An-Najah National University

Faculty of Engineering & Information Technology

Department of Computer Engineering

Health Tracker

Prepared By

Arkan Nezam Fares

Mohammed Najeh Abdullrazeq

Supervised By

Dr.Haya Samaana

Dr.Sufyan Samara

Presented in partial fulfillment of the requirements for a Bachelor's degree in (Name of Specialization).

Jan 27, 2026

Disclaimer

The information and views presented in this report are those of the authors and do not necessarily reflect the official opinion of An-Najah National University, the Department of Computer Engineering, or any affiliated institutions. Every effort has been made to ensure the accuracy of the data and the information presented, but the authors do not assume responsibility for any errors or omissions that may occur.

Table of Contents

Table of Figure

List of Abbreviations

Abbreviation	Meaning
AI	Artificial Intelligence
API	Application Programming Interface
AWS	Amazon Web Services
EHR	Electronic Health Record
FHIR	Fast Healthcare Interoperability Resources
GDPR	General Data Protection Regulation
HIPAA	Health Insurance Portability and Accountability Act
HL7	Health Level Seven
JWT	JSON Web Token
LLM	Large Language Model
OCR	Optical Character Recognition
OTP	One-Time Password
PDF	Portable Document Format
REST	Representational State Transfer
S3	Simple Storage Service
UI/UX	User Interface / User Experience

Acknowledgement

We would like to begin by expressing our deepest gratitude to our families for their unwavering support, encouragement, and love throughout the journey of this project. To our parents, whose sacrifices and guidance have shaped us into who we are today, we are forever grateful.

We also extend our heartfelt thanks to our professors and academic advisors who have accompanied us throughout this journey, helping bring this project to life. A special thanks goes to **Dr. Haya Samaana and Dr. Sufyan Samara** for their continuous guidance, patience, and valuable insights, which have been instrumental in the development and success of our project.

We cannot forget our friends, whose constant support and uplifting messages kept us going through the toughest days. Their encouragement and friendship have played a vital role in our motivation and perseverance.

Lastly, this work is dedicated to the resilient people of Gaza. We proudly pay tribute to the families' enduring hardship and to the martyrs who gave their lives for a noble cause. The strength and willpower of the people of Gaza continue to inspire and motivate us every day.

Abstract

Health Tracker is a cross-platform mobile healthcare management system designed to help users store and understand their medical information, track medications, and receive AI-assisted explanations of laboratory reports and symptoms. The system consists of a React Native (Expo) mobile application and a Laravel REST API backend integrated with cloud services and multiple AI providers. Lab reports can be uploaded as PDFs or images; the backend stores the files in AWS S3, prepares AI prompts, and retrieves structured medical findings in JSON format. The mobile application presents the extracted results in a patient-friendly form and provides additional features such as medication scheduling with reminders, health profile management, and an AI Doctor consultation module that supports text and optional image inputs. The project emphasizes security, modularity, and extensibility through token-based authentication, clean service abstractions, and clear separation between client, API, data, and AI processing layers. The implemented system demonstrates a practical architecture for integrating modern AI services into a healthcare-oriented application while preserving maintainability and future upgrade paths.

Chapter 1: Introduction

1.1 General Background

Digital healthcare applications have expanded rapidly due to increased availability of smartphones, cloud services, and remote care models. However, many patients still struggle to interpret laboratory results, maintain consistent medication adherence, and keep a coherent record of their health data. Recent advances in multimodal AI (models that process text and images) enable automated extraction of findings from documents such as lab reports and prescriptions, which can improve accessibility and reduce misunderstanding.

1.2 Problem Statement

Patients often receive laboratory reports in PDF or printed form with medical terminology and reference ranges that are difficult to understand without clinical training. In parallel, medication schedules are frequently managed manually, leading to missed doses and incomplete logs. Existing applications may provide reminders, but do not integrate document understanding, structured extraction, and consultation in a unified workflow.

1.3 Objectives (Purpose or Aims)

- Design and implement a cross-platform mobile application for managing health profiles, lab reports, and medications.
- Build a secure backend API that supports authentication, storage, and structured persistence of medical data.
- Implement an AI-driven lab-report analysis pipeline that converts user uploads into structured JSON findings.
- Provide an AI Doctor consultation module supporting text and optional image input with provider selection (cloud or local).
- Ensure the system is modular and extensible for future standards-based interoperability (e.g., FHIR) and new AI models.

1.4 Significance of the Work

Health Tracker reduces friction between patients and their health information by transforming raw medical documents into understandable summaries and structured values. It also supports medication adherence through scheduling, reminders, and logging, which are critical for chronic conditions. From a software engineering perspective, the project demonstrates a production-oriented integration of cloud storage, secure APIs, and multiple AI providers within a maintainable architecture.

1.5 Organization of the Report

Chapter 1 introduces the project motivation, objectives, and significance. Chapter 2 reviews theoretical background and related work in mobile health systems, AI document understanding, and interoperability. Chapter 3 describes the methodology and implementation approach, including architecture, standards, and constraints. Chapter 4 presents the developed features and system results. Chapter 5 discusses the outcomes, limitations, and implications. Chapter 6 concludes the report and provides recommendations and future work. References and appendices follow at the end.

Chapter 2: Theoretical Background and Previous Work

2.1 Mobile Health (mHealth) Applications

mHealth applications commonly provide features such as appointment reminders, symptom tracking, medication schedules, and basic analytics. Key design challenges include usability, privacy, accuracy of health information, and reliability of notifications across platforms.

2.2 AI for Medical Document Understanding

Traditional extraction from PDFs relied on OCR and template-based parsing, which is sensitive to layout variations. Modern multimodal models can infer structure, extract tables, and summarize findings from scanned documents. A practical system must still enforce guardrails: structured output formats, validation, and clear disclosure that AI advice is not a clinical diagnosis.

2.3 LLM-Based Medical Consultation (Assistive, Not Diagnostic)

LLM-based chat assistants can provide general explanations, help users ask better questions, and summarize information. In healthcare contexts, the system must avoid presenting outputs as definitive diagnoses. Instead, it should encourage users to consult clinicians for urgent or uncertain cases, and it should support safe prompt design and logging for auditability.

2.4 Interoperability and Standards (HL7/FHIR)

Healthcare systems often exchange data using HL7 and FHIR resources. While Health Tracker stores data in its own database schema, the architecture is designed to enable future conversion of summaries and lab values into FHIR-compatible structures to support integration with hospitals and clinics.

2.5 Previous Work and Gap

Many existing applications provide either medication reminders or health record storage. Fewer systems combine document upload, structured extraction, and an AI consultation flow that supports multiple providers (cloud and local) within a single product. Health Tracker addresses this gap using a layered architecture and explicit AI processing pipelines.

Chapter 3: Methodology

3.1 Development Approach

The project followed an iterative development approach: requirements were captured as user stories, core modules were implemented incrementally, and each sprint focused on a complete user flow (e.g., onboarding, medication tracking, lab report upload). Integration and refinement were performed continuously to maintain a working end-to-end system.

3.2 System Architecture Overview

Health Tracker is organized into four main layers: (1) client layer (React Native mobile app), (2) API layer (Laravel REST backend), (3) data layer (MySQL and caching/queues), and (4) cloud and external services (AWS S3/EC2/RDS, AI providers, and Google Maps). This separation improves maintainability, security boundaries, and allows AI providers to be swapped with minimal impact on the user experience.

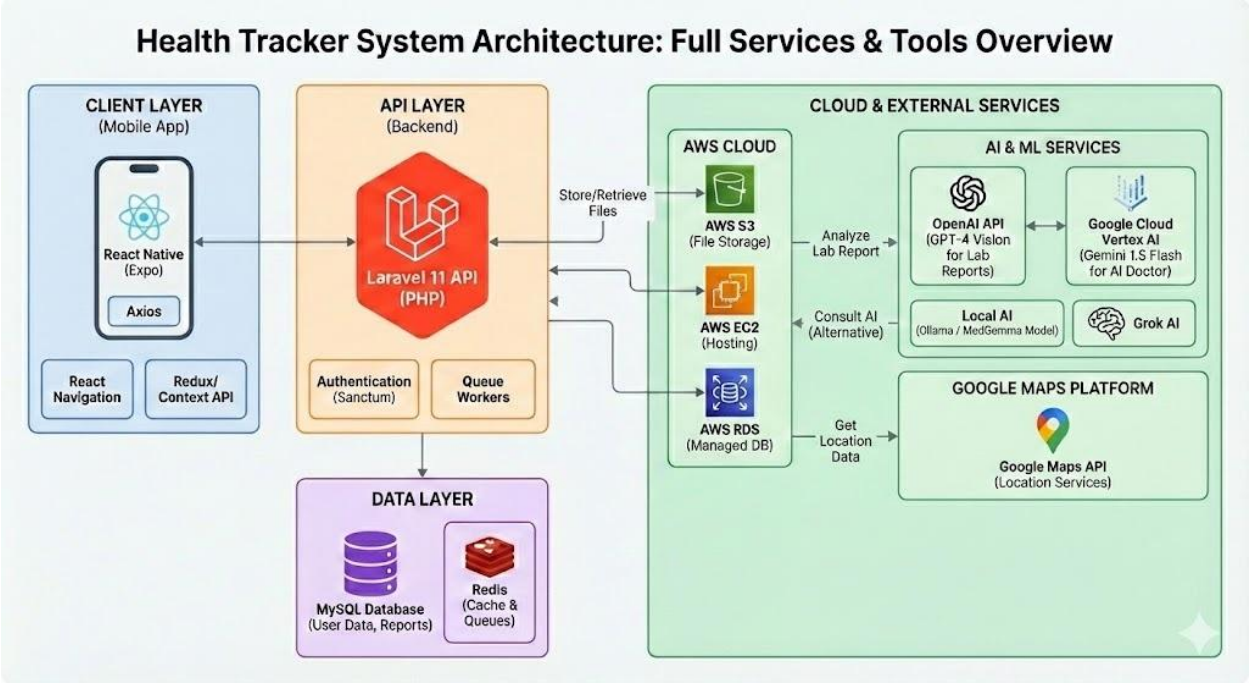

Figure 1 Health Tracker system architecture overview (client, API, data, cloud, and AI services).

3.3 Frontend Implementation (React Native + Expo)

The mobile application was implemented using React Native with Expo, enabling deployment on Android, iOS, and web from a single codebase. Routing is managed using Expo Router (file-based routing), and state is handled using React Context for the onboarding flow, AsyncStorage for persistence, and Expo Secure Store for secure token storage.

Key frontend characteristics derived from the implementation analysis include:

- 25+ screens organized into authentication and tab-based flows.
- 18 reusable UI components for consistent presentation of lab values, medications, and reports.
- 14 service modules to encapsulate API calls and business logic.
- Multi-language support (English, Arabic, Spanish) using i18next.
- Notifications using Expo Notifications for medication reminders and alerts.

3.4 Backend Implementation (Laravel REST API)

The backend is a RESTful API built with Laravel (PHP 8.2+). It provides authentication via Laravel Sanctum tokens, user profile management, lab report and medication endpoints, and integrations with email services for OTP verification and alerts. The design uses controllers for API boundaries, Eloquent models for persistence, and dedicated service classes for AI integrations.

Key backend characteristics include:

- 10 controllers covering authentication, lab reports, medications, profiles, hospital services, and AI chat modules.
- 11 models representing users, profiles, reports, medications, chat sessions, and related entities.
- 50+ API endpoints grouped by feature.
- AWS S3 integration for secure file storage of reports and images.
- PDF-to-image conversion (spatie/pdf-to-image) to support AI vision processing.

