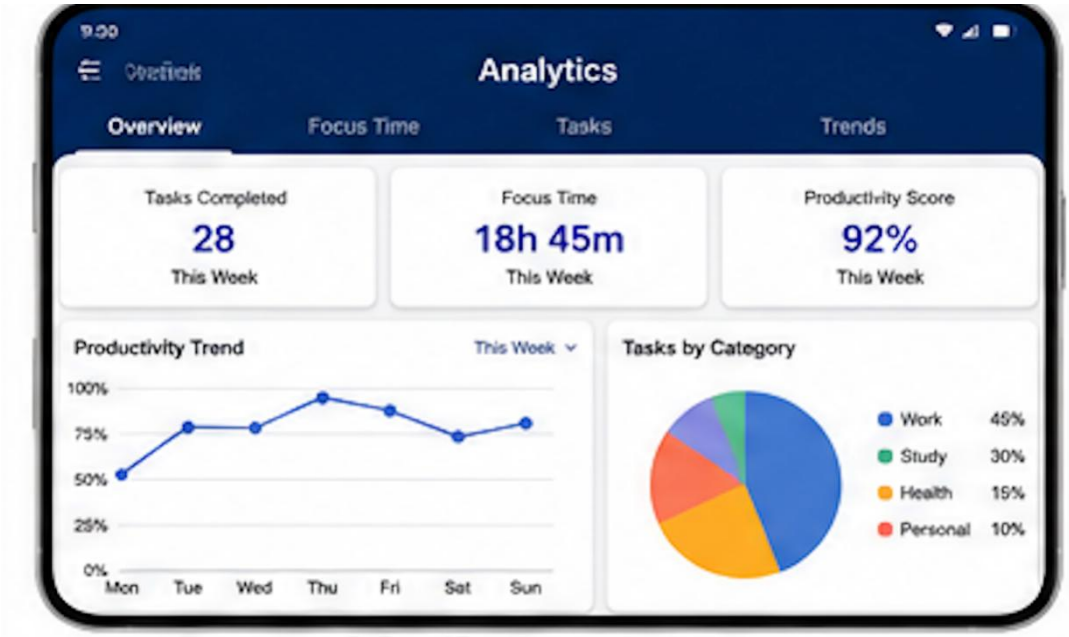


Рис.3.4. Панель аналітики

Тут і графічне оформлення, і зручність взаємодії з інтерфейсом, і навіть технічна підтримка – все це грає свою безпосередню ключову роль у загальному зображенні, яке створюється в уяві користувача, формуючи його довіру і бажання знову звертатися до продукту.

Системи, що надходять на ринок сучасних технологій, теж мають враховувати зростаючу розсудливість користувачів: їхню готовність перевіряти, наскільки нові технології виходять за межі раніше відомого, і на що їх можуть спонукати нові рішення. Адже клієнт завжди має залишатися в центрі уваги, і те, як він відчуває продукт, визначає перспективи для його подальшого розвитку та успіху.

Важливим етапом вдосконалення є інтеграція методів самооцінки, що дозволяють самим користувачам надсилати зворотний зв'язок на основі їхніх спостережень та зауважень щодо власного досвіду. Це суттєво сприяє формуванню активної спільноти, де кожен має можливість поділитися своїм власним розумінням, виклавши свої побоювання чи ідеї щодо поліпшення

характеристик і функцій застосунку. Таке спілкування зазвичай підвищує інтерес до продукту, адже користувач почувається залученим до процесу формування та вдосконалення самого додатку, відчуваючи особисту відповідальність за його якість. Це справді важливо, адже чим більше звернень буде до розробників, тим швидше й точніше можуть бути внесені необхідні зміни, які відповідатимуть потребам споживачів.

Технології, які сьогодні активно використовуються в зборі даних, також постійно вдосконалюються – від нейронних мереж до алгоритмів машинного навчання, які невпинно аналізують величезні масиви інформації, прагнучи формувати цінні інсайти, здатні значно підвищити не лише досвід користувачів, але й загальну ефективність функціонування самої програми.

Таким чином, ці зміни стають не кінцевою метою, а лише частиною еволюційного процесу постійного вдосконалення технологій, які сприяють створенню чіткого бачення і допомагають реалізувати новаторські ідеї, за яким дійсно стає можливим створення якісно нового продукту або послуги. Взаємодія між користувачами та платформою, з одного боку, і самою системою, з іншого, вважається постійно прогресуючим циклом зворотного зв’язку, що не лише формує задоволеність, але й спонукає користувачів активно ділитися своїми враженнями, заохочуючи інших долучатися до цього циклу інтерактивної взаємодії. Важливими залишаються питання безперервності в удосконаленні, адже сьогоднішні рішення стають завтрашніми перепустками до нових успіхів у світі технологій, який не стоїть на місці та постійно еволюціонує.

Модуль збору даних стає тією основою, на якій розвиваються всі подальші сегменти мобільного застосунку, тому його важливість не можна переоцінити.

Завершуючи аналіз, слід зазначити ненадійність рішень без належного та системного аналізу даних, адже в умовах постійно змінюваних потреб та інтересів користувачів системи повинні бути гнучкими та здатними миттєво реагувати на нові виклики, відводячи часу на адаптацію або перепроектування.

Це є особливо важливим в умовах сучасних реалій на ринку, де конкуренти не дрімають, і інновації стають ключовим фактором успіху.

Мобільний додаток має не просто задовольняти споживчі запити, але й стійко підвищувати обслуговування, щоб забезпечити не просто стосунки, а справжній зв'язок між брендом і його аудиторією, що є надзвичайно істотним для довготривалої перспективи розвитку, адже тільки так можна створити фундамент для стабільних відносин і зростаючої лояльності серед споживачів.

3.4 Модуль обробки та аналізу даних

Модуль обробки та аналізу даних має надзвичайно важливу роль у складі розвиненої системи, де його функції стають дійсно невід'ємними для забезпечення інтеграції, ретельного перетворення та глибокої інтерпретації всієї зібраної інформації. Ця інформація, що накопичується в процесі діяльності, може бути використана для витягування важливих, корисних та систематизованих висновків щодо продуктивності користувача. І, звичайно, ця інформація також надає можливість впливати на різні інші аспекти діяльності, які є критично важливими для досягнення успіху.

