

**Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення**

**КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»**

**РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ФІТНЕС-
ОРІЄНТОВАНИМИ ЗАВДАННЯМИ ВИКОРИСТОВУЮЧИ FLUTTER ТА
KOTLIN**

**DEVELOPING A MOBILE APPLICATION FOR MANAGING FITNESS-
ORIENTED TASKS USING FLUTTER AND KOTLIN**

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконала: здобувачка вищої освіти
групи ІПЗ-43
Рижок Соломія Юріївна

Керівник:
Повстяна Юлія Славомирівна

Кваліфікаційну роботу
допущено до захисту
«___» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних та інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Галузь знань: 12 «Інформаційні технології»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітня програма: «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

«__» _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Рижок Соломії Юріївни

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Розробка мобільного застосунку для управління фітнес-орієнтованими завданнями використовуючи Flutter та Kotlin»

Керівник роботи: к.т.н., доцент Повстяна Ю. С.

затверджені наказом закладу вищої освіти від «31» грудня 2025 р. № 541/01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра «29» травня 2026 р.

3. Вихідні дані до роботи: предметна область планування та управління фітнес-орієнтованими завданнями, технології розробки мобільного програмного забезпечення (Flutter, Dart, Kotlin), засоби локального зберігання даних (SQLite, Drift), вимоги до мобільних інформаційних систем і забезпечення зручності користувацького інтерфейсу.

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): аналіз сучасного стану проблем використання мобільних технологій для підтримки фізичної активності та планування завдань, постановка завдання на кваліфікаційну роботу, формування вимог до мобільного застосунку, обґрунтування вибору технологій та засобів розробки, проєктування архітектури програмного забезпечення, практична реалізація мобільного застосунку, розробка структури бази даних, розробка користувацького інтерфейсу, тестування та налагодження застосунку.

5. Перелік графічного матеріалу: 22 рисунок, 14 лістингів, 1 таблиця.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	<i>Повстяна Ю. С.</i>		
Специфікація вимог до розробленої системи	<i>Повстяна Ю. С.</i>		
Розробка об'єкта проектування	<i>Повстяна Ю. С.</i>		
Нормоконтроль	<i>Суринович О. М.</i>		
Гарант ОП	<i>Ліщина Н. М.</i>		
Показник запозичень тексту	_____ %		
Академічна доброчесність	<i>Повстяна Ю. С.</i>		

7. Дата видачі завдання «3» лютого 2026 р.

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 14.02.2026 р.	
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 03.03.2026 р.	
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 14.03.2026 р.	
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 14.04.2026 р.	
5	Практична реалізація об'єкта проектування та розробка бази даних	до 20.04.2026 р.	
6	Тестування та налагодження об'єкта проектування	до 04.05.2026 р.	
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедрі	до 29.05.2026 р.	

Здобувач вищої освіти _____

Соломія РИЖОК

Керівник кваліфікаційної роботи _____

Юлія ПОВСТЯНА

АНОТАЦІЯ

Рижок С. Ю. Розробка мобільного застосунку для управління фітнес-орієнтованими завданнями використовуючи Flutter та Kotlin. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2026.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел.

У першому розділі проведено дослідження предметної області, пов'язаної з використанням мобільних технологій для планування особистих завдань та підтримки фізичної активності користувачів, проаналізовано сучасні програмні рішення, їх переваги та недоліки, а також обґрунтовано доцільність розробки власного мобільного застосунку. У другому розділі сформовано функціональні та нефункціональні вимоги до програмного продукту, виконано проектування архітектури системи із використанням UML-моделей та обґрунтовано вибір технологій, методів і засобів розробки. У третьому розділі описано процес реалізації мобільного застосунку засобами Flutter, Dart та Kotlin, розроблено структуру бази даних із використанням SQLite та Drift, реалізовано механізми роботи із завданнями, повторюваними подіями, шаблонами вправ, тренувальними сетами, таймерами та нагадуваннями, а також наведено результати тестування, налагодження та підготовки документації програмного продукту. У висновках підсумовано результати виконаної роботи, підтверджено досягнення поставленої мети та повне виконання визначених завдань.

Ключові слова: мобільний застосунок, фітнес-орієнтовані завдання, Flutter, Kotlin, Dart, SQLite, Drift, планування завдань, фізична активність.

ABSTRACT

Ryzhok S. Yu. Developing a Mobile Application for Managing Fitness-Oriented Tasks Using Flutter and Kotlin. Manuscript. Bachelor's Qualification Thesis of the Educational Program "Software Engineering" in the specialty "Software Engineering". Lutsk National Technical University. Lutsk, 2026.

The bachelor's qualification thesis consists of an introduction, three chapters, conclusions, and a list of references.

The first chapter presents a study of the subject area related to the use of mobile technologies for personal task planning and support of users' physical activity, analyzes modern software solutions, their advantages and disadvantages, and substantiates the feasibility of developing a proprietary mobile application. The second chapter defines the functional and non-functional requirements for the software product, designs the system architecture using UML models, and substantiates the choice of technologies, methods, and development tools. The third chapter describes the implementation process of the mobile application using Flutter, Dart, and Kotlin, develops the database structure using SQLite and Drift, implements mechanisms for working with tasks, recurring events, exercise templates, workout sets, timers, and reminders, and presents the results of testing, debugging, and preparation of the software documentation. The conclusions summarize the results of the completed work, confirm the achievement of the stated goal, and the full completion of the defined tasks.

Keywords: mobile application, fitness-oriented tasks, Flutter, Kotlin, Dart, SQLite, Drift, task planning, physical activity.

ЗМІСТ

ВСТУП	7
РОЗДІЛ 1 АНАЛІЗ МОБІЛЬНИХ ЗАСТОСУНКІВ ДЛЯ УПРАВЛІННЯ ФІТНЕС-ОРІЄНТОВАНИМИ ЗАВДАННЯМИ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ	9
1.1 Аналіз сучасного стану проблеми	9
1.2 Постановка завдання на кваліфікаційну роботу бакалавра	15
РОЗДІЛ 2 СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ	16
2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення	16
2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання	20
РОЗДІЛ 3 РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ФІТНЕС-ОРІЄНТОВАНИМИ ЗАВДАННЯМИ ВИКОРИСТОВУЮЧИ FLUTTER ТА KOTLIN	26
3.1 Практична реалізація об'єкта проектування	26
3.2 Тестування та налагодження інформаційно-комп'ютерної системи	38
3.3 Розробка бази даних мобільного застосунку	43
3.4 Розробка користувацького інтерфейсу мобільного застосунку	47
3.5 Розробка документації для програмного продукту	52
ВИСНОВКИ	54
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	56

ВСТУП

Актуальність теми. У сучасних умовах мобільні технології стали невід'ємною складовою повсякденного життя та активно використовуються для організації роботи, навчання, відпочинку й підтримки здорового способу життя. Одним із важливих напрямів застосування мобільних пристроїв є планування та контроль фізичної активності, оскільки регулярні тренування потребують систематичності, дисципліни та постійного контролю результатів. Водночас існуючі програмні рішення часто орієнтовані або на керування повсякденними завданнями, або виключно на фітнес-функції, що створює потребу у створенні програмного продукту, здатного поєднати можливості планування завдань та підтримки фізичної активності в межах єдиної інформаційної системи.

Сучасний розвиток мобільних технологій характеризується широким використанням кросплатформних засобів розробки, локальних сховищ даних, інтелектуальних механізмів нагадувань та засобів аналізу активності користувачів. Світові тенденції розвитку програмного забезпечення орієнтовані на створення мобільних застосунків, які забезпечують автономну роботу, персоналізацію функціональних можливостей, зручний користувацький інтерфейс та ефективні механізми контролю діяльності користувача. У зв'язку з цим актуальним є дослідження та застосування сучасних технологій розробки мобільного програмного забезпечення для створення систем підтримки фізичної активності та планування особистих завдань.

Метою кваліфікаційної роботи бакалавра є розробка мобільного застосунку для управління фітнес-орієнтованими завданнями з використанням Flutter та Kotlin, який забезпечуватиме планування, контроль та підтримку фізичної активності користувачів.

Об'єктом дослідження є процес управління фітнес-орієнтованими завданнями та підтримки фізичної активності користувачів із використанням мобільних технологій.

Предметом дослідження є методи, засоби та технології розробки мобільних застосунків для управління фітнес-орієнтованими завданнями.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- провести аналіз предметної області, сучасних мобільних технологій та існуючих програмних рішень;
- спроектувати архітектуру мобільного застосунку;
- реалізувати користувацький інтерфейс засобами фреймворку Flutter із використанням мови програмування Dart;
- розробити механізми створення, редагування та планування завдань із використанням Flutter і Dart;
- реалізувати функції роботи з шаблонами вправ, тренувальними сетами, таймерами та нагадуваннями із використанням Flutter і Kotlin;
- розробити модель бази даних і систему локального зберігання та обробки даних із використанням SQLite та Drift;
- реалізувати механізми формування статистики, резервного копіювання та відновлення даних користувача;
- провести тестування та налагодження програмного забезпечення.

Практичне значення роботи полягає у можливості використання мобільного застосунку для підвищення ефективності планування особистої діяльності та підтримки регулярної фізичної активності. Результати роботи можуть бути використані під час створення та підтримки програмних продуктів у сфері персонального планування, фітнесу та здорового способу життя.

РОЗДІЛ 1

АНАЛІЗ МОБІЛЬНИХ ЗАСТОСУНКІВ ДЛЯ УПРАВЛІННЯ ФІТНЕС-ОРІЄНТОВАНИМИ ЗАВДАННЯМИ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Нині мобільні технології дедалі частіше використовуються для підтримки фізичної активності та формування здорового способу життя. Дослідження показують, що цифрові інструменти здатні не лише фіксувати інформацію про виконані тренування, а й позитивно впливати на поведінку користувачів.

Однією з основних причин використання мобільних застосунків є функція самоконтролю. Користувачі активно використовують інструменти відстеження показників, історію виконаних тренувань та механізми порівняння результатів за певні проміжки часу. Наявність збережених даних про попередні досягнення сприяє підвищенню мотивації, оскільки користувачі прагнуть покращувати власні результати та перевищувати попередні показники. Крім цього, важливу роль у підтримці активності відіграють додаткові функціональні можливості застосунків, серед яких постановка цілей, нагадування, механізми зворотного зв'язку та персоналізація. Поєднання декількох функцій у межах одного застосунку сприяє формуванню більш стійких звичок і підтримує довготривале використання системи [1].

Зі зростанням популярності таких мобільних застосунків виникає необхідність вибору технічного підходу, який забезпечить високий рівень продуктивності, функціональності та ефективності використання ресурсів. Для вирішення цього питання у дослідженні проводиться порівняльний аналіз трьох основних підходів до створення мобільних застосунків: нативного, гібридного та кросплатформного. Для їх оцінювання використовувалися такі критерії, як швидкодія, вартість розробки, доступ до можливостей пристрою та складність підтримки програмного продукту.

Згідно з результатами дослідження [2], нативний підхід забезпечує найвищу продуктивність та повний доступ до апаратних можливостей пристрою, проте потребує створення окремих програмних рішень для кожної платформи, що збільшує витрати часу та ресурсів. Гібридні рішення дозволяють використовувати єдину кодову базу для різних платформ і спрощують процес розробки, однак можуть мати певні обмеження щодо продуктивності. Суттєвий акцент у межах даного дослідження зроблено на кросплатформні рішення, які поєднують переваги двох попередніх підходів, що дозволяє скоротити витрати на створення та підтримку програмного забезпечення, забезпечуючи при цьому достатній рівень швидкодії та функціональності.

На основі цього можна зробити висновок, що сучасні тенденції у сфері мобільної розробки демонструють зростання популярності кросплатформних рішень. Однією з найбільш поширених технологій такого типу є Flutter, який використовує єдину кодову базу, що спрощує процес розробки, сприяє скороченню витрат часу і ресурсів, а також дозволяє розширити охоплення користувачів та підтримувати однакову функціональність застосунків на різних платформах.

У дослідженні, присвяченому аналізу Flutter як засобу кросплатформної розробки [3], зазначено, що високий рівень продуктивності цього фреймворку досягається завдяки використанню власного механізму рендерингу, який дозволяє зменшити залежність від додаткових програмних компонентів та забезпечити ефективне відображення інтерфейсу на різних платформах. Використання мови програмування Dart, яка підтримує різні механізми компіляції та забезпечує ефективне виконання програмного коду також впливає на оптимізацію програмного продукту.

У ході дослідження наукових джерел встановлено, що сучасні мобільні застосунки є ефективним засобом для підтримки фізичної активності користувачів та формування корисних звичок. Також з'ясовано, що використання кросплатформних технологій дозволяє зменшити витрати на розробку програмного забезпечення без суттєвої втрати продуктивності. Результати

аналізу дають підстави розглядати фреймворк Flutter як один із можливих інструментів створення мобільного застосунку призначеного для управління фітнес-орієнтованими завданнями.

Сучасний ринок мобільних застосунків пропонує значну кількість програмних продуктів для планування діяльності, підтримки фізичної активності та формування корисних звичок, тому доцільно буде провести аналіз існуючих рішень майбутньої системи, який дозволить визначити сильні та слабкі сторони вже наявних рішень, оцінити реалізований у них функціонал та виявити їх недоліки, які можна буде врахувати під час розробки програмного продукту. Для проведення аналізу були обрані такі мобільні застосунки як Todoist, Strong та Habitica, оскільки вони мають функції, які пов'язані з керуванням завданнями, організацією фізичної активності та підтримкою мотивації користувачів.

Додаток Todoist зображений на рисунку 1.1 належить до категорії програмного забезпечення для керування завданнями та організації особистої діяльності [4].

