**Форма № 24**

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
**Національний університет «Запорізька політехніка»**

Комп'ютерних наук і технологій
(повне найменування факультету)

Комп'ютерні системи та мережі
(повне найменування кафедри )

# Пояснювальна записка

до дипломного проєкту (роботи)
бакалаврський
(ступінь вищої освіти)

на тему: РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ ПЕРСОНАЛЬНОГО МЕДИЧНОГО ЩОДЕННИКА


Виконав(ла): студент(ка) 4 курсу,
групи КНТ-512

Спеціальності
123 Комп'ютерна інженерія
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерна інженерія
(назва освітньої програми (спеціалізації)

ДАНИЛЕВСЬКИЙ І.І.
(ПРІЗВИЩЕ та ініціали)

Керівник СКРУПСЬКИЙ С.Ю.
(ПРІЗВИЩЕ та ініціали)
Рецензент ХАРИТОНОВ О.Б.
(ПРІЗВИЩЕ та ініціали)


2026

**Форма № 25**

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
**Національний університет «Запорізька політехніка»**

Факультет комп'ютерних наук і технологій
Кафедра комп'ютерних систем та мереж
Ступінь вищої освіти бакалаврський
Спеціальність 123 комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) комп'ютерна інженерія
(назва освітньої програми (спеціалізації)

**ЗАТВЕРДЖУЮ**
**Завідувач кафедри** КУДЕРМЕТОВ Р.К.
_____
« » 2026 року

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

ДАНИЛЕВСЬКОГО Івана Ігоровича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка комп'ютерної системи персонального медичного щоденника
керівник проєкту (роботи) к.т.н., доцент, СКРУПСЬКИЙ С.Ю.,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від « 14 » квітня 2026 року № 160
2. Строк подання студентом проєкту (роботи) 01.06.2026 р.
3. Вихідні дані до проєкту (роботи) опис предметної області, технології розробки

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1) Аналіз технологій розробки персональних медичних щоденників;
2) Проєктування комп'ютерної системи персонального медичного щоденника;
3) Реалізація комп'ютерної системи персонального медичного щоденника;
4) Випробування комп'ютерної системи персонального медичного щоденника.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

| Розділ | ПРІЗВИЩЕ, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | прийняв виконане завдання |
| 1-5 | СКРУПСЬКИЙ С.Ю. | | |
| Нормоконтроль | ДЬЯЧУК Т.С. | | |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання «14» квітня 2026 року.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів дипломного проєкту (роботи) | Строк виконання етапів проєкту ( роботи ) | **Примітка** |
|---|---|---|---|
| 1 | Аналіз предметної області | до 20.04.2026 | |
| 2 | Вибір засобів побудови системи | до 24.04.2026 | |
| 3 | Проєктування архітектури системи | до 29.04.2026 | |
| 4 | Розробка варіантів використання | до 03.05.2026 | |
| 5 | Проєктування бази даних | до 08.05.2026 | |
| 6 | Розробка користувацького інтерфейсу | до 18.05.2026 | |
| 7 | Розробка модуля мережевої взаємодії з AI-моделлю | до 23.05.2026 | |
| 8 | Випробування комп’ютерної системи | до 26.05.2026 | |
| 9 | Оформлення пояснювальної записки | до 28.05.2026 | |
| 10 | Проходження нормоконтролю | до 29.05.2026 | |
| 11 | Перевірка на наявність академічного плагіату | до 30.05.2026 | |
| 12 | Проходження рецензування | до 01.06.2026 | |
| | | | |

**Студент(ка)** _____ Іван ДАНИЛЕВСЬКИЙ
( підпис ) (Ім’я ПРІЗВИЩЕ)

**Керівник проєкту (роботи)** _____ Степан СКРУПСЬКИЙ
( підпис ) (Ім’я ПРІЗВИЩЕ)

# РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 103 с., 2 табл., 25 рис., 1 дод., 25 джерел.

ПЕРСОНАЛЬНИЙ МЕДИЧНИЙ ЩОДЕННИК, ANDROID, ШТУЧНИЙ ІНТЕЛЕКТ, JAVA, SQLITE, JSON, HANDLER, OKHTTP

Предмет розробки – комп'ютерна система персонального медичного щоденника, реалізована у вигляді Android-застосунку з локальною базою даних та інтеграцією зовнішнього AI-сервісу для інтелектуальної обробки медичних показників. Мета роботи – підвищення ефективності самостійного моніторингу та первинного аналізу стану здоров'я людини шляхом довготривалого накопичення медичних даних та їх автоматизованої інтерпретації за запитом користувача. Новизна роботи – поєднання автономного зберігання персональних медичних показників у локальній реляційній базі даних мобільного пристрою з можливістю інтелектуального аналізу даних без розгортання власної серверної інфраструктури та без постійної залежності від хмарних сервісів. Практична цінність роботи полягає у можливості використання розробленої системи як персонального інструмента повсякденного контролю стану здоров'я.

У роботі виконано аналіз предметної області персональних медичних інформаційних систем та визначено обмеження, притаманні хмарним рішенням. Запропоновано архітектуру мобільного застосунку на платформі Android, реалізованого мовою. Для збереження даних застосовано вбудовану реляційну базу SQLite. Реалізовано механізми введення, редагування та перегляду медичних показників, локальне зберігання облікових даних користувача, захист доступу за допомогою пароля, а також інтеграцію із зовнішнім AI-сервісом через HTTP-запити з обробкою відповідей у форматі JSON. Перевірку працездатності розробленої системи проведено шляхом функціональних випробувань з використанням інструментів Android Studio.

# ЗМІСТ

# ВСТУП

Питання підтримки персонального здоров’я все більше зміщується в сторону цифрових рішень, орієнтованих на індивідуального користувача. Масове поширення смартфонів перетворило їх на універсальні інструменти збору, накопичення та первинного аналізу медичної інформації, яка раніше не зберігалася взагалі. За цих умов особливого значення набувають системи, що забезпечують довготривале ведення персональних медичних даних, їх структуроване зберігання та можливість осмислення накопичених показників у динаміці без постійного доступу до мережевих ресурсів.

Актуальність даної дипломної роботи визначається необхідністю створення комп’ютерної системи, здатної поєднувати локальне зберігання медичних показників із можливістю інтелектуальної обробки даних за запитом користувача. Сучасні медичні застосунки часто зосереджені виключно на фіксації даних без глибшого аналізу, або повністю залежать від хмарних сервісів, що підвищує вимоги до підключення та створює ризики для конфіденційності. Запропонований підхід орієнтований на мінімізацію зовнішніх залежностей і збереження контролю над персональними даними.

Метою роботи є підвищення ефективності самостійного діагностування здоров’я людини та виялвення проблем на ранніх стадіях. Об’єктом розробки є комп’ютерна система персонального медичного щоденника, реалізована у вигляді Android-застосунку з локальною базою даних та інтеграцією зовнішнього AI-сервісу. Створення системи забезпечує користувачу цілісне уявлення про власний фізіологічний стан на основі накопичених даних і автоматизованого аналізу.

Для досягнення поставленої мети у роботі вирішено такі завдання:

- проаналізовано особливості предметної області персональних медичних застосунків;
- визначено вимоги до структури даних, інтерфейсу та безпеки зберігання інформації;

- спроєктовано архітектуру мобільної системи з урахуванням автономної роботи;
- реалізовано засоби введення, збереження та перегляду медичних показників;
- забезпечено інтеграцію з віддаленою AI-моделлю для формування текстових інтерпретацій результатів вимірювань;
- проведено комплексні випробування розробленого рішення.

В роботі ключова логіка зосереджена на стороні мобільного клієнта, а зовнішній AI-сервіс виконує роль аналітичного інструмента, що залучається лише за потреби. Така організація системи дозволяє зменшити навантаження на мережеву взаємодію та забезпечити збереження історичних даних у режимі офлайн.

Новизна роботи полягає у поєднанні локальної реляційної бази даних та інтелектуального аналізу показників без використання власної серверної інфраструктури. Практичне значення отриманих результатів – це можливість використання розробленої системи для індивідуального моніторингу стану здоров’я та підтримки прийняття рішень на побутовому рівні.

# 1 АНАЛІЗ ТЕХНОЛОГІЙ РОЗРОБКИ ПЕРСОНАЛЬНИХ МЕДИЧНИХ ЩОДЕННИКІВ

## 1.1 Функціональні вимоги до системи

Актуальність персональних медичних щоденників істотно зросла в останні роки у зв’язку зі збільшенням частки хронічних неінфекційних захворювань, старінням населення та переходом медицини до пацієнтоорієнтованої моделі, у якій людина самостійно бере участь у моніторингу власного стану здоров’я [1]. Регулярне самостійне вимірювання артеріального тиску, пульсу та рівня глюкози в крові рекомендоване міжнародними медичними організаціями як базовий інструмент раннього виявлення відхилень та контролю стану пацієнтів поза межами лікувальних закладів [2]. У цьому контексті персональні медичні щоденники виступають засобом систематизації таких даних і підвищення усвідомленості користувача щодо власного здоров’я.

На сьогодні найбільш поширеними є веб-орієнтовані системи електронного здоров’я та хмарні сервіси моніторингу показників [3]. Вони забезпечують централізоване зберігання даних і можливість доступу з різних пристроїв, однак мають низку суттєвих недоліків. По-перше, веб-системи критично залежать від стабільного підключення до мережі Інтернет, що унеможливлює фіксацію показників у будь-який момент часу. По-друге, передавання персональних медичних даних на віддалені сервери підвищує ризики витоку інформації та викликає занепокоєння щодо конфіденційності. По-третє, веб-інтерфейси зазвичай менш зручні для швидкого щоденного введення даних, особливо на мобільних пристроях із сенсорним керуванням.

Мобільний застосунок усуває обмеження веб-систем, оскільки смартфон або планшет є персональним пристроєм, який постійно знаходиться поруч із користувачем. Це дозволяє вносити результати вимірювань одразу після їх отримання без необхідності запуску браузера чи переходу на веб-сайт. Крім того, мобільні пристрої забезпечують інтуїтивно зрозумілий сенсорний інтерфейс,

підтримку портретної та альбомної орієнтацій екрана, а також можливість автономної роботи без доступу до мережі. Завдяки цьому мобільний персональний медичний щоденник є більш придатним для щоденного використання та сприяє формуванню стабільної звички регулярного самоконтролю показників здоров’я.

Мобільні застосунки персональних медичних щоденників часто обмежуються збереженням показників та їх візуалізацією у вигляді списків або графіків. Як показує аналіз сучасних мобільних health-застосунків, інтелектуальна інтерпретація даних або відсутня, або реалізована у вигляді жорстко закодованих правил без урахування індивідуальних особливостей користувача та його історії вимірювань [4]. Крім того, значна частина таких застосунків орієнтована виключно на смартфони і не враховує сценарії використання на планшетах, зокрема в альбомній орієнтації, що обмежує зручність перегляду великих обсягів історичних даних.

Перевагою мобільної комп’ютерної системи, реалізованої як Android-застосунок із локальним зберіганням даних, є можливість поєднання автономної роботи, високого рівня приватності та використання апаратних і програмних можливостей мобільного пристрою. Локальна база даних дозволяє зберігати історичні вимірювання без постійного доступу до мережі. Адаптивний інтерфейс забезпечує комфортну роботу як на смартфонах, так і на планшетах у портретній та альбомній орієнтаціях. Інтеграція зовнішніх AI-сервісів відкриває можливість реалізації довідкової аналітики та інтерпретації накопичених даних без необхідності розгортання власної серверної інфраструктури [5].

З урахуванням аналізу існуючих аналогів та виявлених недоліків формулюються такі функціональні вимоги до комп’ютерної системи персонального медичного щоденника:

- реєстрація користувача із створенням персонального профілю, що включає логін, захищений пароль, вік, стать та масу тіла;

- автентифікація користувача при кожному запуску застосунку з метою обмеження доступу сторонніх осіб до медичних даних;

- введення користувачем показників артеріального тиску, пульсу в стані спокою та рівня глюкози в крові з автоматичним фіксуванням дати і часу вимірювання;

- локальне зберігання усіх вимірювань у базі даних з можливістю накопичення історії за тривалий період;

- перегляд історичних даних у вигляді списків із прокручуванням та фільтрацією за обраними часовими діапазонами (день, тиждень, місяць, рік);

- підтримка коректної роботи інтерфейсу як у портретній, так і в альбомній орієнтації екрану, з урахуванням використання застосунку на смартфонах і планшетах;

- інтеграція з зовнішнім AI-асистентом для отримання довідкових інтерпретацій і текстових рекомендацій на основі історичних даних користувача;

- збереження отриманих AI-інтерпретацій у базі даних для подальшого перегляду та аналізу;

- налаштування користувачем нагадувань про необхідність проведення вимірювань із використанням системи сповіщень Android;

- забезпечення конфіденційності персональних і медичних даних шляхом локального зберігання та шифрування чутливих параметрів, зокрема ключа доступу до AI-сервісу.

Реалізація зазначених функціональних вимог дозволяє створити мобільну комп’ютерну систему, що усуває ключові недоліки наявних веб-та мобільних аналогів і забезпечує зручний, безпечний та інформативний інструмент персонального медичного самоконтролю.

## 1.2 Нефункціональні вимоги до системи

Нефункціональні вимоги визначають якісні характеристики системи персонального медичного щоденника, що забезпечують її придатність до

практичного використання, стабільність, безпеку та зручність для кінцевого користувача. Для медичних інформаційних систем ці вимоги мають критичне значення, оскільки система працює з персональними та потенційно чутливими даними, а також використовується регулярно в побутових умовах [6].

Однією з ключових нефункціональних вимог є зручність використання (usability). Інтерфейс мобільного застосунку повинен бути інтуїтивно зрозумілим, логічно структурованим та не вимагати від користувача спеціальних технічних знань. Усі основні операції - введення вимірів, перегляд історії, отримання інтерпретацій та налаштування нагадувань - мають виконуватися з мінімальною кількістю дій. Система повинна коректно працювати як на смартфонах, так і на планшетах, з повною підтримкою портретної та альбомної орієнтації екрана, що відповідає сучасним рекомендаціям щодо проєктування мобільних інтерфейсів [7].

Вимога до продуктивності полягає у забезпеченні швидкої реакції застосунку на дії користувача. Збереження вимірювань у локальній базі даних SQLite, формування списків записів та фільтрація даних за часовими інтервалами повинні виконуватися без помітних затримок. Мережева взаємодія з AI-сервісом має реалізовуватися в окремому потоці виконання, щоб не блокувати головний потік інтерфейсу та не погіршувати користувацький досвід.

Важливими є продуктивність і невибагливість до апаратних ресурсів. Мобіний застосунок має коректно функціонувати на широкому спектрі мобільних пристроїв - смартфонах і планшетах різних цінових категорій, включаючи пристрої з обмеженим обсягом оперативної пам’яті, середньою продуктивністю процесора та без спеціалізованих сенсорів. Час запуску застосунку, відкриття екранів та виконання операцій з базою даних має залишатися мінімальним навіть при накопиченні значного обсягу історичних даних.

Надійність і стійкість до збоїв є важливою нефункціональною вимогою. Система повинна зберігати працездатність за відсутності підключення до мережі Інтернет, оскільки введення медичних даних має бути можливим у будь-яких умовах. Усі дані користувача повинні зберігатися локально та не втрачатися у разі

аварійного завершення роботи застосунку або перезапуску пристрою. Помилки мережевої взаємодії мають оброблятися контрольовано з інформуванням користувача у зрозумілій формі [8].

Окрему групу нефункціональних вимог становлять вимоги до безпеки та конфіденційності. Персональні дані користувача, зокрема логін, хеш пароля, медичні показники та ключ доступу до AI-сервісу, повинні бути захищені від несанкціонованого доступу. Паролі мають зберігатися виключно у вигляді криптографічних хешів, а чутливі дані - у зашифрованому вигляді відповідно до загальноприйнятих рекомендацій з інформаційної безпеки. Доступ до функціональності застосунку повинен бути обмежений механізмом автентифікації [9].

Вимоги до сумісності та переносимості передбачають коректну роботу застосунку на різних версіях операційної системи Android та на пристроях із різними апаратними характеристиками та розмірами екранів. Архітектура системи повинна забезпечувати можливість подальшого розширення функціональності без суттєвих змін у структурі бази даних або логіці застосунку. Це відповідає сучасним підходам до розробки мобільних інформаційних систем з орієнтацією на довгострокову підтримку та масштабованість [10].

Важливою вимогою є можливість повноцінної офлайн-роботи. Основні функції системи - введення медичних показників, збереження даних, перегляд історії вимірювань та доступ до раніше отриманих інтерпретацій - повинні бути доступні без постійного підключення до мережі Інтернет. Це особливо актуально для користувачів у дорозі, сільській місцевості або в умовах нестабільного з’єднання. Мережева взаємодія з AI-асистентом розглядається як додаткова функція, яка активується лише за наявності підключення, що підвищує надійність і автономність системи.

Таким чином, дотримання нефункціональних вимог щодо зручності, продуктивності, надійності, безпеки та сумісності є необхідною умовою створення ефективної комп’ютерної системи персонального медичного щоденника, придатної до щоденного використання в реальних умовах.

## 1.3 Вибір засобів побудови системи

При проєктуванні мобільної комп’ютерної системи персонального медичного щоденника одним із перших технічних рішень є вибір підходу до розробки: нативна чи кросплатформна розробка. Нативна розробка передбачає використання інструментів і мов програмування, рекомендованих для конкретної платформи (Java/Kotlin для Android), що дозволяє найтісніше взаємодіяти з апаратними та програмними можливостями пристрою. Кросплатформні фреймворки (наприклад, Flutter, React Native) дозволяють створювати застосунки під різні платформи з однієї кодової бази, але при цьому можуть мати обмеження щодо доступу до специфічних API, продуктивності та адаптації під дизайн кожної платформи [11].

Для нашої системи, орієнтованої на Android, нативний підхід є більш доцільним з кількох причин. Медичний щоденник вимагає тісної інтеграції з локальною базою даних, системою автентифікації, управління нотифікаціями та адаптивним інтерфейсом, що найкраще реалізується через нативні SDK та інструменти [12]. Інтеграція з модулем мережевої взаємодії та AI-асистентом потребує точного контролю потоків виконання, роботи з HTTP і обробки результатів у UI. Цього легше досягти в нативному Android із використанням існуючих бібліотек і механізмів платформи. Крім того, нативні рішення забезпечують вищу продуктивність та UX, що є критично важливим для щоденних інтерактивних дій користувача.

Наступним кроком вибору є мова програмування. Android підтримує Java та Kotlin як основні мови розробки. Kotlin, будучи сучаснішою мовою з кращою безпекою null-посилань і консьюмерськими можливостями, є популярним вибором у нових проєктах [13]. Втім, Java залишається домінуючою для великої кількості існуючих бібліотек, прикладів, документації та інструментальних рішень, а також є універсальною мовою для інтеграції з низькорівневими API Android, що значно спрощує реалізацію локального управління базою даних,

Thread/Handler, HTTP і синхронізації з UI. Java має переваги насамперед через більшу кількість прикладів коду, стабільність і широке покриття навчальними матеріалами, що сприяє зниженню ризиків реалізації та прискоренню розробки [14]. А ще дуже важливо, що ми на кафедрі вивчали Java як основну мову, тому логічно використати набуті знання на практиці.

Ще одним критичним аспектом є багатопоточність. Будь-яка сучасна мобільна система повинна виконувати довготривалі або блокуючі операції (наприклад, мережеві запити або обробку великих обсягів даних) поза головним потоком UI, щоб не викликати «заморожування» інтерфейсу. Android пропонує кілька засобів багатопоточності, включно з Thread, Handler, AsyncTask, Executors, HandlerThread, Coroutine (у Kotlin) тощо [15]. У контексті нашої Java-реалізації доцільно використовувати Thread для виконання окремих задач у фоновому потоці та Handler для передачі результатів обробки у головний потік UI, оскільки це відповідає базовому механізму Android для обробки повідомлень і дозволяє чітко розділити фонову логіку та оновлення інтерфейсу. Такий підхід є менш складним у порівнянні з більш абстрактними фреймворками, але водночас надає повний контроль над порядком і контекстом виконання [16]. Він також добре узгоджується зі структурою застосунку, де запити до AI та обробка великих наборів даних є окремими задачами.

Окрім механізмів багатопоточності, важливою є підтримка обробки повернення з мережі у UI без блокування, що реалізується через Handler або аналогічні механізми передачі повідомлень між потоками. Це дозволяє реалізувати модуль мережевої взаємодії, що працює у фоновому потоці, обробляє відповіді від AI і потім коректно оновлює UI. Такий підхід є простим, прогнозованим і відповідає рекомендаціям офіційної документації Android щодо роботи з потоками на Java [17].

Таким чином, поєднання нативної розробки Android, Java як мови реалізації та Thread + Handler як механізмів багатопоточності забезпечує баланс між продуктивністю, контролем, зручністю реалізації та відповідністю сучасним практикам мобільної розробки. Цей вибір є обґрунтованим з точки зору технічної

доцільності, якості, надійності та часу розробки.

### 1.4 Огляд API AI-асистентів

Сучасні AI-асистенти надають доступ до моделей штучного інтелекту через програмні інтерфейси прикладного програмування (API), які дозволяють клієнтським застосункам формувати запити, передавати їх на сервер з моделлю та отримувати структуровану текстову відповідь. Для мобільних застосунків API AI-асистентів є ключовим технічним засобом реалізації інтелектуальної інтерпретації даних без необхідності розгортання власної серверної інфраструктури або навчання моделей локально.

З технічної точки зору взаємодія з AI-асистентом зазвичай реалізується через HTTP(S)-запити у форматі JSON. Клієнт формує текстовий запит, який містить вхідні дані (наприклад, медичні показники користувача за певний період) та інструкцію для моделі щодо характеру відповіді. Запит надсилається на відповідний endpoint API, після чого сервер повертає JSON-відповідь, що містить згенерований текст. Такий підхід є універсальним і добре інтегрується з Android-застосунками, реалізованими мовою Java, через стандартні засоби роботи з мережею [18].

Більшість AI-платформ використовують механізм ключа доступу (API key) для автентифікації запитів. API-ключ є унікальним ідентифікатором користувача або застосунку, який передається в HTTP-заголовку запиту та дозволяє сервісу контролювати доступ, обмежувати навантаження та вести облік використання. З практичної точки зору користувач повинен самостійно зареєструватися на платформі AI-сервісу, згенерувати персональний ключ і ввести його в мобільний застосунок. Застосунок, у свою чергу, зберігає цей ключ локально та використовує його для подальших запитів до AI [19].

Серед найбільш відомих AI-платформ, які надають API для роботи з

мовними моделями, можна виділити OpenAI, Google (Gemini), Microsoft Azure AI. Більшість з них пропонують потужні моделі загального призначення, однак у багатьох випадках доступ до них є обмеженим платними тарифами або потребує складної конфігурації облікового запису. Окрему увагу заслуговує платформа Hugging Face, яка є відкритим хабом моделей машинного навчання та штучного інтелекту. Hugging Face надає доступ до великої кількості мовних моделей, зокрема моделей класу instruction-tuned та chat-oriented, які придатні для генерації пояснень, узагальнень і довідкових інтерпретацій [20]. Важливою перевагою цієї платформи є наявність безкоштовного рівня доступу, який дозволяє виконувати запити до моделей без прив'язки платіжних даних, що є принципово важливим для бакалаврської роботи.

З технічної точки зору Hugging Face API є простим у використанні. Запит формується у вигляді JSON-об'єкта, який містить текстове поле з вхідними даними користувача та інструкцією для моделі. Відповідь повертається у структурованому вигляді, що дозволяє легко виконати подальший парсинг JSON та відобразити отриманий текст у користувацькому інтерфейсі Android-застосунку. Платформа також не накладає жорстких обмежень на тип клієнта, що дозволяє використовувати її безпосередньо з мобільного застосунку без проміжного серверного шару [21].

З точки зору застосування у персональному медичному щоденнику, AI-моделі Hugging Face можуть використовуватися для довідкової інтерпретації медичних показників, порівняння їх з загальноприйнятими референтними діапазонами, аналізу динаміки значень у часі та формування текстових рекомендацій загального характеру. При цьому важливо, що відповідь моделі не розглядається як медичний діагноз або лікувальна рекомендація, а має виключно інформаційний характер. Ми використаємо модель meta-llama/Llama-3.1-8B-Instruct, яка підходить для інтерпретації медичних даних.

Таким чином, використання API AI-асистента дозволяє значно розширити функціональність персонального медичного щоденника без ускладнення архітектури системи. Вибір платформи Hugging Face є обґрунтованим з огляду на

відкритість, безкоштовний доступ, простоту інтеграції та наявність моделей, придатних для текстових інтерпретацій медичних даних.