Основною функцією даного модуля є обробка великих обсягів даних, які були ретельно зібрані й перетворені за допомогою попереднього модуля збору інформації. Ця обробка має на меті формування цінних аналітичних результатів, що можуть бути надзвичайно важливими для подальшого розвитку користувача, а також підвищення загальної ефективності його роботи.

Для досягнення цієї важливої мети реалізуються найсучасніші алгоритми фільтрації, агрегації та класифікації даних. Ці алгоритми дозволяють ефективно очищати, структурувати та класифікувати інформацію, враховуючи її характер, специфіку та цілі проведених досліджень. В результаті цього складного, проте необхідного процесу забезпечується відповідність отриманих даних строгим вимогам дослідження, а також здатність точно виявляти важливі інсайти, що можна вважати суттєвим досягненням у сфері аналізу даних за допомогою новітніх технологій.

Крім цього, активно задіюються методи статистичного аналізу, які сприяють виявленню ключових тенденцій, закономірностей і патернів у поведінці користувача. Це надає можливість не лише відстежувати наявні сегменти з підвищеною продуктивністю, але й здійснювати точне прогнозування можливих змін у його режимі роботи. Це прогнозування спирається на отримані дані та аналітичні моделі, що дозволяє бути на крок попереду від вимог і потреб.

Для підвищення якості та точності аналізу розроблені інноваційні алгоритми машинного навчання, які здатні автоматично класифікувати різні види активності та виявляти рівень продуктивності в реальному часі. Все це створює можливість отримати більш детальну та об’єктивну картину використання часу та ресурсів, що є надзвичайно важливим для оптимізації робочих процесів у різних сферах діяльності. Адже це сприяє створенню більш ефективних робочих умов, що особливо цінно в сучасному часі.

Важливою складовою цього належного процесу є побудова адаптивних моделей, які враховують індивідуальні особливості кожного користувача. Такий підхід суттєво підвищує точність аналізу і відкриває широкі можливості для створення персоналізованих рекомендацій. Ці індивідуалізовані рекомендації стають неймовірно корисними в повсякденному житті та різноманітній професійній діяльності, допомагаючи розвивати індивідуальні навички та підвищувати рівень продуктивності.

Обробка даних завжди здійснюється з дотриманням найвищих стандартів безпеки та конфіденційності. Використання захищених, надійних та перевірених протоколів для передачі та збереження інформації забезпечує додатковий рівень захисту чутливих даних користувача. Це викликає довіру до системи, що є необхідним у сучасному стрімкому світі. Крім того, модуль аналізу активно співпрацює з різними зовнішніми джерелами даних, а також системами. Це співробітництво надає можливість значно підвищити функціональність та достовірність отриманих результатів, гарантуючи найбільш якісні дані. Ця ефективна синергія різних джерел інформації створює

ще й потужну базу для аналітичних висновків. Вона також підтримує точнішу оцінку ситуації на ринку, що стає важливим для прийняття зважених рішень, орієнтуючись на інформацію, що швидко змінюється в умовах сучасності.

У загальному підсумку, цей модуль забезпечує ефективне перетворення сировинних, неструктурованих даних у корисну, чітку та зрозумілу наочну інформацію, яка легко сприймається користувачем. Це, у свою чергу, допомагає користувачам краще усвідомлювати власну поведінку, розумніше й більш продуктивно планувати свій час, тим самим підвищуючи загальний рівень особистої продуктивності.

У результаті позитивний вплив цього процесу відчувається на досягненні як всіх особистісних, так і професійних цілей у житті. Це є основним прагненням та метою кожної людини, яка хоче максимально реалізувати свій потенціал у сучасному світі, завоювавши успіх у всіх сферах свого життя.

3.5 Інтерфейс користувача та взаємодія

Інтерфейс користувача є одним з найважливіших елементів, що гарантує ефективну та безперебійну взаємодію з мобільним застосунком. Цей інтерфейс повинен бути не тільки інтуїтивно зрозумілим та простим у використанні, але й адаптованим до різних рівнів комп'ютерної грамотності користувачів. Це дозволяє їм не лише швидко орієнтуватися в функціоналі, а й без особливих проблем виконувати всі необхідні дії без зайвих зусиль і витрат часу. Звична структура головного меню зазвичай включає основні розділи, які пропонують можливість для перегляду статистики, налаштувань, а також історії активності. Це суттєво спрощує та прискорює доступ до всіх важливих функцій, які можуть знадобитися користувачеві в різний час і в різних ситуаціях.

Візуальний дизайн розроблений з урахуванням принципів мінімалізму та зручності, що є основними та визначальними факторами у створенні максимально комфортного використання.

Використання гармонійних кольорів, чітких, легко читаемых шрифтів та відповідних іконок дозволяє не лише полегшити сприйняття інформації, але й істотно уникнути надмірного інформаційного навантаження на користувача.

Для підвищення зручності та швидкості була передбачена можливість використання сучасного та інтуїтивного керування жестами, таких як свайпи та тапи, що сприяє швидкій навігації між різними розділами програми. Це робить переміщення між елементами простим і зрозумілим для користувача.

Водночас, важливу роль відіграє адаптивність інтерфейсу під різні розміри екранів – від компактних смартфонів до великих планшетів і екранів комп'ютерів. Це забезпечує стабільну і надійну роботу застосунку в різних умовах та на різних пристроях.

Взаємодія користувача з системою базується на чітко структурованих елементах управління, що надає відчуття впевненості під час використання.

Наприклад, для введення даних використовуються зручні та зрозумілі форми з автоматичною перевіркою валідності. Це значно знижує можливість виникнення помилок під час звичайного використання застосунку.

Відгуки системи – у вигляді повідомлень, звукових сигналів або візуальних індикаторів – сприяють тому, щоб користувачі могли швидко орієнтуватися в процесі виконання різних завдань і миттєво отримувати інформацію про статус своїх дій. Крім того, було реалізовано можливість персоналізації інтерфейсу, яка включає налаштування кольорової палітри, вибору шрифтів та розмірів елементів керування. Ці налаштування дозволяють максимально відповідати індивідуальним уподобанням кожного користувача та роблять процес роботи ще більш комфортним.