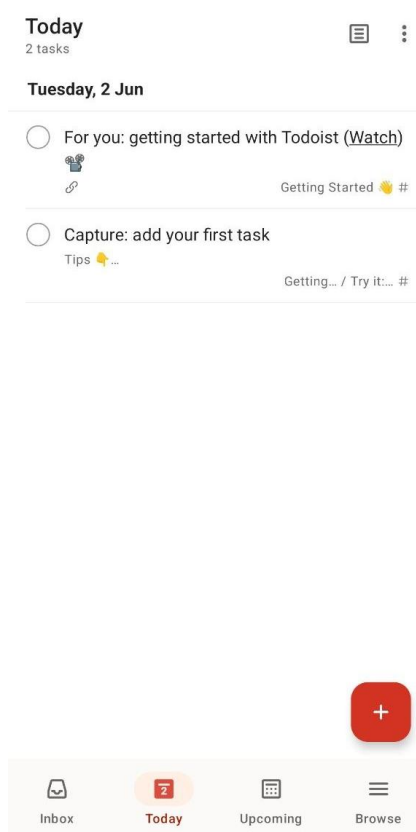


Рисунок 1.1 – Інтерфейс застосунку Todoist [4]

Основним призначенням цього застосунку є створення, редагування та контроль виконання списків справ різного типу. Система дозволяє користувачам створювати окремі завдання, встановлювати для них дати виконання, налаштовувати пріоритети, використовувати нагадування та застосовувати механізми повторення завдань.

До переваг цього застосунку можна віднести наявність простого та інтуїтивно зрозумілого інтерфейсу, підтримку календарного планування, можливість налаштування регулярних подій та зручні механізми сортування задач, а також підтримку роботи на різних платформах, що забезпечує зручність його використання незалежно від операційної системи. Завдяки таким можливостям користувачі можуть ефективно організовувати власну діяльність і контролювати виконання запланованих дій.

Проте застосунок орієнтований переважно на загальне планування та не враховує специфіку спортивних або тренувальних завдань. У ньому відсутня підтримка таких функцій, як збереження кількості повторів вправ, використання таймерів для виконання фізичних вправ, створення шаблонів тренувань та групування окремих завдань у тренувальні комплекси або сети, що обмежує можливість його використання для організації персональних тренувань.

Мобільний застосунок Strong [5], вже орієнтований саме на планування та контроль виконання тренувань. Основою його функціональних можливостей є ведення журналу фізичної активності користувачів та збереження інформації про виконані вправи. Додаток дозволяє виконувати та створювати власні тренувальні програми, додавати вправи, налаштовувати кількість підходів та повторень, відслідковувати тривалість тренування, а також вести історію попередніх тренувань. Використання таких функцій спрощує процес організації та дозволяє підтримувати регулярність тренувань. Головний екран застосунку Strong наведено на рисунку 1.2.

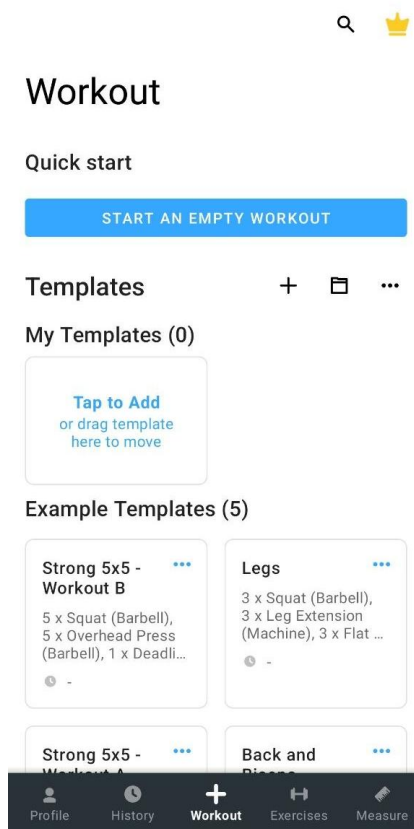


Рисунок 1.2 – Головний екран застосунку Strong [5]

Водночас його вузька спрямованість призводить до певних обмежень у використанні застосунку для планування інших типів діяльності. У системі відсутній повноцінний функціонал для створення універсальних списків завдань, підтримки різних типів подій та календарного планування повсякденних справ. З огляду на це, Strong більше схожий на спеціалізований журнал тренувань, ніж на універсальний застосунок для керування різними особистими завданнями.

Застосунок Habitica [6] поєднує в собі функції планування завдань із механізмами підтримки мотивації користувача (рис. 1.3). Основною особливістю цього програмного продукту є використання елементів гейміфікації, які перетворюють виконання повсякденних завдань на ігровий процес. Користувачі отримують винагороди за виконані дії, можуть підвищувати рівень власного персонажа та відстежувати прогрес виконання поставлених цілей. Використання елементів гри сприяє підвищенню мотивацію до виконання запланованих дій і позитивно впливає на формування нових корисних звичок. Однак використання значної кількості ігрових елементів може створювати певні труднощі під час

практичного застосування системи. Надмірна орієнтація на гейміфікацію може відволікати користувачів від основної функції планування своєї діяльності.

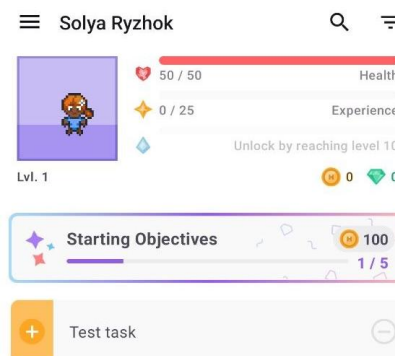


Рисунок 1.3 – Гейміфікований головний екран Habitica [6]

Аналіз існуючих програмних рішень показав, що кожен із розглянутих застосунків орієнтований на реалізацію певних функціональних можливостей, проте жоден не забезпечує комплексного поєднання можливостей класичного менеджера завдань із функціями підтримки фізичної активності в межах одного застосунку.

Виявлені обмеження існуючих програмних рішень вказують на необхідність створення нового мобільного застосунку, який дозволить поєднати функціональні можливості систем керування завданнями та фітнес-орієнтованих інструментів. Реалізація такого рішення надасть користувачам можливість використовувати єдину систему для планування особистої діяльності, організації тренувального процесу та контролю прогресу.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Відповідно до поставленої мети кваліфікаційної роботи бакалавра, а також з урахуванням результатів аналізу предметної області та існуючих програмних рішень визначено перелік завдань, необхідних для розробки мобільного застосунку для управління фітнес-орієнтованими завданнями.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- провести аналіз предметної області, сучасних мобільних технологій та існуючих програмних рішень для визначення вимог до мобільного застосунку;
- спроектувати архітектуру мобільного застосунку з урахуванням особливостей управління фітнес-орієнтованими завданнями;
- реалізувати користувацький інтерфейс засобами фреймворку Flutter із використанням мови програмування Dart;
- розробити механізми створення, редагування, видалення та планування фітнес-орієнтованих завдань із підтримкою пріоритетів, календарного представлення та повторюваних подій засобами Flutter та Dart;
- реалізувати функції створення шаблонів вправ, тренувальних сетів, таймерів та нагадувань із використанням Flutter, а також нативних можливостей операційної системи Android за допомогою мови програмування Kotlin;
- розробити модель бази даних і реалізувати систему локального зберігання та обробки даних із використанням SQLite та бібліотеки Drift для забезпечення автономної роботи застосунку;
- реалізувати функції формування статистики, резервного копіювання та відновлення даних користувача з використанням SQLite та локального сховища мобільного пристрою;
- провести тестування мобільного застосунку для перевірки коректності роботи всіх реалізованих функцій, а також виявити та усунути можливі недоліки.

Таким чином, виконання поставлених завдань забезпечить досягнення мети кваліфікаційної роботи бакалавра та створення мобільного застосунку, який підвищить ефективність планування регулярної фізичної активності.

РОЗДІЛ 2

СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Метою розробки системи є створення мобільного застосунку для управління фітнес-орієнтованими завданнями, який забезпечуватиме можливість планування, контролю та аналізу активності користувача. Застосунок повинен поєднувати функціональні можливості класичних систем керування завданнями із додатковими інструментами, орієнтованими на використання у спортивних сценаріях.

У процесі формування вимог до програмного забезпечення використано декілька методів дослідження, серед яких ключовим став аналіз предметної області. Його застосування дозволило визначити особливості організації процесів планування фізичної активності та керування завданнями. Крім того, проведено дослідження наявних рішень і потреб користувачів, що дало змогу виявити переваги, недоліки й основні функціональні можливості існуючих рішень, а також окреслити найпоширеніші сценарії використання системи та сформулювати перелік функцій, необхідних для реалізації застосунку. До функціональних вимог належать можливості створення, редагування, видалення та налаштування завдань, підтримка календарного планування, використання нагадувань, встановлення повторень, робота з бібліотекою шаблонів вправ, створення тренувальних сетів, формування статистики та виконання резервного копіювання даних. До нефункціональних вимог належать забезпечення зручності використання, швидкодії застосунку, підтримка локального зберігання інформації, стабільність роботи та можливість функціонування без постійного підключення до мережі Інтернет.

Для моделювання програмної системи та її елементів обрано об'єктно-орієнтований підхід із використанням мови моделювання UML [7]. Засоби UML-моделювання дозволяють відображати компоненти системи у вигляді діаграм,

забезпечуючи структурований опис архітектури системи та спрощуючи подальшу реалізацію програмного продукту. Також використання цього підходу дозволяє описати структуру системи, взаємодію її компонентів і поведінку окремих елементів у процесі роботи застосунку, що є доцільним для розроблюваної системи, оскільки функціональність застосунку представлена окремими логічними сутностями, які мають власні властивості та взаємозв'язки.

Розроблюваний застосунок є однокористувацьким мобільним рішенням та не передбачає використання механізмів авторизації або багатокористувацької взаємодії, тому на діаграмі прецедентів (рис. 2.1) представлено тільки одного актора, яким є користувач. Для зручнішого відображення прецеденти об'єднано у функціональні групи відповідно до їхнього призначення.

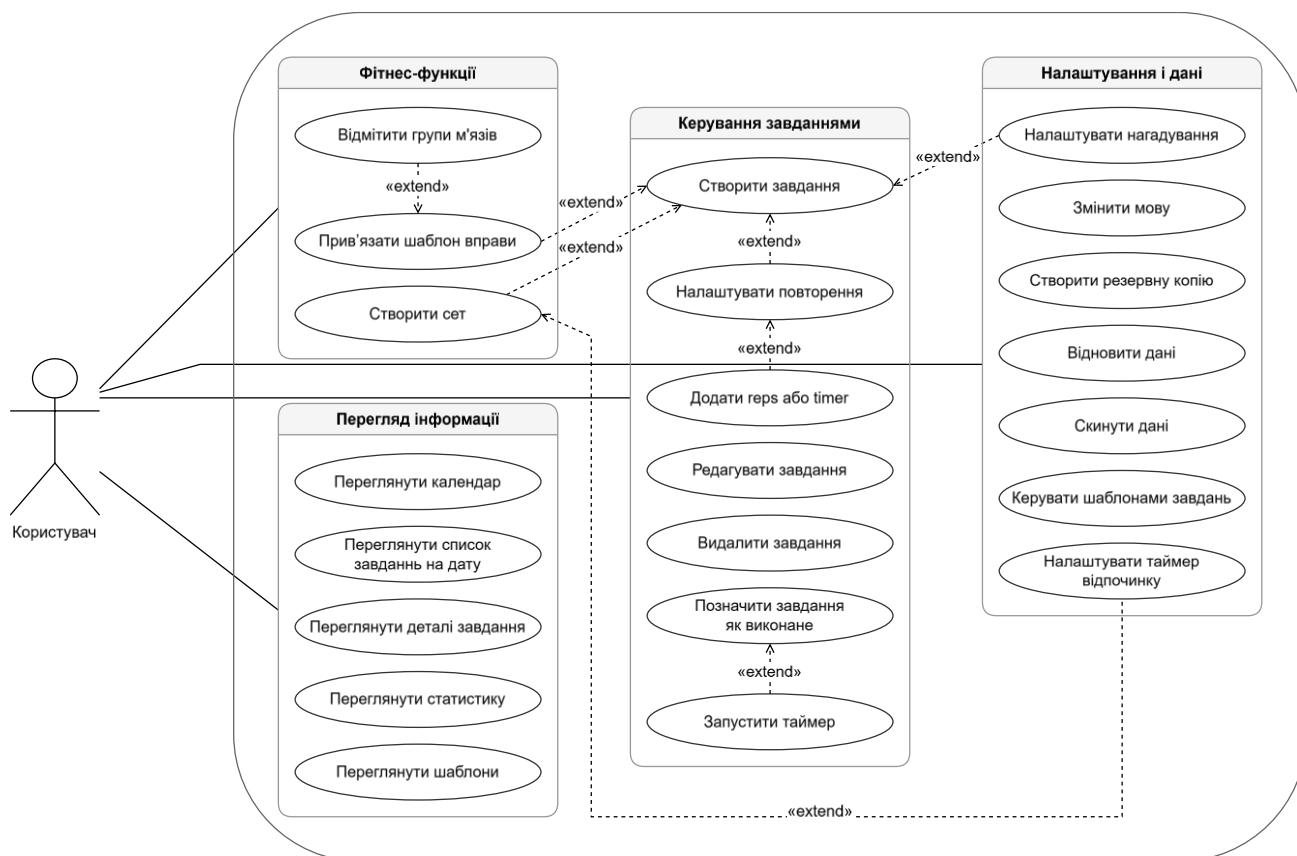


Рисунок 2.1 – Діаграма прецедентів мобільного застосунку

Група «Керування завданнями» реалізує основний функціонал системи, а саме взаємодію із завданнями. До цієї групи входять створення, редагування та

видалення завдань, позначення як виконаних, а також налаштування регулярних подій та додавання додаткових параметрів у вигляді кількості повторів або таймера. Відображено зв'язки між прецедентами, де окремі функції виступають як додаткові сценарії, що розширюють основний процес створення завдання.

Група «Фітнес-функції» містить функціональні можливості, які відрізняють розроблюваний застосунок від класичних систем управління завданнями. До цієї групи належить прив'язка шаблонів вправ до завдань, створення тренувальних сетів та визначення груп м'язів для вправ а також налаштування кількості повторів та таймера виконання безпосередньо в межах завдання. Реалізація цих функцій дозволяє адаптувати систему для використання у спортивних сценаріях та забезпечити додаткові можливості персоналізації тренувального процесу.

Група «Перегляд інформації» забезпечує доступ користувача до інформації, що формується під час роботи системи. До неї належать перегляд календаря, списку завдань, деталей завдання, статистики виконання та бібліотеки шаблонів.

