НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

“До захисту допущено”
Завідувач кафедри
_____ Наталія АУШЕВА
“___”_____2025 р.

Дипломна робота
на здобуття ступеня бакалавр

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “Мобільний застосунок супроводження пацієнтів під час проведення сеансів гіпокистерапії”

Виконав: студент 4 курсу, групи ТР-14 Філонок Василь Сергійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент каф ЦТЕ, к.т.н, доцент Любов ПОЛЯГУШКО _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Рецензент завідувач каф. ТАЕ, д.т.н., проф Ольга ЧЕРНОУСЕНКО _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Н.контроль асистент каф. ЦТЕ, Володимир РУДИК _____
(посада, ім’я, ПРІЗВИЩЕ) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ — 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ
Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ
Рівень вищої освіти — перший (бакалаврський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА
(підпис)
“___” _____2025 р.

ЗАВДАННЯ
на дипломну роботу студенту

Філонку Василю Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи “Мобільний застосунок супроводження пацієнтів під час проведення сеансів гіпокистерапії”

Науковий керівник Полягушко Любов Григорівна, к.т.н, доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “ 02 ” червня 2025 року № 1875-с

2. Термін подання студентом роботи 09.06.2025р.

3. Вихідні дані до роботи: мова програмування — Dart, Фреймворк — Flutter, СУБД — Firebase, середовище розробки — Android Studio.

4. Перелік питань, які потрібно розробити 1) провести аналіз існуючих мобільних застосунків для моніторингу фізіологічних показників; 2) визначити ключовий функціонал мобільного додатка; 3) обґрунтувати вибір технологій та алгоритмів; 4) спроєктувати архітектуру мобільного застосунку та структуру бази даних; 5) реалізувати прикладний мобільний інтерфейс; 6) продемонструвати сценарії взаємодії лікаря з додатком.

5. Орієнтований перелік ілюстративного матеріалу: діаграма сутностей і зв’язків бази даних; діаграма прецедентів; приклади екранних форм (скріншоти проведення процедур, списку пацієнтів, моніторинг, порівняння показників);

6. Дата видачі завдання 13.09.2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми роботи	19.09.2024 р.	Виконано
2.	Аналіз методів та засобів розв’язання задачі	20.09.2024 – 10.12.2024 р.	Виконано
3.	Розробка архітектури та загальної структури системи	11.12.2024 – 26.02.2025 р.	Виконано
4.	Розробка окремих підсистем	27.02.2025 – 02.04.2025 р.	Виконано
5.	Програмна реалізація системи	03.04.2025 – 28.04.2025 р.	Виконано
6.	Оформлення пояснювальної записки	29.04.2025 – 14.05.2025 р.	Виконано
7.	Захист програмного забезпечення	14.05.2025 – 16.05.2025 р.	Виконано
8.	Передзахист	26.05.2025 – 28.05.2025 р.	Виконано
9.	Захист	15.06.2025 – 20.06.2025 р.	Виконано

Студент _____ __Василь ФІЛОНОК_
(підпис) (прізвище та ініціали)

Керівник _____ Любов ПОЛЯГУШКО
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота виконана на 62 сторінках, містить 16 ілюстрацій, 1 додаток і 10 джерел в переліку посилань.

Метою роботи є створення мобільного додатку який забезпечує лікарю можливість комплексного контролю фізіологічних показників пацієнтів під час сеансів гіпокситерапії у режимах live і тренувальної симуляції.

Апаратною основою застосунку виступає підключення по Bluetooth Low Energy до медичного сенсора та генерація реалістичних даних для тестового режиму, а ключовий функціонал охоплює CRUD-операції з пацієнтами та процедурами, візуалізацію HR, SpO_2 і SYS/DIA за допомогою інтерактивних графіків fl_chart та табличні звіти, груповий аналіз показників, розширений пошук і offline-режим із кешуванням через SharedPreferences. [7]

Інтерфейс реалізовано на Flutter з Material 3, для керування станом застосовано Provider у поєднанні з MVVM, а аутентифікація та зберігання даних покладені на Firebase Authentication і Firestore. [3] Для реалізації UI використано Flutter і Material 3, стан — Provider + MVVM, бекенд — Firebase Authentication та Firestore.

Результатом є надійний інструмент, що забезпечує лікарю зручність тестування, аналізу та порівняння даних у реальному часі.

Ключові слова: мобільний додаток; гіпокситерапія; Flutter; BLE; симуляція; live-моніторинг; Provider; Firebase; fl_chart; offline.

ABSTRACT

The thesis is 62 pages long, contains 16 illustrations, 1 appendix, and 10 sources in the reference list.

Its aim was to develop a mobile application that enables physicians to comprehensively monitor patients' physiological parameters during hypoxia therapy sessions in both live and training-simulation modes.

The application connects via Bluetooth Low Energy to a medical sensor and generates realistic data for testing. Its core functionality covers CRUD operations for patients and procedures, visualization of heart rate, SpO_2, and systolic/diastolic blood pressure using interactive fl_chart graphs and tabular reports, group comparison of metrics, advanced search, and an offline mode with caching via SharedPreferences.

The user interface is implemented in Flutter with Material 3; state management is handled by Provider combined with MVVM; authentication and data storage leverage Firebase Authentication and Cloud Firestore.

The result is a reliable tool that provides physicians with convenient real-time testing, analysis, and comparison of patient data.

Keywords: mobile application; hypoxia therapy; Flutter; BLE; simulation; live monitoring; Provider; Firebase; fl_chart; offline.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

BLE (Bluetooth Low Energy) — енергоефективна версія Bluetooth для підключення медичних сенсорів.MCDA (Multi-Criteria Decision Analysis) — багатокритеріальний аналіз — група методів для прийняття рішень з урахуванням декількох критеріїв.

CRUD — Create (створення), Read (читання), Update (оновлення), Delete (видалення) — базові операції з об’єктами (пацієнти, сеанси).

HR (Heart Rate) — частота серцевих скорочень у bpm.AHP (Analytic Hierarchy Process) — метод аналітичної ієрархії для ранжування критеріїв при невеликій кількості показників.

SpO_2 — рівень насичення крові киснем у відсотках.

SYS / DIA — систолічний та діастолічний артеріальний тиск у мм рт. ст.

MVVM (Model–View–ViewModel) — архітектурний патерн, що відокремлює представлення, дані і логіку.JWT (JSON Web Token) — стандарт авторизації для захищеної передачі інформації між клієнтом і сервером.

Firestore — NoSQL-сховище даних від Firebase для зберігання документів у колекціях. [3]

ВСТУП

У сучасній медицині дедалі зростає значення мобільних технологій для збору та аналізу фізіологічних даних пацієнтів у режимі реального часу. Особливо це стосується методів гіпокситерапії, де контроль частоти серцевих скорочень, насичення крові киснем та артеріального тиску є критично важливим для безпеки й ефективності лікування. З огляду на необхідність інтеграції декількох параметрів, швидкої візуалізації та можливості проведення сеансів, традиційні підходи до моніторингу потребують модернізації та автоматизації.

Метою даної дипломної роботи стало створення мобільного додатку HypoxiTherapy Monitor, який поєднує живий збір даних через Bluetooth Low Energy з тренувальною симуляцією, забезпечує повний цикл CRUD-операцій для пацієнтів і сеансів, а також надає інтерактивні графіки та табличні звіти.

Для реалізації поставленої мети було сформовано наступні завдання:

1) провести аналіз існуючих систем підтримки прийняття рішень і методів багатокритеріального аналізу;

2) визначити ключовий функціонал системи для створення, оцінювання та аналізу альтернативних рішень;

3) обґрунтувати вибір мов, бібліотек, фреймворків і алгоритмів для реалізації;

4) спроєктувати архітектуру програмної системи та структуру бази даних;

5) реалізувати прикладний вебінтерфейс для користувачів із різними ролями;

6) продемонструвати сценарії взаємодії користувачів із системою.

Застосунок розроблено на Flutter із Material 3 для крос-платформного інтерфейсу, Provider + MVVM для логіки управління станом, а аутентифікацію й зберігання даних реалізовано через Firebase Authentication і Firestore. Локальне кешування забезпечує стабільність роботи в offline-режимі. [1]

Реалізований застосунок дає змогу лікарю контролювати фізіологічні показники в режимі реального часу та проводити сеанси, порівнювати групові дані пацієнтів, працювати з записами. Це сприяє підвищенню якості прийняття клінічних рішень, діагностики та покращенню результатів гіпокситерапії.

1 ПОСТАНОВКА ЗАДАЧІ СТВОРЕННЯ МОБІЛЬНОГО ДОДАТКА

Метою цієї роботи є створення мобільного додатку, який надасть лікарю інструменти для повноцінного контролю фізіологічних показників пацієнтів під час сеансів гіпокситерапії. Додаток має забезпечити безпечне підключення до BLE-сенсора, збір та візуалізацію даних у реальному часі, а також механізми пошуку, фільтрації та порівняння показників окремих пацієнтів і груп.

Об’єкт дослідження — процес моніторингу життєвих параметрів пацієнта в ході гіпокситерапії, що вимагає точного й оперативного відображення даних.

1.1 Предмет дослідження

Предметом дослідження є методи та засоби побудови мобільної інформаційної системи, що забезпечує лікарю можливість здійснювати одночасний live-моніторинг пульсу (HR), насичення крові киснем (SpO_2) та артеріального тиску (SYS/DIA). Для досягнення поставленої мети необхідно вирішити такі завдання:

— проаналізувати існуючі підходи до мобільного моніторингу фізіологічних параметрів (HR, SpO_2, тиск), визначити сильні й слабкі сторони окремих рішень. обґрунтувати вибір методів багатокритеріального аналізу, які доцільно застосовувати залежно від специфіки задач;

— визначити ключовий набір функцій мобільного застосунку: безперервний збір даних через BLE-сенсор, тренувальна симуляція, CRUD-операції з пацієнтами та сеансами, груповий аналіз тощо;

— обґрунтувати необхідність інтерактивної візуалізації — плавні графіки fl_chart та таблиці з можливістю сортування й фільтрації;

— розробити архітектуру системи за принципом MVVM + Provider, визначити основні компоненти (ViewModel, сервіси для роботи з BLE, локальне кешування, Firestore);

— реалізувати модуль live-моніторингу із підпискою на потік вимірювань із BLE-сенсора;

— забезпечити механізм offline-підтримки: локальне кешування даних (SharedPreferences, Sqflite) і фонову синхронізацію змін із Firestore через WorkManager із політикою експоненційного back-off;

— розробити інтерфейс лікаря: форми CRUD-операцій (пацієнт, сеанс), екран групового аналізу з накладеними графіками, екрани зі списком пацієнтів та сеансів з пошуком і фільтрацією;

— впровадити безпечну аутентифікацію через Google Sign-In / Firebase Authentication та налаштувати правила доступу Firestore, щоб користувач-лікар бачив лише власні записи;

— протестувати застосунок у ключових сценаріях: авторизація, додавання/редагування пацієнтів, створення сеансів, live-моніторинг, тренувальна симуляція, offline-режим і подальша синхронізація.

Вхідні дані системи:

— профіль пацієнта: ПІБ, дата народження, діагноз, контакти, група. параметри задачі;

— параметри сеансу: дата/час, ідентифікатор пацієнта;

— потік даних від BLE-сенсора: (timestamp, heartRate, spo_2, systolic, diastolic);

— внутрішня інформація: id пристрою, налаштування інтервалів передачі.

Вихідні дані:

— інтерактивні графіки fl_chart із динамікою HR, SpO_2 і SYS/DIA за час сеансу;

— табличні журнали вимірювань із сортуванням і фільтрацією;

— зведені звіти для групового аналізу, порівняння показників кількох пацієнтів;

— локально закешовані дані під час offline-режиму.

Основні функціональні можливості:

— додавання, редагування та видалення профілів пацієнтів;

— створення сеансів гіпокситерапії з автоматичним підключенням до BLE-

сенсора;

— побудова інтерактивних графіків fl_chart для HR, SpO_2 і тиску;

— групове порівняння показників із накладеними графіками та зведеними таблицями;

— безпечна авторизація через Google Sign-In / Firebase Authentication і розмежування доступу через Firestore Security Rules.

Розроблений мобільний застосунок підвищує оперативність і об’єктивність клінічних рішень, забезпечуючи лікарю швидкий доступ до критичних даних та інструментів аналізу.

1.2 Огляд програмних рішень

У цьому підрозділі розглядаються існуючі програмні засоби та пристрої, призначені для моніторингу фізіологічних показників пацієнтів, зокрема в контексті гіпокситерапії. Окрему увагу приділено аналізу мобільних додатків, носимих пристроїв та універсальних рішень, а також визначенню їхніх сильних і слабких сторін щодо необхідного функціоналу.

1.2.1 Pulse Oximeter Apps

Pulse Oximeter Apps — це категорія мобільних додатків, які використовують або камеру смартфона, або зовнішній BLE-сенсор для вимірювання рівня насичення крові киснем (SpO_2) та частоти серцевих скорочень (HR).

Камера смартфона (PPG-метод), приклади: Tap Oximeter (Android/iOS) та Instant SpO_2 (Android/iOS).

Переваги: не потрібен додатковий пристрій, достатньо наявності камери й ліхтарика, безкоштовно, з мінімальними апаратними витратами.