3.5 AI Processing Pipelines

AI is integrated as a separate processing layer accessed through service interfaces. This approach centralizes prompt construction, output validation, and error handling. Two primary AI pipelines are implemented: (1) lab report analysis (file-based, structured extraction) and (2) AI Doctor consultation (conversational, text with optional image).

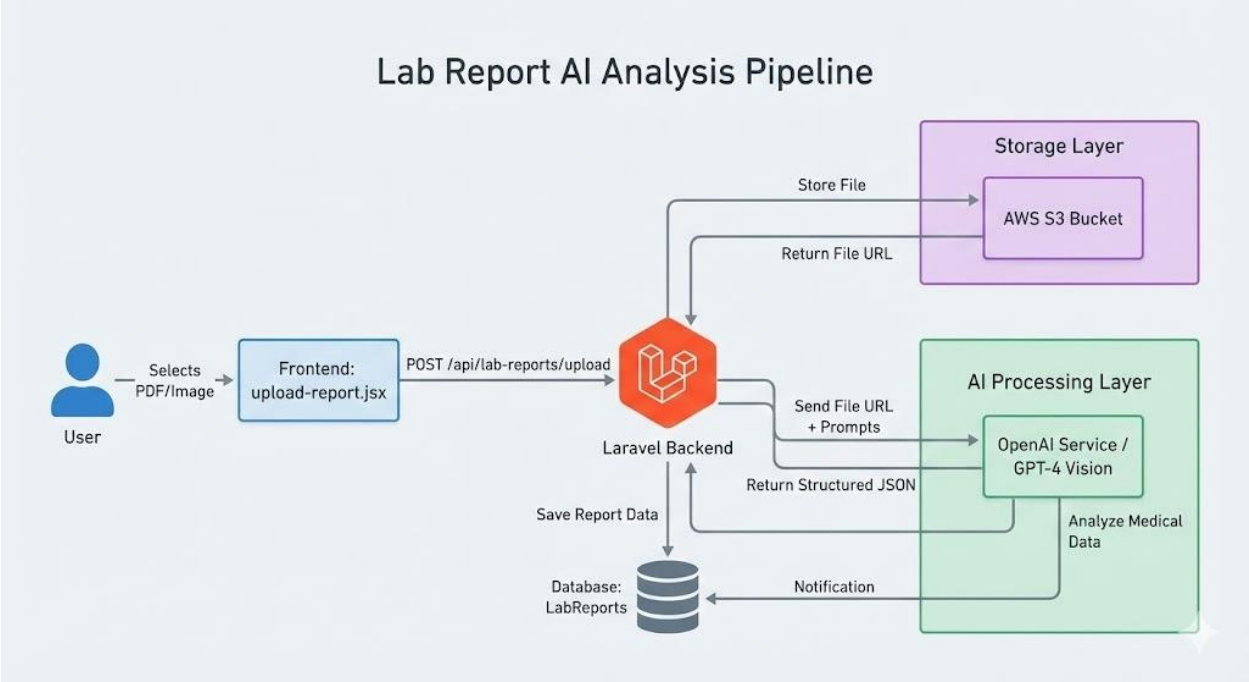

Figure 2 Lab report AI analysis pipeline (upload, storage in S3, AI analysis, structured JSON, and persistence).

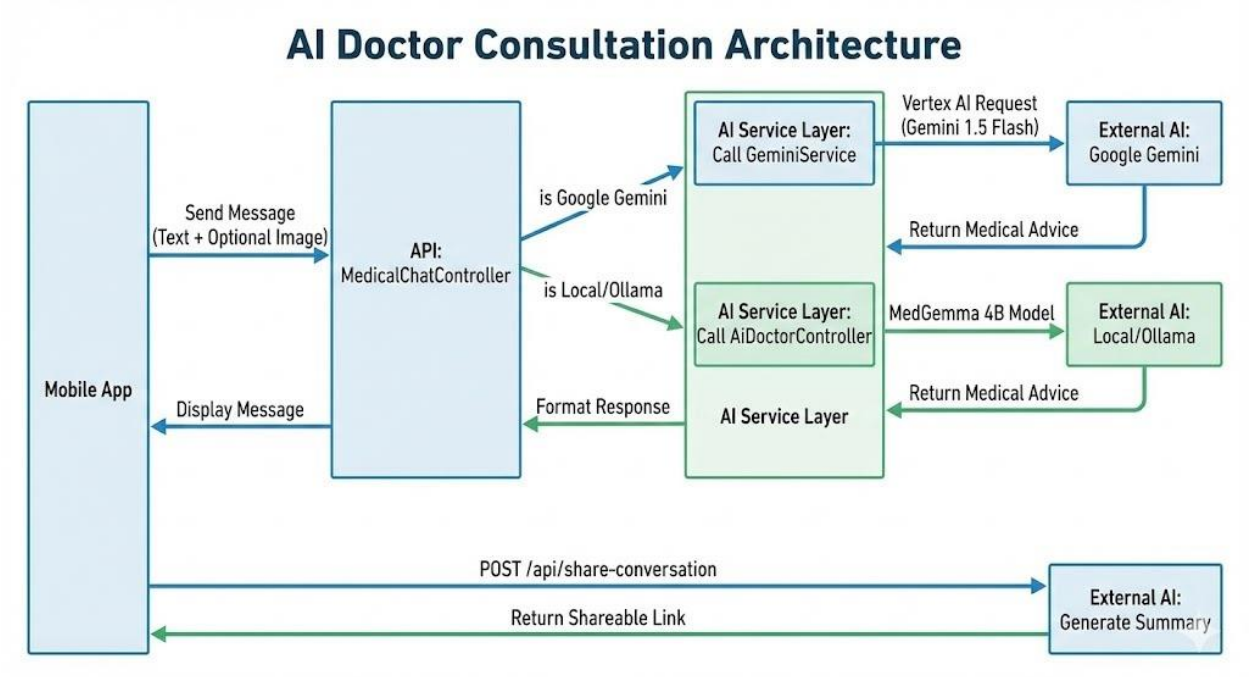

Figure 3 AI Doctor consultation architecture with selectable provider (Gemini or local/Ollama).

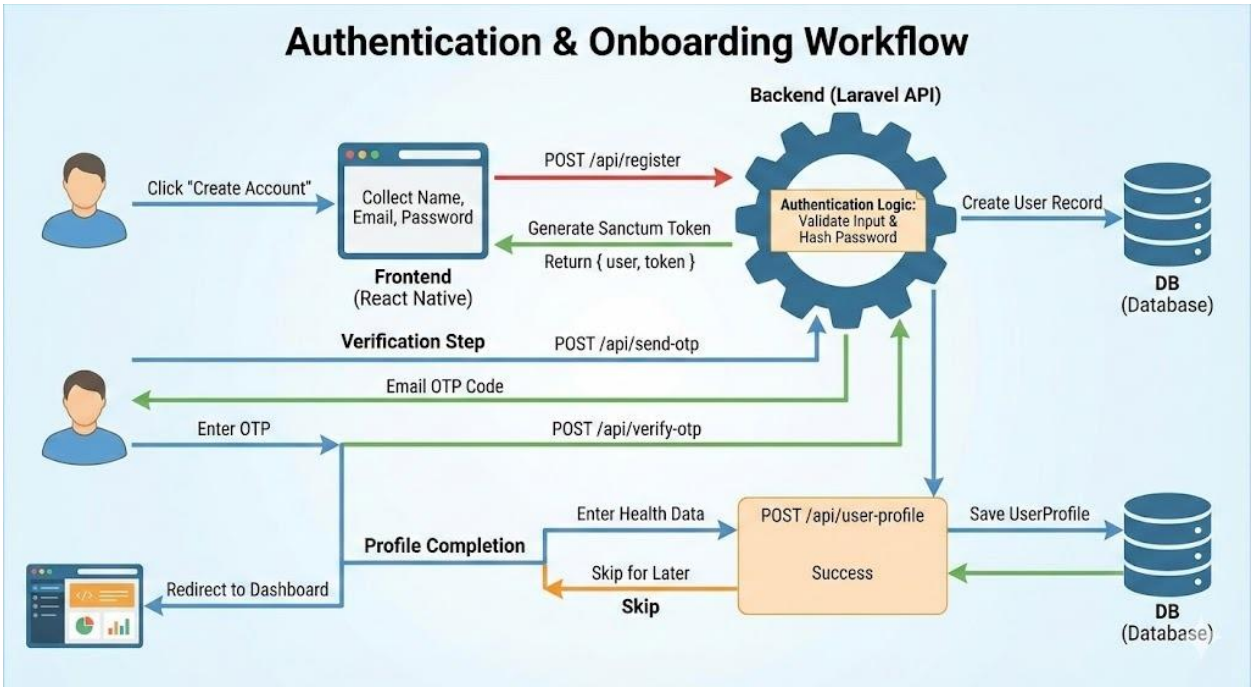

Figure 4 Authentication and onboarding workflow (registration, OTP verification, profile completion).

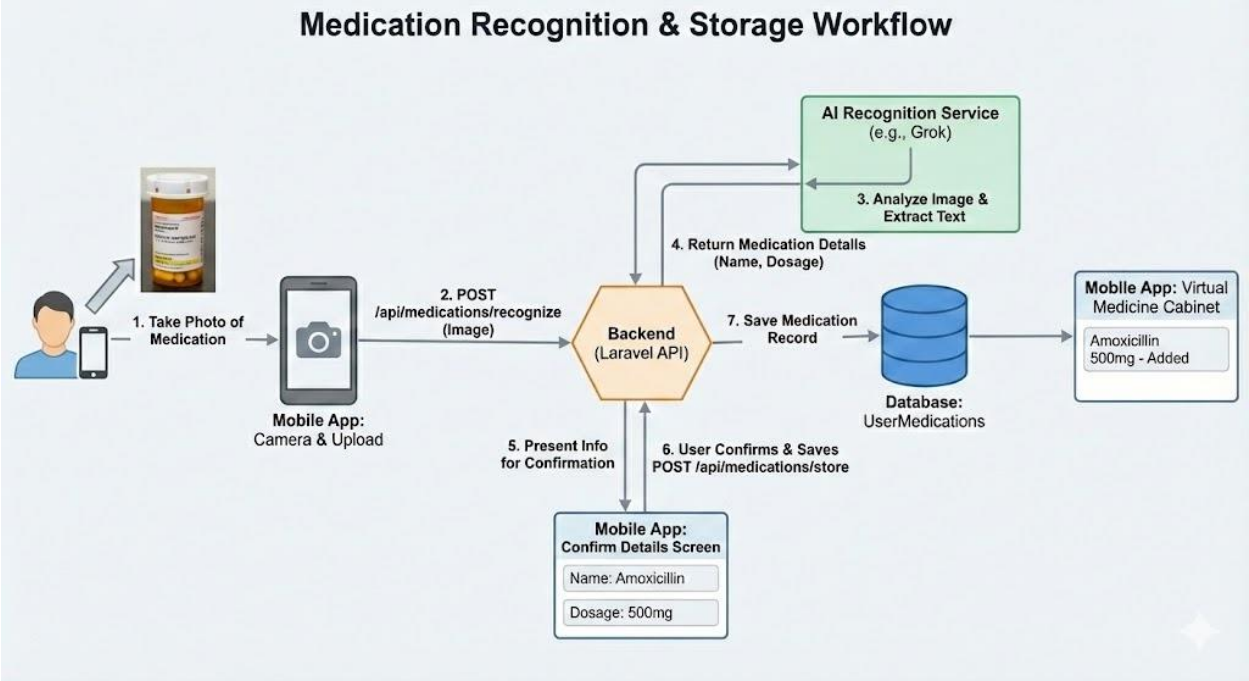

Figure 5 Medication recognition and storage workflow (image analysis and user confirmation).