Група «Налаштування і дані» належать функції керування параметрами застосунку та локальними даними користувача, а саме налаштування нагадувань, зміна мови інтерфейсу, створення резервних копій, відновлення та скидання даних, керування шаблонами завдань та налаштування таймера відпочинку. Реалізація цих функцій у майбутній системі підвищить надійність роботи застосунку та покращить зручність його використання.

Для детальнішого опису процесів, які відбуваються під час взаємодії користувача із застосунком, побудовано діаграму діяльності представлену на рисунку 2.2, яка представляє сценарій створення та виконання фітнес-орієнтованого завдання, оскільки саме він охоплює основні функції системи та відображає ключову логіку її роботи.

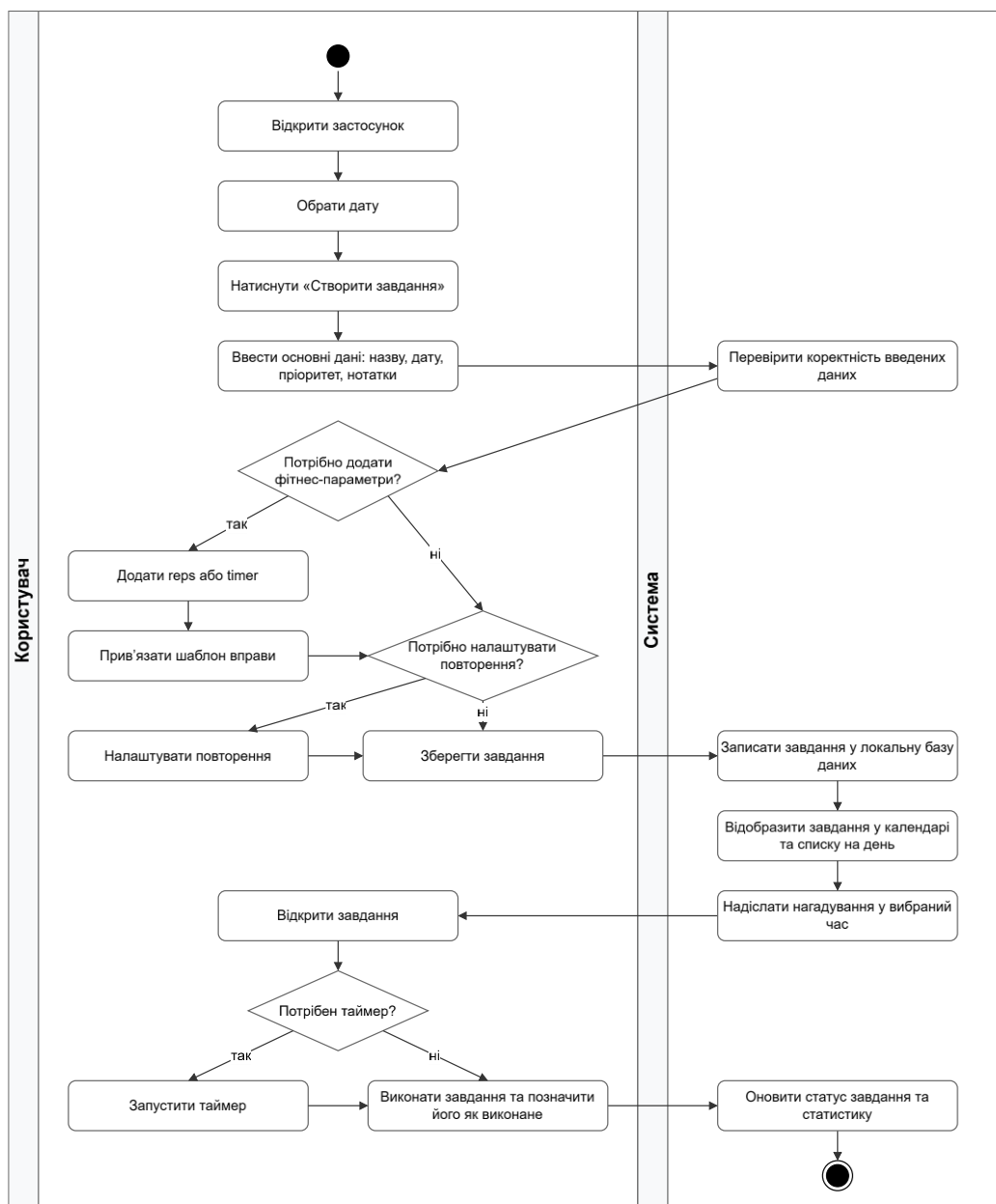


Рисунок 2.2 – Діаграма діяльності створення та виконання фітнес-орієнтованого завдання

Процес роботи починається з відкриття застосунку користувачем та вибору дати, на яку необхідно створити завдання. Після цього користувач переходить до форми створення завдання та вводить основні дані. Система виконує перевірку коректності введеної інформації, що дозволяє уникнути збереження неповних або некоректних даних.

Якщо завдання пов'язане з фізичною активністю, користувач додає кількість повторів або таймер, а також може прив'язати до завдання шаблон

вправи. Якщо такі параметри не потрібні, процес переходить до наступного етапу.

Після заповнення необхідних даних користувач зберігає завдання, а система записує його у локальну базу даних. Зберігши завдання воно з'являється у календарі та списку завдань на відповідну дату. У вибраний час система надсилає нагадування, після чого користувач може відкрити завдання та виконати його.

Завершивши дію користувач позначає завдання як виконане, якщо в завданні передбачено таймер користувач може запустити його і завдання відмітиться автоматично, після чого система оновлює статус завдання та враховує виконання під час формування статистики.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

З огляду на велику кількість засобів для створення мобільних застосунків, які відрізняються архітектурними підходами, функціональними можливостями, продуктивністю та складністю використання, важливим етапом у процесі розробки є вибір технологічного стеку, оскільки саме він визначає можливості реалізації функціоналу, продуктивність системи та ефективність подальшої підтримки програмного забезпечення.

Крім вибору засобів для розробки інтерфейсу та логіки системи, також необхідно визначити технологію для обробки та збереження інформації. Для реалізації мобільних застосунків використовуються локальні та віддалені сховища даних, серед яких можна виділити: Firebase, SharedPreferences та SQLite.

Firebase являє собою хмарну платформу, яка дозволяє організувати зберігання інформації, автентифікацію користувачів та синхронізацію даних між пристроями. Перевагою такого рішення є можливість роботи із віддаленими даними та підтримка багатокористувацьких систем. Водночас використання Firebase потребує постійного підключення до мережі Інтернет та створює

залежність від зовнішньої інфраструктури. SharedPreferences використовується для збереження невеликих обсягів інформації, що є ефективним для зберігання налаштувань застосунку, параметрів інтерфейсу та службових значень. Однак цей інструмент не призначений для роботи зі складними структурованими даними або великою кількістю взаємопов'язаних записів.

SQLite є вбудованою реляційною системою керування базами даних, яка забезпечує структуроване зберігання інформації без необхідності використання віддаленого сервера. До її переваг належать автономність, висока швидкість роботи, підтримка зв'язків між даними та відсутність залежності від мережеских ресурсів, проте вона потребує додаткової організації взаємодії між програмним кодом і базою даних.

На основі проведеного аналізу для розробки системи обрано Flutter як основний засіб створення користувацького інтерфейсу та реалізації логіки застосунку [8], а Kotlin для реалізації функцій, що потребують безпосередньої взаємодії з операційною системою Android [9].

Для організації локального сховища даних обрано SQLite [10] та бібліотеку Drift [11], яка забезпечує зручний механізм взаємодії між програмним кодом і базою даних, спрощуючи створення таблиць, виконання запитів та роботу зі зв'язками між окремими сутностями системи. Для збереження користувацьких параметрів, зокрема часу нагадувань, мови інтерфейсу та інших налаштувань застосунку, буде використовуватись Shared Preferences [12], що дозволяє ефективно зберігати й обробляти структуровані дані та допоміжну інформацію, спрощуючи організацію роботи системи.

Для представлення внутрішньої структури розроблюваного програмного забезпечення побудовано діаграму класів, зображену на рисунку 2.3. Вона містить основні сутності системи, їхні атрибути та взаємозв'язки між ними. До діаграми включено класи завдань, шаблонів вправ, категорій, сетів, повторюваних подій, а також допоміжні класи, необхідні для організації логіки роботи системи. Кожна із наведених сутностей виконує визначену функцію в межах предметної області та бере участь у реалізації основних процесів додатка.

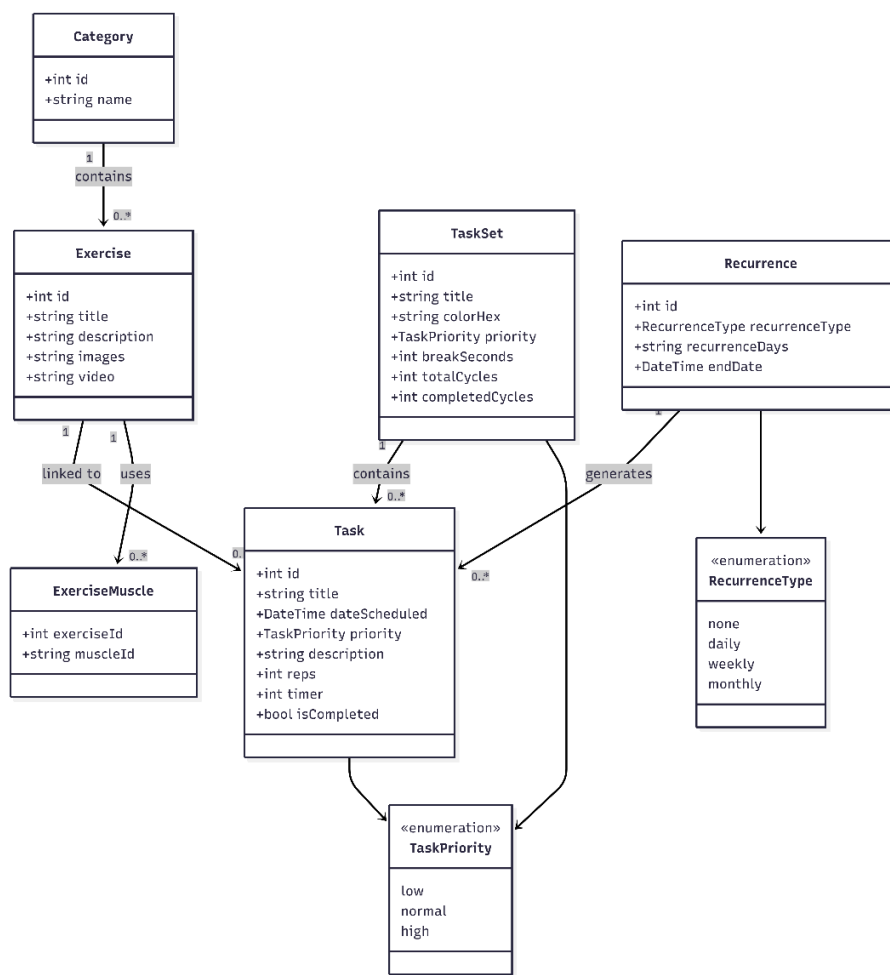


Рисунок 2.3 – Діаграма класів мобільного застосунку для управління фітнес-орієнтованими завданнями

Основною сутністю розроблюваної системи є клас Task, з яким взаємодіє більшість компонентів застосунку. Цей клас використовується для зберігання інформації про завдання і містить основні параметри, зокрема назву, дату виконання, опис, кількість повторів, параметри таймера та статус виконання.

Клас Exercise використовується для зберігання інформації про шаблони вправ та містить їх основні характеристики, зокрема назву, опис, зображення та відеоматеріали. Наявність окремого класу для вправ дозволяє уникнути дублювання даних і забезпечує можливість використання створених шаблонів вправ у різних завданнях.

Клас TaskSet використовується для реалізації тренувальних сетів та дає можливість об'єднати декілька завдань у єдину структуру із визначеною

кількістю циклів та параметрами перерв між ними, що дозволяє організовувати послідовне виконання декількох вправ у межах одного тренувального процесу.

Для підтримки механізму повторюваних завдань створено клас Recurrence, який містить параметри регулярності, тип періодичності та дату завершення повторів. Використання окремого класу дозволяє відокремити логіку циклічних завдань від основної структури даних, що спрощує реалізацію механізму автоматичного створення наступних завдань.

Сформована модель забезпечує логічну організацію даних для подальшої реалізації функцій керування завданнями та шаблонами вправ, підтримки регулярних подій, механізму нагадувань та формування статистичної інформації.

Для опису логіки виконання одного з ключових функціональних процесів застосунку побудовано блок-схему алгоритму роботи створення повторюваних завдань (рис. 2.4).

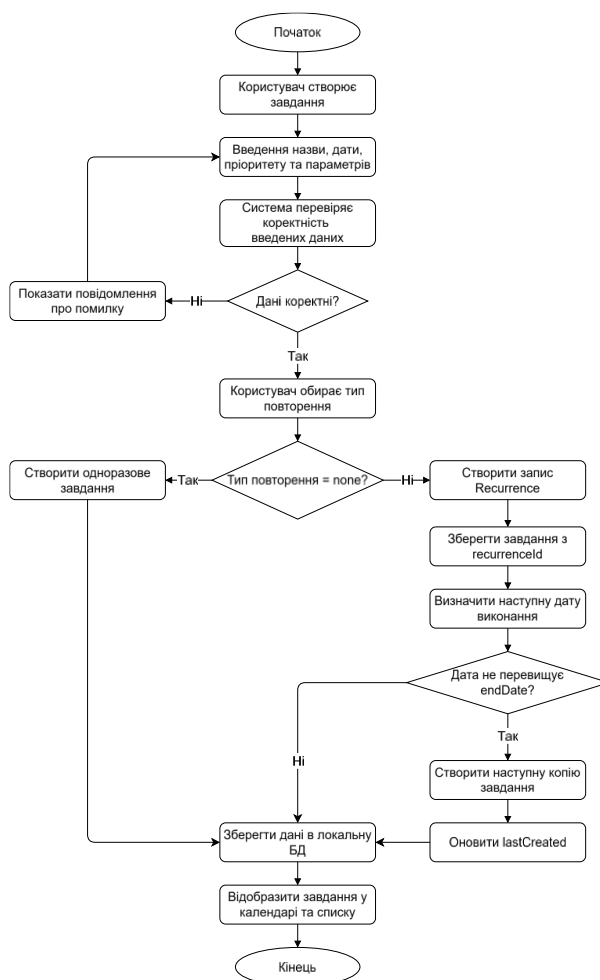


Рисунок 2.4 – Блок-схема алгоритму створення повторюваного завдання

Робота алгоритму починається зі створення користувачем нового завдання та валідацією даних. Якщо користувач не обирає періодичність, система створює звичайне одноразове завдання та зберігає його у локальній базі даних. Якщо для завдання встановлено параметри циклічності, система створює відповідний запис із параметрами багаторазової події та визначає правила формування наступних екземплярів завдання.