### 1.5 Вибір системи керування базами даних

Для реалізації персонального медичного щоденника ключовим є вибір типу системи керування базами даних, оскільки характер даних (персональні, чутливі, відносно невеликі за обсягом) і сценарії використання (індивідуальне ведення щоденника на одному пристрої) суттєво відрізняються від класичних корпоративних або веб-систем.

Серверні системе керування базами даних (PostgreSQL, MySQL, MS SQL Server, Oracle) орієнтовані на багатокористувацьку роботу, централізоване зберігання даних, високу паралельність доступу та масштабування. Вони передбачають наявність окремого сервера, постійного мережевого з’єднання, адміністрування, резервного копіювання та налаштування доступів. Для персонального медичного щоденника такий підхід є надмірним: користувачеві довелося б постійно залежати від мережі, довіряти персональні медичні дані зовнішньому серверу та нести додаткові витрати на інфраструктуру. Крім того, ускладнюється забезпечення конфіденційності та відповідності вимогам захисту персональних даних.

Кишенькові (вбудовані) – призначені для роботи безпосередньо на пристрої користувача як частина застосунку. Вони не потребують окремого серверного програмного забезпечення, працюють локально, мають мінімальні вимоги до ресурсів і забезпечують високу швидкість доступу до даних за рахунок відсутності мережевих затримок. Такий підхід є оптимальним для мобільних застосунків, орієнтованих на офлайн-роботу та автономність.

Серед кишенькових СКБД найбільш поширеними є SQLite [22], Realm [23] та ObjectBox [24]. Порівняльний аналіз цих СКБД наведено в таблиці 2.1.

Таблиця 2.1 – Порівняльний аналіз кишенькових СКБД

| Критерій | SQLite | Realm | ObjectBox |
|---|---|---|---|
| Тип | Реляційна | Об’єктна | Об’єктна |
| Наявність у складі Android | Вбудована | Зовнішня бібліотека | Зовнішня бібліотека |
| Необхідність додаткового встановлення | Не потребує | Потрібно підключати SDK | Потрібно підключати SDK |
| Офлайн-робота | Повна | Повна | Повна |
| Споживання ресурсів | Мінімальне | Середнє | Середнє |
| Контроль і переносимість даних | Повний, стандартний SQL | Нестандартний формат | Нестандартний формат |
| Стабільність і зрілість | Дуже висока | Висока | Середня |
| Інтеграція з Android SDK | Нативна | Через API | Через API |
| Ліцензійні ризики | Відсутні | Залежність від вендора | Залежність від вендора |

Realm і ObjectBox пропонують зручну об’єктно-орієнтовану модель даних і високу продуктивність, однак вони є сторонніми бібліотеками, що збільшує розмір застосунку, ускладнює міграції та створює залежність від конкретного вендора. Крім того, частина їхніх можливостей обмежена у безкоштовних версіях або змінюється залежно від ліцензійної політики.

SQLite є вбудованою реляційною СКБД, яка постачається разом з операційною системою Android і не потребує жодного додаткового встановлення чи налаштування. База даних створюється автоматично під час першого запуску застосунку і зберігається у внутрішньому сховищі пристрою. Для користувача це

означає максимально простий сценарій: достатньо завантажити застосунок, запустити його і одразу почати роботу без будь-яких додаткових дій.

Перевагами SQLite для персонального медичного щоденника є мала ресурсомісткість, надійність, підтримка транзакцій, стандартний SQL-синтаксис та повна інтеграція з Android через SQLiteOpenHelper. Це дозволяє реалізувати структуроване зберігання даних користувача, медичних вимірювань, історії взаємодії з AI-асистентом та налаштувань без ризику втрати даних при відсутності мережі. Локальне зберігання також підвищує рівень конфіденційності, оскільки персональні медичні дані не передаються на зовнішні сервери без явної необхідності. З урахуванням автономного характеру системи, вимог до офлайн-роботи, невибагливості до ресурсів та простоти використання для кінцевого користувача, доцільним є вибір кишенькової СКБД SQLite як основного засобу зберігання даних у персональному медичному щоденнику.

## 1.6 Висновки розділу

У першому розділі було виконано комплексний аналіз предметної області та обґрунтовано архітектурні і технологічні рішення, необхідні для розробки комп’ютерної системи персонального медичного щоденника. Показано актуальність таких систем в умовах зростання обсягів персональних медичних даних і потреби користувачів у їх систематизації, збереженні та інтерпретації без постійної залежності від зовнішніх сервісів і стаціонарних пристроїв.

На основі порівняння з існуючими аналогами, зокрема веборієнтованими рішеннями, виявлено їх ключові недоліки: залежність від браузера і комп’ютера, обмежена мобільність, складність використання в повсякденних умовах та недостатня адаптованість до сценаріїв регулярного введення медичних показників. Це дозволило сформулювати функціональні вимоги до системи, орієнтовані на мобільне використання, автономну роботу, підтримку смартфонів і

планшетів у портретній та альбомній орієнтаціях, а також зручний перегляд і аналіз медичних даних у часових діапазонах.

У межах визначення нефункціональних вимог акцент зроблено на продуктивності, надійності, безпеці персональних даних, невибагливості до апаратних ресурсів та можливості повноцінної офлайн-роботи. Це є критично важливим для персонального медичного щоденника, який має бути доступним користувачу незалежно від наявності мережевого з’єднання та типу мобільного пристрою.

Порівняльний аналіз засобів побудови системи підтвердив доцільність вибору нативної Android-розробки як найбільш контрольованого і стабільного підходу для такого проєкту. Обґрунтовано використання мови програмування Java, а також класичних механізмів багатопоточності Thread і Handler, що забезпечують прозоре розділення фонових операцій та оновлення користувацького інтерфейсу.

Окрему увагу приділено огляду API AI-асистентів, що дозволяють виконувати інтерпретацію медичних показників. Обґрунтовано вибір платформи Hugging Face як безкоштовного та гнучкого інструменту для інтеграції AI-функціональності з можливістю використання персонального ключа доступу користувача.

У результаті аналізу систем керування базами даних доведено, що для персонального медичного щоденника ефективним є використання кишенькової СКБД SQLite, яка забезпечує повну автономність, відсутність необхідності додаткового встановлення, високу стабільність та повну інтеграцію з платформою Android.

# 2 ПРОЄКТУВАННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ ПЕРСОНАЛЬНОГО МЕДИЧНОГО ЩОДЕННИКА

## 2.1 Проєктування архітектури системи

Архітектура системи будується за принципом незалежних клієнтів, які взаємодіють з зовнішнім AI-сервісом, не маючи між собою жодних прямих зв’язків. Кожен екземпляр мобільного застосунку є автономним клієнтом, що зберігає персональні дані користувача локально та виконує мережеві запити до сервера AI-асистента лише за потреби інтерпретації медичних показників.

З боку клієнта система реалізується у вигляді нативного Android-застосунку. Вся критично важлива інформація, включно з обліковими даними користувача, історією вимірювань, результатами AI-інтерпретацій та налаштуваннями, зберігається у локальній базі даних SQLite. Такий підхід забезпечує повну автономність роботи застосунку, можливість використання без доступу до мережі Інтернет та мінімальну залежність від зовнішньої інфраструктури.

Серверна складова в класичному розумінні (власний бекенд) у системі відсутня. Роль сервера виконує сторонній AI-сервіс Hugging Face, який надає API для доступу до мовних моделей. Взаємодія з ним здійснюється за клієнт-серверною моделлю через HTTPS-запити. Кожен клієнт використовує власний персональний ключ доступу до API, що зберігається в зашифрованому вигляді у локальній базі даних. Це дозволяє масштабувати систему без обмежень на кількість користувачів, оскільки клієнти не створюють навантаження на єдиний централізований сервер розробника.

Логічна архітектура мобільного застосунку базується на одному Activity, яке виступає контейнером для набору Fragment, кожен з яких відповідає окремому екрану системи. Такий підхід відповідає сучасним рекомендаціям Android-розробки та спрощує керування життєвим циклом інтерфейсу, а також підтримку різних орієнтацій екрана. MainActivity не містить бізнес-логіки і відповідає

виключно за керування життєвим циклом застосунку, ініціалізацію доступу до локальної бази даних SQLite через SQLiteOpenHelper, збереження інформації про поточну сесію користувача (ідентифікатор користувача та стан автентифікації), а також за навігацію між фрагментами за допомогою FragmentManager. Такий підхід спрощує керування станом інтерфейсу, зменшує кількість помилок, пов’язаних із життєвим циклом Activity, і полегшує підтримку коректної роботи застосунку при зміні орієнтації екрана.

WelcomeFragment є стартовим екраном застосунку та відображається одразу після запуску. Він призначений для ознайомлення користувача з призначенням персонального медичного щоденника та містить медичний дисклеймер щодо довідкового характеру інформації. На цьому екрані користувач може обрати подальший сценарій роботи, натиснувши кнопку входу до вже існуючого облікового запису або кнопку переходу до реєстрації нового користувача. Даний фрагмент не взаємодіє з базою даних і виконує виключно навігаційну та інформаційну функцію.

RegistrationFragment забезпечує створення нового облікового запису користувача. У межах цього фрагмента користувач вводить логін, пароль, вік, стать і масу тіла, а також персональний ключ доступу до AI API сервісу Hugging Face. Для зручності користувача поле введення ключа супроводжується гіперпосиланням на офіційну сторінку реєстрації сервісу, де користувач може отримати власний API-ключ, скопіювати його та вставити в застосунок. Під час реєстрації пароль користувача зберігається у вигляді хешу, а ключ доступу до AI сервісу шифрується з використанням симетричного алгоритму AES-128, де ключ шифрування формується на основі введеного пароля. Після успішної реєстрації дані користувача зберігаються в локальній базі даних SQLite, і користувач переходить до головного меню застосунку.

LoginFragment призначений для автентифікації вже зареєстрованого користувача. У цьому фрагменті користувач вводить логін і пароль, які перевіряються шляхом порівняння хешу введеного пароля з хешем, що зберігається в базі даних. У разі успішної перевірки здійснюється дешифрування

збереженого ключа доступу до AI API, після чого користувачу надається доступ до функціональних можливостей системи. Даний фрагмент забезпечує базовий рівень захисту персональних медичних даних від несанкціонованого доступу.

MainMenuFragment є центральним навігаційним елементом застосунку після успішної автентифікації користувача. Він надає доступ до основних сценаріїв роботи системи, зокрема введення нових медичних вимірювань, перегляду історичних даних та отримання інтерпретацій від AI-асистента. Фрагмент не виконує обчислювальної логіки, а використовується як точка переходу до інших функціональних екранів.

MeasurementInputFragment призначений для введення нових медичних показників користувача, таких як систолічний і діастолічний артеріальний тиск, пульс у стані спокою та, за потреби, рівень глюкози в крові. Дата та час вимірювання формуються автоматично на основі системного часу мобільного пристрою, що виключає можливість помилок ручного введення. Після перевірки коректності значень дані зберігаються в таблиці Measurements локальної бази даних SQLite з прив’язкою до відповідного користувача.

MeasurementsListFragment забезпечує перегляд історичних даних вимірювань у вигляді прокручуваного списку. Користувач може обрати часовий діапазон, за який необхідно відобразити дані, наприклад день, тиждень, місяць або довільний період. Завантаження даних з бази даних здійснюється в окремому потоці, а оновлення інтерфейсу відбувається через механізм Handler, що дозволяє зберегти відзивчивість користувацького інтерфейсу навіть при роботі з великими обсягами інформації.

InterpretationFragment реалізує функціональність взаємодії з AI-асистентом. У цьому фрагменті користувач обирає часовий інтервал, за який необхідно проаналізувати медичні показники, після чого застосунок формує текстовий запит на основі агрегованих даних із таблиці бази даних і надсилає його до AI-сервісу через мережевий інтерфейс. Отримана відповідь відображається на екрані користувача та зберігається в таблиці Interpretations для подальшого перегляду.

SettingsFragment реалізує налаштування сповіщень з нагадуванням про

необхідність виконати виміри тиску, пульсу та рівня глюкози.

Мережева взаємодія з AI-сервісом реалізується в окремому потоці виконання з використанням класу Thread. Формування HTTP-запиту, серіалізація даних у формат JSON та обробка відповіді виконуються у фоновому потоці, що запобігає блокуванню головного UI-потоку. Для передачі результатів виконання у графічний інтерфейс застосовується механізм Handler, який дозволяє безпечно оновлювати стан елементів UI після завершення мережевої операції.

Заплановані нагадування про необхідність виконання вимірювань реалізуються з використанням стандартних засобів Android для роботи з нотифікаціями. Параметри нагадувань зберігаються в окремій таблиці бази даних і враховуються під час ініціалізації відповідних системних подій.

Архітектурні рішення стосовно захисту інформації. Система не використовує власний сервер для зберігання персональних та медичних даних. Усі чутливі відомості: облікові дані користувача, історія вимірювань та результати AI-інтерпретацій зберігаються виключно локально на мобільному пристрої в базі даних SQLite. Такий підхід суттєво зменшує ризики витоку інформації через мережеві атаки або несанкціонований доступ до віддалених сховищ. Механізм автентифікації користувача реалізований із дотриманням базових принципів безпеки. Пароль користувача ніколи не зберігається у відкритому вигляді - у базі даних фіксується лише його криптографічний хеш. Це унеможливлює відновлення початкового пароля навіть у разі фізичного доступу до файлу бази даних. Додатково застосунок реалізує блокування доступу до функціональних можливостей без успішного введення коректних облікових даних, що захищає медичну інформацію від сторонніх осіб, які можуть мати доступ до пристрою. Особлива увага приділяється захисту ключа доступу до AI API. Персональний ключ Hugging Face зберігається в зашифрованому вигляді з використанням симетричного алгоритму AES-128, де криптографічний ключ формується на основі пароля користувача. Це означає, що навіть у разі компрометації локальної бази даних зловмисник не зможе використати API-ключ без знання пароля власника облікового запису. Дешифрування ключа відбувається

лише в оперативній пам’яті після успішної автентифікації користувача. Додатковим елементом захисту є розмежування відповідальності між компонентами системи. Мережева взаємодія з AI-сервісом виконується в окремому потоці, а передача даних здійснюється лише у вигляді агрегованих та текстово узагальнених показників за обраний період, без передачі ідентифікаційних даних користувача.

## 2.2 Аналіз системних вимог

Системні вимоги до комп’ютерної системи персонального медичного щоденника визначають мінімальні апаратні та програмні характеристики мобільного пристрою, за яких забезпечується коректна і стабільна робота. Апаратні вимоги наведено в табл. 2.1.

Застосунок не висуває вимог до наявності спеціалізованих апаратних сенсорів або медичних пристроїв, оскільки всі показники вводяться користувачем вручну. Виміри користувач має виконати за допомогою домашнього тонометра та глюкометра будь-якого виробника та моделі. Робота системи можлива і без постійного доступу до мережі, за винятком функції AI-інтерпретації.

Мінімальною підтримуваною версією операційної системи обрано Android 10 (API Level 29). Такий вибір забезпечує баланс між сучасними механізмами безпеки платформи та рівнем сумісності з реальними пристроями користувачів. Android 10 підтримує актуальні API для роботи з фрагментами, багатопоточністю, локальними базами даних SQLite та мережевими з’єднаннями через HTTPS. Для коректної роботи застосунку не потребується встановлення жодного додаткового програмного забезпечення з боку користувача. SQLite є вбудованою СКБД платформи Android і не вимагає окремої інсталяції. Мережеві запити до AI-сервісу виконуються через стандартні бібліотеки Java та Android SDK.

Таким чином, сформульовані системні вимоги забезпечують можливість

використання персонального медичного щоденника на широкому спектрі сучасних мобільних пристроїв від бюджетних моделей до флагманських.

Таблиця 2.1 – Апаратні вимоги до системи

| Параметр | Мінімальна вимога | Обгрунтування |
|---|---|---|
| Тип пристрою | Смартфон або планшет | Система орієнтована на персональне використання та мобільність. Стаціонарний комп’ютер чи ноутуб не потрібні |
| SOC | ARMv8 (64-bit), 4 ядра, ≥ 1.5 ГГц, наприклад, Snapdpragon 715 | Забезпечує достатню продуктивність для роботи UI, SQLite та мережевих запитів |
| Оперативна пам’ять | ≥ 4 ГБ RAM | Цього вистачить для роботи одного Activity з кількома Fragment та фоновими потоками в такий спосіб, щоб ОС не вивільняла пам’ять шляхом знищення зайвих фрагментів |
| Вбудована пам’ять | ≥ 200 МБ вільного простору | Зберігання застосунку, локальної SQLite та історичних даних |
| Діагональ екрану | ≥ 6" (смартфон), ≥ 7" (планшет) | Забезпечує зручне відображення великих списків і форм введення |
| Роздільна здатність екрану | ≥ 1280×720 | Коректна верстка у портретній та альбомній орієнтаціях |
| Сенсорний екран | Обов’язково | Це основний спосіб взаємодії користувача із системою |
| Мережеве з’єднання | Wi-Fi або LTE/5G | Потрібне для отримання AI-інтерпретацій. Не обов’язково |

## 2.3 Проєктування варіантів використання

Персональний медичний щоденник орієнтований на повсякденне використання користувачем для збереження, систематизації та аналізу власних медичних показників. Всі варіанти використання описані з точки зору користувача (Actor – User), який взаємодіє із застосунком через графічний інтерфейс на мобільному пристрої.

Користувач має можливість відкрити застосунок, після чого з’являється екран привітання (WelcomeFragment) із коротким дисклеймером. На цьому екрані користувач може або перейти до реєстрації нового облікового запису, або увійти до існуючого. Цей варіант використання включає наступні дії: перегляд інформаційного повідомлення про конфіденційність та безпеку даних, натискання кнопки "Sign up " або "Login", вибір сценарію авторизації.

При виборі реєстрації (RegistrationFragment) користувач створює обліковий запис. Система надає форми для введення логіну та пароля, а також додаткових даних профілю: вік, стать, вага. Крім того, користувач вставляє персональний ключ доступу до AI-сервісу (Hugging Face) із буфера обміну. Поле для ключа містить гіперпосилання на сторінку реєстрації сервісу. Верифікація ключа та пароля здійснюється на стороні пристрою з використанням хешування пароля та шифрування ключа AES-128. Після успішного створення облікового запису дані зберігаються в SQLite.

У випадку входу в систему (LoginFragment) користувач вводить логін та пароль. Система перевіряє відповідність хешованого пароля, а у разі успіху дешифрує ключ API для подальших мережевих операцій. Після успішної автентифікації користувач потрапляє до головного меню.

Головне меню (MainMenuFragment) надає три ключові сценарії використання: введення нових медичних вимірювань, перегляд історичних даних, отримання інтерпретації від AI. Користувач обирає відповідний пункт, після чого відкривається відповідний фрагмент.

При введенні нових даних (MeasurementInputFragment) користувач заповнює форму для фіксації систолічного та діастолічного тиску, пульсу у спокої та рівня глюкози. Дані автоматично доповнюються поточним таймштампом і зберігаються в локальній базі SQLite. Додатково система може перевіряти формат введених значень та попереджати про потенційно критичні показники.

Сценарій перегляду історичних даних (MeasurementsListFragment) дозволяє користувачу обирати часовий діапазон: день, тиждень, місяць, рік або довільний період. Дані виводяться у вигляді списку з прокруткою, де кожен запис містить дату, час, тиск, пульс і рівень глюкози.

У разі вибору AI-інтерпретації (InterpretationFragment) користувач може надіслати обраний діапазон історичних даних на сервер Hugging Face. Система формує запит, відправляє його в AI модель, а результати отримує асинхронно. AI надає текстову довідкову інтерпретацію, наприклад, порівняння показників із нормою, тенденції змін та рекомендації щодо повторних вимірів чи звернення до лікаря. Отриманий результат зберігається в SQLite для подальшого перегляду.

Крім основних сценаріїв користувач може налаштовувати параметри застосунку (SettingsFragment), зокрема час нагадування про вимірювання та включення або вимкнення нотифікацій. Ці параметри також зберігаються в SQLite і забезпечують персоналізацію взаємодії.

Таким чином, система підтримує повний цикл роботи користувача: від створення профілю та введення медичних показників до їх перегляду та інтерпретації AI.

Діаграма варіантів використання показана на рис. 2.1.

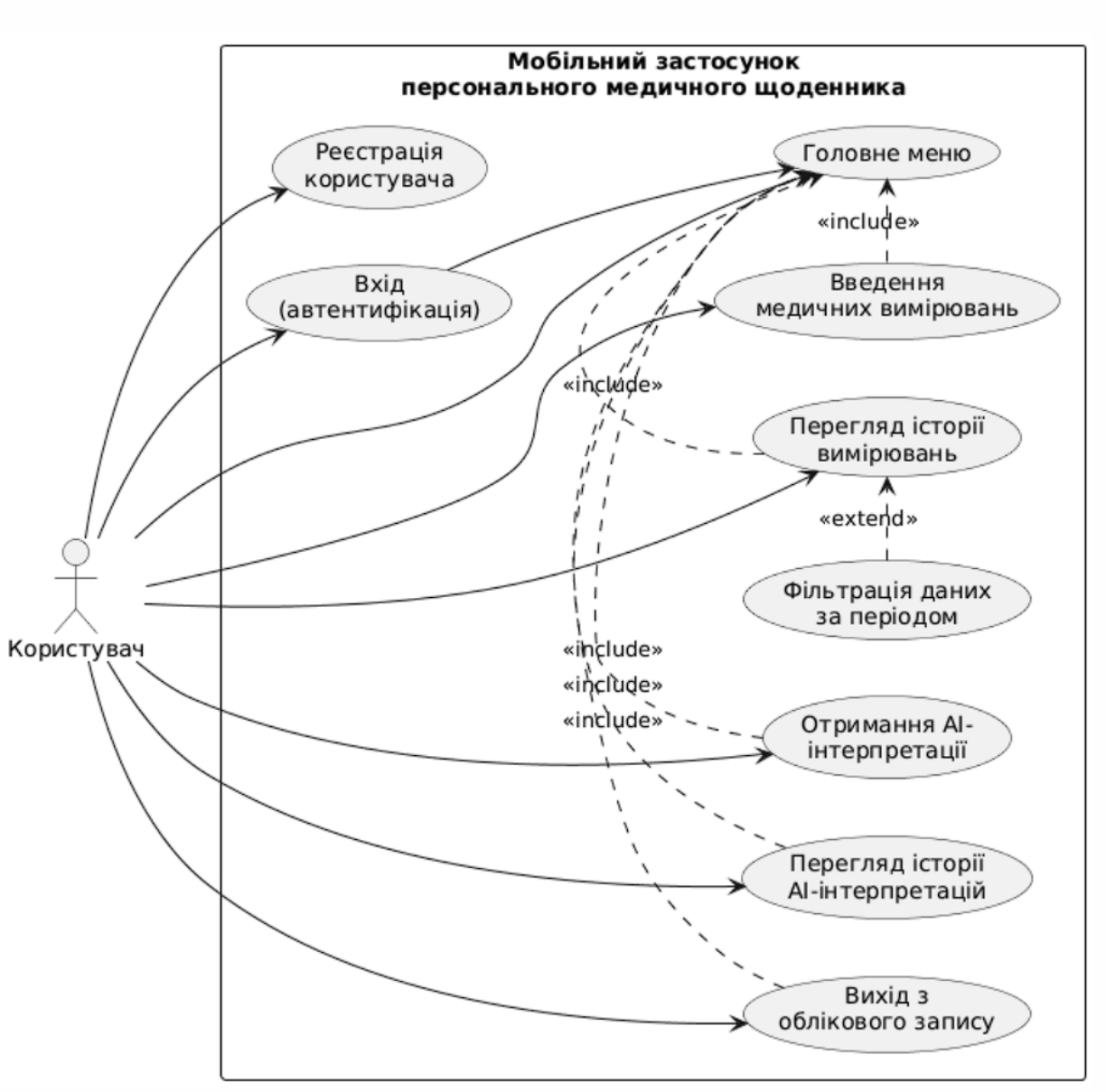


Рисунок 2.1 – Діаграма варіантів використання

## 2.4 Розробка ієрархії складових проєкту

Почнемо розробку ієрархії складових проєкту з діаграми класів (рис. 2.2). На діаграмі класів навмисно не відображено поля класів, фрагменти та стандартні методи життєвого циклу (onCreate, onCreateView, тощо). Діаграма фокусується виключно на логічній структурі системи, відповідальностях класів та їх взаємодії.

