НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____ **Ігор БОЛБОТ**	_____ **Белла ГОЛУБ**
(підпис)	(підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмне забезпечення системи обліку особистого здоров'я»

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми к.т.н., доцент	_____ (підпис)	**Ганна ВАЙГАНГ**
Керівник бакалаврської кваліфікаційної роботи	_____ (підпис)	**Георгій Бородкін**
Виконав	_____ (підпис)	**Анатолій Рубан**

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2025 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ
Рубану Анатолію Олександровичу

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи

«Програмне забезпечення системи обліку особистого здоров'я»
затверджена наказом від «19» грудня 2025 р. № 3204 «С»
Термін подання завершеної роботи на кафедру «22» травня 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту): Документи МОЗ України щодо оформлення медичної документації
Перелік питань, що підлягають дослідженню:
1. Провести аналіз предметної області.
2. Дослідити стан існуючих аналогів.
3. Спроектувати інформаційне забезпечення системи.
4. Реалізувати мобільний застосунок для структурованого введення візитів і результатів досліджень із прив'язкою до МКХ-10.
5. Провести тестування системи.
6. Підготувати рекомендації щодо впровадження застосунку.

Перелік графічних документів (за необхідності).
Надати графічні ілюстрації (за необхідності)

Дата видачі завдання «19» грудня 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) **Георгій Бородкін**

Завдання взяв до виконання _____ (підпис) **Анатолій Рубан**

ЗМІСТ

ВСТУП

Здоров'я є фундаментальним та ключовим чинником якості життя людини. Упродовж свого життя людина проходить численні медичні обстеження, відвідує лікарів різних спеціальностей, отримує результати лабораторних досліджень і накопичує великий масив медичної документації. В умовах сучасної системи охорони здоров'я ця документація залишається переважно паперовою: результати аналізів видаються у вигляді роздруківок, записи лікарів ведуться у паперових картках, направлення та рецепти зберігаються вдома без будь-якої систематизації. Наслідком є регулярна втрата важливих документів, неможливість відстежити динаміку показників здоров'я в часі та фрагментованість інформації при зверненні до різних спеціалістів.

Актуальність теми. Цифровізація охорони здоров'я є одним із пріоритетних напрямів розвитку в Україні та світі. Разом із тим більшість існуючих рішень орієнтовані на медичні установи (електронні медичні картки, Медичні інформаційні системи) і недоступні для самостійного ведення записів пацієнтом. Мобільні застосунки для персонального обліку медичних даних або мають обмежену функціональність, або не адаптовані до потреб українського користувача, або вимагають платної підписки. Потреба у простому, безпечному та безкоштовному інструменті для цифрового ведення особистого медичного архіву є актуальною для широкого кола громадян.

Розроблена в рамках цієї роботи система вирішує проблему персонального обліку медичних документів шляхом надання пацієнту єдиного цифрового архіву з функціями пошуку, фільтрації, структурованого введення даних та аналізу динаміки показників здоров'я.

Мета роботи — розробка програмного забезпечення мобільного застосунку для управління особистими медичними записами, що забезпечує зручне зберігання, пошук і аналіз медичної документації користувача.

Задачі дослідження:

1. Провести аналіз предметної галузі та виявити основні проблеми ведення персональної медичної документації.
2. Дослідити існуючі програмні рішення у сфері персонального обліку здоров'я та визначити їх переваги і недоліки.
3. Обґрунтувати вибір методів, технологій та інструментів для реалізації системи.
4. Розробити структуру бази даних та UML-модель системи.
5. Реалізувати програмне забезпечення мобільного застосунку для платформ Android та iOS.
6. Провести тестування розробленої системи та підготувати рекомендації щодо її впровадження й експлуатації.

Об'єкт дослідження — процес персонального обліку медичних документів пацієнта.

Предмет дослідження — методи та засоби розробки мобільного застосунку для централізованого зберігання та управління особистою медичною документацією.

Методи дослідження:

- системний аналіз — для дослідження предметної галузі та виявлення вимог до системи;
- об'єктно-орієнтоване моделювання (UML) — для проектування архітектури системи;
- порівняльний аналіз — для оцінки існуючих аналогів та обґрунтування технологічного стеку;
- методи проектування реляційних баз даних — для розробки фізичної моделі даних;
- методи тестування програмного забезпечення — для верифікації коректності роботи системи.

Практичне значення роботи. Розроблений застосунок є повністю функціональним мобільним продуктом, готовим до розгортання в магазинах Google Play та Apple App Store. Система підтримує автентифікацію через email/пароль та Google OAuth, структуроване введення даних про візити до лікарів і результати лабораторних досліджень, прикріплення медичних файлів (PDF, JPEG, PNG, HEIC), повнотекстовий пошук і фільтрацію, а також статистичний аналіз показників здоров'я. Хмарна архітектура на основі Supabase забезпечує захищене зберігання даних із ізоляцією на рівні рядків (Row-Level Security), доступність із будь-якого пристрою та можливість масштабування.

Структура роботи. Дипломна робота складається з вступу, чотирьох розділів, висновків, списку використаних джерел та додатків.

У **першому розділі** проведено аналіз предметної галузі персонального обліку медичних документів, розглянуто основні проблеми паперового документообігу, досліджено існуючі аналоги та обґрунтовано постановку задачі.

У **другому розділі** розроблено інформаційне забезпечення системи: фізичну модель бази даних, діаграми класів, кооперації, пакетів і компонентів.

У **третьому розділі** описано практичну реалізацію системи: обґрунтування вибору технологій, архітектуру застосунку, реалізацію ключових алгоритмів і функціональність інтерфейсу користувача.

У **четвертому розділі** наведено рекомендації щодо впровадження та експлуатації системи: результати тестування, вимоги до апаратного і програмного забезпечення та склад інсталяційного пакету.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис проблеми та постановка задачі

Предметна область даної роботи — **персональний облік медичних документів пацієнта**. До учасників процесу належать:

- **Пацієнт** — особа, яка відвідує лікарів, проходить обстеження та накопичує медичну документацію.
- **Лікар** — медичний фахівець, що проводить прийом, встановлює діагноз, видає направлення та призначення.
- **Лабораторія** — медичний заклад, що виконує аналізи й дослідження та видає результати.

На сьогодні процес ведення особистих медичних записів є переважно **паперовим**: результати аналізів і записи лікарів видаються у вигляді роздруківок або записів від руки, зберігаються вдома без будь-якої системи, нерідко губляться або залишаються недоступними під час наступних візитів до інших спеціалістів.

Аналіз поточного процесу ведення медичної документації виявив чотири ключові проблеми, які наведені в Табл. 1.1.

Таблиця 1.1

Ключові проблеми ведення медичної документації

Проблема	Наслідок
Втрата документів	Паперові результати та записи лікарів легко губляться або псуються
Фрагментованість	Документи розпорошені по різних закладах, єдиного профілю пацієнта немає
Дублювання досліджень	Без доступу до попередніх результатів лікар змушений призначати повторні аналізи
Відсутність динаміки	Неможливо відстежити зміну показників здоров'я у часі без порівняння паперів

1.1.1 Діаграма прецедентів

Діаграма відображає основні сценарії взаємодії акторів у предметній галузі — без урахування реакції автоматизованої системи.

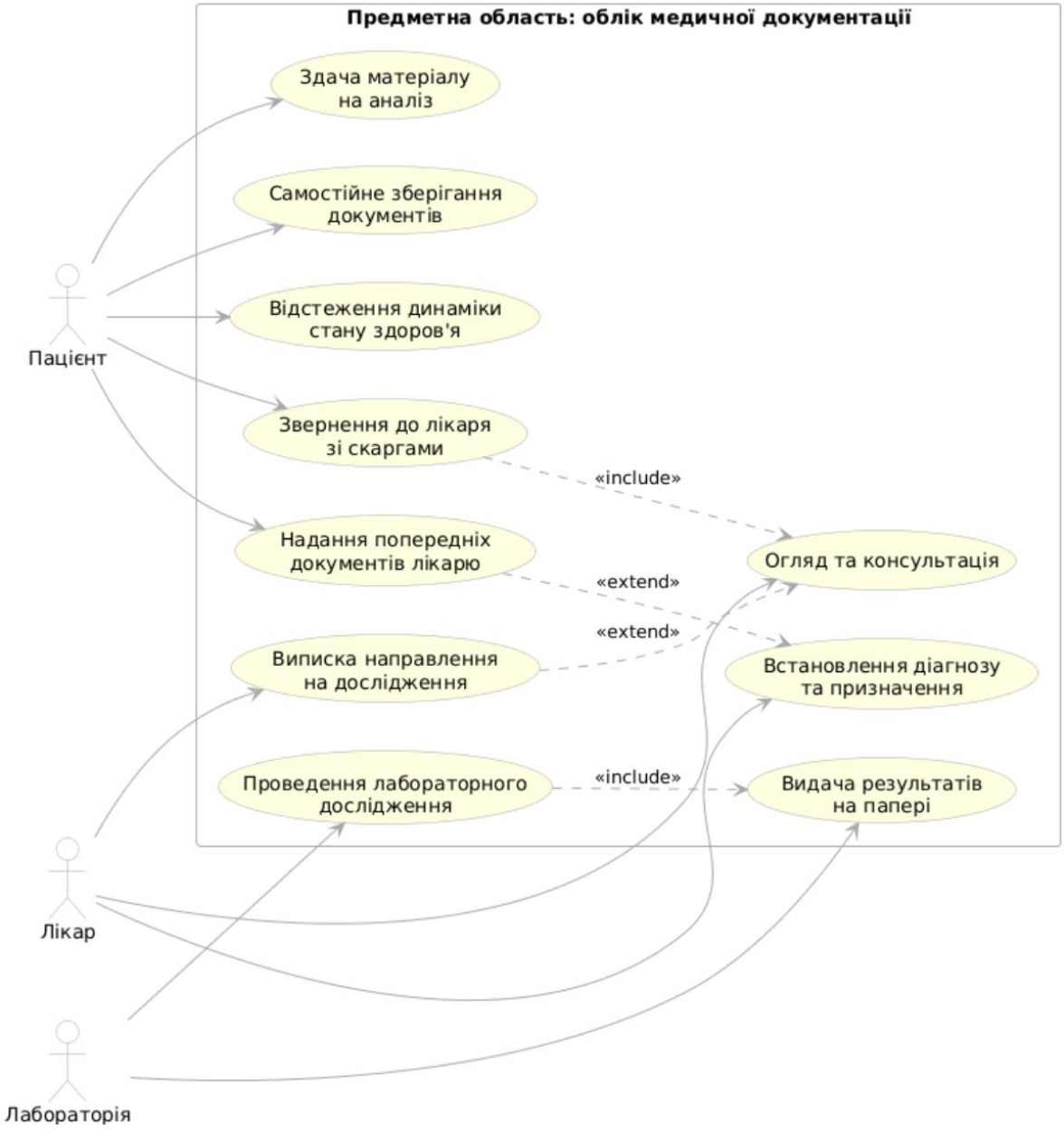


Рис. 1 Діаграма прецедентів

1.1.2 Діаграма послідовності — первинний прийом та направлення на аналізи

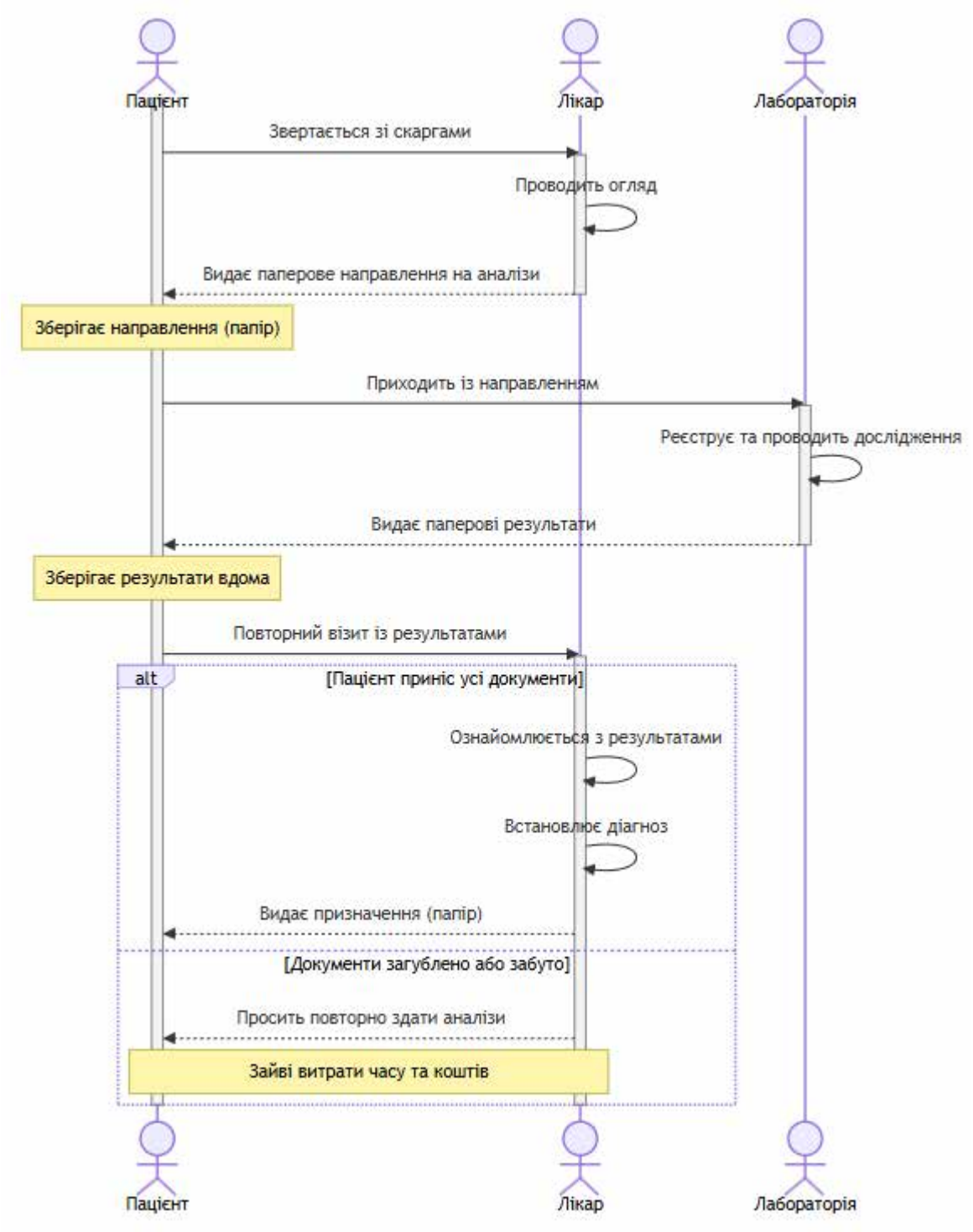


Рис. 2 Діаграма послідовності

1.1.3 Діаграма активності — поточний процес ведення медичної документації

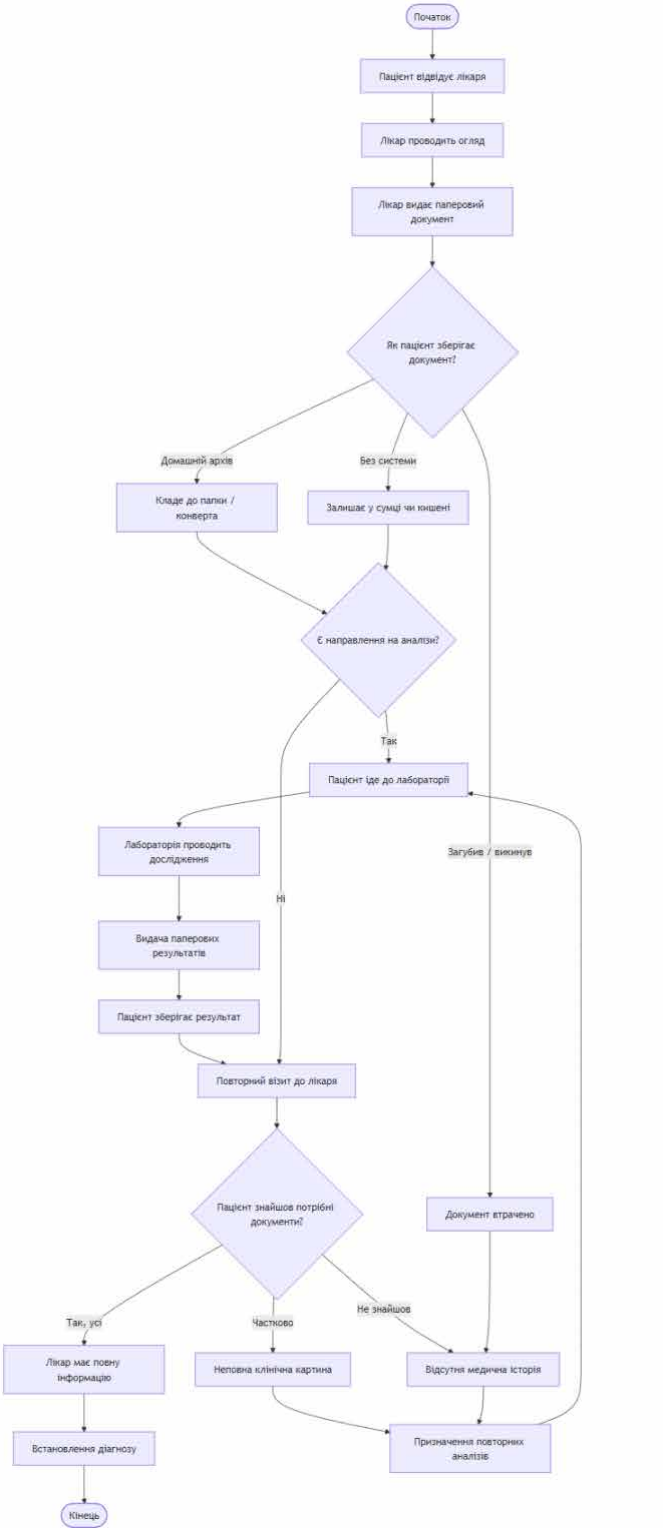


Рис. 3 Діаграма активності

1.2 Аналіз існуючих програмних рішень

На ринку персонального управління медичними даними існує ряд застосунків, кожен із яких має специфічну цільову аудиторію та набір функцій. Розглянемо найбільш поширені рішення.