Після збереження параметрів система визначає наступну дату виконання завдання та виконує перевірку умов для завершення повторення. Якщо дата не перевищує встановленого користувачем обмеження, система створює наступну копію завдання та оновлює інформацію, необхідну для подальшої роботи алгоритму. Після завершення процесу інформація зберігається у локальній базі даних та відображається у календарі й списку завдань.

Діаграму компонентів наведену на рисунку 2.5 побудовано відповідно до багаторівневого підходу організації системи, який передбачає поділ програмного забезпечення на окремі логічні рівні, кожен із яких виконує визначений набір функцій, що забезпечує зменшення залежностей між окремими складовими та спрощує підтримку й подальше розширення функціональних можливостей.

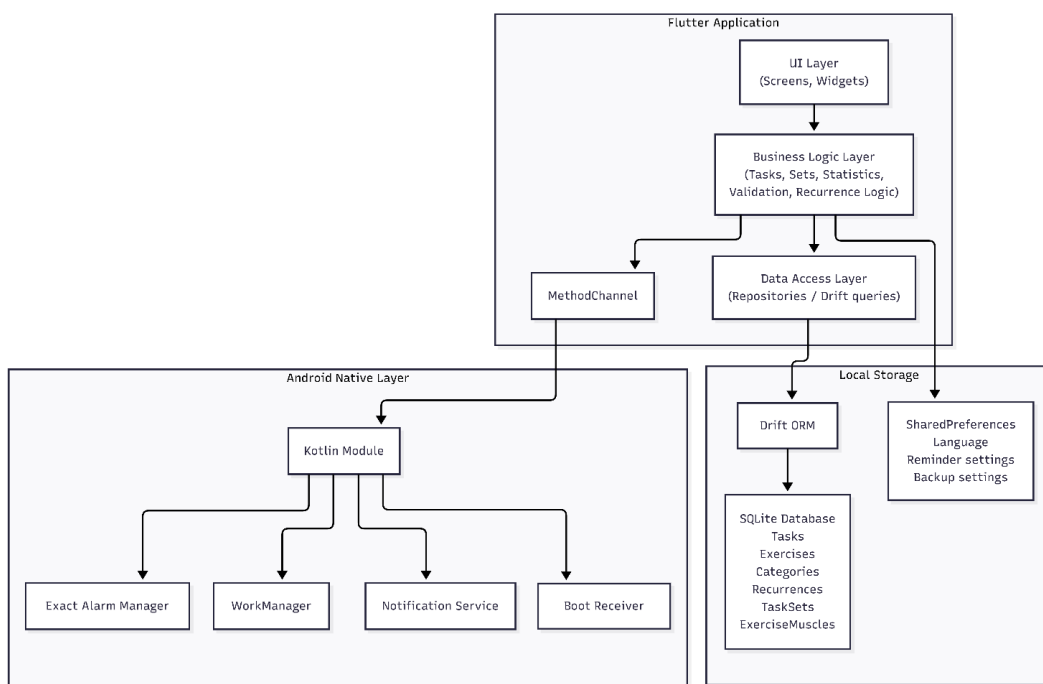


Рисунок 2.5 – Діаграма компонентів мобільного застосунку

На діаграмі представлено три основні групи компонентів системи: Flutter Application, Local Storage та Android Native Layer, кожна з яких містить окремі складові застосунку та визначає механізми взаємодії між ними.

Група компонентів Flutter Application це основна частина програмного забезпечення, яка містить елементи інтерфейсу користувача, бізнес-логіку застосунку та механізми доступу до даних. Компонент UI Layer відповідає за відображення інформації та взаємодію користувача із системою через екрани застосунку. Business Logic Layer забезпечує реалізацію основних функціональних можливостей, зокрема створення та обробку завдань, роботу з повтореннями, перевірку введених даних та формування статистики. Компонент Data Access Layer використовується для взаємодії із локальним сховищем даних та виконання операцій читання, зміни й збереження інформації.

Група компонентів Local Storage використовується для організації локального зберігання інформації. Компонент Drift ORM забезпечує взаємодію між програмним кодом та базою даних SQLite і використовується для виконання запитів та роботи зі структурованими даними. SQLite Database містить основні таблиці застосунку. SharedPreferences використовується для зберігання службових параметрів застосунку, зокрема налаштувань інтерфейсу та параметрів нагадувань.

Група Android Native Layer відповідає за безпосередню взаємодію із операційною системою Android. Компонент Kotlin Module забезпечує реалізацію нативних функцій та взаємодію із Flutter через модуль MethodChannel. Сюди також входять Exact Alarm Manager, який використовується для створення точних нагадувань, WorkManager, що забезпечує виконання фонових процесів, Notification Service, який відповідає за відображення системних повідомлень, та Boot Receiver, який використовується для відновлення роботи нагадувань після перезавантаження пристрою.

РОЗДІЛ 3

РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ФІТНЕС-ОРІЄНТОВАНИМИ ЗАВДАННЯМИ ВИКОРИСТОВУЮЧИ FLUTTER ТА KOTLIN

3.1 Практична реалізація об'єкта проєктування

Розробка мобільного застосунку здійснювалася в середовищі Android Studio [13] з використанням кросплатформного фреймворку Flutter та мови програмування Dart. Основна частина програмного забезпечення, зокрема користувацький інтерфейс, бізнес-логіка та механізми керування даними, реалізована засобами Flutter. Для впровадження функцій, що потребують безпосередньої взаємодії з можливостями операційної системи Android, використано мову програмування Kotlin.

Під час створення програмного забезпечення застосовано компонентний підхід, відповідно до якого система поділена на окремі функціональні модулі. Така організація сприяє повторному використанню коду, спрощує супровід програмного забезпечення та забезпечує незалежність окремих модулів системи.

Кожен модуль відповідає за реалізацію окремого набору функцій та взаємодіє з іншими компонентами через визначені інтерфейси, що дозволяє зменшити залежність між частинами системи та спрощує процес тестування й налагодження програмного забезпечення. Крім того, такий підхід забезпечує можливість подальшого розширення функціональності без суттєвих змін у вже реалізованих компонентах.

Для підвищення зрозумілості коду та спрощення подальшого розширення функціональності структура проєкту була організована відповідно до призначення його компонентів. Логічне розділення модулів полегшує навігацію між файлами та сприяє ефективній підтримці й розвитку програмного продукту. Структуру проєкту мобільного застосунку наведено на рисунку 3.1.

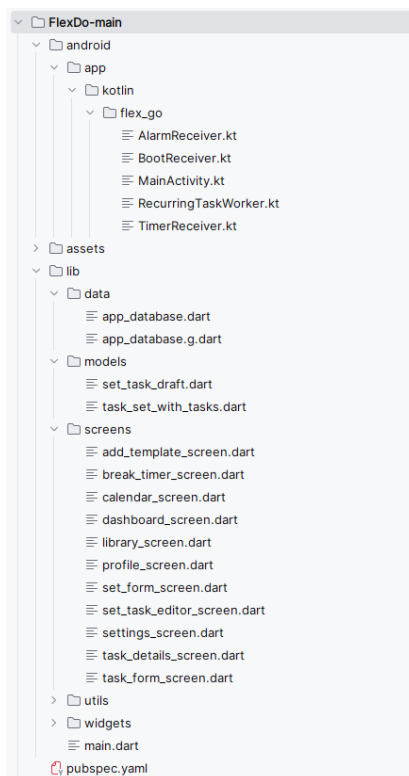


Рисунок 3.1 – Структура проєкту мобільного застосунку

Основна частина програмного коду розташована в директорії `lib`. Файл `main.dart` використовується як точка входу в застосунок та забезпечує початкову ініціалізацію компонентів системи. Тека `data` відповідає за роботу із локальною базою даних та містить файли реалізації сховища даних. Зокрема, файл `app_database.dart` використовується для опису структури бази даних і взаємодії із таблицями, а `app_database.g.dart` містить автоматично згенерований програмний код бібліотеки `Drift`. У директорії `models` розміщено моделі даних, які використовуються для представлення окремих сутностей системи та забезпечують взаємодію між різними компонентами застосунку.

Папка `screens` містить екрани користувацького інтерфейсу, що відповідають за реалізацію функціональних можливостей застосунку. Для повторного використання окремих компонентів інтерфейсу використовується директорія `widgets`, а допоміжні функції та службова логіка системи реалізовані у папці `utils`.

Для реалізації функцій, які потребують взаємодії з операційною системою `Android`, використовується нативна частина, розміщена у директорії `android/app/kotlin`, де реалізовано механізми створення системних нагадувань,

фонового створення завдань, відновлення запланованих подій після перезавантаження пристрою та роботи із системними таймерами. Використання цієї структури дозволяє забезпечити організацію програмного коду, зменшити взаємозалежність між окремими компонентами системи та спростити процес подальшої підтримки програмного забезпечення.

Одним із перших етапів практичної реалізації мобільного застосунку стало створення точки входу, через яку здійснюється запуск програмного забезпечення та початкова підготовка середовища до роботи. Точкою входу є функція `main()`, яка виконується одразу після запуску застосунку та забезпечує ініціалізацію середовища Flutter (ліст. 3.1). Виконання цих операцій в одному місці спрощує керування процесом запуску та забезпечує підготовку всіх необхідних служб перед початком роботи застосунку.

Лістинг 3.1 – Реалізація точки входу та початкової ініціалізації застосунку

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await EasyLocalization.ensureInitialized();
  final notificationsPlugin = FlutterLocalNotificationsPlugin();
  final db = AppDatabase();
  await db.ensureOtherCategoryExists();
  await runSportModeAutoEnableOnce(db);
  notificationsPlugin.resolvePlatformSpecificImplementation
<AndroidFlutterLocalNotificationsPlugin>().requestNotificationsPe
rmission();
  tz.initializeTimeZones();
  runApp(
    EasyLocalization(
      supportedLocales: const [Locale('en', 'US'), Locale('uk',
'UA')],
      path: 'assets/translations',
      fallbackLocale: const Locale('en', 'US'),
      child: const MyApp(),
    ),
  );
}
```

кінець лістингу 3.1

У наведеному фрагменті коду використовується метод `WidgetsFlutterBinding.ensureInitialized()`, який забезпечує ініціалізацію компонентів Flutter до

виконання асинхронних операцій. Далі здійснюється підготовка механізмів локалізації через бібліотеку EasyLocalization, створюється екземпляр бази даних та виконуються службові операції. Також в межах наведеної функції реалізовано отримання дозволу на використання системних сповіщень та часових зон, необхідних для коректної роботи механізму нагадувань.

Після успішного виконання початкових операцій здійснюється запуск основного компонента програмного забезпечення через функцію runApp(), яка передає керування головному віджету застосунку, реалізованому в класі MyApp. Віджет відповідає за налаштування параметрів інтерфейсу та запуск головного екрана. Код реалізації даного компонента представлено в лістингу 3.2.

Лістинг 3.2 – Реалізація головного компонента застосунку

```
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      localizationsDelegates: context.localizationDelegates,
      supportedLocales: context.supportedLocales,
      locale: context.locale,
      title: 'Fitness App'.tr(),
      theme: ThemeData.dark(),
      home: const HomeScreen(),
    );
  }
}
```

кінець лістингу 3.2

Наведений фрагмент демонструє створення головного віджета із використанням компонента MaterialApp [14] у межах якого задається тема оформлення та встановлюється головний екран застосунку.

Після реалізації механізму запуску застосунку реалізовано функціонал для створення та збереження завдань користувача, який забезпечує введення інформації про заплановані активності, перевірку коректності введених даних, формування завдання та його подальше збереження. Процес створення

починається з використання форми для введення даних, яка дозволяє користувачу вказати основні параметри майбутнього завдання (рис. 3.2).

Рисунок 3.2 – Форма створення нового завдання

У формі передбачено введення назви завдання, опису, вибору дати виконання, встановлення рівня пріоритету та налаштування додаткових параметрів, пов'язаних із фізичною активністю, а саме кількості повторів, тривалості таймера та використання шаблонів вправ. Після введення користувачем певної інформації система виконує перевірку наявності обов'язкових полів, обробляє отримані дані та формує об'єкт завдання для подальшого збереження в базі даних (ліст. 3.3).

Лістинг 3.3 – Метод формування завдання в системі

```
final taskId = await db.insertTask(
  TasksCompanion(
    title: drift.Value(_titleController.text),
    priority: drift.Value(_priority),
    reps: _reps > 0
      ? drift.Value(_reps)
      : const drift.Value.absent(),
    timer: totalSeconds > 0
      ? drift.Value(totalSeconds)
```

```

        : const drift.Value.absent(),
dateScheduled: drift.Value(_selectedDate),
createdAt: drift.Value(DateTime.now()),
description:
    _descriptionController.text.trim().isEmpty
        ? drift.Value(
            _descriptionController.text.trim())
        : const drift.Value.absent(),),
);

```

кінець лістингу 3.3

Наведений фрагмент коду реалізує підготовку даних перед їх збереженням у локальній базі даних за допомогою бібліотеки Drift. Значення, введені користувачем через елементи інтерфейсу, використовуються для формування об'єкта типу `TasksCompanion`, який містить усі необхідні атрибути завдання. Для необов'язкових полів застосовується механізм `Value.absent()`, що дозволяє не передавати до бази даних параметри, для яких не було введено значення.

Після успішного формування об'єкта завдання виконується його передача до методу, який відповідає за збереження нового запису. Реалізація методу збереження завдання наведена в лістингу 3.4.

Лістинг 3.4 – Метод збереження завдання в базі даних