3.6 Data Model and Persistence

The database schema is designed around user-centric ownership of data: each user has a health profile, multiple lab reports, medications, medication logs, and conversation sessions. Eloquent ORM migrations define tables and relationships, allowing the system to evolve safely over time.

Table 1 summarizes key data entities managed by the backend.

Entity	Purpose	Examples of Fields
User / UserProfile	Identity and health profile data	name, age, gender, bloodType, allergies, conditions
LabReport	Store report file metadata and AI results	fileUrl, uploadedAt, extractedJson, summary
Medication / MedicationLog	Track medications and adherence logs	name, dosage, scheduleTimes, taken/missed
ConversationSession / ConversationReport	Persist AI chat and generated summaries	messages, provider, createdAt, shareLink

Table 1: Core data entities in Health Tracker backend.

3.7 Security and Privacy Controls

Security is addressed through token-based authentication (Sanctum), server-side validation, access control checks on user-owned resources, and secure storage of uploaded documents in S3. Sensitive client tokens are stored using secure storage on the device. AI prompts are designed to minimize unnecessary personal identifiers and focus on clinical values and context.

3.8 Standards and Specifications (Codes)

The project aligns with widely accepted software and healthcare-adjacent standards and best practices. While Health Tracker is an academic project and not a certified medical device, its design adopts standards to improve safety, security, and interoperability:

- RESTful API design conventions and HTTP status code semantics for predictable client-server interaction.
- OWASP Top 10 secure development guidance (input validation, authentication, access control, and secure storage).

- HL7/FHIR concepts as a target interoperability format for future export of structured medical data.
- OAuth-style token usage patterns (via Sanctum) and secure mobile token storage practices.
- Cloud storage security practices: signed URLs, least-privilege access policies, and separation of public and private assets.

3.9 Constraints

The design and implementation were developed under practical constraints typical of real-world systems. Key constraints and their handling are summarized below:

- Economy:
  - Use of cross-platform development (React Native + Expo) to reduce time and cost compared to separate native apps.
  - Modular AI provider integration to allow selecting cost-effective models depending on usage and budget.
- Environment:
  - Reduction of device-side computation by delegating heavy AI processing to backend/cloud services when appropriate.
  - Optional local AI (Ollama) to support environments with limited internet connectivity.
- Society:
  - Accessibility features through multilingual support (English, Arabic, Spanish).
  - Clear separation between informational guidance and clinical diagnosis, encouraging professional consultation.
- Politics:
  - Data sovereignty considerations: architecture supports hosting components locally or in selected regions.
  - Design avoids discriminatory profiling and focuses on user-provided health data and consent.
- Ethics:
  - Disclosure that AI outputs are assistive and may contain errors.
  - Respect for privacy through minimal data exposure and secure storage.
- Health and Safety:

    - Avoiding hard diagnostic claims; recommending clinical follow-up for alarming values or symptoms.
    - Auditability: storing analysis outputs and timestamps to support review.
- Manufacturability:
    - Use of mature frameworks (Laravel, React Native) and standard cloud services to simplify deployment and maintenance.
    - Service-layer abstractions to make future replacement of AI vendors feasible.
- Sustainability:
    - Clean modular architecture that supports future upgrades (new AI models, FHIR export, additional analytics).
    - Database migrations and structured storage to enable long-term maintenance.

3.10 Doctor-Patient Connection Design Methodology

The Doctor-Patient Connection feature was developed using an iterative methodology that combines requirements analysis, data modeling, API contract definition, and user interface prototyping. The first iteration focused on defining the relationship lifecycle and the minimum safe access rules. Subsequent iterations added advanced discovery filters, notifications, and deeper integration with clinical workflows (chat, prescriptions, appointments, and visit history).

The design process followed an 'authorization-first' principle: database schema and backend enforcement were defined before building patient and doctor user interfaces. This ensured that privacy and consent rules could not be bypassed by UI mistakes. Once backend guards were stable, frontend development focused on usability and clarity of consent.

Key methodological steps included: (1) defining roles and trust boundaries, (2) modeling the relationship state machine and permission levels, (3) designing endpoints and request/response JSON contracts, (4) implementing middleware and validation rules, (5) implementing doctor and patient workflows, and (6) validating end-to-end scenarios with test accounts.

Trade-offs were considered explicitly. For messaging and badge updates, a REST plus polling strategy was selected for simplicity and ease of deployment. For future iterations, migration to WebSockets is planned to reduce network overhead and improve real-time responsiveness.

3.11 Verification and Testing Methodology

Verification combined manual end-to-end testing, API-level validation, and negative testing for authorization. Manual testing used separate doctor and patient accounts to ensure that each role sees the correct interface and cannot access restricted endpoints. API validation included testing of request validation errors (invalid permission values, duplicate relationships), correct state transitions, and correct denial of access when relationships are not active.

Chapter 4: Results and Analysis

4.1 Implemented Features

- Authentication and onboarding with OTP email verification and profile completion.
- Health profile management (demographics, allergies, conditions, emergency contacts).
- Medication tracking: add/edit medications, schedules, reminders, and adherence logs.
- Lab report upload (PDF/image), storage, and AI-powered structured analysis.

- AI Doctor chat with selectable provider (Gemini cloud or local/Ollama) and optional image input.
- Hospital and pharmacy discovery using Google Maps/Places (location-based services).
- Conversation sharing and report generation for consultations.
- Doctor-patient connection module: public doctor discovery, connection requests, permission-based data sharing, secure messaging, and care workflows (appointments, prescriptions, and visits).

4.2 Frontend Delivery Results (Implementation Evidence)

Based on the frontend implementation analysis, the application includes 25+ screens, 18 reusable UI components, and 14 service modules. This indicates that the app is not a prototype page but a complete multi-flow application with clear separation between UI components and service logic.

4.3 Backend Delivery Results (Implementation Evidence)

The backend implementation provides 50+ endpoints across 10 controllers and persists data in 20+ database tables through migrations. Dedicated service classes encapsulate integrations with OpenAI, Gemini, Grok, and local/Ollama models. This modular structure supports maintainability and upgrades.

4.4 Data Validation and Output Structure

AI outputs are returned to the backend as structured JSON. The backend stores both the raw model response (for traceability) and normalized fields required by the mobile UI. The system validates expected keys and handles failure cases by returning user-friendly error messages and allowing re-try without losing the uploaded file reference.

4.5 Doctor-Patient Connection Feature (Doctor Portal and Patient Portal)

This section documents the design and implementation of the Doctor-Patient Connection feature, which establishes a secure, auditable relationship between patients and healthcare providers. The module enables doctor discovery, connection requests, permission-based data sharing, and clinical workflows such as messaging, prescriptions, appointments, and visit history review. The implementation spans the backend (Laravel REST API, database schema, middleware, and notifications) and the frontend (doctor portal pages and the patient connection page).

4.5.1 Objectives and Scope

The Doctor-Patient Connection feature was introduced to solve a common problem in digital healthcare applications: enabling a patient to share health information with a specific doctor while maintaining privacy, consent, and accountability. Rather than exposing patient data broadly, the system enforces an explicit relationship lifecycle

(request -> accept/decline -> active -> revoke) and stores the exact permission level granted to each doctor.

Scope of this module includes: (1) public doctor discovery and search, (2) connection request creation and management, (3) enforcement of role-based access control and per-relationship permissions, (4) user interface flows for doctors and patients, (5) integration with messaging, prescriptions, appointments, and visits.

4.5.2 User Roles and Responsibility Boundaries

The module defines clear responsibility boundaries between three main actors. Patients initiate connections and control the level of data sharing. Doctors accept/decline requests and provide care through the platform. Administrators may verify doctor profiles to improve trust in search results and reduce impersonation risk.

Role	Primary Capabilities	Key Constraints / Protections
Patient	Search doctors, send/cancel requests, manage connected doctors, chat, view prescriptions, book appointments	Cannot access doctor-only endpoints; can only share data after relationship is active; can revoke access
Doctor	Manage profile, review requests, access connected patient data (within permissions), prescribe, schedule, chat, record visits	Must have active relationship to access patient data; permissions restrict visibility; actions audited via timestamps
Admin	Verify doctor profiles, review compliance and reported issues, manage system-wide policies	No access to patient content unless explicitly authorized by policy; verification actions stored (verified_at/by)

Table 4.1: Role boundaries and access constraints for the Doctor-Patient Connection module.

4.5.3 Relationship Lifecycle and State Machine

Connections are modeled as first-class entities in the database using a relationship table. Each relationship is uniquely identified by a (doctor_id, patient_id) pair and transitions through a controlled set of states. This approach prevents ambiguous access rules and enables accurate auditing of when access was granted or revoked.

Status	Meaning	Allowed Actions (Examples)
Pending	A request exists and is awaiting a decision from the target user (doctor or patient)	Accept/Decline; requester may cancel

Active	Connection is approved and data sharing is allowed within the stored permission level	View patient data (doctor); chat; prescribe; book appointments; update permissions
Declined	Target user rejected the request; no data sharing occurs	Requester may send a new request (subject to rules)
Revoked	An active relationship was terminated by one side; access must be removed immediately	No patient data access; optionally allow re-request with new consent

Table 4.2: Relationship state machine used to manage doctor-patient access.

In addition to status, the relationship stores 'requested_by' to record who initiated the connection, and timestamps such as connected_at and revoked_at to support audit trails. Business rules prevent duplicate relationships via a unique constraint, and validation ensures that the doctor_id is always a doctor account and patient_id is always a patient account.

4.5.4 Permission Model and Data Access Control

The system uses a three-level permission model to align with real-world consent patterns. Patients decide how much information a doctor can access at the time of connection, and permissions can be adjusted later. Permissions are stored per relationship, meaning a patient may grant different levels to different doctors.

Data Category	full_access	limited_access	emergency_only
Basic profile & contacts	Yes	Yes	Emergency contacts only
Lab reports & AI analysis history	All reports	Recent / relevant subset	Critical highlights only
Medications & prescriptions	All medications + history	Current medications	Active medications only
Visits, notes, and long-term history	Full history	Restricted	Not available

Table 4.3: High-level permission matrix (data visibility per relationship).

Permission enforcement is implemented in two layers. First, the frontend adapts the user interface and hides actions that are not allowed for a given permission level to reduce user confusion. Second, the backend enforces permissions as the source of truth using middleware and per-endpoint checks, preventing bypass attempts even if a malicious client modifies requests.

4.5.5 Database Design

The feature relies on two primary tables: doctor_profiles and doctor_patient_relationships. The users table is extended with a role field to differentiate

doctors from patients. Doctor profiles contain professional metadata (license number, clinic details, specializations, verification status). Relationships store consent and connection state.

Table	Key Fields	Design Rationale
doctor_profiles	user_id, license_number (unique), specializations (JSON), city/area, works_online, verification_status	Separates professional data from generic user fields; supports public search and verification workflows
doctor_patient_relationships	doctor_id, patient_id, status, permissions, requested_by, connection_notes, connected_at, revoked_at	Captures consent and lifecycle; unique constraint prevents duplicates; timestamps support auditing
users (extended)	role, basic identity fields	Role-based routing and authorization; simplifies guards on endpoints

Table 4.4: Database tables supporting doctor discovery and doctor-patient relationships.

Indexing was added to optimize frequent queries (doctor_id, patient_id, status). For example, the doctor portal frequently lists pending requests and connected patients, while the patient portal frequently loads connected doctors and pending requests. The unique constraint on (doctor_id, patient_id) ensures each pair has at most one relationship record.