1.2.1 Apple Health (iOS)

Apple Health — вбудована платформа Apple для зберігання медичних даних на iPhone. Підтримує автоматичний збір даних із носимих пристроїв (Apple Watch), інтеграцію з медичними закладами через стандарт FHIR, зберігання рецептів, вакцинацій, лабораторних результатів. Має функцію Health Records для отримання даних безпосередньо від клінік.

Переваги: глибока інтеграція з екосистемою Apple, підтримка стандартів FHIR/HL7, зручна візуалізація тенденцій.

Недоліки: доступна **лише на iOS**, Health Records недоступна в Україні (підтримується тільки в США, Канаді та ряді країн ЄС), неможливість прикріплення довільних файлів від лікарів.

1.2.2 Google Health (Android)

Google Health — екосистема Google для збору та аналізу даних про здоров'я. Включає Google Fit (фізична активність), інтеграцію з Health Connect API та партнерськими програмами охорони здоров'я. У 2023 році Google оголосив про реструктуризацію та призупинення ряду функцій.

Переваги: широка інтеграція з Android-пристроями, фітнес-трекінг.

Недоліки: відсутня підтримка ручного введення медичних документів, функція Health Records недоступна в Україні, відсутні інструменти для структурованого зберігання лікарських записів і файлів.

1.2.3 MyChart (Epic Systems)

MyChart — клієнтська частина електронної медичної картки від Epic Systems, однієї з найбільших MIC у світі. Надає пацієнту доступ до своїх медичних записів у клініках, що використовують Epic.

Переваги: повна інтеграція з медичними установами, перегляд результатів аналізів у реальному часі, запис на прийом, обмін повідомленнями з лікарями.

Недоліки: прив'язаний до конкретних клінік, які використовують Epic (в Україні Epic практично не застосовується), неможливо додавати власні документи з будь-якої установи.

1.2.4 Medisafe

Medisafe — застосунок для нагадування про прийом ліків та відстеження медикаментозного лікування. Допомагає пацієнтам дотримуватися схеми прийому препаратів.

Переваги: ефективне нагадування про прийом ліків, перевірка взаємодії препаратів, підтримка кількох пацієнтів.

Недоліки: вузька спеціалізація — орієнтований лише на медикаментозне лікування, не підтримує зберігання медичних документів, результатів аналізів, записів лікарів.

1.2.5 Helsi.me (Україна)

Helsi.me — українська платформа електронної охорони здоров'я, інтегрована з реєстром пацієнтів МОЗ України (eHealth). Дозволяє записатися до лікаря в державних медичних закладах, переглядати рецепти та направлення, виписані в системі.

Переваги: інтеграція з українською системою eHealth, запис на прийом у державних клініках, перегляд електронних рецептів.

Недоліки: прив'язана до державних клінік, не підтримує ручне введення приватних медичних записів, результати аналізів із приватних лабораторій не додаються, відсутня функція завантаження файлів.

1.3 Порівняльний аналіз існуючих рішень

На основі аналізу виявлено ключові критерії, які важливі для персонального обліку медичних документів. Порівняння проведено за 9 критеріями (Табл. 1.2).

Таблиця 1.2

Критерії, які важливі для персонального обліку медичних документів

Критерій	Apple Health	Google Health	MyChart	Medisafe	Helsi.me	Ясність
Платформа	iOS тільки	Android тільки	iOS + Android	iOS + Android	iOS + Android	iOS + Android
Ручне введення документів	Частково	Ні	Ні	Ні	Ні	**Так**
Прикріплення файлів (PDF/фото)	Ні	Ні	Ні	Ні	Ні	**Так (до 5 файлів)**
Пошук за МКХ-10 діагнозами	Ні	Ні	Так	Ні	Ні	**Так (17 000+ кодів)**
Доступність в Україні	Обмежена	Обмежена	Ні	Так	Так	**Так**
Незалежність від клінік	Частково	Ні	Ні	Так	Ні	**Так**
Інтеграція з приватними лабораторіями	Ні	Ні	Ні	Ні	Ні	**Так (ручне введення)**
Статистика динаміки показників	Так	Часткова	Так	Ні	Ні	**Так**
Безкоштовність	Так	Так	Так	Частково	Так	**Так**

Висновки з порівняльного аналізу:

Жодне з розглянутих рішень не задовольняє цілком потреби українського користувача у персональному обліку медичної документації. Основні прогалини:

1. **Відсутність незалежного сховища документів** — усі рішення прив'язані до конкретних медичних установ або платформ.
2. **Неможливість завантаження власних файлів** — результати аналізів із приватних лабораторій, рецепти, направлення не можна зберегти у жодному з розглянутих застосунків.
3. **Відсутність підтримки МКХ-10** — стандарт кодування захворювань, що використовується в Україні, підтримується лише MyChart, який недоступний у вітчизняних клініках.
4. **Мовний та регіональний бар'єр** — більшість застосунків орієнтовані на англомовний ринок.

Ці прогалини обумовлюють необхідність розробки спеціалізованого рішення, що поєднує структуроване введення медичних записів, зберігання файлів і аналіз даних для українського ринку.

1.4 Обґрунтування вибору методів розробки

1.4.1 Методологія розробки

Для розробки програмного забезпечення обліку особистого здоров’я обрано **ітеративну методологію** розробки програмного забезпечення. Ця методологія передбачає поступове нарощування функціональності через короткі ітерації (1–2 тижні), що дозволяє:

- оперативно реагувати на зміни вимог;
- регулярно перевіряти коректність реалізованих функцій;
- виявляти архітектурні проблеми на ранніх стадіях.

Альтернативи, що розглядалися:

- **Водоспадна модель** — підходить для проєктів із чітко фіксованими вимогами; не підходить через еволюційний характер вимог до UI.

- **Scrum** — повноцінний Scrum вимагає команди; для індивідуального розробника використовуються лише його практики (ітерації, backlog, щоденна перевірка прогресу).

1.4.2 Методи моделювання

Для проектування системи застосовано **UML 2.5.1** (Unified Modeling Language) — стандартизовану мову візуального моделювання. Використано такі типи діаграм (Табл. 1.3).

Таблиця 1.3

Типи діаграм, що були використані для проектування

Діаграма	**Призначення**
Прецедентів (Use Case)	Визначення акторів та сценаріїв взаємодії
Послідовності (Sequence)	Моделювання часових взаємодій між об'єктами
Активності (Activity)	Опис потоку керування в процесах
Класів (Class)	Структура даних та зв'язки між класами
Кооперації (Collaboration)	Взаємодія об'єктів у конкретних сценаріях
Пакетів (Package)	Логічна організація компонентів системи
Компонентів (Component)	Фізична структура програмного забезпечення
Розгортання (Deployment)	Топологія розгортання системи в інфраструктурі

1.4.3 Методи проектування бази даних

Для проектування бази даних застосовано:

- **ER-моделювання** (Entity-Relationship) — для розробки концептуальної моделі даних;

- **Нормалізацію до 3НФ** (третьої нормальної форми) — для усунення надмірності даних;
- **Row-Level Security (RLS)** — для забезпечення безпеки на рівні рядків без додаткової логіки на стороні застосунку.

1.5 Постановка задачі

На основі проведеного аналізу предметної галузі та існуючих рішень сформульовано наступну задачу:

Розробити мобільний застосунок «Ясність» для платформ Android та iOS, що забезпечує:

1. **Автентифікацію** користувача через email/пароль та через обліковий запис Google (OAuth 2.0).
2. **Структуроване введення медичних записів** двох типів:
    - *Візит до лікаря* — з вибором спеціалізації (за класифікатором МОЗ), дати, діагнозу за МКХ-10, призначення та нотаток;
    - *Результати дослідження* — з вказівкою типу дослідження, дати та числових показників, пов’язаного візиту, нотаток.
3. **Прикріплення медичних файлів** — до 5 файлів на запис (формати: JPG, JPEG, PNG, HEIC, PDF; до 10 МБ кожен) із захищеним хмарним зберіганням.
4. **Пошук і фільтрацію** записів за ключовими словами, діагнозом, типом документа, часовим діапазоном, спеціалізацією та наявністю файлів.
5. **Статистичний аналіз** — зведена плитка з лічильниками записів, стовпчаста діаграма щомісячної активності, кругова діаграма розподілу за спеціалізаціями.
6. **Управління акаунтом** — зміна пароля, зміна email із підтвердженням, вихід із системи, повне видалення акаунту з усіма даними.

Вимоги до системи:

- Дані зберігаються у хмарній базі даних PostgreSQL із Row-Level Security — кожен користувач має доступ лише до власних записів.
- Файли зберігаються у хмарному об’єктному сховищі (Supabase Storage) з ізоляцією за user_id.
- Інтерфейс виконаний повністю українською мовою.
- Застосунок функціонує на Android (API 26+) та iOS (14+).
- Час відповіді на запити до API не перевищує 3 секунди за умов стандартного мобільного з’єднання.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ ТА ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Проектування бази даних

2.1.1 Вибір СУБД

Для проєкту «Ясність» обрано **PostgreSQL** через хмарну платформу **Supabase**. Обрана СУБД надає всі необхідні механізми для реалізації безпечного сховища медичних даних (Табл. 2.1).

Таблиця 2.1

Перелік механізмів реалізації безпечного сховища медичних даних

Критерій	Обґрунтування
UUID як тип даних	Усі записи ідентифікуються UUID, що унеможливлює перебір ID через API
Row-Level Security (RLS)	Вбудований контроль доступу на рівні рядків — кожен користувач бачить лише власні дані
TIMESTAMPTZ	Зберігання дат із часовим поясом критично для медичних записів
CHECK-обмеження	Валідація перелічуваних значень на рівні БД
Каскадне видалення	ON DELETE CASCADE гарантує цілісність при видаленні записів
Supabase Auth	Вбудована автентифікація інтегрується з RLS через auth.uid()
Supabase Storage	Зберігання медичних файлів із тими ж RLS-правилами

Альтернативи, що не були обрані:

- *Firebase Firestore* — NoSQL, не підходить для реляційних зв'язків (МКХ-10 ієрархія)
- *MySQL* — відсутній UUID як вбудований тип, слабша підтримка JSON, відсутній RLS
- *SQLite* — локальна БД, неможлива синхронізація між пристроями

2.1.2 Фізична модель бази даних

База даних застосунку складається з **14 таблиць**, розподілених на три групи:

Довідники (Reference Tables) — глобальні незмінні дані, спільні для всіх користувачів:

- medical_specialties — медичні спеціальності за класифікатором МОЗ (~40 записів)
- icd_codes — коди МКХ-10 (17 000+ записів), ієрархічна структура з self-FK
- lab_test_types — типи лабораторних досліджень

Таблиці користувачів (User Tables) — ізольовані через RLS за user_id:

- profiles — профіль користувача (розширення таблиці auth.users)
- visits — записи про візити до лікарів
- lab_results — результати лабораторних досліджень
- medical_files — метадані прикріплених файлів (фізичні файли — у Storage)
- lab_result_indicators — числові показники результатів аналізів

Зв'язкові таблиці (Junction Tables):

- visit_specialties — зв'язок M:N між візитами і спеціальностями
- visit_diagnoses — зв'язок M:N між візитами і діагнозами МКХ-10
- visit_lab_results — зв'язок M:N між візитами і дослідженнями
- lab_result_diagnoses — зв'язок M:N між дослідженнями і діагнозами
- Кожна таблиця знаходиться в 3НФ: відсутні транзитивні залежності, багатозначні поля нормалізовані в окремі зв'язкові таблиці (visit_specialties, visit_diagnoses).

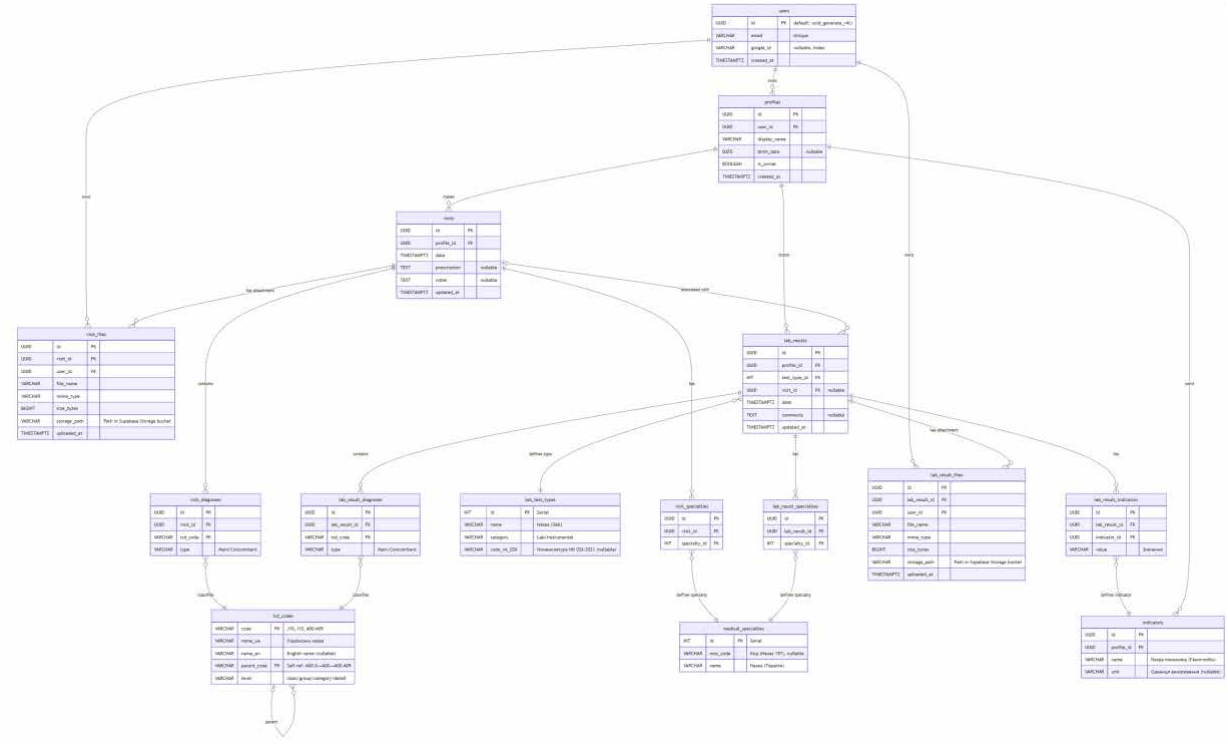


Рис. 4 ER-діаграма

2.1.3 Механізми безпеки

Row-Level Security (RLS) забезпечує ізоляцію даних на рівні СУБД. Для кожної таблиці користувачів визначено політику:

CREATE POLICY "users_own_visits" ON visits FOR ALL USING (auth.uid() = user_id);

Це означає, що навіть якщо клієнт надішле некоректний user_id у запиті — PostgreSQL автоматично відфільтрує чужі записи.

Тригери використовуються для підтримки узгодженості даних:

- update_updated_at — автоматичне оновлення поля updated_at при зміні запису (тригер BEFORE UPDATE)
- Каскадне видалення файлів зі Storage при видаленні запису (реалізовано через Edge Function)

Індекси оптимізують типові запити:

```
CREATE INDEX idx_visits_user_date ON visits(user_id, date DESC);
CREATE INDEX idx_lab_results_user_date ON lab_results(user_id, date DESC);
CREATE INDEX idx_medical_files_visit ON medical_files(visit_id);
CREATE INDEX idx_icd_codes_level ON icd_codes(level);
```

2.2 Діаграми класів та кооперацій

2.2.1 Загальна діаграма класів

На Рис. 5 зображено структуру всіх класів системи та асоціації між ними.

2.2.1 Кооперація 1 — Управління медичними документами

Механізм: Створення, редагування та видалення медичних записів із вкладеними файлами.

ВізитДоЛікаря та РезультатДослідження є спеціалізаціями абстрактного класу МедичнийЗапис (узагальнення). Файли не можуть існувати окремо від батьківського запису — між ними встановлено композицію. Зв'язок Користувач → МедичнийЗапис — асоціація один-до-багатьох.

2.2.3 Кооперація 2 — Класифікація лікарського прийому

Механізм: Прив'язка візиту до медичної спеціальності та кодів діагнозів МКХ-10.

Ієрархія кодів МКХ-10 моделюється через узагальнення: КодКатегорії та КодДеталі є спеціалізаціями ДіагнозМКХ10. КодДеталь посилається на батьківський КодКатегорії через асоціацію (поле parent_code).