Таким чином, загалом дизайн інтерфейсу розроблено з урахуванням основних принципів зручності, швидкості, естетичності та інтуїтивності. Ці принципи в комплексі здатні надавати користувачеві неначе магічний досвід.

Загалом, це все разом постійно сприяє високій залученості, а також задоволеності користувачів при відстеженні власної продуктивності в різних аспектах їхньої діяльності, що суттєво підвищує якість використання застосунку. Надійна та гнучка взаємодія забезпечує максимально комфортний досвід користування додатком і суттєво сприяє досягненню цілей його використання в процесі щоденної роботи. Це є надзвичайно важливим для

користувачів у сучасному динамічному світі, де час та ефективність мають особливу цінність.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи розроблено мобільний застосунок для відстеження продуктивності на платформі Android. Отримано такі основні результати:

1. проаналізовано предметну область мобільних застосунків для відстеження продуктивності та розглянуто наявні програмні рішення, що дало змогу визначити їхні переваги й недоліки та обґрунтувати потребу в розробці нативного застосунку з підтримкою офлайн-роботи;

2. досліджено сучасні технології розробки мобільних застосунків та обґрунтовано вибір мови програмування Kotlin, інструментарію Jetpack Compose для побудови інтерфейсу, системи керування базами даних SQLite (через бібліотеку Room) і середовища розробки Android Studio;

3. сформульовано функціональні та нефункціональні вимоги до застосунку, зокрема щодо керування завданнями, обліку часу, аналітики продуктивності, офлайн-доступу та стабільності роботи;

4. обґрунтовано програмну архітектуру на основі шаблону MVVM із застосуванням механізму впровадження залежностей Hilt, що забезпечує модульність, тестованість і масштабованість рішення;

5. спроєктовано структуру локальної бази даних (сутності Task, PomodoroSession, MoodEntry, Habit, HabitCompletion) та реалізовано доступ до неї за допомогою об’єктів DAO і репозиторіїв;

6. реалізовано основні функціональні модулі застосунку: керування завданнями, таймер за методикою Pomodoro, відстеження звичок, щоденник настрою та інформаційну панель (dashboard);

7. реалізовано обробку та агрегацію даних – обчислення щоденної й тижневої статистики, серій виконання звичок і формування показників продуктивності користувача;

8. розроблено інтерфейс користувача засобами Jetpack Compose з темою Material 3 та навігацією між екранами, а також перевірено працездатність основних сценаріїв використання застосунку.

Розроблений застосунок є працездатним і відповідає поставленим вимогам. Подальший розвиток системи доцільно спрямувати на синхронізацію даних між пристроями, хмарне резервне копіювання та розширення аналітичних можливостей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gamma E., Helm R., Johnson R., Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston : Addison-Wesley, 1994. 395 p.
2. Pressman R. S., Maxim B. R. *Software Engineering: A Practitioner's Approach*. 9th ed. New York : McGraw-Hill Education, 2019. 992 p.
3. Sommerville I. *Software Engineering*. 10th ed. Boston : Pearson Education, 2016. 816 p.
4. Fowler M. *Patterns of Enterprise Application Architecture*. Boston : Addison-Wesley Professional, 2002. 533 p.
5. Martin R. C. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Boston : Prentice Hall, 2017. 432 p.
6. Martin R. C. *Clean Code: A Handbook of Agile Software Craftsmanship*. Boston : Prentice Hall, 2008. 464 p.
7. Burnette E. *Hello, Android: Introducing Google's Mobile Development Platform*. Dallas : Pragmatic Bookshelf, 2021. 420 p.
8. Phillips B., Stewart C., Hardy K., Marsicano K. *Android Programming: The Big Nerd Ranch Guide*. 5th ed. Atlanta : Big Nerd Ranch Guides, 2022. 768 p.
9. Jemerov D., Isakova S. *Kotlin in Action*. 2nd ed. Shelter Island : Manning Publications, 2024. 520 p.
10. JetBrains. Kotlin Documentation. URL: https://kotlinlang.org/docs/home.html (дата звернення: 05.06.2026).
11. Google. Android Developers Documentation. URL: https://developer.android.com (дата звернення: 05.06.2026).
12. Google. Jetpack Compose Documentation. URL: https://developer.android.com/jetpack/compose (дата звернення: 05.06.2026).
13. SQLite Consortium. SQLite Documentation. URL: https://www.sqlite.org/docs.html (дата звернення: 05.06.2026).
14. Google. Material Design Guidelines. URL: https://m3.material.io (дата звернення: 05.06.2026).
15. Tidwell J., Brewer C., Valencia A. *Designing Interfaces*. 3rd ed. Sebastopol : O'Reilly Media, 2020. 600 p.
16. Norman D. *The Design of Everyday Things*. Revised and Expanded Edition. New York : Basic Books, 2013. 368 p.
17. Cooper A., Reimann R., Cronin D., Noessel C. *About Face: The Essentials of Interaction Design*. 4th ed. Indianapolis : Wiley, 2014. 720 p.
18. Krug S. *Don't Make Me Think, Revisited: A Common Sense Approach to Web and Mobile Usability*. 3rd ed. Berkeley : New Riders, 2014. 216 p.
19. Nielsen J. *Usability Engineering*. Burlington : Morgan Kaufmann, 1993. 362 p.
20. Chacon S., Straub B. *Pro Git*. 2nd ed. New York : Apress, 2014. 458 p.
21. Freeman E., Robson E. *Head First Design Patterns*. 2nd ed. Sebastopol : O'Reilly Media, 2020. 694 p.
22. Hunt A., Thomas D. *The Pragmatic Programmer*. 20th Anniversary Edition. Boston : Addison-Wesley, 2019. 352 p.