```

Future<int> insertTask(
    TasksCompanion task) {
    return into(tasks).insert(task);
}

```

кінець лістингу 3.4

Наведений код демонструє реалізований метод збереження нового завдання. Метод `insertTask()` приймає об'єкт типу `TasksCompanion` та виконує його додавання до таблиці завдань за допомогою засобів бібліотеки Drift. У результаті його виконання в систему повертається ідентифікатор створеного запису, який може використовуватися для подальшої роботи із завданням.

Після реалізації функціональної логіки створення і збереження завдань у застосунку впроваджено підтримку повторюваних завдань. Це є особливо

важливим для розробленого застосунку, оскільки значна частина тренувань виконується за визначеним графіком у конкретні дні тижня або через певні проміжки часу.

Під час створення завдання користувач може визначити тип повторення, який встановлює періодичність появи нових записів. У застосунку реалізовано підтримку щоденного, щотижневого та щомісячного повторення. Також передбачено можливість встановлення дати завершення повторів, після досягнення якої система припиняє створення нових завдань.

Для налаштування параметрів повторення використовується окрема форма під час створення завдання, де користувач може визначити тип повторення та встановити додаткові параметри (рис. 3.3).

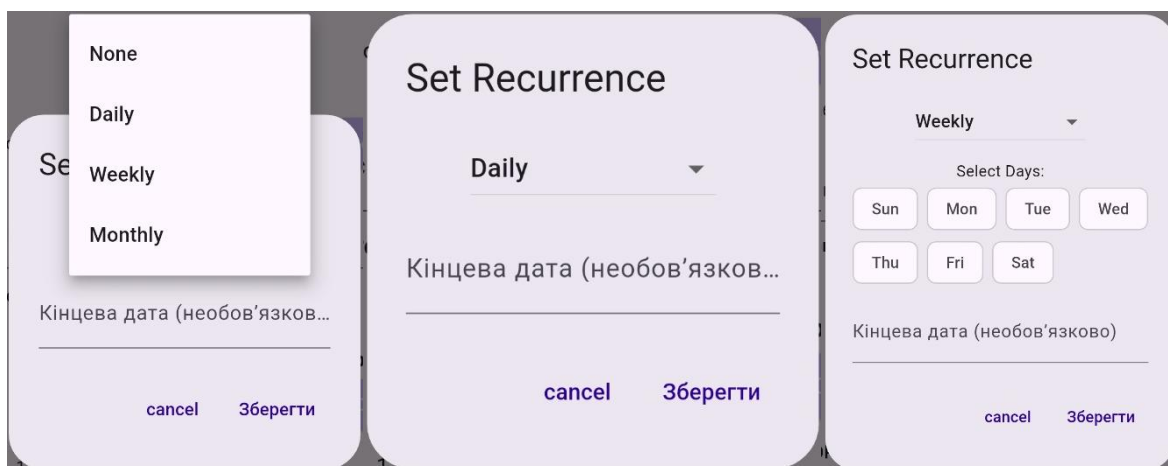


Рисунок 3.3 – Форма для налаштування параметрів повторення завдання

Для підтримки повторюваних завдань у системі реалізовано окремий компонент, який виконується у фоновому режимі та періодично перевіряє необхідність створення нових записів. Під час роботи система визначає поточну дату, аналізує параметри повторення та виконує перевірку умов формування наступного завдання (ліст. 3.5).

Лістинг 3.5 – Визначення умов для створення повторюваного завдання

```
private fun shouldCreateTaskToday(
    recurrenceType: String,
```

```

        recurrenceDaysStr: String?,
        lastDate: Calendar,
        today: Calendar
    ): Boolean {

        return when (recurrenceType) {
            "daily" -> !today.before(lastDate.apply {
add(Calendar.DAY_OF_MONTH, 1) })
            "weekly" -> {
                val recurrenceDays = recurrenceDaysStr?.split(",")?.map {
it.trim().toInt() }

                recurrenceDays?.contains((today.get(Calendar.DAY_OF_WEEK) -
1) % 7) ?: false
            }
            "monthly" -> today.get(Calendar.DAY_OF_MONTH) ==
lastDate.get(Calendar.DAY_OF_MONTH)
            else -> false
        }
    }
}

```

кінець лістингу 3.5

Наведений метод перевіряє тип повторення та визначає, чи необхідно створювати нове завдання на поточну дату. Для щоденного режиму виконується перевірка наступного дня після останнього створеного запису, для щотижневого повторення аналізуються обрані користувачем дні тижня, а для щомісячного режиму виконується порівняння дати поточного та попереднього запису.

Після виконання перевірки система формує та зберігає новий екземпляр завдання з використанням параметрів початкового запису, що дозволяє уникнути повторного введення інформації користувачем. Функція створення нового екземпляра повторюваного завдання наведена в лістингу 3.6.

Лістинг 3.6 – Створення нового екземпляра повторюваного завдання

```

private fun createTask(
    db: SQLiteDatabase,
    recurrenceId: Int,
    title: String,
    priority: String,
    timestamp: Calendar
) {
    val values = ContentValues().apply {
        put("recurrence_id", recurrenceId)
        put("title", title)
    }
}

```

```

        put("priority", priority)
        put("date_scheduled",
            timestamp.timeInMillis / 1000)
        put("is_completed", 0)
    }

    db.insert("tasks", null, values)
}

```

кінець лістингу 3.6

Під час розробки мобільного застосунку окремі функції, зокрема робота з таймерами, системними сповіщеннями та дозволами на використання будильників, потребували безпосередньої взаємодії з операційною системою Android. Для забезпечення взаємодії та обміну даними між Dart-кодом і нативною частиною Android використано механізм MethodChannel [15].

MethodChannel дозволяє передавати виклики з Flutter до Kotlin-коду та отримувати результат їх виконання. У застосунку цей механізм використовується для запуску та відображення таймера. Виклик нативного методу наведено в лістингу 3.7.

Лістинг 3.7 – Виклик MethodChannel із Flutter

```

static const _ch =
    MethodChannel('com.example.notification');

static Future<void> schedule({
    required int requestId,
    required int taskId,
    required DateTime endAt,
    String title = 'Timer',
}) async {
    await _ch.invokeMethod(
        'scheduleTimerExact',
        {
            'epochMillis':
                endAt.millisecondsSinceEpoch,
            'id': requestId,
            'title': title,
            'taskId': taskId,
        },
    );
}

```

кінець лістингу 3.7

У наведеному коді метод `schedule()` отримує ідентифікатор запиту, ідентифікатор завдання, час завершення таймера та назву сповіщення. Далі ці параметри передаються до Kotlin-частини за допомогою методу `invokeMethod()`. Для часу завершення використовується значення `millisecondsSinceEpoch`, що дозволяє передати момент спрацювання таймера у вигляді числового значення.

На стороні Android виклик обробляється у файлі `MainActivity.kt`. Для цього у методі `configureFlutterEngine()` створюється обробник `MethodChannel`, який аналізує назву викликаного методу та виконує відповідну нативну функцію. Реалізацію обробки виклику наведено в лістингу 3.8.

Лістинг 3.8 – Обробка виклику `MethodChannel` у Kotlin

```
MethodChannel(flutterEngine.dartExecutor.binaryMessenger,
CHANNEL).setMethodCallHandler {
    call,
    result ->
    when (call.method) {
        "scheduleTimerExact" -> {
            val epochMillis = call.argument<Long>("epochMillis")!!

            val id = call.argument<Int>("id") ?: 1000

            val title = call.argument<String>("title") ?: "Timer"
            val taskId = call.argument<Int>("taskId") ?: -1
            scheduleTimerExact(epochMillis, id, title, taskId)
            result.success(null)
        }
        else -> result.notImplemented()
    }
}
```

кінець лістингу 3.8

Наведений код демонструє, що Kotlin-частина отримує параметри, передані з Flutter, та викликає метод `scheduleTimerExact()`, тоді система отримує час спрацювання, ідентифікатор таймера, назву та ідентифікатор завдання, після чого передає ці дані до Android.

Для підвищення зручності використання мобільного застосунку реалізовано механізм щоденних нагадувань. Його основне призначення полягає у тому, щоб повідомляти користувача про наявність невиконаних завдань,

запланованих на поточний день, що дозволяє підтримувати регулярність виконання завдань (рис. 3.4).

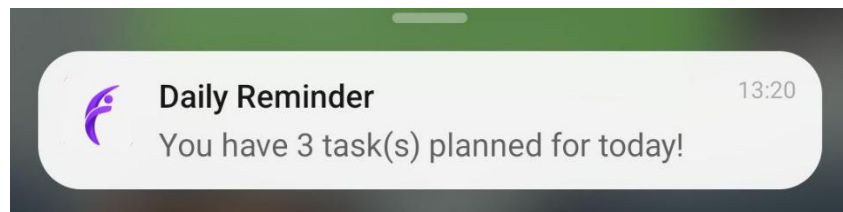


Рисунок 3.4 – Нагадування про заплановані завдання

Механізм нагадувань реалізовано з використанням системного сервісу `AlarmManager`, який дозволяє планувати виконання подій у визначений користувачем час. Під час створення нагадування система формує об'єкт `PendingIntent`, пов'язаний із класом `AlarmReceiver`, який спрацює після настання запланованого часу. Фрагмент коду, який забезпечує планування щоденних нагадувань, наведено в лістингу 3.9.

Лістинг 3.9 – Планування щоденних нагадувань

```
private fun scheduleDailyNotification(hour: Int, minute: Int) {
    val now = System.currentTimeMillis()
    val calendar = java.util.Calendar.getInstance().apply {
        timeInMillis = now
        set(java.util.Calendar.HOUR_OF_DAY, hour)
        set(java.util.Calendar.MINUTE, minute)
        set(java.util.Calendar.SECOND, 0)
        set(java.util.Calendar.MILLISECOND, 0)
    }

    if (calendar.timeInMillis <= now) {
        calendar.add(java.util.Calendar.DAY_OF_MONTH, 1)
    }
    val intent = Intent(this, AlarmReceiver::class.java)
    val pendingIntent = PendingIntent.getBroadcast(this, 0, intent,
        PendingIntent.FLAG_UPDATE_CURRENT or PendingIntent.FLAG_IMMUTABLE)
    alarmManager?.setExactAndAllowWhileIdle(
        AlarmManager.RTC_WAKEUP,
        calendar.timeInMillis,
        pendingIntent
    )
}
```

кінець лістингу 3.9

Під час виконання наведеної функції формується об'єкт `Calendar`, у якому задається час спрацювання нагадування. Якщо вибраний час уже минув у поточний день, нагадування автоматично переноситься на наступний день. Після цього створюється `Intent`, який вказує на клас `AlarmReceiver`, та формується `PendingIntent`, що передається до `AlarmManager.setExactAndAllowWhileIdle()` використовується для точного планування сповіщення навіть у режимі енергозбереження пристрою, що дозволяє користувачу своєчасно отримувати нагадування незалежно від поточного стану роботи мобільного пристрою.

Після спрацювання запланованого нагадування керування передається до класу `AlarmReceiver`. Даний компонент відкриває локальну базу даних, визначає кількість невиконаних завдань на поточну дату і якщо такі завдання існують, формує сповіщення для користувача за допомогою `NotificationCompat.Builder` [16], фрагмент коду наведено в лістингу 3.10.

Лістинг 3.10 – Формування системного сповіщення про невиконані завдання