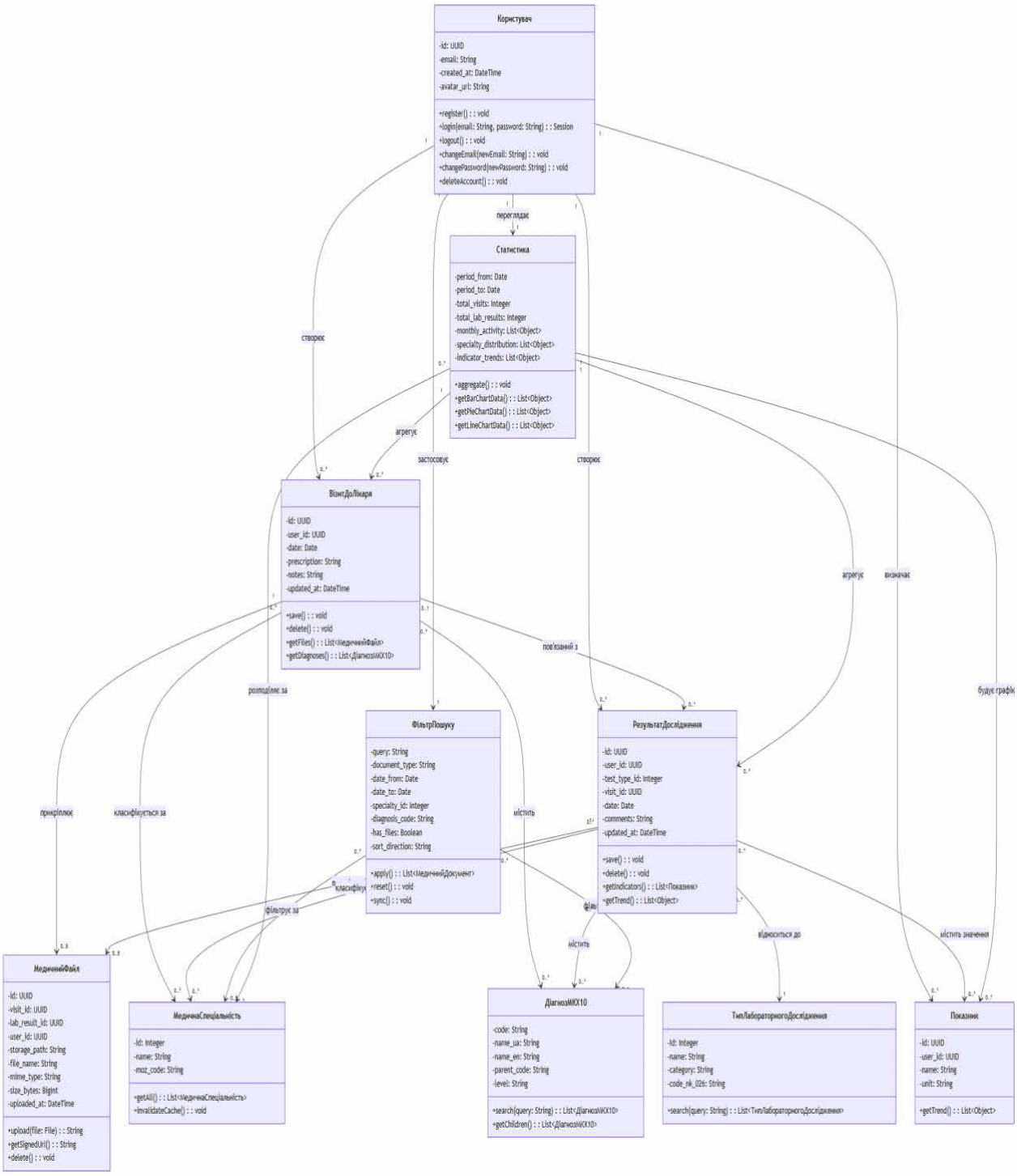


Рис. 5 Загальна діаграма класів

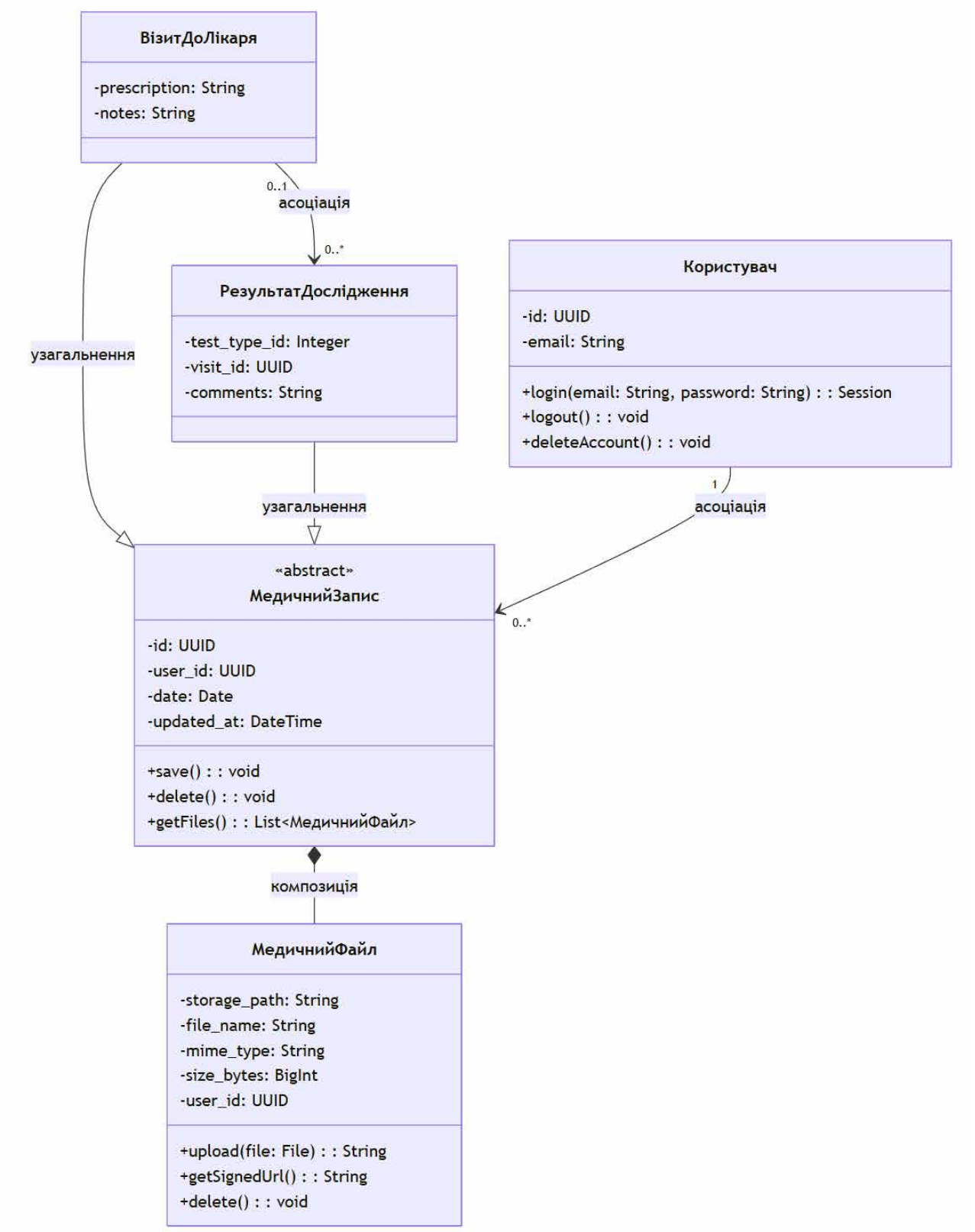


Рис. 6 Кооперація 1 — Управління медичними документами

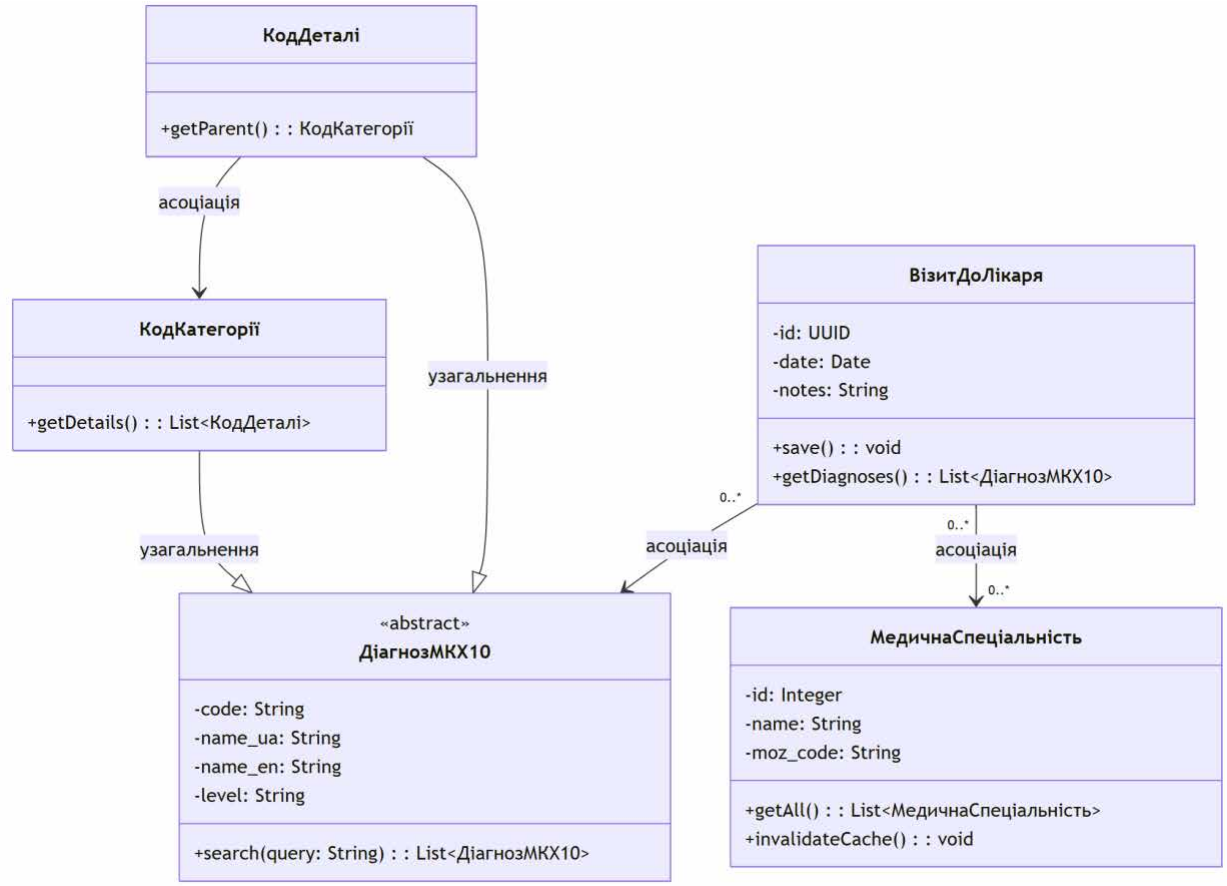


Рис. 7 Кооперація 2 — Класифікація лікарського прийому

2.3 Діаграми пакетів

2.3.1 Структура пакету Client

На Рис. 8 зображена клієнтська частина застосунку, що містить підпакети ClientGUI (графічний інтерфейс) та ClientNetwork (мережева комунікація). GUI залежить від мережевого шару через «import».

2.3.2 Структура пакету Server

Серверна частина зображена на Рис. 9 та містить чотири підпакети: публічні ServerBusinessLogic та Utils, приватні RequestDB та ServerNetwork.

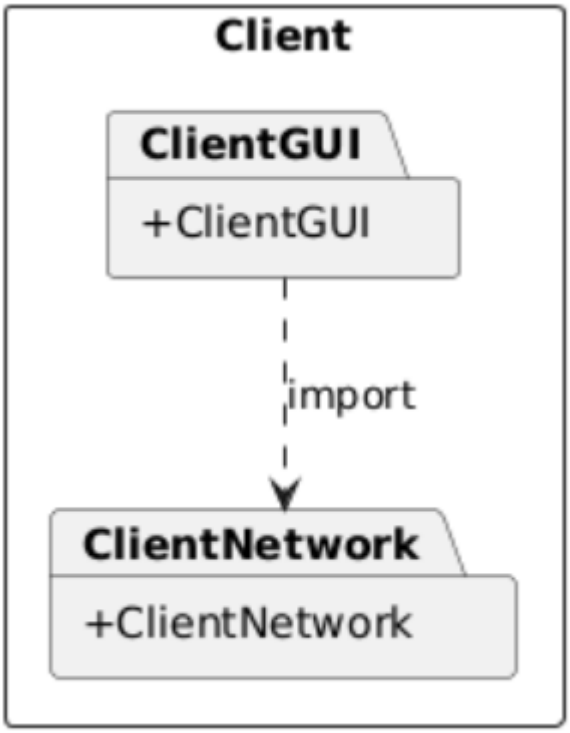


Рис. 8 Діаграма пакетів - структура пакету Client

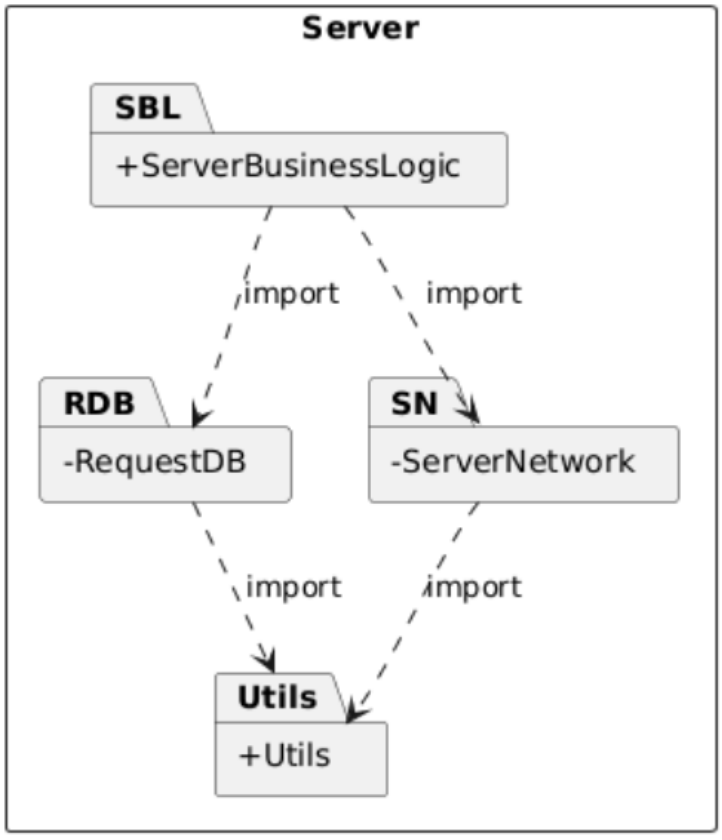


Рис. 9 Діаграма пакетів - структура пакету Server

2.3.3 Діаграма пакетів застосунку

Організаційна структура проєкту складається з шести основних пакетів. Стрілки спрямовані лише донизу — app → components → lib → types, що забезпечує однонаправленість потоку даних.

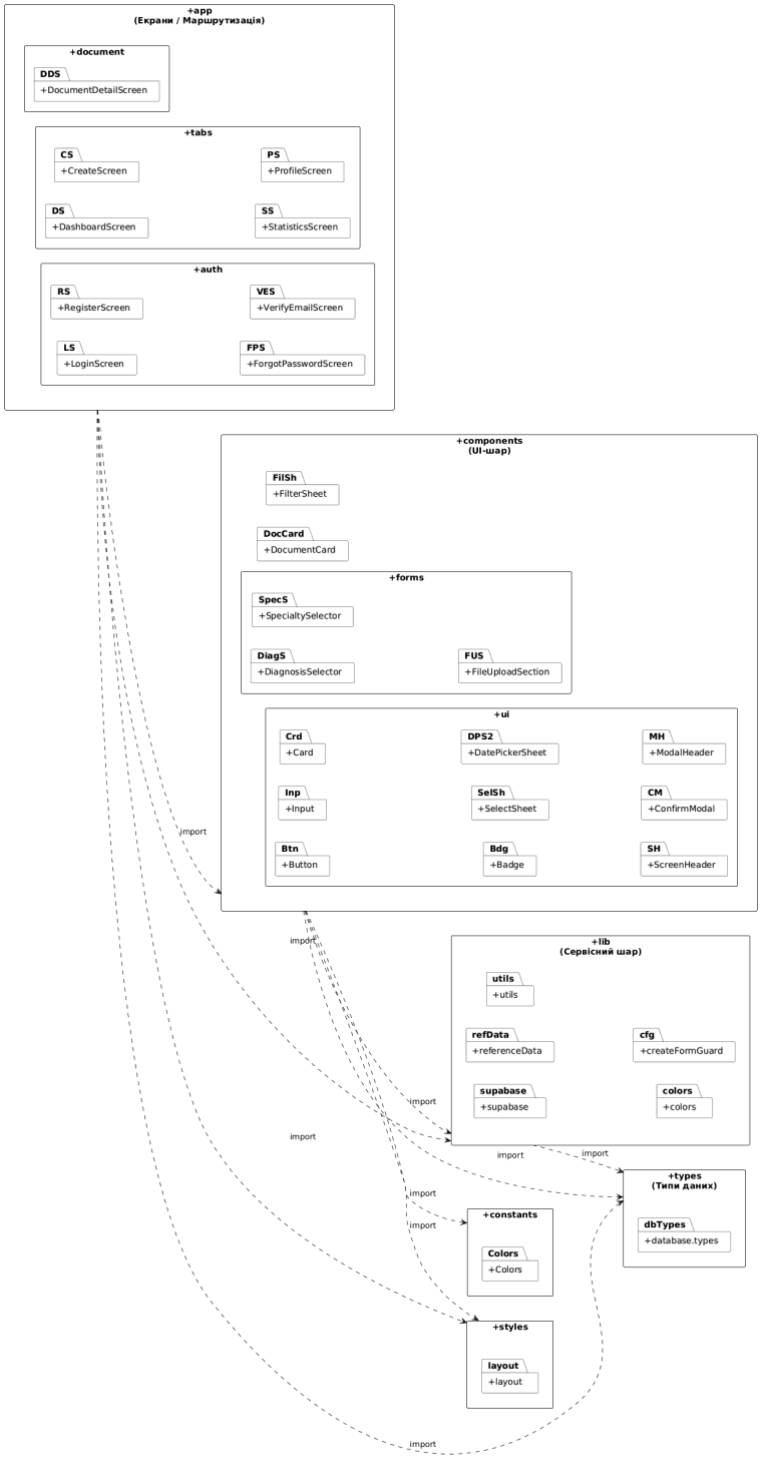


Рис. 10 Діаграма пакетів застосунку

Опис пакетів системи наведений у Табл. 2.2.