Недоліки: значна похибка (особливо при недостатньому освітленні або тремтінні руки), нестабільність вимірювань: результати можуть “стрибати” від дрібного руху пальця, відсутність підтримки артеріального тиску (SYS/DIA), немає інтегрованих засобів аналізу (лише відображення поточної величини, іноді з простою історією).

1.2.2 Blood Pressure Trackers

Blood Pressure Trackers — мобільні додатки, які призначені для відстеження артеріального тиску.

Ручне введення вимірювань, приклади: Blood Pressure Diary (Android/iOS) та SmartBP (Android/iOS);

Переваги: дозволяють вручну фіксувати SYS/DIA, формувати історію, зручні графіки трендів, з можливістю експорту даних (CSV, PDF).

Недоліки: ручне введення збільшує ризик помилки (переплутати SYS і DIA), не мають автоматичної синхронізації із BLE-тонометрами (цю функцію частіше реалізовано у фірмових додатках окремих виробників), відсутня інтеграція із показниками SpO_2 та HR.

1.2.3 All-in-One Wearables

All-in-One Wearables — носимі пристрої (“розумні годинники” та браслети), які поєднують датчики пульсу, SpO_2, а іноді й вимір тиску.

Приклади смарт-годинників:

— Apple Watch Series 7 (тонометр, SpO_2, ЕКГ);

— Samsung Galaxy Watch4 (датчик SpO_2, HR, оцінка артеріального тиску шляхом калібрування);

— Fitbit Sense (датчик SpO_2, HR, оцінка стресу);

— Garmin Venu 2 (SpO_2, HR, моніторинг стресу).

Переваги: висока точність PPG-датчиків (HR, SpO_2) завдяки вдосконаленим алгоритмам, одночасний комплексний моніторинг без додаткових пристроїв, інтеграція з екосистемою виробника (Apple Health, Fitbit Web, Garmin Connect) із можливістю аналізу трендів.

Недоліки: закритий формат даних: для інтеграції необхідні офіційні API (часто платні чи обмежені), залежність від батареї: пристрій потребує регулярного підзаряджання (1–2 рази на добу), що може порушити безперервний моніторинг, висока вартість (250–400 USD і вище), що робить такі пристрої неуніверсальними для всіх категорій пацієнтів.

1.2.4 Критичний аналіз недоліків існуючих систем

Проведений огляд показав, що на ринку немає єдиного мобільного рішення, яке б поєднувало: одночасний live-моніторинг HR, SpO_2, SYS, DIA через BLE-сенсор.), інтерактивну візуалізацію даних (підказки на графіку, адаптивне масштабування, сортування у таблицях), тренувальну симуляцію даних для розробки та навчання медичного персоналу, комплексний груповий аналіз — порівняння результатів кількох пацієнтів на одному полотні.

Основні недоліки наявних рішень:

— обмежений набір показників — переважно SpO_2 + HR або лише тиск, але не всі чотири параметри одночасно;

— слабка інтеграція та закриті екосистеми — фірмові додатки часто не надають відкритого API для інтеграції, що унеможливлює формування єдиного інтерфейсу;

— нестача інструментів групового аналізу — навіть просунуті носимі пристрої не забезпечують можливості накладати дані кількох пацієнтів, порівнювати їхні тренди в одному віджеті;

— високі витрати — професійні носимі пристрої та їхні хмарні послуги є недоступними для бюджетних закладів охорони здоров’я.

1.2.3 Особливості даних у власній системі

Щоб успішно реалізувати завдання, потрібно чітко визначити структуру й обсяг даних, які додаток прийматиме, оброблятиме та відображатиме:

Формат одиничного вимірювання (Measurement), що містить: частоту надходження: ≥ 1 запис/сек (можливо до 3–5 Hz за потреби) та динамічні налаштування, що дають змогу відігравати різні клінічні сценарії (від стабільного до екстремального стану пацієнта). Обсяги даних за сеанс:

— при стандартному 10-хвилинному сеансі зі швидкістю 1 Hz накопичується 600 точок × 4 полів ≈ 2400 полів (у середньому ~60 байт/JSON) → ~36 кБ.);

— у прискореному режимі (5 Hz) тих самих 10 хв це вже 3000 точок × 60 байт ≈ 180 кБ;

— важливо, щоб UI (графік fl_chart) відображав ці 600–3000 точок без суттєвого просідання FPS (< 60 FPS), на що розраховано оптимізації у fl_chart та Flutter.

Offline-підтримка та синхронізація. Використовується SharedPreferences для збереження простих ключ-значення:

— lastSyncTime (час останньої успішної синхронізації);

— cachedPatientsList (JSON-рядок із переліком пацієнтів);

— cachedProceduresList (JSON-рядок із переліком процедур);

— offlineFlag (bool) — чи зараз застосунок перебуває в offline.

1.3 Програмні засоби

Розробка мобільного застосунку вимагає ретельного підходу до вибору технологічного стеку, що забезпечує продуктивність, надійність, безпеку та можливість подальшого розширення. У цьому підрозділі обґрунтовано вибір мови

програмування, фреймворку UI, інструментів для управління станом, роботи з BLE-сервісами, візуалізації графіків, локального кешування та бази даних.

1.3.1 Вибір мови програмування

Dart забезпечує компіляцію у нативний машинний код, що дозволяє отримати швидке завантаження та мінімальні затримки під час запуску додатка. Для клінічних ситуацій та екстреного моніторингу це критично: кожна секунда очікування може впливати на своєчасність реагування.

Асинхронна модель (Future/Stream): обробка live-даних від BLE-сенсора здійснюється як потік (Stream<Measurement>), синтаксис async/await, а також можливості StreamController і StreamSubscription у Dart дозволяють організувати чисту, легко читану обробку нових вимірювань. [2]

Об’єктно-орієнтована типізація: кожна сутність (Patient, Procedure, Measurement, Group) представлена класом із чіткими полями й методами toJson(), fromJson(), жорстке типізування знижує ризик помилок у рантаймі та підвищує надійність коду.

Екосистема pub.dev: велика кількість перевірених пакетів для роботи з BLE (flutter_reactive_ble), графіками (fl_chart), управлінням станом (provider), базою даних (firebase_core, cloud_firestore), локальним сховищем (shared_preferences, sqflite), фоновими задачами (workmanager), регулярні оновлення, активна спільнота та велика база прикладів спрощують розробку й підтримку. [5]

1.3.2 Вибір фреймворку

Flutter (версія 3.7.0) обрано як платформу для розробки UI за такими аргументами.

Кросплатформеність: дозволяє створювати застосунок єдиним кодом для Android (8.0+) та iOS (14.0+), гарантуючи однаковий вигляд і поведінку на обох ОС, зменшує витрати на підтримку окремих репозиторіїв для кожної платформи.

Вбудований рушій Skia: Flutter рендерить весь інтерфейс через Skia, що задовільняє високі вимоги до швидкості та передбачуваності UI, інтерактивні елементи, анімації та плавне оновлення графіків fl_chart забезпечують ≥ 60 FPS під час відображення до 3000 точок.

Hot Reload / Hot Restart: під час налагодження та дизайну інтерфейсу можна вносити зміни в код і відразу бачити їх у запущеному емуляторі чи фізичному пристрої без повного перезапуску, допомагає швидко тестувати різні варіанти віджетів (графіків, списків, форм).

Material 3: Flutter має готові ThemeData й віджети, що відповідають рекомендаціям Material 3: використання ColorScheme.fromSeed, анімації переходів, адаптивний розмір елементів залежно від налаштувань ОС.), підтримка Semantics для доступності: скринрідери (TalkBack, VoiceOver) коректно озвучуватимуть кнопки, поля та підказки.

Гнучкість кастомізації UI: можливість власноруч реалізувати високопродуктивні віджети (наприклад, MeasurementChartWidget із CustomPainter), які будуть однаково працювати на будь-яких пристроях, зручні механізми створення адаптивного дизайну: LayoutBuilder, MediaQuery та власні breakpoints.

1.3.3 Вибір інструментів для клієнтської частини

Provider (пакет provider: ^6.0.5): забезпечує реєстрацію ViewModel–ів (ChangeNotifier) у корені додатка, за зміни даних у ViewModel достатньо викликати notifyListeners(), щоб віджети з Consumer<T>() або context.watch<T>() автоматично перерисувалися, забезпечує тестованість бізнес-логіки окремо від інтерфейсу.

fl_chart (пакет fl_chart: ^0.65.1), використовується для побудови лінійних графіків із одночасним відображенням до чотирьох серій даних (HR, SpO_2, SYS, DIA), використовуються LineChartData параметри.

shared_preferences (пакет shared_preferences: ^2.0.17), зберігає прості налаштування, використовується для швидкого відновлення даних під час запуску

застосунку.

sqflite: для зберігання об’ємних локальних структур (зміни пацієнтів, процедур, вимірювань) у вигляді окремої таблиці changesQueue(id TEXT PRIMARY KEY, entityType TEXT, jsonData TEXT, status TEXT), запис кожного нового/змінного/видаленого об’єкта відразу додається в цю таблицю зі статусом pending, після успішної синхронізації в WorkManager статус змінюється або запис видаляється.

1.3.4 Вибір системи управління базами даних

У якості бекенда для зберігання медичних та користувацьких даних було обрано Cloud Firestore (частина платформи Firebase) через наступні переваги:

Гнучка NoSQL-модель: структура у вигляді документів і колекцій відповідає логіці предметної області (лікарі, пацієнти, процедури, групи, вимірювання), можливість додавати нові поля без складних міграцій бази, автоматичний offline-перший підхід.

Firestore кешує останні дані локально, дозволяючи продовжувати роботу без Інтернету, після відновлення підключення зміни автоматично синхронізуються.

Розширена система індексів: procedures за patientId і dateTime для отримання хронології процедур, patients за doctorId і groupId для фільтрації пацієнтів певного лікаря у вибраній групі.

2 АНАЛІЗ СУЧАСНИХ ПРОГРАМНИХ РІШЕНЬ ТА МЕТОДІВ

Забезпечення надійного моніторингу фізіологічних показників пацієнтів під час гіпокситерапії є ключовим завданням клінічної практики, оскільки рішення про коригування лікування повинні базуватися на швидких і точних даних. Існуючі мобільні рішення часто обмежені нестабільним BLE-з'єднанням, статичними графіками та відсутністю можливості порівняння показників між пацієнтами.

У цьому розділі проведено всебічний огляд існуючих рішень для відстеження ключових фізіологічних показників: частоти серцевих скорочень, насичення крові киснем і артеріального тиску. Розглянуто як нативні мобільні додатки для пульсоксиметрії та ручного введення тиску, так і рішення для носимих пристроїв із live-стрімінгом даних.

2.1 Характеристика гіпоксикаторів та особливості даних

У цьому підрозділі описано класифікацію гіпоксикаторів (стаціонарні, портативні, побутові), наведено технічні характеристики BLE-сенсорів, які застосовуються для зчитування HR, SpO_2 та артеріального тиску, а також розглянуто типовий формат передавання даних від сенсорів до мобільного додатку. Уся ця інформація необхідна для розуміння обмежень, пов'язаних із стабільністю зв'язку, частотою дискретизації та структурою пакету даних, що напряму впливають на точність і надійність побудови реального часу моніторингу.

2.1.1 Класифікація гіпоксикаторів

Гіпоксикатори — це медичні пристрої, які забезпечують пацієнта газовою сумішшю з пониженою концентрацією кисню задля терапевтичного впливу.

Залежно від сценарію застосування та умов використання їх поділяють на три основні категорії:

Стаціонарні гіпоксикатори використовуються переважно у стаціонарних умовах (лікарняні відділення, клініки, спеціалізовані центри), де є постійне джерело електроенергії, окреме місце для розміщення обладнання та кваліфікований медперсонал.

Як правило, такі апарати мають високу продуктивність, здатні подавати регульовану концентрацію кисню від 10 % до 95 %, обладнані системами регулювання потоку газу, системами альтернати-вного енергопостачання (UPS) і розширеним набором налаштувань.

Для медичних стаціонарних гіпоксикаторів рідко використовують безконтактні BLE-сенсори; дані про пацієнтів передаються локально через Ethernet або Wi-Fi у закриті медичні мережі. Інтеграція з мобільним застосунком у такому разі може здійснюватися через API медичної інформаційної системи, але це виходить за межі створюваного мобільного рішення.

Портативні гіпоксикатори призначені для виїзних бригад, польових умов (гірські вершини, спортсмени на висоті), санаторно-курортних закладів. Мають компактніший корпус, вбудовані акумулятори або легкі генератори, що дозволяє працювати автономно до кількох годин.

Часто забезпечують діапазон концентрацій кисню 15 %–45 %, мають обмежений потік газу (0,5–3 л/хв), оснащені механічними чи електронними регуляторами.

У портативних пристроях для отримання фізіологічних показників пацієнта широко застосовують зовнішні BLE-сенсори (наприклад, пульсоксиметри), які передають дані безпосередньо в мобільний застосунок через бездротовий інтерфейс Bluetooth Low Energy. Частота дискретизації зазвичай становить 1–5 Hz. Структура переданих пакетів може варіюватися залежно від виробника сенсора, проте здебільшого включає: часову мітку, значення HR, SpO_2, SYS і DIA.

