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 ДОДАТКУ ДЛЯ ФІТНЕС-ТРЕКІНГУ З ВИКОРИСТАННЯМ FLUTTER, NODE.JS, EXPRESS.JS ТА FIREBASE/POSTGRESQL

DEVELOPMENT OF A MOBILE FITNESS TRACKING APPLICATION USING FLUTTER, NODE.JS, EXPRESS.JS AND FIREBASE/POSTGRESQL

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

Виконав: здобувач вищої освіти
групи ІПЗз-41
Левкова Вікторія Ярославівна

Керівник:
Суринович Олена Миколаївна

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

Луцьк – 2025 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» 12 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Левковій Вікторії Ярославівні
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка мобільного додатку для фітнес-трекінгу з використанням Flutter, Node.js, Express.js, та Firebase/PostgreSQL»*
Керівник роботи: *к.т.н., доцент Суринович О. М.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *Flutter, Node.js, Express.js, Firebase/PostgreSQL, методичні вказівки до виконання кваліфікаційної роботи бакалавра*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *необхідно проаналізувати предметну область фітнес-трекінгу, вивчити аналогічні мобільні додатки та сформулювати вимоги до функціональності системи. Потрібно спроєктувати архітектуру додатку на основі Flutter для фронтенду та Node.js з Express.js для бекенду, використовуючи Firebase для push-сповіщення і PostgreSQL для зберігання даних. Також слід реалізувати основні модулі, підготувати інструкцію користувача та виконати оцінку ефективності.*

5. Перелік графічного матеріалу: *рисунків 7, таблиць 4, додатків 1.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Суринович О. М.*		
Специфікація вимог до розробленої системи	*Суринович О. М.*		
Розробка об'єкта проектування	*Суринович О. М.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	1,61 %		
Академічна доброчесність	*Суринович О. М.*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	вик.
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 01.03.2025 р.	вик.
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	вик.
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 29.03.2025 р.	вик.
5	Практична реалізація об'єкта проектування та розробка бази даних	до 26.04.2025 р.	вик.
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	вик.
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	вик.

Здобувач вищої освіти _____ Вікторія ЛЕВКОВА

Керівник кваліфікаційної роботи _____ Олена СУРИНОВИЧ

АНОТАЦІЯ

Левкова В. Я. Розробка мобільного додатку для фітнес-трекінгу з використанням Flutter, Node.js, Express.js та Firebase/PostgreSQL. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

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

У першому розділі здійснено аналіз галузі фітнес-додатків, розглянуто наявні підходи до відстеження фізичної активності, виявлено їхні переваги й недоліки, а також сформульовано науково-технічне завдання проєкту з урахуванням сучасних потреб користувачів.

У другому розділі наведено специфікацію вимог до функціональності системи, здійснено аналіз архітектурних рішень, обґрунтовано вибір інструментів для реалізації клієнт-серверного застосунку, а також спроєктовано загальну структуру програмного забезпечення.

У третьому розділі представлено реалізацію мобільного додатку на основі Flutter із використанням Node.js, Express.js та Firebase/PostgreSQL. Докладно описано логіку основних функціональних модулів, реалізацію інтерфейсу користувача та налаштування взаємодії з серверною частиною. Також наведено інструкцію користувача.

У висновках підбито підсумки виконаної роботи, оцінено ефективність розробленого рішення, окреслено переваги застосованих технологій, а також визначено можливі напрями подальшого вдосконалення системи.

Ключові слова: фітнес-додаток, Flutter, мобільна розробка, Firebase, Node.js, Express.js, PostgreSQL, трекінг активності, push-сповіщення, клієнт-серверна архітектура, інтерактивний інтерфейс, фізична активність, аналіз вимог, розробка ПЗ, інформаційна система.

ABSTRACT

Levkova V. Development of a Mobile Fitness Tracking Application Using Flutter, Node.js, Express.js and Firebase/PostgreSQL. Manuscript. Qualification work of the bachelor's program "Software engineering" in the specialty "Software engineering". Lutsk National Technical University. Lutsk, 2025.

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

The first chapter analyzes the field of fitness applications, considers existing approaches to tracking physical activity, identifies their advantages and disadvantages, and formulates the scientific and technical task of the project, taking into account the modern needs of users.

The second section provides a specification of the requirements for the system's functionality, analyzes architectural solutions, justifies the choice of tools for implementing a client-server application, and designs the overall structure of the software.

The third section presents the implementation of a mobile application based on Flutter using Node.js, Express.js, and Firebase/PostgreSQL. The logic of the main functional modules, the implementation of the user interface, and the configuration of interaction with the server side are described in detail. The user manual is also provided.

The conclusions summarize the results of the work performed, evaluate the effectiveness of the developed solution, outline the advantages of the applied technologies, and identify possible areas for further improvement of the system.

Keywords: fitness application, Flutter, mobile development, Firebase, Node.js, Express.js, PostgreSQL, activity tracking, push notifications, client-server architecture, interactive interface, physical activity, requirements analysis, software development, information system.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі зростає попит на цифрові рішення, які сприяють веденню здорового способу життя. Мобільні фітнес-додатки дозволяють користувачам відстежувати фізичну активність, контролювати свої досягнення та підтримувати мотивацію до спорту. Особливо актуальним є розроблення таких застосунків із використанням кросплатформених технологій, які забезпечують зручність і доступність для широкого кола користувачів. Використання Flutter у поєднанні з хмарними сервісами Firebase та надійною базою даних PostgreSQL забезпечує швидку розробку, гнучкість і масштабованість системи.

Таким чином, метою даної кваліфікаційної роботи є розробка мобільного додатку для фітнес-трекінгу, що дозволяє відстежувати фізичну активність користувача, зберігати дані на сервері та надсилати push-сповіщення.

Об’єктом роботи є процеси взаємодії користувача з мобільним застосунком для контролю фізичної активності.

Предметом роботи є методи й засоби розробки кросплатформених мобільних додатків з функціональністю фітнес-трекінгу.

Для досягнення поставленої мети були поставлені наступні завдання:

1) аналіз існуючих рішень у сфері фітнес-додатків;
2) формування функціональних і нефункціональних вимог до системи;
3) вибір інструментів та архітектури для реалізації;
4) проєктування структури клієнтської та серверної частин;
5) реалізація мобільного застосунку з використанням Flutter;
6) створення серверного API на Node.js/Express.js;
7) інтеграція з Firebase push-сповіщень.

РОЗДІЛ 1
АНАЛІЗ ГАЛУЗІ ВИКОРИСТАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

За останнє десятиліття мобільні фітнес-технології пройшли шлях від простих лічильників кроків до комплексних систем, що поєднують біомоніторинг, гейміфікацію і штучний інтелект [1]. Персоналізований підхід до здоров'я активно підтримується як споживачами, так і державами, що стимулює створення нових продуктів та підвищує вимоги до безпеки даних [2].

Світовий ринок прогнозує обсяг світових доходів від фітнес-додатків приблизно 23 млрд USD та демонструє CAGR ≈ 13,9 % на горизонті 2025-2030 рр. Носима електроніка (браслети, годинники, смарт-кільця) у 2024 р. досягла 534,6 млн пристроїв, з прогнозом помірного зростання 4,1 % у 2025 р. Таким чином, додатки стають важливим інтерфейсом між користувачем і багатофункціональними гаджетами, забезпечуючи агрегацію даних, мотивацію та аналітику [3].

CAGR (Compound Annual Growth Rate) – це середньорічний темп приросту з урахуванням складного відсотка. Він показує, яким був би один і той самий річний відсоток зростання, якби показник (виручка, ринкова вартість тощо) зростав рівномірно від початкового до кінцевого значення за весь період *n* років.

Пандемія COVID-19 радикально змінила тренувальні звички: домашні тренування та активності (біг, ходьба, вело) замінили закриті спортзали. Хоча локдауни зняті, сформована звичка самостійного трекінгу зберігається, стимулюючи попит на додатки з глибокою аналітикою та соціальними елементами (онлайн-челенджі і таке інше).

Для кращого розуміння поточного стану ринку фітнес-технологій проведено аналіз існуючих рішень, згрупованих за типом продукту. У

таблиці 1.1 представлено типові класи пристроїв і додатків, ключові можливості кожної категорії та основні обмеження їх використання [4]. Такий поділ дозволяє виявити переваги й недоліки кожного класу, що враховувалося при розробці власного рішення.

Таблиця 1.1 – Аналіз існуючих рішень

Клас продукту	Типові представники	Ключові можливості	Обмеження
1	2	3	4
Екосистемні трекери	Google Fit, Apple Health, Samsung Health	Централізоване сховище даних, синхронізація з годинниками	Закриті API, сувора екосистема
Спеціалізовані спорт-додатки	Strava, Nike Run Club, Adidas Running	GPS-маршрути, соціальний рейтинг, клуби за інтересами	Орієнтація лише на кардіо-вид, брак моніторингу сну й пульсу
AI-фітнес-коучі	Freeletics, FitGenie, Apple Workout Buddy (watchOS 26)	ML-рекомендації, голосові підказки, корекція техніки	Більшість рішень англомовні, проблеми з локалізацією та приватністю
Смарт-кільця	Oura Ring 4, Samsung Galaxy Ring	Безперервний пульс, SpO_2, температура	Висока ціна, обмежений дисплей

Технологічні тренди:

1) сенсори та носимі пристрої – акселерометри, гіроскопи та PPG-сенсори стали стандартом. Водночас смарт-кільця набирають популярності завдяки меншій вазі та цілодобовому комфорту носіння, із CAGR ≈ 17 % (2025-2028);

2) штучний інтелект і персоналізація – у 2025 р. ключовими драйверами є гіпер-персоналізовані AI-рекомендації, AR-тренування та «виртуальні коучі» [5]. Алгоритми обробляють трекінг у реальному часі й адаптують навантаження до стану користувача (HRV, фаза сну, рівень стресу);

3) кросплатформені фреймворки – Flutter утримує лідерство: 46 % розробників використовували його у 2023 р. [6], і тренд зростає завдяки hot-reload, єдиному код-базису та офіційній підтримці Google. Це знижує time-to-market і спрощує підтримку Android;

4) бекенд-стек – Node.js + Express: легкий неблокуючий сервер, зручний для REST/GraphQL-API і WebSocket-стримів реального часу. Firebase: push-нотифікації та хмарні функції для MVP-етапу. PostgreSQL: реляційне сховище для нормалізованої структури (таблиці activities, sleep, steps, pulse), підтримка геоданих (PostGIS) та аналітичних запитів.

Виявлені прогалини ринку мобільних фітнес-додатків в Україні:

1) брак локалізованих AI-коучів. Немає рішень із повною українською локалізацією, підтримкою метричної системи, кирилиці та національних харчових довідників МОЗ;

2) обмежена інтеграція сон-трекерів**.** Переважна більшість популярних додатків надає лише базову статистику сну без кореляції зі щоденною активністю або пульсовими даними;

3) закритість даних. Експорт результатів часто можливий лише у власних форматах або за платною підпискою, що ускладнює міграцію й аналітику;

4) низька довгострокова мотивація. Дослідження показують, що 30-40 % користувачів припиняють користуватись додатком після трьох місяців через одноманітний контент і брак соціальної взаємодії.

Ринок мобільних фітнес-рішень перебуває на етапі помірного, але стабільного росту. Носимі гаджети генерують дедалі більший обсяг даних, що потребує AI-аналітики та надійного бекенду. Водночас користувачі очікують повної прозорості обробки особистої інформації та локалізованого контенту.

Обрана архітектура Flutter Node.js [7], Express, PostgreSQL відповідає сучасним трендам, мінімізує time-to-market і забезпечує масштабованість. Запропонований додаток прагне об’єднати розрізнені підходи (кроки, пульс, сон, тренування) в єдину платформу, дотримуючись вимог GDPR і враховуючи специфіку українського користувача. Це створює потенційну конкурентну перевагу та наукову цінність розробки.

У результаті проведеного аналізу встановлено, що ринок мобільних фітнес-технологій демонструє стійке зростання, зростаючий попит на персоналізовані та інтегровані рішення, а також активне використання

штучного інтелекту та кросплатформенних технологій. Популяризація носимих пристроїв, таких як смарт-кільця, створює потребу у додатках, здатних працювати як інтерфейс між користувачем і багатофункціональними трекерами.