23. Horstmann C. S. *Core Java Volume I – Fundamentals*. 12th ed. Hoboken : Wiley, 2022. 928 p.
24. Gupta A. *Android Application Development with Kotlin*. Birmingham : Packt Publishing, 2023. 480 p.
25. Darcey L., Conder S. *Android Wireless Application Development*. 5th ed. Boston : Addison-Wesley, 2021. 768 p.
26. Shneiderman B., Plaisant C., Cohen M., Jacobs S. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 6th ed. Boston : Pearson, 2017. 624 p.
27. Goodfellow I., Bengio Y., Courville A. *Deep Learning*. Cambridge : MIT Press, 2016. 800 p.
28. Russell S., Norvig P. *Artificial Intelligence: A Modern Approach*. 4th ed. Hoboken : Pearson, 2021. 1136 p.
29. Android Open Source Project. Android Architecture Guide. URL: https://developer.android.com/topic/architecture (дата звернення: 05.06.2026).
30. Google. Android Studio Documentation. URL: https://developer.android.com/studio (дата звернення: 05.06.2026).

Додаток А

Фрагменти програмного коду мобільного застосунку

Лістинг А.1. Конфігурація складання та залежності (build.gradle.kts)

```
plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.kotlin.android)
    alias(libs.plugins.kotlin.compose)
    alias(libs.plugins.hilt.android)
    alias(libs.plugins.ksp)
}

android {
    namespace = "com.productivity.tracker"
    compileSdk = 35

    defaultConfig {
        applicationId = "com.productivity.tracker"
        minSdk = 26          // LocalDate доступний з API 26
        targetSdk = 35
        versionCode = 1
        versionName = "1.0"
    }

    buildTypes {
        release {
            isMinifyEnabled = true
            proguardFiles(getDefaultProguardFile("proguard-android-
optimize.txt"), "proguard-rules.pro")
        }
    }

    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_17
        targetCompatibility = JavaVersion.VERSION_17
    }
    kotlinOptions { jvmTarget = "17" }

    buildFeatures { compose = true }
}

dependencies {
    // ── Jetpack Compose (BOM) ──────────────────────────────────────
    val composeBom = platform("androidx.compose:compose-bom:2024.09.00")
    implementation(composeBom)
    implementation("androidx.compose.ui:ui")
    implementation("androidx.compose.ui:ui-tooling-preview")
    implementation("androidx.compose.material3:material3")
    implementation("androidx.compose.material:material-icons-extended")
    debugImplementation("androidx.compose.ui:ui-tooling")

    // ── Navigation Compose ─────────────────────────────────────────
    implementation("androidx.navigation:navigation-compose:2.8.3")

    // ── Lifecycle / ViewModel ──────────────────────────────────────
    implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.8.6")
```

```
    implementation("androidx.lifecycle:lifecycle-viewmodel-compose:2.8.6")
    implementation("androidx.activity:activity-compose:1.9.2")

    // ── Room ──────────────────────────────────────────────────────────
    val roomVersion = "2.6.1"
    implementation("androidx.room:room-runtime:$roomVersion")
    implementation("androidx.room:room-ktx:$roomVersion")
    ksp("androidx.room:room-compiler:$roomVersion")

    // ── Hilt (DI) ─────────────────────────────────────────────────────
    implementation("com.google.dagger:hilt-android:2.52")
    ksp("com.google.dagger:hilt-android-compiler:2.52")
    implementation("androidx.hilt:hilt-navigation-compose:1.2.0")

    // ── Coroutines ────────────────────────────────────────────────────
    implementation("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.9.0")

    //                ──                DataStore                (налаштування)
──────────────────────────────────────────
    implementation("androidx.datastore:datastore-preferences:1.1.1")
}
```

Лістинг А.2. Моделі даних та сутності Room (Models.kt)

```
package com.productivity.tracker.data.model

import androidx.room.Entity
import androidx.room.PrimaryKey
import java.time.LocalDate
import java.time.LocalDateTime

// ──── Завдання ──────────────────────────────────────────────────────

enum class Priority { LOW, MEDIUM, HIGH }
enum class TaskStatus { TODO, IN_PROGRESS, DONE }
enum class TaskCategory { WORK, STUDY, HEALTH, PERSONAL, OTHER }

@Entity(tableName = "tasks")
data class Task(
    @PrimaryKey(autoGenerate = true) val id: Long = 0,
    val title: String,
    val description: String = "",
    val priority: Priority = Priority.MEDIUM,
    val status: TaskStatus = TaskStatus.TODO,
    val category: TaskCategory = TaskCategory.OTHER,
    val estimatedMinutes: Int = 0,          // запланований час (хв)
    val actualMinutes: Int = 0,             // фактично витрачений час (хв)
    val dueDate: LocalDate? = null,
    val createdAt: LocalDateTime = LocalDateTime.now(),
    val completedAt: LocalDateTime? = null,
    val isRecurring: Boolean = false,
    val recurringDays: Int = 0              // кожні N днів (0 = не повторюється)
)

// ──── Сесія Помодоро ────────────────────────────────────────────────

enum class PomodoroType { WORK, SHORT_BREAK, LONG_BREAK }
```

```
@Entity(tableName = "pomodoro_sessions")
data class PomodoroSession(
    @PrimaryKey(autoGenerate = true) val id: Long = 0,
    val taskId: Long? = null,
    val type: PomodoroType = PomodoroType.WORK,
    val durationMinutes: Int = 25,
    val startedAt: LocalDateTime = LocalDateTime.now(),
    val endedAt: LocalDateTime? = null,
    val wasCompleted: Boolean = false
)

// ——— Щоденник настрою ——————————————————————————————————————

@Entity(tableName = "mood_entries")
data class MoodEntry(
    @PrimaryKey(autoGenerate = true) val id: Long = 0,
    val date: LocalDate = LocalDate.now(),
    val moodScore: Int,                    // 1–5
    val energyLevel: Int,                  // 1–5
    val focusLevel: Int,                   // 1–5
    val notes: String = "",
    val createdAt: LocalDateTime = LocalDateTime.now()
)

// ——— Звичка ————————————————————————————————————————————————

@Entity(tableName = "habits")
data class Habit(
    @PrimaryKey(autoGenerate = true) val id: Long = 0,
    val name: String,
    val description: String = "",
    val iconEmoji: String = "✓",
    val targetDaysPerWeek: Int = 7,
    val createdAt: LocalDate = LocalDate.now(),
    val isActive: Boolean = true
)

@Entity(tableName = "habit_completions")
data class HabitCompletion(
    @PrimaryKey(autoGenerate = true) val id: Long = 0,
    val habitId: Long,
    val date: LocalDate = LocalDate.now(),
    val completedAt: LocalDateTime = LocalDateTime.now()
)

// ——— Статистика (агрегована) ——————————————————————————————————

data class DailyStats(
    val date: LocalDate,
    val tasksCompleted: Int,
    val totalTasksPlanned: Int,
    val pomodoroCount: Int,
    val focusMinutes: Int,
    val avgMoodScore: Float,
    val habitsCompleted: Int,
    val totalHabits: Int
) {
    val completionRate: Float
```

```
        get() = if (totalTasksPlanned > 0) tasksCompleted.toFloat() /
totalTasksPlanned else 0f
    val habitRate: Float
        get() = if (totalHabits > 0) habitsCompleted.toFloat() / totalHabits
else 0f
}

data class WeeklyReport(
    val weekStart: LocalDate,
    val weekEnd: LocalDate,
    val dailyStats: List<DailyStats>,
    val totalFocusMinutes: Int,
    val totalTasksCompleted: Int,
    val avgMood: Float,
    val bestDay: LocalDate?,
    val streakDays: Int
)
```