Побутові гіпоксикатори використовуються вдома пацієнтами за призначенням лікаря або у спортклубах для тренувань дихальних вправ. У

порівнянні з портативними, зазвичай дешевші, простіші за конструкцією, не мають складних систем контролю, але забезпечують базовий рівень зниження концентрації кисню.

Найчастіше обладнані простими регуляторами без можливості тонкої настройки, мають обмежений ресурс генерації кисню (до 2–3 л/хв), відсутні складні системи фільтрації.

Побутові пристрої зазвичай не мають прямої електронної інтеграції з медичним програмним забезпеченням; фізіологічні параметри пацієнта збираються через зовнішні носимі датчики (розумні браслети, пульсоксиметри) або моніторяться вручну (затиск пальця в стандартному пульсоксиметрі). Для мобільного застосунку наявність побутового гіпоксикатора означає, що дані про HR/SpO_2/тиск надходитимуть від незалежного BLE-сенсора, а не від самого апарата.

2.1.2 Технічні характеристики BLE-сенсорів

Розглянуто основні параметри BLE-сенсорів, що дозволяють отримувати фізіологічні дані в реальному часі: частоту дискретизації, затримки передачі, точність вимірювань, енергоспоживання тощо. Знання цих параметрів дає змогу врахувати можливі обмеження у реалізації live-моніторингу й вибудувати коректну архітектуру обміну даними.

Частота дискретизації для більшості комерційних пульсоксиметрів і HR-сенсорів знаходиться в діапазоні 1–5 Hz, тобто датчик надсилає нове вимірювання від 1 до 5 разів на секунду.

Точність і похибка вимірювань HR (Heart Rate):

— типова похибка становить ±1–2 bpm (удари за хвилину) за сприятливих умов (стабільне положення пальця, достатнє освітлення, мінімальний рух);

— у випадку рухів або слабкого кровотоку похибка може збільшуватись до ±5 bpm.

Точність і похибка вимірювань SpO_2 (насичення крові киснем):

— Зазвичай сенсори забезпечують точність ±2 % у діапазоні від 70 % до 100 %;

— нижче 70 % похибка може досягати ±3–4 %;

— результати можуть спотворюватися через холодні кінцівки, набряки чи рухи пальця.

Точність і похибка вимірювань артеріального тиску (SYS/DIA):

— для BLE-тонометрів похибка вимірювання зазвичай становить ±3 мм рт. ст. у межах нормальних значень (гортання 90–140 для систоли та 60–90 для діастоли);

— не всі портативні BLE-сенсори підтримують повний цикл автоматичного вимірювання тиску; деякі зчитують тільки показники, налаштовані попередньо через калібрування користувача (через манжет чи проміжне програмне забезпечення).

Затримки передачі (latency): типова латентність з моменту зчитування фізіологічного параметра до появи його в мобільному застосунку — 100–300 мс., у разі втрати пакету або розриву зв’язку затримка може зрости до 1–2 с у процесі перепідключення.

Час ініціалізації з’єднання та перепідключення: сканування доступних BLE-пристроїв займає приблизно 2–3 с., підключення до сенсора та встановлення зв’язку (GATT Discovery) може тривати від 1 до 5 с залежно від моделі пристрою й умов середовища, автоматичні перепідключення після втрати зв’язку відбуваються протягом 2–4 с за умови стабільного інтерфейсу Bluetooth.

Протоколи шифрування та захисту даних: BLE-сенсори можуть підтримувати шифрування зв’язку (LE Secure Connections), що гарантує захист даних про фізіологічний стан пацієнта під час передачі, для медичних застосунків рекомендується використовувати захищений режим із підтвердженням (Numeric Comparison), щоб забезпечити цілісність і конфіденційність даних пацієнта.

2.1.3 Типові формати даних

Визначено структурні особливості пакету даних, що надсилається BLE-сенсором, і описано рекомендовані способи організації інформації в мобільному застосунку. Говориться про поля даних, їх розміри, допустимі діапазони, а також синтаксис ізбереження у вигляді JSON-об’єкту для подальшої обробки.

Частота дискретизації та інтервал передачі: сенсор надсилає пакет даних через BLE приблизно раз на одну секунду (1 Hz) за звичайного режиму, у високочастотному режимі (3–5 Hz) сенсор може групувати кілька вимірювань у один BLE-пакет або надсилати їх окремими пакетами з інтервалом від 200 до 333 мс. Між рядковими (text-based) або бінарними пакетами BLE, зазвичай, використовується власний протокол виробника.

Обмін даними між ViewModel та UI: ViewModel, отримавши Measurement (перетворений із бінарного пакета або JSON-рядка), зберігає об’єкт у своєму внутрішньому списку. Після кожного додавання Measurement викликається notifyListeners(), і відповідні віджети (графік, таблиця) автоматично оновлюються. Це забезпечує плавне інкрементальне додавання нових точок на графік і оновлення табличного представлення в реальному часі.

2.2 Формат та структура даних гіпокситерапії

Цей підрозділ глибше розглядає кожне поле Measurement (timestamp, heartRate, spo_2, systolic, diastolic), описує допустимі діапазони значень, їх розподіл та вплив зовнішніх факторів, а також особливості побудови симульованого потоку даних у режимі Simulation Mode. Завдання — обґрунтувати, як саме дані повинні зберігатися, оброблятися та візуалізуватися у застосунку, щоб забезпечити клінічно коректні результати та адекватну реакцію UI.

2.2.1 Поле “timestamp”

Інтервали дискретизації: за стандартного режиму дискретизації (1 Hz) кожний наступний timestamp відрізняється приблизно на 1000 мс. У високочастотному режимі (3–5 Hz) інтервал скорочується до 200–333 мс. Для симульованого режиму застосунок генерує timestamp послідовно з фіксованим кроком (1000 мс або менше за налаштуванням), щоб імітувати реальний потік даних.

Усунення розривів і корекція часових міток: якщо під час сеансу було втрачено 1–2 пакети (наприклад, через нестабільний BLE-зв’язок), у локальному списку Measurement генерується фіктивний об’єкт з полем isComplete: false і з timestamp, що відповідає пропущеному інтервалу. Для відображення “пропусків” на графіку UI може позначати ділянку зі зменшеним opacity або пунктиром.

Усунення розривів і корекція часових міток: для відображення “пропусків” на графіку UI може позначати ділянку зі зменшеним opacity або пунктиром.

2.2.2 Поле “heartRate”

Одиниці та одиничні показники: heartRate (HR) вимірюється в одиницях ударів за хвилину (bpm). У медичній літературі нормальні значення HR у стані спокою для дорослої людини перебувають у діапазоні 60–100 bpm. Під час фізичного навантаження чи стресу HR може зростати до 120–160 bpm, а в екстремальних умовах (спортивні тренування, інтенсивні гіпоксичні сеанси) може тимчасово підніматися до 180–200 bpm.

Діапазон допустимих значень: нижня межа: 30 bpm (у разі дуже глибокого сну або патологій). Верхня межа: 220 bpm (розрахунок максимального HR як 220 – вік пацієнта, зазвичай у рамках спортивної медицини). У застосунку налаштовано, що значення < 40 bpm або > 200 bpm розглядаються як потенційно артефактні й вимагають додаткової валідації.

Вплив зовнішніх факторів: рухи пацієнта (якщо під час гіпокситерапії пацієнт рухає рукою або пальцем, датчик може зафіксувати неправильні значення HR або пропуски), погане прилягання сенсора (за слабкого прилягання пульсоксиметра (надто широкий палець або рух перепон), значення HR може “плавати”), температура та кровообіг (у холодних умовах кровообіг у кінцівках погіршується, PPG-датчик може давати некоректні сигнали, що впливають і на HR).

2.2.3 Поле “spo2”

Одиниці та базова інтерпретація: spo2 — це відсоток насичення артеріальної крові киснем (%).

Нормальні значення SpO_2 для здорової людини при sea level (≈ 0 м над рівнем моря) — 95–100 %. У межах 90–94 % вважаються нижньою межею нормального стану; < 90 % потребує медичного втручання.

Похибка та точність:комерційні пульсоксиметри забезпечують точність SpO_2 ≈ ±2 % за діапазону 70–100 %. Нижче 70 % похибка може зростати до ±3–4 %.

Фільтрація та контроль даних: застосовується exponential moving average (EMA) для згладжування короткочасних флуктуацій. Якщо протягом 3 с похибка перевищує ±4 % або зчитування видає значення нижче 80 % без подальшого підвищення, застосунок виводить попередження “Низьке SpO_2: перевірте стан пацієнта”.

У Simulation Mode алгоритм моделювання SpO_2 передбачає дисперсію σ = 2 %, але за потреби він може “імітувати” епізоди гіпоксемії (SpO_2 < 85 %) для демонстрації відмови та корекції лікувальних параметрів.

2.2.4 Поля “systolic” та “diastolic”

Формат і одиниці: systolic — систолічний артеріальний тиск (мм рт. ст.),

відображає максимальний тиск у судинах під час скорочення серця, а diastolic — діастолічний артеріальний тиск (мм рт. ст.), показує мінімальний тиск у судинах під час розслаблення серця.

Допустимі значення: для дорослої людини у спокійному стані нормальний SYS = 90–120 мм рт. ст., DIA = 60–80 мм рт. ст. У мобільному застосунку в разі отримання SYS < 70 або DIA < 40 позначають як “патологічне низьке значення”, а SYS > 200 або DIA > 120 — як “екстремальне підвищення”, що вимагає негайного попередження користувача.

Взаємозалежність показників: зазвичай між SYS і DIA існує певна кореляція (SYS – DIA ≈ 40–60 мм рт. ст.). Якщо різниця (пульсовий тиск) різко змінюється (наприклад, SYS = 150, DIA = 70 vs. SYS = 150, DIA = 90), це може свідчити про зміну судинної резистентності чи напруги в організмі.

Моделі фільтрації враховують розбіжності: якщо пульсовий тиск > 60 мм рт. ст., застосунок підсвічує зміни та пропонує лікарю перевірити справність манжети чи сенсора.

Алгоритм валідації: послідовна перевірка: якщо в межах одного сеансу за п’ять підряд вимірювань SYS або DIA відхиляється від попереднього на > 20 мм, ці значення вважаються потенційними артефактами. Перевірка співвідношення: якщо пульсовий тиск (SYS – DIA) раптово збільшується або зменшується на > 30 мм порівняно з середнім за попередні 10 с, такі дані маркуються як “сумнівні”.

2.2.5 Особливості даних у Simulation Mode

Частота та синхронізація: Simulation Mode ґрунтується на тих самих частотах, що й BLE-режим (1 Hz або 3–5 Hz). Кожен ідентифікатор точки генерується з інтервалом, відповідним поточному налаштуванню — наприклад, крок 1000 мс або крок 250 мс.

За необхідності можна налаштовувати частоту симуляції окремо для кожного показника, щоб перевірити стрес-тести інтерфейсу при високій щільності даних.

2.3 Методи обробки та фільтрації медичних даних

У цьому підрозділі наведено алгоритми, які забезпечують попередню обробку сирих вимірювань (валідація, інтерполяція пропусків), згладжування шуму (медіанні та ковзні фільтри), виявлення артефактів (раптові стрибки, втрати пакетів) і обчислення статистичних показників (середні, пікові значення). Ці методи гарантують, що дані, передані до UI, будуть максимально коректними та інформативними.

2.3.1 Базова обробка сирих даних

Валідація полів Measurement: якщо значення виходить за межі або є нульовим/нульовоподібним (наприклад, HR = 0), Measurement вважається потенційно артефактним. Такий об’єкт позначається полем isValid=false і може не передаватися безпосередньо в UI, але зберігається у локальному кеші для подальшого аналізу.

Інтерполяція пропусків: якщо під час сеансу фіксується відсутність нових вимірювань протягом часу, який перевищує очікуваний інтервал на 50 % (наприклад, за частоти 1 Hz пропуск > 1500 мс), застосунок організовує інтерполяцію лінійного або зворотно-зворотно-пропорційного (linear interpolation) типу. Це дає змогу уникнути “перерв” на графіку і зберегти плавність кривих. Інтерпольовані точки маркуються полем isInterpolated=true, щоб на графіку відобразити їх з дещо меншою інтенсивністю кольору.

Схема збереження у ViewModel: після попередньої валідації і, за потреби, інтерполяції Measurement додається до колекції measurements у відповідному ViewModel, якщо Measurement визнано некоректним (isValid=false) та не інтерпольованим, він додається до окремого списку invalidMeasurements, а в UI відображається як “пропуск” або “артефакт” із позначкою жовтим трикутником.

2.3.2 Згладжування та фільтрація шуму

Медіанний фільтр використовується для усунення короткочасних стрибків у сигналі, які можуть з’являтися, наприклад, через рух пальця або випадкові артефакти сенсора. Для цього обирають розмір «вікна» 3 або 5 вимірювань.

Алгоритм: кожного разу, коли надходить нове вимірювання M_t, розглядають разом із ним два попередні вимірювання M_{t-1} і M_{t-2}. Для кожного параметра (HR, SpO_2, SYS, DIA) беруть три останні значення (відповідно з вимірювань у моментах t, t−1 і t−2), впорядковують їх за зростанням і вибирають середнє (медіану). Обрані медіани замінюють у масиві filteredMeasurements[t] оригінальні «гучні» значення.

Результат замінює поточне значення Measurement у робочому масиві filteredMeasurements.