4.5.6 Backend Architecture and Implementation

The backend is implemented as a Laravel REST API and follows an API-first approach: controllers validate input, invoke service-layer logic, and return consistent JSON responses. The module is structured around two controllers: DoctorController (profile and public search) and DoctorConnectionController (relationship lifecycle).

DoctorController provides endpoints for doctor registration, profile completion, profile updates, and public doctor discovery. The discovery endpoint supports advanced filtering by specialization, city/area, clinic name, experience, fee range, language, online availability, and verification status. To improve usability in the target region, city matching supports bilingual naming variations (Arabic/English) and performs case-insensitive matching.

DoctorConnectionController manages the full relationship lifecycle. Core validations include: (1) confirming user roles (doctor vs patient), (2) preventing duplicate requests, (3) enforcing valid state transitions (e.g., only pending can be accepted), and (4) enforcing permission values (full_access, limited_access, emergency_only). When a

request is created, the backend emits a notification record for the target user to ensure it appears in the doctor portal notification center and badge counts.

A dedicated middleware (e.g., EnsureDoctorPatientConnection) is applied to endpoints that expose patient data to doctors. This middleware verifies that an active relationship exists between the authenticated doctor and the requested patient. Where required, an additional permission check is performed to ensure the doctor is entitled to view the requested resource category (e.g., full history vs emergency-only information).

4.5.7 API Specification

API endpoints are grouped into four functional sets: (1) doctor profile management and search, (2) connection management, (3) messaging between doctors and patients, and (4) protected patient data access. All authenticated endpoints use token-based authentication (Laravel Sanctum). The tables below summarize the main contracts.

Method	Endpoint	Auth	Purpose
POST	/api/doctors/register	No	Register doctor account
POST	/api/doctors/complete-profile	Yes	Create/complete doctor profile
GET	/api/doctors/profile	Yes	Fetch doctor self profile
PUT	/api/doctors/profile	Yes	Update profile (may reset verification)
GET	/api/doctors/search	No	Public doctor search with filters
GET	/api/doctors/{id}/profile	No	Public doctor profile view

Table 4.5: Doctor profile and public search endpoints.

Method	Endpoint	Auth	Purpose
POST	/api/doctor-connections/request	Yes	Send connection request
GET	/api/doctor-connections/pending	Yes	List pending requests (role-based)
POST	/api/doctor-connections/{id}/accept	Yes	Accept request
POST	/api/doctor-connections/{id}/decline	Yes	Decline request
GET	/api/doctor-connections/patients	Yes	Doctor: list connected patients
GET	/api/doctor-	Yes	Patient: list

	connections/doctors		connected doctors
DELETE	/api/doctor-connections/{id}	Yes	Revoke active connection
PUT	/api/doctor-connections/{id}/permissions	Yes	Doctor: update permissions
GET	/api/doctor-connections/search-doctors	Yes	Patient: search doctors

Table 4.6: Connection management endpoints.

Messaging endpoints enable doctor-patient communication. In the current implementation, near real-time behavior is achieved using REST endpoints combined with client-side polling (periodic refresh) for new messages and typing status. This design is simpler to deploy than WebSockets but can be upgraded in future work.

Method	Endpoint	Auth	Purpose
POST	/api/doctor-messages/send	Yes	Send a message
GET	/api/doctor-messages/conversation/{userId}	Yes	Load conversation thread
GET	/api/doctor-messages/conversations	Yes	List conversations
POST	/api/doctor-messages/conversation/{userId}/read-all	Yes	Mark conversation as read
POST	/api/doctor-messages/typing	Yes	Set typing status
GET	/api/doctor-messages/typing/{userId}	Yes	Get typing status
POST	/api/doctor-messages/upload-attachment	Yes	Upload attachment for chat

Table 4.7: Messaging endpoints used for doctor-patient chat.

4.5.8 Frontend Implementation - Doctor Portal

The doctor-facing interface is implemented as a set of dedicated pages (15 pages in total) optimized for clinical workflows: reviewing requests, searching patient lists, viewing patient data, writing prescriptions, managing appointments and availability, and communicating with patients. A consistent layout (header, sidebar, content panels) is used to reduce navigation friction. Badge counts for pending requests and unread messages are refreshed periodically to provide timely awareness without requiring manual reload.

The frontend interacts with the backend through service modules (e.g., doctorService, doctorConnectionService, doctorMessageService). These services encapsulate HTTP calls, handle token injection, standardize error handling, and return normalized data structures to the UI layer.

4.5.8.1 Doctor Home (doctor-home.jsx)

Doctor Home serves as the clinical dashboard. It presents an overview of the doctor's operational workload using compact metric cards such as total connected patients, upcoming appointments, pending prescriptions, and unread messages. The page supports quick actions that route to frequent tasks (open AI assistant, view schedule, view patient list). Badge counters are updated through periodic polling (for example, a 12-second interval) to keep alerts current.

4.5.8.2 Doctor Patients (doctor-patients.jsx)

Doctor Patients is the core relationship management page. A two-tab layout separates Connected relationships from Pending requests. For connected patients, the doctor can search by name or contact information, view connection metadata (date and permission badge), and execute shortcuts such as viewing patient details, starting a chat, or creating a prescription. For pending requests, the doctor can inspect the patient's message/notes and then accept or decline the request.

4.5.8.3 Doctor Profile (doctor-profile.jsx)

Doctor Profile displays the doctor's professional information used by patients when searching. This includes license number and verification status, specializations, clinic information, languages, consultation fee, office hours, and location metadata (city/area and optional coordinates). A verification status badge is displayed prominently to communicate trust level to both the doctor and patients.

4.5.8.4 Doctor Profile Edit (doctor-profile-edit.jsx)

Doctor Profile Edit provides a structured form for updating professional details. Input validation ensures required fields are present and that unique fields (e.g., license number) do not conflict. A key safeguard is the verification reset rule: when a sensitive professional identifier such as license number changes, the profile verification status can be reset to pending to trigger re-validation.

4.5.8.5 Doctor Messages and Chat (doctor-messages.jsx, doctor-chat.jsx)

Messaging is split into a conversation list view and a per-patient chat view. The list view shows active conversations, search, and unread counters. The chat view supports text messages, attachments, read receipts, and typing indicators. In the current architecture, the client periodically loads the conversation and typing status using REST endpoints, producing a real-time-like experience without a persistent socket connection.

4.5.8.6 Doctor Appointments (doctor-appointments.jsx)

Doctor Appointments manages booking records and supports operational filtering by status (pending, confirmed, cancelled, completed). Each appointment includes patient identity, date/time, and reason for visit, with actions to update status. The page also links back to patient details to reduce context switching for the doctor.

4.5.8.7 Doctor Availability (doctor-availability.jsx)

Doctor Availability allows configuration of the weekly schedule, slot duration, and advance booking rules. The UI supports recurring weekly patterns while also allowing the doctor to block individual times or specific dates (holidays). This module is essential for appointment reliability because it ensures patients only request slots that are actually available.

4.5.8.8 Doctor Prescriptions and Create Prescription (doctor-prescriptions.jsx, doctor-create-prescription.jsx)

Prescription pages provide both an overview list and a creation workflow. Doctors can filter prescriptions by status or patient, view medication instructions, and mark prescriptions as completed. The creation page supports adding multiple medications with dosage, frequency, duration, and special instructions. After saving, the patient receives a notification that a new prescription is available.

4.5.8.9 Doctor Patient Detail (doctor-patient-detail.jsx)

Doctor Patient Detail consolidates patient data into a unified clinical view. It shows patient demographics, a health summary, lab reports and AI analysis, medications, visit history, and conversation shortcuts. The page is permission-aware: it adjusts which panels are visible and which actions are enabled according to the relationship's permission level. This reduces accidental privacy violations and keeps the UI consistent with backend enforcement.

4.5.8.10 Doctor Visits and Visit History (doctor-visit.jsx, doctor-visit-history.jsx)

Visit recording supports structured clinical notes, diagnosis entries, and treatment plans. If audio capture is enabled, a visit can include an audio recording or upload, followed by transcription and storage in the visit record. The visit history page provides a timeline view with filtering by patient and date range to support longitudinal follow-up.

4.5.8.11 Doctor Notifications (doctor-notifications.jsx)

Doctor Notifications provides a centralized feed for connection requests, new messages, appointment reminders, and prescription updates. The UI supports filtering and marking notifications as read. Badge counters on the navigation elements are derived from unread notifications to highlight urgent actions.

4.5.8.12 Doctor MedGemma (doctor-medgemma.jsx)

MedGemma integration provides an AI assistant for doctors. It offers a chat-like interface where doctors can ask clinical questions or generate structured summaries. To keep the system safe, the output is positioned as decision support rather than definitive diagnosis, and clinicians remain responsible for final judgment.

Across all doctor pages, common UX concerns were addressed: consistent empty states when lists are empty, clear error messages on failed requests, loading indicators during API calls, and responsive layouts for different screen sizes.

4.5.9 Frontend Implementation - Patient Connection Page (connect-to-doctor.jsx)

On the patient side, the Connect to Doctor page consolidates discovery and relationship management into a single workspace. A multi-tab design separates Search, Connected, Pending, and Prescriptions views. This reduces navigation complexity for patients and mirrors the mental model of 'finding a doctor, requesting access, then working with connected doctors'.

Doctor discovery supports advanced filtering across multiple criteria: specialization, location (city/area), online consultation availability, minimum years of experience, maximum consultation fee, languages spoken, and optional verification-only filtering. Search input is debounced (for example, 500 ms) to reduce unnecessary API calls, and pagination is used to handle large result sets efficiently.

Search results are rendered as doctor cards. Each card summarizes identity and credibility (name, verification badge), service capabilities (specializations, clinic and location, years of experience, fee), and availability indicators (works online). The primary action button changes based on state: Connect for new relationships, Pending for requests awaiting approval (with cancel option), or Chat/Book Appointment for active connections.

When the patient initiates a connection, a modal form collects optional notes and requires explicit selection of a permission level. This design choice makes consent explicit and reduces misunderstanding. Upon submission, a notification is created for the doctor, and the request appears in both the patient's Pending tab and the doctor's Pending tab.

The interface includes user-experience features designed for the target audience: right-to-left layout support for Arabic, bilingual city naming support, and clear empty-state guidance when no results or connections are present.

UI evidence for the Connect Doctor page (doctor listing, profile preview, and filtering by city/specialization) is provided in Appendix A (Batch 8).

4.5.10 User Flow Walkthroughs

This subsection describes the main user flows implemented for the feature, emphasizing state transitions, security checks, and user interface behavior.

4.5.10.1 Patient: Discover Doctor and Send Connection Request

1) The patient opens the Connect to Doctor page and uses search and filters to locate a suitable doctor. 2) The patient reviews the doctor card or opens the detailed profile view for more information. 3) The patient selects Connect, writes an optional message, and

selects a permission level. 4) The backend validates roles, checks for existing relationships, creates a relationship record in pending status, and emits a notification to the doctor. 5) The patient sees the request listed under Pending with an option to cancel.

4.5.10.2 Doctor: Review and Accept/Decline Request

1) The doctor receives a notification and sees an updated pending badge count. 2) In Doctor Patients (Pending tab), the doctor reviews patient details and connection notes. 3) If accepted, the relationship transitions to active, connected_at is recorded, and both parties can use chat and clinical features. 4) If declined, status becomes declined and no patient data is shared.