Таблиця 2.2

Опис пакетів системи

Пакет	Видимість	Призначення
app	+	Точки входу застосунку (екрани). Верхній рівень, залежить від усіх інших
components	+	UI-шар. Підпакет ui — перевикористовувані примітиви; forms — доменні селектори
lib	+	Сервісний шар: клієнт Supabase, запити до довідників, утиліти
types	+	Інтерфейси TypeScript, що відображають схему БД. Найнижчий рівень
styles	+	Спільні стилі (safeAreaTop, textInputStyle)
constants	+	Константи кольорів та налаштувань

2.4 Діаграми компонентів

2.4.1 Структура вихідного коду

На Рис. 11 представлена діаграма, яка відображає компонентну структуру застосунку та інтерфейси між шарами.

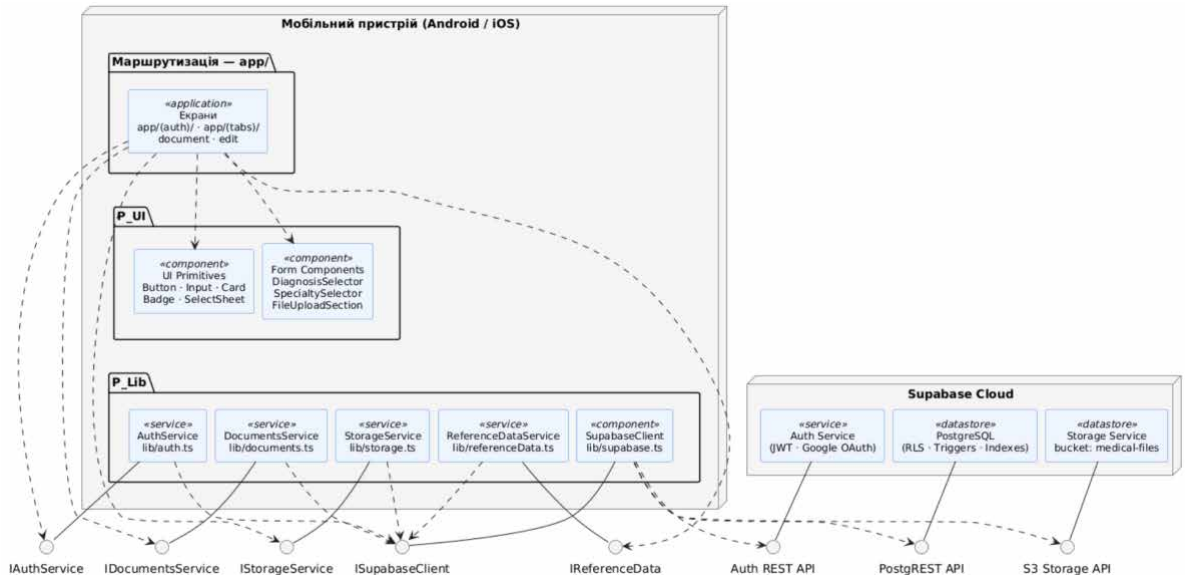


Рис. 11 Діаграми компонентів вихідного коду

Легенда:

- ○— (lollipop) — наданий інтерфейс: компонент реалізує цей інтерфейс
- ..> — залежність (required): компонент потребує цей інтерфейс
- <<service>> — сервіс без власного UI
- <<datastore>> — сховище даних
- <<application>> — виконуваний застосунок із UI

2.4.2 Діаграма компонентів — етапи збірки

На Рис. 12 надана діаграма, яка відображає компонентну структуру застосунку на етапі збірки

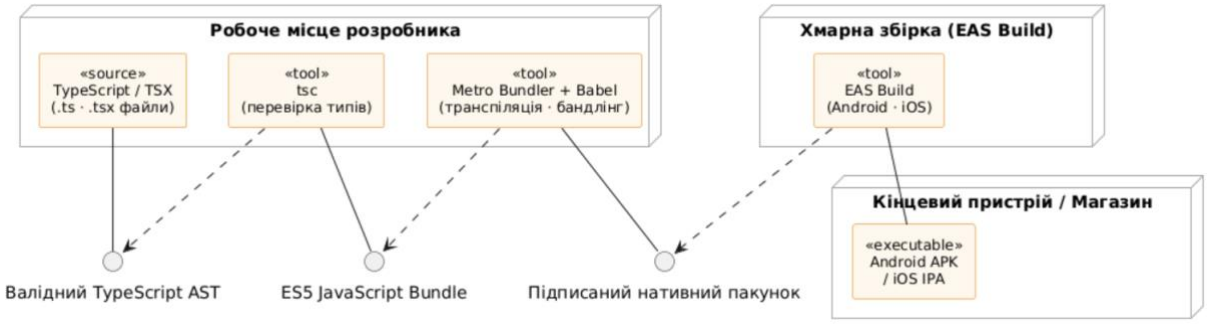


Рис. 12 Діаграма компонентів — етапи збірки

Опис компонентів системи наведений у Табл. 2.3.

Таблиця 2.3

Опис компонентів системи

Компонент	Тип	Що робить
TypeScript / TSX	<<source>>	Вихідний код застосунку
tsc	<<tool>>	Перевіряє типи; надає валідний AST
Metro + Babel	<<tool>>	Транспілює TS → ES5, збирає всі import у єдиний бандл
EAS Build	<<tool>>	Компілює нативний шар, підписує пакунок
APK / IPA	<<executable>>	Готовий пакунок для встановлення

3 ПРОЄКТНА ЧАСТИНА ТА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Загальна характеристика системи

Система «Ясність» — мобільний застосунок для ведення особистих медичних записів — реалізована на базі клієнт-серверної архітектури. Клієнтська частина розроблена на React Native (Expo), а серверна інфраструктура побудована на платформі Supabase, що надає PostgreSQL-базу даних, API-шлюз, систему автентифікації та об'єктне сховище файлів.

Мобільний застосунок встановлюється на пристрій користувача (Android або iOS). Усі мережеві запити надходять по HTTPS до Supabase Cloud, де PostgREST API-шлюз транслює їх у SQL-запити до бази даних PostgreSQL. Файли медичних документів завантажуються до бакету medical-files Supabase Storage та зберігаються з унікальним шляхом виду userId/documentId/filename.

Row-Level Security (RLS) PostgreSQL забезпечує ізоляцію даних: кожен користувач бачить виключно свої записи. Правила RLS визначаються безпосередньо на рівні таблиць бази даних, що унеможливлює несанкціонований доступ навіть при прямих зверненнях до API.

На Рис. 13 зображено діаграму розгортання.

3.2 Обґрунтування вибору технологій

3.2.1 React Native + Expo

React Native — фреймворк від Meta для розробки нативних мобільних застосунків на JavaScript/TypeScript. Замість WebView він компілює компоненти у нативні UI-елементи платформи (Android View / iOS UIView), що забезпечує продуктивність, яка наближається до нативних застосунків.

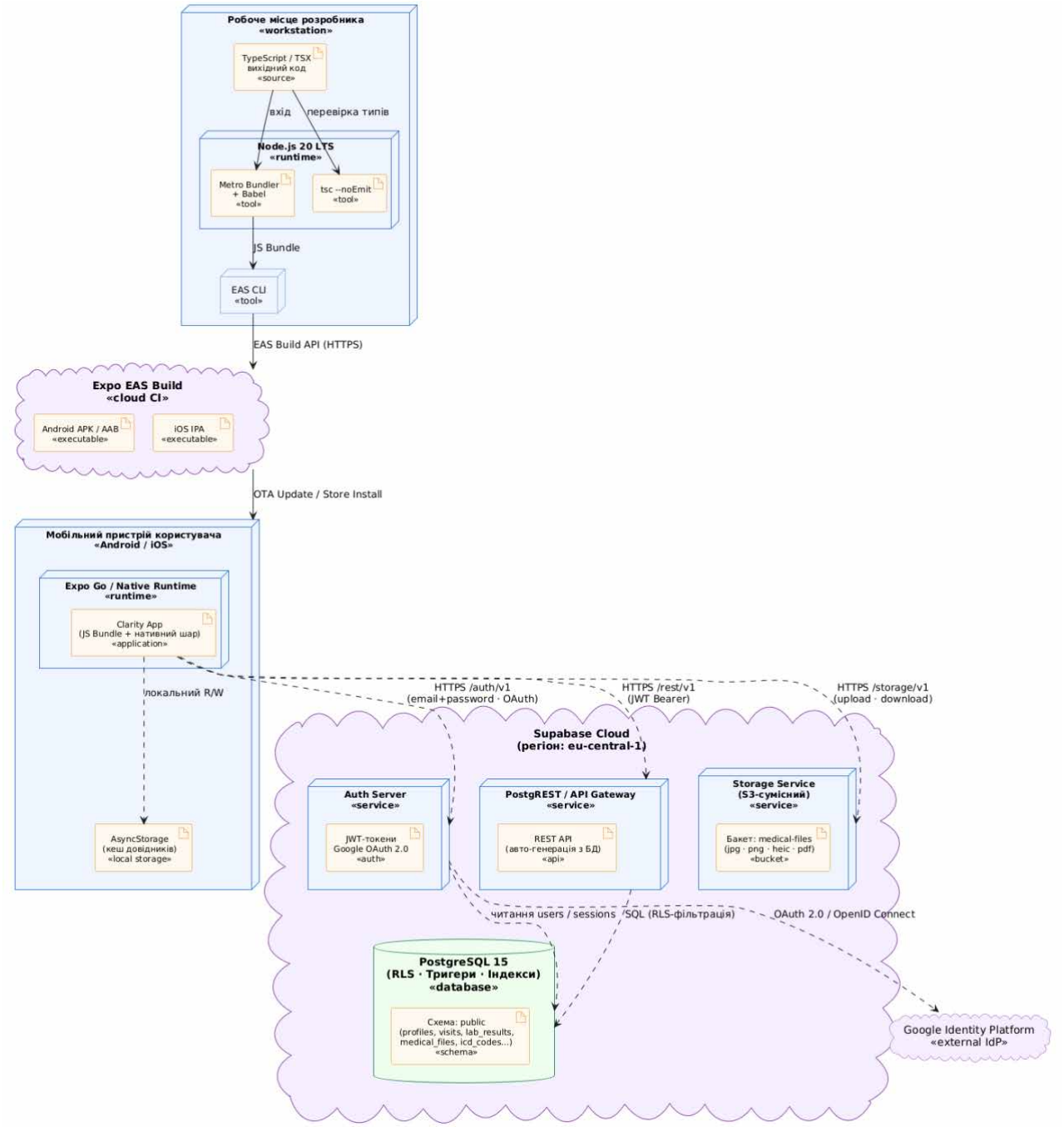


Рис. 13 Діаграма розгортання

Expo — платформа над React Native, що надає:

- готовий набір нативних SDK (камера, файлова система тощо);
- **Expo Go** — клієнт для миттєвого тестування на реальному пристрої без збірки;
- **EAS Build** — хмарна CI/CD-збірка APK/IPA без локального налаштування Xcode/Android Studio.

У Табл. 3.1 наведений перелік критеріїв та властивості їх реалізації у залежності від фреймворку та операційної системи, для якої проектується застосунок.

Таблиця 3.1

Критерії обирання фреймворку

Критерій	React Native + Expo	Нативний Android/iOS	Flutter
Цільова платформа	Android + iOS з однієї кодової бази	Окрема база для кожної	Android + iOS з однієї бази
Мова	TypeScript	Kotlin / Swift	Dart
Час на прототипування	Швидкий (hot reload, Expo Go)	Повільний (повна збірка)	Швидкий
Нативна продуктивність	Висока (JSI/Hermes)	Максимальна	Висока (Skia)

Оскільки розроблювана система — медичний застосунок для кінцевих користувачів, що має підтримувати і Android, і iOS, то фреймворк і набір інструментів React Native + Expo є оптимальним балансом між швидкістю розробки, продуктивністю та розміром команди.

3.2.2 NativeWind v4 (Tailwind CSS для React Native)

Бібліотека **NativeWind** надає можливість переносити концепцію utility-first CSS із Tailwind CSS у середовище React Native. Під час збірки Babel-плагін перетворює рядки className у відповідні стилі React Native (StyleSheet) (Табл. 3.2).

3.2.3 Supabase

Supabase — платформа Backend-as-a-Service на базі PostgreSQL, що надає автоматично генерований REST API (PostgREST), систему автентифікації

(GoTrue), об'єктне сховище (S3-сумісне) та Row-Level Security. Вибрана як бекенд для реалізації системи через:

- вбудовану підтримку RLS (ізоляція даних на рівні БД);
- автоматичне генерування REST API з типами TypeScript;
- безкоштовний тариф достатній для MVP;
- відкритий вихідний код, можливість самостійного хостингу.

Таблиця 3.2

Застосування бібліотеки NativeWind

Критерій	**NativeWind v4**	**StyleSheet.create**	**Styled Components**
Підхід до стилів	Utility-class (className)	Inline об'єкти	CSS-in-JS (styled)
Консистентність	Єдина система токенів через tailwind.config.js	Ручне дублювання	Потребує власну тему
Перфоманс	Компілюється на етапі збірки	Нативний механізм	Runtime-обчислення
Підтримка темної теми	dark: модифікатор	Ручний useColorScheme	Ручна тема

3.2.4 Supabase

Supabase — платформа Backend-as-a-Service на базі PostgreSQL, що надає автоматично генерований REST API (PostgREST), систему автентифікації (GoTrue), об'єктне сховище (S3-сумісне) та Row-Level Security. Вибрана як бекенд для реалізації системи через:

- вбудовану підтримку RLS (ізоляція даних на рівні БД);
- автоматичне генерування REST API з типами TypeScript;
- безкоштовний тариф достатній для MVP;
- відкритий вихідний код, можливість самостійного хостингу.

3.3 Реалізація ключових функцій

3.3.1 Обирання архітектури реалізації

Компоненти розбиті на два шари (Табл. 3.3).

Таблиця 3.3

Принципи розбудови двошарової компонентної архітектури

Шар	Директорія	Призначення	Приклади
UI Primitives	components/ui/	Загальні примітиви з можливістю адаптованого використання	Button, Input, Card, Badge, SelectSheet
Form Components	components/forms/	Доменні поля з модальними вікнами	DiagnosisSelector, SpecialtySelector, FileUploadSection

Патерн модального аркуша — складні пікери (спеціалізація, МКХ-10, дата) відкриваються через Modal з animationType="slide" та presentationStyle="pageSheet".

Структура: заголовок із кнопкою закриття, поле пошуку, FlatList елементів, кнопка підтвердження.