Ковзний середній фільтр (Moving Average, SMA): для більш “гладких” кривих застосовується ковзний середній із вікном 5–10 точок, що дозволяє вловлювати повільні тренди даних. Кожне нове smoothedValue (1) обчислюється як середнє арифметичне останніх k вимірювань:

$$smoothedValue_t = \frac{1}{k}\sum_{i=0}^{k-1} x_{t-i}, \quad (1)$$

де $smoothedValue_t$ згладжене (ковзне середнє) значення параметра в момент t;

k — кількість точок, які входять у вікно фільтра (наприклад, 5 або 10);

x_{t-i} — значення того самого параметра (HR, SpO_2, SYS або DIA) у попередній момент часу;

якщо i = 0, то $x_{t-0} = x_t$ — це актуальне (найсвіжіше) вимірювання;

якщо i = 1, то x_{t-1} — це значення, отримане одне вимірювання тому, і так далі.

Контроль затримки (lag) фільтра: при застосуванні медіанного й SMA фільтрів виникає затримка, визначена половиною розміру вікна. У мобільному застосунку ця затримка компенсується через паралельну відмальовку.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

У цьому розділі представлено детальний опис реалізації мобільного застосунку: від загальної архітектури системи до алгоритмів обробки даних і взаємодії з користувацьким інтерфейсом. Метою розділу є надати читачеві повне технічне уявлення про структуру, компоненти, схеми взаємодії й основні алгоритми, що лежать в основі розробленого програмного забезпечення.

3.1 Архітектура програмної системи

У якості фундаменту для організації коду застосовано патерн MVVM (Model–View–ViewModel), який чітко розділяє відображення, бізнес-логіку та доступ до даних [6].

На найвищому рівні розташована презентаційна частина (UI), створена на Flutter. Вона складається з екранних компонентів—від екрана авторизації до Live-моніторингу та перегляду історії сеансів. Прямо під UI розгорнутий шар ViewModel, у якому кожна логічна одиниця (пацієнти, сеанси, групи) має власний клас ViewModel. Ці класи збирають дані з відповідних сервісів та транслюють їх у видимий стан, який відображає інтерфейс.

Найнижчий рівень — це сервісний шар, до якого входять два ключові компоненти: BLEService і SyncService. Перший відповідає за роботу з Bluetooth Low Energy, тобто сканування, підключення до медичного сенсора та обробку потоку вимірювань. Другий забезпечує збереження будь–яких змін локально, формуючи чергу операцій, а потім у фоновому режимі відправляє їх у Cloud Firestore, обробляючи можливі конфлікти та повторні спроби.

Зрештою, шар даних забезпечується двома сховищами: локальною базою Sqflite (для offline–режиму та кешування) та хмарою Firestore (для централізованого зберігання й синхронізації).

Взаємодія між компонентами відбувається таким чином: коли користувач у UI ініціює дію (наприклад, створення пацієнта чи запуск Live-моніторингу), методи інтерфейсу викликають відповідний ViewModel. ViewModel обробляє бізнес-логіку—наприклад, формує об’єкт Patient, передає його до SyncService для локального збереження та постановки в чергу, а також оновлює стан, який повертається у UI через механізм прослуховування (Provider/ChangeNotifier).

Якщо ж йдеться про Live-моніторинг, ViewModel звертається до BLEService, який спочатку виконує Bluetooth-сканування, потім ініціалізує підключення до сенсора, підписується на потік характеристик і поступово передає оброблені вимірювання назад у ViewModel. Після кожного нового Measurement ViewModel у свою чергу повідомляє UI про оновлення, і клавіатура графіка динамічно доповнюється новими точками.

Фоновий процес синхронізації запускається приблизно кожні 15–20 хвилин за допомогою WorkManager. У цей момент SyncService проходить через всі накопичені операції (створення чи оновлення пацієнтів, сеансів, груп або вимірювань) і робить відповідні запити до Firestore.

Якщо мережеве з’єднання недоступне, SyncService зберігає зміни у локальній черзі до перших вигідних умов і лише потім відправляє їх у хмару.

Якщо під час передачі виникає конфлікт (наприклад, версії документа не співпадають), Service завантажує актуальну копію з Firestore, виконує злиття локальних змін і повторно надсилає запит. Таким чином, усі рівні архітектури працюють у тісному зв’язку, гарантуючи відмовостійкість, можливість роботи в offline-режимі та надійну обробку Bluetooth-даних.

3.2 Модель представлення даних

Для збереження інформації про пацієнтів, сеанси та фізіологічні показники програма використовує об’єктно-орієнтовану модель, де кожна сутність описана окремим класом із методами серіалізації в JSON (рис. 3.1).

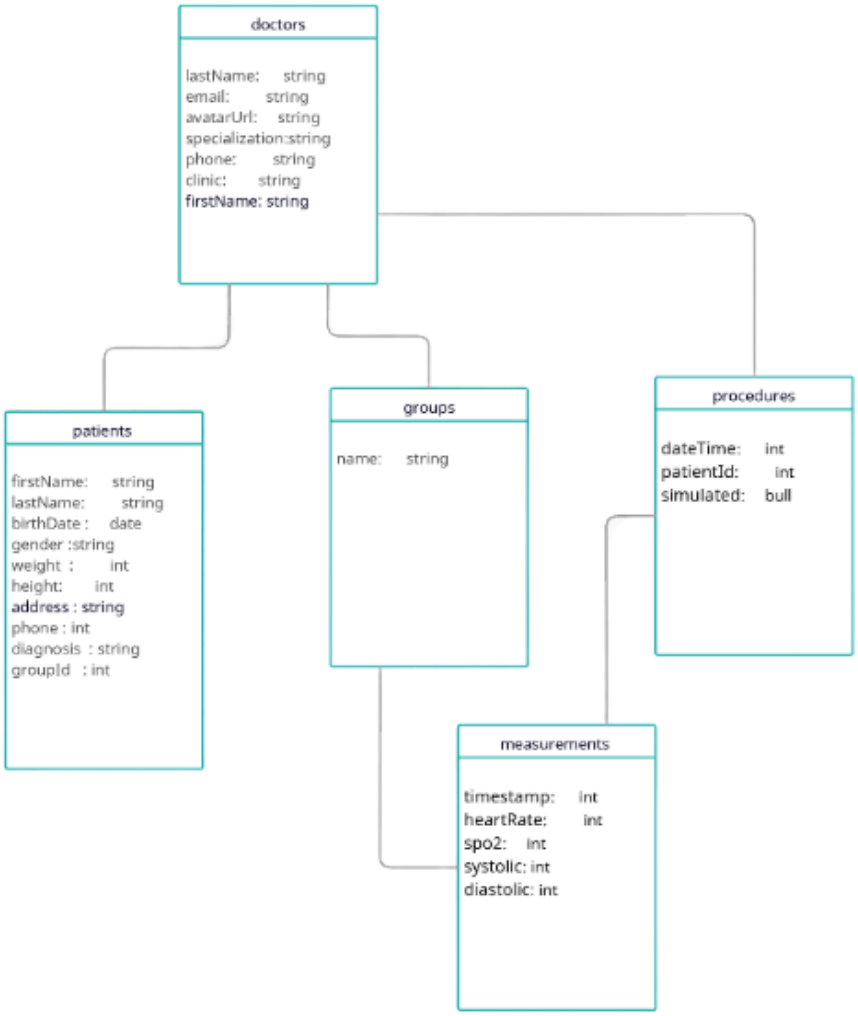

Рисунок 3.1 — Діаграма бази даних

Клас Patient містить унікальний ідентифікатор, повне ім’я, дату народження (що дозволяє розрахувати вік), короткий діагноз та ідентифікатор групи (якщо пацієнт належить до групи порівняння).

Клас Procedure представляє один сеанс гіпокситерапії: до нього входять посилання на пацієнта, час початку процедури, прапорець “симульований чи реальний” і список об’єктів Measurement.

Клас Measurement фіксує момент зчитування й значення таких параметрів, як частота серцевих скорочень, насичення крові киснем та систолічний/діастолічний тиск, а також відмічає час створення запису.

У локальній базі Sqflite кожна з цих сутностей зберігається у власній таблиці з полями, які відповідають властивостям об’єкта: наприклад, таблиця patients має стовпці для id, fullName, birthDate, diagnosis, groupId, createdAt та updatedAt.

Аналогічно, у таблиці procedures зберігаються id, patientId, dateTime, isSimulated, createdAt, updatedAt, а в таблиці measurements -фіксуються timestamp, heartRate, spo2, systolic, diastolic та createdAt.

Окремою таблицею ведеться черга pending_operations, у якій зберігаються всі CRUD-операції над сутностями у вигляді серіалізованих JSON-об’єктів разом з позначенням типу операції (create, update, delete) та міткою часу. [8] Ця таблиця гарантує безпеку змін у випадку відсутності мережі: усі локальні операції чекають у черзі на свою чергу для відправки у Firestore.

У Cloud Firestore застосовано модель “документ–колекція”. Кожний лікар (ідентифікатор якого походить з Firebase Authentication) має власну колекцію patients — у ній документи з полями, які відповідають властивостям класу Patient.

Аналогічно, колекція procedures містить документи-сеанси, а підколекція measurements для кожного документа procedure зберігає окремі записи Measurement. Колекція groups утримує документи, які містять назву групи та масив ідентифікаторів пацієнтів. Firestore автоматично індексує поля patientId та dateTime у колекції procedures, щоб можна було швидко отримувати хронологічний список сеансів для конкретного пацієнта.

3.3 Діаграма прецедентів

У діаграмі прецедентів (Use Case) визначено основні ролі та сценарії взаємодії (рис. 3.2). Головним актором є лікар, який може: авторизуватись через Google Sign-In, керувати пацієнтами (створення, редагування, видалення), ініціацювати сеанс гіпокситерапії у Live і у Simulation режимі, бачити історію вимірювань, формувати групи пацієнтів та виконувати порівняльний аналіз показників.

Рисунок 3.2 — Use Case діаграма

Другим актором є сам BLE-сенсор, який у процесі Live-сеансу безперервно надсилає дані про ЧСС, SpO_2, SYS і DIA. Актор “Система” утілює функціональність контексту Firestore й AuthService, обробляючи запити на збереження/отримання даних та перевірку прав користувача.

3.4 Діаграма класів

Діаграма класів відображає структуру основних об’єктів і зв’язків у застосунку. Клас Patient містить властивості, що описують пацієнта, та методи серіалізації/десеріалізації. Клас Procedure зберігає інформацію про сеанс: зв’язок із пацієнтом за допомогою поля patientId, дату й час, сортування за прапорцем

“isSimulated”, а також включає колекцію measurements для конкретного сеансу. Клас Measurement фіксує одне значення фізіологічних параметрів. Додатково існує клас Group — він містить назву групи та список ідентифікаторів пацієнтів для подальшого порівняльного аналізу (рис. 3.3).

Клас BLEService окреслює атрибути й методи, необхідні для роботи з Bluetooth: поля для підписки на потік характеристик, обробники подій підключення/відключення, а також логіку повторного підключення за таймаутом. Клас SyncService відповідає за чергу операцій: його атрибути включають інстанси локальної бази й Firestore, а методи—за роботу з таблицею pending_operations, виконання циклу записів у хмару та обробку помилок/повторних спроб.

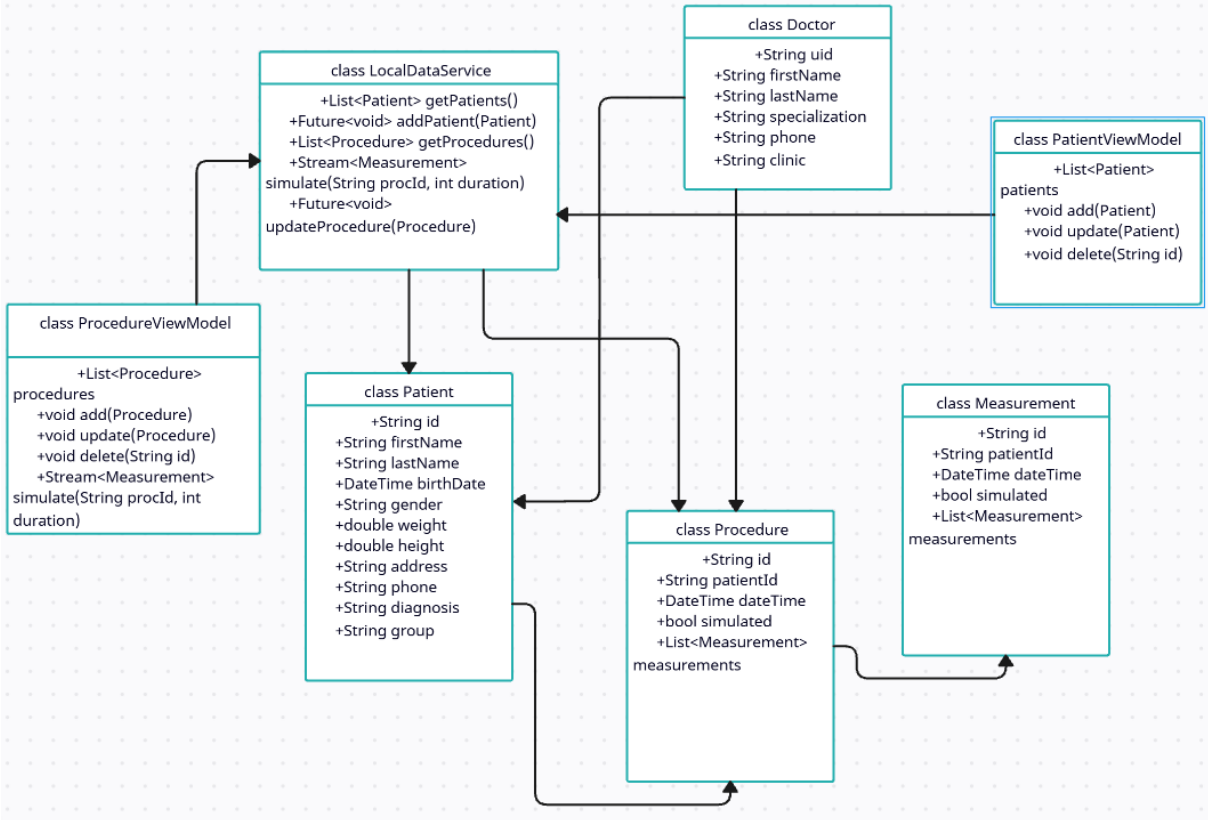

Рисунок 3.3 — Діаграма класів

ViewModel-и (PatientViewModel, ProcedureViewModel, GroupViewModel) наслідують від ChangeNotifier і містять у собі списки відповідних сутностей, методи завантаження/фільтрації/збереження даних та виклик notifyListeners() при

будь-якій зміні. Зі зв’язків видно, що один пацієнт може мати багато процедур, а одна процедура містить багато вимірювань.