Сучасні платформи, такі як Flutter і Node.js, забезпечують ефективну реалізацію MVP-рішень із подальшою можливістю масштабування та підтримки аналітичних функцій. Водночас виявлено суттєві прогалини у локалізації, доступності даних, інтеграції сон-трекінгу та довгостроковій залученості користувача. Це відкриває нішу для створення українськомовного мобільного додатку для спортивного клубу, що забезпечить глибоку аналітику, адаптацію до фізіологічних даних користувача та інтеграцію з популярними носимими пристроями, враховуючи вимоги безпеки та прозорості даних.

Таким чином, обрана архітектура проєкту є доцільною та відповідає поточним тенденціям ринку й технічним вимогам користувачів, що створює передумови для ефективної розробки і впровадження інноваційного продукту.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Для досягнення поставленої мети розробки мобільного додатку для фітнес-трекінгу необхідно виконати такі основні завдання:

– провести аналіз сучасних фітнес-додатків, виявити їхні функціональні можливості, сильні та слабкі сторони, а також визначити пріоритетні потреби користувачів на українському ринку;

– сформувати функціональні та нефункціональні вимоги до системи, зокрема вимоги до швидкості реакції інтерфейсу, підтримки offline-режиму, сумісності з пристроями та безпеки зберігання даних;

– обґрунтувати вибір технологій та архітектури програмного забезпечення, визначити структуру клієнтської та серверної частини, включно з протоколами обміну, форматами даних і базовими схемами взаємодії;

– розробити дизайн та реалізувати мобільний застосунок з використанням Flutter, що включає екрани перегляду біометричних даних, налаштування профілю користувача, цілей та тренувальної активності;

– реалізувати серверну частину за допомогою Node.js та Express.js з побудовою REST API, обробкою запитів, перевіркою автентичності користувачів і збереженням даних у базі PostgreSQL;

– інтегрувати систему з Firebase для реалізації push-сповіщень, що нагадують про активність, досягнення цілей або інші важливі події, підвищуючи мотивацію користувача;

– протестувати стабільність і функціональність застосунку, перевірити коректність збору, передачі та візуалізації біометричних даних, а також відповідність вимогам продуктивності та надійності системи.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

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

У цьому підрозділі здійснюється систематизація функціональних та нефункціональних вимог до інформаційної системи фітнес-трекінгу, яка розробляється як мобільний додаток із серверною частиною.

Програмне забезпечення повинно реалізовувати функціонал збору, обробки, зберігання та візуалізації фізіологічних даних користувача (кількість кроків, пульс, сон, тренування), а також генерувати персоналізовані рекомендації. Система має складатися з мобільного клієнта та серверної частини з базою даних.

Функціональні вимоги:

1) збір даних з фітнес-браслета через Bluetooth (BLE) [8];
2) відображення пульсу користувача;
3) підрахунок і зберігання кількості кроків, відстані, витрачених калорій;
4) реєстрація тренувань (тип, тривалість, дистанція, швидкість);
5) аналіз сну: збереження часу засинання, пробудження, фаз сну;
6) надсилання push-нотифікацій (нагадування, рекомендації);
7) формування звітів на день, тиждень і місяць;
8) синхронізація локального кешу з сервером.

Нефункціональні вимоги:

1) кросплатформеність (Android);
2) безперебійна робота в офлайн-режимі не менше 24 год;
3) висока продуктивність (відповідь сервера ≤ 200 мс);
4) збереження даних користувача протягом ≥ 5 років;
5) відповідність вимогам GDPR щодо обробки персональних даних;
6) шифрування даних у БД;

7) зручний інтерфейс (UI/UX) з темною темою та адаптивністю до різних екранів.

Архітектурні рішення:

1) клієнт: реалізований на Flutter [9], забезпечує взаємодію з BLE-девайсами, відображення даних і UI;

2) серверна частина: побудована на Node.js з Express.js;

3) надає REST/WebSocket API, обробляє бізнес-логіку;

4) база даних: PostgreSQL – реляційна БД для зберігання кроків, пульсу, сну, тренувань.

Хмарні сервіси: Firebase Cloud Messaging – push-сповіщення.

У результаті аналізу були визначені функціональні й нефункціональні вимоги до програмного забезпечення, що охоплюють всі ключові аспекти роботи системи: від збору даних з браслета до формування рекомендацій та забезпечення інформаційної безпеки. На основі цих вимог сформовано технічне завдання, яке слугуватиме основою для подальшого проєктування, реалізації та тестування системи.

Комунікація між компонентами фітнес-додатку реалізована за клієнт-серверною архітектурою. Дані з BLE-пристрою (наприклад, фітнес-браслета) передаються до мобільного додатку через Bluetooth LE. Flutter-клієнт приймає ці дані, кешує їх локально, а потім надсилає на сервер через REST API, розроблений на Node.js.

Серверна частина обробляє запити та зберігає дані у базу PostgreSQL. При виконанні певних умов Firebase Cloud Messaging генерує push-сповіщення, які доставляються на пристрій користувача й відображаються у застосунку. Архітектурну схему описаного процесу зображено на рисунку 2.1.

Зазначене програмне забезпечення охоплює повний цикл: збір, обробку, зберігання та виведення фізіологічних даних користувача. Обрана архітектура підтримує масштабованість, офлайн-режим і забезпечує можливість гнучкого розширення функціональності в подальшому. Сформовані вимоги лягли в основу реалізації системи та визначають її подальший розвиток.

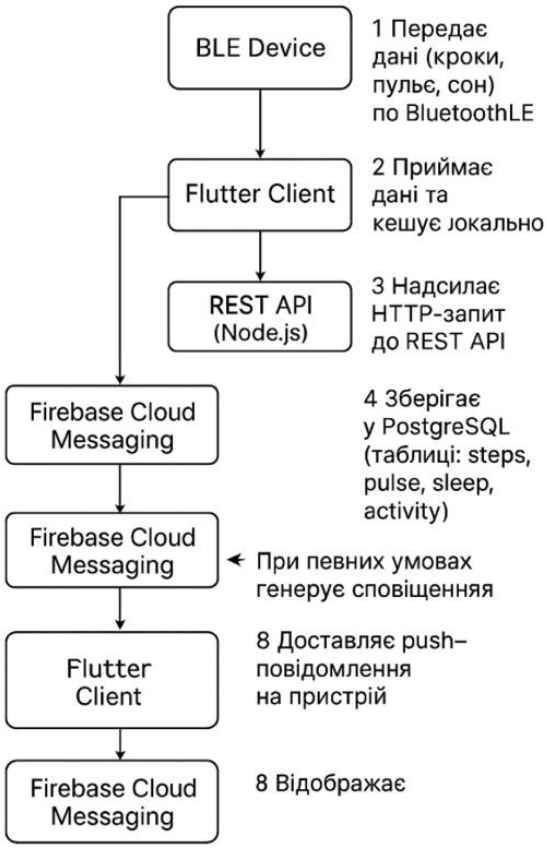

Рисунок 2.1 – Архітектурна схема

BLE Device (Bluetooth Low Energy-пристрій, фітнес-браслет):

1) вимірює фізіологічні дані користувача: кроки, пульс, глибину сну, відстань;

2) передає ці дані через GATT-повідомлення по Bluetooth LE;

3) не має доступу до Інтернету, а лише передає дані на телефон через BLE;

4) не автентифікує користувача – працює в парі з пристроєм;

5) GATT-повідомлення – це Bluetooth-пакет, у якому браслет передає нові дані (наприклад, пульс або кількість кроків) на телефон.

Bluetooth LE (Low Energy**)** – це різновид Bluetooth, розроблений спеціально для пристроїв, які передають невеликі обсяги даних і мають працювати на батареї дуже довго (місяцями або роками).

У таблиці 2.1 наведено основні відмінності між технологіями Bluetooth Classic і Bluetooth Low Energy (BLE). BLE є оптимальним варіантом для фітнес-

браслетів, оскільки передає невеликі обсяги даних (наприклад, пульс, кількість кроків), споживає мінімум енергії та не потребує постійного з’єднання. Натомість Bluetooth Classic використовується для передачі великих потоків, як-от музика, але витрачає значно більше заряду.

Таблиця 2.1 – Різниця між Bluetooth і Bluetooth LE

Bluetooth Classic	Bluetooth Low Energy (BLE)
1	2
Передає музику, файли	Передає датчики, стани, короткі дані
Потребує більше енергії	Економить енергію
Постійне з’єднання	З’єднання коротке і за потреби
Приклад: навушники	Приклад: фітнес-браслет, розумний термометр

Flutter Client (Мобільний додаток, встановлений на Android):

1) підключається до BLE-пристрою та читає дані в режимі реального часу;

2) зберігає їх локально (в offline-кеш ≥ 24 год) і при наявності Інтернету передає на сервер;

3) відображає інформацію у вигляді дашбордів, графіків і звітів;

4) також приймає push-нотифікації з Firebase Cloud Messaging;

5) REST API (Серверний бекенд, реалізований на Node.js + Express.js);

6) приймає HTTP-запити з клієнта (Flutter): збереження кроків, пульсу, сну, тренувань тощо;

7) валідує дані, обробляє запити;

8) відповідає за логіку рекомендацій, агрегації, обробку повідомлень;

9) працює з БД і чергами.

Firebase Cloud Messaging (Хмарна система для push-повідомлень):

1) дозволяє надсилати push-нагадування користувачеві. Наприклад для нашого випадку, «Ви вже день не займались тренуванням!»;

2) сервер Express.js викликає API FCM через firebase-admin.messaging() із переданим FCM-токеном користувача;

3) Flutter-клієнт автоматично отримує ці повідомлення та показує їх у системному UI.

Database (PostgreSQL) (Сховище всіх даних фітнес-додатку):

1) зберігає всі типи вимірювань: кроки, пульс, сон, тренування. У вигляді нормалізованих таблиць;

2) зберігає GPS-треки (у форматі PostGIS), raw JSON-пакети, позначки «offline», статус синхронізації;

3) також може містити FCM-токени, список пристроїв користувача, мету активності, індикатори прогресу.

На основі проведеного аналізу було сформульовано повний перелік функціональних та нефункціональних вимог до мобільного фітнес-додатку, що забезпечує збір, обробку, зберігання та візуалізацію фізіологічних даних користувача. Визначені вимоги враховують як технічні аспекти (офлайн-режим, продуктивність, безпека), так і зручність користувача (адаптивний інтерфейс, push-нагадування, аналітика).

Функціональні вимоги охоплюють ключові сценарії взаємодії користувача з додатком: підключення до фітнес-браслета, зчитування і збереження даних про пульс, кількість кроків, тренування, сон, формування аналітичних звітів та отримання push-нотифікацій. Особлива увага приділена реалізації режиму офлайн-роботи з подальшою синхронізацією, що критично для забезпечення безперервності збору даних.

Нефункціональні вимоги враховують технічні аспекти продуктивності, безпеки, масштабованості та зручності інтерфейсу. Система повинна функціонувати швидко (відповідь API до 200 мс), зберігати історичні дані користувача протягом п’яти років, відповідати вимогам GDPR щодо обробки персональних даних, а також адаптуватися під різні розміри екранів і підтримувати темну тему.

Запропонована архітектура ґрунтується на клієнт-серверному підході, де Flutter-додаток виконує роль інтерфейсу користувача та збирача даних, серверна частина на Node.js – відповідає за обробку запитів і бізнес-логіку, а PostgreSQL – за надійне зберігання інформації. Система підтримує push-

сповіщення за допомогою Firebase, що підвищує ефективність комунікації з користувачем. Комунікація між компонентами реалізована через REST API.

Особливе місце займає використання Bluetooth Low Energy як каналу обміну з фітнес-пристроєм. Завдяки низькому енергоспоживанню та оптимізації обміну короткими повідомленнями (GATT), BLE є ідеальним протоколом для щоденного моніторингу життєвих показників. Інтеграція BLE з Flutter дає змогу гнучко реалізувати сценарії зчитування фізіологічних даних у реальному часі, що відкриває перспективи для впровадження AI-аналітики, персоналізованих тренувальних рекомендацій і автоматичного виявлення критичних станів.

Таким чином, розроблена концепція та архітектура фітнес-додатку повністю відповідає поточним вимогам ринку й користувачів. Вона дозволяє ефективно масштабувати систему, адаптувати функціональність під нові потреби (наприклад, моніторинг стресу, гідратації, інтеграція з хмарними медичними системами) та слугує надійною основою для впровадження сучасного цифрового помічника здоров’я. Визначені вимоги та обрана архітектура ляжуть в основу наступних етапів: проєктування, реалізації, тестування та впровадження системи у реальних умовах.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

У сучасних інформаційних системах прийнято розділяти програмне забезпечення на клієнтську та серверну частини. Це дозволяє оптимізувати архітектуру, масштабованість і ефективність обробки даних.