Лістинг А.3. Конвертери типів, DAO завдань і клас бази даних (ProductivityDatabase.kt)

```
package com.productivity.tracker.data.database

import androidx.room.*
import androidx.room.Database
import androidx.room.RoomDatabase
import androidx.room.TypeConverter
import androidx.room.TypeConverters
import com.productivity.tracker.data.model.*
import kotlinx.coroutines.flow.Flow
import java.time.LocalDate
import java.time.LocalDateTime

// ─── Конвертери типів для Room ──────────────────────────────────────

class Converters {
    @TypeConverter fun fromLocalDate(date: LocalDate?): String? =
date?.toString()
    @TypeConverter fun toLocalDate(value: String?): LocalDate? = value?.let
{ LocalDate.parse(it) }

    @TypeConverter fun fromLocalDateTime(dt: LocalDateTime?): String? =
dt?.toString()
    @TypeConverter fun toLocalDateTime(value: String?): LocalDateTime? =
        value?.let { LocalDateTime.parse(it) }

    @TypeConverter fun fromPriority(p: Priority): String = p.name
    @TypeConverter fun toPriority(v: String): Priority = Priority.valueOf(v)

    @TypeConverter fun fromTaskStatus(s: TaskStatus): String = s.name
    @TypeConverter fun toTaskStatus(v: String): TaskStatus =
TaskStatus.valueOf(v)

    @TypeConverter fun fromCategory(c: TaskCategory): String = c.name
    @TypeConverter fun toCategory(v: String): TaskCategory =
TaskCategory.valueOf(v)

    @TypeConverter fun fromPomodoroType(t: PomodoroType): String = t.name
```

```
    @TypeConverter    fun    toPomodoroType(v:    String):    PomodoroType    =
PomodoroType.valueOf(v)
}

// ─── DAO: Завдання ──────────────────────────────────────────────────────

@Dao
interface TaskDao {

    @Query("SELECT * FROM tasks ORDER BY priority DESC, dueDate ASC")
    fun getAllTasks(): Flow<List<Task>>

    @Query("SELECT * FROM tasks WHERE status != 'DONE' ORDER BY priority DESC,
dueDate ASC")
    fun getActiveTasks(): Flow<List<Task>>

    @Query("SELECT * FROM tasks WHERE status = 'DONE' ORDER BY completedAt DESC")
    fun getCompletedTasks(): Flow<List<Task>>

    @Query("SELECT * FROM tasks WHERE category = :category ORDER BY priority
DESC")
    fun getTasksByCategory(category: TaskCategory): Flow<List<Task>>

    @Query("SELECT * FROM tasks WHERE dueDate = :date")
    fun getTasksByDate(date: LocalDate): Flow<List<Task>>

    @Query("""
        SELECT * FROM tasks
        WHERE dueDate BETWEEN :start AND :end
        ORDER BY dueDate ASC, priority DESC
    """)
    fun getTasksInRange(start: LocalDate, end: LocalDate): Flow<List<Task>>

    @Query("SELECT * FROM tasks WHERE id = :id")
    suspend fun getTaskById(id: Long): Task?

    @Query("""
        SELECT COUNT(*) FROM tasks
        WHERE status = 'DONE' AND date(completedAt) = :date
    """)
    suspend fun getCompletedCountByDate(date: String): Int

    @Query("""
        SELECT COUNT(*) FROM tasks
        WHERE date(createdAt) = :date OR dueDate = :localDate
    """)
    suspend fun getPlannedCountByDate(date: String, localDate: LocalDate): Int

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertTask(task: Task): Long

    @Update
    suspend fun updateTask(task: Task)

    @Delete
    suspend fun deleteTask(task: Task)
```

```
    @Query("UPDATE tasks SET status = 'DONE', completedAt = :completedAt WHERE
id = :id")
    suspend fun completeTask(id: Long, completedAt: LocalDateTime =
LocalDateTime.now())

    @Query("UPDATE tasks SET actualMinutes = actualMinutes + :minutes WHERE id
= :id")
    suspend fun addFocusTime(id: Long, minutes: Int)
}

// ─── DAO: Сесії Помодоро ──────────────────────────────────────────

@Dao
interface PomodoroDao {

    @Query("""
        SELECT * FROM pomodoro_sessions
        WHERE date(startedAt) = :date

// ... DAO PomodoroDao, MoodDao, HabitDao – аналогічно ...

@Database(
    entities = [Task::class, PomodoroSession::class, MoodEntry::class,
Habit::class, HabitCompletion::class],
    version = 1,
    exportSchema = false
)
@TypeConverters(Converters::class)
abstract class ProductivityDatabase : RoomDatabase() {
    abstract fun taskDao(): TaskDao
    abstract fun pomodoroDao(): PomodoroDao
    abstract fun moodDao(): MoodDao
    abstract fun habitDao(): HabitDao

    companion object {
        const val DATABASE_NAME = "productivity_db"
    }
}
```