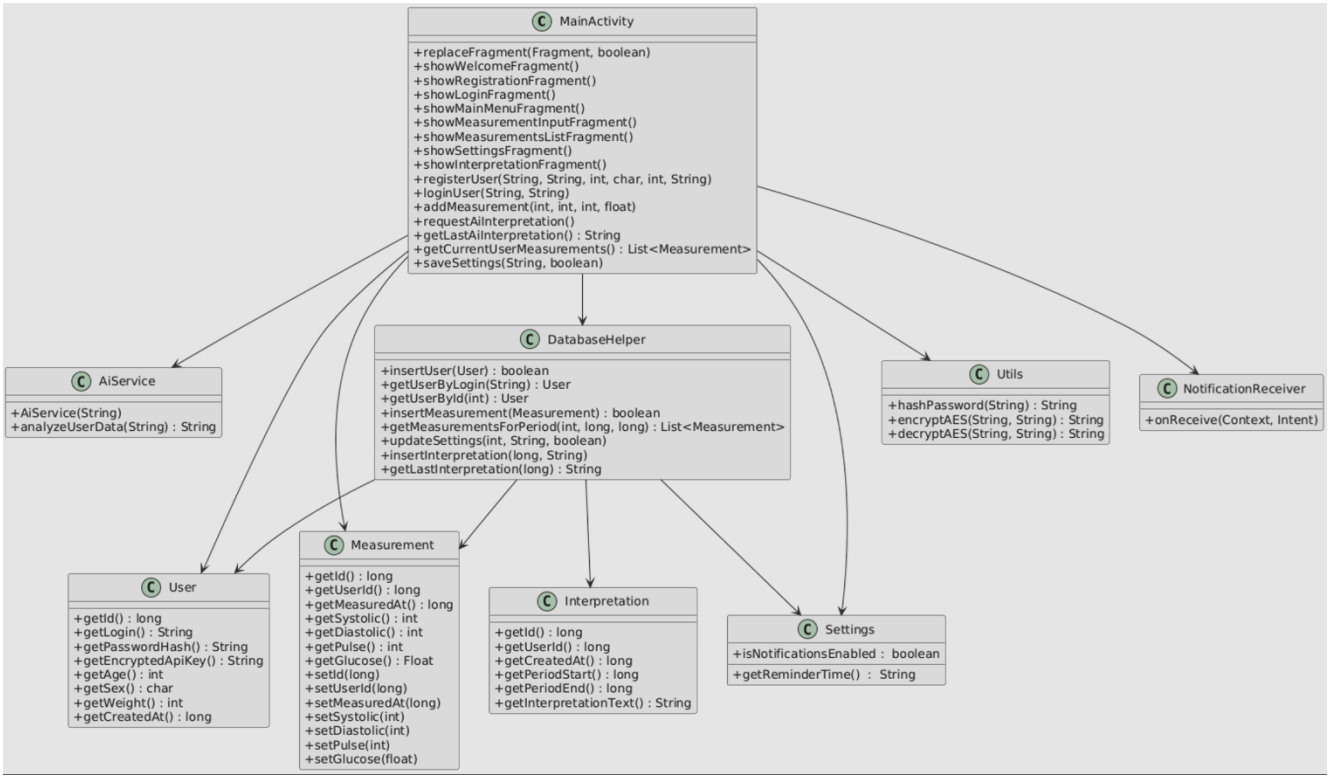


Рисунок 2.2 – Діаграма класів мобільного застосунку

MainActivity – центральний керуючий клас клієнтського застосунку. Виступає як координатор між фрагментами, бізнес-логікою та рівнем збереження даних. Основні функції:

- керування навігацією між фрагментами (відповідні методи showFragment,);
- обробка ключових сценаріїв користувача: реєстрація, логін, додавання вимірювань;
- ініціація запиту AI-інтерпретації медичних даних;
- взаємодія з локальною базою даних через DatabaseHelper;
- збереження та застосування налаштувань користувача;
- запуск механізмів сповіщень через NotificationReceiver.

MainActivity має залежності від DatabaseHelper, AiService, Utils та модельних класів, що відповідає патерну «Activity як фасад прикладної логіки».

AiService – сервісний клас для взаємодії із зовнішнім AI API. Призначення:

- інкапсулює всю логіку мережевих запитів до AI-сервісу;
- приймає дешифрований API-ключ користувача;

- виконує аналіз медичних даних у фоновому потоці;
- повертає текстову інтерпретацію результатів.

Відокремлення AiService від Activity дозволяє ізолювати мережеву логіку та спростити її можливу заміну або модифікацію.

DatabaseHelper – клас доступу до локальної SQLite бази даних. Відповідальності:

- створення та оновлення структури бази даних;
- CRUD-операції для користувачів, вимірювань, інтерпретацій та налаштувань;
- інкапсуляція SQL-запитів;
- повернення об’єктів моделі (User, Measurement, Interpretation, Settings) замість звичайних Cursor.

MainActivity взаємодіє з DatabaseHelper напряму, що зменшує кількість проміжних шарів-посередників.

Utils – допоміжний клас зі статичними методами. Призначення:

- хешування паролів користувачів (SHA-256);
- симетричне шифрування та дешифрування API-ключа (AES);
- централізація криптографічних операцій без дублювання коду.

Utils не зберігає стан і не має залежностей від інших компонентів системи.

NotificationReceiver – це широкомовний ресівер для обробки системних або запланованих подій. Використовується для реалізації нагадувань користувачу про необхідність внесення медичних вимірювань, а також для реакції на системні події. Зв’язок із MainActivity обмежується ініціалізацією та передачею параметрів налаштувань.

User – це модельний клас, що представляє обліковий запис користувача. Містить логічне представлення:

- ідентифікатора користувача;
- логіну та хешу пароля;
- зашифрованого AI API-ключа;
- базових антропометричних параметрів;

- дати створення облікового запису.

Використовується як транспортний об’єкт між DatabaseHelper та Activity. Measurement – це модель медичного вимірювання. Описує:

- часову мітку вимірювання;
- артеріальний тиск (систолічний та діастолічний);
- пульс;
- рівень глюкози в крові.

Interpretation – це модель результату AI-інтерпретації. Використовується DatabaseHelper для довготривалого зберігання результатів. Призначення:

- збереження текстових рекомендацій AI;
- прив’язка інтерпретації до користувача;
- фіксація періоду даних, за який виконано аналіз;
- збереження історії консультацій.

Settings – це модель налаштувань користувача. Містить параметри сповіщень, час нагадувань про необхідність здійснити виміри і ознаку активації нагадувань.

Структурна діаграма проєкту (рис. 2.3) відображає логічну організацію програмних компонентів мобільного застосунку персонального медичного щоденника та їх групування за функціональним призначенням. На верхньому рівні центральним елементом архітектури виступає MainActivity, яка реалізує one-Activity підхід і відповідає за керування життєвим циклом застосунку, навігацію між екранами, ініціацію бізнес-логіки та координацію взаємодії між основними підсистемами.

Пакет fragments містить набір UI-компонентів, кожен з яких відповідає окремому екрану застосунку. WelcomeFragment реалізує початковий екран з дисклеймером і вибором сценарію входу. RegistrationFragment та LoginFragment забезпечують реєстрацію та автентифікацію користувача. MainMenuFragment є центральним навігаційним вузлом авторизованої частини системи. MeasurementInputFragment та MeasurementsListFragment відповідають відповідно за введення нових медичних показників і перегляд історичних даних.

InterpretationFragment використовується для відображення результатів AI-інтерпретації, а SettingsFragment - для керування налаштуваннями користувача. Усі фрагменти не взаємодіють між собою напряму, а передають керування виключно через MainActivity, що забезпечує слабке зв’язування та спрощує підтримку навігаційної логіки.

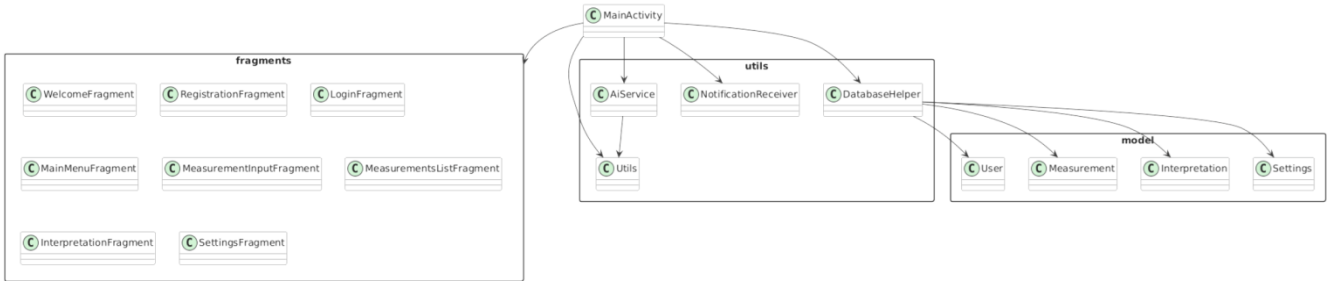


Рисунок 2.3 – Структурна діаграма застосунку

Пакет model містить класи моделі, які описують основні сутності предметної області. Клас User інкапсулює облікові та персональні дані користувача. Клас Measurement представляє окремий медичний вимір з часовою прив’язкою. Клас Interpretation зберігає результати AI-аналізу для визначеного періоду. Клас Settings описує індивідуальні налаштування користувача. Ці класи не містять логіки збереження чи обробки даних і використовуються як структури передавання інформації між шарами застосунку.

Пакет utils об’єднує допоміжні та сервісні компоненти. Клас DatabaseHelper реалізує доступ до локальної бази даних SQLite та відповідає за створення таблиць, виконання CRUD-операцій і зв’язок із класами моделі. Клас Utils надає спільні методи для хешування паролів і симетричного шифрування API-ключів. Клас AiService інкапсулює логіку взаємодії з зовнішнім AI API та використовує Utils для роботи з криптографічними операціями. NotificationReceiver відповідає за обробку системних подій, пов’язаних зі сповіщеннями, і може ініціювати відповідні дії в застосунку.

MainActivity має залежності від пакетів fragments і utils, що відображає її роль координатора між інтерфейсом користувача та сервісною логікою.

DatabaseHelper залежить від класів моделі, оскільки саме вони використовуються для представлення даних, збережених у базі. AiService додатково залежить від Utils, що забезпечує повторне використання криптографічних механізмів.

Компонентна діаграма (рис. 2.4) демонструє взаємодію основних програмних компонентів мобільного застосунку під час його виконання та зв’язки із зовнішніми системами. Компонент Android UI (Fragments) представляє набір екранів застосунку, через які користувач взаємодіє із системою. Всі дії користувача передаються до компонента MainActivity, що відповідає логіці єдиної точки керування.

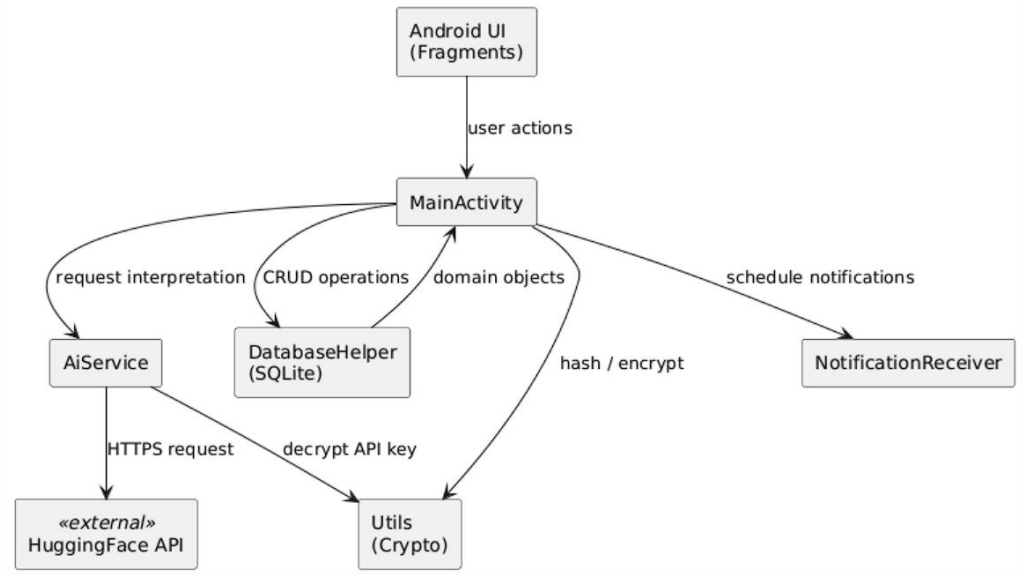


Рисунок 2.4 – Компонента діаграма застосунку

Компонент MainActivity виступає центральним елементом системи, який координує взаємодію між інтерфейсом користувача, базою даних, AI-сервісом та допоміжними модулями. Через нього здійснюються операції реєстрації, авторизації, збереження вимірювань, перегляду вимірювань та запуск AI-аналізу.

Компонент DatabaseHelper (SQLite) забезпечує локальне зберігання персональних медичних даних користувача. Компонент AiService відповідає за формування HTTP-запитів до зовнішнього AI-сервісу та обробку отриманих відповідей. Він отримує дешифрований API-ключ і використовує його для автентифікації запитів. Компонент Utils (Crypto) використовується як спільний

допоміжний модуль для криптографічних операцій, зокрема хешування паролів і шифрування або дешифрування API-ключів. Це забезпечує базовий рівень захисту персональних даних користувача. Компонент HuggingFace API позначений як зовнішній, оскільки він не входить до складу застосунку та використовується як віддалений сервіс для AI-інтерпретації медичних показників.

Компонент NotificationReceiver взаємодіє з MainActivity для реалізації механізму нагадувань і системних сповіщень відповідно до налаштувань користувача.

Діаграма послідовностей навігації між екранами (рис. 2.5) відображає порядок взаємодії користувача з мобільним застосунком персонального медичного щоденника та механізм переходів між фрагментами. Ініціатором усіх дій виступає користувач, який через елементи інтерфейсу фрагментів ініціює події навігації та бізнес-операції. Після запуску застосунку MainActivity ініціалізує та відображає WelcomeFragment, який виконує функцію початкового екрана та надає користувачу можливість перейти до реєстрації або автентифікації. У процесі реєстрації в RegistrationFragment користувач вводить облікові дані та власний ключ доступу до AI API, який шифрується та зберігається в локальній базі даних. Після успішної реєстрації система автоматично переводить користувача на екран входу, де в LoginFragment виконується перевірка логіну та хешованого пароля з використанням даних SQLite. У разі успішної автентифікації MainActivity здійснює перехід до MainMenuFragment, який є центральною точкою навігації авторизованої частини застосунку. Саме з головного меню користувач отримує доступ до основних сценаріїв роботи системи. Перехід до MeasurementInputFragment дозволяє ввести нові медичні показники, які після підтвердження зберігаються в локальній базі даних та повертають користувача до головного меню. Перехід до MeasurementsListFragment забезпечує перегляд історичних вимірювань, отриманих з бази даних за запитом через MainActivity.

Окремим сценарієм є отримання AI-інтерпретації медичних даних. Після переходу до InterpretationFragment користувач ініціює запит інтерпретації, який обробляється в окремому потоці через сервіс взаємодії з AI. MainActivity

відповідає за формування запиту на основі даних вимірювань, виклик AI API та збереження отриманого результату в таблиці інтерпретацій SQLite. Після завершення обробки результат передається назад у InterpretationFragment для відображення в інтерфейсі користувача з використанням Handler для коректного оновлення UI.

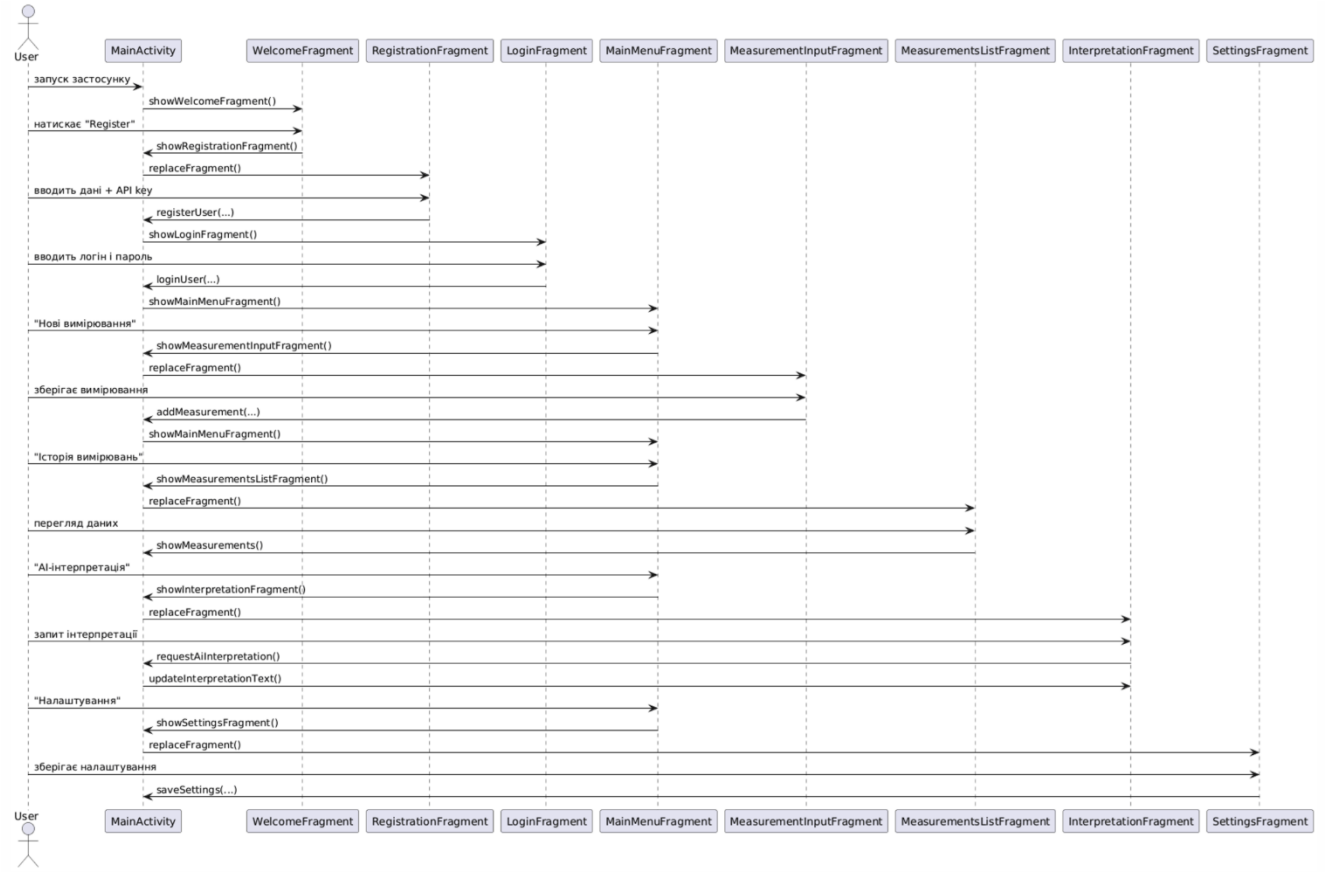


Рисунок 2.5 – Діаграма послідовностей навігації між екранами

Додатково з головного меню користувач може перейти до SettingsFragment, де налаштовуються параметри застосунку, зокрема керування сповіщеннями та часом нагадувань. Збереження налаштувань також здійснюється через MainActivity з подальшим записом у локальну базу даних. Таким чином, фрагменти відповідають за інтерфейс, а MainActivity - за координацію навігації, доступ до даних та взаємодію з AI-сервісом.

## 2.5 Проєктування бази даних

Проєктування бази даних персонального медичного щоденника (рис. 2.6) орієнтоване на локальне зберігання чутливої медичної інформації користувача та забезпечення простоти доступу та мінімальних вимог до ресурсів мобільного пристрою. Для реалізації обрано реляційну модель даних на основі SQLite, що є вбудованою СКБД платформи Android і не потребує додаткового встановлення або конфігурації з боку користувача. SQLite не потребує окремого серверного процесу, зберігає усі дані в одному файлі. Це робить її придатною для персональних застосунків із помірним обсягом даних, де доступ здійснюється з одного клієнта. Для медичного щоденника це дозволяє зберігати всі вимірювання та інтерпретації, що належать одному користувачу, локально.

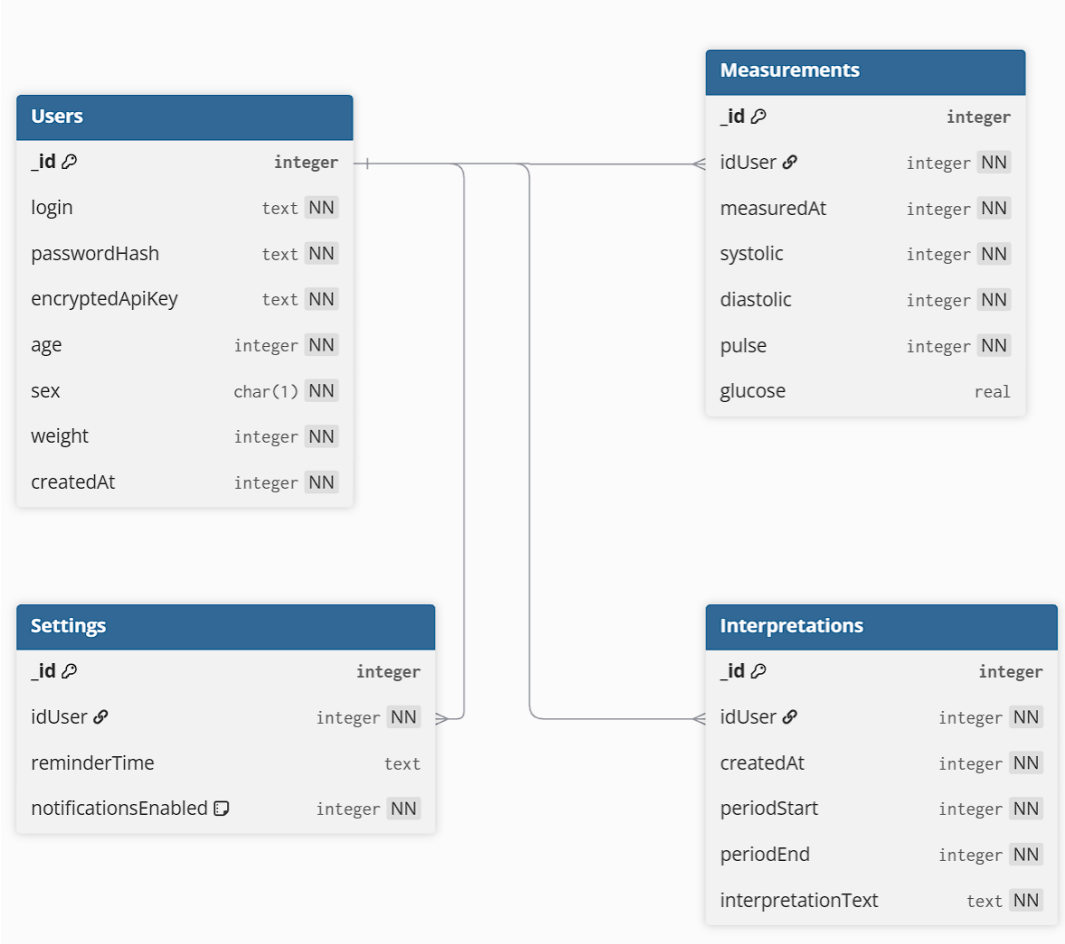


Рисунок 2.6 – ER-модель бази даних системи

Таблиця Users є головною сутністю системи та зберігає реєстраційні й базові дані користувача. Поле _id використовується як первинний ключ, що відповідає рекомендаціям Android щодо роботи з курсорами та адаптерами. Тип integer з атрибутом increment забезпечує унікальну ідентифікацію записів. Поле login має обмеження unique, що виключає дублювання облікових записів. Пароль зберігається у вигляді хешу (passwordHash), що виключає зберігання відкритих облікових даних. Поле encryptedApiKey містить ключ доступу до AI API, зашифрований із використанням пароля користувача, що підвищує рівень захисту персональних даних. Вік, масса і стать зберігаються як прості скалярні значення (integer, char(1)), оскільки ці дані використовуються в аналітичних запитах до AI та не потребують складної структури. Поле createdAt типу integer зберігає часову мітку реєстрації у форматі Unix timestamp, що спрощує сортування та фільтрацію.

Таблиця Measurements призначена для накопичення історичних медичних вимірювань користувача. Зв’язок з таблицею Users реалізований через зовнішній ключ idUser, що формує відношення типу один-до-багатьох: одному користувачу відповідає необмежена кількість вимірювань. Час вимірювання зберігається у полі measuredAt як integer, що дозволяє легко виконувати запити за діапазоном дат і не залежить від локальних форматів дати та часу. Показники тиску, пульсу та рівня глюкози представлені числовими типами (integer, real), що забезпечує коректну агрегацію, статистичний аналіз і передачу даних до AI-моделі. Поле glucose зроблено nullable, оскільки не кожне вимірювання обов’язково включає показник глюкози.