```

if (count > 0) {
    val builder =
        NotificationCompat.Builder(
            context,
            "daily_reminder_channel"
        )
        .setSmallIcon(R.mipmap.ic_launcher)
        .setContentTitle("Daily Reminder")
        .setContentText(
            "You have $count task(s) planned for today!"
        )
        .setPriority(NotificationCompat.PRIORITY_HIGH)
        .setAutoCancel(true)
    val notificationManager = NotificationManagerCompat.from(context)
    notificationManager.notify(1, builder.build())
}

```

кінець лістингу 3.10

Для повідомлення використовується канал `daily_reminder_channel`, який призначений для щоденних нагадувань. Текст сповіщення містить кількість невиконаних завдань, запланованих на поточний день, що дозволяє користувачу

отримати інформацію про наявність невиконаних активностей. Якщо невиконаних завдань немає, повідомлення не відображається.

Крім показу повідомлення, у класі `AlarmReceiver` реалізовано повторне планування нагадування на наступний день. Для цього після спрацювання поточного нагадування система перевіряє, чи увімкнена відповідна функція у налаштуваннях користувача, отримує збережені параметри часу та знову створює запланований системний будильник, що дозволяє забезпечити регулярне надсилення нагадувань без необхідності повторного ручного налаштування.

Окремо реалізовано відновлення нагадувань після перезавантаження мобільного пристрою. Для цього використовується клас `BootReceiver`, який реагує на системну подію `BOOT_COMPLETED`. Після запуску пристрою даний компонент перевіряє збережені налаштування застосунку і якщо нагадування були увімкнені, ініціює їх повторне планування, що дозволяє зберегти коректну роботу нагадувань навіть після вимкнення або перезавантаження пристрою.

Також у застосунку створено окремий тип сповіщення про завершення таймера виконання завдання. Для цього використовується клас `TimerReceiver`, який спрацьовує після завершення заданого проміжку часу, оновлює статус відповідного завдання та формує системне повідомлення про завершення його виконання. Завдяки цьому користувач автоматично отримує інформацію про завершення завдань із часовими обмеженнями.

3.2 Тестування та налагодження інформаційно-комп'ютерної системи

Важливим етапом розробки мобільного застосунку є проведення тестування та налагодження програмного забезпечення. Тестування дозволяє перевірити правильність реалізації функціональних можливостей програмного продукту, виявити помилки у роботі окремих компонентів та оцінити зручність використання застосунку. Налагодження програмного забезпечення забезпечує усунення виявлених недоліків і підвищує стабільність роботи системи. Для розроблюваного мобільного застосунку тестування проводилося протягом усього

процесу розробки з метою перевірки коректності роботи нових функціональних можливостей, що дозволяло своєчасно виявляти та усувати помилки.

Для перевірки працездатності застосунку використовувалися методи функціонального та інтеграційного тестування, де основна увага приділялася перевірці правильності роботи окремих функціональних модулів, а також коректності їх взаємодії між собою.

Основним середовищем тестування була інтегрована система розробки Android Studio, яка дозволяє використовувати емулятори мобільних пристроїв із різними версіями операційної системи Android (рис. 3.5). Завдяки цьому була забезпечена можливість перевірки роботи застосунку в різних програмних середовищах без необхідності використання великої кількості фізичних пристроїв. Додатково застосунок тестувався на реальному мобільному пристрої, що дозволило оцінити його роботу в умовах реальної експлуатації.

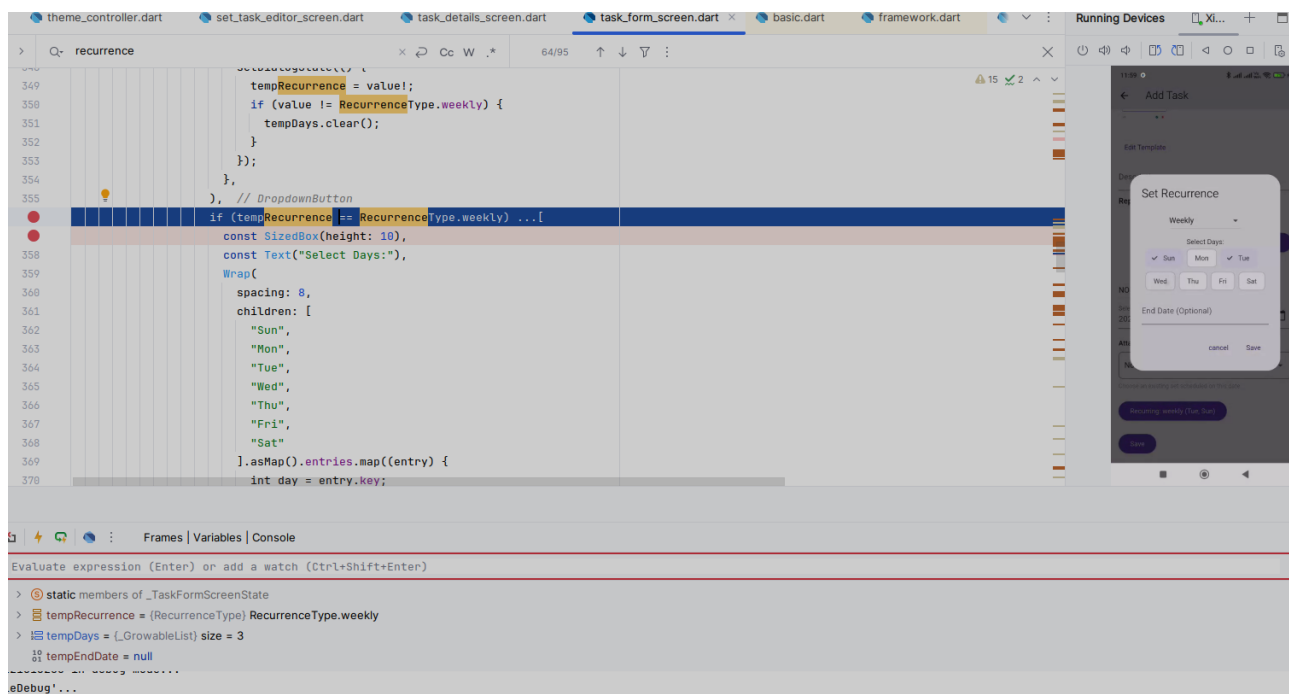


Рисунок 3.5 – Тестування мобільного застосунку

У процесі тестування перевірялися основні функціональні можливості системи, зокрема створення, редагування та видалення завдань, робота повторюваних подій, формування тренувальних сетів, використання таймерів,

робота нагадувань, створення резервних копій даних, зміна налаштувань застосунку та механізми локалізації інтерфейсу.

Крім технічного тестування, проводилася перевірка зручності використання інтерфейсу користувача. Для цього застосунок використовувався у реальних сценаріях планування повсякденних та спортивних активностей, що дозволило оцінити зрозумілість навігації, логіку розташування елементів та зручність виконання основних операцій. Також застосунок тестувався іншими користувачами, зауваження яких дозволили виявити окремі недоліки системи та інтерфейсу й внести відповідні покращення. Усі знайдені помилки проаналізовано та усунено в процесі налагодження програмного забезпечення.

Однією з найсуттєвіших помилок, виявлених під час тестування, була проблема в роботі механізму повторюваних завдань. Встановлено, що після позначення одного з них як виконаного система автоматично відображала виконаними всі екземпляри цього завдання, через що користувач втрачав можливість коректно відстежувати виконання повторюваних активностей. Приклад прояву помилки та результат виправлення наведено на рисунку 3.6.

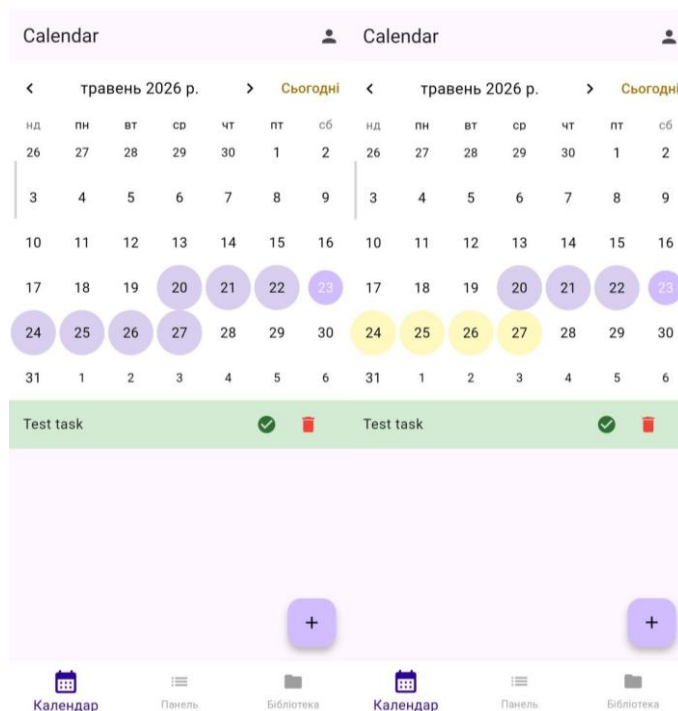


Рисунок 3.6 – Відображення повторюваних завдань до та після виправлення помилки

Причиною проблеми була особливість логіки обробки повторюваних записів, через яку зміна статусу виконання застосовувалася до всієї групи пов'язаних завдань замість окремого екземпляра. Для усунення недоліку впроваджено додаткову перевірку дати виконання та належності завдання до конкретного екземпляра повторюваної події. Після внесення змін система почала змінювати статус лише того завдання, яке фактично виконане користувачем. Це дозволило усунути спотворення інформації про стан запланованих завдань та забезпечити коректну роботу механізму повторення.

Ще один недолік застосунку був виявлений під час перевірки інтерфейсу. Проблема стосувалася відображення блоку з поточною датою у верхній частині екрана. Через особливості побудови інтерфейсу елемент центрувався відносно контейнера, у якому знаходився, а не відносно всієї ширини екрана. У поєднанні з іконкою профілю користувача це створювало враження візуального зміщення заголовка та погіршувало сприйняття інтерфейсу. Для усунення даного недоліку змінено логіку розташування елементів верхньої панелі та реалізовано коректне центрування блоку поточної дати відносно екрана. Після внесення змін інтерфейс став більш симетричним і візуально збалансованим. Приклад відображення верхньої панелі до та після виправлення наведено на рисунку 3.7.

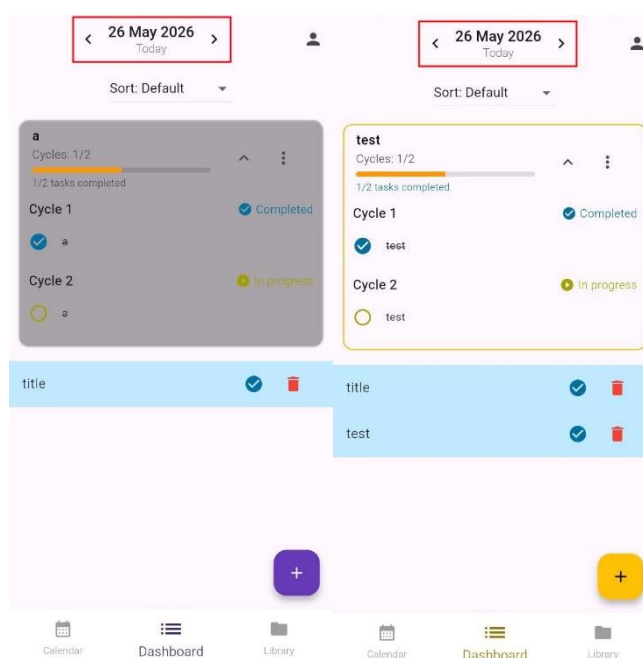


Рисунок 3.7 – Відображення верхньої панелі до та після виправлення

Також під час проведення тестування на різних мобільних пристроях виявлено проблему з автоматичним визначенням теми оформлення інтерфейсу. Початкова реалізація застосунку передбачала автоматичне використання системної теми пристрою користувача, проте на окремих версіях операційної системи це призводило до некоректного поєднання кольорових схем. У результаті частина елементів відображалася у світлих кольорах, тоді як інші компоненти залишалися темними, що негативно впливало на зовнішній вигляд застосунку.

Після аналізу проблеми прийнято рішення впровадити механізм ручного вибору між світлою та темною темами замість автоматичного визначення, що дозволило забезпечити цілісність інтерфейсу та стабільне відображення всіх елементів незалежно від налаштувань операційної системи. Приклад відображення інтерфейсу до та після впровадження ручного вибору теми наведено на рисунку 3.8.

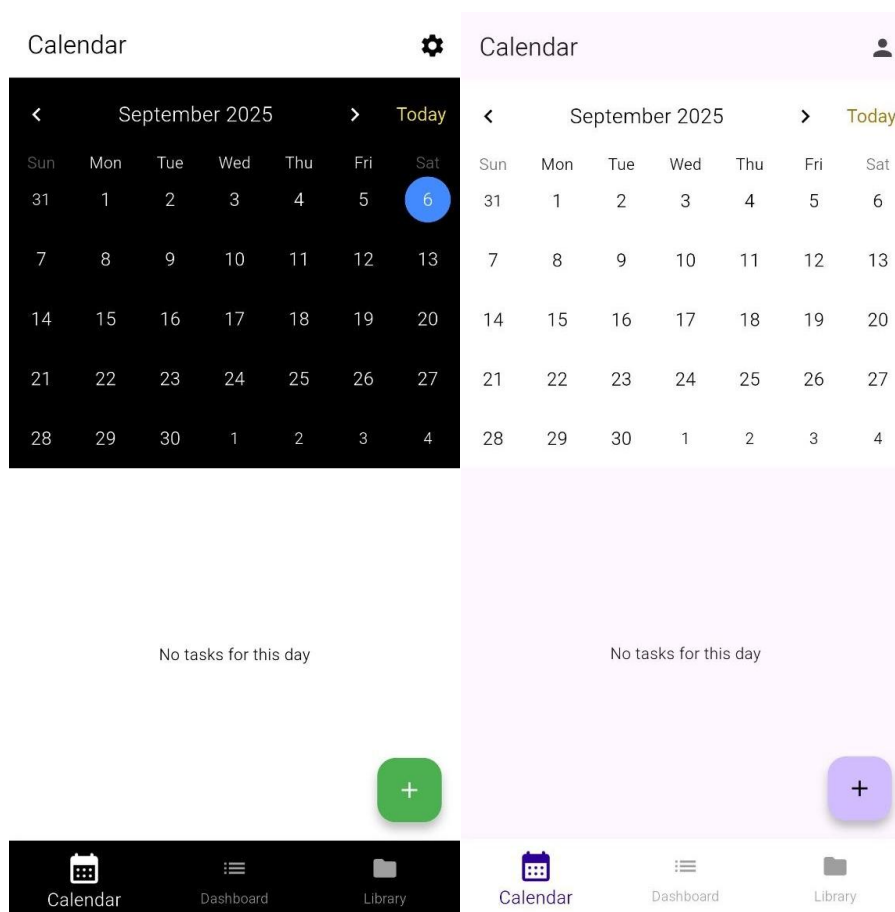


Рисунок 3.8 – Відображення інтерфейсу до та після впровадження ручного вибору теми

3.3 Розробка бази даних мобільного застосунку

Для збереження інформації у мобільному застосунку реалізовано локальну базу даних. Оскільки система не використовує серверну частину та працює за принципом автономного зберігання даних, основним сховищем інформації виступає SQLite. Для взаємодії з базою даних у Flutter-застосунку використовується бібліотека Drift, яка дозволяє описувати структуру таблиць засобами мови Dart та виконувати типізовані запити до локального сховища. База даних складається з кількох взаємопов'язаних таблиць, що забезпечують зберігання та обробку інформації, необхідної для роботи застосунку (рис. 3.9).

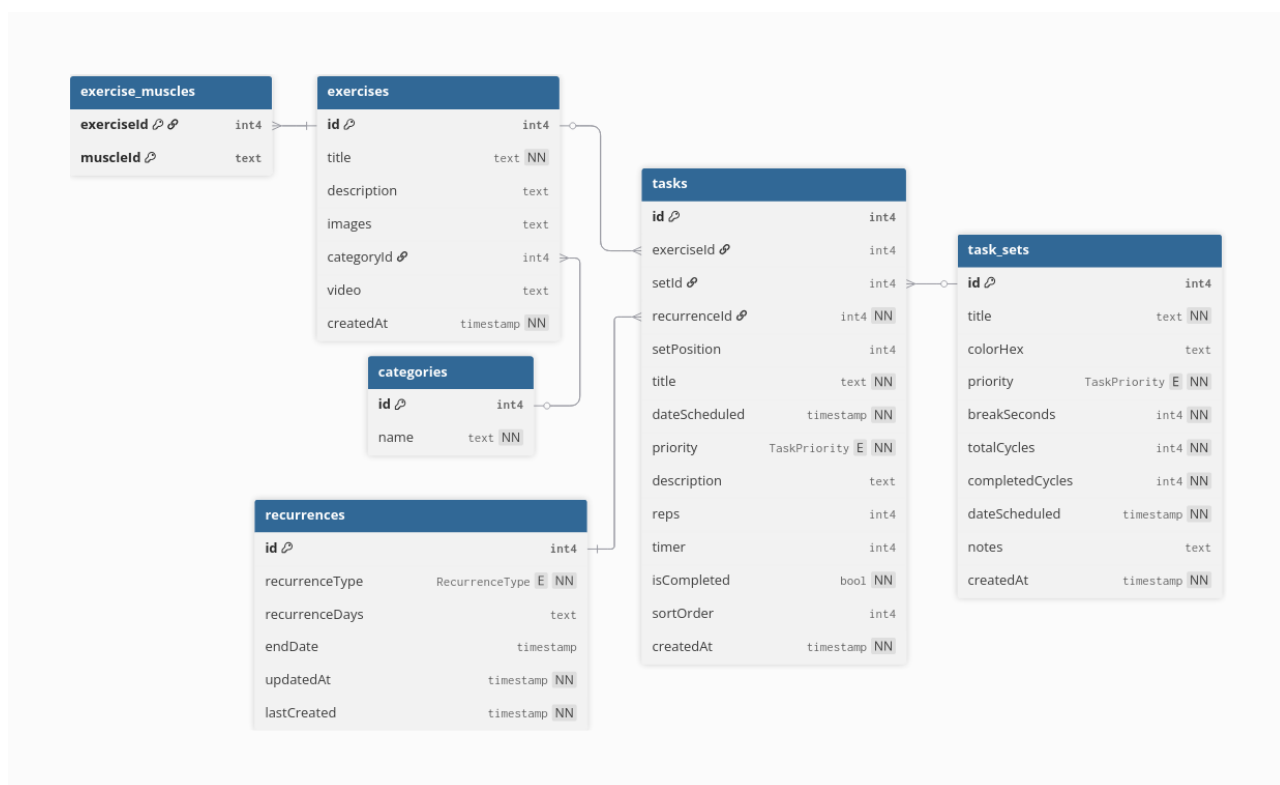


Рисунок 3.9 – ER-діаграма бази даних мобільного застосунку

На діаграмі представлено основні таблиці бази даних розробленого застосунку та зв'язки між ними. Центральною таблицею системи є `tasks`, яка зберігає загальну інформацію про завдання користувача, а також містить додаткові фітнес-орієнтовані поля, зокрема кількість повторів і таймер (ліст. 3.11). Таблиця `exercises` використовується для шаблонів вправ, `categories`

забезпечує їх групування за категоріями, `recurrences` містить параметри повторюваних завдань, а `task_sets` використовується для збереження тренувальних сетів. Допоміжна таблиця `exercise_muscles` використовується для встановлення зв'язку між вправами та групами м'язів. Таблиця `tasks` пов'язана з таблицями `exercises`, `recurrences` та `task_sets`, що дозволяє пов'язувати завдання із вправами, правилами повторення та тренувальними сетами. Крім того, зв'язок між `exercises` і `categories` забезпечує класифікацію вправ, а використання таблиці `exercise_muscles` дозволяє асоціювати одну вправу з кількома м'язовими групами.

Лістинг 3.11 – Реалізація таблиці завдань Tasks