3.3.2 Валідація форми

Функція validate() перевіряє обов'язкові поля залежно від обраного типу документа. Помилки зберігаються у словнику errors: Record<string, string>, де ключ — назва поля, а значення — текст повідомлення. Кожне поле виправляє відповідну помилку у разі зміни значення:

```
// app/(tabs)/create.tsx
function validate(): boolean {
  const e: Record<string, string> = {};
  if (docType === 'visit') {
    if (visitSpecialties.length === 0)
```

```
    e.specialty = 'Поле обов\'язкове';
   if (!visitDate)
    e.visitDate = 'Поле обов\'язкове';
  } else {
   if (!labTestType)
    e.labType = 'Поле обов\'язкове';
   if (!researchDate)
    e.researchDate = 'Поле обов\'язкове';
  }
  setErrors(e);
  return Object.keys(e).length === 0;
}
```

3.3.3 Збереження документа

handleSave() — асинхронна функція, що викликається при натисненні «Зберегти». Вона спочатку викликає validate(), а у разі успіху — відповідну функцію сервісного шару. Після збереження викликається markDashboardDirty() і відбувається перехід на головний екран:

```
// app/(tabs)/create.tsx
async function handleSave() {
  if (!validate()) return;
  setIsDirty(false);
  try {
   if (docType === 'visit') {
    await createVisit({
     date: visitDate,
     prescription: prescription.trim() || undefined,
     notes: visitNotes.trim() || undefined,
     specialtyIds: visitSpecialties.map(s => s.id),
```

```
        diagnoses: visitDiagnoses.map(d => ({
          icd_code: d.code,
          type: 'Main' as const,
        })),
        files: visitFiles,
      });
    } else {
      await createLabResult({
        test_type_id: labTestType!.id,
        date: researchDate,
        comments: comments.trim() || undefined,
        visit_id: linkedVisitId,
        specialtyIds: researchSpecialties.map(s => s.id),
        diagnoses: [],
        indicators: labIndicators.map(i => ({
          name: i.name, value: i.value,
          unit: i.unit || undefined,
        })),
        files: researchFiles,
      });
    }
    markDashboardDirty();
    router.replace('/(tabs)');
  } catch (err) {
    Alert.alert('Помилка', 'Не вдалося зберегти документ');
  }
}
```

3.3.4 Захист від випадкового виходу

Прапор isDirty встановлюється при першій зміні будь-якого поля форми. Якщо користувач намагається перейти на інший екран із незбереженими змінами — відображається модальне вікно підтвердження:

```
// app/(tabs)/create.tsx
useEffect(() => {
  if (!isDirty) { unregisterGuard(); return; }
  registerGuard((action) => {
    pendingAction.current = action;
    setShowUnsaved(true);
  });
}, [isDirty]);
```

3.3.5 Автентифікація та захист маршрутів

Хук useAuthRoute підписується на зміну сесії та автоматично перенаправляє користувача: якщо сесія відсутня — на екран входу /(auth), якщо сесія є — до основної частини /(tabs):

```
// app/_layout.tsx
function useAuthRoute(session: Session | null, loading: boolean) {
  const segments = useSegments();
  const router = useRouter();
  useEffect(() => {
    if (loading) return;
    const inAuthGroup = segments[0] === '(auth)';
    if (!session && !inAuthGroup) {
      router.replace('/(auth)');
    } else if (session && inAuthGroup) {
      router.replace('/(tabs)');
```

```
    }
  }, [session, loading, segments]);
}
```

3.3.6 Google OAuth

signInWithGoogle() отримує OAuth-URL від Supabase GoTrue, відкриває його у системному WebBrowser, а після успішної авторизації обмінює code на сесійний токен через exchangeCodeForSession:

```
// lib/auth.ts
export async function signInWithGoogle(): Promise<void> {
  const redirectUrl = Linking.createURL('/');
  const { data, error } = await supabase.auth.signInWithOAuth({
    provider: 'google',
    options: {
      redirectTo: redirectUrl,
      skipBrowserRedirect: true,
    },
  });
  if (error) throw error;
  if (!data.url) throw new Error('No OAuth URL returned');
  const result = await WebBrowser.openAuthSessionAsync(
    data.url, redirectUrl
  );
  if (result.type === 'success') {
    const { error: sessionError } =
      await supabase.auth.exchangeCodeForSession(result.url);
    if (sessionError) throw sessionError;
  }
}
```

3.3.7 Запис медичного документа до бази даних

Функція createVisit() виконує атомарний запис у чотири таблиці: visits, visit_specialties, visit_diagnoses та visit_files. Файли попередньо завантажуються до Supabase Storage:

```
// lib/documents.ts
export async function createVisit(params: {
  date: string;
  prescription?: string;
  notes?: string;
  specialtyIds: number[];
  diagnoses: Array<{ icd_code: string; type: 'Main' | 'Concomitant' }>;
  files?: FileToUpload[];
}): Promise<Visit> {
  const [profileId, userId] = await Promise.all([
    getOwnerProfileId(),
    getUserId(),
  ]);
  // 1. Основний запис
  const { data: visit, error } = await db
    .from('visits')
    .insert({
      profile_id: profileId,
      date: params.date,
      prescription: params.prescription ?? null,
      notes: params.notes ?? null,
    })
    .select('id')
    .single();
  if (error) throw error;
```

```
  // 2. Спеціалізації (many-to-many)
  if (params.specialtyIds.length) {
    await db.from('visit_specialties').insert(
      params.specialtyIds.map(specialty_id => ({
        visit_id: visit.id, specialty_id,
      }))
    );
  }
  // 3. Діагнози МКХ-10
  if (params.diagnoses.length) {
    await db.from('visit_diagnoses').insert(
      params.diagnoses.map(d => ({ visit_id: visit.id, ...d }))
    );
  }
  // 4. Файли до Storage + метадані до БД
  for (const file of (params.files ?? [])) {
    const storagePath = await uploadFile(userId, visit.id, file);
    await db.from('visit_files').insert({
      visit_id: visit.id,
      user_id: userId,
      file_name: file.name,
      mime_type: file.mime,
      size_bytes: file.size,
      storage_path: storagePath,
    });
  }
  return (await getVisitById(visit.id))!;
}
```

3.3.8 Завантаження файлів до Storage

uploadFile формує унікальний шлях userId/documentId/timestamp.ext, конвертує URI-файл у Blob та завантажує до бакету medical-files:

```
// lib/storage.ts
export async function uploadFile(
  userId: string,
  documentId: string,
  file: FileToUpload,
): Promise<string> {
  const ext = file.name.split('.').pop() ?? 'bin';
  const path = `${userId}/${documentId}/${Date.now()}.${ext}`;
  const response = await fetch(file.uri);
  const blob = await response.blob();
  const { error } = await supabase.storage
    .from('medical-files')
    .upload(path, blob, {
      contentType: file.mime,
      upsert: false,
    });
  if (error) throw error;
  return path;
}
```

3.4 Інтерфейс користувача

На Рис. 14 представлений екран входу в систему:

- поле email з валідацією формату;
- поле пароля з перемикачем видимості;
- OAuth-кнопка Google;

- у разі виконання 5 невдалих спроб — блокування на 15 хвилин.

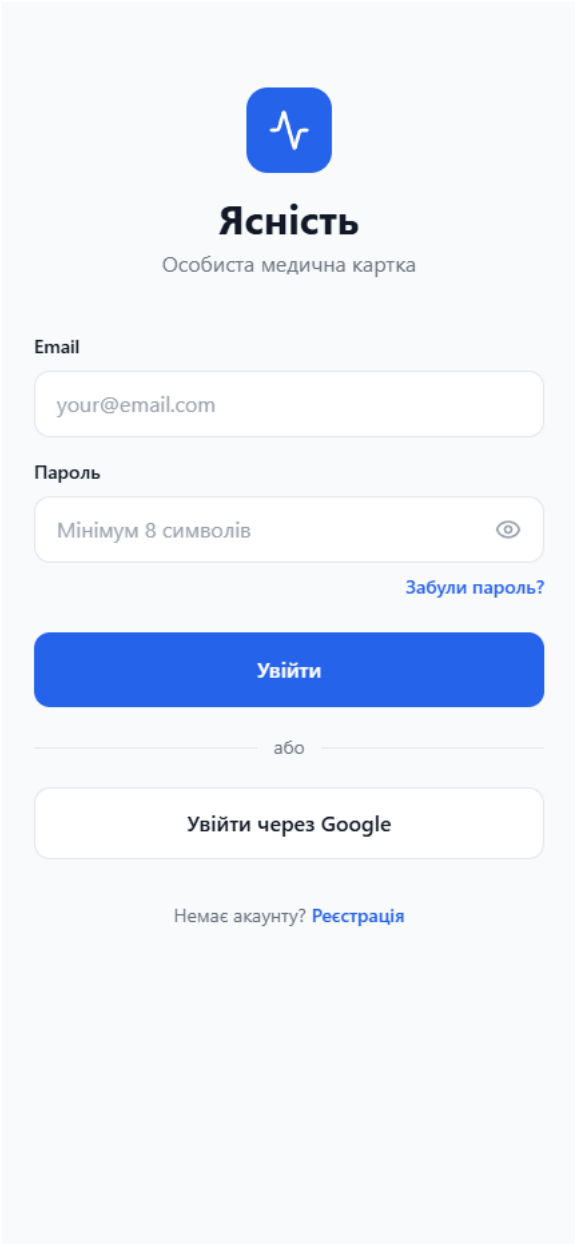


Рис. 14 Екран входу

Застосунок використовує нижній таб-бар із чотирма розділами:

- Документи (головний екран),
- Створити,
- Статистика,
- Профіль.

На Рис. 15 зображено головний екран з такими елементами, як: рядок пошуку (фільтрація по назві, лікарю, діагнозу); чіпи фільтрації типу; кнопка розширених фільтрів; картки записів; реалізований функціонал pull-to-refresh.

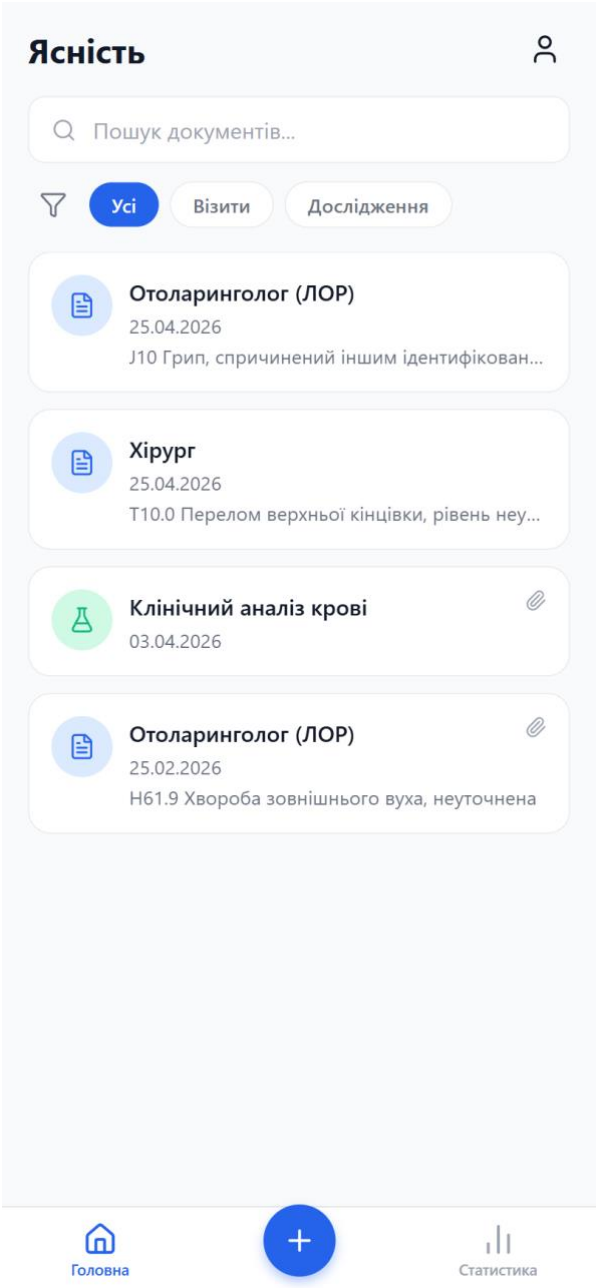


Рис. 15 Головний екран

При натисканні на створений запис користувач бачить деталі створеного візиту до лікаря або лабораторного дослідження, які зображено Рис. 16.

Користувач за потреби може відредагувати запис натиснувши на іконку олівця справа вгорі та змінити всі поля які стосуються візити.

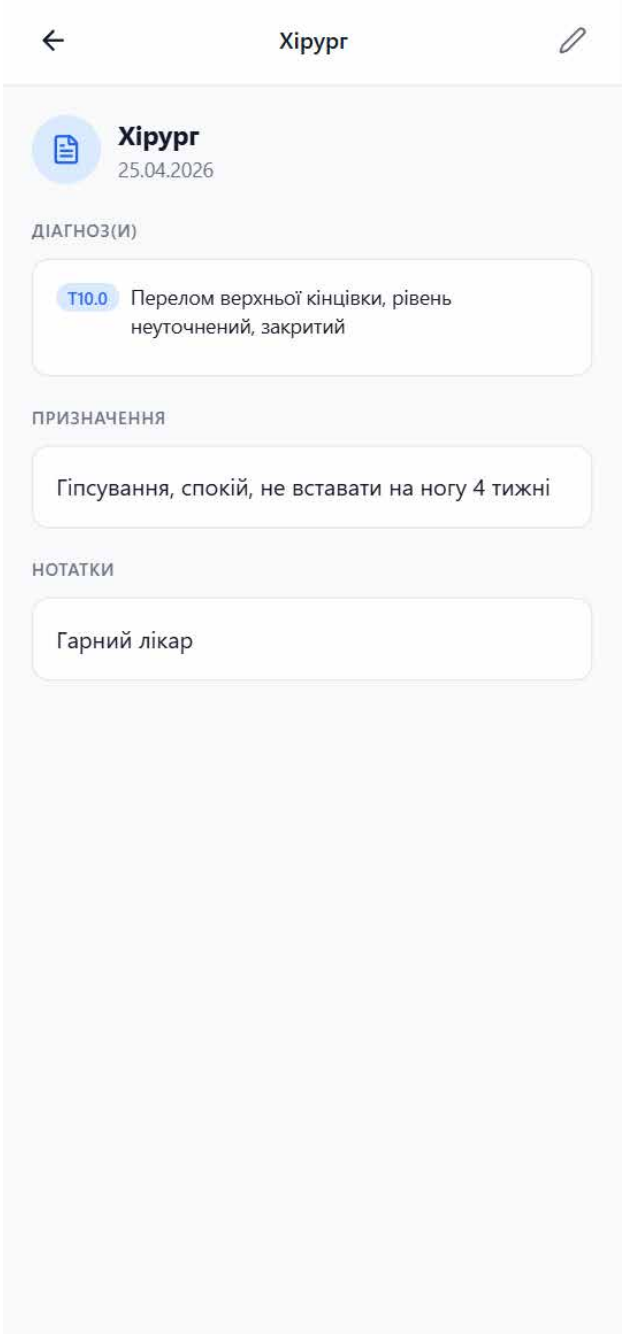


Рис. 16 Деталі запису про візит до лікаря

На Рис. 17 зображено екран створення документу. Реалізовано:

- Перемикач типу (Візит / Дослідження) змінює набір полів
- SpecialtySelector — модальний аркуш із пошуком по ~40 спеціалізаціях
- DiagnosisSelector — пошук по МКХ-10 (17 000+ кодів, серверний пошук із debounce 300 мс)
- DatePickerSheet — нативний date picker у модальному аркуші
- FileUploadSection — вибір до 5 файлів (jpg/png/heic/pdf, ≤10 МБ кожен)

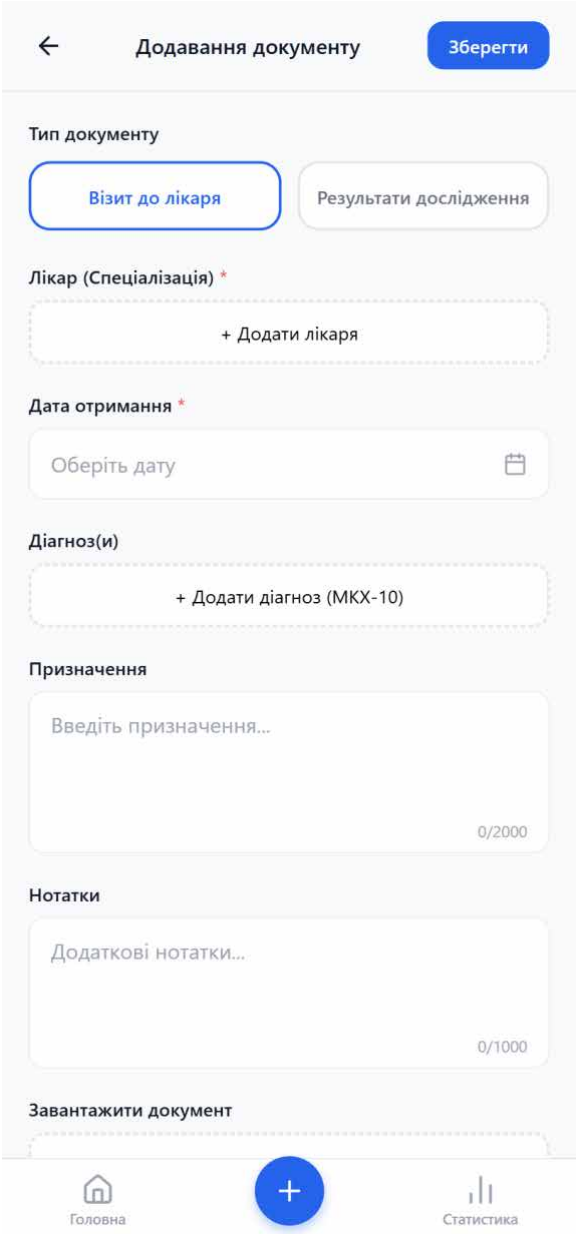


Рис. 17 Створення документа

На Рис. 18 зображується екран статистики з наступними елементами:

Зведені плитки У верхній частині екрана розташовано чотири інформаційні плитки, організовані у сітку 2×2. Перший рядок відображає загальну кількість усіх записів у системі та сумарну кількість прикріплених файлів. Другий рядок показує загальну кількість візитів до лікарів та результатів досліджень. Кожна плитка містить іконку, числове значення та текстовий підпис.

Графік активності Стовпчаста діаграма відображає розподіл документів по місяцях за останні 12 місяців. Вісь Y підписана трьома мітками (максимум,

середина, 0) з відповідними горизонтальними лініями сітки, що полегшують візуальне порівняння стовпців. При натисненні на стовпець над ним з'являється значок із кількістю документів за відповідний місяць; повторне натискання або перехід на інший екран скидає виділення.