Клієнтська частина (front-end, клієнт) – це частина додатку, яка виконується на пристрої користувача (наприклад, смартфоні). У межах даного проєкту клієнтська частина реалізована за допомогою Flutter – кросплатформеного фреймворка, що дозволяє створювати інтерфейси користувача,

відображати графіки, керувати Bluetooth-з’єднанням із фітнес-браслетом та взаємодіяти із сервером через API.

Серверна частина (back-end, сервер) – програмна логіка, яка обробляє запити клієнтів, зберігає, обробляє і повертає дані. У нашому випадку серверна частина реалізована з використанням Node.js і Express.js – ця комбінація дозволяє будувати REST/WebSocket API, керувати сесіями, перевіряти автентичність користувача, генерувати відповіді на запити клієнта та взаємодіяти з базою даних PostgreSQL.

Flutter було обрано як основну платформу для створення клієнтської частини додатку. Flutter – це фреймворк з відкритим вихідним кодом, розроблений компанією Google у 2017 році. Його головна перевага полягає в тому, що він дозволяє створювати нативні мобільні застосунки під Android та iOS з єдиної кодової бази, використовуючи мову програмування Dart.

Flutter активно розвивається та підтримується Google, має потужну екосистему віджетів, що дозволяє розробникам створювати привабливий і адаптивний UI без потреби в окремому верстуванні для кожної платформи. Завдяки технології hot reload процес розробки значно пришвидшується. Серед інших переваг – висока продуктивність, бо Flutter рендерить UI безпосередньо через власний графічний рушій Skia, минаючи стандартні компоненти Android/iOS.

Node.js – це середовище виконання JavaScript на сервері, засноване на рушії V8 від Google. Воно дозволяє створювати масштабовані й високопродуктивні серверні додатки з використанням неблокуючої моделі обробки вводу/виводу. Node.js з'явився у 2009 році й з того часу став основним інструментом у багатьох веб-проєктах.

Express.js – це мінімалістичний вебфреймворк для Node.js, який забезпечує базову структуру для створення REST API, маршрутизації, обробки запитів і відповідей. Його простота й гнучкість дозволяє адаптувати серверну частину під специфіку фітнес-додатку, реалізувати авторизацію, обробку push-повідомлень та обробку користувацьких даних.

Для зберігання і обробки даних у фітнес-додатку обрано PostgreSQL – потужну об'єктно-реляційну СКБД з відкритим вихідним кодом, яка активно розвивається з 1996 року. PostgreSQL підтримує транзакції, складні SQL-запити, типи JSONB, а також геодані через розширення PostGIS. Це робить її ідеальною платформою для зберігання нормалізованих таблиць активності користувача (сон, пульс, кроки), маршрутів тренувань (GPS-треки) та структурованих даних. Її надійність, швидкодія та гнучка схема безпеки відповідають вимогам щодо захисту персональних даних користувачів, зокрема вимогам GDPR.

Хмарні сервіси (cloud services) – це зовнішні сервіси, які надають інфраструктуру, інструменти або готові рішення через Інтернет. Вони використовуються як частина backend-архітектури або як доповнення до неї.

У проєкті використовується Firebase Cloud Messaging (FCM) – це хмарна система для надсилання push-повідомлень. Firebase Cloud Messaging (FCM) від Google використовується для надсилання push-нотифікацій на мобільні пристрої. FCM підтримує як односторонні, так і двосторонні повідомлення, може обробляти персоналізовані запити та дії, зокрема сповіщення про нові тренування, рекомендації або пропущену активність.

Firebase легко інтегрується з Flutter, підтримує масштабованість, має зручний SDK та механізми автентифікації. Це забезпечує ефективну зворотну комунікацію з користувачем і підвищує його залученість.

Відмінність хмарних сервісів від баз даних полягає в їх призначенні та способі взаємодії з системою. База даних (у нашому випадку PostgreSQL) є ядром для зберігання всієї інформації про користувача: кроки, пульс, фази сну, параметри тренувань. Вона працює за допомогою SQL-запитів і забезпечує довготривале збереження, аналітику й цілісність даних. Натомість хмарні сервіси, як-от Firebase, не виконують зберігання основної інформації, а забезпечують комунікацію, обробку подій або аналітику. Вони працюють за API-принципом, часто в режимі реального часу, та не замінюють базу даних, а лише доповнюють її з метою покращення взаємодії з користувачем.

Обрана архітектура – клієнт-серверна, що забезпечує чіткий поділ відповідальностей між клієнтом (UI/збір даних) та сервером (обробка, зберігання, логіка). У серверній частині застосовується Clean Architecture, що передбачає поділ логіки на шари: контролери (інтерфейс), сервіси (бізнес-логіка), репозиторії (робота з БД). Це полегшує тестування, масштабування й повторне використання компонентів.

REST API – стандарт взаємодії між клієнтом і сервером, що забезпечує передбачуваність, кросплатформеність та підтримку широким спектром інструментів.

Для реалізації фітнес-додатку було використано сучасне програмне забезпечення, яке забезпечує ефективну розробку як клієнтської, так і серверної частин системи. Основними середовищами стали Visual Studio Code та Android Studio.

Visual Studio Code (VS Code) – це легкий, але потужний редактор коду з відкритим вихідним кодом, розроблений компанією Microsoft. Завдяки широкій екосистемі розширень, він забезпечує повноцінну підтримку мов Dart та JavaScript, а також фреймворків Flutter і Node.js, що робить його ідеальним інструментом для створення повного стеку додатку. VS Code також забезпечує інтелектуальне автодоповнення, налагодження коду (debugging), інтеграцію з Git та можливість швидкого запуску мобільного застосунку у емуляторі або на фізичному пристрої.

Android Studio – офіційне середовище розробки для Android-додатків, створене на основі IntelliJ IDEA. Воно надає розширений функціонал для розробників мобільних застосунків, зокрема повну підтримку емуляторів Android, профілювання продуктивності, налаштування збірки Gradle та створення APK-файлів. Завдяки глибокій інтеграції з Flutter SDK, Android Studio дозволяє виконувати повноцінну компіляцію і тестування застосунку перед розгортанням.

Комбінація цих двох середовищ дозволяє зручно і ефективно керувати всіма етапами розробки – від написання коду до налагодження і тестування.

Використання такого ПЗ забезпечує короткий цикл зворотного зв’язку, що особливо важливо в умовах ітеративної розробки, характерної для сучасного підходу до створення MVP-продуктів. З огляду на постійно зростаючу складність мобільних систем і необхідність підтримки інтеграції з хмарними сервісами, вибране ПЗ є оптимальним за критеріями гнучкості, продуктивності та зручності для розробника.

РОЗДІЛ 3
РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ

3.1 Практична реалізація об’єкта проєктування

У розділі практичної реалізації наведено приклади інтерфейсів мобільного фітнес-додатку, а також представлено фрагменти структури бази даних, яка використовується для зберігання інформації про користувача.

На головному екрані (рис. 3.1) відображається зведена інформація щодо активності користувача: кількість кроків, дистанція, калорії, пульс, фази сну, а також історія тренувань. Для зручності додано кнопку «плюс» у правому верхньому куті, яка дозволяє швидко запустити нову активність.

На екрані профілю (рис. 3.1) реалізовано можливість додавання нового фітнес-пристрою, перегляду заряду батареї підключеного браслета та встановлення щоденної цілі кроків. Ім’я користувача також відображається у верхній частині.

Під цією логікою працює таблиця client_basse у PostgreSQL, що містить унікальний ідентифікатор користувача id_account, ім’я облікового запису name_account, а також додаткові метадані у полі meta_client.

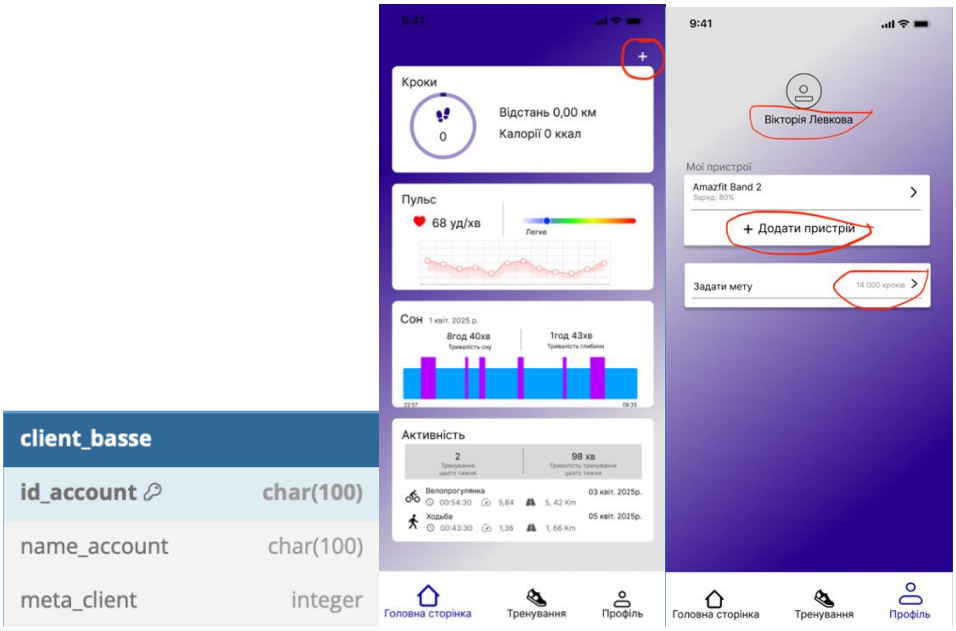

Рисунок 3.1 – Сторінка «Головний екран» та «Профіль»

На сервер надсилається HTTP POST-запит на маршрут /devices/add, що відповідає за додавання нового пристрою до облікового запису користувача після успішного підключення через Bluetooth.

Коли користувач натискає кнопку «+ Додати пристрій» (вкладка профіль) або «+» (вкладка головна сторінка), мобільний додаток ініціює процес сканування доступних Bluetooth-пристроїв поблизу через Bluetooth LE (BLE). У лістингу 3.1 подано інформацію успішного з'єднання пристрою (ідентифікатор (id_account), назва, заряд) надсилається на сервер через REST API, перевіряє чи пристрій вже існує, а ні – то тіло запиту (JSON).

Лістинг 3.1 – JSON-запит, новий обліковий запис користувача

```
{
  «id_account»: {id},
  «name_account»: NULL,
  «meta_client»: NULL
}
```

кінець лістингу 3.1

Цей фрагмент ілюструє PUT-запит до маршруту /users/updateName, який використовується для оновлення імені користувача в полі name_account таблиці client_basse. Така операція дозволяє змінювати збережене ім'я користувача після створення облікового запису або підключення пристрою.

Коли користувач змінює своє ім'я (ПІБ) у профілі додатку, нове значення надсилається на сервер через REST API. Сервер виконує оновлення поля «name_account» у таблиці «client_basse» за вказаним «id_account». Тіло запиту (JSON) представлено в лістингу 3.2.

Лістинг 3.2 – JSON-запит, новий обліковий запис користувача

```
{
  «id_account»: {id},
```

```
  «name_account»: {name},
}
```

кінець лістингу 3.2

Цей запит PUT /users/updateName використовується для оновлення цілі користувача – поля meta_client у таблиці client_basse. Значенням цієї цілі є кількість кроків, яку користувач встановлює як свою денну норму.

Коли користувач у додатку змінює свою мету, нове значення надсилається на сервер через REST API. Тіло запиту (JSON) представлено в лістингу 3.3.

Лістинг 3.3 – JSON-запит, новий обліковий запис користувача

```
{
  «id_account»: {id},
  «meta_client»: {meta_integer}
}
```

кінець лістингу 3.3

На рисунку 3.2 представлено реалізацію відображення події «кроки» у мобільному фітнес-додатку. Зліва наведена структура таблиці steping у базі даних, яка зберігає інформацію про кількість зроблених кроків, пройдену відстань (distinct_trenning_km), витрачені калорії та час фіксації. Кожен запис прив’язаний до конкретного користувача через id_account.