4.5.10.3 Active Connection: Data Access and Care Actions

Once active, doctors can open the patient detail page and access permitted data categories. The system checks relationship existence and permission level before exposing sensitive resources. Doctors can initiate chat, create prescriptions, and manage appointments directly from patient context to reduce workflow friction.

4.5.10.4 Revocation and Permission Updates

Patients may revoke access (disconnect) at any time, causing the relationship to move to revoked and immediately preventing further access. Additionally, doctors (or patients, depending on policy) may update permission level to adjust data visibility. All changes are stored in the relationship record to maintain an audit trail.

4.5.11 Security, Privacy, and Compliance Considerations

Healthcare systems must treat patient data as highly sensitive. This module applies multiple safeguards: strict role-based access control, explicit consent through a relationship record, server-side enforcement through middleware, and minimization of exposed data based on permissions. Token-based authentication ensures only logged-in users can perform protected operations. Input validation prevents privilege escalation attempts such as sending a request while spoofing requested_by.

The design also supports accountability: connected_at and revoked_at timestamps can be used to establish when access existed. Where required, the system can be extended with audit logging for access to individual resources (lab report views, downloads, note edits).

4.5.12 Performance and Reliability

Several performance optimizations were implemented to keep the experience responsive. On the patient search side, debounced input and pagination reduce server load and network usage. On the doctor side, patient list pages use server-side pagination to handle large patient panels. Database indexing on doctor_id, patient_id, and status accelerates common queries for pending requests and active connections.

For messaging and badge counters, polling is used to approximate real-time updates. Polling provides simplicity and compatibility across environments, but increases network calls. Future enhancements can migrate high-frequency updates (chat, typing) to WebSockets while keeping low-frequency updates (appointments) on REST.

4.5.13 Testing and Validation

Testing focused on correctness of state transitions, authorization enforcement, and user experience continuity. Core scenarios were validated end-to-end using real devices and test accounts for both roles (doctor and patient).

Representative test scenarios include:

- Patient can search doctors using specialization and location filters and receives paginated results.
- Patient can send a connection request with a selected permission level; doctor receives a pending request and notification.
- Doctor can accept a pending request; relationship becomes active; patient appears in Connected list with correct permission badge.
- Doctor cannot access patient data without an active relationship (middleware denies access).
- Doctor access to lab reports and medications changes according to permission level (full vs limited vs emergency).
- Patient can cancel a pending request; request disappears from both sides and no data is shared.
- Either side can revoke an active connection; subsequent access attempts fail immediately.
- Chat sends and receives messages; read status updates; typing indicator updates using polling.

4.5.14 Limitations and Future Enhancements

Current limitations include reliance on polling for chat updates and the absence of a comprehensive audit trail for individual data access events. Additionally, permission categories can be refined into more granular scopes (e.g., allow medications but not lab reports). Future work can add: WebSocket-based messaging, push notifications, configurable consent forms, fine-grained scopes, and FHIR export for sharing structured medical data with other systems.

4.6 UI Screenshots and Captions

The following screenshots document the Health Tracker mobile UI and illustrate the main user flows. They cover dashboard navigation, report upload and management, AI-driven report analysis, detailed lab/vital interpretations, visual analytics, and in-context assistant support (MedGemma).

This appendix-style section documents key patient-facing mobile UI screens used in the Health Tracker system. The screenshots demonstrate how the user navigates the home dashboard, uploads and reviews lab reports, and explores the AI-generated analysis tabs. All screens support Arabic RTL layout and are designed around quick access to critical actions (e.g., emergency information and nearest hospital lookup) while keeping clinical content organized into modular views.

4.6.1 Patient Home Dashboard & Navigation

The home dashboard acts as the entry point for daily health monitoring. It surfaces critical shortcuts (emergency access and hospital discovery), highlights personal health indicators such as BMI, and provides quick actions to upload new reports and review recent results. A side navigation drawer supports fast switching between major modules (reports, medications, AI doctor, emergency tools) while keeping the experience consistent.

Figure 6 Patient Home Dashboard

This dashboard acts as the patient's "health landing page". It summarizes key information at a glance (greeting header, quick health snapshot, and shortcuts) while keeping critical actions one tap away. The Emergency shortcut is intentionally prominent to reduce time-to-action during urgent situations. The "Nearest Hospital" card demonstrates location-based service discovery, enabling users to quickly find nearby care facilities and access key contact details without navigating multiple screens.

Figure 7 BMI and body metrics card displaying BMI value, category indicator, and height/weight summary

The BMI card demonstrates how personal profile data (height and weight) is translated into an immediately understandable health metric. The UI shows the calculated BMI value, a category label (e.g., underweight/normal/overweight), and a compact visual indicator to help users interpret the result without medical jargon. This module is designed to support longitudinal tracking: once multiple readings exist, the same component can visualize change over time and highlight notable trends.

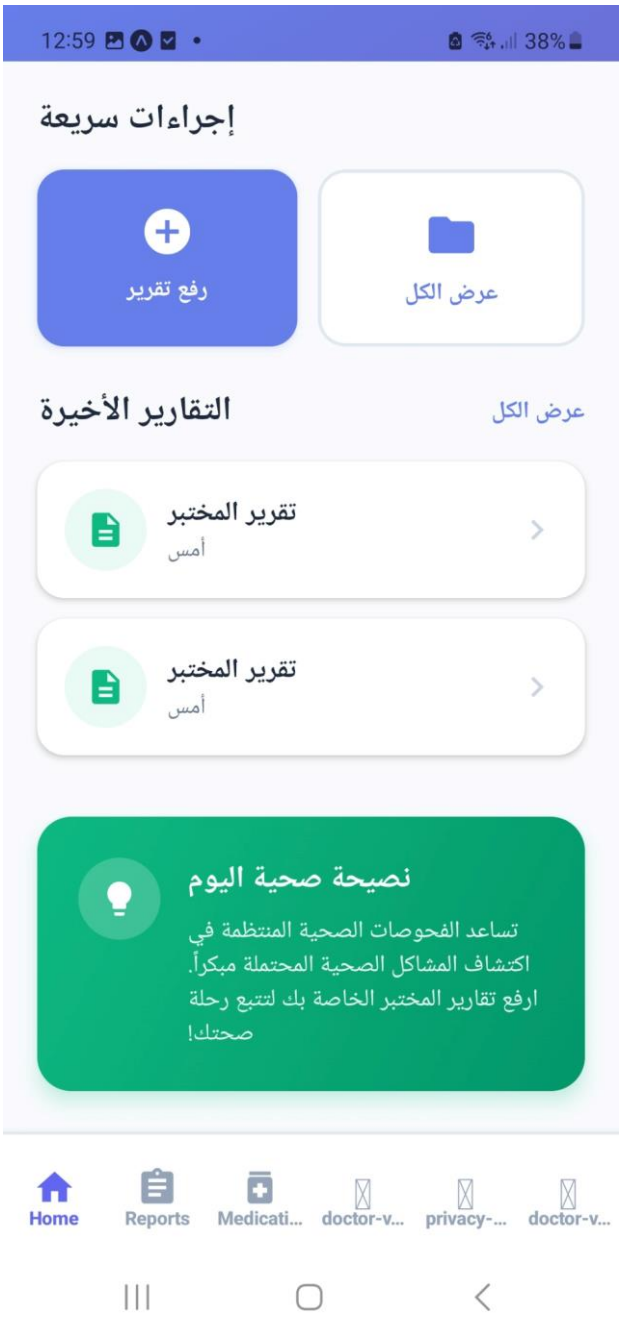

Figure 8 Quick actions and recent reports area

The Reports home section focuses on speed: users can either upload a new report immediately or review previously analyzed reports. “Upload Report” launches the import workflow (camera capture or file selection), while “View All” opens the full reports dashboard with filtering and management options. Recent report cards display status and key metadata so users can open a report and jump directly into the AI analysis tabs (original file, summary, overview, lab values, vitals, and charts).

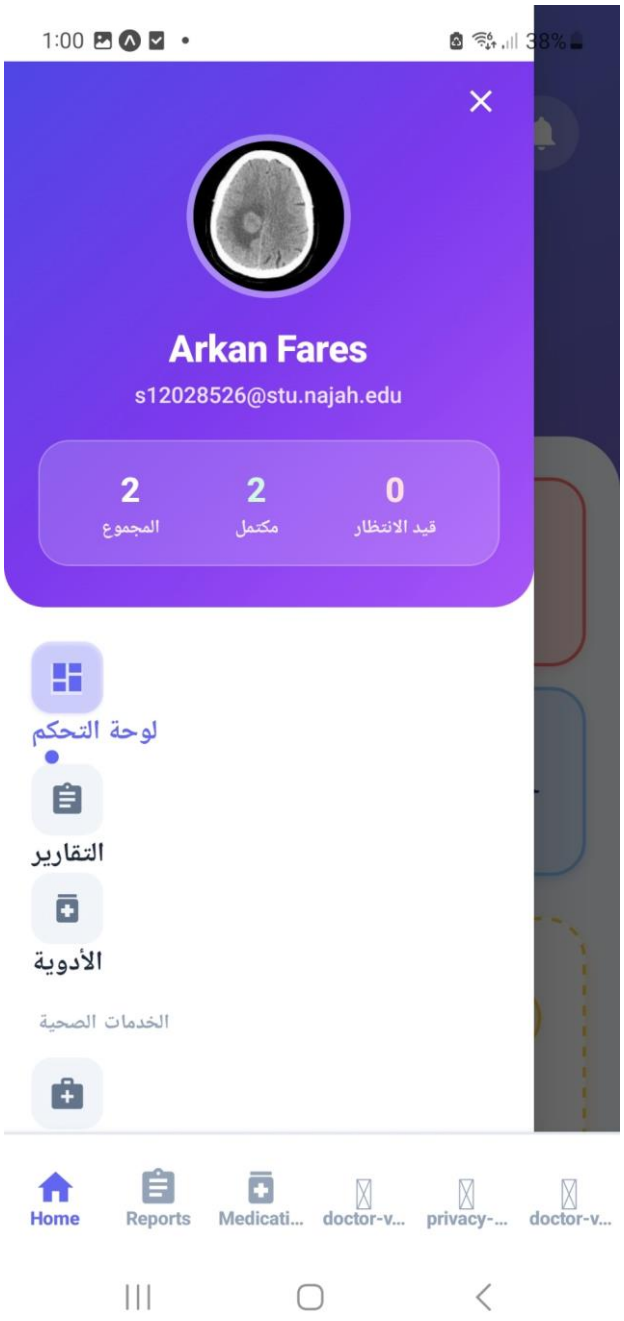

Figure 9 Navigation drawer with user identity, report counters (total/completed/pending), and primary navigation links (dashboard, reports, medications).

The navigation drawer provides consistent, app-wide access to major modules. A compact profile header gives identity context, and the report counters (total/completed/pending) act as a lightweight progress indicator for the user's medical history. Centralizing navigation reduces deep menu paths, improves usability on small screens, and keeps workflows predictable across sessions.