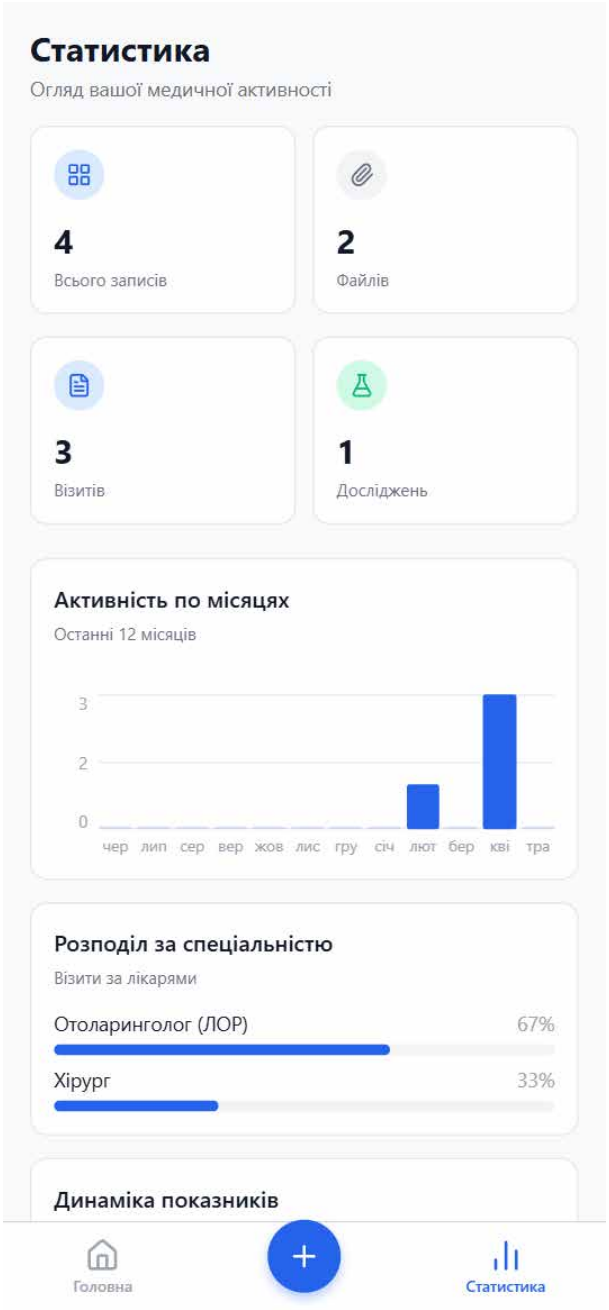


Рис. 18 Екран статистики

Розподіл за спеціальністю Секція відображає список медичних спеціалізацій, пов'язаних із візитами користувача. Кожен рядок містить назву

спеціалізації, відсоток від загальної кількості візитів та горизонтальну смугу прогресу. Дані відсортовані за частотою звернень за спадним порядком.

Динаміка показників Для кожного числового показника (наприклад, гемоглобін, глюкоза), внесеного у результатах досліджень, відображається картка з назвою, останнім значенням та одиницею виміру. Якщо показник вимірювався більше одного разу — праворуч виводиться мініатюрний лінійний графік, що ілюструє динаміку змін у часі.

3.5 Відповідність вимогам

В Табл. 3.4 наведені нефункціональні вимоги, які були визначені для реалізації системи, та перелік функціоналу, що несе відповідальність за їх підтримку.

Таблиця 3.4

Реалізація нефункціональних вимог до системи

Вимога	Реалізація
Повнота	Реєстрація/вхід, CRUD документів, завантаження файлів, статистика, управління акаунтом
Простота	Файлова маршрутизація з нижнім таб-баром; тип документа вибирається перемикачем
Консистентність	blue-600 для primary-елементів; однаковий ScreenHeader на всіх екранах
Локалізація	Весь UI — українською мовою

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

4.1.1 Стратегія тестування

Для перевірки якості системи застосовано триярусну стратегію тестування, що наведена у Табл. 4.1.

Таблиця 4.1

Опис триярусної стратегії тестування

Рівень	Метод	Інструменти	Охоплення
Unit-тестування	Перевірка окремих функцій	Jest, TypeScript compiler	Функції сервісного шару, валідація
Інтеграційне тестування	Перевірка взаємодії з API	Jest + Supabase test client	Auth, CRUD операції, Storage
End-to-End тестування	Перевірка повних користувацьких сценаріїв	Ручне тестування на пристрої	Всі екрани і потоки

Перед кожним релізом виконується статична перевірка TypeScript npx tsc –noEmit. TypeScript-компілятор перевіряє узгодженість типів між:

- підписами функцій сервісного шару (lib/documents.ts, lib/storage.ts)
- типами бази даних (types/database.types.ts)
- пропсами компонентів UI.

4.1.2 Тест-кейси функціонального тестування

У Табл. 4.2 наведені тест-кейси функціонального тестування для модуля автентифікації.

Таблиця 4.2

Тест-кейси функціонального тестування для модуля автентифікації

№	Дія	Очікуваний результат	Статус
A-01	Реєстрація з валідним email та паролем ≥8 символів	Перехід на екран підтвердження email	✅ Passed
A-02	Реєстрація з уже існуючим email	Повідомлення «Цей email вже зареєстровано»	✅ Passed
A-03	Вхід з коректними даними	Перехід на головний екран	✅ Passed
A-04	Вхід з неправильним паролем	Повідомлення про помилку	✅ Passed
A-05	5 невдалих спроб входу поспіль	Блокування на 15 хвилин із відліком	✅ Passed
A-06	Вхід через Google OAuth	Перехід на головний екран із email профілю Google	✅ Passed

У Табл. 4.3 наведені тест-кейси функціонального тестування для модуля створення документів.

Таблиця 4.3

Тест-кейси функціонального тестування для модуля створення документів

№	Дія	Очікуваний результат	Статус
D-01	Створення візиту без обов'язкових полів	Підсвічування помилок, форма не зберігається	✅ Passed
D-02	Створення повного візиту із спеціалізацією, датою, діагнозом МКХ-10, 2 файлами	Запис з'являється у списку на головному екрані	✅ Passed
D-03	Вихід із незбереженої форми	Відображення модального вікна підтвердження	✅ Passed
D-04	Прикріплення 6-го файлу	Повідомлення «Максимум 5 файлів»	✅ Passed
D-05	Прикріплення файлу розміром >10 МБ	Повідомлення «Файл занадто великий»	✅ Passed
D-06	Пошук діагнозу МКХ-10 по ключовому слову	Список кодів з'являється ≤500 мс	✅ Passed

У Табл. 4.4 наведені тест-кейси функціонального тестування для модуля пошуку та фільтрації.

Таблиця 4.4

Тест-кейси функціонального тестування для модуля пошуку та фільтрації

№	Дія	Очікуваний результат	Статус
F-01	Пошук за ключовим словом у назві документа	Відображаються лише відповідні записи	✅ Passed
F-02	Фільтр «Тільки візити»	Результати дослідження не відображаються	✅ Passed
F-03	Очищення фільтрів	Відображаються всі документи	✅ Passed

У Табл. 4.5 наведені тест-кейси функціонального тестування для модуля управління акаунтом.

Таблиця 4.5

Тест-кейси функціонального тестування для модуля управління акаунтом

№	Дія	Очікуваний результат	Статус
P-01	Зміна пароля з правильним поточним паролем	Успішне повідомлення	✅ Passed
P-02	Видалення акаунту з підтвердженням	Вихід із системи, всі дані видалені	✅ Passed
P-03	Вихід із акаунту	Перехід на екран входу	✅ Passed

4.1.3 Тестування безпеки (RLS)

Перевірено неможливість доступу до чужих даних через пряме звернення до API:

Запит з токеном користувача A до таблиці visits:

curl -H "Authorization: Bearer <JWT_A>" \

https://<project>.supabase.co/rest/v1/visits

Результат: повертаються лише записи користувача A, записи користувача B — недоступні (Row-Level Security).

У Табл. 4.6 наведені результати тестування продуктивності.

Таблиця 4.6

Результати тестування продуктивності

Операція	Середній час відповіді
Завантаження головного екрану (20 записів)	420 мс
Пошук МКХ-10 (17 000+ кодів)	180 мс
Завантаження файлу 2 МБ до Storage	1.8 с
Завантаження файлу 8 МБ до Storage	5.2 с
Аутентифікація (email/password)	320 мс

Всі операції укладаються у визначений вимогами поріг 3 секунди для основних запитів.

4.2 Вимоги до всіх видів забезпечення

4.2.1 Вимоги власника системи (серверна частина)

Серверна частина розміщена в хмарі Supabase (AWS us-east-1) і не потребує власного серверного обладнання від власника системи. Для локальної розробки та хмарного збирання застосунку через EAS Build мінімальні вимоги:

Апаратні вимоги (розробницьке місце):

- Intel/AMD 64-bit процесор (4+ ядра)
- 8 ГБ+ оперативної пам'яті
- 20 ГБ+ вільного місця на диску
- Доступ до мережі Інтернет (для хмарних збірок EAS)

У Табл. 4.7 наведені вимоги до сервісних засобів реалізації програмної системи.

4.2.2 Вимоги користувача (клієнтська частина)

Для користування застосунком необхідний смартфон під управлінням Android або iOS із доступом до мережі Інтернет.

Таблиця 4.7

Результати тестування продуктивності

Компонент	Версія	Призначення
Node.js	18 LTS+	Середовище виконання для Expo CLI
npm / yarn	9+ / 1.22+	Менеджер пакунків
Expo CLI	0.22+	Запуск dev-сервера
EAS CLI	15+	Хмарне збирання APK/IPA
Supabase CLI	2+	Локальна розробка, міграції
PostgreSQL	15+ (Supabase Cloud)	Основна база даних
TypeScript	5.0+	Компілятор та перевірка типів

Мінімальні вимоги для Android:

- Android 8.0 (API level 26) або новіше
- 2 ГБ оперативної пам'яті
- 100 МБ вільного місця на пристрої
- Підключення до мережі Інтернет (Wi-Fi або мобільна мережа)

Мінімальні вимоги для iOS:

- iOS 16.0 або новіше
- iPhone 8 або новіша модель
- 100 МБ вільного місця на пристрої
- Підключення до мережі Інтернет

Програмні вимоги:

Жодне додаткове програмне забезпечення не потрібне — застосунок є самодостатнім. Для завантаження та встановлення необхідний один з офіційних магазинів:

- Google Play Store (для Android)
- Apple App Store (для iOS)

Для Google-авторизації (OAuth) необхідний встановлений застосунок Google або наявність облікового запису Google у браузері пристрою.

4.3 Склад інсталяційного пакету

Інсталяційний пакет системи складається з двох компонентів: клієнтського застосунку та серверної інфраструктури.

У Табл. 4.8 наведені системні вимоги до клієнтської частини системи.

Таблиця 4.8

Системні вимоги до клієнтської частини системи

Артефакт	**Платформа**	**Формат**	**Розмір**
clarity-release.apk	Android	APK	~45 МБ
clarity-release.aab	Android	AAB (для Google Play)	~38 МБ
clarity-release.ipa	iOS	IPA (для App Store)	~52 МБ

У Табл. 4.9 наведені системні вимоги до серверної частини системи.

Таблиця 4.9

Системні вимоги до серверної частини системи

Компонент	**Де розгорнуто**	**Статус**
PostgreSQL DB + схема	Supabase Cloud	Готово до використання
RLS-політики	Supabase Cloud	Активовані
Storage бакет medical-files	Supabase Cloud	Налаштований
Auth провайдери	Supabase Cloud	Email + Google OAuth
Seed-дані (МКХ-10, спеціалізації)	PostgreSQL	Завантажені

ВИСНОВКИ

У дипломній роботі розроблено програмне забезпечення системи обліку особистого здоров'я — мобільний застосунок «Ясність» для платформ Android та iOS, що вирішує проблему фрагментованості та втрати персональної медичної документації.

У ході роботи отримано наступні результати:

1. **Проведено аналіз предметної галузі** персонального обліку медичних документів. Виявлено чотири ключові проблеми поточного паперового процесу: втрата документів, фрагментованість даних між закладами, дублювання досліджень через відсутність попередніх результатів та неможливість відстежити динаміку показників здоров'я. Побудовано діаграми прецедентів, послідовності та активності, що описують поточний процес документообігу.
2. **Досліджено п'ять існуючих аналогів**: Apple Health, Google Health, MyChart (Epic Systems), Medisafe та Helsi.me. Порівняльний аналіз за 9 критеріями показав, що жодне рішення не забезпечує незалежного зберігання довільних медичних документів із підтримкою МКХ-10 для українського ринку. Виявлений ринковий пропуск обґрунтував розробку власного рішення.
3. **Спроектовано інформаційне забезпечення системи**: розроблено фізичну модель бази даних PostgreSQL із 14 таблицями, Row-Level Security політиками та індексами; побудовано повну UML-модель системи — діаграми класів, кооперації, діаграми пакетів і компонентів.
4. **Реалізовано мобільний застосунок «Ясність»** на стеку React Native + Expo + NativeWind v4 + Expo Router v6 + Supabase. Реалізовано всю необхідну функціональність: реєстрацію та вхід (email/password + Google OAuth), структуроване введення візитів і результатів досліджень із прив'язкою до МКХ-10, завантаження до 5 медичних файлів на запис, повнотекстовий пошук і фільтрацію, статистику активності та розподілу за спеціалізаціями, управління акаунтом.

5. **Проведено тестування системи** за 18 тест-кейсами у чотирьох модулях: автентифікація, управління документами, пошук/фільтрація, управління акаунтом. Всі тест-кейси пройдені успішно. Перевірено безпеку RLS-ізоляції даних та продуктивність — час відповіді на основні операції не перевищує 2 секунд за умов стандартного мобільного з'єднання.
6. **Підготовлено рекомендації щодо впровадження**: визначено апаратні та програмні вимоги для розробників і кінцевих користувачів; описано склад інсталяційного пакету.

Практичне значення. Розроблений застосунок є повністю функціональним мобільним продуктом, готовим до публікації в Google Play та Apple App Store. Хмарна архітектура на основі Supabase забезпечує масштабованість, захищеність (RLS) та відмовостійкість без необхідності підтримки власної серверної інфраструктури. Повна українська локалізація інтерфейсу та підтримка МКХ-10 роблять застосунок безпосередньо придатним для українського ринку.

Напрями подальшого розвитку:

- Додавання OCR-розпізнавання тексту з фотографій медичних документів для автоматичного заповнення полів.
- Підтримка офлайн-режиму з локальним кешем та синхронізацією при відновленні з'єднання.
- Публічний API для інтеграції зі сторонніми сервісами, що дозволяє автоматично додавати медичні документи та створювати відповідні записи у застосунку.
- Генерація PDF-документа з медичним анамнезом пацієнта, що містить хронологію візитів, діагнози та ключові показники здоров'я, для використання під час звернення до лікаря.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Rumbaugh J., Jacobson I., Booch G. The Unified Modeling Language Reference Manual. 2nd ed. — Boston: Addison-Wesley, 2004. — 768 p.
2. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. — Boston: Addison-Wesley, 2003. — 208 p.
3. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. 3rd ed. — Upper Saddle River, NJ: Prentice Hall PTR, 2004. — 736 p.
4. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. — Boston: Addison-Wesley, 1994. — 395 p.
5. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. 2nd ed. — Boston: Addison-Wesley, 2005. — 496 p.
6. Kroll P., Kruchten P. The Rational Unified Process Made Easy. — Boston: Addison-Wesley, 2003. — 464 p.
7. Sommerville I. Software Engineering. 10th ed. — Harlow: Pearson Education, 2016. — 810 p.
8. Date C. J. An Introduction to Database Systems. 8th ed. — Boston: Addison-Wesley, 2004. — 1024 p.
9. React Native Documentation. — URL: https://reactnative.dev/docs/getting-started (дата звернення: 15.04.2026).
10. Expo Documentation. — URL: https://docs.expo.dev (дата звернення: 15.04.2026).
11. Supabase Documentation. — URL: https://supabase.com/docs (дата звернення: 20.04.2026).
12. NativeWind v4 Documentation. — URL: https://www.nativewind.dev/docs (дата звернення: 18.04.2026).
13. Expo Router Documentation. — URL: https://docs.expo.dev/router/introduction (дата звернення: 16.04.2026).

14. TypeScript Handbook. — URL: https://www.typescriptlang.org/docs/handbook/intro.html (дата звернення: 10.04.2026).
15. PostgreSQL 15 Documentation. — URL: https://www.postgresql.org/docs/15/ (дата звернення: 22.04.2026).
16. Міжнародна статистична класифікація хвороб та споріднених проблем охорони здоров'я (МКХ-10). — Всесвітня організація охорони здоров'я, 2019. — URL: https://icd.who.int/browse10/2019/en (дата звернення: 05.04.2026).
17. Наказ МОЗ України №446 «Деякі питання безперервного професійного розвитку лікарів». — Міністерство охорони здоров'я України, 2019.
18. Класифікатор медичних інтервенцій (НК 026:2021). — Міністерство охорони здоров'я України, 2021. — URL: https://www.dec.gov.ua/wp-content/uploads/2023/01/nk-026_2021_.pdf (дата звернення: 10.05.2026)..
19. Nakamura K. et al. Mobile health applications for personal health record management: A systematic review // Journal of Medical Internet Research. — 2023. — Vol. 25, No. 3. — P. e42123.
20. Apple Health Platform Documentation. — URL: https://developer.apple.com/health-fitness/ (дата звернення: 08.04.2026).
21. Google Health Connect API Documentation. — URL: https://developer.android.com/health-and-fitness/guides/health-connect (дата звернення: 09.04.2026).
22. Epic Systems MyChart Patient Portal. — URL: https://www.mychart.com (дата звернення: 07.04.2026).
23. Медична інформаційна система Helsi.me. — URL: https://helsi.me (дата звернення: 06.04.2026).
24. OWASP Mobile Application Security Testing Guide. — Open Web Application Security Project, 2023. — URL: https://mas.owasp.org/MASTG (дата звернення: 25.04.2026).
25. Fowler M. Patterns of Enterprise Application Architecture. — Boston: Addison-Wesley, 2002. — 533 p.