3.5 Ключові алгоритми

У цьому підрозділі детально розглянуто ключові алгоритми, які забезпечують роботу застосунку: ініціалізацію й управління Bluetooth-підключенням, обробку й фільтрацію вимірювань, динамічне масштабування графіків у режимі реального часу, а також побудову черги CRUD-операцій та синхронізацію даних із Cloud Firestore. Для кожного алгоритму наведено текстове пояснення та ілюструвальну блок-схему.

3.5.1 Алгоритм ініціалізації та підключення BLE-пристрою

Після запуску застосунку, коли лікар вирішує почати Live-моніторинг, необхідно встановити надійне підключення до зовнішнього BLE-сенсора. Алгоритм ініціалізації складається з таких етапів:

Перевірка стану Bluetooth: на старті BLEService звертається до системного API (через FlutterReactiveBle) і перевіряє, чи увімкнено модуль Bluetooth у пристрої. Якщо статус “off”, застосунок показує діалог із проханням увімкнути Bluetooth. Після підтвердження користувачем відкривається системне вікно налаштувань, і, як тільки Bluetooth активується, алгоритм переходить до наступного кроку.

Сканування навколишніх BLE-пристроїв: BLEService виконує виклик до методу scanForDevices(withServices: [UUID_HeartRate, UUID_SpO2, UUID_BP]). Ця операція створює стрім результатів — кожні кілька секунд (наприклад, 5 s) надходять усі знайдені поблизу BLE-ґаджети, які відповідають зазначеним UUID.

Вибір пристрою та підключення: коли лікар натискає кнопку “Підключити”

біля обраного пристрою, UI викликає метод ViewModel-а, який делегує виклик у BLEService–з'єднання з вибраним ідентифікатором (deviceId). BLEService виконує системний виклик connectToDevice(deviceId) і переходить у стан очікування. Якщо підключення вдале, Service отримує об'єкт з інформацією про підключений сенсор та формує об'єкт типу QualifiedCharacteristic для кожної потрібної характеристики (HR, SpO_2, SYS, DIA).

Підписка на потік характеристик: після підключення BLEService викликає метод підписки на стріми характеристик, наприклад, bleDevice.subscribeToCharacteristic(characteristicUuid). Як тільки сенсор починає надсилати пакети байтів, BLEService отримує їх у колбеку (або через Stream) і передає в компонент-парсер, який розпаковує байти в числові значення кожного параметра.

3.5.2 Алгоритм обробки й фільтрації вимірювань

У режимі Live-моніторингу вимірювання можуть містити випадкові шумові значення (артефакти), тому перед відображенням у графіку застосовано кілька етапів обробки:

Отримання сирого потоку байтів: як тільки BLEService отримує черговий пакет байтів із сенсора, він передає їх у компонент-парсер (наприклад, MeasurementParser), який перетворює послідовність байтів у об'єкт Measurement із полями heartRate, spo2, systolic, diastolic і timestamp.

Створення кінцевого Measurement: після згладжування формується новий об'єкт Measurement, у якому поля heartRate, spo2, systolic, diastolic замінено на їхні SMA-значення, а timestamp залишено оригінальним. Цей об'єкт передається назад у BLEService.

Публікація в стрім ViewModel: BLEService публікує згладжене Measurement у стрім (наприклад, measurementStream.add(smoothedMeasurement)). ViewModel підписаний на цей стрім, тому кожен новий об'єкт приходить у ViewModel, який

додає його до внутрішнього списку currentMeasurements і виконує notifyListeners(). Після цього UI отримує оновлену колекцію точок і додає їх у графік.

3.5.3 Алгоритм побудови черги змін і синхронізації

У процесі Live-моніторингу важливо, аби графік завжди відображав найсучасніші дані та не виходив за межі видимої області. Для цього застосовано алгоритм динамічного підлаштування меж осей:

Встановлення початкових меж: при ініціалізації Live-графіка задаємо початкове вікно часу (наприклад, остання хвилина). Ліва межа осі X встановлена на t_0 — windowSize, права — на t_0, де t_0 — час початку сеансу, а windowSize = 60 с. Між осей Y за замовчуванням береться типовий діапазон, наприклад: heartRate [60, 100], spo2 [90, 100], SYS [80, 120], DIA [60, 80]. Ці межі можна коригувати вручну.

Додавання нової точки (t, value): якщо t (час нового вимірювання) перевищує поточну праву межу осі X (currentMaxX), необхідно пересунути вікно: нова ліва межа стає t — windowSize, а права — t. Потім перевіряємо, чи value виходить за поточні границі осі Y (currentMinY, currentMaxY). Якщо так, оновлюємо межі осі Y таким чином, щоб включити нове значення плюс невеликий запас (bufferMargin). Після зміни меж викликається віджет fl_chart для оновлення конфігурації осей (наприклад, через встановлення параметрів minX, maxX, minY, maxY);

4 ІНСТАЛЯЦІЯ ДОДАТКА ТА ДЕМОНСТРАЦІЯ ФУНКЦІОНАЛУ

У цьому розділі описано всі необхідні кроки для встановлення та налаштування мобільного застосунку, а також детально описано, як лікар взаємодіє з програмою на кожному етапі. Замість переліків ми використовуємо суцільний текст, де плавно переходять одне до одного пояснення, а там, де це найдоцільніше, вказано місце для вставки скріншотів—щоб на фінальному етапі в доповіді або документації було видно реальні знімки екрана, які ілюструють сказане.

4.1 Вимоги до обчислювальної техніки

Щоб додаток працював стабільно, мобільний пристрій повинен мати принаймні чотириядерний процесор із частотою не менше 1,5 GHz і близько 2 ГБ оперативної пам’яті. Такі мінімальні параметри гарантують, що прилад ізможе швидко обробляти потік даних від BLE-сенсора й оновлювати інтерактивні графіки без затримок. Для коректного відображення інтерфейсу з графіками, таблицями і формами рекомендується екран із роздільною здатністю щонайменше 720×1280 пікселів.

Що стосується BLE-модуля, то пристрій має підтримувати стандарт Bluetooth 4.0 (Low Energy) або новіший, адже лише за таких умов можна з’єднатися з медичним сенсором і отримувати дані про серцевий ритм, насичення киснем та артеріальний тиск. У разі використання старіших моделей телефонів без BLE-підтримки застосунок не зможе виконувати Live-моніторинг.

Операційна система має бути Android версії 8.0 або вище (рекомендовано Android 10 і новіші) або iOS мінімум версії 12.0 (з iOS 14 вважається ідеальною). Це пов’язано з тим, що Flutter-бібліотеки, які використовуються у проєкті (наприклад, flutter_reactive_ble для роботи з Bluetooth і fl_chart для побудови графіків), гарантовано підтримують саме ці версії ОС. Для розробки та

налагодження застосунку на Android необхідно мати встановлений Android Studio з Android SDK (API 29 чи вище) і налаштований емулятор або фізичний пристрій із увімкненим режимом розробника. На macOS потрібно додатково встановити Xcode (версії 12.0 чи новішої) для складання та налагодження під iOS.

Перед самим запуском проєкту у середовищі Flutter слід переконатися, що встановлена актуальна версія Flutter SDK (≥ 3.0.0) з відповідним Dart SDK. Команда flutter doctor у терміналі покаже статус усіх необхідних компонентів (Android SDK, Xcode, емулятори тощо). [1]

Далі в корені проєкту слід виконати flutter pub get, щоб завантажити всі залежності, зазначені у pubspec.yaml: окрім Flutter і Dart, це flutter_reactive_ble, fl_chart, provider, cloud_firestore, firebase_auth, shared_preferences і sqflite. [2] Після успішної інсталяції пакунків можна запускати застосунок командою flutter run.

Далі необхідно інтегрувати проект з Firebase. Для цього в консолі Firebase створюють новий проєкт, додають до нього Android-додаток, завантажують файл google-services.json і поміщають його в папку android/app/.

Аналогічно додають iOS-додаток, завантажують GoogleService-Info.plist і копіюють його до ios/Runner/. [10] Після цього, згідно з інструкціями Firebase, у файлі android/app/build.gradle підключають плагін Google Services, а в ios/Podfile переконуються, що всі бібліотеки CocoaPods інсталюються коректно. З цим налаштуванням при кожному вході через Google Sign-In застосунок отримуватиме токен аутентифікації, а Firestore зможе зберігати дані лікаря, пацієнтів, сеансів та вимірювань.

4.2 Демонстрація функціоналу та сценарії роботи

У цьому розділі описано, як лікар взаємодіє з застосунком, починаючи з авторизації та закінчуючи груповим аналізом вимірювань. Народні фрагменти коду замінено детальним описом дій та їхніх наслідків.

4.2.1 Авторизація через Google Sign-In

Після інсталяції та першого запуску застосунку лікар потрапляє на екран, де великий логотип HypoxiTherapy Monitor виділений на білому фоні, а під ним розташована єдина кнопка “Увійти через Google”. Натискання на цю кнопку викликає вбудований у Flutter плагін Firebase Auth, який відкриває стандартний системний діалог вибору облікового запису Google. [10] Якщо лікар уже має хоча б один обліковий запис, йому достатньо тапнути по відповідній іконці Google, і після успішної аутентифікації він негайно повернеться в застосунок (рис. 4.1).

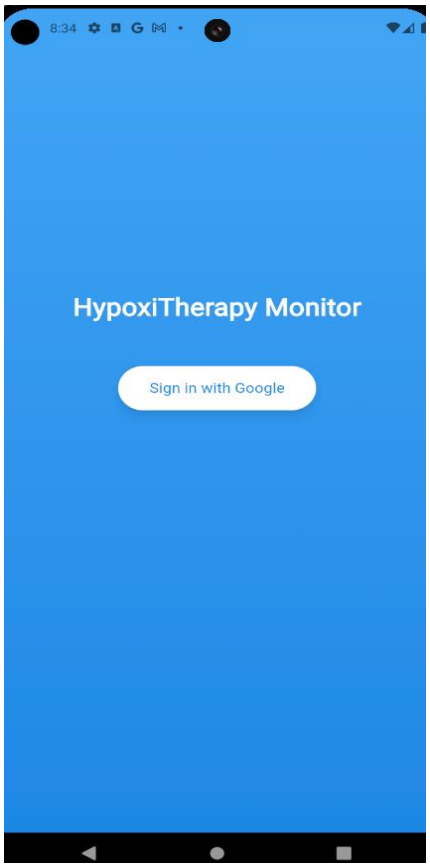

Рисунок 4.1 — Екран авторизації

У разі першого входу в Firebase перевірятиметься, чи існує документ із цим email у колекції doctors. Якщо документа немає, створюється новий запис із полями uid (ідентифікатор користувача від Firebase), fullName, email і photoURL. Всі ці дані зберігаються одразу й у Firestore, й у локальних налаштуваннях (за допомогою

shared_preferences). Тому при наступному вході лікар бачить одразу свій аватар та ім’я у верхній частині Dashboard без зайвих затримок.

4.2.2 Головний екран (Dashboard) та навігація

Після успішної автентифікації лікар потрапляє на головний екран Dashboard. У верхній частині видно звернення: “Ласкаво просимо, Dr. [ПІБ]”, де замість [ПІБ] динамічно підтягується ім’я, яке було збережено в документах Firestore. Праворуч збоку — невелика іконка виходу з системи (три вертикальні точки → “Вийти”). Під цим банером розташовані чотири картки, розміщені у двоколонній сітці. Кожна картка відповідає окремому модулю: пацієнти, процедури, групи та профіль (показано піктограму у стилі Material 3). Кольорова схема карток ґенерується динамічно за допомогою ColorScheme.fromSeed(seedColor: Colors.blue), забезпечуючи єдиний стиль інтерфейсу (рис. 4.2).