Лістинг А.4. Репозиторії: бізнес-логіка, обчислення серій та агрегація статистики (Repositories.kt)

```
package com.productivity.tracker.data.repository

import com.productivity.tracker.data.database.*
import com.productivity.tracker.data.model.*
import kotlinx.coroutines.flow.Flow
import java.time.LocalDate
import java.time.LocalDateTime
import java.time.format.DateTimeFormatter
import javax.inject.Inject
import javax.inject.Singleton

private val SQL_DATE = DateTimeFormatter.ofPattern("yyyy-MM-dd")

// ─── Репозиторій завдань ──────────────────────────────────────────
```

```
@Singleton
class TaskRepository @Inject constructor(private val dao: TaskDao) {

    fun getAllTasks(): Flow<List<Task>> = dao.getAllTasks()
    fun getActiveTasks(): Flow<List<Task>> = dao.getActiveTasks()
    fun getCompletedTasks(): Flow<List<Task>> = dao.getCompletedTasks()
    fun getTasksByCategory(cat: TaskCategory) = dao.getTasksByCategory(cat)
    fun getTasksForDate(date: LocalDate) = dao.getTasksByDate(date)
    fun     getTasksInRange(start:     LocalDate,     end:     LocalDate)     =
dao.getTasksInRange(start, end)

    suspend fun getById(id: Long): Task? = dao.getTaskById(id)

    suspend fun createTask(task: Task): Long = dao.insertTask(task)

    suspend fun updateTask(task: Task) = dao.updateTask(task)

    suspend fun deleteTask(task: Task) = dao.deleteTask(task)

    suspend fun completeTask(id: Long) = dao.completeTask(id)

    suspend     fun     addFocusTime(taskId:     Long,     minutes:     Int)     =
dao.addFocusTime(taskId, minutes)

    suspend fun getCompletedCount(date: LocalDate): Int =
        dao.getCompletedCountByDate(date.format(SQL_DATE))

    suspend fun getPlannedCount(date: LocalDate): Int =
        dao.getPlannedCountByDate(date.format(SQL_DATE), date)
}

// ─── Репозиторій Помодоро ──────────────────────────────────────────

@Singleton
class PomodoroRepository @Inject constructor(private val dao: PomodoroDao) {

    fun getSessionsForDate(date: LocalDate): Flow<List<PomodoroSession>> =
        dao.getSessionsByDate(date.format(SQL_DATE))

    suspend fun startSession(taskId: Long?, type: PomodoroType, durationMinutes:
Int): Long {
        val session = PomodoroSession(
            taskId = taskId,
            type = type,
            durationMinutes = durationMinutes
        )
        return dao.insertSession(session)
    }

    suspend fun finishSession(id: Long, completed: Boolean) {
        dao.finishSession(id, LocalDateTime.now(), completed)
    }

    suspend fun getTodayFocusMinutes(): Int =
        dao.getTotalFocusMinutesByDate(LocalDate.now().format(SQL_DATE))
```

```
    suspend fun getTodayPomodoroCount(): Int =
        dao.getCompletedPomodorosCount(LocalDate.now().format(SQL_DATE))

    suspend fun getFocusMinutesInRange(start: LocalDate, end: LocalDate): Int =
        dao.getTotalFocusMinutesInRange(start.format(SQL_DATE),
end.format(SQL_DATE))
}

// ─── Репозиторій настрою ──────────────────────────────────────────

@Singleton
class MoodRepository @Inject constructor(private val dao: MoodDao) {

    fun getAllEntries(): Flow<List<MoodEntry>> = dao.getAllMoodEntries()

    suspend fun getTodayMood(): MoodEntry? = dao.getMoodByDate(LocalDate.now())

    suspend fun getMoodForDate(date: LocalDate): MoodEntry? =
dao.getMoodByDate(date)

    fun getMoodEntriesInRange(start: LocalDate, end: LocalDate):
Flow<List<MoodEntry>> =
        dao.getMoodEntriesInRange(start, end)

    suspend fun getAvgMoodInRange(start: LocalDate, end: LocalDate): Float =
        dao.getAvgMoodInRange(start, end) ?: 0f

    suspend fun saveMoodEntry(entry: MoodEntry): Long =
dao.insertMoodEntry(entry)

    suspend fun updateMoodEntry(entry: MoodEntry) = dao.updateMoodEntry(entry)
}

// ─── Репозиторій звичок ───────────────────────────────────────────

@Singleton
class HabitRepository @Inject constructor(private val dao: HabitDao) {

    fun getActiveHabits(): Flow<List<Habit>> = dao.getActiveHabits()

    fun getCompletionsForHabit(habitId: Long): Flow<List<HabitCompletion>> =
        dao.getCompletionsForHabit(habitId)

    suspend fun isCompletedToday(habitId: Long): Boolean =
        dao.isHabitCompletedOnDate(habitId, LocalDate.now())

    suspend fun getTodayCompletions(): List<HabitCompletion> =
        dao.getCompletionsByDate(LocalDate.now())

    suspend fun getCompletionCountInRange(habitId: Long, start: LocalDate, end:
LocalDate): Int =
        dao.getCompletionCountInRange(habitId, start, end)

    suspend fun createHabit(habit: Habit): Long = dao.insertHabit(habit)
```

```
    suspend fun updateHabit(habit: Habit) = dao.updateHabit(habit)

    suspend fun toggleHabitCompletion(habitId: Long, date: LocalDate =
LocalDate.now()) {
        if (dao.isHabitCompletedOnDate(habitId, date)) {
            dao.removeCompletion(habitId, date)
        } else {
            dao.insertCompletion(HabitCompletion(habitId = habitId, date = date))
        }
    }

    /** Розраховує поточну серію для звички (кількість послідовних днів) */
    suspend fun calculateStreak(habitId: Long): Int {
        var streak = 0
        var date = LocalDate.now()
        while (dao.isHabitCompletedOnDate(habitId, date)) {
            streak++
            date = date.minusDays(1)
        }
        return streak
    }
}

// ─── Репозиторій статистики (агрегація) ──────────────────────────────

@Singleton
class StatsRepository @Inject constructor(
    private val taskRepo: TaskRepository,
    private val pomodoroRepo: PomodoroRepository,
    private val moodRepo: MoodRepository,
    private val habitDao: HabitDao
) {

    suspend fun getDailyStats(date: LocalDate): DailyStats {
        val allHabits = mutableListOf<Habit>()
        // Отримуємо список звичок синхронно через collect (спрощено)
        val completedTasks = taskRepo.getCompletedCount(date)
        val plannedTasks = taskRepo.getPlannedCount(date)
        val focusMinutes = pomodoroRepo.getFocusMinutesInRange(date, date)
        val pomodoroCount = habitDao.getCompletionsByDate(date).size
        val avgMood = moodRepo.getAvgMoodInRange(date, date)
        val habitCompletions = habitDao.getCompletionsByDate(date).size

        return DailyStats(
            date = date,
            tasksCompleted = completedTasks,
            totalTasksPlanned = plannedTasks,
            pomodoroCount = focusMinutes / 25, // приблизно
            focusMinutes = focusMinutes,
            avgMoodScore = avgMood,
            habitsCompleted = habitCompletions,
            totalHabits = 0 // буде заповнено у ViewModel
        )
    }

    suspend fun getWeeklyReport(weekStart: LocalDate): WeeklyReport {
        val weekEnd = weekStart.plusDays(6)
        val dailyStats = (0..6).map { offset ->
            getDailyStats(weekStart.plusDays(offset.toLong()))
        }
```

```
        val totalFocus = pomodoroRepo.getFocusMinutesInRange(weekStart, weekEnd)
        val totalCompleted = dailyStats.sumOf { it.tasksCompleted }
        val avgMood = moodRepo.getAvgMoodInRange(weekStart, weekEnd)
        val bestDay = dailyStats.maxByOrNull { it.completionRate }?.date

        return WeeklyReport(
            weekStart = weekStart,
            weekEnd = weekEnd,
            dailyStats = dailyStats,
            totalFocusMinutes = totalFocus,
            totalTasksCompleted = totalCompleted,
            avgMood = avgMood,
            bestDay = bestDay,
            streakDays = 0
        )
    }
}
```