Таблиця Interpretations зберігає результати інтерпретацій, згенерованих AI-асистентом. Вона також пов’язана з таблицею Users через поле idUser, що дозволяє однозначно визначити власника інтерпретації. Поля periodStart та periodEnd задають часовий діапазон вимірювань, на основі яких була сформована відповідь AI, що забезпечує простежуваність результатів. Текст інтерпретації зберігається у полі interpretationText типу text, оскільки його довжина може бути змінною та потенційно значною. Поле createdAt фіксує момент генерації відповіді та використовується для впорядкування історії рекомендацій.

Таблиця Settings призначена для зберігання індивідуальних налаштувань застосунку. Вона має зв’язок один-до-одного з таблицею Users, що реалізовано через унікальне поле idUser. Це гарантує наявність не більше одного запису налаштувань для кожного користувача. Поле reminderTime зберігає час нагадування у текстовому форматі HH:mm, що спрощує інтеграцію з механізмами планування нотифікацій Android. Поле notificationsEnabled реалізоване як integer, оскільки SQLite не має окремого булевого типу.

DBML-опис бази даних наведено в лістингу 2.1.

Лістинг 2.1 - DBML-опис бази даних

```
Table Users {
  _id integer [pk, increment]
  login text [not null, unique]
  passwordHash text [not null]
  encryptedApiKey text [not null]
  age integer [not null]         // повних років
  sex char(1) [not null]         // 'M' або 'F'
  weight integer [not null]      // кг
  createdAt integer [not null]  // timestamp
}

Table Measurements {
  _id integer [pk, increment]
  idUser integer [not null, ref: > Users._id]
  measuredAt integer [not null] // timestamp
  systolic integer [not null]   // мм рт. ст.
  diastolic integer [not null]  // мм рт. ст.
  pulse integer [not null]      // уд/хв
  glucose real                  // ммоль/л, nullable
}

Table Interpretations {
  _id integer [pk, increment]
  idUser integer [not null, ref: > Users._id]
  createdAt integer [not null]  // коли згенерована AI-відповідь
  periodStart integer [not null]
  periodEnd integer [not null]
  interpretationText text [not null]
}

Table Settings {
  _id integer [pk, increment]
  idUser integer [not null, unique, ref: > Users._id]
  reminderTime text             // HH:mm
  notificationsEnabled integer [not null, default: 1]}
```

## 2.6 Прототипування системи

Прототип системи (рис. 2.7) демонструє повний цикл взаємодії користувача з системою: від етапу авторизації до використання інтелектуальних функцій аналізу медичних показників. Екран вітання виступає точкою входу, пропонуючи вибір між реєстрацією нового профілю та входом у вже існуючий. Екран реєстрації містить поля для введення персональних даних та технічних параметрів в тому числі API-ключа Hugging Face для доступу до нейромережевих моделей. Після завершення реєстрації система автоматично переспрямовує користувача на форму входу. Екран входу забезпечує автентифікацію користувача. Після успішної перевірки облікових даних через MainActivity, здійснюється перехід до головного модуля управління. Центральним вузлом навігації є головне меню, яке реалізоване за принципом «зіркоподібної» топології. Це дозволяє користувачу отримати швидкий доступ до будь-якого функціонального модуля в один клік:

- форма введення даних для фіксації поточних показників здоров’я (систолічний/діастолічний тиск, пульс, рівень глюкози). Логіка «Save/Back» передбачає автоматичне повернення до головного меню після успішного запису даних у базу SQLite;
- історія вимірювань: список усіх збережених показників з можливістю фільтрації. Кнопка «Interpret» дозволяє миттєво перейти до аналізу поточних даних без повернення у головне меню;
- AI-інтерпретація відображає результати аналізу, згенеровані моделлю Hugging Face. Інтерфейс підтримує динамічне оновлення тексту та можливість повторного запиту аналізу;
- налаштування, де користувач може встановити час щоденних нагадувань та керувати статусом push-повідомлень.

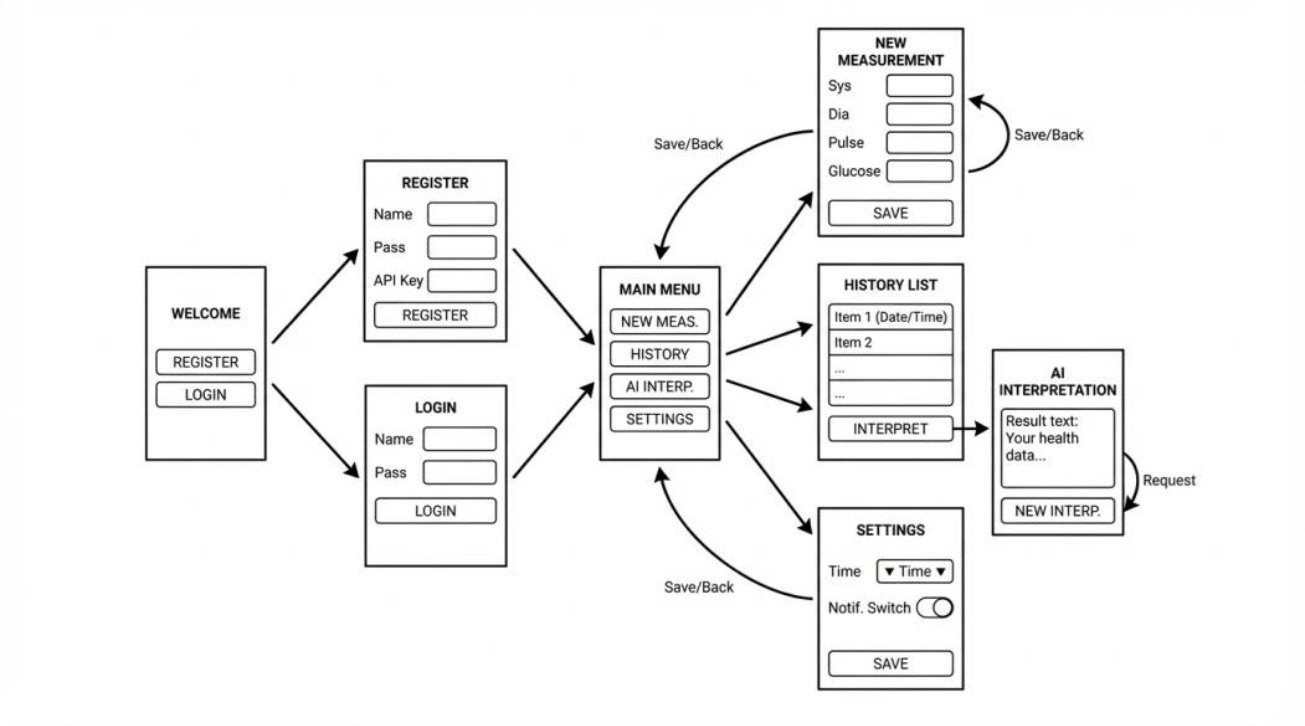


Рисунок 2.7 – Прототип системи

## 2.7 Висновки розділу

У другому розділі виконано комплексне проєктування комп’ютерної системи персонального медичного щоденника, що охоплює архітектурні, функціональні, структурні та інформаційні аспекти розроблюваного мобільного застосунку. У результаті проведеного аналізу та проєктування сформовано уявлення про логіку роботи системи, взаємодію її складових та способи забезпечення безпеки персональних медичних даних.

Обґрунтовано вибір клієнт-орієнтованої архітектури без власного серверного бекенду, що базується на автономній роботі Android-застосунку з локальним зберіганням даних та використанням зовнішнього AI-сервісу виключно для аналітичних задач. Такий підхід забезпечує масштабованість, незалежність клієнтів, зменшення ризиків витоку даних та можливість роботи без постійного мережевого з’єднання. Обрано one-Activity архітектуру з

використанням Fragment, що спрощує керування життєвим циклом інтерфейсу.

Визначено системні вимоги до апаратного та програмного забезпечення, які гарантують стабільну роботу застосунку на широкому спектрі сучасних мобільних пристроїв. Встановлено, що система не потребує спеціалізованого обладнання та може функціонувати на стандартних смартфонах і планшетах під управлінням Android 10 і вище.

Виконано аналіз варіантів використання, що дозволив формалізувати основні сценарії взаємодії користувача із системою: реєстрацію, автентифікацію, введення та перегляд медичних вимірювань, отримання AI-інтерпретацій і налаштування параметрів застосунку. Розроблено ієрархію складових проєкту з використанням UML-діаграм. Діаграма класів, структурна та компонентна діаграми дозволили відобразити логічну організацію програмних модулів, їх відповідальності та зв’язки між ними. Діаграма послідовностей навігації між екранами наочно продемонструвала механізм взаємодії користувача з фрагментами та централізовану роль MainActivity у керуванні навігацією, доступом до даних та взаємодією з AI-сервісом. Спроєктовано структуру локальної бази даних SQLite. Розроблена ER-модель та DBML-опис забезпечують цілісність даних та зв’язки між сутностями. Обрана структура бази даних відповідає вимогам безпеки та простоти доступу. Представлено прототип системи, який демонструє логіку інтерфейсу та повний цикл взаємодії користувача із застосунком.

# 3 РЕАЛІЗАЦІЯ КОМП'ЮТЕРНОЇ СИСТЕМИ ПЕРСОНАЛЬНОГО МЕДИЧНОГО ЩОДЕННИКА

## 3.1 Реалізація графічного інтерфейсу

Для реалізації графічного інтерфейсу потрібно написати layout xml-документи для кожного фрагменту. В layout activity_main.xml слід вставити

контейнер-плейсхолдер FrameLayout з атрибутом id="@+id/fragmentContainer". В цей контейнер ми будемо в java-коді вставляти фрагменти, і в такий спосіб MainActivity стане хостом для всіх фрагментів. Для демонстрації верстки нашого типового фрагменту покажемо layout фрагменту реєстрації в лістингу 3.1.

Лістинг 3.1 – Верстка фрагменту реєстрації

```
<LinearLayout  android:id="@+id/registerRoot"
    android:orientation="vertical"
    android:padding="24dp"
    android:background="@color/bg_medical">
    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical">
            <EditText
                android:id="@+id/etLogin"
                android:hint="Login" />
            <EditText
                android:id="@+id/etPassword"
                android:hint="Password"
                android:inputType="textPassword" />
            <EditText
                android:id="@+id/etAge"
                android:hint="Age"
                android:inputType="number" />
            <EditText
                android:id="@+id/etSex"
                android:hint="Sex (M/F)" />
            <EditText
                android:id="@+id/etWeight"
                android:hint="Weight (kg)"
                android:inputType="number" />
            <TextView
                android:id="@+id/tvApiKeyLink"
                android:text="Get Hugging Face API key"
                android:textColor="#0000FF" />
            <EditText
                android:id="@+id/etApiKey"
                android:hint="Paste API key here" />
            <Button
                android:id="@+id/btnRegisterConfirm"
                android:text="Register" />
        </LinearLayout>
    </ScrollView>
</LinearLayout>
```

В основі інтерфейсу використано вертикальний LinearLayout, який задає єдину логічну вісь розміщення елементів і забезпечує інтуїтивно зрозумілий порядок заповнення форми зверху вниз. Заданий внутрішній відступ і однотонний фон медичної тематики. Для забезпечення коректної роботи на пристроях з різною діагоналлю екрана та в альбомній орієнтації у верстці використано ScrollView, що дозволяє користувачу безперешкодно прокручувати форму навіть у разі обмеженого вертикального простору. Всі елементи введення згруповані в окремому вкладеному контейнері, що спрощує масштабування інтерфейсу.

Інтерфейс поєднує стандартні поля введення облікових і медичних даних із елементом, який забезпечує інтеграцію із зовнішнім AI-сервісом. Наявність текстового посилання для отримання API-ключа вписується в сценарій реєстрації. Завершується форма кнопкою підтвердження реєстрації.

Для забезпечення коректної роботи застосунку при зміні орієнтації пристрою у проєкті передбачено використання альтернативних ресурсів верстки для альбомного режиму (layout-land). Файли верстки, розміщені в каталозі layout-land, автоматично підхоплюються системою під час перевороту екрана в альбомну орієнтацію. Це дає змогу змінювати пропорції, відступи та структуру розміщення елементів, враховуючи обмежену висоту екрана та збільшену ширину. Для форм введення це дозволяє зменшити вертикальні відступи та змінити групування елементів без впливу на їх ідентифікатори та логіку доступу з коду.

Під час перевороту пристрою Android перестворює MainActivity, однак у межах one-Activity архітектури з фрагментами це не призводить до порушення навігації. Поточний фрагмент повторно відображається з використанням відповідної верстки, а керування переходами між екранами залишається у MainActivity. Для збереження стану Activity використовуємо метод onSaveInstanceState, де запам’ятовуємо користувача – об’єкт класу User. При перевороті перевіряємо savedInstanceState і якщо він не null, тоді відновлюємо стан користувача. Аналогічний підхід використовуємо у фрагментах. Таким чином, користувач може перевертати екран в будь-який момент і застосунок не

забуде поточний стан. Верстка інших фрагментів створюється аналогічно.

Методи навігації в класі MainActivity (лістинг 3.2) виконують роль керування графічним інтерфейсом.

Лістинг 3.2 – Методи навігації в класі MainActivity

```
public void replaceFragment(Fragment fragment,
                            boolean addToBackStack)
{
        FragmentTransaction ft =
getSupportFragmentManager().beginTransaction();
        ft.replace(R.id.fragmentContainer, fragment);
        if (addToBackStack) ft.addToBackStack(null);
        ft.commit();
}
    public void showWelcomeFragment()
{ replaceFragment(new WelcomeFragment(), false); }
    public void showRegistrationFragment()
{ replaceFragment(new RegistrationFragment(), true); }
    public void showLoginFragment()
{ replaceFragment(new LoginFragment(), true); }
    public void showMainMenuFragment()
{ replaceFragment(new MainMenuFragment(), false); }
    public void showMeasurementInputFragment()
{ replaceFragment(new MeasurementInputFragment(), true); }
    public void showMeasurementsListFragment()
{ replaceFragment(new MeasurementsListFragment(), true); }
    public void showSettingsFragment()
{ replaceFragment(new SettingsFragment(), true); }
    public void showInterpretationFragment()
{ replaceFragment(new InterpretationFragment(), true); }
```

Уся логіка переходів між екранами зосереджена в одному класі, що усуває дублювання коду у фрагментах і спрощує контроль життєвого циклу UI.

Основним елементом навігації є метод заміни фрагментів replaceFragment, який використовує FragmentManager та FragmentTransaction для підміни вмісту контейнера FrameLayout. Саме цей метод забезпечує фізичну зміну екрана шляхом операції replace, та за потреби додає транзакцію до стеку зворотніх викликів. Це дозволяє коректно обробляти кнопку «Назад» і повертатися до попередніх екранів.

Методи showWelcomeFragment, showRegistrationFragment, showLoginFragment, showMainMenuFragment та інші є навігаційними обгортками.

Вони створюють відповідний фрагмент і передають його до replaceFragment з необхідними параметрами. Такий підхід підвищує читабельність коду, оскільки кожен екран має окрему точку входу в навігації.

Фрагменти не здійснюють безпосередніх транзакцій і не взаємодіють з FragmentManager напряму. Натомість вони викликають відповідні методи MainActivity, передаючи лише події користувача (натискання кнопок, підтвердження дій). Це чітко розмежовує відповідальності: фрагменти відповідають за відображення та збір введених даних, а Activity - за навігацію та зміну екранів.

### 3.2 Реалізація бази даних та моделі системи

Для створення бази даних потрібно написати клас DatabaseHelper, який спадкує SQLiteOpenHelper. Визначення методу onCreate цього класу наведено в лістингу 3.3. Метод onCreate реалізує ініціалізацію локальної бази даних застосунку та формує її логічну структуру під час першого запуску. В цьому методі створюються всі основні таблиці, необхідні для функціонування персонального медичного щоденника.

Структура бази даних побудована навколо таблиці Users, яка виступає центральною сутністю та зберігає облікові й персональні дані користувача. Від неї логічно залежать усі інші таблиці, що відображає модель «один користувач - багато пов’язаних даних». Таблиця Measurements призначена для накопичення історії медичних вимірювань і пов’язується з користувачем через зовнішній ключ. Це дає змогу зберігати довільну кількість записів у часовому розрізі та використовувати їх для перегляду історії. Таблиця Interpretations зберігає результати роботи AI-асистента разом із часовими межами аналізованого періоду. Таким чином, інтерпретації не генеруються повторно без потреби, а можуть бути повторно використані.

Лістинг 3.3 – Метод onCreate класу DatabaseHelper

```
public void onCreate(SQLiteDatabase db) {
db.execSQL("CREATE TABLE Users (" +
                        "_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
                        "login TEXT NOT NULL UNIQUE, " +
                        "passwordHash TEXT NOT NULL, " +
                        "encryptedApiKey TEXT NOT NULL, " +
                        "age INTEGER NOT NULL, " +
                        "sex CHAR(1) NOT NULL, " +
                        "weight INTEGER NOT NULL, " +
                        "createdAt INTEGER NOT NULL)");
db.execSQL("CREATE TABLE Measurements (" +
                        "_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
                        "idUser INTEGER NOT NULL REFERENCES
                         Users(_id), " +
                        "measuredAt INTEGER NOT NULL, " +
                        "systolic INTEGER NOT NULL, " +
                        "diastolic INTEGER NOT NULL, " +
                        "pulse INTEGER NOT NULL, " +
                        "glucose REAL)");
db.execSQL("CREATE TABLE Interpretations (" +
                        "_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
                        "idUser INTEGER NOT NULL REFERENCES
                         Users(_id), " +
                        "createdAt INTEGER NOT NULL, " +
                        "periodStart INTEGER NOT NULL, " +
                        "periodEnd INTEGER NOT NULL, " +
                        "interpretationText TEXT NOT NULL)");
db.execSQL("CREATE TABLE Settings (" +
                        "_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
                        "idUser INTEGER NOT NULL UNIQUE REFERENCES
                         Users(_id), " +
                        "reminderTime TEXT, " +
                        "notificationsEnabled INTEGER NOT NULL
                         DEFAULT 1)");}
```

Таблиця Settings винесена в окрему сутність для збереження налаштувань користувача, зокрема параметрів нагадувань і сповіщень. Її зв’язок «один до одного» з таблицею Users підкреслює, що кожен користувач має власний набір налаштувань, незалежний від інших даних. З лістингу 3.3 видно як створюються таблиці згідно DBML-опису, наведеному в лістингу 2.1.

Сутностям в базі даних відповідають класи в пакеті model, а саме: User, Settings, Interpretation та Measurement. Поля цих класів, що відображають предметну область, наведено в лістинг 3.4

Лістинг 3.4 – Поля класів, що відображають модель системи

```
public class User {
    private long id;
    private String login;
    private String passwordHash;
    private String encryptedApiKey;
    private int age;
    private char sex;
    private int weight;
    private long createdAt;
//конструктор, getters, setters
}

public class Measurement {
    private long id;
    private long userId;
    private long measuredAt;
    private int systolic;
    private int diastolic;
    private int pulse;
    private float glucose;
//конструктор, getters, setters
}

public class Interpretation {
    private long id;
    private long userId;
    private long createdAt;
    private long periodStart;
    private long periodEnd;
    private String interpretationText;
//конструктор, getters, setters
}

public class Settings {
    private long id;
    private long userId;
    private String reminderTime;
    private boolean notificationsEnabled;
//конструктор, getters, setters
}
```

Методи вставки нового зареєстрованого користувача та нового вимірювання наведено в лістингу 3.5. Метод insertUser відповідає за збереження нового користувача в таблиці Users. Він виконує трансформацію об’єкта предметної області User у формат, придатний для зберігання в реляційній базі даних. Метод формує запис, фіксує момент створення облікового запису та повертає логічний

результат операції. Метод insertMeasurement реалізує аналогічний підхід для збереження медичних вимірювань. Він забезпечує прив’язку кожного запису до конкретного користувача через ідентифікатор.

Лістинг 3.5 – Методи вставки даних в базу

```
public boolean insertUser(User user) {
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues cv = new ContentValues();
        cv.put("login", user.getLogin());
        cv.put("passwordHash", user.getPasswordHash());
        cv.put("encryptedApiKey", user.getEncryptedApiKey());
        cv.put("age", user.getAge());
        cv.put("sex", String.valueOf(user.getSex()));
        cv.put("weight", user.getWeight());
        cv.put("createdAt", System.currentTimeMillis());
        long id = db.insert("Users", null, cv);
        return id != -1;}

public boolean insertMeasurement(Measurement m) {
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues cv = new ContentValues();
        cv.put("idUser", m.getUserId());
        cv.put("measuredAt", m.getMeasuredAt());
        cv.put("systolic", m.getSystolic());
        cv.put("diastolic", m.getDiastolic());
        cv.put("pulse", m.getPulse());
        cv.put("glucose", m.getGlucose());
        long id = db.insert("Measurements", null, cv);
        return id != -1;}
```

Методи отримання даних з бази наведено в лістингу 3.6. Метод getMeasurementsForPeriod реалізує вибірку історичних медичних вимірювань користувача за заданий часовий інтервал і дозволяє отримання даних для відображення статистики або подальшого аналізу. Метод формує параметризований запит до таблиці Measurements, який поєднує дві умови: прив’язку записів до конкретного користувача та обмеження за часовими межами вимірювання. Отримані з бази дані перетворюються з табличного представлення у список об’єктів Measurement. Сортування за часом у зворотному порядку дозволяє одразу отримати дані у вигляді, зручному для інтерфейсу користувача.

Метод getLastInterpretation відповідає за отримання останнього текстового висновку, згенерованого AI-асистентом для конкретного користувача. Він

реалізує вибірку найсвіжішого запису з таблиці Interpretations, використовуючи сортування за датою створення та обмеження результату до одного рядка.

Лістинг 3.6 – Методи виборки даних з бази

```
public List<Measurement> getMeasurementsForPeriod(int userId, long
startTime, long endTime) {
List<Measurement> list = new ArrayList<>();
SQLiteDatabase db = this.getReadableDatabase();
String selection = "idUser = ? AND measuredAt BETWEEN ? AND ?";
String[] selectionArgs = {String.valueOf(userId),
String.valueOf(startTime), String.valueOf(endTime)};
Cursor cursor = db.query("Measurements", null, selection,
selectionArgs, null, null, "measuredAt DESC");
if (cursor != null)
{
 while (cursor.moveToNext()) {
Measurement m = new Measurement();
m.setId(cursor.getInt(cursor.getColumnIndexOrThrow("_id")));
m.setUserId(cursor.getInt(cursor.getColumnIndexOrThrow("idUser")));
m.setMeasuredAt(cursor.getLong(cursor.getColumnIndexOrThrow("measure
dAt")));
m.setSystolic(cursor.getInt(cursor.getColumnIndexOrThrow("systolic")
));
m.setDiastolic(cursor.getInt(cursor.getColumnIndexOrThrow("diastolic
")));
m.setPulse(cursor.getInt(cursor.getColumnIndexOrThrow("pulse")));
m.setGlucose(cursor.getFloat(cursor.getColumnIndexOrThrow("glucose")
));
               list.add(m);
}
            cursor.close();
}
return list;}

public String getLastInterpretation(long userId) {
        SQLiteDatabase db = this.getReadableDatabase();
        String result = null;
        Cursor cursor = db.query("Interpretations", new
String[]{"interpretationText"},
                "idUser = ?", new String[]{String.valueOf(userId)},
                null, null, "createdAt DESC", "1");
        if (cursor != null && cursor.moveToFirst()) {
            result = cursor.getString(0);
            cursor.close();
        }
 return result;
}
```

## 3.3 Розробка модуля інтерпретації та мережевої взаємодії

Код класу AiService (лістинг 3.7) реалізує послідовність дій, необхідних для звернення до зовнішнього AI-сервісу та отримання текстової інтерпретації медичних даних користувача. Спочатку відбувається ініціалізація сервісу. Під час створення об’єкта AiService до нього передається API-ключ користувача, який зберігається у полі класу та надалі використовується для автентифікації запитів до сервісу Hugging Face. Таким чином, клас є прив’язаним до конкретного користувача і його персонального доступу до AI API.

Другий етап – формування HTTP-клієнта та підготовка запиту. Усередині методу analyzeUserData створюється екземпляр OkHttpClient, який відповідає за виконання мережевих запитів у середовищі Android. Далі програмно формується JSON-тіло запиту відповідно до специфікації API чат-комплішенів: задається назва мовної моделі, створюється масив повідомлень і додається повідомлення з роллю користувача, яке містить зведені медичні дані та інструкцію щодо аналізу.

Далі виконується налаштування HTTP-запиту. На основі сформованого JSON створюється об’єкт RequestBody з відповідним MIME-типом. Будується HTTP POST-запит до фіксованої адреси API, у який додається заголовок Authorization з bearer-токеном та прикріплюється тіло запиту. Тут визначається які дані і з якими правами передаються зовнішньому сервісу.

Після цього здійснюється мережевий запит та обробка відповіді. Запит синхронно виконується через OkHttpClient, після чого перевіряється успішність відповіді сервера та зчитується текст відповіді у форматі JSON.