Центральна та права частини рисунка демонструють інтерфейс користувача в мобільному додатку. На головному екрані користувач бачить загальну кількість кроків, відстань і спалені калорії. У розгорнутій статистиці (екран справа) відображено деталізацію за днями, часом активності та прогресом щодо встановленої мети.

Цей механізм дозволяє відстежувати активність користувача в реальному часі та зберігати історію фізичної активності для подальшого аналізу.

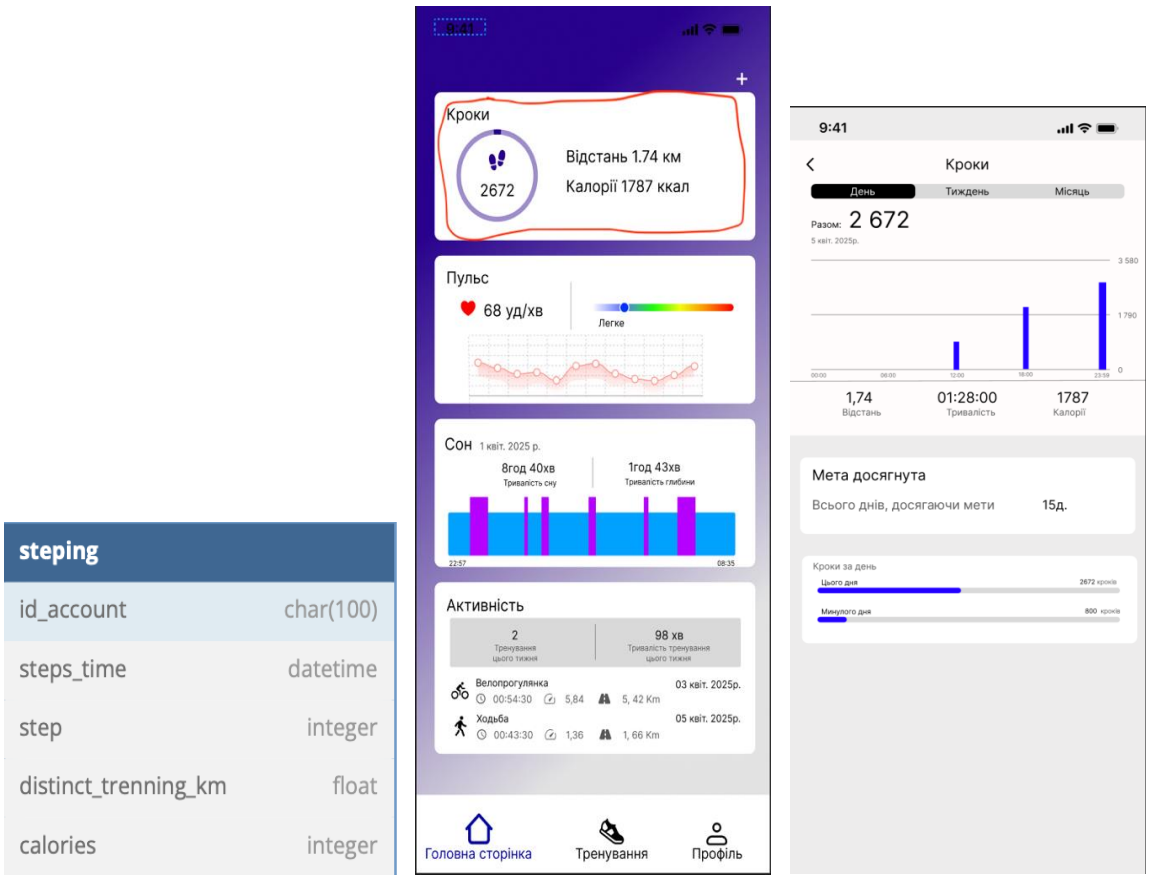

Рисунок 3.2 – Сторінка «Кроки»

Цей запит GET /steps/{id_account}?period={PERIOD} дозволяє отримати інформацію про кількість кроків користувача за вказаний період – день, тиждень або місяць. Він використовується для побудови аналітики активності у додатку.

Коли користувач відкриває розділ «Кроки» і перемикає вкладки («День», «Тиждень», «Місяць»), додаток надсилає GET-запит із параметром period, який вказує потрібний часовий діапазон.

Обчислення всього днів, досягаючи мети обчислюється за формулою (1):

$$\frac{meta_{step}}{step_{avg}} = \frac{meta_{step}}{step_{sum} \div count_{day}} \quad , \tag{1}$$

де $meta_{step}$ – кількість кроків задана для мети користувача;

$step_{sum}$ – сума кількості кроків за всесь обраний період;

$count_{day}$ – кількість днів в які користувач бігав.

На рисунку 3.3 зображено реалізацію функціональності моніторингу пульсу у мобільному додатку для фітнес-трекінгу. Ця складова є важливим елементом загальної системи збору фізіологічних показників і дозволяє здійснювати безперервний контроль серцевої активності користувача в реальному часі. Фіксація пульсу забезпечується зчитуванням даних з носимого пристрою через протокол Bluetooth Low Energy, після чого показники зберігаються у базі даних і відображаються у візуальному форматі в інтерфейсі застосунку. Зліва наведено структуру таблиці pulsing, у якій зберігається:

1) id_account – ідентифікатор користувача;
2) puls_time – мітка часу вимірювання пульсу;
3) heartbeats – кількість ударів серця за хвилину.

Праворуч – фрагменти інтерфейсу:

1) основний екран з поточним пульсом і динамікою за день;
2) деталізований перегляд з гістограмою змін пульсу, середнім, мінімальним і максимальним значенням;
3) текстове пояснення важливості цього показника.

Вся функціональність реалізована з урахуванням можливості локального збереження даних при відсутності мережі, синхронізації з сервером, а також надсилання push-сповіщень у разі виходу значень пульсу за межі допустимих норм. Система може автоматично реагувати на критичні значення, формуючи повідомлення на основі account_id, що дозволяє адресно сповіщати користувача про необхідність звернення уваги на стан свого організму.

Таким чином, реалізація сторінки «Пульс» об'єднує медичну доцільність, естетичний інтерфейс і технічну надійність, що дозволяє користувачу ефективно відстежувати стан свого серцево-судинного здоров’я у щоденному житті. Інформація про пульс оновлюється в реальному часі, що створює відчуття безперервного контролю за фізіологічними показниками. Такий підхід не лише підвищує обізнаність користувача про власний стан, а й формує здорові звички та мотивацію до регулярної фізичної активності.

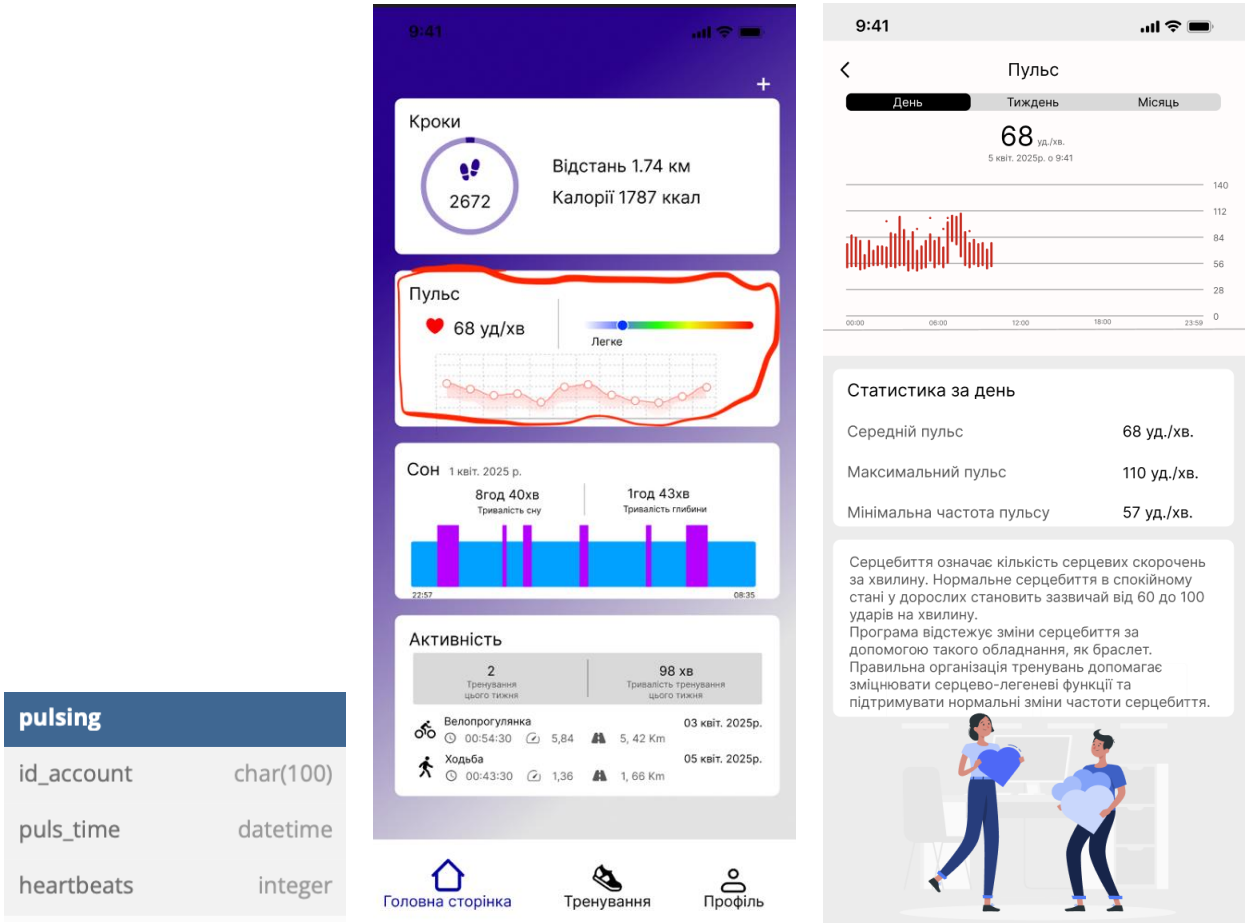

Рисунок 3.3 – Сторінка «Пульс»

Цей API-запит типу GET за маршрутом /pulse/{id_account}?period ={PERIOD} призначений для отримання історії пульсу користувача за обраний період. Параметр period дозволяє вказати бажаний інтервал часу – день, тиждень або місяць – і отримати відповідні дані про частоту серцевих скорочень, збережені у таблиці pulsing.

На рисунку 3.4 представлено реалізацію відображення події сну в мобільному додатку. Користувач має змогу переглядати тривалість сну за обраний день, а також візуалізувати його структуру – зокрема глибокі та швидкі фази сну.

Зліва показано фрагмент таблиці sleeping, яка містить такі поля:

1) id_account – ідентифікатор користувача;
2) sleep_time – дата і час початку фіксації сну;
3) sleep_type – тип сну (глибокий чи швидкий);

4) deep_sleep – тривалість глибокого сну у хвилинах.

Ці дані використовуються для побудови графіка, що допомагає користувачу слідкувати за якістю сну, а також оптимізувати власний режим відпочинку.