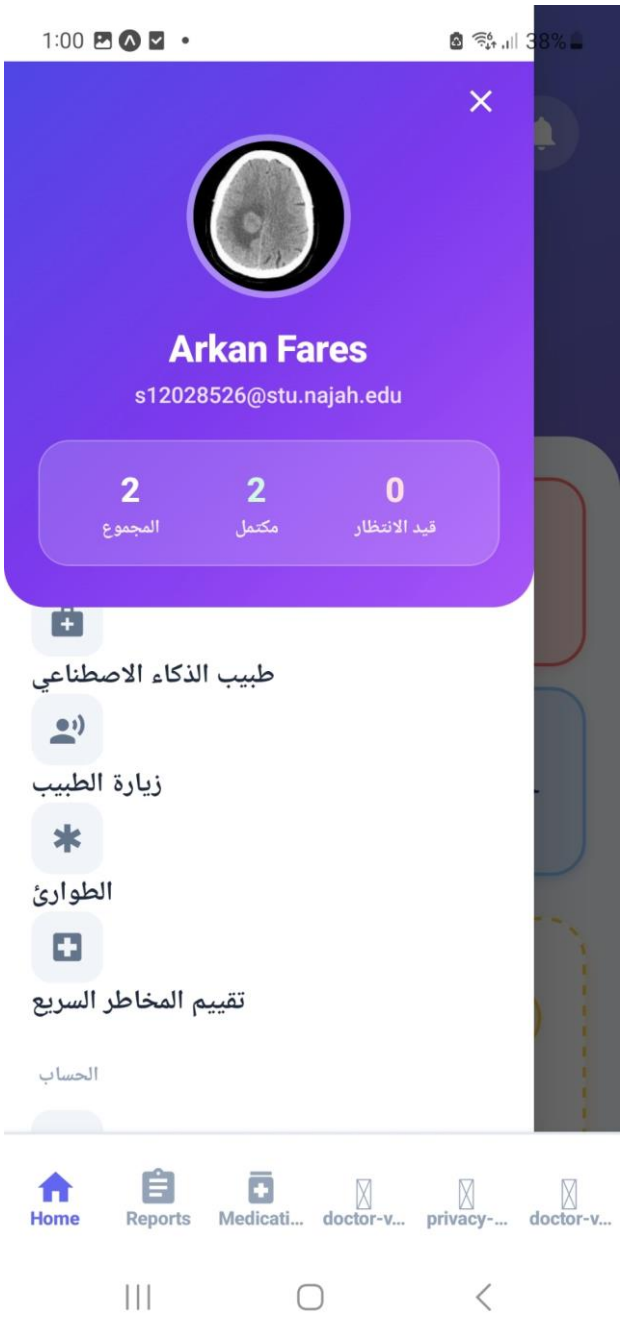

Figure 10 Expanded navigation drawer highlighting access to AI doctor assistant, emergency services, and rapid risk assessment features

This expanded drawer view highlights the platform's core feature set beyond basic report storage. Users can quickly access the AI doctor assistant, doctor visit tools, emergency services, and rapid risk assessment modules from a single navigation surface. Grouping these actions improves discoverability and encourages feature adoption while maintaining clear separation between clinical support features and administrative settings.

4.6.2 Lab Reports Dashboard & AI Report Analysis Tabs

The Lab Reports module provides an overview of uploaded reports and their processing status, then offers a detailed analysis view organized into tabs. The analysis interface separates raw source material (the original PDF) from AI-generated summaries and statistics, helping users validate the source while still benefiting from automation. This structure also enables future expansion (e.g., charting trends over time, highlighting critical values, and sharing outputs).

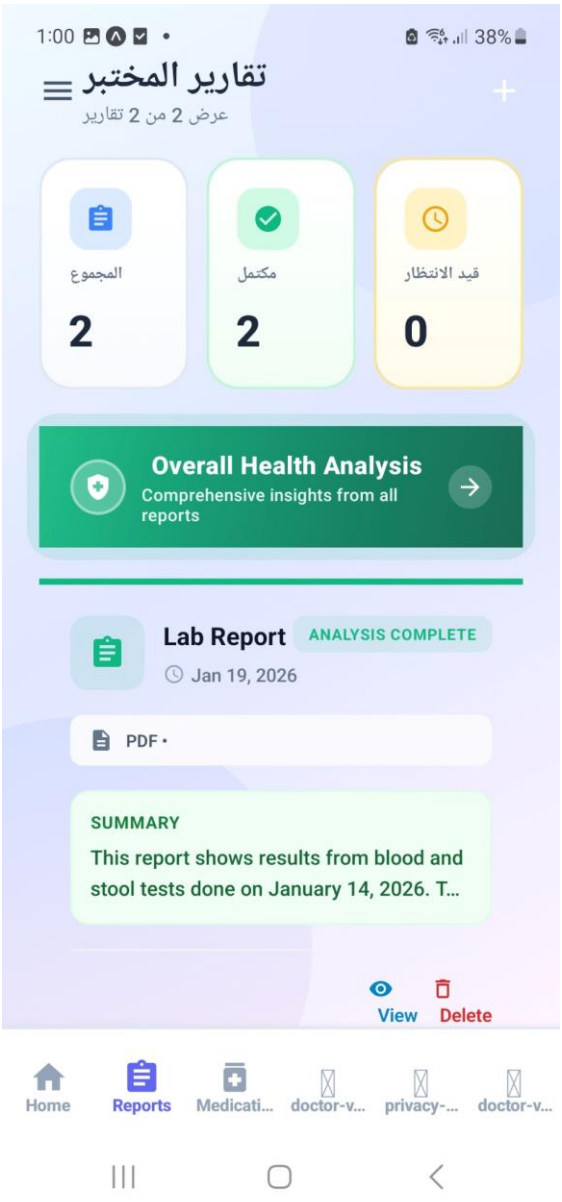

Figure 11 dashboard with status counters and an entry

The Lab Reports dashboard presents a structured overview of all stored reports and their processing status. Top-level counters provide immediate feedback on how many reports

exist and whether any are awaiting analysis, which is essential when uploads and AI processing occur asynchronously. Each report card includes metadata (date, file type) and a short summary, along with direct actions (open/view and delete) to support the full Report lifecycle.

Figure 12 showing uploaded PDF metadata and user actions (download/share).

The “Original File” tab preserves the source-of-truth document. It displays file metadata (type and size) and embeds a viewer that loads the uploaded PDF inside the app. Download/share actions support interoperability—users can forward the original report to physicians or external systems without losing context. The loading state (“جاري التحميل...”) is important for mobile UX, clearly communicating that the viewer is fetching content and preventing confusion during slower network conditions.

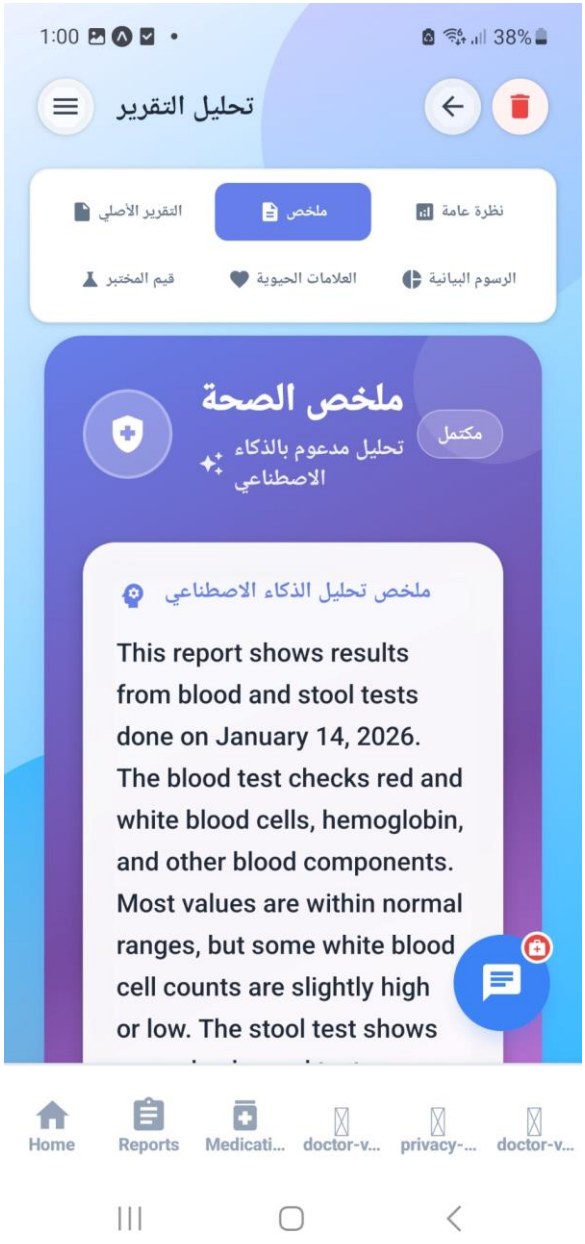

Figure 13 (AI Summary tab) presenting a plain-language summary generated by the AI pipeline.

The AI Summary tab converts raw laboratory content into plain-language explanations. The goal is to reduce cognitive load by presenting key findings first, followed by interpretation and suggested follow-up topics. This is particularly useful for users unfamiliar with reference ranges or medical abbreviations. In the system design, the summary is generated after extraction/OCR and cached so it can be reopened instantly without re-processing the report.

Figure 14 (Overview tab) presenting high-level statistics

The Overview tab provides a triage layer by aggregating extracted results into counts of normal, abnormal, and high-priority findings. This allows users to understand the overall “shape” of a report before drilling down into individual parameters. High-priority findings are surfaced explicitly to guide attention to potentially important results, while the app still avoids making definitive diagnoses and encourages professional review when needed.

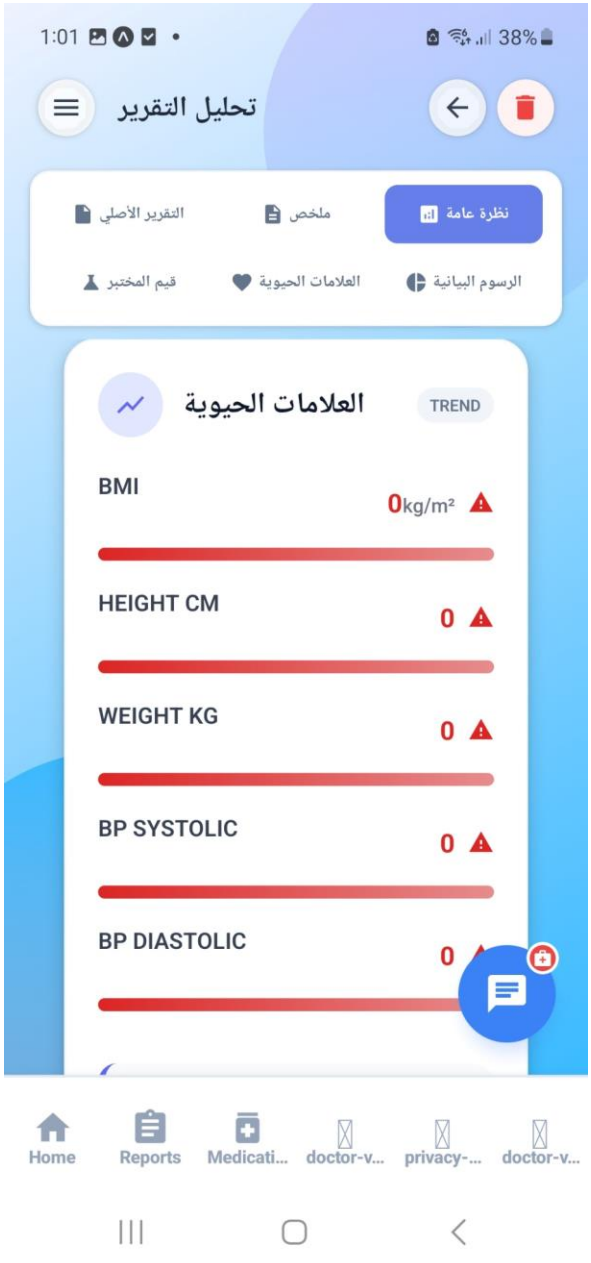

Figure 15 (Vital Signs tab) listing extracted/recorded vitals (BMI, height, weight, blood pressure) with alert indicators for missing or out-of-range values.