Далі відбувається аналіз відповіді AI-сервісу. Отриманий JSON-рядок розбирається, з нього вилучається масив choices, який містить згенеровані варіанти відповідей моделі. Із першого елемента масиву дістається текст повідомлення, що є результатом інтерпретації медичних даних. Якщо AI-сервіс не повернув жодного варіанту відповіді, метод повертає повідомлення про відсутність результату.

Лістинг 3.7 – Клас зв’язку з зовнішім API штучного інтелекту

```
public class AiService {
    private static final String TAG = "AiService";
    private final String apiKey;
    private static final String API_URL =
"https://router.huggingface.co/v1/chat/completions";
    private static final String MODEL_NAME = "meta-llama/Llama-3.1-
8B-Instruct";
    public AiService(String apiKey) {
        this.apiKey = apiKey;
    }
    public String analyzeUserData(String medicalData) {
        OkHttpClient client = new OkHttpClient();
            JSONObject jsonBody = new JSONObject();
            jsonBody.put("model", MODEL_NAME);
            JSONArray messages = new JSONArray();
            JSONObject message = new JSONObject();
            message.put("role", "user");
            message.put("content", "Ти медичний асистент.
Проаналізуй дані: " + medicalData + ". Відповідь дай українською
мовою.");
            messages.put(message);
            jsonBody.put("messages", messages);
            RequestBody body = RequestBody.create(
                    jsonBody.toString(),
            MediaType.get("application/json; charset=utf-8"));
            Request request = new Request.Builder()
                    .url(API_URL)
                    .addHeader("Authorization", "Bearer " + apiKey)
                    .post(body)
                    .build();
            Response response = client.newCall(request).execute()) {
                if (!response.isSuccessful()) {
                    String errorResp = response.body() != null ?
response.body().string() : "Unknown error";}
                String responseBody = response.body().string();
                JSONObject obj = new JSONObject(responseBody);
                JSONArray choices = obj.getJSONArray("choices");
                if (choices.length() > 0) {
                    return choices.getJSONObject(0)
                            .getJSONObject("message")
                            .getString("content");}
                return "AI не надав відповіді";}
        }
    }
```

Для того, щоб мережевий запит міг відбуватися потрібно в маніфесті прописати дозвіл:

<uses-permission android:name="android.permission.INTERNET"/>.

Він не потребує підтвердження користувача в рантаймі і видається один раз

на етапі встановлення застосунку. Крім того потрібно підключити бібліотеку implementation("com.squareup.okhttp3:okhttp:4.12.0").

Виклик модуля AiService відбувається з MainActivity (лістинг 3.8) після натискання кнопки на відповідному фрагменті.

Лістинг 3.8 – Виклик модуля інтерпретації за допомогою AI

```
public void requestAiInterpretation() {
if (currentUser == null || currentPassword == null) return;
        new Thread(() -> {
            List<Measurement> measurements =
dbHelper.getMeasurementsForPeriod(
(int) currentUser.getId(), 0, System.currentTimeMillis());
                StringBuilder sb = new StringBuilder();
                sb.append("Дані користувача:\n");
                sb.append(String.format("- Стать: %s\n",
currentUser.getSex() == 'M' ? "чоловіча" : "жіноча"));
                sb.append(String.format("- Вік: %d років\n",
currentUser.getAge()));
                sb.append(String.format("- Вага: %d кг\n\n",
currentUser.getWeight()));
                sb.append("Історія вимірювань:\n");
                if (measurements.isEmpty()) {
                    sb.append("Дані про вимірювання відсутні.");
                } else {
                    for (Measurement m : measurements) {
                        sb.append(String.format("Тиск: %d/%d, Пульс:
%d, Цукор: %.1f; ",
                      m.getSystolic(), m.getDiastolic(), m.getPulse(),
m.getGlucose()));}
                }
                String apiKey =
Utils.decryptAES(currentUser.getEncryptedApiKey(), currentPassword);
                AiService ai = new AiService(apiKey);
                String aiResult = ai.analyzeUserData(sb.toString());
                dbHelper.insertInterpretation(currentUser.getId(),
aiResult);
                mainHandler.post(() -> {
                Fragment f =
getSupportFragmentManager().findFragmentById(R.id.fragmentContainer)
;
                 if (f instanceof InterpretationFragment) {
((InterpretationFragment) f).setInterpretationText(aiResult);
                    }
                });
            }).start();
}
```

Виконання починається з перевірки наявності авторизованого користувача та збереженого пароля поточної сесії. За відсутності цих даних виклик AI є неможливим, оскільки не може бути отриманий доступ до зашифрованого API-ключа.

Подальша логіка виконується в окремому потоці, що дозволяє ізолювати операції читання з бази даних і мережевий запит від головного потоку інтерфейсу. У фоновому виконанні з локальної бази даних зчитується повна історія медичних вимірювань користувача, після чого формується узагальнений текстовий опис. У цьому описі поєднуються статичні дані профілю користувача та динамічні показники вимірювань. Перед зверненням до зовнішнього AI-сервісу виконується дешифрування API-ключа. Для цього використовується метод Utils.decryptAES, який приймає зашифрований ключ із бази даних та пароль користувача. Таким чином, відкритий API-ключ існує лише в оперативній пам’яті під час виконання запиту і не зберігається у відкритому вигляді в жодному постійному сховищі.

Отриманий ключ передається в конструктор AiService, після чого викликається метод analyzeUserData. Тут відбувається формування HTTP-запиту до AI API, надсилання підготовленого тексту з медичними даними та отримання відповіді у вигляді текстової інтерпретації. Після отримання відповіді результат одразу зберігається в локальній базі даних як окрема інтерпретація, прив’язана до користувача. Це забезпечує накопичення історії AI-висновків та усуває потребу повторного звернення до зовнішнього сервісу для перегляду вже отриманих результатів.

Результат інтерпретації передаєтьсяв головний потік застосунку за допомогою mainHandler. Поточний активний фрагмент перевіряється, і якщо ним є InterpretationFragment, отриманий текст інтерпретації встановлюється у відповідний елемент інтерфейсу. Така організація дозволяє безпечно оновлювати графічний інтерфейс.

Клас Utils реалізує допоміжні криптографічні та хеш-функції для безпечного зберігання і обробки конфіденційних даних користувача. Методи хешування паролей, шифрування та дешифрування ключа від зовнішнього API

наведено в лістингу 3.9.

Лістинг 3.9 – Методи-утиліти класу Utils

```
public class Utils {
    public static String hashPassword(String password) {
       MessageDigest digest = MessageDigest.getInstance("SHA-256");
            byte[] hash = digest.digest(password.getBytes());
            StringBuilder hexString = new StringBuilder();
            for (byte b : hash) {
                String hex = Integer.toHexString(0xff & b);
                if (hex.length() == 1) hexString.append('0');
                hexString.append(hex);
            }
            return hexString.toString();
}

public static String encryptAES(String plainText, String password) {
            SecretKeySpec key = generateKey(password);
            Cipher cipher =
Cipher.getInstance("AES/ECB/PKCS5Padding");
            cipher.init(Cipher.ENCRYPT_MODE, key);
            byte[] encrypted = cipher.doFinal(plainText.getBytes());
            return Base64.encodeToString(encrypted, Base64.DEFAULT);
    }

    public static String decryptAES(String cipherText, String
password) {
            SecretKeySpec key = generateKey(password);
            Cipher cipher =
Cipher.getInstance("AES/ECB/PKCS5Padding");
            cipher.init(Cipher.DECRYPT_MODE, key);
            byte[] decoded = Base64.decode(cipherText,
Base64.DEFAULT);
            byte[] decrypted = cipher.doFinal(decoded);
            return new String(decrypted);
   }

    private static SecretKeySpec generateKey(String password) throws
Exception {
        MessageDigest sha = MessageDigest.getInstance("SHA-256");
        byte[] key = sha.digest(password.getBytes("UTF-8"));
        byte[] key16 = new byte[16];
        System.arraycopy(key, 0, key16, 0, 16);
        return new SecretKeySpec(key16, "AES");}
}
```

Метод hashPassword перетворює пароль користувача у фіксовану хеш-строку SHA-256. Хешування робиться односторонньо, результат формується у

шістнадцятковому представленні, що дозволяє зберігати його у базі без ризику відновлення оригінального пароля.

Методи encryptAES та decryptAES забезпечують симетричне шифрування та дешифрування даних (API-ключа). Для цього використовується AES у режимі електронної кодової книги. Ключ шифрування генерується на основі пароля користувача через хешування SHA-256 та обрізання до 16 байт, що відповідає довжині ключа AES-128. Результат шифрування кодується у Base64 для безпечного зберігання. Метод generateKey виконує створення секретного ключа із пароля.

### 3.4 Розробка модуля нотифікацій

У модулі нотифікацій Android-застосунку персонального медичного щоденника використовується подієва модель платформи, на основі системних широкомовних повідомлень Broadcast та сервісу сповіщень. Нотифікації застосовуються для нагадування користувачу про необхідність регулярного внесення медичних вимірювань. Логіка нагадувань наступна:

- час нагадування та стан увімкнення нотифікацій зберігаються у таблиці Settings;
- системний планувальник (AlarmManager) ініціює Broadcast у заданий час;
- Broadcast перехоплюється спеціалізованим приймачем NotificationReceiver;
- приймач формує та відображає системне сповіщення, яке веде користувача до головного екрану застосунку.

Клас NotificationReceiver (лістинг 3.10) спадкує BroadcastReceiver і виконує роль точки входу для системної події нагадування. Його метод onReceive викликається операційною системою у момент спрацювання запланованого нагадування. На початку обробки отримується екземпляр NotificationManager,

який відповідає за створення та показ сповіщень. Для пристроїв з Android 8.0 і вище додатково створюється канал нотифікацій. Канал задає логічну категорію сповіщень медичного застосунку, їх важливість, візуальні та вібраційні параметри. Це дозволяє розрізняти пріоритетність нотифікацій і по суті вводить числовий критерій порівняння з нотифікаціями інших застосунків. Далі формується Intent для запуску MainActivity. Таким чином, натискання на нотифікацію не просто відкриває застосунок, а повертає користувача у вже існуючий екземпляр активності без створення дубліката.

Лістинг 3.10 – Реалізація логіки нотифікацій

```
public class NotificationReceiver extends BroadcastReceiver {
private static final String CHANNEL_ID =
     "medical_diary_notifications";
private static final String CHANNEL_NAME = "Медичні нагадування";
public void onReceive(Context context, Intent intent) {
NotificationManager notificationManager = (NotificationManager)
context.getSystemService(Context.NOTIFICATION_SERVICE);
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
         NotificationChannel channel = new NotificationChannel(
                    CHANNEL_ID, CHANNEL_NAME,
                    NotificationManager.IMPORTANCE_HIGH);
channel.setDescription("Сповіщення про вимірювання показників");
            channel.enableLights(true);
            channel.setLightColor(Color.RED);
            channel.enableVibration(true);
            if (notificationManager != null)
   notificationManager.createNotificationChannel(channel);}
        Intent mainIntent = new Intent(context, MainActivity.class);
        mainIntent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |
Intent.FLAG_ACTIVITY_SINGLE_TOP);
        int flags = PendingIntent.FLAG_UPDATE_CURRENT;
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
            flags |= PendingIntent.FLAG_IMMUTABLE;}
PendingIntent pendingIntent = PendingIntent.getActivity(
                              context, 0, mainIntent, flags);
        NotificationCompat.Builder builder = new
NotificationCompat.Builder(context, CHANNEL_ID)
.setContentTitle("Час вимірювань!")
.setContentText("Внесіть дані у медичний щоденник")
.setPriority(NotificationCompat.PRIORITY_HIGH)
.setCategory(NotificationCompat.CATEGORY_REMINDER)
.setContentIntent(pendingIntent).setAutoCancel(true)
.setDefaults(NotificationCompat.DEFAULT_ALL);
if (notificationManager != null)
    notificationManager.notify(1001, builder.build());}}
```

PendingIntent використовується як контейнер для Intent, який система може виконати від імені застосунку у майбутньому. Параметри прапорців підібрані з урахуванням сучасних вимог безпеки Android, наприклад, використовується FLAG_IMMUTABLE для нових версій ОС.

Створення сповіщення виконується через NotificationCompat.Builder. Тут задається заголовок і текст нагадування, рівень пріоритету, категорія REMINDER, а також автоматичне закриття нотифікації після взаємодії користувача. Прив’язка PendingIntent забезпечує перехід у застосунок при натисканні на сповіщення. Завершальним кроком є передача сформованої нотифікації до NotificationManager для відображення її у системній шторці.

Для коректної роботи нотифікацій застосунок має отримати дозволи:

<uses-permission name="android.permission.POST_NOTIFICATIONS"/>

<uses-permission name="android.permission.SCHEDULE_EXACT_ALARM"/>

### 3.5 Висновки розділу

У третьому розділі реалізовано повний програмний каркас комп’ютерної системи персонального медичного щоденника на платформі Android. Розглянуто та впроваджено ключові компоненти клієнтської частини застосунку, що забезпечують збереження, обробку та інтерпретацію медичних даних користувача.

Реалізовано графічний інтерфейс на основі one-Activity архітектури з використанням фрагментів. Такий підхід дозволив логічно створити навігацію, спростити керування життєвим циклом інтерфейсу та забезпечити коректну роботу при зміні орієнтації пристрою. Верстка фрагментів виконана з урахуванням адаптивності та підтримки альтернативних ресурсів layout-land, що гарантує зручність використання на пристроях з різними екранами.

Розроблено локальну базу даних на основі SQLite, яка відображає

предметну область застосунку та забезпечує збереження облікових даних, історії вимірювань, AI-інтерпретацій і користувацьких налаштувань. Структура бази даних є нормалізованою та логічно пов’язаною. Для роботи з даними реалізовано набір методів вставки та вибірки, які забезпечують перетворення між об’єктною моделлю та реляційним поданням.

Окрему увагу приділено модулю мережевої взаємодії та інтерпретації даних за допомогою зовнішнього AI-сервісу. Реалізовано клас AiService, який інкапсулює логіку формування HTTP-запитів, автентифікації, обробки відповідей та отримання текстових медичних висновків. Виклик AI-асистента відбувається в окремому потоці для розмежування головного і фонового потоків.

Забезпечено мінімально необхідний рівень безпеки обробки конфіденційних даних користувача. Паролі зберігаються у вигляді криптографічних хешів, а API-ключ для доступу до AI-сервісу зберігається у зашифрованому вигляді. Допоміжний клас Utils реалізує криптографічниі операції.

Реалізовано модуль нотифікацій, який використовує системні механізми Android для нагадування користувачу про необхідність внесення медичних вимірювань. Код базується на BroadcastReceiver, NotificationManager та каналах сповіщень, що забезпечує стабільну роботу на різних версіях операційної системи та інтеграцію з системними налаштуваннями користувача.

# 4 ВИПРОБУВАННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ ПЕРСОНАЛЬНОГО МЕДИЧНОГО ЩОДЕННИКА

## 4.1 Випробування користувацького інтерфейсу

Метою випробування користувацького інтерфейсу є перевірка відповідності графічної оболонки функціональним вимогам, контроль логіки переходів між екранами та підтвердження коректності відображення даних. Випробування відбуваються за допомогою вбудованого в Android Studio емулятора мобільного

пристрою. Розглянемо основні сценарії, що демонструють працездатність розробленої системи.

Сценарій 1 – початкова взаємодія та реєстрація нового користувача. Опис: перевірка першого запуску застосунку та створення облікового запису з урахуванням антропометричних даних та API-ключа:

- запуск застосунку та відображення екрана вітання (рис. 4.1);
- перехід до форми реєстрації;
- заповнення полів: логін, пароль, вік, стать, вага та ключ доступу до Hugging Face API;
- очікуваний результат: елементи керування доступні, форма валідує введені дані.

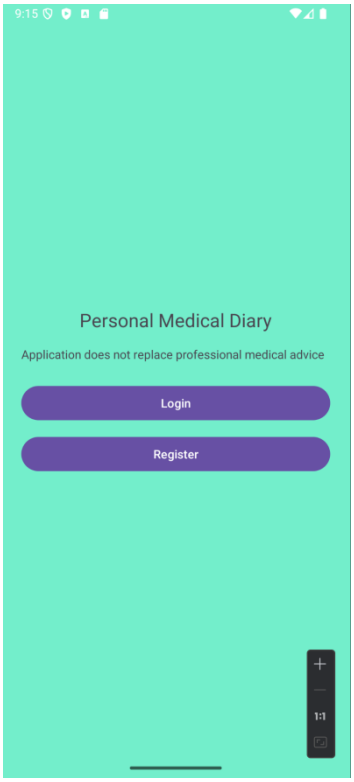


Рисунок 4.1 – Екран вітання

При натисканні на текстовому полі Get Hugging Face API key користувач переходить в браузер, де відкривається офіційний сайт реєстрації в AI-сервісі Hugging Face (рис. 4.2). Звідти користувач копіює свій унікальний ключ, повертається в медичний щоденник та вставляє ключ у відповідне поле (рис. 4.3).

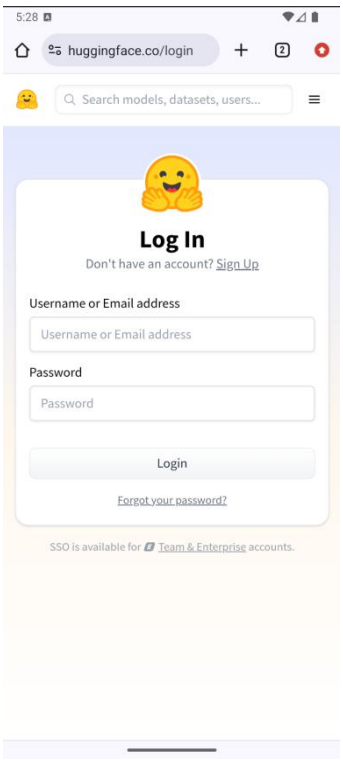


Рисунок 4.2 – Перехід на офіційний сайт Hugging Face

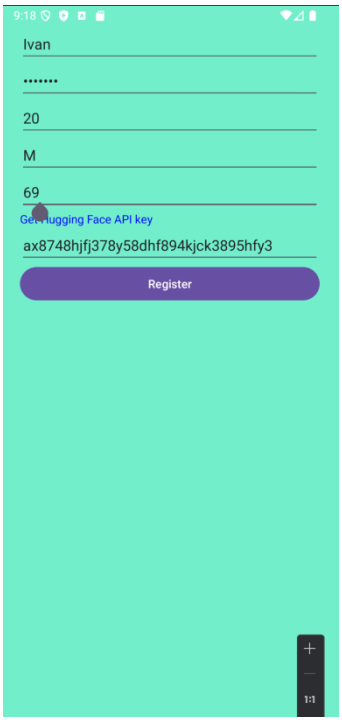


Рисунок 4.3 – Екран реєстрації користувача

Сценарій 2 – автентифікація користувача та головне меню. Опис: перевірка входу в систему за створеним обліковим записом:

- введення логіна «Ivan» та пароля на екрані авторизації (рис. 4.4);

- натискання кнопки «Login»;
- очікуваний результат: перехід до головного меню з чіткою структурою навігації (рис. 4.5).

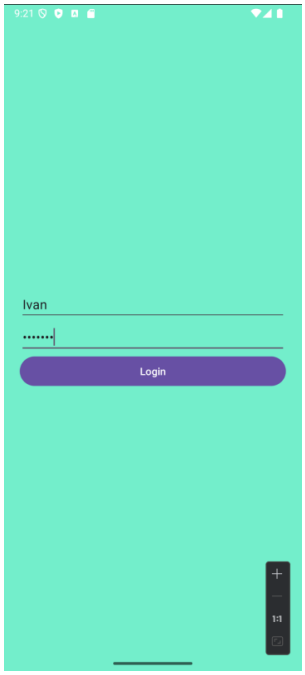


Рисунок 4.4 – Екран реєстрації користувача

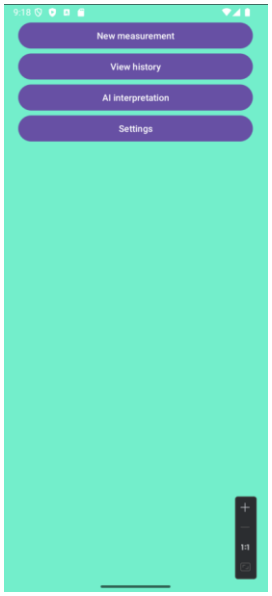


Рисунок 4.5 – Екран головного меню

При введенні некоректних даних облікового запису, наприклад, неіснуючого користувача чи пароля, екран входу не має пропускати далі в

застосунок і виводити відповідне повідомлення (рис. 4.6)

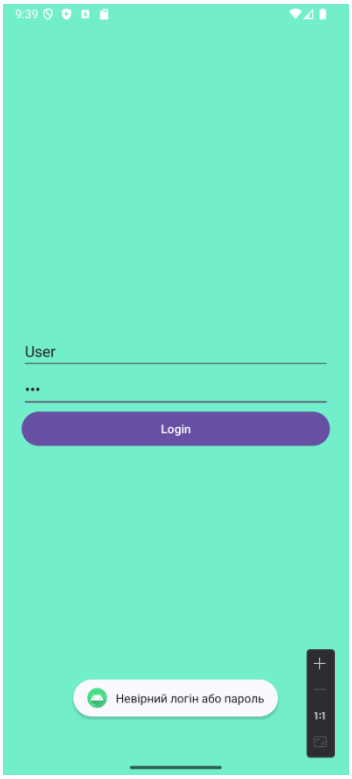


Рисунок 4.6 – Введені некоректні дані облікового запису

Сценарій 3 – введення нових показників (рис. 4.7). Опис: перевірка екрану додавання вимірювань тиску, пульсу та рівня цукру:

- вибір пункту «New measurement»;
- введення показників: Systolic, Diastolic, Pulse, Glucose;
- збереження даних;
- очікуваний результат: поля вводу мають відповідні підказки (hints), кнопка «Save» ініціює запис у локальну базу даних SQLite.

Сценарій 4 – перегляд історії та управління даними. Опис: відображення списку збережених вимірювань, витягнутих з бази даних. Користувач обирає пункт меню «View history». Очікуваний результат: дані відображаються у зручному списковому форматі (рис. 4.8) з можливістю подальшої фільтрації або переходу до інтерпретації.

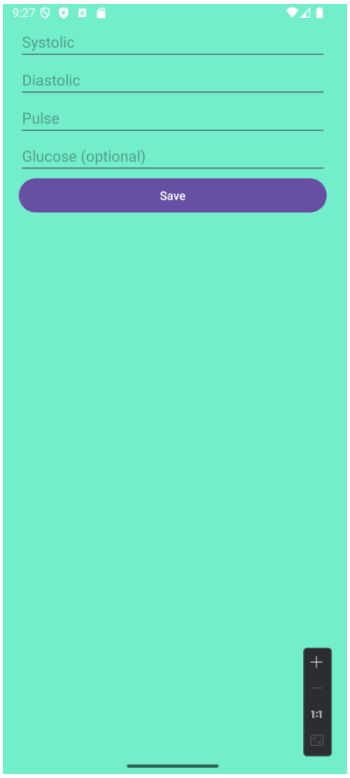


Рисунок 4.7 – Екран введення нових показників

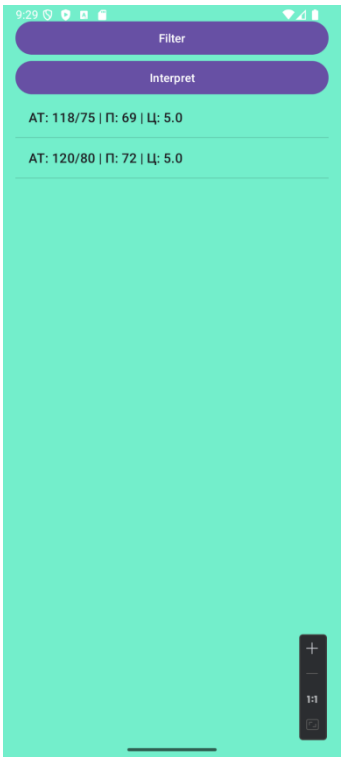


Рисунок 4.8 – Екран перегляду показників

Сценарій 5 – налаштування сповіщень про нагадування здійснити вимір (рис. 4.9). Опис: перевірка можливості конфігурації щоденних нагадувань:

- перехід до розділу «Settings»;
- вибір часу через стандартний TimePicker;
- активація перемикача сповіщень;
- очікуваний результат – коректне відображення діалогового вікна вибору часу, збереження налаштувань у базі даних.