```
@DataClassName("Task")
class Tasks extends Table {
    IntColumn get id => integer().autoIncrement()();

    IntColumn get exerciseId =>
        integer().nullable().references(Exercises, #id)();
    IntColumn get recurrenceId =>
        integer().references(Recurrences, #id)();
    IntColumn get setId =>
        integer().nullable().references(TaskSets, #id)();

    IntColumn get setPosition => integer().nullable()();

    TextColumn get title => text().withLength(min: 1, max: 100)();

    DateTimeColumn get dateScheduled => dateTime()();

    TextColumn get priority => textEnum<TaskPriority>()();

    TextColumn get description => text().nullable()();

    IntColumn get reps => integer().nullable()();

    IntColumn get timer => integer().nullable()();

    BoolColumn get isCompleted =>
        boolean().withDefault(const Constant(false))();

    IntColumn get sortOrder => integer().nullable()();

    DateTimeColumn get createdAt =>
        dateTime().clientDefault(() => DateTime.now())();
}
```

кінець лістингу 3.11

У наведеному фрагменті коду описано структуру таблиці Tasks, у якій поле id використовується як первинний ключ і створюється автоматично, а поля exerciseId, recurrenceId та setId забезпечують зв'язок завдання з іншими таблицями бази даних. Поле title зберігає назву завдання, dateScheduled визначає дату виконання, priority відповідає за рівень важливості, а isCompleted використовується для визначення стану виконання, тоді як поля reps і timer призначені для зберігання фітнес-орієнтованих параметрів завдання.

Крім опису структури таблиць, у базі даних реалізовано методи для виконання запитів до локального сховища. Одним із найважливіших запитів є отримання списку завдань за вибрану дату, оскільки саме ця інформація використовується для формування календаря та відображення переліку завдань на головному екрані. Реалізація даного запиту представлена в лістингу 3.12.

Лістинг 3.12 – Отримання списку завдань за вибраною датою

```
Future<List<Task>> getTasksForDate(DateTime date) async {
    DateTime normalizedDate =
        normalizeToMidnight(date);

    return (select(tasks)
        ..where((t) {
            final startOfDay =
                normalizeToMidnight(normalizedDate);

            final endOfDay = startOfDay
                .add(Duration(days: 1))
                .subtract(Duration(milliseconds: 1));

            return t.dateScheduled.isBetweenValues(startOfDay, endOfDay);
        }))
        .get();
}
```

кінець лістингу 3.12

На початку роботи методу дата нормалізується до початку доби, після чого визначаються межі обраного дня. Далі виконується вибірка записів із таблиці tasks, дата виконання яких потрапляє до заданого часового проміжку.

Під час розробки застосунку структура бази даних змінювалася відповідно до появи нових функціональних можливостей. Для коректного оновлення структури бази даних реалізовано механізм міграцій, який дозволяє оновлювати схему бази даних без втрати наявних даних користувача (ліст. 3.13).

Лістинг 3.13 – Реалізація міграції структури бази даних

```
@override
MigrationStrategy get migration =>
  MigrationStrategy(
    onUpgrade: (migrator, from, to) async {
      if (from < 2) {
        await migrator.addColumn(
          exercises,
          exercises.video
            as GeneratedColumn<Object>,
        );
      }

      if (from < 4) {
        await migrator.createTable(taskSets);
        await migrator.addColumn(tasks, tasks.setId);
        await migrator.addColumn(tasks, tasks.setPosition);
      }

      if (from < 5) {
        await migrator.createTable(exerciseMuscles);
      }
    },
  );
```

кінець лістингу 3.13

Наведена міграція описує оновлення структури бази даних залежно від її попередньої версії. Якщо версія схеми є нижчою за визначене значення, до бази додаються нові поля або створюються нові таблиці. Зокрема, під час розширення функціональності додано підтримку відео для шаблонів вправ, таблицю тренувальних сетів, поля для зв'язку завдань із сетами та таблицю для зв'язку вправ із групами м'язів. Такий підхід спрощує розвиток застосунку та забезпечує можливість розширення структури бази даних у майбутньому.

3.4 Розробка користувацького інтерфейсу мобільного застосунку

Важливою складовою будь-якого мобільного програмного забезпечення є користувацький інтерфейс, оскільки саме через нього здійснюється взаємодія користувача із функціональними можливостями додатку. Під час розробки мобільного застосунку особлива увага приділялася забезпеченню простоти навігації, логічній організації функціональних можливостей та створенню єдиного візуального стилю для всіх екранів системи.

Користувацький інтерфейс застосунку реалізовано засобами фреймворку Flutter із використанням компонентів Material Design, який забезпечує сучасний зовнішній вигляд застосунку, підтримку стандартних елементів керування та узгодженість інтерфейсу на різних мобільних пристроях [17]. Під час проєктування інтерфейсу використовувалися принципи швидкого доступу до найуживаніших функцій та їх логічного розподілу між окремими екранами.

Для забезпечення навігації між основними функціональними розділами застосунку використовується навігаційна панель, яка дозволяє користувачу швидко перемикатися між календарем, головним екраном та бібліотекою вправ без необхідності переходу через додаткові меню (ліст. 3.14).

Лістинг 3.14 – Реалізація навігаційної панелі

```
final List<Widget> _screens = [
  const CalendarScreen(),
  const DashboardScreen(),
  const LibraryScreen(),
];

BottomNavigationBar(
  currentIndex: _selectedIndex,
  onTap: _onItemTapped,
  items: [
    BottomNavigationBarItem(
      icon: Icon(Icons.calendar_month_rounded),
      label: "calendar".tr(),
    ),
    BottomNavigationBarItem(
      icon: Icon(Icons.list),
      label: "dashboard".tr(),
    ),
  ],
)
```