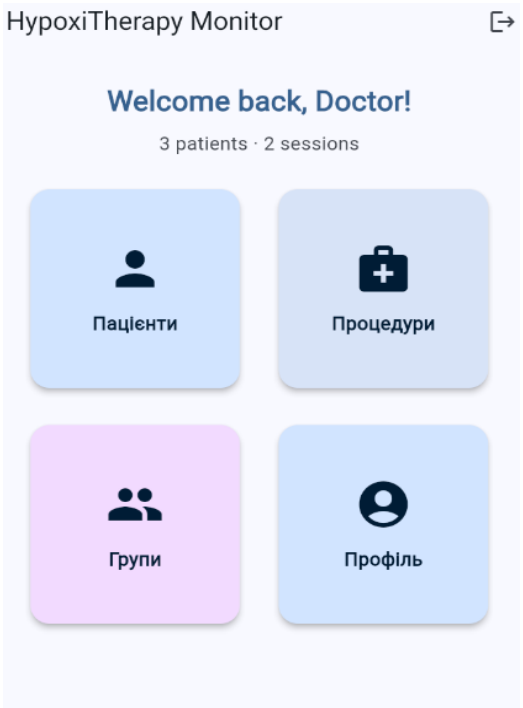

Рисунок 4.2 — Головний екран

При першому завантаженні кожна картка з’являється з ефектом плавного з’явлення та легкого збільшення (fade-in + scale). Якщо лікар натисне й утримає картку довше, з’явиться контекстне меню з опціями швидкого доступу (наприклад, “Створити нового пацієнта” або “Почати Live-сеанс”), що пришвидшує виконання рутинних дій.

Завдяки цьому навігація є інтуїтивною та швидкою: одним натисканням лікар переходить на необхідний екран, а довгим натисканням одразу може розпочати дію, оминути зайві кроки.

4.2.3 Керування пацієнтами

Як тільки лікар торкається картки “Пацієнти”, відкривається екран зі списком усіх зареєстрованих пацієнтів. Він реалізований як вертикальний ListView, де кожна картка містить аватарку пацієнта (або умовну “заглушку”, якщо фото не було додано), великим шрифтом виведено прізвище та ім’я, під ним — поточний вік, що автоматично обчислюється з дати народження, а далі — короткий опис діагнозу та позначка групи, якщо пацієнт до неї належить (рис. 4.3).

Над списком розташовано поле пошуку із піктограмою лупи, куди лікар може вводити прізвище чи ім’я. Після невеликої затримки (300 мілісекунд) застосунок фільтрує список, показуючи лише ті картки, які відповідають тексту. Таким чином, за наявності кількох десятків чи сотень записів лікар швидко знайде потрібного пацієнта.

У правому нижньому куті екрану розміщена плаваюча кнопка з іконкою “+”. При торканні з’являється модальна форма, у якій на одному екрані наведені всі поля для створення нового пацієнта: спочатку два текстові поля для прізвища й імені, далі — поле “Дата народження”, яке відкриває вбудований DatePicker із календарем.

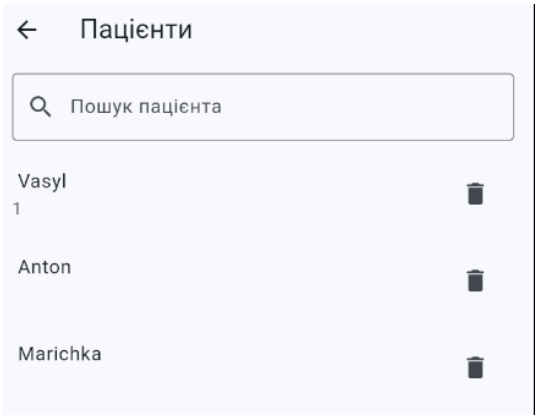

Рисунок 4.3 — Екран зі списком пацієнтів

Поруч розташовані поля для введення контактного телефону та email, а в самому низу — текстове поле “Діагноз” і випадаючий список із назвами груп (можна залишити порожнім, якщо пацієнт ще не входить до жодної групи).

Після введення інформації лікар натискає “Зберегти”, внаслідок чого застосунок перевіряє коректність усіх полів (наприклад, email має містити символ @, телефон — цифри, дата народження не може бути в майбутньому).

Якщо всі дані коректні, ViewModel формує об’єкт Patient і передає його SyncService, а UI одразу закриває форму й відображає оновлений список із новою карткою у відповідному місці.

При торканні по існуючій картці пацієнта відкривається екран “Деталі пацієнта”. Тут докладно наведено все те, що лікар вводив раніше: аватар, прізвище, ім’я, дату народження (з розрахованим віком), контактний телефон, email, діагноз і назву групи (рис. 4.4).

У верхньому правому куті кожного екрану деталізації є кнопки “Редагувати” (піктограма олівця) та “Видалити” (піктограма смітника). Коли лікар обирає “Редагувати”, відкривається та сама форма, що й при створенні, але всі поля вже заповнені поточними значеннями.

Після редагування та натискання “Зберегти” ViewModel передає оновлені дані до SyncService, а UI оновлює картку в списку без перезавантаження всієї сторінки.

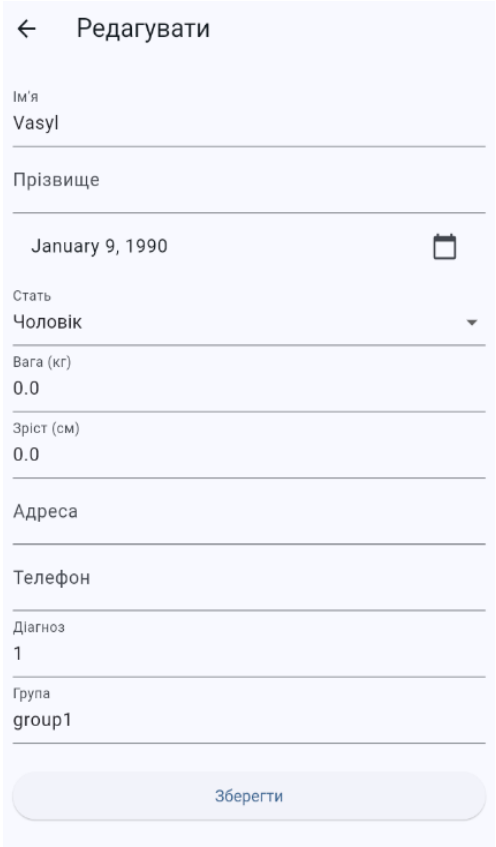

Рисунок 4.4 — Екран деталей про пацієнта

Якщо лікар вирішує видалити пацієнта, натискання “Видалити” викликає підтверджувальний діалог: “Ви впевнені, що хочете видалити пацієнта Іванов Іван? Ця дія незворотна.” У разі підтвердження ViewModel передає запит SyncService, а UI одразу видаляє картку зі списку. Усі подальші дані про цього пацієнта (сеанси, вимірювання) теж видаляються в хмарі за логікою налаштованих правил Firestore (cascade-delete чи ручна перевірка).

4.2.4 Створення та проведення сеансу

Коли лікар перейшов із Dashboard на екран “Процедури”, він бачить список вже створених сеансів із зазначенням імені пацієнта, дати й часу та статусу (наприклад, “Live — завершено” або “Live — у процесі”) (рис. 4.5).

Натискання на плаваючу кнопку “+” відкриває форму для створення нового

сеансу. Спочатку лікар вибирає пацієнта зі списку, що з’являється у вигляді автодоповнення: достатньо ввести перші літери прізвища, і система відфільтрує доступні варіанти. Далі натиснути поле “Дата й час” означає відкриття системного селектора DateTimePicker, де можна встановити дату й точний час. Якщо лікар хоче розпочати сеанс негайно, достатньо залишити поточні дату та час. [9] Потім залишається лише натиснути “Почати Live-моніторинг”, і застосунок автоматично створює запис про новий сеанс.

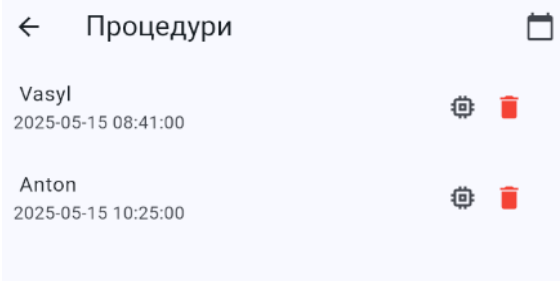

Рисунок 4.5 — Екран зі списком процедур

Якщо Bluetooth у цей момент вимкнено, UI показує спливаюче повідомлення “Увімкніть Bluetooth для підключення до сенсора” і залишає лікаря на екрані для виправлення (рис. 4.6).

Після вмикання Bluetooth починається автоматичне сканування BLE-пристроїв, а знизу з’являється список знайдених сенсорів із назвами (або ідентифікаторами), рівнем сигналу (RSSI) та кнопкою “Підключити” біля кожного.

Як тільки лікар натисне “Підключити” біля правильного сенсора, застосунок встановлює з’єднання, знаходить характеристики, що відповідають за ЧСС, SpO_2 та тиск, і після успішного підключення UI змінює заголовок на “Пристрій готовий — Live-моніторинг”.

У вкладці “Live-моніторинг” з’являється основний графік, де одночасно позначено чотири лінії: серцевий ритм (HR), насичення крові киснем (SpO_2), систолічний тиск (SYS) та діастолічний тиск (DIA). Ось X відповідає часові (у хвилинах / секундах), а вісь Y адаптується динамічно, якщо значення виходять за межі початкових границь.

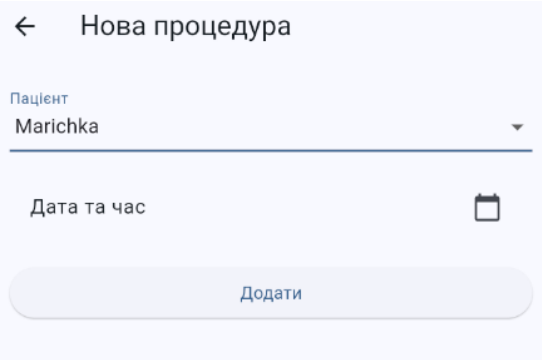

Рисунок 4.6 — Екран створення процедури.

Наприклад, якщо нормальний інтервал HR задано 60–100 bpm, а нова точка показує 110 bpm, вісь Y розшириться так, щоб усі дані були видно. Це оновлення відбувається автоматично без затримок, завдяки алгоритму динамічного масштабування (описано в 3.9.3).

Кожного разу, коли надходить нове вимірювання, BLEService перетворює його у об’єкт Measurement, проганяє через фільтрацію (наприклад, виключає явні артефакти) і потім згладжує значення за допомогою ковзного середнього з останніх п’яти точок. Відтак ViewModel отримує готовий об’єкт і одразу оновлює графік, додаючи нову точку, після чого UI перемальовує лише ту ділянку, яка змінилася. (рис. 4.7)

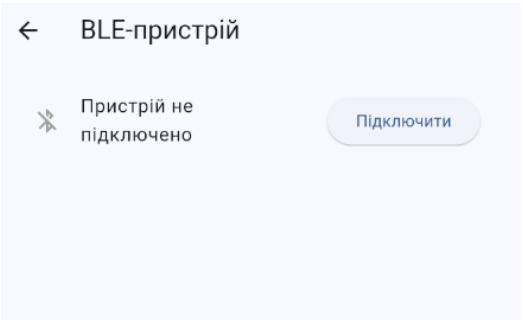

Рисунок 4.7 — Екран підключення до пристрою.

Нижче під графіком розташована таблиця у вигляді DataTable, де наведено останні 20 вимірювань із чотирма стовпцями: “Час” (формат HH:mm:ss), “HR”

(bpm), “SpO_2” (%) и “SYS/DIA” (мм рт. ст.). Таблицю можна сортувати за будь-яким стовпцем, торкнувшись його заголовка (наприклад, щоб побачити найвищий HR чи найнижчий SpO_2). Відтак лікар, не відриваючись від графіка, може перевірити, у який саме момент відбулося максимальне чи мінімальне значення і чи відповідає воно загальному тренду (рис. 4.8).

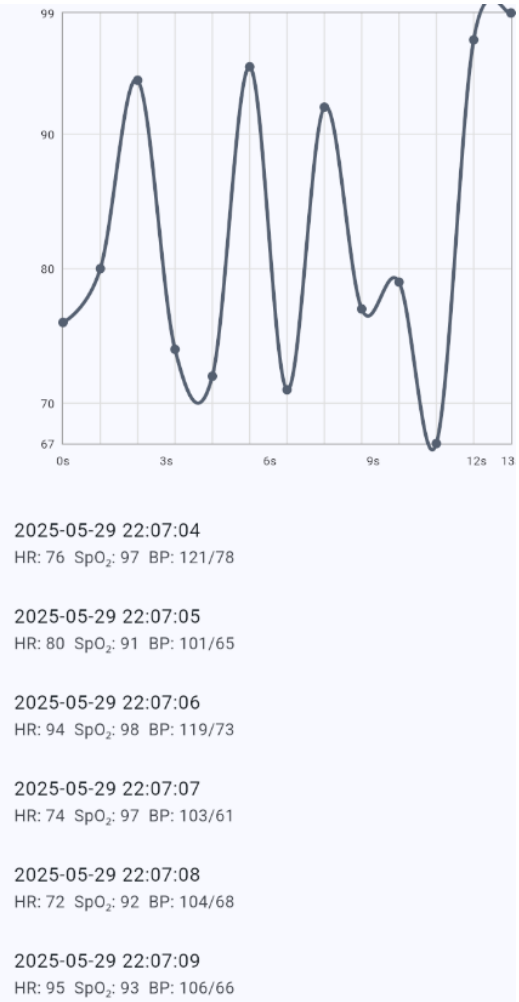

Рисунок 4.8 — Екран проведення процедури в реальному часі

Коли лікар вважає сеанс завершеним (наприклад, досягнуто поставленого часу або виникли клінічні показання до зупинки), він натискає кнопку “Завершити сеанс”. ViewModel зупиняє стрім від BLEService та збирає всі накопичені об’єкти Measurement із внутрішнього буфера.