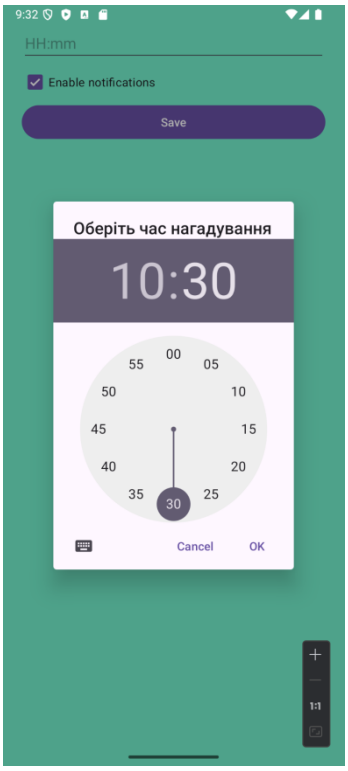


Рисунок 4.9 – Екран налаштування нагадувань

## 4.2 Випробування взаємодії з зовнішнім AI-сервісом

Взаємодія персонального медичного щоденника з AI-моделлю для інтерпретації виміряних показників – це критичний функціонал системи. В процесі випробування цього функціоналу ми перевіримо мережеву взаємодію з сервером AI-моделі, коректну передачу ключа доступу, його попереднє дешифрування та правильність відображення результату.

Сценарій 1 – інтелектуальний аналіз даних (AI Interpretation). Опис:

перевірка взаємодії з LLM-моделлю (Llama 3.1) через API:

- натискання кнопки «Interpret» або «AI interpretation» на відповідному екрані;
- очікування обробки, тобто візуальний зворотний зв'язок (рис. 4.10);
- отримання та відображення розгорнутого висновку;
- очікуваний результат – система відображає статус «Аналізую дані...», після чого виводить структурований текст з аналізом тиску, пульсу та цукру, враховуючи норми та антропометрію (рис. 4.11).

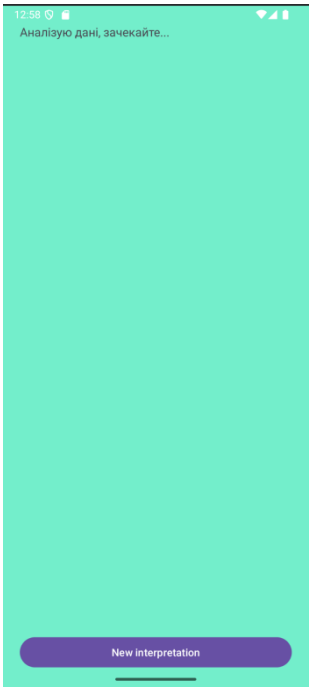


Рисунок 4.10 – Очікування відповіді AI-моделі

Сценарій 2 – перегортання екрану. Опис: перевірка коректності відображення даних при зміні орієнтації екрану з портретної на альбомну або навпаки:

- перегортання екрану на 90 градусів;
- збереження вже отриманої інтерпретації;
- система не надсилає повторний запит до AI-моделі;
- очікуваний результат – дані у фрагменті не знищуються, а зберігаються та відображаються в іншій орієнтації (рис. 4.12).

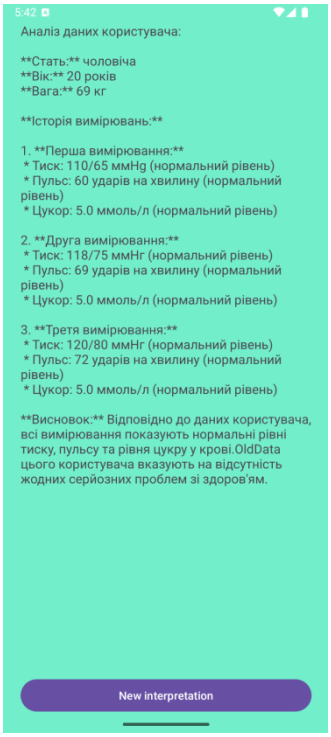


Рисунок 4.11 – Екран інтерпретації від AI-моделі

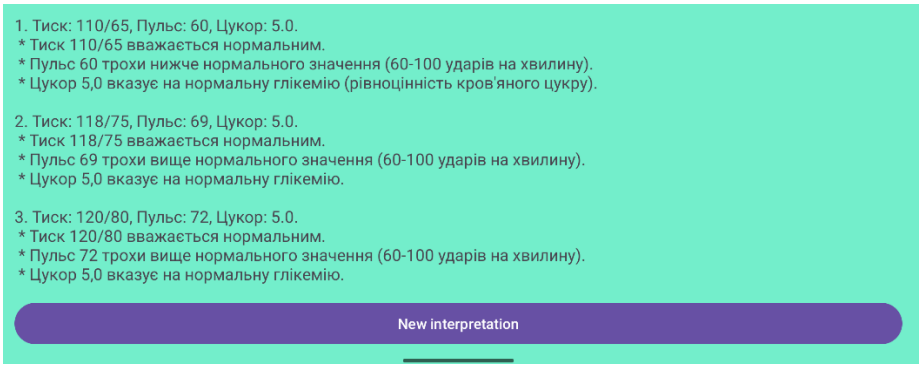


Рисунок 4.12 – Перегортання екрану зі збереженням інтерпретації

Сценарій 3 – обробка помилок зв’язку з AI-моделлю та мережевих виключень. Мета: перевірити стійкість застосунку до помилок на стороні сервера (Hugging Face API) або проблем з мережевим з’єднанням, а також коректність інформування користувача про технічні несправності. Опис: під час формування запиту на інтелектуальну інтерпретацію виникає помилка обробки на стороні API або закінчується ліміт використання ресурсу. Система повинна перехопити виключення та вивести код помилки для подальшої діагностики:

- перейти до розділу «AI interpretation» або натиснути «Interpret» у списку

вимірювань;

- ініціювати запит до сервера при недійсних параметрах доступу (API key) або за наявності проблем на стороні хоста;
- очікувати на відповідь сервера;
- очікуваний результат – застосунок не завершує роботу аварійно (не «вилітає»). Замість результату інтерпретації на екрані з'являється текстове повідомлення з описом проблеми та конкретним кодом відповіді сервера (рис. 4.13). Кнопка «New interpretation» залишається доступною для повторної спроби після усунення причин помилки.

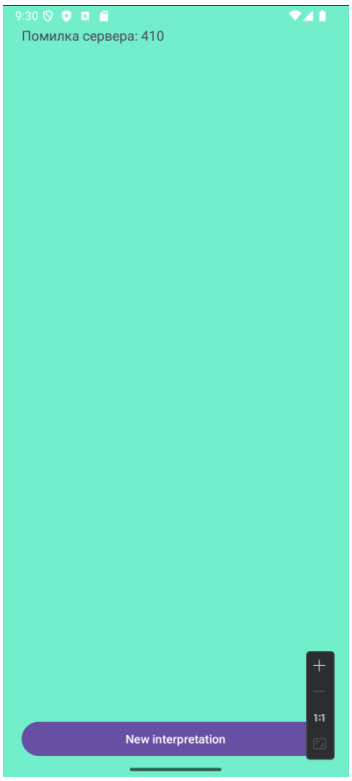


Рисунок 4.13 – Помилка зв'язку з AI-моделлю

Важливою складовою надійності системи є механізм обробки помилок при роботі з зовнішнім API. На Рис. 4.13 продемонстровано випадок отримання коду 410, що в контексті роботи з нейромережевими моделями через HTTP-протокол може свідчити про недійсність ресурсу або завершення сесії. Реалізований інтерфейс дозволяє користувачу ідентифікувати проблему за кодом, що полегшує технічну підтримку та перевірку правильності введеного API-ключа в

налаштуваннях.

### 4.3 Випробування бази даних

Для підтвердження коректності роботи зберігання даних було проведено випробування локальної бази даних SQLite. Метою цього етапу є верифікація відповідності даних, що відображаються в інтерфейсі користувача, фактичним записам у файлі бази даних. Доступ до файлу бази даних MedDiary.db було отримано за допомогою інструменту Android Studio App Inspection. Це дозволило проаналізувати фізичну структуру таблиць та перевірити наявність зв'язків між ними за допомогою відомого інструменту SQLiteViewer [25]. Перелік таблиць бази даних наведено на рис. 4.15.

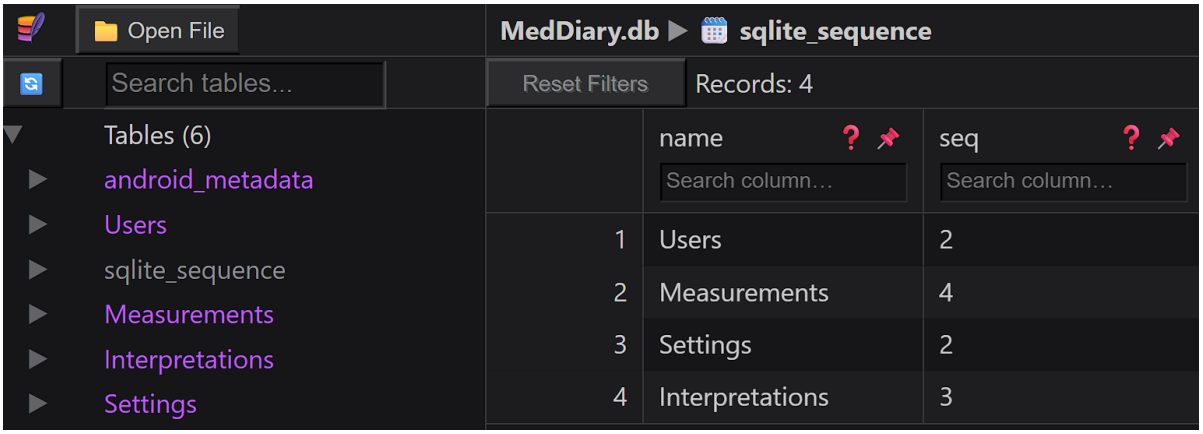


Рисунок 4.15 – Перелік таблиць бази даних MedDiary.db

На рис. 4.16 відображено структуру та вміст таблиці облікових записів Users. Кожен рядок містить унікальний ідентифікатор _id, логін користувача, хешований пароль та зашифрований API-ключ. Таблиця підтверджує успішне збереження антропометричних даних (age, sex, weight), які згодом використовуються AI-моделлю. Поле sex коректно зберігає символьні значення ('M'/'F'), а weight - цілочисельні.

MedDiary.db ▶ Users

Reset Filters | Records: 2

| | _id | login | passwordHash | encryptedApiKey | age | sex | weight | createdAt |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | user | test | hf_IWPTpxuGbbLAwg... | 20 | M | 70 | 1768979927693 |
| 2 | 2 | Ivan | 8bb0cf6eb9b17d0f... | XLYS6FWC13bSfyPxYw... | 20 | M | 69 | 1768979927710 |

Рисунок 4.16 – Вміст таблиці Users

Перший користувач був створений з міркувань тестування працездатності, а другий - Ivan, є основним користувачем системи. Випробування сценаріїв використання системи відбувалося саме з його допомогою.

Таблиця Measurements (вимірювання) демонструє історію внесених біометричних показників (рис. 4.17). Кожен запис чітко прив'язаний до ідентифікатора користувача через зовнішній ключ idUser. Поле measuredAt зберігає часові мітки у форматі Unix Timestamp.

MedDiary.db ▶ Measurements

Reset Filters | Records: 4

| | _id | idUser | measuredAt | systolic | diastolic | pulse | glucose |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1674259200000 | 120 | 80 | 70 | 5 |
| 2 | 2 | 2 | 1768980558136 | 120 | 80 | 72 | 5 |
| 3 | 3 | 2 | 1768980592688 | 118 | 75 | 69 | 5 |
| 4 | 4 | 2 | 1768993801248 | 110 | 65 | 60 | 5 |

Рисунок 4.17 – Вміст таблиці Measurements

Користувач idUser = 1 вводив одні виміри, а користувач idUser = 2 (Ivan) вводив три вимірювання в базу.

На рис. 4.18 видно результати роботи інтелектуального модуля, записані в таблицю Interpretations. Кожен запис містить повний текст відповіді від AI-моделі. Наявність записів у цій таблиці підтверджує, що після кожного успішного запиту до API, дані не лише виводяться на екран, а й фіксуються в базі для подальшого перегляду в режимі офлайн. Поля periodStart та periodEnd демонструють часовий

проміжок, за який проводився аналіз.

MedDiary.db ▶ Interpretations

Reset Filters | Records: 3

| | | idUser | createdAt | periodStart | periodEnd | interpretationText |
|---|---|---|---|---|---|---|
| | Search | Search column | Search column… | Search column… | Search column… | Search column… |
| 1 | 1 | 2 | 1769009515222 | 0 | 1769009515222 | Аналізуючи дані, я можу зробити наступні висновки: 1. Но... |
| 2 | 2 | 2 | 1769010082194 | 0 | 1769010082194 | Мій колега, під час аналізу даних я побачив кілька важлив... |
| 3 | 3 | 2 | 1769010164342 | 0 | 1769010164342 | Аналіз даних користувача: **Стать:** чоловіча **Вік:** 20 р... |

Рисунок 4.18 – Вміст таблиці Interpretations

Таблиця Settings (налаштування) показує конфігураційні параметри для кожного користувача (рис. 4.19). Запис містить часовий рядок reminderTime, наприклад, «13:02», та булевий прапор notificationsEnabled, збережений як 0 або 1. Це підтверджує, що стан нагадувань синхронізовано з налаштуваннями інтерфейсу.

MedDiary.db ▶ Settings

Reset Filters | Records: 2

| | _id | idUser | reminderTime | notificationsEnabled |
|---|---|---|---|---|
| | Search column… | Search column | Search column… | Search column… |
| 1 | 1 | 1 | 08:00 | 1 |
| 2 | 2 | 2 | 13:02 | 1 |

Рисунок 4.19 – Вміст таблиці Settings

Аналіз внутрішньої структури бази даних підтвердив повну цілісність даних. Всі типи даних обрані правильно. Записи системи виконуються в базу коректно.

## 4.4 Висновки розділу

У ході випробувань комп’ютерної системи персонального медичного

щоденника підтверджено коректність реалізації всіх заявлених функціональних та нефункціональних вимог. Перевірка користувацького інтерфейсу показала логічну та послідовну навігацію між екранами, коректну роботу форм введення, механізмів валідації даних і зрозумілу взаємодію користувача з основними сценаріями застосунку. Інтерфейс забезпечує відображення інформації, коректно реагує на помилкові дії та не допускає переходів у некоректні стани.

Випробування сценаріїв реєстрації та автентифікації підтвердили правильність створення облікових записів, збереження антропометричних параметрів та API-ключа користувача, а також захист доступу до персональних даних.

Випробування взаємодії з зовнішнім AI-сервісом Hugging Face підтвердили працездатність мережевого модуля, коректну передачу та використання API-ключа, а також правильне відображення результатів інтелектуальної інтерпретації медичних показників. Застосунок забезпечує зворотний зв’язок під час очікування відповіді, зберігає отримані результати локально та відновлює їх при зміні орієнтації екрану без повторних запитів до сервера. Застосунок не завершує роботу аварійно та надає користувачу повідомлення з кодами помилок.

Аналіз структури та вмісту бази даних підтвердив цілісність і коректність типів даних у всіх таблицях. Випробування функціоналу введення, збереження та перегляду медичних показників засвідчило правильність роботи з локальною базою даних SQLite.

# ВИСНОВКИ

В ході виконання дипломної роботи було розроблено комп’ютерну систему персонального медичного щоденника, що відповідає поставленій меті – створенню інструменту для моніторингу основних показників здоров’я з інтелектуальною підтримкою прийняття рішень. Ефективність медичного самоконтролю було підвищено за рахунок автоматизації процесів збереження даних та впровадження модуля інтерпретації на основі великих мовних моделей, що дозволяє користувачу отримувати персоналізовані рекомендації у режимі реального часу.

У першому розділі проведено системний аналіз предметної області та існуючих рішень для моніторингу стану здоров’я. Виявлено, що більшість аналогів перевантажені зайвим функціоналом і часто не надають аналітики зібраних даних. Це обґрунтувало необхідність розробки спеціалізованого мобільного застосунку, який би поєднував введення параметрів із сучасними методами обробки даних за допомогою штучного інтелекту. Обґрунтовано вибір технологічного стеку. Для розробки обрано платформу Android, що забезпечує високу доступність системи. Використано реляційну базу даних SQLite. Для реалізації інтелектуальних функцій обрано модель Llama 3.1 з безкоштовним доступом через Hugging Face API.

У другому розділі спроектовано архітектуру системи та структуру бази даних, що складається з чотирьох ключових таблиць: Users, Measurements, Interpretations та Settings. Розроблено логічну схему навігації між фрагментами та побудовано UML-діаграми, за допомогою яких можна реалізувати систему. Виконано прототипування системи. Окрему увагу приділено безпеці даних: впроваджено парольний захист та алгоритм шифрування AES для захисту персональних API-ключів користувачів.

У третьому розділі здійснено програмну реалізацію системи. Розроблено адаптивний інтерфейс користувача. Реалізовано сервіс взаємодії зі штучним

інтелектом (AiService), що враховує не лише поточні вимірювання, а й антропометричні дані користувача (вік, вага, стать) для формування точніших висновків. Реалізовано систему фонових нагадувань за допомогою AlarmManager та NotificationChannel, що сприяє регулярності ведення щоденника.

В четвертому розділі проведено комплексне випробування графічного інтерфейсу, бази даних та модуля мережевої взаємодії. Випробування підтвердили цілісність збереженої інформації та стійкість системи до помилок мережевого з’єднання. Перевірка збереження стану при перевороті пристрою підтвердила надійність запропонованої реалізації.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Topol E. Deep Medicine: How Artificial Intelligence Can Make Healthcare Human Again. New York: Basic Books, 2019. 400 с.

2. WHO guideline on self-care interventions for health and well-being. Geneva: WHO Press, 2021. 168 с.

3. Rhode Ghislaine, Nguewo Ngassam. Analysis of the design, evaluation and the adoption of a personal health record: Business administration. Université Montpellier, 2021. 196 с.

4. Syed-Abdul S., Zhu X., Fernandez-Luque L. Digital Health: Mobile and Wearable Devices for Participatory Health Applications. Elsevier, 2020. 232 с.

5. Taulli T. Artificial Intelligence Basics: A Non-Technical Introduction. New York: Apress, 2019. 187 с.

6. WHO guideline: recommendations on digital interventions for health system strengthening. Geneva: World Health Organization, 2019. 124 с.

7. Ismail N. I., Ahmed N. H., Khan A. S. Usability testing of mobile health applications: A systematic review. International Journal of Advanced Computer Science and Applications. 2020. Vol. 11, No. 9. С. 343–351.

8. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner’s Approach. 9th ed. New York: McGraw-Hill Education, 2020. 960 с.

9. National Institute of Standards and Technology. Digital Identity Guidelines (SP 800-63-4). URL: https://csrc.nist.gov/pubs/sp/800/63/4/final (дата звернення: 27.01.2026).

10. Android Developers. Support different display sizes. URL: https://developer.android.com/guide/topics/large-screens/support-different-screen-sizes (дата звернення: 28.01.2026).

11. Dunn M., Lewis S. Native Mobile Development: A Cross-Reference for iOS and Android Native Programming. Sebastopol: O’Reilly Media, 2019. 400 с.

12. Android Developers. Get started with Android – Android Developers guide. URL: https://developer.android.com/guide (дата звернення: 01.02.2026).

13. JetBrains. Kotlin docs. URL: https://kotlinlang.org/docs/home.html (дата звернення: 02.02.2026).

14. Phillips B., Stewart C., Marsicano K. Android Programming: The Big Nerd Ranch Guide. Sebastopol: O'Reilly Media, 2020. 624 с.

15. Android Developers. Processes and threads overview. URL: https://developer.android.com/guide/components/processes-and-threads (дата звернення: 04.02.2026).

16. Farley D. Modern Software Engineering: Doing What Works to Build Better Software Faster. Boston: Addison-Wesley, 2021. 256 с.

17. Android Developers. Better performance through threading. URL: https://developer.android.com/topic/performance/threads (дата звернення: 05.02.2026).

18. Fielding R. T., Nottingham M., Reschke J. HTTP Semantics. RFC 9110. Internet Engineering Task Force, 2022. 327 с. URL: https://www.rfc-editor.org/rfc/rfc9110.html (дата звернення: 05.02.2026).

19. OWASP Foundation. API Security Top 10 2023. URL: https://owasp.org/www-project-api-security/ (дата звернення: 06.02.2026).

20. Hugging Face. Hugging Face Hub Documentation. URL: https://huggingface.co/docs/hub/index (дата звернення: 06.02.2026).

21. Hugging Face. Inference API Documentation. URL: https://huggingface.co/docs/api-inference/index (дата звернення: 06.02.2026).

22. Android Developers. Save data in a local database using Room. URL: https://developer.android.com/training/data-storage/room (дата звернення: 07.02.2026).

23. MongoDB. Atlas Device SDK for Kotlin (Realm). URL: https://www.mongodb.com/docs/atlas/device-sdks/sdk/kotlin/ (дата звернення: 07.02.2026).

24. ObjectBox. ObjectBox for Android/Java Documentation. URL: https://docs.objectbox.io/getting-started (дата звернення: 07.02.2026).

25. SQLiteViewer. URL: https://sqliteviewer.app/ (дата звернення: 18.02.2026).

# ДОДАТОК А
# ЛІСТИНГИ МОДУЛІВ СИСТЕМИ

Лістинг А.1 – Файл WelcomeFragment.java

```
public class WelcomeFragment extends Fragment {
    private Button btnLogin, btnRegister;

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
@Nullable ViewGroup container,
                             @Nullable Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_welcome,
container, false);

        btnLogin = view.findViewById(R.id.btnLogin);
        btnRegister = view.findViewById(R.id.btnRegister);

        btnLogin.setOnClickListener(v -> {
            if (getActivity() instanceof MainActivity) {
                ((MainActivity) getActivity()).showLoginFragment();
            }
        });

        btnRegister.setOnClickListener(v -> {
            if (getActivity() instanceof MainActivity) {
                ((MainActivity)
getActivity()).showRegistrationFragment();
            }
        });

        return view;
    }
}
```

Лістинг А.2 – Файл RegistrationFragment.java

```
public class RegistrationFragment extends Fragment {

    private EditText etLogin, etPassword, etAge, etSex, etWeight,
etApiKey;
    private Button btnRegisterConfirm;
    private TextView tvApiKeyLink;

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
@Nullable ViewGroup container,
```

```
        @Nullable Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_registration,
container, false);

        etLogin = view.findViewById(R.id.etLogin);
        etPassword = view.findViewById(R.id.etPassword);
        etAge = view.findViewById(R.id.etAge);
        etSex = view.findViewById(R.id.etSex);
        etWeight = view.findViewById(R.id.etWeight);
        etApiKey = view.findViewById(R.id.etApiKey);
        tvApiKeyLink = view.findViewById(R.id.tvApiKeyLink);
        tvApiKeyLink.setOnClickListener(v -> {
            Intent intent = new Intent(Intent.ACTION_VIEW,
Uri.parse("https://huggingface.co/login"));
            startActivity(intent);
        });

        btnRegisterConfirm =
view.findViewById(R.id.btnRegisterConfirm);

        btnRegisterConfirm.setOnClickListener(v -> {
            String login = etLogin.getText().toString().trim();
            String pass = etPassword.getText().toString().trim();
            String age = etAge.getText().toString().trim();
            String sex = etSex.getText().toString().trim();
            String weight = etWeight.getText().toString().trim();
            String apiKey = etApiKey.getText().toString().trim();

            if (login.isEmpty() || pass.isEmpty() ||
apiKey.isEmpty() || sex.isEmpty()) {
                Toast.makeText(getContext(), "Логін, пароль, стать
та API ключ обов'язкові", Toast.LENGTH_SHORT).show();
                return;
            }

            if (getActivity() instanceof MainActivity) {
                ((MainActivity) getActivity()).registerUser(
                        login, pass,
                        Integer.parseInt(age.isEmpty() ? "0" : age),
                        sex.toUpperCase().charAt(0), // 'M' або 'F'
                        Integer.parseInt(weight.isEmpty() ? "0" :
weight),
                        apiKey
                );
            }
        });
        return view;
    }
}
```

Лістинг А.3 – Файл LoginFragment.java

```
public class LoginFragment extends Fragment {
```

```
    private EditText etLogin, etPassword;
    private Button btnLoginConfirm;

    public LoginFragment(){}

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
@Nullable ViewGroup container,
                             @Nullable Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_login,
container, false);

        etLogin = view.findViewById(R.id.etLogin);
        etPassword = view.findViewById(R.id.etPassword);
        btnLoginConfirm = view.findViewById(R.id.btnLoginConfirm);

        btnLoginConfirm.setOnClickListener(v -> {
            if (getActivity() instanceof MainActivity) {
                ((MainActivity) getActivity()).loginUser(
                        etLogin.getText().toString(),
                        etPassword.getText().toString()
                );
            }
        });

        return view;
    }
}
```