```

    ),
    BottomNavigationBarItem(
      icon: Icon(Icons.folder),
      label: "library".tr(),
    ),
  ],
)

```

кінець лістингу 3.14

Наведений код демонструє перелік екранів застосунку та елементи нижньої панелі навігації. Після вибору відповідної вкладки система змінює активний екран, забезпечуючи швидкий доступ до необхідного функціоналу, що спрощує навігацію між основними розділами застосунку та робить його можливості доступними з будь-якого місця в програмі.

Основним робочим екраном системи є екран керування завданнями, на якому відображаються завдання, тренувальні сеті, заплановані на вибрану дату, засоби сортування та елементи керування записами. Користувач може переглядати список завдань, позначати їх як виконані, редагувати, видаляти та створювати нові записи за допомогою кнопки швидкого додавання. Для тренувальних сетів додатково відображається прогрес виконання циклів і стан поточного тренувального процесу (рис. 3.10).

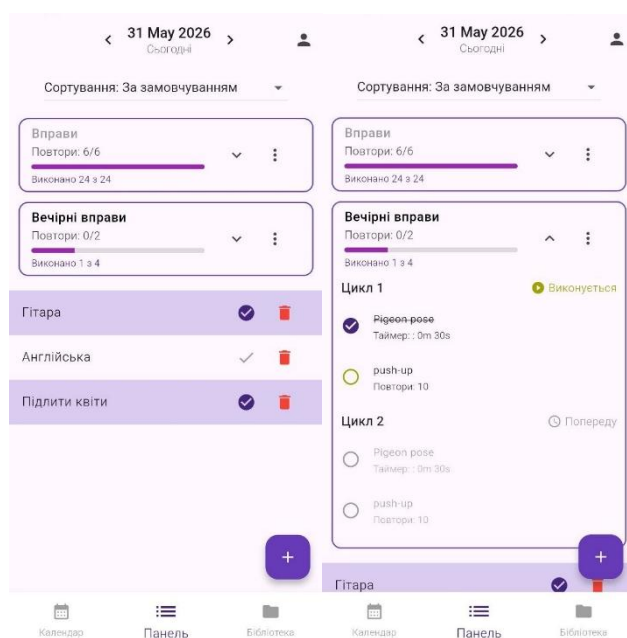


Рисунок 3.10 – Головний екран мобільного застосунку

Для забезпечення зручного планування активностей у застосунку реалізовано окремий екран календаря представлений на рисунку 3.11.

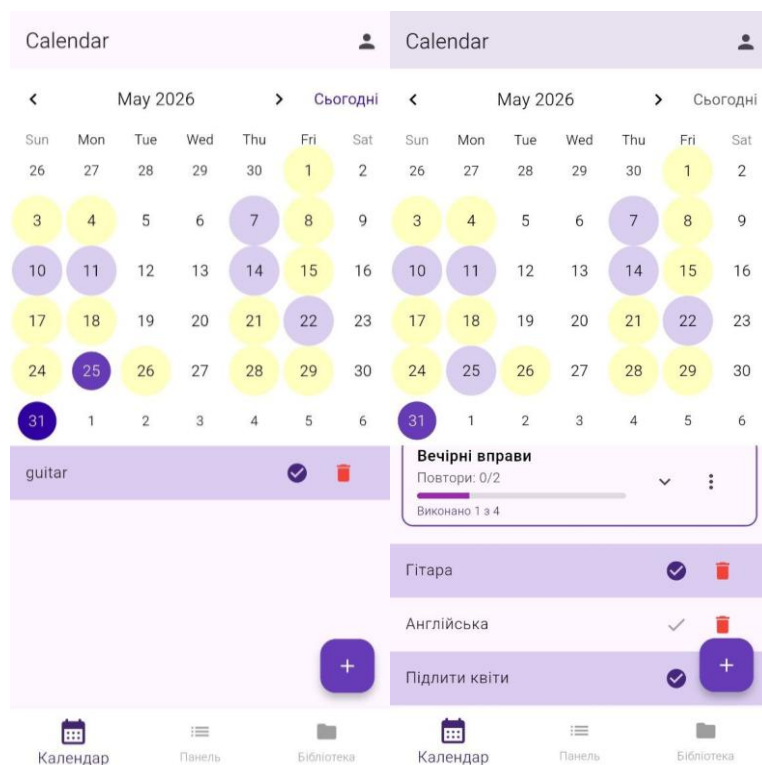


Рисунок 3.11 – Екран календаря мобільного застосунку

Календар дозволяє переглядати дати, перемикається між місяцями та вибирати конкретний день для перегляду запланованих завдань. Після вибору дня система автоматично відображає відповідні записи, що підвищує зручність перегляду та редагування запланованих активностей. Для візуального відображення стану виконання завдань використовується кольорове маркування дат: зеленим кольором позначаються дні, у яких усі заплановані завдання виконано, а жовтим дні з невиконаними завданнями. Кольори маркування також можуть налаштовуватися користувачем відповідно до власних уподобань. Календарне представлення спрощує роботу з великими обсягами завдань та дозволяє користувачу швидко орієнтуватися у власному графіку активностей.

Однією з особливостей розроблюваного застосунку є наявність бібліотеки шаблонів вправ, для роботи з якою реалізовано окремий екран (рис. 3.12).

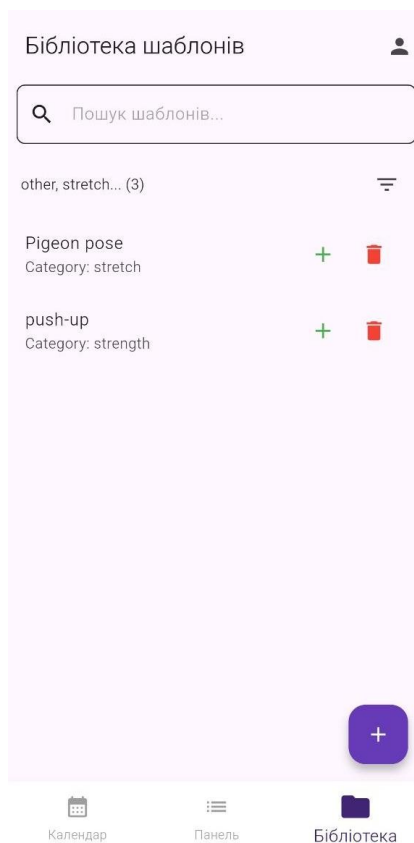


Рисунок 3.12 – Екран бібліотеки шаблонів мобільного застосунку

Бібліотека вправ дозволяє створювати, переглядати та редагувати шаблони, які можуть використовуватися під час створення нових завдань. Їх організація у вигляді окремої бібліотеки дає змогу формувати власний набір шаблонів та багаторазово використовувати їх під час планування фізичної активності, або повсякденних справ. Такий підхід усуває необхідність повторного введення однакових даних і спрощує формування тренувальних планів, особливо для регулярних тренувань, які часто складаються з однакових або подібних вправ.

Для аналізу результатів роботи користувача у застосунку реалізовано окремий екран статистики (рис. 3.13), на якому відображаються показники виконання завдань, інформація про активність користувача та узагальнені результати використання застосунку. Використання графічних елементів і статистичних даних спрощує сприйняття інформації та забезпечує швидкий аналіз досягнутих результатів. Завдяки цьому користувач може контролювати власну активність, оцінювати свій прогрес та відстежувати динаміку виконання поставлених завдань.

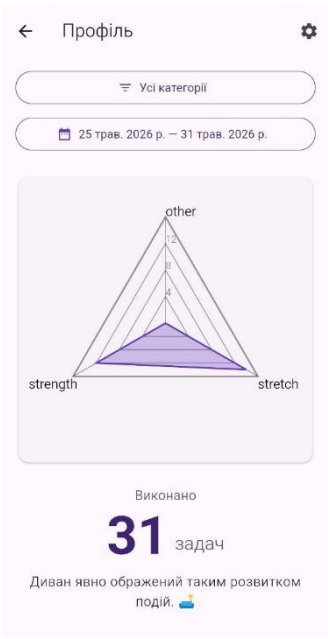


Рисунок 3.13 – Екран статистики користувача

Для налаштування параметрів застосунку реалізовано екран налаштувань, представлений на рисунку 3.14. Він містить засоби зміни теми оформлення, вибору мови інтерфейсу, керування нагадуваннями, створення резервних копій та відновлення даних. Винесення таких можливостей в окремий розділ дозволяє не перевантажувати основні екрани застосунку налаштуваннями та допоміжними функціями, забезпечуючи більш зручну організацію інтерфейсу.

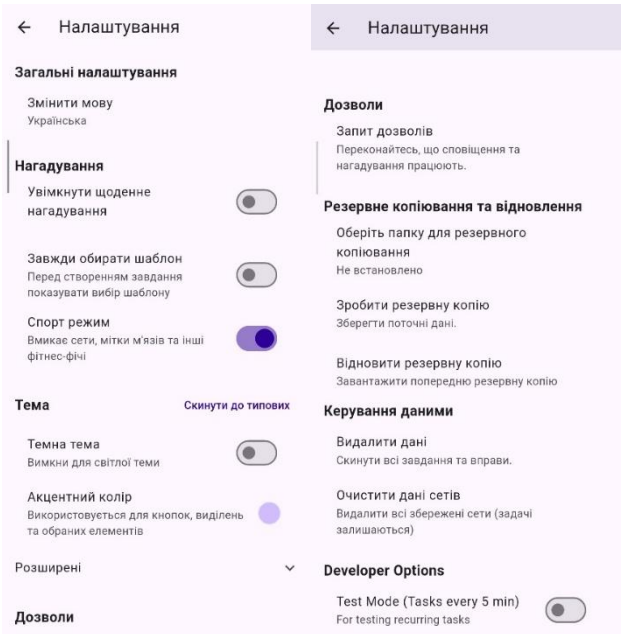


Рисунок 3.14 – Екран налаштувань мобільного застосунку

3.5 Розробка документації для програмного продукту

Завершальним етапом розробки програмного забезпечення є підготовка документації, яка дозволяє користувачам ознайомитися з можливостями програмного продукту, вимогами до його використання та особливостями роботи окремих функцій. Наявність документації спрощує процес встановлення застосунку та дозволяє швидше освоїти його функціональні можливості. Для мобільного застосунку документація містить системні вимоги, рекомендації щодо встановлення та опис основних функціональних можливостей системи.

Для коректної роботи застосунку необхідний мобільний пристрій із підтримкою сучасних версій операційної системи Android. Мінімальні та рекомендовані системні вимоги наведено в таблиці 3.1.

Таблиця 3.1 – Системні вимоги до мобільного застосунку

Параметр	Мінімальні системні вимоги	Рекомендовані системні вимоги
Операційна система	Android 9.0 або новіша	Android 11 або новіша
Оперативна пам'ять	2 ГБ	4 ГБ і більше
Вільне місце у пам'яті пристрою	150 МБ	300 МБ і більше
Дозволи застосунку	Сповіщення	Сповіщення та будильники
Підключення до мережі Інтернет	Не обов'язкове	Необов'язкове, використовується для перегляду зовнішніх відеоматеріалів

Встановлення застосунку здійснюється за допомогою інсталяційного файлу формату APK [18]. Для початку роботи користувачу необхідно завантажити APK файл на мобільний пристрій та виконати його встановлення. Після першого запуску система може запросити дозвіл на відображення сповіщень, що необхідно для роботи механізму нагадувань. На сучасних версіях Android також може знадобитися надання дозволу на використання будильників, які застосовуються для коректного спрацювання нагадувань та таймерів.

На відміну від багатьох сучасних мобільних рішень, застосунк не використовує серверну частину, а усі дані зберігає локально у базі даних SQLite, яка розташовується у внутрішньому сховищі пристрою. Завдяки цьому

користувачу не потрібно виконувати додаткові налаштування підключення до серверів або створювати облікові записи для початку роботи із застосунком.

Для спрощення використання системи користувачами надається набір довідкових відомостей щодо основних функціональних можливостей застосунку. Створення нового завдання здійснюється через відповідну форму введення, де можна вказати назву, дату виконання, рівень пріоритету, додаткові нотатки, кількість повторів або параметри таймера. Для регулярних активностей передбачено налаштування повторення завдань за визначеними правилами. Під час створення завдань також можна використовувати шаблони вправ із бібліотеки, що дозволяє прискорити процес планування тренувань.

Контроль виконання активностей здійснюється через календарний інтерфейс або головний екран застосунку. Після завершення активності завдання може бути позначене як виконане, а інформація автоматично врахується під час формування статистики. Для тренувальних сценаріїв підтримується створення сетів, які дозволяють об'єднати декілька завдань у межах одного тренування.

Для захисту даних у застосунку реалізовано механізми резервного копіювання та відновлення інформації, доступні в розділі налаштувань. Інструменти експорту та імпорту даних дозволяють зберігати поточний стан локальної бази даних у зовнішньому файлі та за потреби відновлювати інформацію після перевстановлення застосунку або перенесення даних на інший пристрій. Використання такого підходу забезпечує захист даних від випадкової втрати та дозволяє зберегти всі створені завдання, шаблони вправ, сети, параметри повторення й іншу інформацію користувача.

Таким чином, розроблена документація охоплює основні аспекти використання мобільного застосунку, включаючи системні вимоги, порядок встановлення, особливості роботи основних функцій та механізми резервного копіювання даних. Наведена інформація дозволяє користувачу швидко ознайомитися з можливостями програмного продукту та ефективно використовувати його без потреби у додатковому навчанні.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи бакалавра розроблено мобільний застосунок для управління фітнес-орієнтованими завданнями з використанням Flutter та Kotlin.

У процесі виконання роботи проведено аналіз предметної області, сучасних мобільних технологій та існуючих програмних рішень, а саме Todoist, Strong та Habitica. У результаті виявлено, що вони орієнтовані або виключно на планування завдань, або виключно на фітнес-функції. Жоден із них не поєднує обидва напрями в межах єдиної системи. Виявлена прогалина обґрунтувала доцільність розробки нового програмного продукту, що інтегрує можливості менеджера завдань з інструментами підтримки фізичної активності.

Спроектовано трирівневу архітектуру застосунку, що включає Flutter Application (UI Layer, Business Logic Layer, Data Access Layer), Local Storage (Drift ORM, SQLite, SharedPreferences) та Android Native Layer (Kotlin-модуль, AlarmManager, WorkManager, NotificationService, BootReceiver). Для опису системи побудовано діаграму прецедентів, діаграму діяльності, діаграму класів із п'ятьма основними сутностями, діаграму компонентів та блок-схема алгоритму створення циклічних завдань.

Реалізовано користувацький інтерфейс на базі компонентів Material Design з використанням Flutter та Dart. Навігація між розділами забезпечується нижньою панеллю з трьома вкладками: календар, головний екран та бібліотека вправ. Додатково реалізовано екрани статистики та налаштувань. Передбачено підтримку двох мов інтерфейсу через бібліотеку EasyLocalization, а саме українська та англійська. Також реалізовано ручний вибір між світлою та темною темою оформлення акцентних кольорів.

Також розроблено механізми керування та планування завданнями. Реалізовано підтримку календарного представлення інформації та механізм повторюваних подій, які дозволяють ефективно організовувати фізичну активність користувача.

У ході роботи реалізовано функції роботи із шаблонами вправ, тренувальними сетами, таймерами та нагадуваннями. Для забезпечення доступу до окремих можливостей операційної системи Android використано мову програмування Kotlin, що дозволило реалізувати точні нагадування, фонові процеси та інші нативні функції. Також розроблено механізми формування статистики, що дозволяє користувачам аналізувати результати власної активності.

Крім того, розроблено модель бази даних та систему локального зберігання інформації із використанням SQLite та бібліотеки Drift. Використання локального сховища забезпечило автономну роботу застосунку без необхідності підключення до віддалених серверів.

Тестування проводилося на емуляторах Android Studio із різними версіями операційної системи, а також на реальному мобільному пристрої. Перевірено усі ключові функціональні модулі: створення та редагування завдань, роботу повторюваних подій, тренувальні сету, таймери, нагадування, резервне копіювання, локалізацію та зміну теми. За результатами тестування виявлено та усунуено два суттєві дефекти: помилку масового оновлення статусу повторюваних завдань та некоректне центрування елементів верхньої панелі інтерфейсу.

Практичне значення роботи полягає у створенні готового до використання мобільного застосунку для Android, що дозволяє планувати як повсякденні завдання, так і спортивну активність в єдиній системі без необхідності підключення до мережі.

Подальший розвиток застосунку може включати: розширену аналітику прогресу з побудовою графіків динаміки активності, інтеграцію з фітнес-трекерами через Google Fit або Health Connect API, реалізацію хмарної синхронізації даних між пристроями.

Результати роботи будуть використані під час створення нових програмних рішень у сфері фітнесу, персонального планування та підтримки здорового способу життя, а також стануть основою для подальших досліджень у цій галузі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sousa Basto P., Ferreira P. Mobile applications, physical activity, and health promotion. BMC Health Services Research. 2025. Vol. 25, no. 1. URL: <https://doi.org/10.1186/s12913-025-12489-z> (дата звернення: 09.02.2026).
2. Zarichuk O. Comparative analysis of frameworks for mobile application development: Native, hybrid, or cross-platform solutions. Вісник Черкаського державного технологічного університету. 2023. Vol. 28, no. 4. P. 19-27. URL: <https://doi.org/10.62660/2306-4412.4.2023.19-27> (дата звернення: 13.02.2026).
3. Väänänen T. Flutter in cross platform development: Tools, performance and optimization. University of Oulu. Information Processing Science. Bachelor's Thesis. 2025. 25 p. URL: <https://oulurepo.oulu.fi/bitstream/handle/10024/54358/nbnfioulu-202502261849.pdf> (дата звернення: 15.02.2026).
4. Todoist. A To-Do List to Organize Your Work & Life. URL: <https://www.todoist.com/> (дата звернення: 19.02.2026).
5. Strong – Workout Tracker & Gym Log. URL: <https://www.strong.app/> (дата звернення: 22.02.2026).
6. Habitica – Gamify Your Life. URL: <https://habitica.com/static/home> (дата звернення: 25.02.2026).
7. UML – Unified Modeling Language. Object Management Group. URL: <https://www.omg.org/uml/> (дата звернення: 05.03.2026).
8. Flutter documentation. URL: <https://docs.flutter.dev/> (дата звернення: 07.03.2026).
9. Kotlin and Android. Android Developers. URL: <https://developer.android.com/kotlin> (дата звернення: 07.03.2026).
10. SQLite Documentation. URL: <https://www.sqlite.org/docs.html> (дата звернення: 10.03.2026).
11. Drift. URL: <https://drift.simonbinder.eu/> (дата звернення: 15.03.2026).
12. Shared_preferences. Flutter package. Dart packages. URL: https://pub.dev/packages/shared_preferences (дата звернення: 23.03.2026).

13. Android Studio & App Tools – Android Developers. URL: <https://developer.android.com/studio> (дата звернення: 01.04.2026).

14. MaterialApp class – material library – Dart API. Flutter – Dart API docs. URL: <https://api.flutter.dev/flutter/material/MaterialApp-class.html> (дата звернення: 02.04.2026).

15. Writing custom platform-specific code. Flutter documentation. URL: <https://docs.flutter.dev/platform-integration/platform-channels> (дата звернення: 02.04.2026).

16. NotificationCompat.Builder. API reference. Android Developers. URL: <https://developer.android.com/reference/androidx/core/app/NotificationCompat.Builder> (дата звернення: 07.04.2026).

17. Material Design. URL: <https://m3.material.io/> (дата звернення: 10.04.2026).

18. APK. Publish your app. Android Studio. Android Developers. URL: <https://developer.android.com/studio/publish> (дата звернення: 29.04.2026).