The Vital Signs tab consolidates measurements such as BMI, height, weight, and (when available) blood pressure and pulse. These values can come from the user profile, manual entry, or automated extraction—so the UI includes alert indicators when fields are missing or outside expected ranges. This design also functions as a data-quality checkpoint: users can quickly spot incomplete records, improving downstream analytics and trend visualization.

Additional UI screenshots for the doctor profile and verification view, settings (notifications and bottom navigation customization), the notifications feed, the MedGemma clinical assistant, and the administrator dashboard/user management screens are provided in Appendix B under “UI Screenshots – Batch 5” (Figures 60–69).

4.6.3 Additional Report Analysis Screens (Batch 2)

This batch expands the report analysis experience by showing deeper sections of the overview, report metadata validation, lab-value cards (normal and abnormal), vital sign details, result-distribution charts, the embedded MedGemma assistant, the report upload workflow, and the settings/personalization screens.

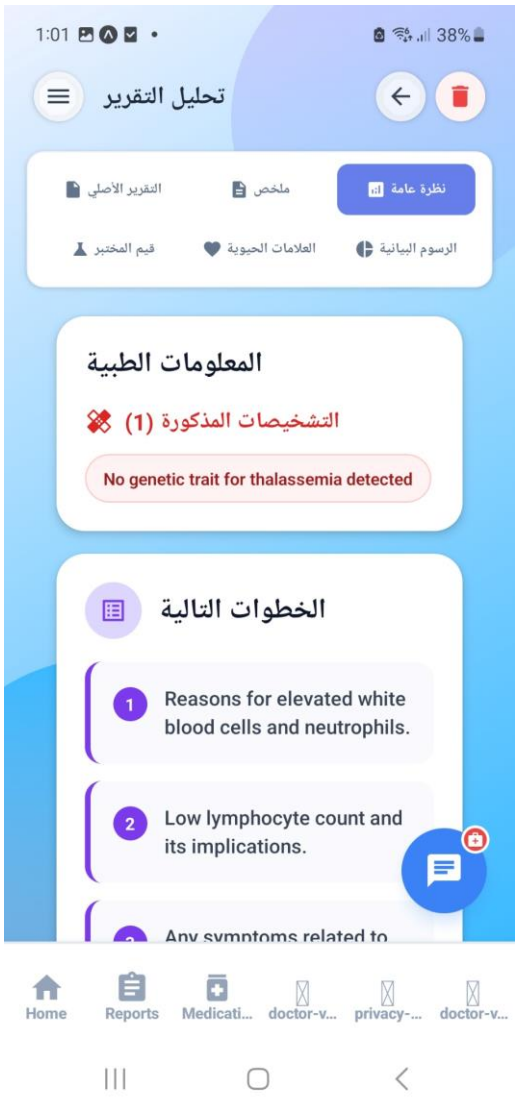

Figure 16 Report analysis – medical information and recommended next steps.

This screen is part of the Overview tab and focuses on actionable interpretation rather than raw numbers. It summarizes detected medical notes/diagnoses (e.g., a negative finding such as “no genetic trait for thalassemia detected”) and then lists “Next Steps” as follow-up prompts. These prompts are designed to help the user ask the right questions during a doctor visit, improving health literacy without replacing professional diagnosis.

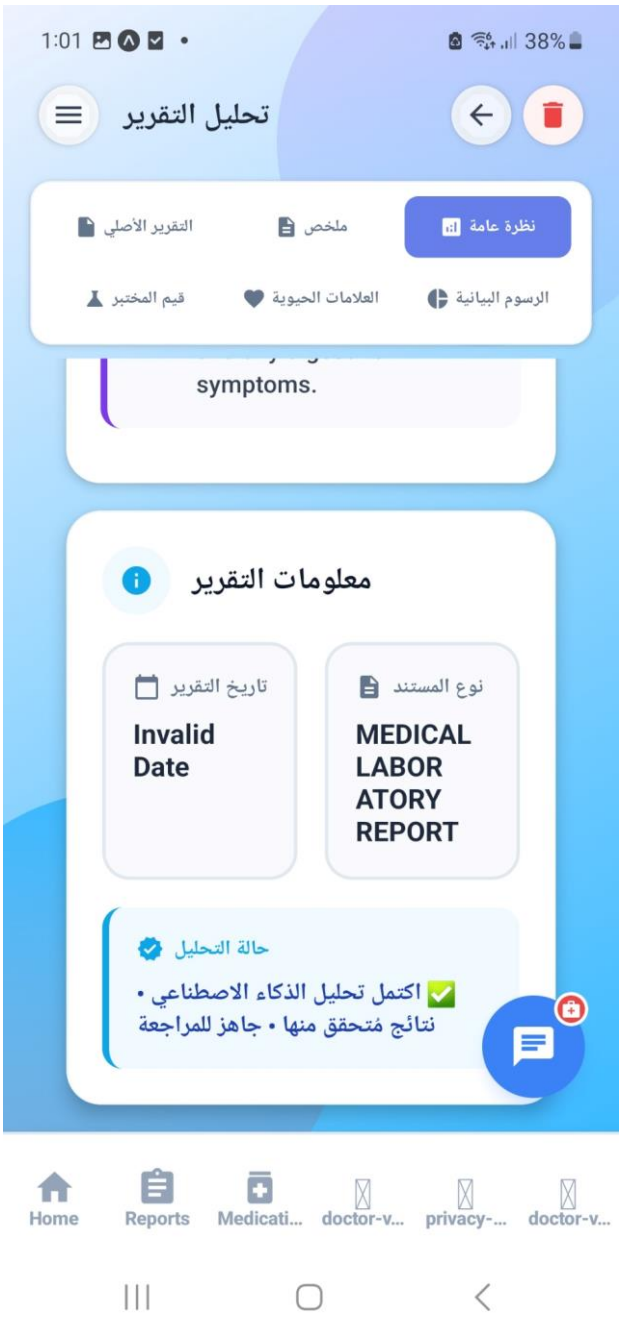

Figure 17 Report analysis – report metadata and AI analysis status.

The Report Information section provides essential context for traceability: report date, document type, and analysis completion status. Displaying metadata helps users confirm they opened the correct file, which matters when multiple reports exist. The "Invalid Date" example highlights an important validation scenario—if the source PDF lacks a parsable date, the UI should fall back to "Unknown" or the upload timestamp, and log the issue for future extraction improvements.

Figure 18 summary counts and example of a normal parameter (HGB).

This view shows the laboratory interpretation module, where all extracted tests are categorized (total, normal, abnormal). Each parameter is represented as a card with the patient's value, measurement unit, and the reference range used for classification. The color-coded status ("NORMAL") provides instant feedback, while the reference range supports transparency and allows users to validate results with clinicians.

Figure 19 example of an abnormal parameter (MCH marked HIGH).

Abnormal findings are intentionally highlighted using strong visual cues (red border, upward arrow, and “HIGH” label). The card still shows the exact value and reference range so the user can understand *why* it is flagged. This design supports quick triage (spot the abnormal items fast) while still encouraging informed discussion with a healthcare professional rather than self-diagnosis.

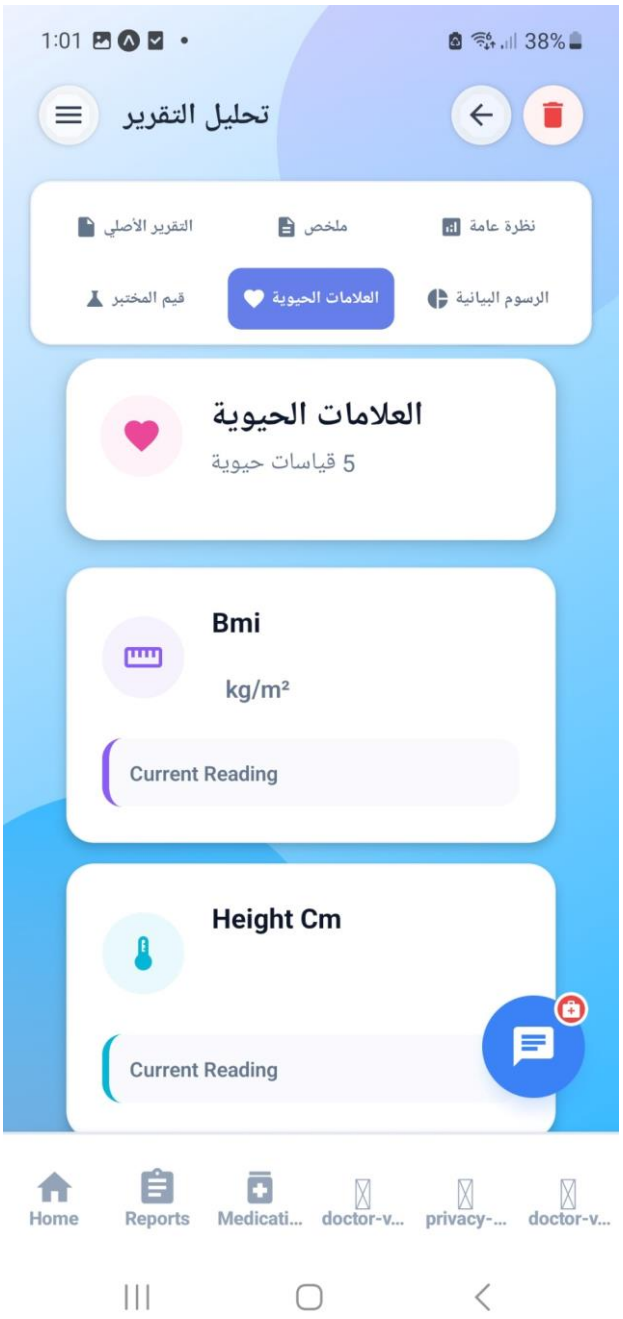

Figure 20 Vital signs – detailed cards for BMI and height readings.

In addition to the main vital-sign summary, users can view detailed cards per vital measurement. Each card is structured to display the current reading and, when available, can be extended with history/trends (e.g., a sparkline or last updated timestamp). The placeholder “Current Reading” indicates a missing value case, enabling the app to prompt the user to complete their profile for more accurate analytics and recommendations.

Figure 21 Charts – distribution of lab results (normal vs. abnormal).

This visualization layer converts tabular lab interpretations into an at-a-glance chart. The donut chart summarizes the distribution of normal and abnormal results to help users understand overall report risk without scanning each test. This section can be extended with additional charts (e.g., per-category breakdown, trend comparisons across multiple reports) to support long-term monitoring.

Figure 22 MedGemma assistant – in-context chat for report Q&A.

The MedGemma chat overlay provides a conversational interface that is directly connected to the currently opened report. The assistant can answer user questions about specific values, explain terminology, and suggest safe follow-up questions. Embedding the assistant inside the report screen reduces context switching and encourages guided interpretation, especially for Arabic-speaking users who prefer natural language explanations.