Лістинг А.4 – Файл MainMenu.java

```
public class MainMenuFragment extends Fragment {

    private Button btnNewMeasurement, btnViewHistory,
btnGetInterpretation, btnSettings;

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
@Nullable ViewGroup container,
                             @Nullable Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_main_menu,
container, false);

        btnNewMeasurement =
view.findViewById(R.id.btnAddMeasurement);
        btnViewHistory = view.findViewById(R.id.btnViewHistory);
        btnGetInterpretation =
view.findViewById(R.id.btnInterpretation);
        btnSettings = view.findViewById(R.id.btnSettings);
```

```
        btnNewMeasurement.setOnClickListener(v -> {
            if (getActivity() instanceof MainActivity) {
                ((MainActivity)
getActivity()).showMeasurementInputFragment();
            }
        });

        btnViewHistory.setOnClickListener(v -> {
            if (getActivity() instanceof MainActivity) {
                ((MainActivity)
getActivity()).showMeasurementsListFragment();
            }
        });

        btnGetInterpretation.setOnClickListener(v -> {
            if (getActivity() instanceof MainActivity) {
                ((MainActivity)
getActivity()).showInterpretationFragment();
            }
        });

        btnSettings.setOnClickListener(v -> {
            if (getActivity() instanceof MainActivity) {
                ((MainActivity)
getActivity()).showSettingsFragment();
            }
        });

        return view;
    }
}
```

Лістинг А.5 – Файл MeasurementInputFragment.java

```
public class MeasurementInputFragment extends Fragment {

    private EditText etSystolic, etDiastolic, etPulse, etGlucose;
    private Button btnSaveMeasurement;

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
@Nullable ViewGroup container,
                             @Nullable Bundle savedInstanceState) {
        View view =
inflater.inflate(R.layout.fragment_measurement_input, container,
false);

        etSystolic = view.findViewById(R.id.etSystolic);
        etDiastolic = view.findViewById(R.id.etDiastolic);
        etPulse = view.findViewById(R.id.etPulse);
        etGlucose = view.findViewById(R.id.etGlucose);
        btnSaveMeasurement =
```

```
view.findViewById(R.id.btnSaveMeasurement);

        btnSaveMeasurement.setOnClickListener(v -> {
            String sSys = etSystolic.getText().toString();
            String sDia = etDiastolic.getText().toString();
            String sPul = etPulse.getText().toString();
            String sGlu = etGlucose.getText().toString();

            if (sSys.isEmpty() || sDia.isEmpty() || sPul.isEmpty()
|| sGlu.isEmpty()) {
                Toast.makeText(getContext(), "Заповніть усі
показники", Toast.LENGTH_SHORT).show();
                return;
            }

            if (getActivity() instanceof MainActivity) {
                ((MainActivity) getActivity()).addMeasurement(
                        Integer.parseInt(sSys),
                        Integer.parseInt(sDia),
                        Integer.parseInt(sPul),
                        Float.parseFloat(sGlu)
                );
            }
        });

        return view;
    }
}
```

Лістинг А.6 – Файл MeasurementsListFragment.java

```
public class MeasurementsListFragment extends Fragment {

    private ListView lvMeasurements;
    private Button btnFilter, btnInterpret;

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
     @Nullable ViewGroup container,
     @Nullable Bundle savedInstanceState) {
        View view =
inflater.inflate(R.layout.fragment_measurements_list, container,
false);

        lvMeasurements = view.findViewById(R.id.lvMeasurements);
        btnFilter = view.findViewById(R.id.btnFilter);
        btnInterpret = view.findViewById(R.id.btnInterpret);
        loadData();
        btnFilter.setOnClickListener(v -> loadData());

        btnInterpret.setOnClickListener(v -> {
            if (getActivity() instanceof MainActivity) {
```

```
                MainActivity activity = (MainActivity)
getActivity();
                activity.showInterpretationFragment();
                activity.requestAiInterpretation();
            }
        });

        return view;
    }

    private void loadData() {
        if (getActivity() instanceof MainActivity) {
            List<Measurement> measurements = ((MainActivity)
getActivity()).getCurrentUserMeasurements();
            List<String> displayList = new ArrayList<>();
            for (Measurement m : measurements) {
                displayList.add(String.format("АТ: %d/%d | П: %d |
Ц: %.1f",
                        m.getSystolic(), m.getDiastolic(),
m.getPulse(), m.getGlucose()));
            }
            ArrayAdapter<String> adapter = new
ArrayAdapter<>(getContext(),
                    android.R.layout.simple_list_item_1,
displayList);
            lvMeasurements.setAdapter(adapter);
        }
    }
}
```

Лістинг А.7 – Файл InterpretationFragment.java

```
public class InterpretationFragment extends Fragment {

    private TextView tvInterpretationResult;
    private Button btnNewInterpretation;
    private String interpretationText;

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
     @Nullable ViewGroup container,
     @Nullable Bundle savedInstanceState) {
        View view =
inflater.inflate(R.layout.fragment_interpretation, container,
false);
        tvInterpretationResult =
view.findViewById(R.id.tvInterpretationResult);
        btnNewInterpretation =
view.findViewById(R.id.btnNewInterpretation);

        if (savedInstanceState != null) {
            interpretationText =
```

```
savedInstanceState.getString("saved_text");
        }

        if (interpretationText == null && getActivity() instanceof
MainActivity) {
            interpretationText = ((MainActivity)
getActivity()).getLastAiInterpretation();
        }

        if (interpretationText != null) {
            tvInterpretationResult.setText(interpretationText);
        } else {
            tvInterpretationResult.setText("Нема збережених даних.
Натисніть кнопку для аналізу.");
        }

        btnNewInterpretation.setOnClickListener(v -> {
            if (getActivity() instanceof MainActivity) {
                tvInterpretationResult.setText("Аналізую дані,
зачекайте...");
                ((MainActivity)
getActivity()).requestAiInterpretation();
            }
        });
        return view;
    }

    @Override
    public void onSaveInstanceState(@NonNull Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putString("saved_text", interpretationText);
    }

    public void setInterpretationText(String text) {
        this.interpretationText = text;
        if (tvInterpretationResult != null) {
            tvInterpretationResult.setText(text);
        }

    }
}
```

Лістинг А.8 – Файл SettingsFragment.java

```
public class SettingsFragment extends Fragment {

    private EditText etReminderTime;
    private CheckBox cbNotifications;
    private Button btnSaveSettings;

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
```

```
@Nullable ViewGroup container,
                             @Nullable Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_settings,
container, false);

        etReminderTime = view.findViewById(R.id.etReminderTime);
        cbNotifications = view.findViewById(R.id.cbNotifications);
        btnSaveSettings = view.findViewById(R.id.btnSaveSettings);

        etReminderTime.setFocusable(false); // Щоб клавіатура не
вилазила
        etReminderTime.setOnClickListener(v -> {
            Calendar mcurrentTime = Calendar.getInstance();
            int hour = mcurrentTime.get(Calendar.HOUR_OF_DAY);
            int minute = mcurrentTime.get(Calendar.MINUTE);
            TimePickerDialog mTimePicker;
            mTimePicker = new TimePickerDialog(getContext(),
(timePicker, selectedHour, selectedMinute) -> {
                etReminderTime.setText(String.format("%02d:%02d",
selectedHour, selectedMinute));
            }, hour, minute, true);
            mTimePicker.setTitle("Оберіть час нагадування");
            mTimePicker.show();
        });

        btnSaveSettings.setOnClickListener(v -> {
            if (getActivity() instanceof MainActivity) {
                ((MainActivity) getActivity()).saveSettings(
                        etReminderTime.getText().toString(),
                        cbNotifications.isChecked()
                );
            }
        });

        return view;
    }
}
```

Лістинг А.9 – Файл MainActivity.java

```
public class MainActivity extends AppCompatActivity {

    private DatabaseHelper dbHelper;
    private Handler mainHandler;
    private User currentUser;
    private String currentPassword;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        dbHelper = new DatabaseHelper(this);
```

```
        mainHandler = new Handler(Looper.getMainLooper());

        if (savedInstanceState != null) {
            long userId = savedInstanceState.getLong("USER_ID", -1);
            if (userId != -1) {
                currentUser = dbHelper.getUserById((int) userId);
                currentPassword =
savedInstanceState.getString("USER_PASS");
            }
        } else {
            showWelcomeFragment();
        }
    }

    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        if (currentUser != null) {
            outState.putLong("USER_ID", currentUser.getId());
            outState.putString("USER_PASS", currentPassword);
        }
    }

    public void replaceFragment(Fragment fragment, boolean
addToBackStack) {
        FragmentTransaction ft =
getSupportFragmentManager().beginTransaction();
        ft.replace(R.id.fragmentContainer, fragment);
        if (addToBackStack) ft.addToBackStack(null);
        ft.commit();
    }

    public void showWelcomeFragment() { replaceFragment(new
WelcomeFragment(), false); }
    public void showRegistrationFragment() { replaceFragment(new
RegistrationFragment(), true); }
    public void showLoginFragment() { replaceFragment(new
LoginFragment(), true); }
    public void showMainMenuFragment() { replaceFragment(new
MainMenuFragment(), false); }
    public void showMeasurementInputFragment() { replaceFragment(new
MeasurementInputFragment(), true); }
    public void showMeasurementsListFragment() { replaceFragment(new
MeasurementsListFragment(), true); }
    public void showSettingsFragment() { replaceFragment(new
SettingsFragment(), true); }
    public void showInterpretationFragment() { replaceFragment(new
InterpretationFragment(), true); }

    public void registerUser(String login, String password, int age,
char sex, int weight, String apiKey) {
        String hash = Utils.hashPassword(password);
        String encryptedKey = Utils.encryptAES(apiKey, password);
```

```
        User user = new User();
        user.setLogin(login);
        user.setPasswordHash(hash);
        user.setAge(age);
        user.setSex(sex);
        user.setWeight(weight);
        user.setEncryptedApiKey(encryptedKey);

        if (dbHelper.insertUser(user)) {
            this.currentUser = dbHelper.getUserByLogin(login);
            this.currentPassword = password;
            showMainMenuFragment();
        } else {
            Toast.makeText(this, "Помилка реєстрації",
Toast.LENGTH_SHORT).show();
        }
    }

    public void loginUser(String login, String password) {
        User user = dbHelper.getUserByLogin(login);
        if (user != null &&
Utils.hashPassword(password).equals(user.getPasswordHash())) {
            this.currentUser = user;
            this.currentPassword = password;
            showMainMenuFragment();
        } else {
            Toast.makeText(this, "Невірний логін або пароль",
Toast.LENGTH_SHORT).show();
        }
    }

    public void addMeasurement(int systolic, int diastolic, int
pulse, float glucose) {
        if (currentUser == null) return;

        Measurement m = new Measurement();
        m.setUserId(currentUser.getId());
        m.setSystolic(systolic);
        m.setDiastolic(diastolic);
        m.setPulse(pulse);
        m.setGlucose(glucose);
        m.setMeasuredAt(System.currentTimeMillis());

        if (dbHelper.insertMeasurement(m)) {
            Toast.makeText(this, "Дані збережено",
Toast.LENGTH_SHORT).show();
            getSupportFragmentManager().popBackStack();
        }
    }

    public void requestAiInterpretation() {
        if (currentUser == null || currentPassword == null) return;
```

```
        new Thread(() -> {
            try {
                List<Measurement> measurements =
dbHelper.getMeasurementsForPeriod(
                        (int) currentUser.getId(), 0,
System.currentTimeMillis());

                StringBuilder sb = new StringBuilder();
                sb.append("Дані користувача:\n");
                sb.append(String.format("- Стать: %s\n",
currentUser.getSex() == 'M' ? "чоловіча" : "жіноча"));
                sb.append(String.format("- Вік: %d років\n",
currentUser.getAge()));
                sb.append(String.format("- Вага: %d кг\n\n",
currentUser.getWeight()));

                sb.append("Історія вимірювань:\n");
                if (measurements.isEmpty()) {
                    sb.append("Дані про вимірювання відсутні.");
                } else {
                    for (Measurement m : measurements) {
                        sb.append(String.format("Тиск: %d/%d, Пульс:
%d, Цукор: %.1f; ",
                                m.getSystolic(), m.getDiastolic(),
m.getPulse(), m.getGlucose()));
                    }
                }

                String apiKey =
Utils.decryptAES(currentUser.getEncryptedApiKey(), currentPassword);
                AiService ai = new AiService(apiKey);

                String aiResult = ai.analyzeUserData(sb.toString());

      dbHelper.insertInterpretation(currentUser.getId(), aiResult);

                mainHandler.post(() -> {
                    Fragment f =
getSupportFragmentManager().findFragmentById(R.id.fragmentContainer)
;
                    if (f instanceof InterpretationFragment) {
                        ((InterpretationFragment)
f).setInterpretationText(aiResult);
                    }
                });
            } catch (Exception e) {
                mainHandler.post(() -> Toast.makeText(this, "AI
Error: " + e.getMessage(), Toast.LENGTH_LONG).show());
            }
        }).start();
    }
```

```
    public String getLastAiInterpretation() {
        if (currentUser == null) return null;
        return dbHelper.getLastInterpretation(currentUser.getId());
    }

    public List<Measurement> getCurrentUserMeasurements() {
        if (currentUser == null) return new ArrayList<>();
        return
dbHelper.getMeasurementsForPeriod((int)currentUser.getId(), 0,
System.currentTimeMillis());
    }

    public void saveSettings(String reminderTime, boolean
notificationsEnabled) {
        if (currentUser == null) return;
        dbHelper.updateSettings((int)currentUser.getId(),
reminderTime, notificationsEnabled);
        if (notificationsEnabled) {
            scheduleNotification(reminderTime);
        } else {
            cancelNotification();
        }
        Toast.makeText(this, "Налаштування збережено",
Toast.LENGTH_SHORT).show();
        getSupportFragmentManager().popBackStack();
    }

    private void scheduleNotification(String time) {
        String[] parts = time.split(":");
        int hour = Integer.parseInt(parts[0]);
        int minute = Integer.parseInt(parts[1]);

        Calendar calendar = Calendar.getInstance();
        calendar.set(Calendar.HOUR_OF_DAY, hour);
        calendar.set(Calendar.MINUTE, minute);
        calendar.set(Calendar.SECOND, 0);

        if (calendar.before(Calendar.getInstance())) {
            calendar.add(Calendar.DATE, 1);
        }

        Intent intent = new Intent(this,
NotificationReceiver.class);
        PendingIntent pendingIntent =
PendingIntent.getBroadcast(this, 100, intent,
                PendingIntent.FLAG_UPDATE_CURRENT |
PendingIntent.FLAG_IMMUTABLE);

        AlarmManager alarmManager = (AlarmManager)
getSystemService(ALARM_SERVICE);
        if (alarmManager != null) {
            alarmManager.setRepeating(AlarmManager.RTC_WAKEUP,
calendar.getTimeInMillis(),
```

```
                    AlarmManager.INTERVAL_DAY, pendingIntent);
        }
    }

    private void cancelNotification() {
        Intent intent = new Intent(this,
NotificationReceiver.class);
        PendingIntent pendingIntent =
PendingIntent.getBroadcast(this, 100, intent,
                PendingIntent.FLAG_UPDATE_CURRENT |
PendingIntent.FLAG_IMMUTABLE);
        AlarmManager alarmManager = (AlarmManager)
getSystemService(ALARM_SERVICE);
        if (alarmManager != null) {
            alarmManager.cancel(pendingIntent);
        }
    }
}
```

Лістинг А.10 – Файл fragment_welcome.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/welcomeRoot"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="24dp"
    android:gravity="center"
    android:background="@color/bg_medical">

    <TextView
        android:id="@+id/tvWelcomeTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Personal Medical Diary"
        android:textSize="22sp" />

    <TextView
        android:id="@+id/tvDisclaimer"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:text="Application does not replace professional
medical advice" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="24dp"
        android:orientation="vertical">

        <Button
```

```
            android:id="@+id/btnLogin"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Login" />

        <Button
            android:id="@+id/btnRegister"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="12dp"
            android:text="Register" />
    </LinearLayout>
</LinearLayout>
```

Лістинг А.11 – Файл fragment_login.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/loginRoot"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"
    android:padding="24dp"
    android:background="@color/bg_medical">

    <EditText
        android:id="@+id/etLogin"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Login"/>
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/etPassword"
        android:hint="Password"
        android:inputType="textPassword"/>

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/btnLoginConfirm"
        android:text="Login"/>
</LinearLayout>
```

Лістинг А.12 – Файл fragment_main_menu.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/menuRoot"
    android:orientation="vertical"
    android:padding="24dp"
```

```
    android:background="@color/bg_medical">

    <Button android:id="@+id/btnAddMeasurement"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="New measurement"/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/btnViewHistory"
        android:text="View history"/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/btnInterpretation"
        android:text="AI interpretation"/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/btnSettings"
        android:text="Settings"/>
</LinearLayout>
```

Лістинг А.13 – Файл fragment_measurement_input.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/inputRoot"
    android:orientation="vertical"
    android:padding="24dp"
    android:background="@color/bg_medical">

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/etSystolic"
        android:hint="Systolic"/>
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/etDiastolic"
        android:hint="Diastolic"/>
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/etPulse"
        android:hint="Pulse"/>
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/etGlucose"
```

```
        android:hint="Glucose (optional)"/>

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/btnSaveMeasurement"
        android:text="Save"/>
</LinearLayout>
```

Лістинг А.14 – Файл fragment_measurement_list.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/listRoot"
    android:orientation="vertical"
    android:padding="16dp"
    android:background="@color/bg_medical">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/btnFilter"
            android:text="Filter"/>
        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/btnInterpret"
            android:text="Interpret"/>
    </LinearLayout>

    <ListView
        android:id="@+id/lvMeasurements"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</LinearLayout>
```

Лістинг А.15 – Файл fragment_interpretation.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/interpretationRoot"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="24dp"
    android:background="@color/bg_medical">

    <ScrollView
```

```
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1">

        <TextView
            android:id="@+id/tvInterpretationResult"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textSize="16sp" />
    </ScrollView>

    <Button
        android:id="@+id/btnNewInterpretation"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="New interpretation" />
</LinearLayout>
```

Лістинг А.15 – Файл fragment_settings.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/settingsRoot"
    android:orientation="vertical"
    android:padding="24dp"
    android:background="@color/bg_medical">

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/etReminderTime"
        android:hint="HH:mm"/>

    <CheckBox
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/cbNotifications"
        android:text="Enable notifications"/>

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/btnSaveSettings"
        android:text="Save"/>
</LinearLayout>
```

Лістинг А.16 – Файл AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.INTERNET"/>
```

```
    <uses-permission
android:name="android.permission.POST_NOTIFICATIONS"/>
    <uses-permission
android:name="android.permission.SCHEDULE_EXACT_ALARM"/>

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.PersonalMedicalDiary"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver
            android:name=".utils.NotificationReceiver"
            android:enabled="true"
            android:exported="false" />
    </application>

</manifest>
```

# ДОДАТОК Б
# СЛАЙДИ ПРЕЗЕНТАЦІЇ

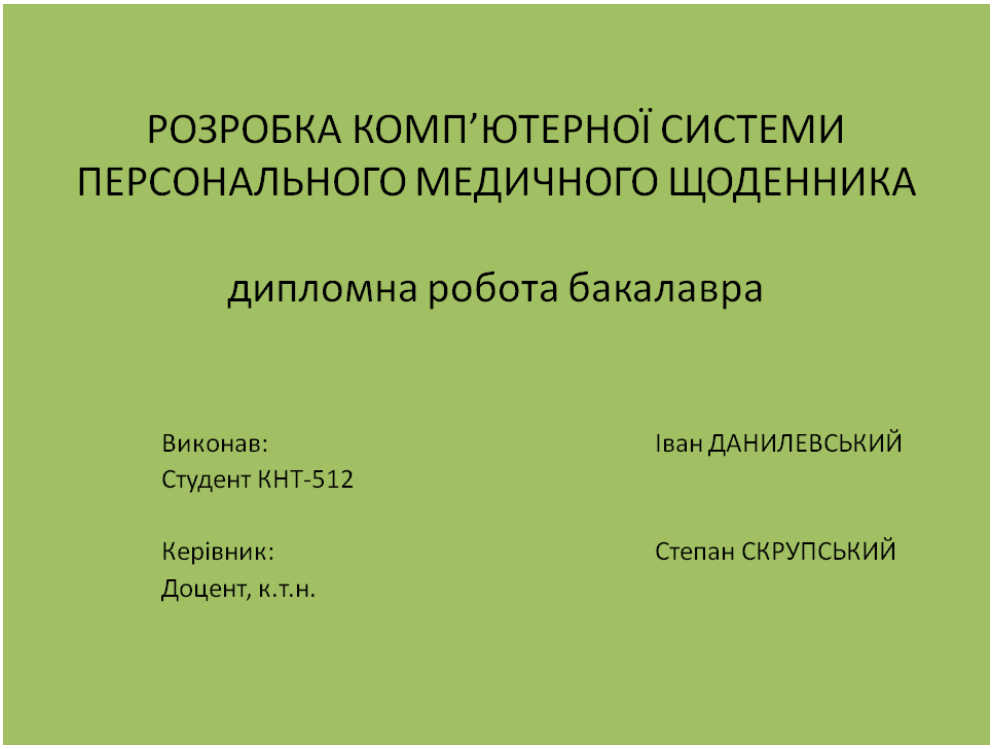


Рисунок Б.1 – Слайд 1

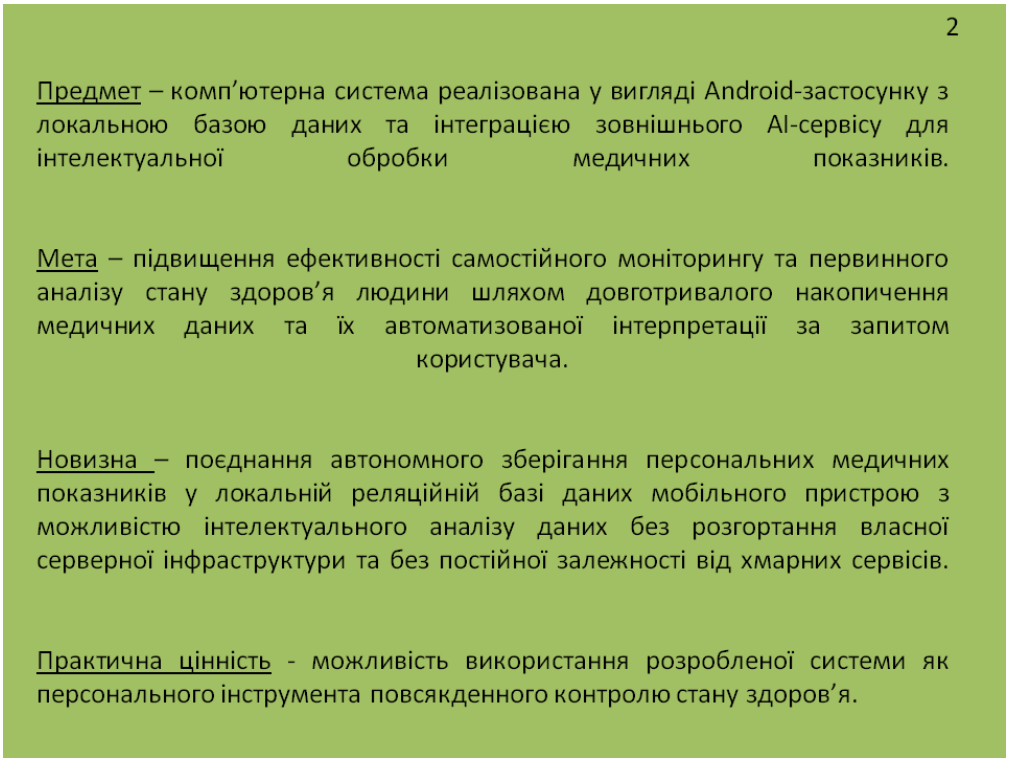


Рисунок Б.2 – Слайд 2

3

## Обрані технології


Рисунок Б.3 – Слайд 3

4

## Апаратні вимоги до системи

| Параметр | Мінімальна вимога | Обгрунтування |
|---|---|---|
| Тип пристрою | Смартфон або планшет | Система орієнтована на персональне використання та мобільність. Стаціонарний комп'ютер чи ноутуб не потрібні |
| SOC | ARMv8 (64-bit), 4 ядра, ≥ 1.5 ГГц, наприклад, Snapdpragon 715 | Забезпечує достатню продуктивність для роботи UI, SQLite та мережевих запитів |
| Оперативна пам'ять | ≥ 4 ГБ RAM | Цього вистачить для роботи одного Activity з кількома Fragment та фоновими потоками в такий спосіб, щоб ОС не вивільняла пам'ять шляхом знищення зайвих фрагментів |
| Вбудована пам'ять | ≥ 200 МБ вільного простору | Зберігання застосунку, локальної SQLite та історичних даних |
| Діагональ екрану | ≥ 6" (смартфон), ≥ 7" (планшет) | Забезпечує зручне відображення великих списків і форм введення |
| Роздільна здатність екрану | ≥ 1280×720 | Коректна верстка у портретній та альбомній орієнтаціях |
| Сенсорний екран | Обов'язково | Це основний спосіб взаємодії користувача із системою |
| Мережеве з'єднання | Wi-Fi або LTE/5G | Потрібне для отримання AI-інтерпретацій. Не обов'язково |