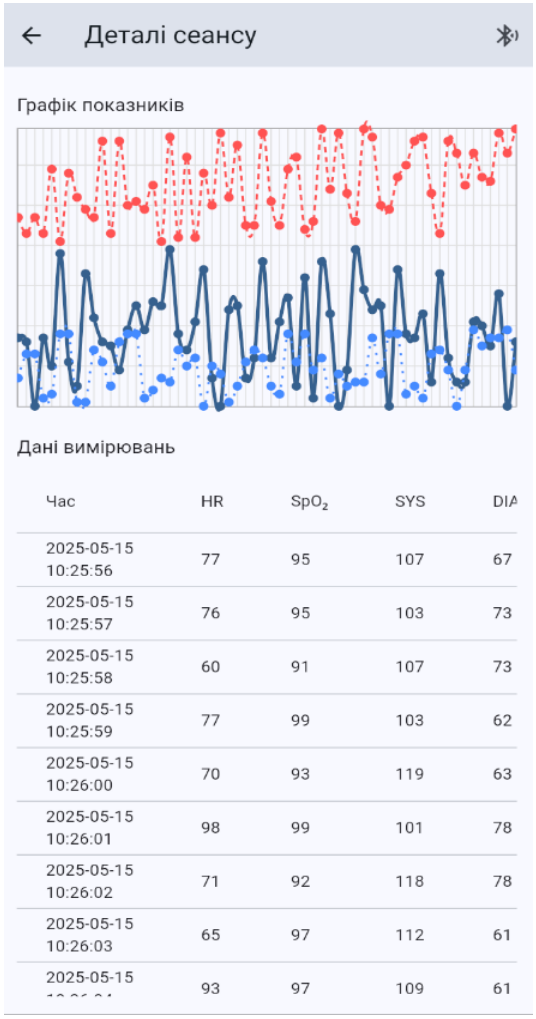

Рисунок 4.9 — Екран таблиці вимірювань проведеної процедури і графік.

Потім формується фінальний об’єкт ProcedureResult із часовою міткою початку, тривалості та списком усіх точок. Цей результат передається SyncService.enqueueOperation із типом “update” (додаєся підколекція measurements у документі seansu Firestore) (рис. 4.9).

Після успішного запису UI повертається до списку “Процедури”, а статус поточного сеансу змінюється з “Live — у процесі” на “Live — завершено”. Якщо в процесі запису виникає якась помилка (наприклад, втрата Інтернету), SyncService робить декілька спроб із короткими затримками (експоненційний back-off), а ViewModel водночас відобразить лікарю сповіщення: “Дані сеансу збережено локально.

4.2.5 Відображення та аналіз даних під час Live-сеансу

Під час самого Live-сеансу графік у режимі реального часу дозволяє лікарю уважно стежити за динамікою всіх показників. Кожне нове значення негайно передається відображається на графіку завдяки бібліотеці fl_chart.

Завдяки інтерактивним підказкам (tooltip) лікар може торкнутися будь-якої точкової позначки і моментально дізнатися точний час та значення параметра, що допомагає виявляти критичні моменти під час сеансу. Зі зміною значень у графіку вісь Y може автоматично розширюватися чи звужуватися, щоб ніхто момент не виходив за межі видимої області.

Це особливо важливо в тих випадках, коли під час гіпокситерапії частота серцебиття може почати знижуватися або підвищуватися за короткий проміжок часу. Таблиця нижче графіка містить останні 20 вимірювань у форматі, що дозволяє обрати будь-яку точку даних і побачити її в табличному вигляді.

Сортування стовпців допомагає швидко знайти найнижчі значення SpO_2 або найвищі значення артеріального тиску серед усіх записів поточного сеансу. Якщо кількість вимірювань стає більше 20, лікар може прокручувати таблицю вгору чи вниз або переходити до попередніх (завершених сеансів), щоб переглянути історію.

Якщо в ході Live-сеансу якийсь із показників виходить за межі допустимого діапазону (наприклад, SpO_2 падає нижче 90 % або HR зростає понад 120 bpm), застосунок виводить автоматичне текстове повідомлення над графіком: “Увага: SpO_2 < 90 % більше 10 секунд”. Це повідомлення допомагає лікарю вчасно відреагувати: наприклад, змінити параметри кисневої подачі або реструктурувати протокол сеансу.

4.2.6 Керування групами пацієнтів і порівняльний аналіз

Для виконання групового аналізу лікар на Dashboard натискає картку “Групи”. Відкривається екран із переліком створених раніше груп.

Під час першого використання список може бути порожнім, тому зверху є кнопка “Нова група”. Після торкання відкривається діалог із полем “Назва групи”, у яке лікар вводить описову назву — наприклад, “Діабетична група” або “Пацієнти ≥ 60 років” (рис. 4.10).

Після підтвердження група миттєво додається до списку на екрані та зберігається в Firestore як документ із полями name (назва) та порожнім масивом patientIds (поки що без учасників).

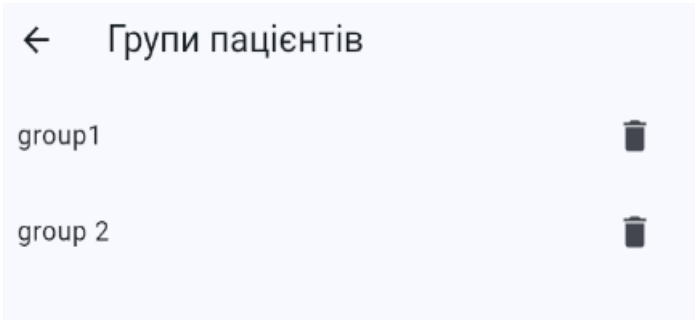

Рисунок 4.10 — Екран зі списком груп.

Щойно група створена, лікар торкається її назви, і відкривається екран деталей групи, поділений на дві колонки.

Ліва колонка містить заголовок “Члени групи” та порожній список (поки що), а права — “Доступні пацієнти” із переліком усіх зареєстрованих пацієнтів (рис. 4.11).

Під час завантаження списків застосунок підтягує з Firestore документи patients і фільтрує наявних пацієнтів: ті, чиї id містяться в масиві patientIds, потрапляють у ліве поле як “члени групи”, а решта — у праве як “доступні”.

Кожен запис показує аватарку (або заглушку), прізвище та ім’я, рік народження (вік) і діагноз. Біля кожного запису в лівій колонці розміщена кнопка “–” (видалити з групи), а в правій — кнопка “+” (додати до групи).

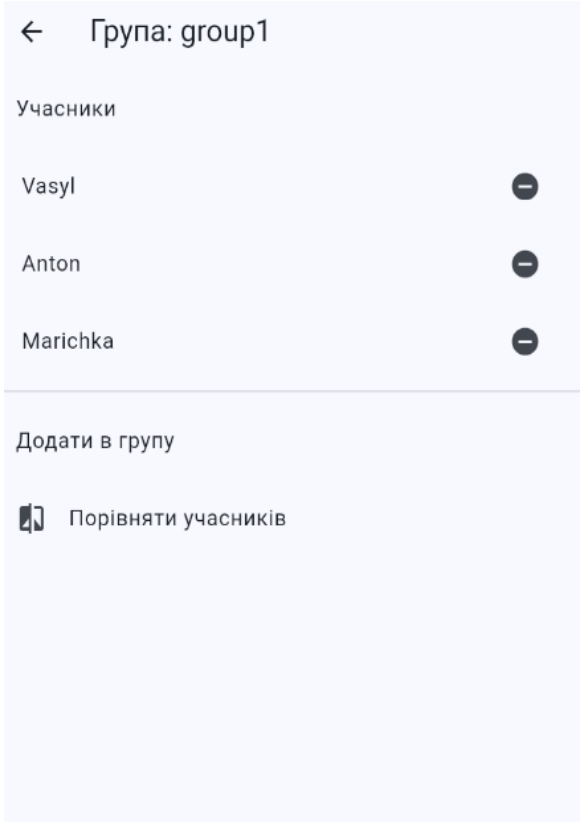

Рисунок 4.11 — Екран деталей про групу

При натисканні “+” обраний пацієнт миттєво “злітає” анімацією в ліву колонку. Одночасно ViewModel оновлює локальний список пацієнтів цієї групи, додаючи patientId до patientIds, а SyncService.enqueueOperation виконує запис змін у Firestore (тип “update” з новим масивом patientIds).

Аналогічно, натискання “–” викликає видалення patientId з масиву, оновлення списків у UI та чергове оновлення хмари.

Після того, як у групі зібрано двох і більше пацієнтів, у верхній частині екрану статистично з’являється кнопка “Порівняти показники”, натискання на яку веде до спеціального екрану порівняння.

Там система будує єдиний комбінований графік, де кожна крива (HR, SpO_2, SYS, DIA) відображається для кожного пацієнта власним стилем лінії (різна колірна гама та пунктирні/суцільні лінії) (рис. 4.12).

Рисунок 4.12 — Екран порівняння процедур в групі

Також доступна таблиця, у якій в одному рядку наведено одночасні значення всіх пацієнтів за певну часову мітку (наприклад, 14:32:01). Таким чином лікар може чітко побачити, хто з пацієнтів має найнижчий рівень насичення киснем у певний момент і порівняти це з іншими показниками.

4.2.7 Редагування профілю лікаря та вихід із системи

Усе, що стосується особистих даних лікаря, зосереджено на екрані “Профіль”, до якого можна потрапити з Dashboard. Тут лікар бачить власний аватар (за замовчуванням це круг різного кольору з ініціалами, якщо фото не було

завантажено раніше), нижче — два текстові поля з ім’ям і прізвищем, а ще нижче — email (який не піддається редагуванню) і телефон.

Непотрібне поле “Спеціалізація” можна заповнити, якщо потрібно додати, наприклад, “Кардіолог” або “Невролог”. Поруч із фото є кнопка “Змінити фото”, яка відкриває діалог вибору між камерою й галереєю. Після вибору нового зображення пацієнтське фото надсилається до Firebase Storage, а в документі лікаря в Firestore оновлюється поле photoURL. Крім того, місцево в SharedPreferences зберігається URL, щоб при наступному вході відразу підтягувався останній аватар.

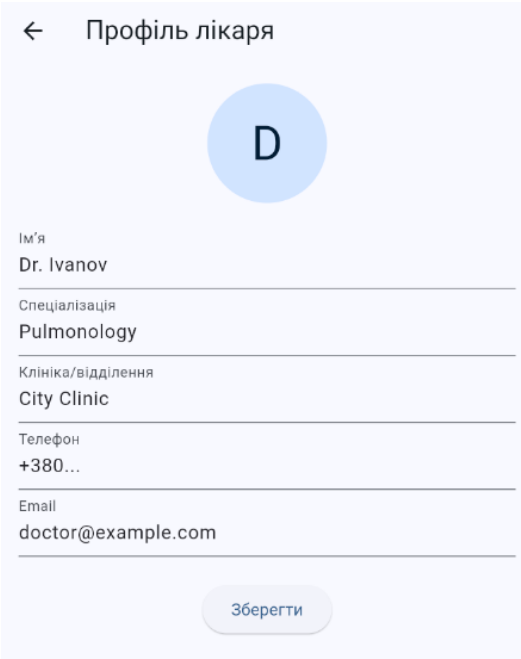

Рисунок 4.13 — Екран профілю лікаря

Якщо лікар змінює текстові дані (ім’я, телефон або спеціалізацію), після натискання “Зберегти” відбувається валідація форматів (наприклад, телефон має починатися з коду країни “+380”), а потім об’єкт DoctorProfile передається SyncService.enqueueOperation із типом “update”. У разі успіху з’являється повідомлення “Профіль успішно оновлено”, а при помилці (наприклад, через втрату з’єднання) — “Не вдалося зберегти зміни. Спробуйте пізніше.” Після оновлення UI миттєво відображає актуальні дані в усіх екранах, де вони можуть використовуватися (рис. 4.13).

ВИСНОВКИ

У рамках дипломної роботи було створено мобільний застосунок, призначений для комплексного контролю фізіологічних показників пацієнтів під час сеансів гіпокситерапії. Розробка охопила весь цикл життєвого циклу продукту: від аналізу вимог медичної галузі й вибору технологічного стека до реалізації UI на Flutter із Material 3, інтеграції Provider + MVVM для керування станом, а також налаштування бекенду на базі Firebase Authentication і Firestore. [4; 6]

Проведено глибокий аналіз існуючих рішень, що підтвердив необхідність гнучкої підтримки live-моніторингу через BLE, тренувальної симуляції та групового порівняння даних. Для збереження автономності й високої доступності реалізовано offline-кешування через SharedPreferences та фонову синхронізацію змін за допомогою WorkManager.