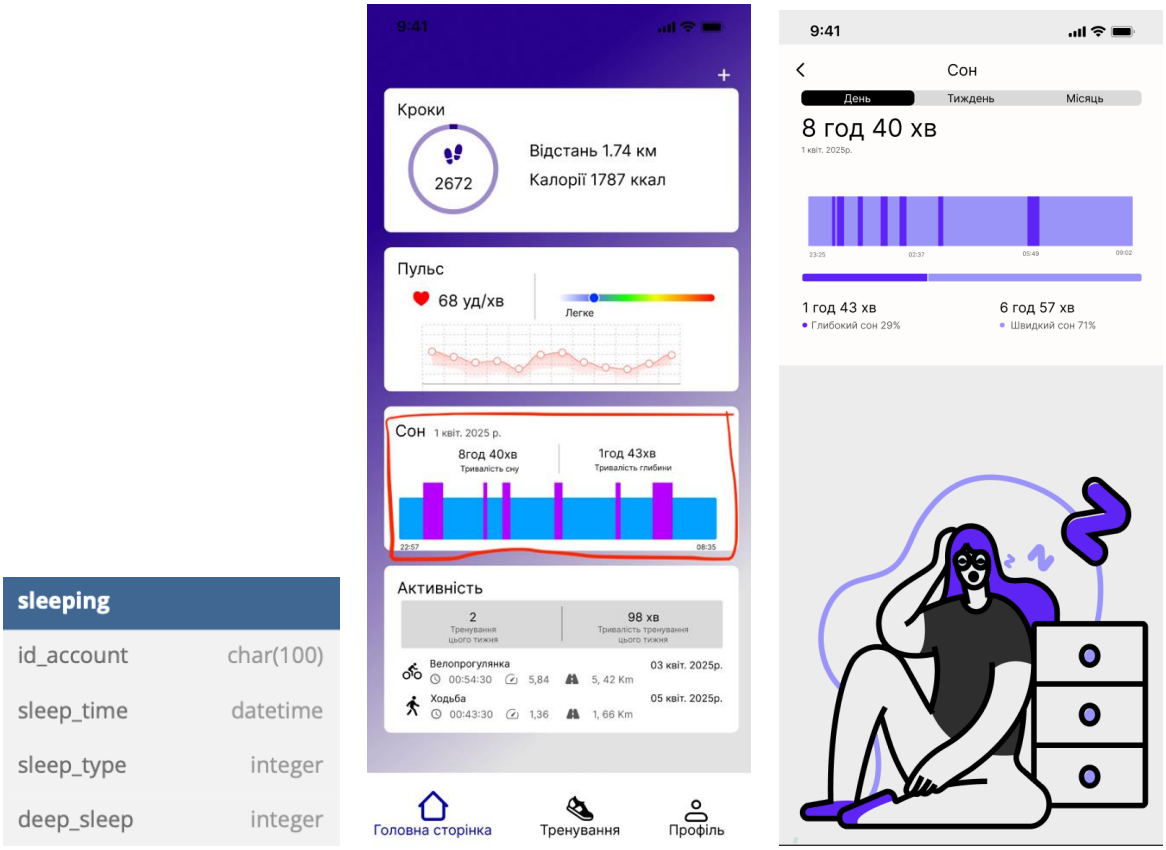

Рисунок 3.4 – Сторінка «Сон»

Цей запит GET /sleep/{id_account}?period={PERIOD} використовується для отримання даних про сон користувача за обраний період (наприклад, день, тиждень або місяць).

Коли користувач відкриває розділ «Сон» і перемикає вкладки («День», «Тиждень», «Місяць»), додаток надсилає GET-запит із параметром period.

Якщо sleep_type = 0, то користувач спить, інакше не спить.

Якщо deep_sleep = 0, то користувач спить у глибокому сні.

Отримана інформація використовується для візуалізації в мобільному додатку, зокрема для побудови графіка фаз сну, тривалості відпочинку тощо. У лістингу 3.4 представлено SQL-запит зі змінним періодом для сну.

Лістинг 3.4 – SQL-запит зі змінним період для сну

```
SELECT
CAST(sleep_time AS DATE),
 sleep_type,
deep_sleep
FROM sleeping
WHERE id_account = :id_account
AND sleep_time >= CURRENT_DATE - INTERVAL :period
ORDER BY sleep_time;
```

кінець лістингу 3.4

Фітнес-браслет визначає, таблиця 3.1, чи спить користувач, і в якій саме фазі, на основі даних від кількох датчиків:

1) оптичний пульсометр (PPG). Вимірює пульс і варіабельність пульсу (HRV). Якщо пульс стабільно низький – це ознака глибокого сну;

2) акселерометр. Відстежує мікрорухи тіла. Якщо руху немає – користувач спить, імовірно в глибокому сні;

3) гіроскоп. Фіксує положення тіла. Допомагає уточнювати, чи змінилася позиція під час сну;

4) температура шкіри / SpO_2. Впливають незначно, доступні лише у браслетах із розширеними функціями.

Таблиця 3.1 – Фаза сну

Ознаки стану користувача	Інтерпретація фази
1	2
Людина нерухома, пульс низький	Глибокий сон
Є мікрорухи, пульс трохи коливається	Поверхневий (швидкий) сон
Часті рухи, високий пульс	Не спить

На рисунку 3.5 зображено реалізацію модуля відстеження фізичної активності користувача. Дані про активність зберігаються в таблиці customer_activity, яка містить такі поля: id_account, type_trenning, start_ trenning, end_trenning, distinct_trenning_km.

У додатку візуалізовано інформацію про кількість проведених тренувань, їх тривалість та тип (наприклад, ходьба, біг, велопрогулянка). Також виводиться статистика щодо пройденої дистанції. Всі ці дані можна переглядати за день, тиждень, місяць або повністю.

Цей функціонал допомагає користувачу відстежувати рівень своєї фізичної активності та формувати здорові звички.

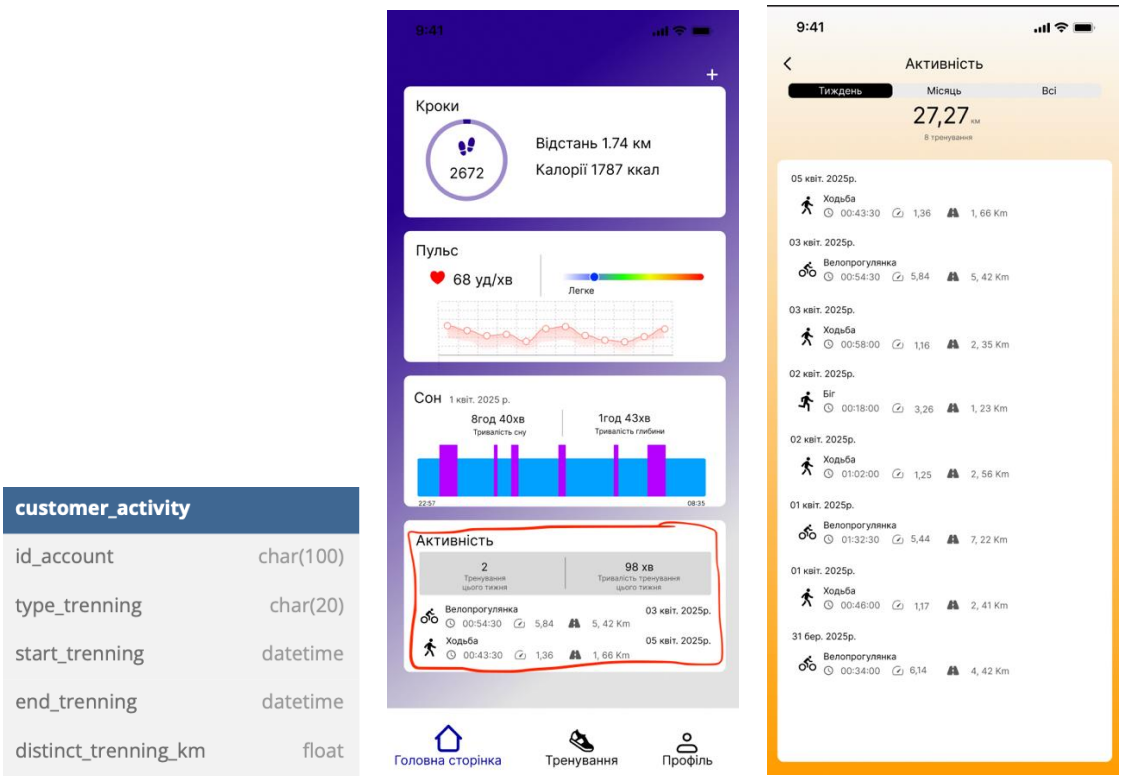

Рисунок 3.5 – Сторінка «Активність»

Цей запит GET /activity/{id_account}?period={PERIOD} використовується для отримання історії фізичної активності користувача за обраний період (наприклад, день, тиждень або місяць).

Отримана інформація використовується для візуалізації в мобільному додатку, зокрема для відображення типу тренування, його тривалості, подоланої дистанції та часу початку/завершення кожної активності.

На рисунку 3.6 зображена сторінка «Тренування», яка є ядром функціоналу фітнес-додатку. Вона відповідає за:

1) запуск фізичної активності;

2) фіксацію геолокації;

3) реєстрацію часу, дистанції, швидкості;

4) передачу даних на сервер та збереження у PostgreSQL.

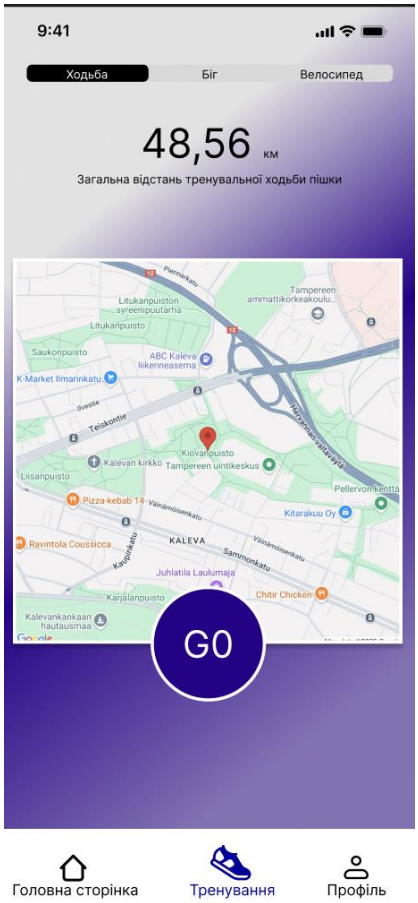

Рисунок 3.6 – Подія тренування

В таблиці 3.2 зображена більш детальна інформація про сторінку «Тренування».

Таблиця 3.2 – UI-компоненти (Flutter)

Елемент	Призначення
1	2
Вкладки (TabBar)	Обирається тип тренування: Ходьба, Біг, Велосипед
Кнопка GO	Запускає трекінг.
Карта (GoogleMap)	Відображає місцезнаходження та дозволяє поставити мітку, маршрут не прокладає.
Статистика	Показує поточну дистанцію, час, калорії.

Користувач обирає тип тренування зверху: Ходьба, Біг, Велосипед → це значення type_trenning.

При натисканні GO:

1) обнуляються попередні показники: час, маршрут, кроки, калорії;

2) кожні 30 мс зчитується GPS-позиція;

3) щохвилини локально:

– додається 1 хвилина до тривалості;

– обраховується пройдена дистанція;

– розраховується швидкість (відстань / час);

– фіксується пульс.

Запити POST в даному випадку для нашої сторінки «Тренування» використовуються запис щохвилини даних у відповідні таблиці бази даних. Зокрема:

1) POST /customer_activity/minute-log – записує дані про фізичну активність користувача у таблицю customer_activity;

2) POST /steping/minute-log – фіксує кількість кроків користувача у таблиці steping;

3) POST /sleeping/minute-log – зберігає інформацію про сон користувача у таблиці sleeping;

4) POST /pulsing/minute-log – передає дані про пульс у таблицю pulsing.

Ці запити забезпечують регулярне оновлення даних для їх подальшого аналізу та візуалізації в мобільному застосунку.

Для забезпечення зручного інформування користувачів про події, досягнення та нагадування в мобільному фітнес-додатку було реалізовано систему push-сповіщень із використанням Firebase Cloud Messaging (FCM).

Процес реалізації включав наступні етапи:

1) інтеграція Firebase з Flutter-додатком. У проєкт було додано пакет firebase_messaging, а також налаштовано Firebase Console;

2) створено Firebase-проєкт;

3) додано Android- (google-services.json / GoogleService-Info.plist);

4) увімкнено Firebase Cloud Messaging;

5) реєстрація токена пристрою.

Токен представлений у лістингу 3.2 зберігається в базі даних (наприклад, у PostgreSQL або у Firebase Firestore) для подальшого надсилання повідомлень.

Лістинг 3.5 – Реєстрація токена пристрою

```
FirebaseMessaging messaging = FirebaseMessaging.instance;
String? token = await messaging.getToken();
```

кінець лістингу 3.5

Отримання повідомлень у фоновому та активному режимах представлено лістингу 3.3.

Лістинг 3.6 – Реєстрація токена пристрою

```
FirebaseMessaging.onMessage.listen((RemoteMessage message) {
});
FirebaseMessaging.onBackgroundMessage(_firebaseMessagingBackgroundHandle
r);
```

кінець лістингу 3.6

Розробка бекенду для надсилання повідомлень. На сервері (Node.js + Express.js) створено REST-ендпоінт для надсилання сповіщень предсталена частина коду лістингу 3.4.