Рисунок Б.4 – Слайд 4


Рисунок Б.5 – Слайд 5


Рисунок Б.6 – Слайд 6

7

## Прототип (wireframe) застосунку системи

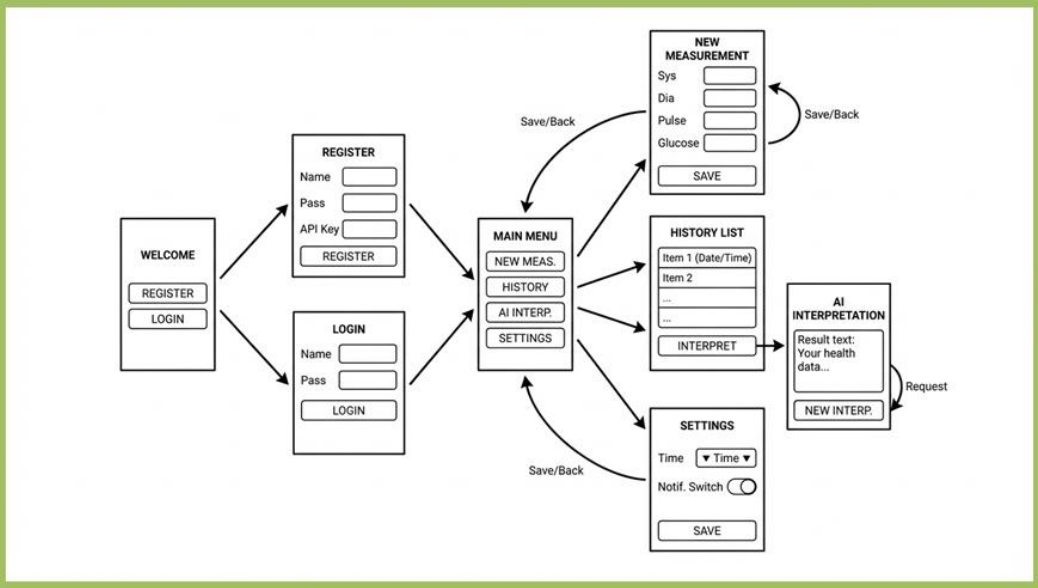


Рисунок Б.7 – Слайд 7

8

## Діаграма послідовностей навігації між екранами

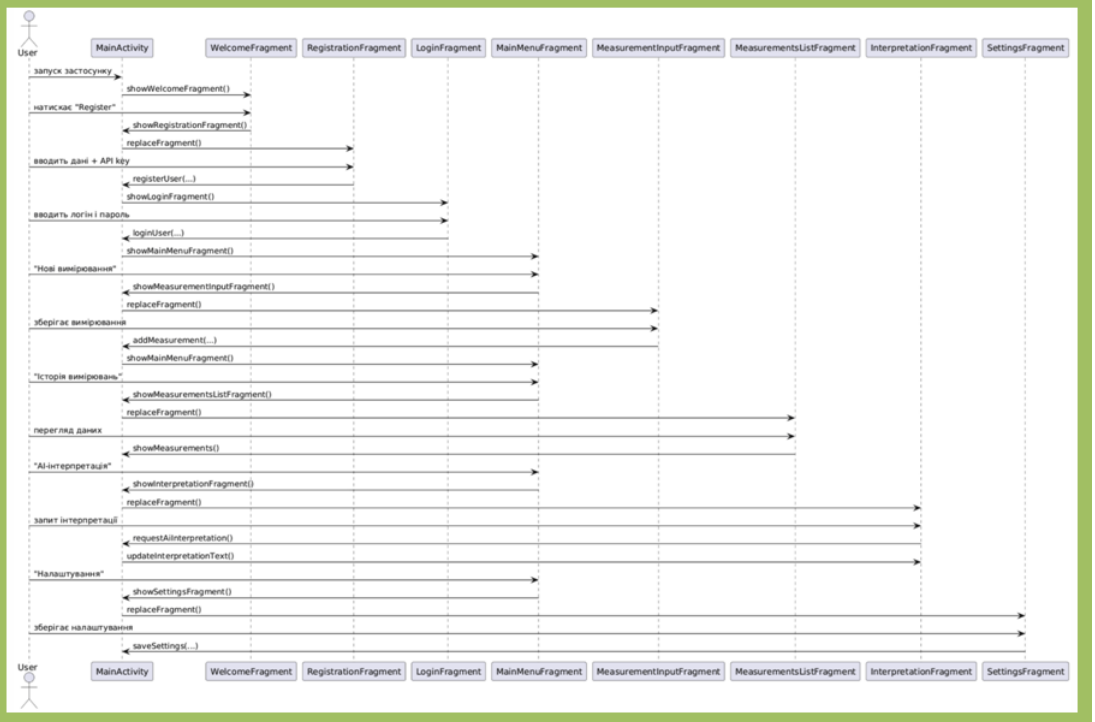


Рисунок Б.8 – Слайд 8

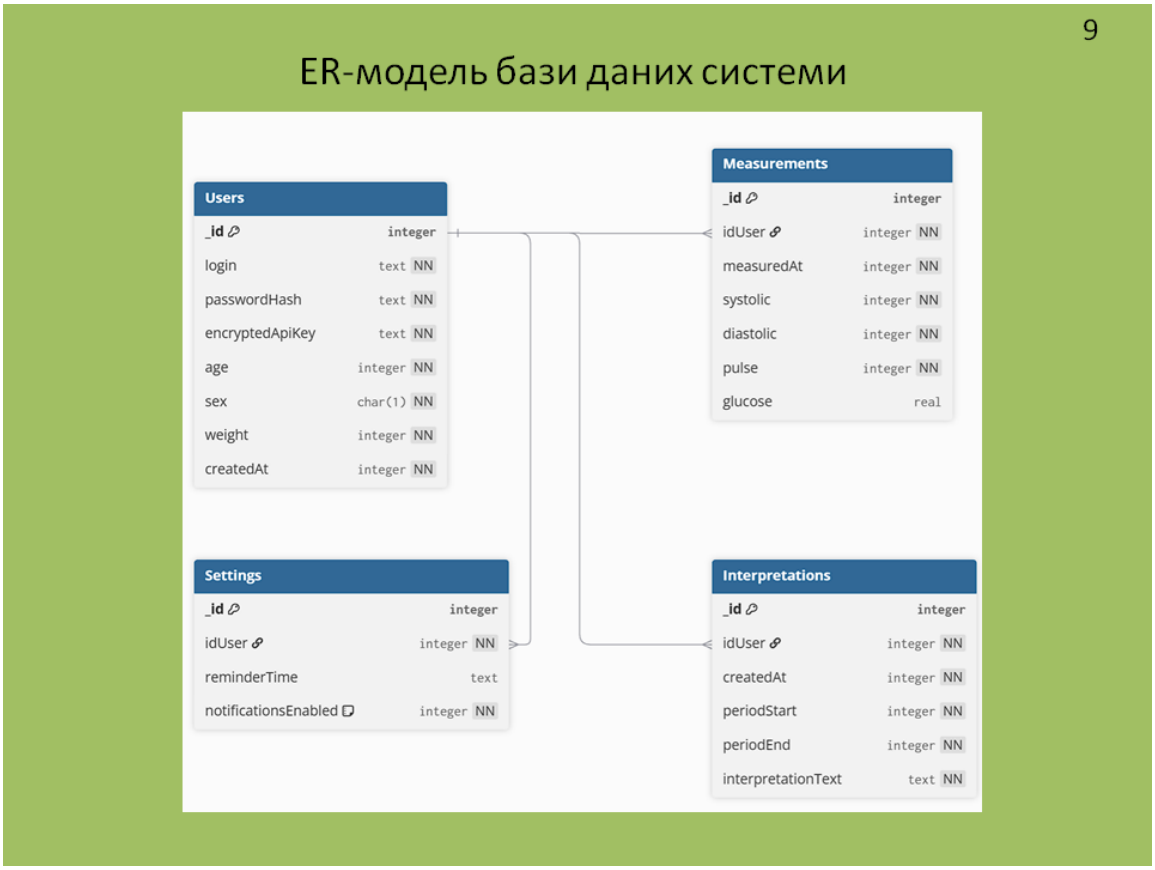


Рисунок Б.9 – Слайд 9

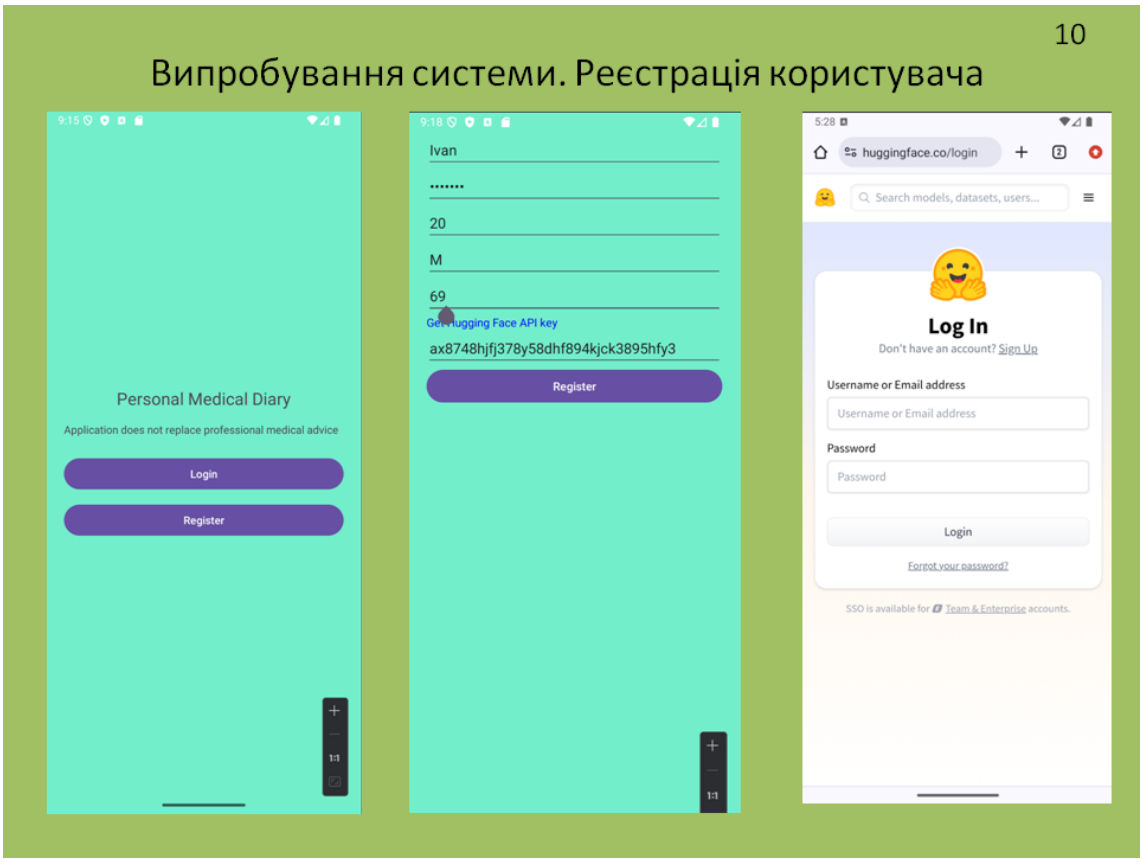


Рисунок Б.10 – Слайд 10

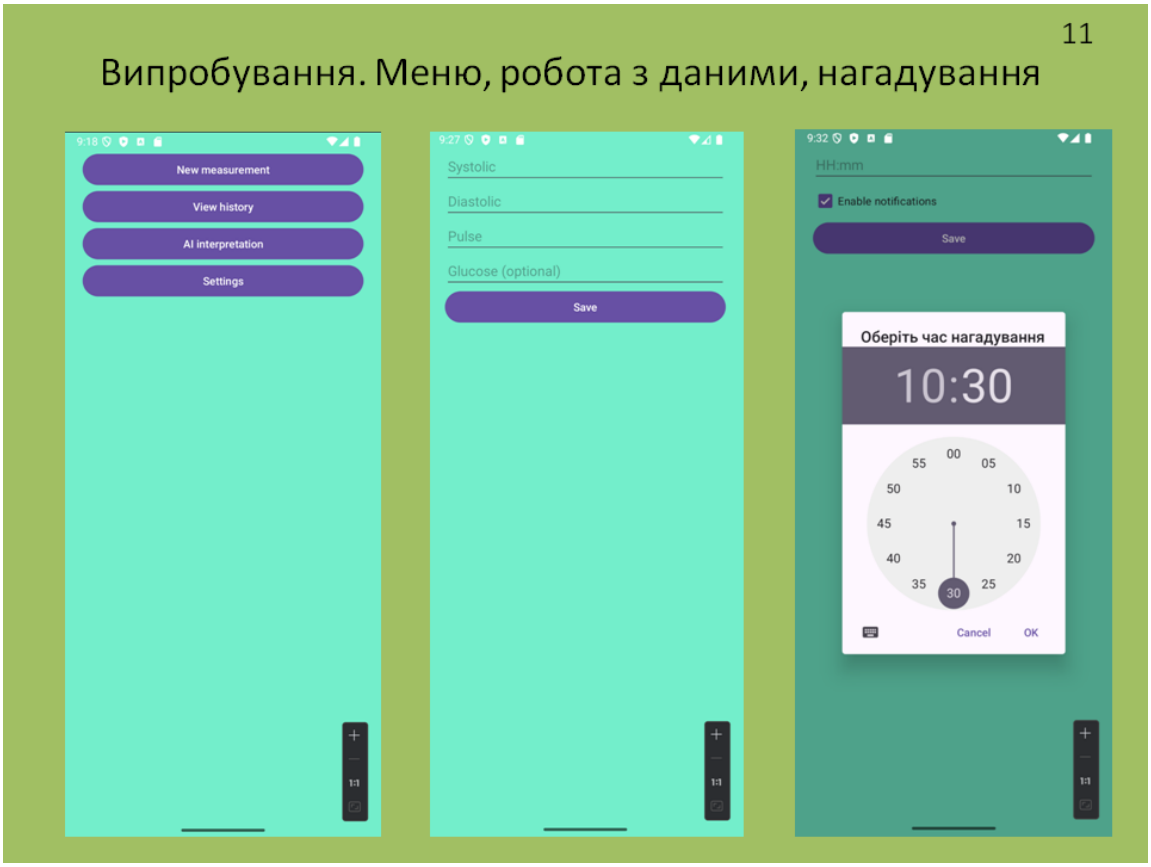


Рисунок Б.11 – Слайд 11

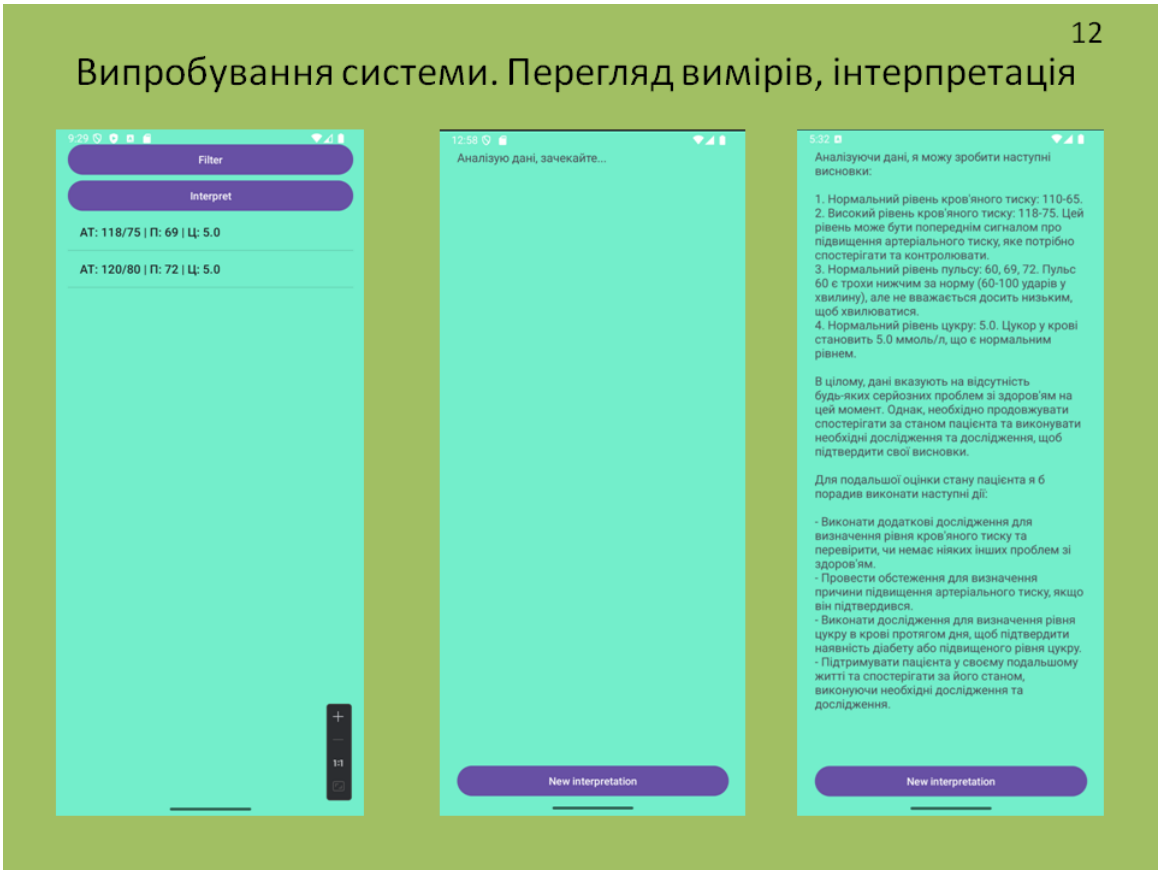


Рисунок Б.12 – Слайд 12

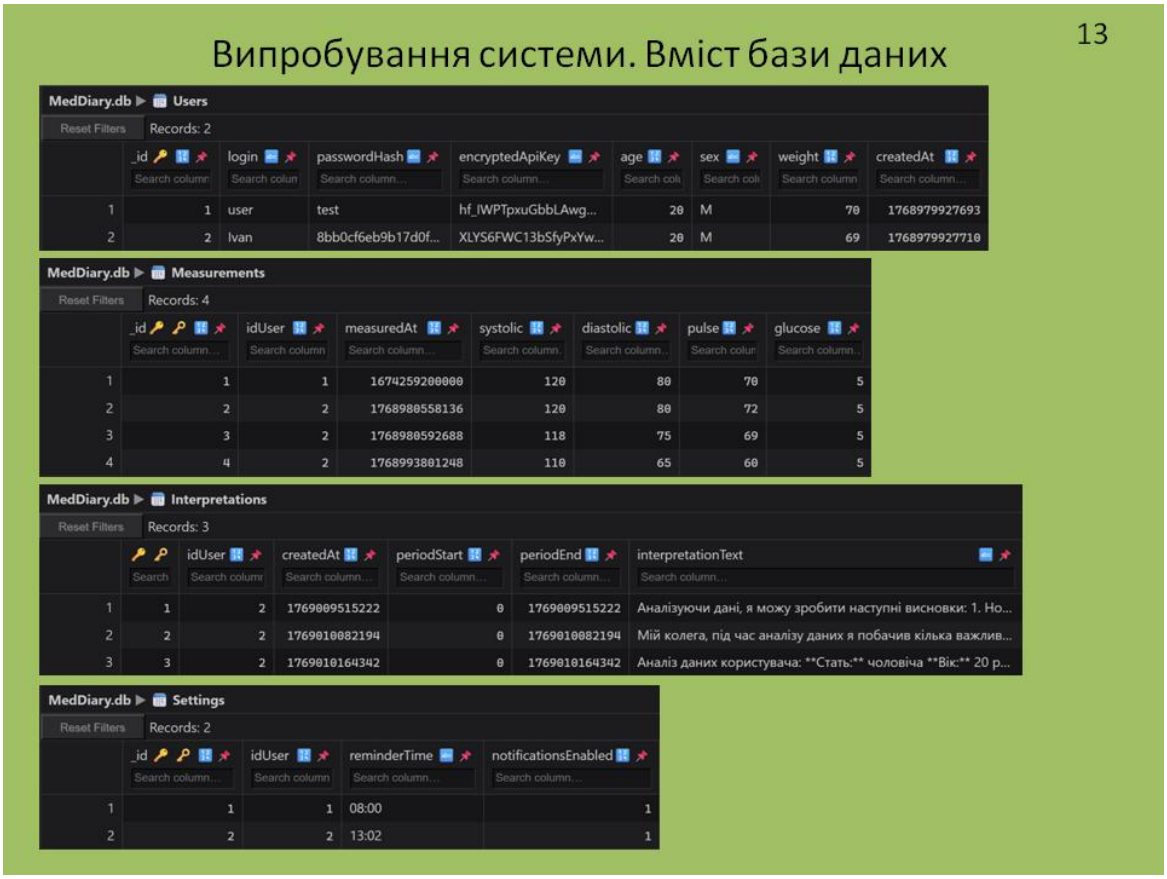


Рисунок Б.13 – Слайд 13

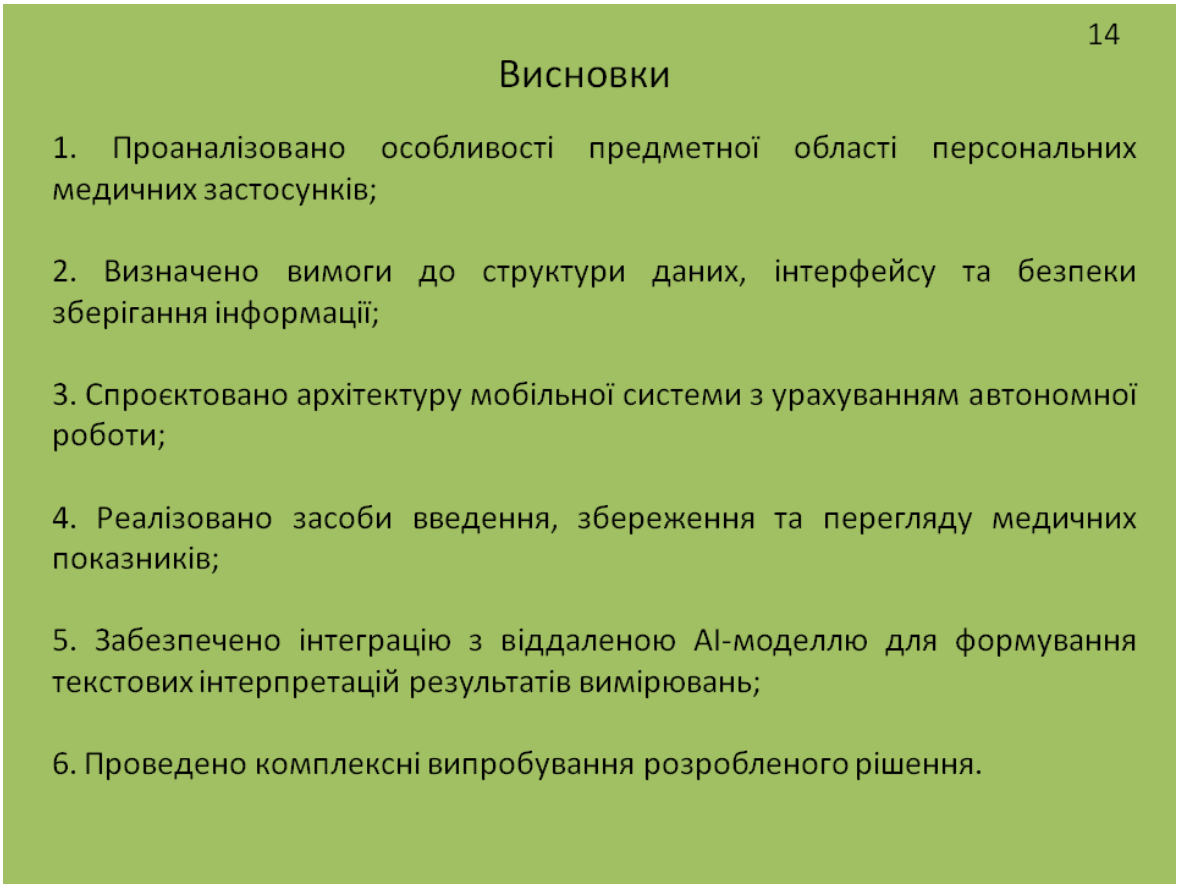


Рисунок Б.14 – Слайд 14