Лістинг А.5. Впровадження залежностей Hilt (AppModule.kt)

```
package com.productivity.tracker

import android.app.Application
import android.content.Context
import androidx.room.Room
import com.productivity.tracker.data.database.*
import com.productivity.tracker.data.repository.*
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.HiltAndroidApp
import dagger.hilt.android.qualifiers.ApplicationContext
import dagger.hilt.components.SingletonComponent
import javax.inject.Singleton

// ─── Application ──────────────────────────────────────────────────────

@HiltAndroidApp
class ProductivityApp : Application()

// ─── Hilt модуль ──────────────────────────────────────────────────────

@Module
@InstallIn(SingletonComponent::class)
object AppModule {

    @Provides
    @Singleton
    fun provideDatabase(@ApplicationContext ctx: Context): ProductivityDatabase
=
        Room.databaseBuilder(
            ctx,
            ProductivityDatabase::class.java,
            ProductivityDatabase.DATABASE_NAME
        ).build()

    @Provides fun provideTaskDao(db: ProductivityDatabase): TaskDao =
db.taskDao()
```

```
    @Provides fun providePomodoroDao(db: ProductivityDatabase): PomodoroDao =
db.pomodoroDao()
    @Provides fun provideMoodDao(db: ProductivityDatabase): MoodDao =
db.moodDao()
    @Provides fun provideHabitDao(db: ProductivityDatabase): HabitDao =
db.habitDao()

    @Provides
    @Singleton
    fun provideTaskRepository(dao: TaskDao): TaskRepository = TaskRepository(dao)

    @Provides
    @Singleton
    fun providePomodoroRepository(dao: PomodoroDao): PomodoroRepository =
PomodoroRepository(dao)

    @Provides
    @Singleton
    fun provideMoodRepository(dao: MoodDao): MoodRepository = MoodRepository(dao)

    @Provides
    @Singleton
    fun provideHabitRepository(dao: HabitDao): HabitRepository =
HabitRepository(dao)

    @Provides
    @Singleton
    fun provideStatsRepository(
        taskRepo: TaskRepository,
        pomodoroRepo: PomodoroRepository,
        moodRepo: MoodRepository,
        habitDao: HabitDao
    ): StatsRepository = StatsRepository(taskRepo, pomodoroRepo, moodRepo,
habitDao)
}
```

Лістинг А.6. Точка входу та навігація (MainActivity.kt)

```
package com.productivity.tracker

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.*
import androidx.compose.material.icons.outlined.*
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.*
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.navigation.NavDestination.Companion.hierarchy
import androidx.navigation.NavGraph.Companion.findStartDestination
import androidx.navigation.compose.*
import com.productivity.tracker.ui.screens.*
import com.productivity.tracker.ui.theme.ProductivityTheme
import dagger.hilt.android.AndroidEntryPoint

// ─── Маршрути навігації ──────────────────────────────────────────────
```

```
sealed class Screen(val route: String) {
    object Dashboard : Screen("dashboard")
    object Tasks     : Screen("tasks")
    object Pomodoro  : Screen("pomodoro")
    object Habits    : Screen("habits")
}

data class BottomNavItem(
    val screen: Screen,
    val label: String,
    val selectedIcon: ImageVector,
    val unselectedIcon: ImageVector
)

val bottomNavItems = listOf(
    BottomNavItem(Screen.Dashboard,    "Дашборд",        Icons.Filled.Dashboard,
Icons.Outlined.Dashboard),
    BottomNavItem(Screen.Tasks,           "Завдання",   Icons.Filled.CheckCircle,
Icons.Outlined.CheckCircle),
    BottomNavItem(Screen.Pomodoro,          "Помодоро",      Icons.Filled.Timer,
Icons.Outlined.Timer),
    BottomNavItem(Screen.Habits,           "Звички",        Icons.Filled.Repeat,
Icons.Outlined.Repeat)
)

// ─── MainActivity ──────────────────────────────────────────────────────

@AndroidEntryPoint
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            ProductivityTheme {
                ProductivityApp()
            }
        }
    }
}

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ProductivityApp() {
    val navController = rememberNavController()

    Scaffold(
        bottomBar = {
            NavigationBar(
                containerColor                                                =
com.productivity.tracker.ui.theme.ProductivityColors.Surface
            ) {
                val                     navBackStackEntry                     by
navController.currentBackStackEntryAsState()
                val currentDestination = navBackStackEntry?.destination

                bottomNavItems.forEach { item ->
                    val isSelected = currentDestination?.hierarchy?.any {
                        it.route == item.screen.route
                    } == true
```

```
                    NavigationBarItem(
                        selected = isSelected,
                        onClick = {
                            navController.navigate(item.screen.route) {

popUpTo(navController.graph.findStartDestination().id) {
                                    saveState = true
                                }
                                launchSingleTop = true
                                restoreState = true
                            }
                        },
                        icon = {
                            Icon(
                                if    (isSelected)    item.selectedIcon    else
item.unselectedIcon,
                                contentDescription = item.label
                            )
                        },
                        label = { Text(item.label) }
                    )
                }
            }
        }
    ) { innerPadding ->
        NavHost(
            navController = navController,
            startDestination = Screen.Dashboard.route,
            modifier = Modifier.padding(innerPadding)
        ) {
            composable(Screen.Dashboard.route) { DashboardScreen() }
            composable(Screen.Tasks.route)     { TasksScreen() }
            composable(Screen.Pomodoro.route)  { PomodoroScreen() }
            composable(Screen.Habits.route)    { HabitsScreen() }
        }
    }
}
```