Лістинг 3.7 – надсилання повідомлень Node.js + Express.js

```
const admin = require(«firebase-admin»);
admin.initializeApp({
   credential: admin.credential.cert(serviceAccount)
    });

app.post(«/sendNotification», async (req, res) => {
   const { token, title, body } = req.body;
   const message = {
   notification: { title, body },
```

```
   token: token,
};
try {
   const response = await admin.messaging().send(message);
   res.status(200).send(«Notification sent: « + response);
} catch (error) {
   res.status(500).send(«Error sending: « + error);
     }
 });
```

кінець лістингу 3.7

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

Тестування інформаційних систем є важливою фазою життєвого циклу програмного забезпечення, під час якої здійснюється перевірка відповідності реалізованого функціоналу встановленим вимогам, а також виявлення помилок до моменту розгортання системи. Метою тестування є забезпечення високої якості програмного продукту, підвищення надійності та зниження витрат на його подальший супровід. Згідно з загальноприйнятою класифікацією, тестування поділяється на кілька рівнів: модульне (unit testing), інтеграційне (integration testing), системне (system testing) та приймальне (acceptance testing).

Модульне тестування передбачає перевірку окремих функціональних блоків або методів у відриві від решти системи. Воно дозволяє локалізувати помилки на рівні окремих функцій, класів або модулів. Інтеграційне тестування має на меті перевірити взаємодію між різними частинами системи: клієнтом, сервером, базою даних, зовнішніми сервісами. Функціональне тестування орієнтоване на поведінку системи з точки зору користувача та перевірку коректності виконання бізнес-логіки.

Налагодження системи (debugging) тісно пов’язане з тестуванням і полягає у виявленні причин помилок, їх виправленні, а також профілюванні

продуктивності програми. Важливою частиною процесу є також логування подій та помилок, що дозволяє проводити постаналіз роботи програми та запобігати повторенню збоїв.

У межах даної роботи тестування та налагодження стали завершальним етапом розробки інформаційно-комп'ютерної системи, спрямованим на перевірку її функціональної коректності, надійності та відповідності вимогам технічного завдання. У рамках цього етапу було реалізовано модульне, інтеграційне та функціональне тестування, що дозволило комплексно оцінити якість реалізованого продукту.

Модульне тестування виконувалося для окремих серверних компонентів, реалізованих на Node.js. Перевірялась правильність обробки запитів REST API, валідація даних, точність алгоритмів розрахунку, а також обробка виняткових ситуацій.

Інтеграційне тестування охоплювало перевірку злагодженої роботи між клієнтською частиною, реалізованою на Flutter, та серверною частиною на базі Express.js. Зокрема, тестувалась стабільність взаємодії з базою даних PostgreSQL, обмін даними у форматі JSON, а також передача сигналів у реальному часі через WebSocket. Важливим аспектом було також тестування надсилання push-повідомлень через Firebase Messaging.

У результаті було виявлено, що повна сумісність з фітнес-браслетами забезпечується лише на моделі Mi Band 7, яка коректно передавала дані про пульс і кроки у фоновому режимі. Цей аспект вимагає подальшої інтеграції з платформами Google Fit та HealthKit.

Окрім того, у ході тестування було виявлено, що графічний інтерфейс дещо розмивається або деформується на екранах з нетиповими роздільними здатностями (наприклад, на дуже вузьких або надширокиx дисплеях). Це пов'язано з особливостями масштабування у Flutter та необхідністю додаткової адаптивної верстки. Проблему частково усунуто шляхом використання гнучких

контейнерів і перевірки розмірів через MediaQuery, однак подальша оптимізація інтерфейсу залишається актуальним напрямом роботи.

Налагодження охоплювало систематичний перегляд логів, профілювання продуктивності та оптимізацію SQL-запитів. низку змін, що зменшили середній час відповіді API.

Загалом тестування підтвердило функціональність і базову стабільність системи, хоча для повноцінного комерційного впровадження необхідна подальша адаптація інтерфейсу під ширший спектр пристроїв і розширення підтримки носимих гаджетів

3.3 Розробка документації для програмного продукту

Коротко опишемо етапи інсталяції та запуску програми:

1) запустіть додаток на смартфоні;

2) у вкладці «Профіль» натисніть «+» Додати пристрій» або вкладка «Головна сторінка» натисніть «+»;

3) додаток автоматично увімкне Bluetooth та перейде в налаштування для пошуку пристрою;

4) оберіть ваш браслет зі списку;

5) після успішного з’єднання пристрій буде збережений і відображений у профілі;

6) автоматично додаток підтягне дані якщо вони є у БД інакше зареєструє автоматично пристрій;

7) у вкладці «Профіль» заповніть ім’я та мету в кроках, яку бажаєте долати протягом тижня;

8) перейдіть на вкладку «Тренування» для початку тренування та натисність кнопку «GO»:

– щохвилини відбувається збір GPS-даних і пульсу;

– визначається швидкість, пройдена відстань, витрачені калорії;

– дані записуються у вашу статистику;

9) по завершенню – натисніть кнопку «STOP». Всі дані збережуться в історії та будуть відображені.

ВИСНОВКИ

Було здійснено аналіз існуючих рішень у сфері фітнес-додатків. У процесі аналітичного етапу було досліджено сильні та слабкі сторони популярних додатків, таких як Mi Fit, Google Fit та інші. Виявлено, що більшість з них орієнтовані на глобальний ринок і не враховують специфіку українських користувачів – зокрема, відсутня локалізація, обмежений безкоштовний функціонал, реклама та складна навігація. Це стало основою для подальших вимог до власного рішення.

Було сформовано функціональні та нефункціональні вимоги до системи. Основні функції включали збір даних про кроки, пульс, сон, тренування, побудову статистики та нагадування. Серед нефункціональних вимог – підтримка Android 9+, кешування даних без інтернету щонайменше на 24 години, затримка відображення не більше 200 мс, збереження безпеки даних і зручність використання.

Було обґрунтовано вибір інструментів та архітектури реалізації. Клієнтську частину реалізовано на Flutter, що забезпечує кросплатформеність і гнучкий інтерфейс. Серверну частину реалізовано на Node.js з Express.js, а PostgreSQL обрано як реляційну СКБД. Firebase було інтегровано для реалізації push-сповіщень. Архітектура «клієнт-сервер» забезпечила гнучкість і масштабованість.

Було здійснено проєктування структури клієнтської та серверної частин. Розроблено логічну модель бази даних, зокрема сутності activity, sleep, pulse, steps. Створено REST API за принципами Clean Architecture, що забезпечує ізоляцію рівнів логіки та зручність масштабування. Реалізовано систему авторизації, валідації даних і обробки запитів на сервері.

Було реалізовано мобільний застосунок на Flutter. Користувачу надано інтуїтивно зрозумілий інтерфейс із доступом до основних функцій: перегляд кроків, пульсу, якості сну, статистики активностей, тренувань і налаштування профілю. Додаток адаптовано під різні розміри екранів і підтримує онбординг

для нових користувачів. Окрема увага приділена відображенню даних у вигляді графіків і дашбордів з оновленням у реальному часі.

Було створено серверне API на Node.js/Express.js, яке забезпечує прийом, обробку та збереження даних у PostgreSQL. Реалізовано логіку маршрутизації, авторизації запитів із перевіркою токена, обробку помилок і підключення WebSocket для передачі пульсу в режимі live. Сервер працює із JSON-запитами, виконує валідацію даних і підтримує буферизацію при нестабільному інтернет-з’єднанні, що дозволяє уникнути втрати інформації.

Було здійснено інтеграцію з Firebase для реалізації push-сповіщень. Система надсилає нагадування про тренування, повідомлення про досягнення цілей або тривалий період без активності. Це дозволяє покращити мотивацію користувача та забезпечити регулярну взаємодію з додатком. Використання Firebase Cloud Messaging дає змогу надсилати повідомлення вибірково – залежно від поведінки користувача, за допомогою унікального account_id.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. PostgreSQL Documentation. PostgreSQL 15. URL: https://www .postgresql.org/docs/ (дата звернення: 03.05.2025).

2. Flutter Documentation. Build apps for any screen. URL: https://docs.flutter.dev/ (дата звернення: 17.04.2025).

3. Node.js Documentation. Node.js v20.x Docs. URL: https://nodejs.org/en/docs/ (дата звернення: 11.04.2025).

4. Express.js Guide. Express 4.x API Reference. URL: https://expressjs.com/ (дата звернення: 14.04.2025).

5. WHOOP Research. Sleep Staging Algorithm Methodology. URL: https://www.whoop.com/thelocker/sleep-stages-explained/ (дата звернення: 21.03.2025).

6. Google. Activity Recognition API. URL: https://developers.google.com/location-context/activity-recognition/ (дата звернення: 27.03.2025).

7. Fittrack. How Are Calories Burned Calculated? URL: https://getfittrack.com/ (дата звернення: 29.04.2025).

8. Grand View Research. Fitness App Market Size To Reach USD 23.21 Billion by 2030: Global Report. April 2025. URL: https://www.grandviewresearch.com/press-release/global-fitness-app-market (дата звернення: 03.04.2025).

9. IDC. Wearable Devices Market Insights – global wearables reached 534.6 M units in 2024 with 5.4 % YoY growth. May 2025. URL: https://www.idc.com/promo /wearablevendor/ (дата звернення: 24.05.2025).

10. Головченко Т. С. Розробка мобільних додатків: навч. посіб. Київ: Видавництво Ліра К, 2020. 248 с.

11. GoodFirms. Flutter 2025: Definition, Key Trends & Statistics. URL: https://www.goodfirms.co/blog/flutter-2025-definition-key-trends-statistics (дата звернення: 05.03.2025).

12. Шевченко І. А. Основи клієнт-серверної архітектури. Харків: ХНУРЕ. 2021. 150 с.

13. Медведєв Д. Ю. Розробка REST API сервісів з використанням Node.js та Express.js. Одеса: ОНУ. 2022. 112 с.

14. Дяченко В. М. Firebase у мобільній розробці: хмарні рішення. Львів: ЛНУ. 2021. 96 с.

15. Webnauts. React Native Mobile App for Training. URL: https://wnauts.com/project/react-native-mobile-app-for-training (дата звернення: 21.04.2025).