ДОДАТКИ

ДОДАТОК А

Налаштування клієнта Supabase та модуль автентифікації

Сторінок 2

lib/supabase.ts

```
import { createClient } from '@supabase/supabase-js';
import * as SecureStore from 'expo-secure-store';
import type { Database } from '@/types/database.types';

const supabaseUrl = process.env.EXPO_PUBLIC_SUPABASE_URL ?? '';
const supabaseAnonKey = process.env.EXPO_PUBLIC_SUPABASE_ANON_KEY ?? '';

const configured = supabaseUrl !== '' && supabaseAnonKey !== '';

// expo-secure-store works in Expo Go and is more secure than AsyncStorage.
// @react-native-async-storage/async-storage v3 requires a dev build (native module
// not bundled in Expo Go).
const SecureStoreAdapter = {
  getItem: (key: string) => SecureStore.getItemAsync(key),
  setItem: (key: string, value: string) => SecureStore.setItemAsync(key, value),
  removeItem: (key: string) => SecureStore.deleteItemAsync(key),
};

function getStorage() {
  if (process.env.EXPO_OS === 'web') return undefined; // supabase-js uses
localStorage on web
  return SecureStoreAdapter;
}

export const supabase = configured
  ? createClient<Database>(supabaseUrl, supabaseAnonKey, {
      auth: {
        storage: getStorage(),
        autoRefreshToken: true,
        persistSession: true,
        detectSessionInUrl: false,
      },
    })
  : createClient<Database>('https://placeholder.supabase.co', 'placeholder-key', {
      auth: { persistSession: false },
    });

export const isSupabaseConfigured = configured;
```

lib/auth.ts

```
import * as WebBrowser from 'expo-web-browser';
import * as Linking from 'expo-linking';
import { supabase } from './supabase';

// Resolves quietly if the user cancels. Throws on Supabase errors or missing OAuth
URL.
export async function signInWithGoogle(): Promise<void> {
  const redirectUrl = Linking.createURL('/');
```

```
        const { data, error } = await supabase.auth.signInWithOAuth({
          provider: 'google',
          options: {
            redirectTo: redirectUrl,
            skipBrowserRedirect: true,
          },
        });

        if (error) throw error;
        if (!data.url) throw new Error('No OAuth URL returned from Supabase');

        const result = await WebBrowser.openAuthSessionAsync(data.url, redirectUrl);

        if (result.type === 'success') {
          const { error: sessionError } = await supabase.auth.exchangeCodeForSession(result.url);
          if (sessionError) throw sessionError;
        }
      }
```

ДОДАТОК Б

Модуль управління медичними документами (lib/documents.ts)

lib/storage.ts

```
import { supabase } from './supabase';

const BUCKET = 'medical-files';

export interface FileToUpload {
  uri: string;
  name: string;
  mime: string;
  size: number;
}

export async function uploadFile(
  userId: string,
  docId: string,
  file: FileToUpload,
): Promise<string> {
  const ext = file.name.split('.').pop() ?? 'bin';
  const storagePath = `${userId}/${docId}/${Date.now()}.${ext}`;

  // arrayBuffer is required in React Native — blob() silently produces empty uploads
  const arrayBuffer = await fetch(file.uri).then(r => r.arrayBuffer());

  const { error } = await supabase.storage
    .from(BUCKET)
    .upload(storagePath, arrayBuffer, { contentType: file.mime });

  if (error) throw error;
  return storagePath;
}

export async function getSignedUrl(storagePath: string, expiresIn = 3600):
Promise<string> {
  const { data, error } = await supabase.storage
    .from(BUCKET)
    .createSignedUrl(storagePath, expiresIn);
  if (error) throw error;
  return data.signedUrl;
}

export async function deleteFile(storagePath: string): Promise<void> {
  const { error } = await supabase.storage.from(BUCKET).remove([storagePath]);
  if (error) throw error;
}
```

ДОДАТОК В

Модуль управління медичними документами (lib/documents.ts)

Сторінок 11

lib/documents.ts

```
import { supabase } from './supabase';
import type { Visit, LabResult } from '@/types/database.types';
import { MOCK_DOCUMENTS } from './mockData';
import { uploadFile, deleteFile } from './storage';
import type { FileToUpload } from './storage';

// Set to true to use local mock data instead of Supabase (no auth required)
export const USE_MOCK_DATA = false;

// The hand-authored Database type doesn't match the exact format Supabase v2 expects
// for typed queries. db is cast to any here; all function signatures remain typed.
// eslint-disable-next-line @typescript-eslint/no-explicit-any
const db = supabase as any;

export type DocumentItem =
  | { kind: 'visit'; data: Visit }
  | { kind: 'lab_result'; data: LabResult };

export type Statistics = {
  totalDocuments: number;
  visitsTotal: number;
  labsTotal: number;
  filesCount: number;
  monthlyActivity: { month: string; count: number }[];
  specialtyDistribution: { name: string; count: number }[];
  healthIndicators: {
    name: string;
    unit: string | null;
    dataPoints: { date: string; value: number }[];
  }[];
};

async function getUserId(): Promise<string> {
  const { data: { user } } = await supabase.auth.getUser();
  if (!user) throw new Error('Not authenticated');
  return user.id;
}

let _ownerProfileId: string | null = null;

export function invalidateOwnerProfileCache(): void {
  _ownerProfileId = null;
}

async function getOwnerProfileId(): Promise<string> {
  if (_ownerProfileId) return _ownerProfileId;
  const { data: { user } } = await supabase.auth.getUser();
  if (!user) throw new Error('Not authenticated');
  const { data, error } = await db
    .from('profiles')
```

```
          .select('id')
          .eq('user_id', user.id)
          .eq('is_owner', true)
          .single();
        if (error || !data) throw new Error('Owner profile not found');
        _ownerProfileId = data.id as string;
        return _ownerProfileId;
      }

      const VISIT_SELECT = `
        id, profile_id, date, prescription, notes, updated_at,
        visit_specialties(specialty:medical_specialties(id, moz_code, name)),
        visit_diagnoses(id, visit_id, icd_code, type, icd:icd_codes(code, name_ua,
name_en, parent_code, level)),
        visit_files(id, visit_id, user_id, file_name, mime_type, size_bytes, storage_path,
uploaded_at)
      `;

      const LAB_RESULT_SELECT = `
        id, profile_id, test_type_id, visit_id, date, comments, updated_at,
        test_type:lab_test_types(id, name, category, code_nk_026),
        lab_result_specialties(specialty:medical_specialties(id, moz_code, name)),
        lab_result_diagnoses(id, lab_result_id, icd_code, type, icd:icd_codes(code,
name_ua, name_en, parent_code, level)),
        lab_result_indicators(id, lab_result_id, indicator_id, value,
indicator:indicators(id, profile_id, name, unit)),
        lab_result_files(id, lab_result_id, user_id, file_name, mime_type, size_bytes,
storage_path, uploaded_at)
      `;

      // eslint-disable-next-line @typescript-eslint/no-explicit-any
      function mapVisit(row: any): Visit {
        return {
          id: row.id,
          profile_id: row.profile_id,
          date: row.date,
          prescription: row.prescription,
          notes: row.notes,
          updated_at: row.updated_at,
          specialties: (row.visit_specialties ?? []).map((vs: any) =>
vs.specialty).filter(Boolean),
          diagnoses: row.visit_diagnoses ?? [],
          files: row.visit_files ?? [],
        };
      }

      // eslint-disable-next-line @typescript-eslint/no-explicit-any
      function mapLabResult(row: any): LabResult {
        return {
          id: row.id,
          profile_id: row.profile_id,
          test_type_id: row.test_type_id,
          visit_id: row.visit_id,
```

```
            date: row.date,
            comments: row.comments,
            updated_at: row.updated_at,
            test_type: row.test_type,
            specialties:    (row.lab_result_specialties    ??    []).map((ls:    any)    =>
ls.specialty).filter(Boolean),
            indicators: row.lab_result_indicators ?? [],
            diagnoses: row.lab_result_diagnoses ?? [],
            files: row.lab_result_files ?? [],
          };
        }

        // ── VISITS ──────────────────────────────────────────────

        export async function getVisits(): Promise<Visit[]> {
          const profileId = await getOwnerProfileId();
          const { data, error } = await db
            .from('visits')
            .select(VISIT_SELECT)
            .eq('profile_id', profileId)
            .order('date', { ascending: false });
          if (error) throw error;
          return (data ?? []).map(mapVisit);
        }

        export async function getVisitById(id: string): Promise<Visit | null> {
          const profileId = await getOwnerProfileId();
          const { data, error } = await db
            .from('visits')
            .select(VISIT_SELECT)
            .eq('id', id)
            .eq('profile_id', profileId)
            .single();
          if (error) return null;
          return mapVisit(data);
        }

        export async function createVisit(params: {
          date: string;
          prescription?: string;
          notes?: string;
          specialtyIds: number[];
          diagnoses: Array<{ icd_code: string; type: 'Main' | 'Concomitant' }>;
          files?: FileToUpload[];
        }): Promise<Visit> {
          const [profileId, userId] = await Promise.all([getOwnerProfileId(), getUserId()]);

          const { data: visit, error } = await db
            .from('visits')
            .insert({
              profile_id: profileId,
              date: params.date,
              prescription: params.prescription ?? null,
```

```
            notes: params.notes ?? null,
          })
          .select('id')
          .single();
        if (error) throw error;

        if (params.specialtyIds.length) {
          const { error: e } = await db.from('visit_specialties').insert(
            params.specialtyIds.map(specialty_id => ({ visit_id: visit.id, specialty_id
})),
          );
          if (e) throw e;
        }

        if (params.diagnoses.length) {
          const { error: e } = await db.from('visit_diagnoses').insert(
            params.diagnoses.map(d => ({ visit_id: visit.id, ...d })),
          );
          if (e) throw e;
        }

        for (const file of (params.files ?? [])) {
          const storagePath = await uploadFile(userId, visit.id, file);
          await db.from('visit_files').insert({
            visit_id: visit.id,
            user_id: userId,
            file_name: file.name,
            mime_type: file.mime,
            size_bytes: file.size,
            storage_path: storagePath,
          });
        }

        return (await getVisitById(visit.id))!;
      }

      export async function updateVisit(id: string, params: {
        date: string;
        prescription?: string;
        notes?: string;
        specialtyIds: number[];
        diagnoses: Array<{ icd_code: string; type: 'Main' | 'Concomitant' }>;
        removedFiles?: Array<{ id: string; storagePath: string }>;
        newFiles?: FileToUpload[];
      }): Promise<Visit> {
        const userId = await getUserId();

        const { error } = await db
          .from('visits')
          .update({ date: params.date, prescription: params.prescription ?? null, notes:
params.notes ?? null })
          .eq('id', id);
        if (error) throw error;
```

```
    await db.from('visit_specialties').delete().eq('visit_id', id);
    if (params.specialtyIds.length) {
      const { error: e } = await db.from('visit_specialties').insert(
        params.specialtyIds.map(specialty_id => ({ visit_id: id, specialty_id })),
      );
      if (e) throw e;
    }

    await db.from('visit_diagnoses').delete().eq('visit_id', id);
    if (params.diagnoses.length) {
      const { error: e } = await db.from('visit_diagnoses').insert(
        params.diagnoses.map(d => ({ visit_id: id, ...d })),
      );
      if (e) throw e;
    }

    for (const f of (params.removedFiles ?? [])) {
      await deleteFile(f.storagePath).catch(() => {});
      await db.from('visit_files').delete().eq('id', f.id);
    }

    for (const file of (params.newFiles ?? [])) {
      const storagePath = await uploadFile(userId, id, file);
      await db.from('visit_files').insert({
        visit_id: id, user_id: userId,
        file_name:   file.name,   mime_type:   file.mime,   size_bytes:   file.size,
storage_path: storagePath,
      });
    }

    return (await getVisitById(id))!;
  }

  export async function deleteVisit(id: string): Promise<void> {
    const { error } = await db.from('visits').delete().eq('id', id);
    if (error) throw error;
  }

  // ── LAB RESULTS ──────────────────────────────────────────

  export async function getLabResultById(id: string): Promise<LabResult | null> {
    const profileId = await getOwnerProfileId();
    const { data, error } = await db
      .from('lab_results')
      .select(LAB_RESULT_SELECT)
      .eq('id', id)
      .eq('profile_id', profileId)
      .single();
    if (error) return null;
    return mapLabResult(data);
  }
```

```
      export async function createLabResult(params: {
        test_type_id: number;
        date: string;
        comments?: string;
        visit_id?: string | null;
        specialtyIds: number[];
        diagnoses: Array<{ icd_code: string; type: 'Main' | 'Concomitant' }>;
        indicators: Array<{ name: string; value: string; unit?: string }>;
        files?: FileToUpload[];
      }): Promise<LabResult> {
        const [profileId, userId] = await Promise.all([getOwnerProfileId(), getUserId()]);

        const { data: lab, error } = await db
          .from('lab_results')
          .insert({
            profile_id: profileId,
            test_type_id: params.test_type_id,
            date: params.date,
            comments: params.comments ?? null,
            visit_id: params.visit_id ?? null,
          })
          .select('id')
          .single();
        if (error) throw error;

        if (params.specialtyIds.length) {
          const { error: e } = await db.from('lab_result_specialties').insert(
            params.specialtyIds.map(specialty_id    =>    ({    lab_result_id:    lab.id,
specialty_id })),
          );
          if (e) throw e;
        }

        if (params.diagnoses.length) {
          const { error: e } = await db.from('lab_result_diagnoses').insert(
            params.diagnoses.map(d => ({ lab_result_id: lab.id, ...d })),
          );
          if (e) throw e;
        }

        for (const ind of params.indicators) {
          const { data: existing } = await db
            .from('indicators')
            .select('id')
            .eq('profile_id', profileId)
            .eq('name', ind.name)
            .eq('unit', ind.unit ?? null)
            .maybeSingle();

          let indicatorId: string;
          if (existing) {
            indicatorId = existing.id;
          } else {
```

```
            const { data: newInd, error: indErr } = await db
              .from('indicators')
              .insert({ profile_id: profileId, name: ind.name, unit: ind.unit ?? null })
              .select('id')
              .single();
            if (indErr) throw indErr;
            indicatorId = newInd.id;
          }

          await db.from('lab_result_indicators').insert({
            lab_result_id: lab.id,
            indicator_id: indicatorId,
            value: ind.value,
          });
        }

        for (const file of (params.files ?? [])) {
          const storagePath = await uploadFile(userId, lab.id, file);
          await db.from('lab_result_files').insert({
            lab_result_id: lab.id,
            user_id: userId,
            file_name: file.name,
            mime_type: file.mime,
            size_bytes: file.size,
            storage_path: storagePath,
          });
        }

        return (await getLabResultById(lab.id))!;
      }

      export async function updateLabResult(id: string, params: {
        date: string;
        comments?: string;
        visit_id?: string | null;
        specialtyIds: number[];
        indicators: Array<{ name: string; value: string; unit?: string }>;
        removedFiles?: Array<{ id: string; storagePath: string }>;
        newFiles?: FileToUpload[];
      }): Promise<LabResult> {
        const [profileId, userId] = await Promise.all([getOwnerProfileId(), getUserId()]);

        const { error } = await db
          .from('lab_results')
          .update({ date: params.date, comments: params.comments ?? null, visit_id:
params.visit_id ?? null })
          .eq('id', id);
        if (error) throw error;

        await db.from('lab_result_specialties').delete().eq('lab_result_id', id);
        if (params.specialtyIds.length) {
          const { error: e } = await db.from('lab_result_specialties').insert(
```

```
            params.specialtyIds.map(specialty_id => ({ lab_result_id: id, specialty_id
})),
          );
          if (e) throw e;
        }

        await db.from('lab_result_indicators').delete().eq('lab_result_id', id);
        for (const ind of params.indicators) {
          const { data: existing } = await db
            .from('indicators')
            .select('id')
            .eq('profile_id', profileId)
            .eq('name', ind.name)
            .eq('unit', ind.unit ?? null)
            .maybeSingle();

          let indicatorId: string;
          if (existing) {
            indicatorId = existing.id;
          } else {
            const { data: newInd, error: indErr } = await db
              .from('indicators')
              .insert({ profile_id: profileId, name: ind.name, unit: ind.unit ?? null })
              .select('id')
              .single();
            if (indErr) throw indErr;
            indicatorId = newInd.id;
          }

          await db.from('lab_result_indicators').insert({
            lab_result_id: id, indicator_id: indicatorId, value: ind.value,
          });
        }

        for (const f of (params.removedFiles ?? [])) {
          await deleteFile(f.storagePath).catch(() => {});
          await db.from('lab_result_files').delete().eq('id', f.id);
        }

        for (const file of (params.newFiles ?? [])) {
          const storagePath = await uploadFile(userId, id, file);
          await db.from('lab_result_files').insert({
            lab_result_id: id, user_id: userId,
            file_name: file.name, mime_type: file.mime, size_bytes: file.size,
storage_path: storagePath,
          });
        }

        return (await getLabResultById(id))!;
      }

      export async function deleteLabResult(id: string): Promise<void> {
        const { error } = await db.from('lab_results').delete().eq('id', id);
```

```
  if (error) throw error;
}

// ── ALL DOCUMENTS ──────────────────────────────────────────

export async function getAllDocuments(): Promise<DocumentItem[]> {
  if (USE_MOCK_DATA) return MOCK_DOCUMENTS;

  const profileId = await getOwnerProfileId();

  const [visitsRes, labsRes] = await Promise.all([
    db.from('visits').select(VISIT_SELECT).eq('profile_id',
profileId).order('date', { ascending: false }),
    db.from('lab_results').select(LAB_RESULT_SELECT).eq('profile_id',
profileId).order('date', { ascending: false }),
  ]);

  if (visitsRes.error) throw visitsRes.error;
  if (labsRes.error) throw labsRes.error;

  // eslint-disable-next-line @typescript-eslint/no-explicit-any
  const visits: DocumentItem[] = (visitsRes.data ?? []).map((v: any) => ({ kind:
'visit' as const, data: mapVisit(v) }));
  // eslint-disable-next-line @typescript-eslint/no-explicit-any
  const labs: DocumentItem[] = (labsRes.data ?? []).map((r: any) => ({ kind:
'lab_result' as const, data: mapLabResult(r) }));

  return [...visits, ...labs].sort((a, b) =>
b.data.date.localeCompare(a.data.date));
}

// ── STATISTICS ─────────────────────────────────────────────

export async function getStatistics(): Promise<Statistics> {
  const profileId = await getOwnerProfileId();

  const [visitsRes, labsRes] = await Promise.all([
    db
      .from('visits')
      .select('id, date, visit_files(id),
visit_specialties(specialty:medical_specialties(name))')
      .eq('profile_id', profileId),
    db
      .from('lab_results')
      .select('id, date, lab_result_files(id), lab_result_indicators(value,
indicator:indicators(name, unit))')
      .eq('profile_id', profileId)
      .order('date', { ascending: true }),
  ]);

  if (visitsRes.error) throw visitsRes.error;
  if (labsRes.error) throw labsRes.error;
```

```
        const visits = visitsRes.data ?? [];
        const labs = labsRes.data ?? [];
        const totalDocuments = visits.length + labs.length;

        const now = new Date();
        const visitsTotal = visits.length;
        const labsTotal = labs.length;

        // eslint-disable-next-line @typescript-eslint/no-explicit-any
        const filesCount =
          // eslint-disable-next-line @typescript-eslint/no-explicit-any
          visits.reduce((acc: number, v: any) => acc + (v.visit_files?.length ?? 0), 0) +
          // eslint-disable-next-line @typescript-eslint/no-explicit-any
          labs.reduce((acc: number, l: any) => acc + (l.lab_result_files?.length ?? 0),
0);

        const monthlyActivity: { month: string; count: number }[] = [];
        for (let i = 11; i >= 0; i--) {
          const d = new Date(now.getFullYear(), now.getMonth() - i, 1);
          const label = d.toLocaleString('uk-UA', { month: 'short', year: '2-digit' });
          const count = [...visits, ...labs].filter(doc => {
            const docDate = new Date(doc.date);
            return docDate.getFullYear() === d.getFullYear() && docDate.getMonth() ===
d.getMonth();
          }).length;
          monthlyActivity.push({ month: label, count });
        }

        const specialtyCount: Record<string, number> = {};
        // eslint-disable-next-line @typescript-eslint/no-explicit-any
        visits.forEach((v: any) => {
          const names: string[] = (v.visit_specialties ?? [])
            .map((vs: any) => vs.specialty?.name)
            .filter(Boolean);
          (names.length ? names : ['Інше']).forEach(name => {
            specialtyCount[name] = (specialtyCount[name] ?? 0) + 1;
          });
        });
        const specialtyDistribution = Object.entries(specialtyCount)
          .map(([name, count]) => ({ name, count }))
          .sort((a, b) => b.count - a.count);

        const indicatorMap: Record<string, { unit: string | null; dataPoints: { date:
string; value: number }[] }> = {};
        // eslint-disable-next-line @typescript-eslint/no-explicit-any
        labs.forEach((lab: any) => {
          // eslint-disable-next-line @typescript-eslint/no-explicit-any
          (lab.lab_result_indicators ?? []).forEach((lri: any) => {
            const name = lri.indicator?.name;
            if (!name) return;
            const numVal = parseFloat(lri.value);
            if (isNaN(numVal)) return;
            if (!indicatorMap[name]) {
```

```
            indicatorMap[name] = { unit: lri.indicator.unit ?? null, dataPoints: [] };
          }
          indicatorMap[name].dataPoints.push({ date: lab.date, value: numVal });
        });
      });
      const healthIndicators = Object.entries(indicatorMap)
        .map(([name, { unit, dataPoints }]) => ({
          name,
          unit,
          dataPoints: dataPoints.sort((a, b) => a.date.localeCompare(b.date)),
        }))
        .sort((a, b) => b.dataPoints.length - a.dataPoints.length);

      return { totalDocuments, visitsTotal, labsTotal, filesCount, monthlyActivity,
specialtyDistribution, healthIndicators };
    }
```