Лістинг А.7. Палітра кольорів та тема Material 3 (Theme.kt)

```
package com.productivity.tracker.ui.theme

import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.graphics.Color

// ─── Палітра кольорів ──────────────────────────────────────────────

object ProductivityColors {
    val Primary       = Color(0xFF4F6EF7)   // синьо-фіолетовий акцент
    val PrimaryDark   = Color(0xFF3A56D4)
    val Secondary     = Color(0xFF22C55E)   // зелений успіху
    val Background    = Color(0xFF0F1117)   // темний фон
    val Surface       = Color(0xFF1A1D27)
    val SurfaceCard   = Color(0xFF22263A)
    val OnBackground  = Color(0xFFE8EAF6)
    val OnSurface     = Color(0xFFCDD1E8)
    val Hint          = Color(0xFF6B7280)
```

```
    val PriorityHigh   = Color(0xFFEF4444)
    val PriorityMedium = Color(0xFFF59E0B)
    val PriorityLow    = Color(0xFF10B981)

    val PomodoroWork       = Color(0xFFEF4444)
    val PomodoroShortBreak = Color(0xFF10B981)
    val PomodoroLongBreak  = Color(0xFF3B82F6)

    val Mood1 = Color(0xFFEF4444)
    val Mood2 = Color(0xFFF97316)
    val Mood3 = Color(0xFFF59E0B)
    val Mood4 = Color(0xFF84CC16)
    val Mood5 = Color(0xFF22C55E)
}

private val DarkColorScheme = darkColorScheme(
    primary          = ProductivityColors.Primary,
    onPrimary        = Color.White,
    primaryContainer = ProductivityColors.PrimaryDark,
    secondary        = ProductivityColors.Secondary,
    background       = ProductivityColors.Background,
    surface          = ProductivityColors.Surface,
    onBackground     = ProductivityColors.OnBackground,
    onSurface        = ProductivityColors.OnSurface
)

@Composable
fun ProductivityTheme(content: @Composable () -> Unit) {
    MaterialTheme(
        colorScheme = DarkColorScheme,
        typography  = Typography(),
        content     = content
    )
}
```

Лістинг А.8. Екран завдань на Jetpack Compose (фрагмент TasksScreen)

```
// ═══════════════════════════════════════════════════════════════
// Екран завдань
// ═══════════════════════════════════════════════════════════════

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun TasksScreen(viewModel: TaskViewModel = hiltViewModel()) {
    val state by viewModel.uiState.collectAsState()
    var showAddDialog by remember { mutableStateOf(false) }
    var selectedTab by remember { mutableStateOf(0) }

    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text("Мої завдання", fontWeight = FontWeight.Bold) },
                actions = {
                    IconButton(onClick = { showAddDialog = true }) {
                        Icon(Icons.Default.Add, "Додати завдання")
                    }
                },
                colors = TopAppBarDefaults.topAppBarColors(
```

```
                    containerColor = ProductivityColors.Surface
                )
            )
        },
        containerColor = ProductivityColors.Background
    ) { padding ->
        Column(Modifier.padding(padding)) {

            // Фільтр категорій
            CategoryFilterRow(
                selectedCategory = state.selectedCategory,
                onCategorySelected = viewModel::filterByCategory
            )

            // Вкладки активні / виконані
            TabRow(
                selectedTabIndex = selectedTab,
                containerColor = ProductivityColors.Surface
            ) {
                Tab(
                    selected = selectedTab == 0,
                    onClick = { selectedTab = 0 },
                    text = { Text("Активні (${state.activeTasks.size})") }
                )
                Tab(
                    selected = selectedTab == 1,
                    onClick = { selectedTab = 1 },
                    text = { Text("Виконані (${state.completedTasks.size})") }
                )
            }

            val tasks = if (selectedTab == 0) state.activeTasks else
state.completedTasks

            if (tasks.isEmpty()) {
                EmptyTasksPlaceholder(isCompleted = selectedTab == 1)
            } else {
                LazyColumn(
                    contentPadding = PaddingValues(16.dp),
                    verticalArrangement = Arrangement.spacedBy(10.dp)
                ) {
                    items(tasks, key = { it.id }) { task ->
                        TaskCard(
                            task = task,
                            onComplete = { viewModel.completeTask(task.id) },
                            onDelete = { viewModel.deleteTask(task) }
                        )
                    }
                }
            }
        }

        // Повідомлення-снекбар
        state.message?.let { msg ->
            Snackbar(
                modifier = Modifier.padding(16.dp)
            ) { Text(msg) }
        }
    }
```

```
if (showAddDialog) {
    AddTaskDialog(
        onDismiss = { showAddDialog = false },
        onConfirm = { title, desc, priority, category, estimated ->
            viewModel.addTask(title, desc, priority, category, estimated)
```