ДОДАТКИ

Додаток А – Лістинг програми

Flutter

```
import 'package:flutter/material.dart';
import 'package:fl_chart/fl_chart.dart';

void main() {
  runApp(const HealthApp());
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Health App UI',
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        brightness: Brightness.dark,
        primaryColor: const Color(0xFF8A2BE2),
        scaffoldBackgroundColor: const Color(0xFF1C1C2E),
        textTheme: const TextTheme(
          bodyLarge: TextStyle(color: Colors.white),
          bodyMedium: TextStyle(color: Colors.white70),
        ),
        cardTheme: CardTheme(
          color: const Color(0xFF2A2A4A),
          elevation: 0,
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(16.0),
          ),
        ),
      ),
      home: const HomePage(),
    );
  }
}

class HomePage extends StatefulWidget {
  const HomePage({super.key});

  @override
```

```
  State<HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  int _selectedIndex = 0;

  void _onItemTapped(int index) {
    setState(() {
      _selectedIndex = index;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('9:41', style: TextStyle(fontWeight: FontWeight.bold)),
        backgroundColor: const Color(0xFF1C1C2E),
        elevation: 0,
        centerTitle: true,
        actions: [
          IconButton(
            icon: const Icon(Icons.add),
            onPressed: () {},
          ),
        ],
      ),
      body: SingleChildScrollView(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          children: const [
            StepsCard(),
            SizedBox(height: 16),
            PulseCard(),
            SizedBox(height: 16),
            SleepCard(),
            SizedBox(height: 16),
            ActivityCard(),
          ],
        ),
      ),
      bottomNavigationBar: BottomNavigationBar(
        items: const <BottomNavigationBarItem>[
          BottomNavigationBarItem(
            icon: Icon(Icons.home),
```

```
          label: 'Головна сторінка',
        ),
        BottomNavigationBarItem(
          icon: Icon(Icons.fitness_center),
          label: 'Тренування',
        ),
        BottomNavigationBarItem(
          icon: Icon(Icons.person),
          label: 'Профіль',
        ),
      ],
      currentIndex: _selectedIndex,
      onTap: _onItemTapped,
      backgroundColor: const Color(0xFF2A2A4A),
      selectedItemColor: Colors.white,
      unselectedItemColor: Colors.white54,
      type: BottomNavigationBarType.fixed,
      showUnselectedLabels: true,
    ),
   );
  }
}

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

  @override
  Widget build(BuildContext context) {
    return Card(
      child: Padding(
        padding: const EdgeInsets.all(20.0),
        child: Row(
          children: [
            SizedBox(
              width: 100,
              height: 100,
              child: Stack(
                alignment: Alignment.center,
                children: [
                  const CircularProgressIndicator(
                    value: 0.75,
                    strokeWidth: 8,
                    backgroundColor: Colors.white24,
                    valueColor: AlwaysStoppedAnimation<Color>(Color(0xFF8A2BE2)),
                  ),
```

```
                Column(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: const [
                    Icon(Icons.footsteps, color: Color(0xFF8A2BE2), size: 28),
                    Text(
                      '2672',
                      style: TextStyle(
                        fontSize: 22,
                        fontWeight: FontWeight.bold,
                      ),
                    ),
                  ],
                ),
              ],
            ),
          ),
          const SizedBox(width: 24),
          Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: const [
              Text(
                'Відстань 1.74 км',
                style: TextStyle(fontSize: 16),
              ),
              SizedBox(height: 8),
              Text(
                'Калорії 1787 ккал',
                style: TextStyle(fontSize: 16),
              ),
            ],
          ),
        ],
      ),
    ),
  );
 }
}

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

 @override
 Widget build(BuildContext context) {
  return Card(
   child: Padding(
```

```
      padding: const EdgeInsets.all(16.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          const Text('Пульс', style: TextStyle(fontSize: 18, fontWeight:
FontWeight.bold)),
          const SizedBox(height: 12),
          Row(
            children: [
              const Icon(Icons.favorite, color: Colors.red, size: 28),
              const SizedBox(width: 8),
              const Text('68 уд/хв', style: TextStyle(fontSize: 22, fontWeight:
FontWeight.bold)),
              const Spacer(),
              Column(
                crossAxisAlignment: CrossAxisAlignment.end,
                children: [
                  const Text('Легке', style: TextStyle(color: Colors.white70)),
                  Container(
                    width: 100,
                    height: 8,
                    decoration: BoxDecoration(
                      borderRadius: BorderRadius.circular(4),
                      gradient: const LinearGradient(
                        colors: [Colors.blue, Colors.green, Colors.yellow, Colors.orange,
Colors.red],
                      ),
                    ),
                  )
                ],
              )
            ],
          ),
          const SizedBox(height: 20),
          SizedBox(
            height: 100,
            child: LineChart(
              LineChartData(
                gridData: FlGridData(show: true, drawVerticalLine: false),
                titlesData: FlTitlesData(show: false),
                borderData: FlBorderData(show: false),
                lineBarsData: [
                  LineChartBarData(
                    spots: const [
                      FlSpot(0, 3), FlSpot(1, 1), FlSpot(2, 4), FlSpot(3, 2),
```

```
                  FlSpot(4, 5), FlSpot(5, 3), FlSpot(6, 4), FlSpot(7, 3),
                  FlSpot(8, 5), FlSpot(9, 6), FlSpot(10, 5),
                ],
                isCurved: true,
                color: Colors.redAccent,
                barWidth: 3,
                isStrokeCapRound: true,
                dotData: FlDotData(show: false),
                belowBarData: BarAreaData(
                  show: true,
                  gradient: LinearGradient(
                    colors: [Colors.red.withOpacity(0.3), Colors.red.withOpacity(0.0)],
                    begin: Alignment.topCenter,
                    end: Alignment.bottomCenter,
                  ),
                ),
              ),
            ],
          ),
        ),
      ),
    ),
  );
 }
}

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

 @override
 Widget build(BuildContext context) {
  return Card(
   child: Padding(
    padding: const EdgeInsets.all(16.0),
    child: Column(
     crossAxisAlignment: CrossAxisAlignment.start,
     children: [
      const Text('Сон', style: TextStyle(fontSize: 18, fontWeight:
FontWeight.bold)),
      const Text('1 квіт. 2025 р.', style: TextStyle(color: Colors.white70)),
      const SizedBox(height: 16),
      Row(
       mainAxisAlignment: MainAxisAlignment.spaceBetween,
```

```
            children: [
              _buildSleepInfo('8год 40хв', 'Тривалість сну'),
              _buildSleepInfo('1год 43хв', 'Тривалість глибини'),
            ],
          ),
          const SizedBox(height: 20),
          SizedBox(
            height: 100,
            child: BarChart(
              BarChartData(
                alignment: BarChartAlignment.spaceAround,
                barTouchData: BarTouchData(enabled: false),
                titlesData: FlTitlesData(
                  show: true,
                  bottomTitles: AxisTitles(
                    sideTitles: SideTitles(
                      showTitles: true,
                      getTitlesWidget: (value, meta) {
                        switch (value.toInt()) {
                          case 0: return const Text('22:57', style: TextStyle(fontSize: 10, color: Colors.white70));
                          case 9: return const Text('08:35', style: TextStyle(fontSize: 10, color: Colors.white70));
                        }
                        return const Text('');
                      },
                      reservedSize: 20,
                    ),
                  ),
                  leftTitles: AxisTitles(sideTitles: SideTitles(showTitles: false)),
                  topTitles: AxisTitles(sideTitles: SideTitles(showTitles: false)),
                  rightTitles: AxisTitles(sideTitles: SideTitles(showTitles: false)),
                ),
                gridData: FlGridData(show: false),
                borderData: FlBorderData(show: false),
                barGroups: [
                  _makeGroupData(0, 5, 2), _makeGroupData(1, 6.5, 3),
                  _makeGroupData(2, 5, 0), _makeGroupData(3, 7.5, 4),
                  _makeGroupData(4, 8, 0), _makeGroupData(5, 6, 2.5),
                  _makeGroupData(6, 6.5, 0), _makeGroupData(7, 7, 3),
                  _makeGroupData(8, 8, 4), _makeGroupData(9, 6, 0),
                ],
              ),
            ),
          ),
```

```
          ],
        ),
      ),
    );
  }

  Widget _buildSleepInfo(String value, String label) {
    return Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Text(value, style: const TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),
        Text(label, style: const TextStyle(color: Colors.white70, fontSize: 12)),
      ],
    );
  }

  BarChartGroupData _makeGroupData(int x, double y1, double y2) {
    return BarChartGroupData(
      x: x,
      barRods: [
        BarChartRodData(
          toY: y1,
          color: const Color(0xFF00BFFF),
          width: 8,
          borderRadius: BorderRadius.zero,
        ),
        BarChartRodData(
          toY: y2,
          color: const Color(0xFF8A2BE2),
          width: 8,
          borderRadius: BorderRadius.zero,
        ),
      ],
    );
  }
}

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

  @override
  Widget build(BuildContext context) {
    return Card(
      child: Padding(
        padding: const EdgeInsets.all(16.0),
```

```
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          const Text('Активність', style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),
          const SizedBox(height: 16),
          Row(
            children: [
              Expanded(
                child: Container(
                  padding: const EdgeInsets.all(12),
                  decoration: BoxDecoration(
                    color: Colors.white10,
                    borderRadius: BorderRadius.circular(12),
                  ),
                  child: Column(
                    children: const [
                      Text('2', style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold)),
                      Text('Тренування цього тижня', textAlign: TextAlign.center, style: TextStyle(color: Colors.white70, fontSize: 12)),
                    ],
                  ),
                ),
              ),
              const SizedBox(width: 16),
              Expanded(
                child: Container(
                  padding: const EdgeInsets.all(12),
                   decoration: BoxDecoration(
                    color: Colors.white24,
                    borderRadius: BorderRadius.circular(12),
                  ),
                  child: Column(
                    children: const [
                      Text('98 хв', style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold)),
                      Text('Тривалість тренування цього тижня', textAlign: TextAlign.center, style: TextStyle(color: Colors.white70, fontSize: 12)),
                    ],
                  ),
                ),
              ),
            ],
          ),
          const SizedBox(height: 16),
```

```
          _buildActivityRow(
            icon: Icons.directions_bike,
            title: 'Велопрогулянка',
            time: '00:54:30',
            distance: '5,84',
            calories: '5, 42 Km',
            date: '03 квіт. 2025р.',
          ),
          const Divider(color: Colors.white24),
          _buildActivityRow(
            icon: Icons.directions_walk,
            title: 'Ходьба',
            time: '00:43:30',
            distance: '1,36',
            calories: '1, 66 Km',
            date: '05 квіт. 2025р.',
          ),
        ],
      ),
    ),
  );
}

Widget _buildActivityRow({
  required IconData icon,
  required String title,
  required String time,
  required String distance,
  required String calories,
  required String date,
}) {
  return Padding(
    padding: const EdgeInsets.symmetric(vertical: 12.0),
    child: Column(
      children: [
        Row(
          children: [
            Icon(icon, color: Colors.white),
            const SizedBox(width: 12),
            Text(title, style: const TextStyle(fontWeight: FontWeight.bold)),
            const Spacer(),
            Text(date, style: const TextStyle(color: Colors.white70, fontSize: 12)),
          ],
        ),
        const SizedBox(height: 8),
```

```
          Row(
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: [
              _buildActivityDetail(Icons.timer_outlined, time),
              _buildActivityDetail(Icons.straighten, distance),
              _buildActivityDetail(Icons.local_fire_department_outlined, calories),
            ],
          )
        ],
      ),
    );
  }

  Widget _buildActivityDetail(IconData icon, String value) {
    return Row(
      children: [
        Icon(icon, color: Colors.white70, size: 16),
        const SizedBox(width: 4),
        Text(value, style: const TextStyle(color: Colors.white70)),
      ],
    );
  }
}

class PulseStatistics {
  final int average;
  final int max;
  final int min;

  PulseStatistics({required this.average, required this.max, required this.min});

  factory PulseStatistics.fromJson(Map<String, dynamic> json) {
    return PulseStatistics(
      average: json['average'],
      max: json['max'],
      min: json['min'],
    );
  }
}

class PulseDetailData {
  final int currentPulse;
  final DateTime timestamp;
  final PulseStatistics statistics;
  final List<double> history;
```

```
  PulseDetailData({
    required this.currentPulse,
    required this.timestamp,
    required this.statistics,
    required this.history,
  });

  factory PulseDetailData.fromJson(Map<String, dynamic> json) {
    var historyFromJson = json['history'] as List;
    List<double> historyList = historyFromJson.map((i) => (i as
num).toDouble()).toList();

    return PulseDetailData(
      currentPulse: json['currentPulse'],
      timestamp: DateTime.parse(json['timestamp']),