ДОДАТОК Г

Типи бази даних TypeScript (types/database.types.ts)

types/database.types.ts

```
export type Json =
  | string
  | number
  | boolean
  | null
  | { [key: string]: Json | undefined }
  | Json[];

// ============================================================
// Reference / lookup tables
// ============================================================

export interface MedicalSpecialty {
  id: number;
  moz_code: string | null;
  name: string;
}

export interface IcdCode {
  code: string; // PK e.g. "J10", "I10", "A00-A09"
  name_ua: string;
  name_en: string | null;
  parent_code: string | null; // self-FK: A00.0 → A00 → A00-A09
  level: 'class' | 'group' | 'category' | 'detail';
}

export interface LabTestType {
  id: number;
  name: string;
  category: 'Lab' | 'Instrumental';
  code_nk_026: string | null; // НК 026:2021 procedure code
}

// ============================================================
// User data
// ============================================================

export interface User {
  id: string;
  email: string;
  google_id: string | null;
  created_at: string;
}

export interface Profile {
  id: string;
  user_id: string;
  display_name: string;
  birth_date: string | null;
  is_owner: boolean;
```

```
  created_at: string;
}

// ============================================================
// Visits
// ============================================================

export interface Visit {
  id: string;
  profile_id: string;
  date: string;
  prescription: string | null;
  notes: string | null;
  updated_at: string;
  // joined
  specialties?: MedicalSpecialty[];
  diagnoses?: VisitDiagnosis[];
  files?: VisitFile[];
}

export interface VisitSpecialty {
  id: string;
  visit_id: string;
  specialty_id: number;
  // joined
  specialty?: MedicalSpecialty;
}

export interface VisitDiagnosis {
  id: string;
  visit_id: string;
  icd_code: string;
  type: 'Main' | 'Concomitant';
  // joined
  icd?: IcdCode;
}

export interface VisitFile {
  id: string;
  visit_id: string;
  user_id: string;
  file_name: string;
  mime_type: string;
  size_bytes: number;
  storage_path: string;
  uploaded_at: string;
}

// ============================================================
// Lab results
// ============================================================

export interface LabResult {
```

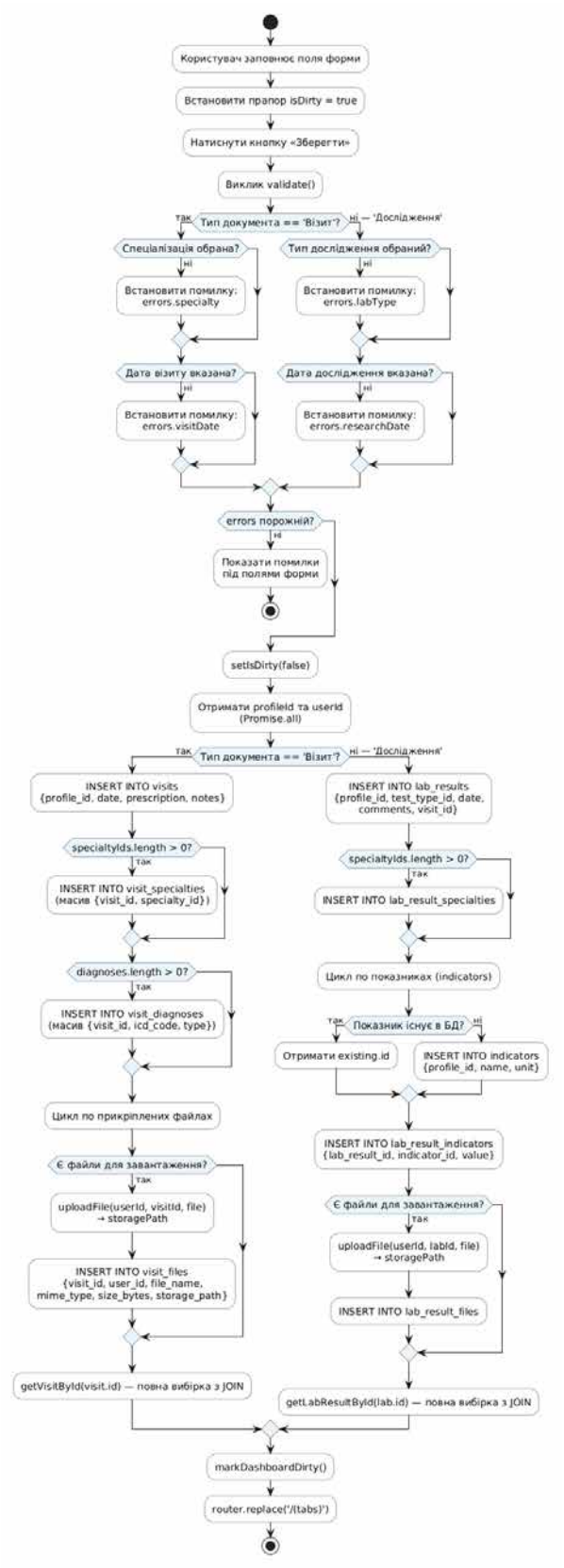
```
  id: string;
  profile_id: string;
  test_type_id: number;
  visit_id: string | null;
  date: string;
  comments: string | null;
  updated_at: string;
  // joined
  test_type?: LabTestType;
  specialties?: MedicalSpecialty[];
  indicators?: LabResultIndicator[];
  diagnoses?: LabResultDiagnosis[];
  files?: LabResultFile[];
}

export interface LabResultSpecialty {
  id: string;
  lab_result_id: string;
  specialty_id: number;
  // joined
  specialty?: MedicalSpecialty;
}

export interface LabResultDiagnosis {
  id: string;
  lab_result_id: string;
  icd_code: string;
  type: 'Main' | 'Concomitant';
  // joined
  icd?: IcdCode;
}

export interface LabResultFile {
  id: string;
  lab_result_id: string;
  user_id: string;
  file_name: string;
  mime_type: string;
  size_bytes: number;
  storage_path: string;
  uploaded_at: string;
}

export interface Indicator {
  id: string;
  profile_id: string;
  name: string;
  unit: string | null;
}

export interface LabResultIndicator {
  id: string;
  lab_result_id: string;
```

```
  indicator_id: string;
  value: string;
  // joined
  indicator?: Indicator;
}

// ============================================================
// Supabase Database shape (for typed client)
// ============================================================

// Column-only types used for Insert/Update (no joined relations)
interface VisitColumns {
  id: string; profile_id: string; date: string;
  prescription: string | null; notes: string | null; updated_at: string;
}
interface LabResultColumns {
  id: string; profile_id: string; test_type_id: number; visit_id: string | null;
  date: string; comments: string | null; updated_at: string;
}
interface VisitSpecialtyColumns { id: string; visit_id: string; specialty_id: number; }
interface VisitDiagnosisColumns { id: string; visit_id: string; icd_code: string; type: 'Main' | 'Concomitant'; }
interface LabResultSpecialtyColumns { id: string; lab_result_id: string; specialty_id: number; }
interface LabResultDiagnosisColumns { id: string; lab_result_id: string; icd_code: string; type: 'Main' | 'Concomitant'; }
interface LabResultIndicatorColumns { id: string; lab_result_id: string; indicator_id: string; value: string; }

export interface Database {
  public: {
    Views: Record<string, never>;
    Functions: Record<string, never>;
    Enums: Record<string, never>;
    CompositeTypes: Record<string, never>;
    Tables: {
      users: {
        Row: User;
        Insert: Omit<User, 'id' | 'created_at'>;
        Update: Partial<Omit<User, 'id' | 'created_at'>>;
      };
      profiles: {
        Row: Profile;
        Insert: Omit<Profile, 'id' | 'created_at'>;
        Update: Partial<Pick<Profile, 'display_name' | 'birth_date'>>;
      };
      medical_specialties: {
        Row: MedicalSpecialty;
        Insert: Omit<MedicalSpecialty, 'id'>;
        Update: Partial<Omit<MedicalSpecialty, 'id'>>;
      };
      icd_codes: {
```

```
      Row: IcdCode;
      Insert: IcdCode;
      Update: Partial<IcdCode>;
    };
    lab_test_types: {
      Row: LabTestType;
      Insert: Omit<LabTestType, 'id'>;
      Update: Partial<Omit<LabTestType, 'id'>>;
    };
    visits: {
      Row: VisitColumns;
      Insert: Omit<VisitColumns, 'id' | 'updated_at'>;
      Update: Partial<Omit<VisitColumns, 'id' | 'updated_at'>>;
    };
    visit_specialties: {
      Row: VisitSpecialtyColumns;
      Insert: Omit<VisitSpecialtyColumns, 'id'>;
      Update: Partial<Omit<VisitSpecialtyColumns, 'id'>>;
    };
    visit_diagnoses: {
      Row: VisitDiagnosisColumns;
      Insert: Omit<VisitDiagnosisColumns, 'id'>;
      Update: Partial<Omit<VisitDiagnosisColumns, 'id'>>;
    };
    visit_files: {
      Row: VisitFile;
      Insert: Omit<VisitFile, 'id' | 'uploaded_at'>;
      Update: Partial<Omit<VisitFile, 'id' | 'uploaded_at'>>;
    };
    lab_results: {
      Row: LabResultColumns;
      Insert: Omit<LabResultColumns, 'id' | 'updated_at'>;
      Update: Partial<Omit<LabResultColumns, 'id' | 'updated_at'>>;
    };
    lab_result_specialties: {
      Row: LabResultSpecialtyColumns;
      Insert: Omit<LabResultSpecialtyColumns, 'id'>;
      Update: Partial<Omit<LabResultSpecialtyColumns, 'id'>>;
    };
    lab_result_diagnoses: {
      Row: LabResultDiagnosisColumns;
      Insert: Omit<LabResultDiagnosisColumns, 'id'>;
      Update: Partial<Omit<LabResultDiagnosisColumns, 'id'>>;
    };
    lab_result_files: {
      Row: LabResultFile;
      Insert: Omit<LabResultFile, 'id' | 'uploaded_at'>;
      Update: Partial<Omit<LabResultFile, 'id' | 'uploaded_at'>>;
    };
    indicators: {
      Row: Indicator;
      Insert: Omit<Indicator, 'id'>;
      Update: Partial<Omit<Indicator, 'id'>>;
```

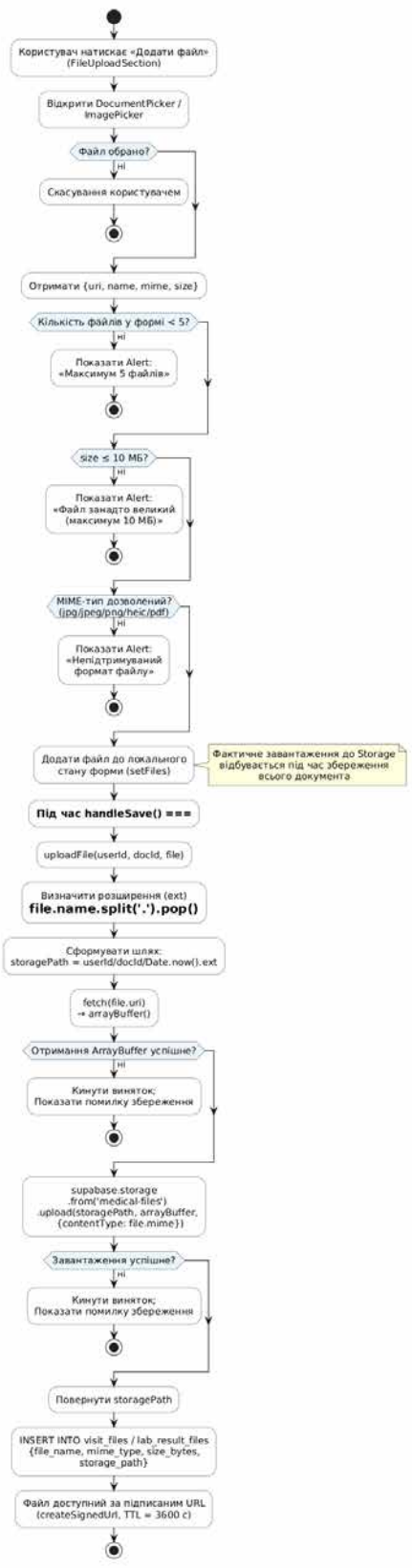
```
      };
      lab_result_indicators: {
        Row: LabResultIndicatorColumns;
        Insert: Omit<LabResultIndicatorColumns, 'id'>;
        Update: Partial<Omit<LabResultIndicatorColumns, 'id'>>;
      };
    };
  };
}
```

Додаток Ґ



Алгоритм збереження медичного документа (createVisit / createLabResult)

Додаток Д



Алгоритм завантаження медичного файлу до Supabase Storage

Додаток Е

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Рубан Анатолій Олександрович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмне забезпечення системи обліку особистого здоров'я» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
    - [ ] інструменти генеративного ШІ не використовувалися.
    - [ ] інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, _____інші (вказати назву)) були використані для:
        - [ ] перекладу іншомовних джерел;
        - [ ] стилістичного редагування та перевірки граматики;
        - [ ] допомоги у написанні/відлагодженні програмного коду;
        - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___» _____ 2026 р. _____ **Анатолій РУБАН**
(підпис)