Ключові екрани — від реєстрації й Dashboard до деталей сеансу і порівняльної аналітики — забезпечують зрозумілу навігацію й інтуїтивну взаємодію лікаря з додатком.

Тестування показало високу стабільність UI, точність візуалізації графіків fl_chart і надійність BLE-з’єднання. [7] Використані рішення гарантують безпечне зберігання й обробку медичної інформації, готовність до подальшого розширення функціональності та масштабування в межах сучасних вимог охорони здоров’я.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Flutter Documentation. *Flutter*. URL: https://flutter.dev/docs (date of access: 22.05.2025).

2. Dart Language Tour. *Dart*. URL: https://dart.dev/guides/language/language-tour (date of access: 22.05.2025).

3. Firebase Authentication Documentation. *Firebase*. URL: https://firebase.google.com/docs/auth (date of access: 20.05.2025).

4. Cloud Firestore Documentation. *Firebase*. URL: https://firebase.google.com/docs/firestore (date of access: 20.05.2025).

5. Sobolenko M. Provider package. *pub.dev*. URL: https://pub.dev/packages/provider (date of access: 21.05.2025).

6. MVVM architecture with Provider. *Flutter*. URL: https://flutter.dev/docs/development/data-and-backend/state-mgmt/simple (date of access: 21.05.2025).

7. Susmelj I. fl_chart 0.65.1. *pub.dev*. URL: https://pub.dev/packages/fl_chart (date of access: 20.05.2025).

8. Akers A. intl package. *pub.dev*. URL: https://pub.dev/packages/intl (date of access: 21.05.2025).

9. uuid package. *pub.dev*. URL: https://pub.dev/packages/uuid (date of access: 21.05.2025).

10. Field D. google_fonts package. *pub.dev*. URL: https://pub.dev/packages/google_fonts (date of access: 21.05.2025).

ДОДАТОК А

Програмна реалізація мобільного застосунку супроводження пацієнтів під час проведення сеансів гіпокистерапії

НТУУ “КПІ імені Ігоря Сікорського”

Аркушів 6

Київ — 2025

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

import '../models/procedure.dart';
import '../viewmodels/procedure_viewmodel.dart';
import '../viewmodels/patient_viewmodel.dart';
import 'simulation_screen.dart';
import 'bluetooth_device_screen.dart';

class ProcedureDetailScreen extends StatefulWidget {
  final Procedure? procedure;
  const ProcedureDetailScreen({this.procedure, Key? key}) : super(key: key);

  @override
  State<ProcedureDetailScreen> createState() => _ProcedureDetailScreenState();
}

class _ProcedureDetailScreenState extends State<ProcedureDetailScreen> {
  String? _patientId;
  DateTime? _dateTime;

  @override
  void initState() {
    super.initState();
    _patientId = widget.procedure?.patientId;
    _dateTime  = widget.procedure?.dateTime;
  }

  @override
  Widget build(BuildContext context) {
    final procVm = Provider.of<ProcedureViewModel>(context);
    final patVm  = Provider.of<PatientViewModel>(context);
    final measurements = widget.procedure?.measurements ?? [];

    // Побудова точок для графіків
    final spotsHR  = measurements.asMap().entries
        .map((e) => FlSpot(e.key.toDouble(), e.value.heartRate.toDouble()))
        .toList();
    final spotsSYS = measurements.asMap().entries
        .map((e) => FlSpot(e.key.toDouble(), e.value.systolic.toDouble()))
        .toList();
    final spotsDIA = measurements.asMap().entries
        .map((e) => FlSpot(e.key.toDouble(), e.value.diastolic.toDouble()))
        .toList();

    return Scaffold(
      appBar: AppBar(
        title: Text(
          widget.procedure == null ? 'Нова процедура' : 'Деталі сеансу',
        ),
        actions: [
          if (widget.procedure != null)
            IconButton(
              icon: const Icon(Icons.bluetooth_searching),
              tooltip: 'Live з BLE-пристрою',
              onPressed: () => Navigator.push(
                context,
                MaterialPageRoute(
                  builder: (_) =>
                      BluetoothDeviceScreen(procedure: widget.procedure!),
                ),
              ),
            ),
          if (widget.procedure != null && !widget.procedure!.simulated)
            IconButton(
              icon: const Icon(Icons.play_circle_outline),
              tooltip: 'Симуляція',
              onPressed: () => Navigator.push(
                context,
                MaterialPageRoute(
                  builder: (_) =>
                      SimulationScreen(procedure: widget.procedure!),
                ),
              ),
            ),
        ],
      ),
      body: Padding(
        padding: const EdgeInsets.all(16),
```

```
child: ListView(
  children: [
    // Вибір пацієнта
    DropdownButtonFormField<String>(
      value: _patientId,
      decoration: const InputDecoration(labelText: 'Пацієнт'),
      items: patVm.patients.map((p) {
        return DropdownMenuItem(
          value: p.id,
          child: Text('${p.lastName} ${p.firstName}'),
        );
      }).toList(),
      onChanged: (v) => setState(() => _patientId = v),
    ),
    const SizedBox(height: 16),
    // Вибір дати/часу
    ListTile(
      title: Text(
        _dateTime == null
            ? 'Дата та час'
            : _dateTime!.toLocal().toString().split('.')[0],
      ),
      trailing: const Icon(Icons.calendar_today),
      onTap: () async {
        final date = await showDatePicker(
          context: context,
          initialDate: _dateTime ?? DateTime.now(),
          firstDate: DateTime(2000),
          lastDate: DateTime.now().add(const Duration(days: 365)),
        );
        if (date != null) {
          final time = await showTimePicker(
            context: context,
            initialTime: TimeOfDay.fromDateTime(
                _dateTime ?? DateTime.now()),
          );
          if (time != null) {
            setState(() {
              _dateTime = DateTime(
                date.year,
                date.month,
                date.day,
                time.hour,
                time.minute,
              );
            });
          }
        }
      },
    ),
    const SizedBox(height: 16),
    // Кнопка Додати/Зберегти
    ElevatedButton(
      onPressed: () async {
        if (_patientId != null && _dateTime != null) {
          final id = widget.procedure?.id ?? procVm.newId();
          final pr = Procedure(
            id: id,
            patientId: _patientId!,
            dateTime: _dateTime!,
            simulated: widget.procedure?.simulated ?? false,
            measurements: measurements,
          );
          if (widget.procedure == null) {
            await procVm.add(pr);
          } else {
            await procVm.update(pr);
          }
          Navigator.pop(context);
        }
      },
      child: Text(
          widget.procedure == null ? 'Додати' : 'Зберегти'),
    ),

    // Графік та таблиця, якщо є вимірювання
    if (measurements.isNotEmpty) ...[
      const SizedBox(height: 32),
      Text('Графік показників',
          style: Theme.of(context).textTheme.titleMedium),
```

```
            const SizedBox(height: 8),
            SizedBox(
              height: 250,
              child: LineChart(
                LineChartData(
                  gridData: FlGridData(
                    show: true,
                    drawVerticalLine: true,
                    drawHorizontalLine: true,
                    verticalInterval: 1,
                    horizontalInterval: 10,
                    getDrawingHorizontalLine: (y) =>
                        FlLine(color: Colors.grey.shade300, strokeWidth: 1),
                    getDrawingVerticalLine: (x) =>
                        FlLine(color: Colors.grey.shade300, strokeWidth: 1),
                  ),
                  borderData: FlBorderData(
                    show: true,
                    border: Border.all(color: Colors.grey),
                  ),
                  titlesData: FlTitlesData(show: false),
                  lineTouchData: LineTouchData(enabled: true),
                  lineBarsData: [
                    LineChartBarData(
                      spots: spotsHR,
                      isCurved: true,
                      color: Theme.of(context).colorScheme.primary,
                      barWidth: 3,
                    ),
                    LineChartBarData(
                      spots: spotsSYS,
                      isCurved: true,
                      color: Colors.redAccent,
                      barWidth: 2,
                      dashArray: [5, 4],
                    ),
                    LineChartBarData(
                      spots: spotsDIA,
                      isCurved: true,
                      color: Colors.blueAccent,
                      barWidth: 2,
                      dashArray: [2, 6],
                    ),
                  ],
                ),
              ),
            ),
            const SizedBox(height: 24),
            Text('Дані вимірювань',
                style: Theme.of(context).textTheme.titleMedium),
            const SizedBox(height: 8),
            DataTable(
              columns: const [
                DataColumn(label: Text('Час')),
                DataColumn(label: Text('HR')),
                DataColumn(label: Text('SpO₂')),
                DataColumn(label: Text('SYS')),
                DataColumn(label: Text('DIA')),
              ],
              rows: measurements.map((m) {
                return DataRow(cells: [
                  DataCell(Text(m.timestamp
                      .toLocal()
                      .toString()
                      .split('.')[0])),
                  DataCell(Text('${m.heartRate}')),
                  DataCell(Text('${m.spo2}')),
                  DataCell(Text('${m.systolic}')),
                  DataCell(Text('${m.diastolic}')),
                ]);
              }).toList(),
            ),
          ],
        ],
      ),
    ),
  );
 }
}
```

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../services/auth_service.dart';
import 'login_screen.dart';
import 'patient_list_screen.dart';
import 'procedure_list_screen.dart';
import 'profile_screen.dart';
import 'patient_groups_screen.dart';
import '../viewmodels/patient_viewmodel.dart';
import '../viewmodels/procedure_viewmodel.dart';

class DashboardScreen extends StatelessWidget {
  const DashboardScreen({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final patientCount = context.watch<PatientViewModel>().patients.length;
    final procedureCount = context.watch<ProcedureViewModel>().procedures.length;
    final colorScheme = Theme.of(context).colorScheme;

    return Scaffold(
      appBar: AppBar(
        title: const Text('HypoxiTherapy Monitor'),
        actions: [
          IconButton(
            icon: const Icon(Icons.logout),
            onPressed: () async {
              await AuthService.signOut();
              Navigator.pushReplacement(
                context,
                MaterialPageRoute(builder: (_) => const LoginScreen()),
              );
            },
          ),
        ],
      ),
      body: SafeArea(
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.center,
          children: [
            const SizedBox(height: 24),
            Text(
              'Welcome back, Doctor!',
              style: TextStyle(
                fontSize: 24,
                fontWeight: FontWeight.bold,
                color: colorScheme.primary,
              ),
            ),
            const SizedBox(height: 8),
            Text(
              '$patientCount patients · $procedureCount sessions',
              style: TextStyle(
                fontSize: 16,
                color: colorScheme.onSurfaceVariant,
              ),
            ),
            const SizedBox(height: 24),
            Expanded(
              child: GridView.count(
                padding: const EdgeInsets.symmetric(horizontal: 32),
                crossAxisCount: 2,
                mainAxisSpacing: 24,
                crossAxisSpacing: 24,
                childAspectRatio: 1,
                children: [
                  _DashboardCard(
                    icon: Icons.person,
                    label: 'Пацієнти',
                    color: colorScheme.primaryContainer,
                    onTap: () => Navigator.push(
                      context,
                      MaterialPageRoute(builder: (_) => const PatientListScreen()),
                    ),
                  ),
                  _DashboardCard(
                    icon: Icons.medical_services,
                    label: 'Процедури',
                    color: colorScheme.secondaryContainer,
                    onTap: () => Navigator.push(
```

```
                  context,
                  MaterialPageRoute(builder: (_) => const ProcedureListScreen()),
                ),
              ),
              _DashboardCard(
                icon: Icons.group,
                label: 'Групи',
                color: colorScheme.tertiaryContainer,
                onTap: () => Navigator.push(
                  context,
                  MaterialPageRoute(builder: (_) => const PatientGroupsScreen()),
                ),
              ),
              _DashboardCard(
                icon: Icons.account_circle,
                label: 'Профіль',
                color: colorScheme.primaryContainer,
                onTap: () => Navigator.push(
                  context,
                  MaterialPageRoute(builder: (_) => const ProfileScreen()),
                ),
              ),
            ],
          ),
        ),
      ],
    ),
  ),
 );
 }
}

class _DashboardCard extends StatelessWidget {
 final IconData icon;
 final String label;
 final Color color;
 final VoidCallback onTap;

 const _DashboardCard({
  required this.icon,
  required this.label,
  required this.color,
  required this.onTap,
  Key? key,
 }) : super(key: key);

 @override
 Widget build(BuildContext context) {
  return Card(
   shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(16)),
   elevation: 4,
   color: color,
   child: InkWell(
    borderRadius: BorderRadius.circular(16),
    onTap: onTap,
    child: Center(
     child: Column(
      mainAxisSize: MainAxisSize.min,
      children: [
       Icon(icon, size: 48, color: Theme.of(context).colorScheme.onPrimaryContainer),
       const SizedBox(height: 12),
       Text(
        label,
        style: TextStyle(
         fontSize: 16,
         fontWeight: FontWeight.w600,
         color: Theme.of(context).colorScheme.onPrimaryContainer,
        ),
       ),
      ],
     ),
    ),
   ),
  );
 }
}
```