      statistics: PulseStatistics.fromJson(json['statistics']),
      history: historyList,
    );
  }
}

import 'package:flutter/material.dart';
import 'package:fl_chart/fl_chart.dart';
import 'dart:math';

class PulseDetailPage extends StatefulWidget {
  const PulseDetailPage({super.key});

  @override
  State<PulseDetailPage> createState() => _PulseDetailPageState();
}

class _PulseDetailPageState extends State<PulseDetailPage> {
  final List<bool> _isSelected = [true, false, false];

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: const Color(0xFF1C1C2E),
      appBar: AppBar(
        title: const Text('Пульс'),
        backgroundColor: Colors.transparent,
        elevation: 0,
        leading: IconButton(
```

```
        icon: const Icon(Icons.arrow_back_new),
        onPressed: () => Navigator.of(context).pop(),
      ),
    ),
    body: SingleChildScrollView(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.stretch,
        children: [
          Center(
            child: ToggleButtons(
              isSelected: _isSelected,
              onPressed: (int index) {
                setState(() {
                  for (int i = 0; i < _isSelected.length; i++) {
                    _isSelected[i] = i == index;
                  }
                });
              },
              color: Colors.white,
              selectedColor: Colors.black,
              fillColor: Colors.white,
              borderRadius: BorderRadius.circular(8.0),
              borderWidth: 0,
              children: const [
                Padding(padding: EdgeInsets.symmetric(horizontal: 24), child: Text('День')),
                Padding(padding: EdgeInsets.symmetric(horizontal: 24), child: Text('Тиждень')),
                Padding(padding: EdgeInsets.symmetric(horizontal: 24), child: Text('Місяць')),
              ],
            ),
          ),
          const SizedBox(height: 32),
          Column(
            children: const [
              Text.rich(
                TextSpan(
                  children: [
                    TextSpan(
                      text: '68',
                      style: TextStyle(fontSize: 48, fontWeight: FontWeight.bold, color: Colors.white),
                    ),
```

```
                TextSpan(
                  text: ' уд./хв.',
                  style: TextStyle(fontSize: 16, color: Colors.white70),
                ),
              ],
            ),
          ),
          SizedBox(height: 4),
          Text(
            '5 квіт. 2025р. о 9:41',
            style: TextStyle(color: Colors.white70, fontSize: 14),
          ),
        ],
      ),
      const SizedBox(height: 24),
      SizedBox(
        height: 200,
        child: BarChart(
          _mainBarData(),
        ),
      ),
      const SizedBox(height: 24),
      _buildStatisticsCard(),
      const SizedBox(height: 16),
      _buildInfoCard(),
    ],
  ),
 ),
);
}

Widget _buildStatisticsCard() {
 return Card(
  color: const Color(0xFF2A2A4A),
  shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(16)),
  child: Padding(
   padding: const EdgeInsets.all(16.0),
   child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
     const Text('Статистика за день', style: TextStyle(fontSize: 18, fontWeight:
FontWeight.bold, color: Colors.white)),
     const SizedBox(height: 16),
     _buildStatRow('Середній пульс', '68 уд./хв.'),
     const Divider(color: Colors.white24, height: 24),
```

```
          _buildStatRow('Максимальний пульс', '110 уд./хв.'),
          const Divider(color: Colors.white24, height: 24),
          _buildStatRow('Мінімальна частота пульсу', '57 уд./хв.'),
        ],
      ),
    ),
  );
 }

 Widget _buildStatRow(String label, String value) {
  return Row(
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
    children: [
      Text(label, style: const TextStyle(color: Colors.white70, fontSize: 16)),
      Text(value, style: const TextStyle(color: Colors.white, fontSize: 16, fontWeight:
FontWeight.bold)),
    ],
  );
 }

 Widget _buildInfoCard() {
  return Card(
    color: const Color(0xFF2A2A4A),
    shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(16)),
    child: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        children: [
          const Text(
            'Серцебиття означає кількість серцевих скорочень за хвилину.
Нормальне серцебиття в спокійному стані у дорослих становить зазвичай від 60
до 100 ударів на хвилину.\nПрограма відстежує зміни серцебиття за допомогою
такого обладнання, як браслет. Правильна організація тренувань допомагає
зміцнювати серцево-легеневі функції та підтримувати нормальні зміни частоти
серцебиття.',
            style: TextStyle(color: Colors.white70, fontSize: 14, height: 1.5),
          ),
          const SizedBox(height: 20),
          Image.network('https://i.imgur.com/eL402x2.png', height: 150), Placeholder
image
        ],
      ),
    ),
  );
 }
```

```
  BarChartData _mainBarData() {
    final random = Random();
    final barGroups = List.generate(100, (index) {
      return BarChartGroupData(
        x: index,
        barRods: [
          BarChartRodData(
            toY: 50 + random.nextDouble() * 50,
            color: Colors.redAccent,
            width: 2,
            borderRadius: BorderRadius.zero,
          ),
        ],
      );
    });

    return BarChartData(
      barTouchData: BarTouchData(
        touchTooltipData: BarTouchTooltipData(
          tooltipBgColor: Colors.blueGrey,
          getTooltipItem: (group, groupIndex, rod, rodIndex) => null,
        ),
        touchCallback: (FlTouchEvent event, barTouchResponse) {},
      ),
      titlesData: FlTitlesData(
        show: true,
        rightTitles: AxisTitles(sideTitles: SideTitles(showTitles: false)),
        topTitles: AxisTitles(sideTitles: SideTitles(showTitles: false)),
        bottomTitles: AxisTitles(
          sideTitles: SideTitles(
            showTitles: true,
            getTitlesWidget: (value, meta) {
              const style = TextStyle(color: Colors.white70, fontSize: 10);
              String text;
              switch (value.toInt()) {
                case 0: text = '00:00'; break;
                case 25: text = '06:00'; break;
                case 50: text = '12:00'; break;
                case 75: text = '18:00'; break;
                case 99: text = '23:59'; break;
                default: return Container();
              }
              return SideTitleWidget(axisSide: meta.axisSide, space: 4, child: Text(text, style: style));
```

```
          },
          reservedSize: 28,
        ),
      ),
      leftTitles: AxisTitles(
        sideTitles: SideTitles(
          showTitles: true,
          getTitlesWidget: (value, meta) {
            if (value % 28 != 0 && value != 0) return Container();
            return Text(
              '${value.toInt()}',
              style: const TextStyle(color: Colors.white70, fontSize: 10),
              textAlign: TextAlign.left,
            );
          },
          reservedSize: 28,
        ),
      ),
    ),
    borderData: FlBorderData(show: false),
    gridData: FlGridData(
      show: true,
      drawVerticalLine: false,
      horizontalInterval: 28,
      getDrawingHorizontalLine: (value) => const FlLine(color: Colors.white24,
strokeWidth: 0.5),
    ),
    barGroups: barGroups,
  );
 }
}


class PulseStatistics {
 final int average;
 final int max;
 final int min;

 PulseStatistics({required this.average, required this.max, required this.min});

 factory PulseStatistics.fromJson(Map<String, dynamic> json) {
  return PulseStatistics(
   average: json['average'],
   max: json['max'],
   min: json['min'],
```

```
    );
  }
}

class PulseDetailData {
  final int currentPulse;
  final DateTime timestamp;
  final PulseStatistics statistics;
  final List<double> history;

  PulseDetailData({
    required this.currentPulse,
    required this.timestamp,
    required this.statistics,
    required this.history,
  });

  factory PulseDetailData.fromJson(Map<String, dynamic> json) {

    var historyFromJson = json['history'] as List;
    List<double> historyList = historyFromJson.map((i) => (i as
num).toDouble()).toList();

    return PulseDetailData(
      currentPulse: json['currentPulse'],
      timestamp: DateTime.parse(json['timestamp']),
      statistics: PulseStatistics.fromJson(json['statistics']),
      history: historyList,
    );
  }
}

import 'package:your_project_name/models/pulse_data.dart';
import 'dart:convert';
import 'package:http/http.dart' as http;

class ApiService {
  final String _baseUrl = «http://10.0.2.2:3000/api»;

  Future<PulseDetailData> fetchPulseDetails(String userId) async {
    final response = await http.get(Uri.parse('$_baseUrl/pulse/details/$userId'));

    if (response.statusCode == 200) {
      return PulseDetailData.fromJson(jsonDecode(response.body));
    } else {
```

```
      throw Exception('Не вдалося завантажити деталі пульсу');
    }
  }
}

import 'package:flutter/material.dart';
import 'package:fl_chart/fl_chart.dart';
import 'package:intl/intl.dart';

import 'services/api_service.dart';
import 'models/pulse_data.dart';

class PulseDetailPage extends StatefulWidget {
  const PulseDetailPage({super.key});

  @override
  State<PulseDetailPage> createState() => _PulseDetailPageState();
}

class _PulseDetailPageState extends State<PulseDetailPage> {
  final List<bool> _isSelected = [true, false, false];
  final ApiService _apiService = ApiService();
  late Future<PulseDetailData> _pulseDataFuture;

  @override
  void initState() {
    super.initState();

    _pulseDataFuture = _apiService.fetchPulseDetails('user123');
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: const Color(0xFF1C1C2E),
      appBar: AppBar(
        title: const Text('Пульс'),
        backgroundColor: Colors.transparent,
        elevation: 0,
        leading: IconButton(
          icon: const Icon(Icons.arrow_back_new),
          onPressed: () => Navigator.of(context).pop(),
        ),
      ),
      body: FutureBuilder<PulseDetailData>(
```

```
        future: _pulseDataFuture,
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.waiting) {
            return const Center(child: CircularProgressIndicator());
          } else if (snapshot.hasError) {
            return Center(child: Text('Помилка завантаження: <span class=»math-inline»>\{snapshot\.error\}', style\: TextStyle\(color\: Colors\.white\)\)\);
\} else if \(snapshot\.hasData\) \{
final data \= snapshot\.data\!;
final formattedDate \= DateFormat\('d MMM\. yyyyр\. о HH\:mm', 'uk\_UA'\)\.format\(data\.timestamp\);
return SingleChildScrollView\(
padding\: const EdgeInsets\.all\(16\.0\),
child\: Column\(
crossAxisAlignment\: CrossAxisAlignment\.stretch,
children\: \[
Center\(
child\: <134\>ToggleButtons\(
isSelected\: \_isSelected,
onPressed\: \(int index\) \{
setState\(\(\) \{
for \(int i \= 0; i < \_isSelected\.length; i\+\+\) \{
\_isSelected\[i\] \= i \=\= index;</134\>
\}
\}\);
\},
color\: Colors\.white,
selectedColor\: Colors\.black,
fillColor\: Colors\.white,
borderRadius\: BorderRadius\.circular\(8\.0\),
borderWidth\: 0,
children\: const \[
Padding\(padding\: EdgeInsets\.symmetric\(horizontal\: 24\), child\: Text\('День'\)\),
Padding\(padding\: EdgeInsets\.symmetric\(horizontal\: 24\), child\: Text\('Тиждень'\)\),
Padding\(padding\: EdgeInsets\.symmetric\(horizontal\: 24\), child\: Text\('Місяць'\)\),
\],
\),
\),
const SizedBox\(height\: 32\),
Column\(
children\: \[
Text\.rich\(
TextSpan\(
```

```
children\: \[
TextSpan\(
text\: '</span>{data.currentPulse}',
                        style: const TextStyle(fontSize: 48, fontWeight: FontWeight.bold,
color: Colors.white),
                      ),
                      const TextSpan(
                        text: ' уд./хв.',
                        style: TextStyle(fontSize: 16, color: Colors.white70),
                      ),
                    ],
                  ),
                ),
                const SizedBox(height: 4),
                Text(
                  formattedDate,
                  style: const TextStyle(color: Colors.white70, fontSize: 14),
                ),
              ],
            ),
            const SizedBox(height: 24),
            SizedBox(
              height: 200,
              child: BarChart(
                _mainBarData(data.history),
              ),
            ),
            const SizedBox(height: 24),
            _buildStatisticsCard(data.statistics),
            const SizedBox(height: 16),
            _buildInfoCard(),
          ],
        ),
      );
    } else {
      return const Center(child: Text(«Немає даних», style: TextStyle(color:
Colors.white)));
    }
  },
 ),
 );
}

Widget _buildStatisticsCard(PulseStatistics stats) {
 return Card(
```

```
      color: const Color(0xFF2A2A4A),
      shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(16)),
      child: Padding(
       padding: const EdgeInsets.all(16.0),
       child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
         const Text('Статистика за день', style: TextStyle(fontSize: 18, fontWeight:
FontWeight.bold, color: Colors.white)),
         const SizedBox(height: 16),
         _buildStatRow('Середній пульс', '<span class=»math-
inline»>\{stats\.average\} уд\./хв\.'\),
const Divider\(color\: Colors\.white24, height\: 24\),
\_buildStatRow\('Максимальний пульс', '</span>{stats.max} уд./хв.'),
         const Divider(color: Colors.white24, height: 24),
         _buildStatRow('Мінімальна частота пульсу', '${stats.min} уд./хв.'),
        ],
       ),
      ),
     );
    }

    Widget _buildStatRow(String label, String value) {
     return Row(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: [
       Text(label, style: const TextStyle(color: Colors.white70, fontSize: 16)),
       Text(value, style: const TextStyle(color: Colors.white, fontSize: 16, fontWeight:
FontWeight.bold)),
      ],
     );
    }

    Widget _buildInfoCard() {
     return Card(
      color: const Color(0xFF2A2A4A),
      shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(16)),
      child: Padding(
       padding: const EdgeInsets.all(16.0),
       child: Column(
        children: [
         const Text(
          'Серцебиття означає кількість серцевих скорочень за хвилину.
Нормальне серцебиття в спокійному стані у дорослих становить зазвичай від 60
до 100 ударів на хвилину.\nПрограма відстежує зміни серцебиття за допомогою
```

```
такого обладнання, як браслет. Правильна організація тренувань допомагає зміцнювати серцево-легеневі функції та підтримувати нормальні зміни частоти серцебиття.',
              style: TextStyle(color: Colors.white70, fontSize: 14, height: 1.5),
            ),
            const SizedBox(height: 20),
            Image.network('https://i.imgur.com/eL402x2.png', height: 150),
          ],
        ),
      ),
    );
  }
```