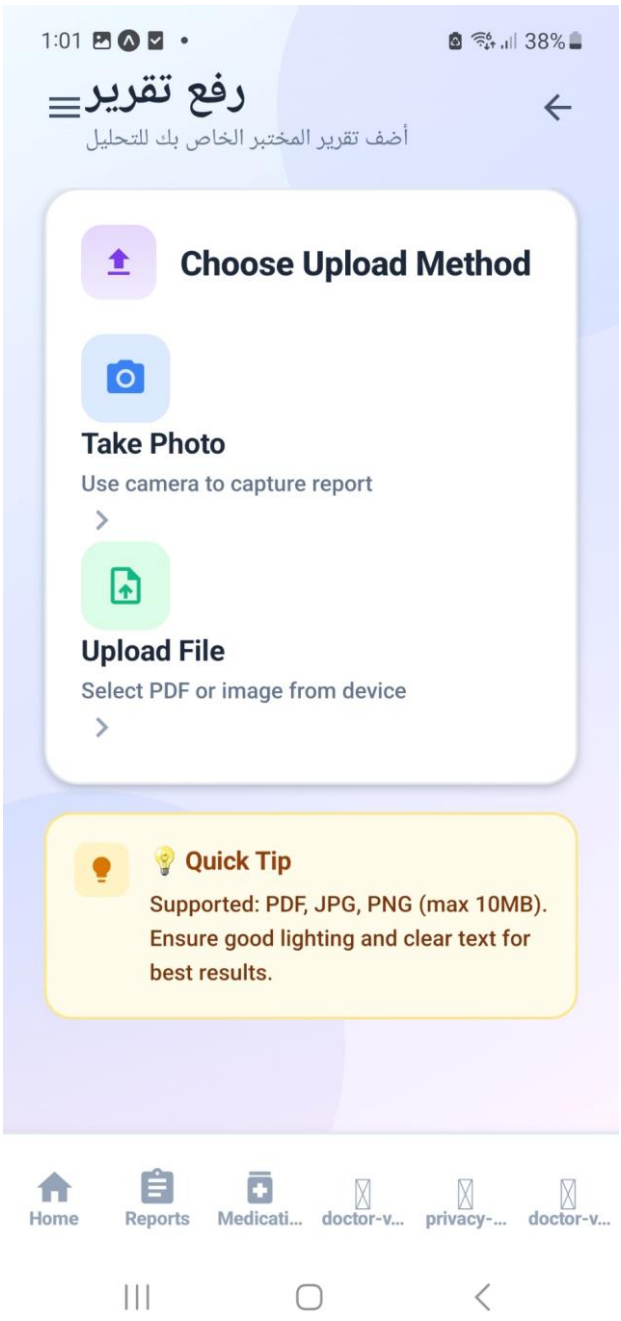

Figure 23 Report upload – choosing the import method (camera or file).

This screen represents the entry point of the upload workflow. Users can either capture the report via the camera (useful for printed documents) or upload an existing PDF/image from the device. The “Quick Tip” box communicates supported formats and size limits, and provides practical guidance (lighting/clarity) to improve extraction accuracy and reduce OCR errors.

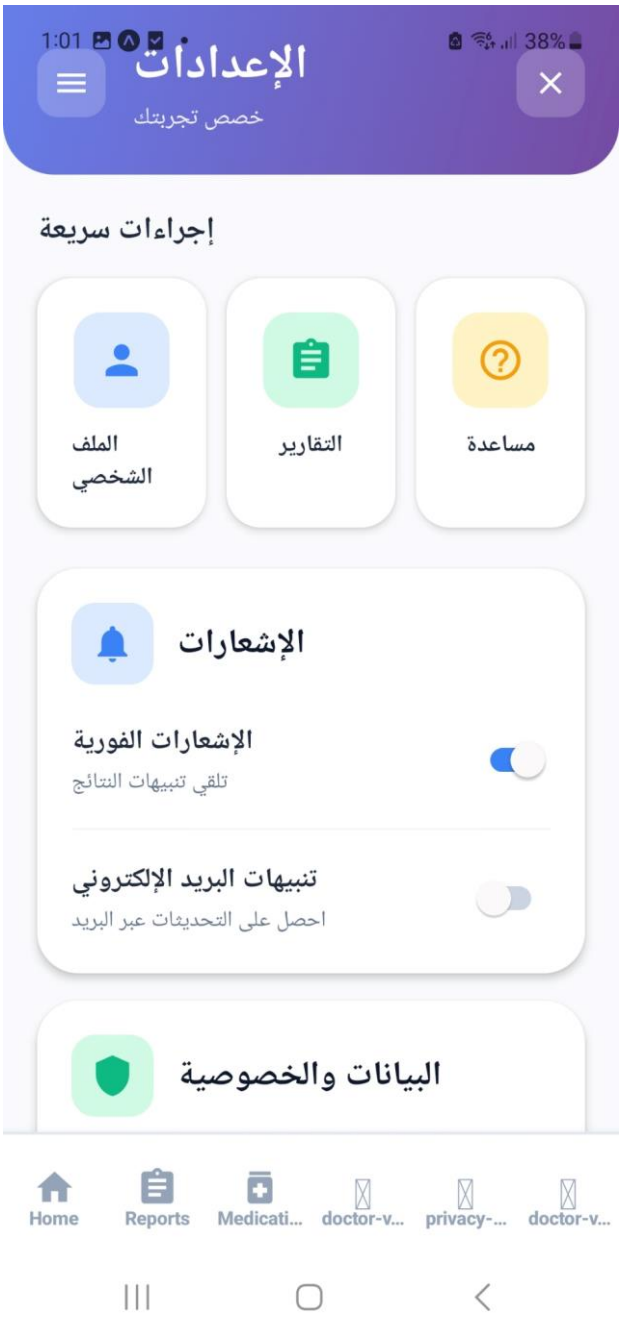

Figure 24 Settings – quick actions and notification preferences.

The settings screen gives users control over their experience and privacy. Quick action tiles offer fast access to profile management, reports, and help, while notification toggles let users choose how they receive reminders and analysis updates (in-app and email). Providing these controls is important for trust and usability, particularly in a healthcare context where users may have different preferences for sensitive notifications.

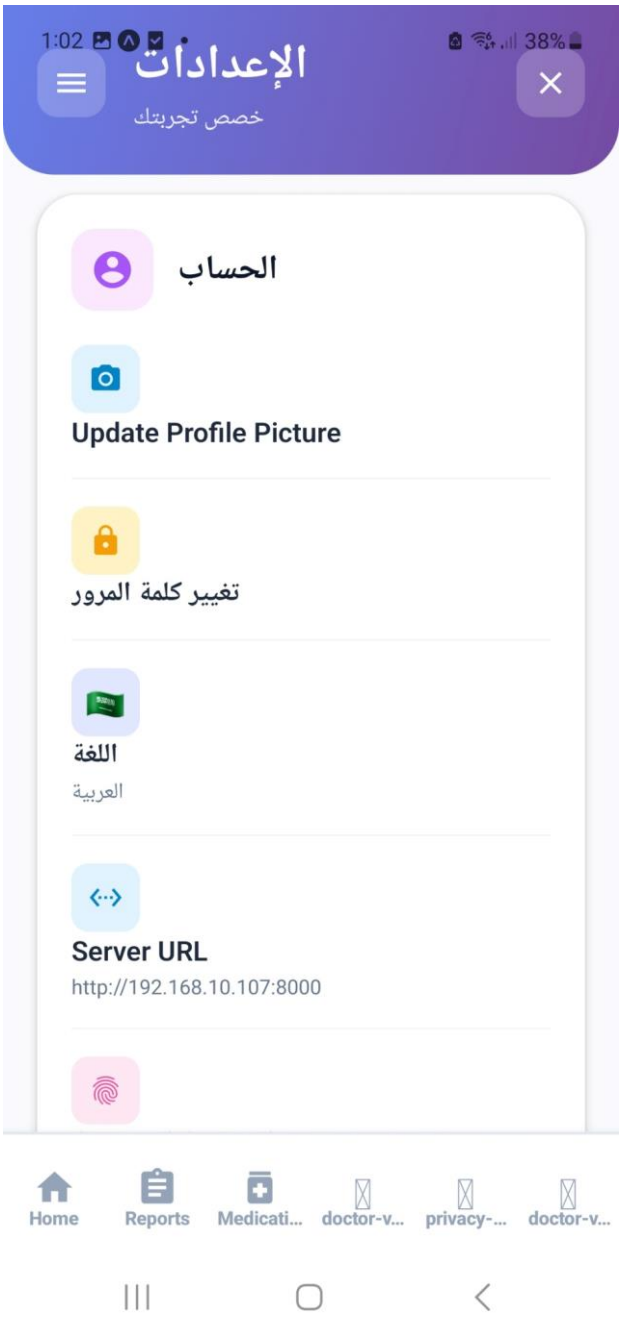

Figure 25 Settings – account options, language, and server configuration.

This view contains account-level controls such as updating the profile picture, changing the password, and switching the app language (Arabic in this example). Displaying the Server URL is useful during development and testing to confirm the active backend environment (local/LAN/staging). In a production release, this field would typically be hidden behind a developer mode or protected settings to avoid accidental misconfiguration by end users.

4.6.4 Emergency Profile, Location-Based Services, and Medication Recognition (Batch 3)

This set of UI screenshots demonstrates the emergency and medication-support capabilities of the Health Tracker application. The screens focus on rapid access to critical patient information, location-based discovery of nearby hospitals and pharmacies, and AI-assisted medication identification with an analysis history for later reference.

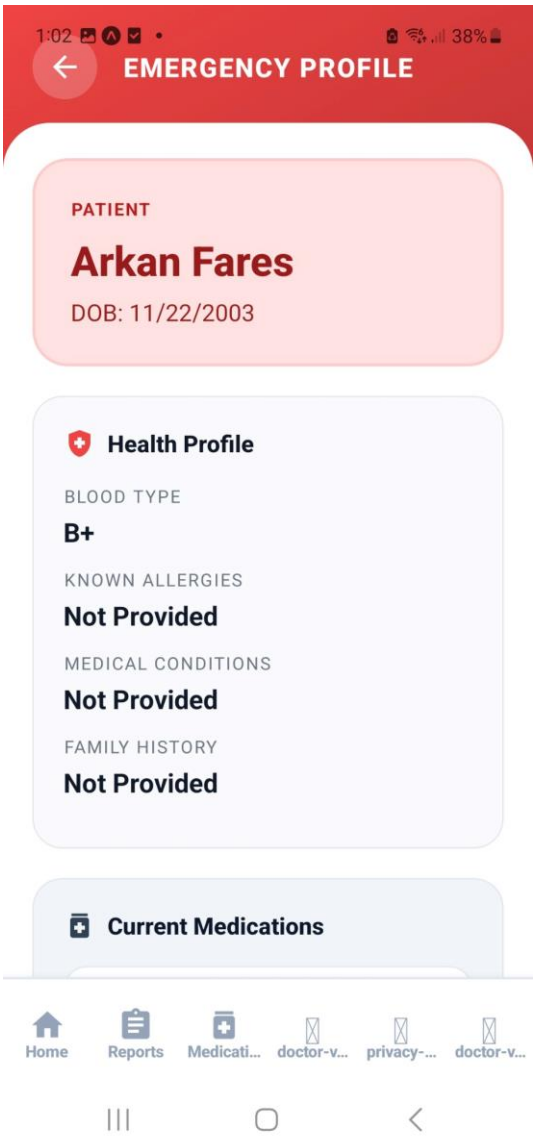

Figure 26 Emergency Profile overview showing patient identity and core health information

This screen is designed for urgent situations where a clinician, paramedic, or family member needs the most important medical information in a few seconds. The header

clearly labels the page as an Emergency Profile and provides a back button for quick navigation.

At the top, the Patient card displays the user's full name and date of birth (DOB). Presenting identity information first reduces the risk of mixing records, especially when multiple people are using the same device or when sharing a screenshot during an emergency.

The Health Profile section summarizes critical, high-impact fields: blood type, known allergies, medical conditions, and family history. When information is missing, the system shows "Not Provided" instead of leaving a blank value. This is a deliberate safety choice: it avoids misleading assumptions and prevents null/empty-field UI failures.

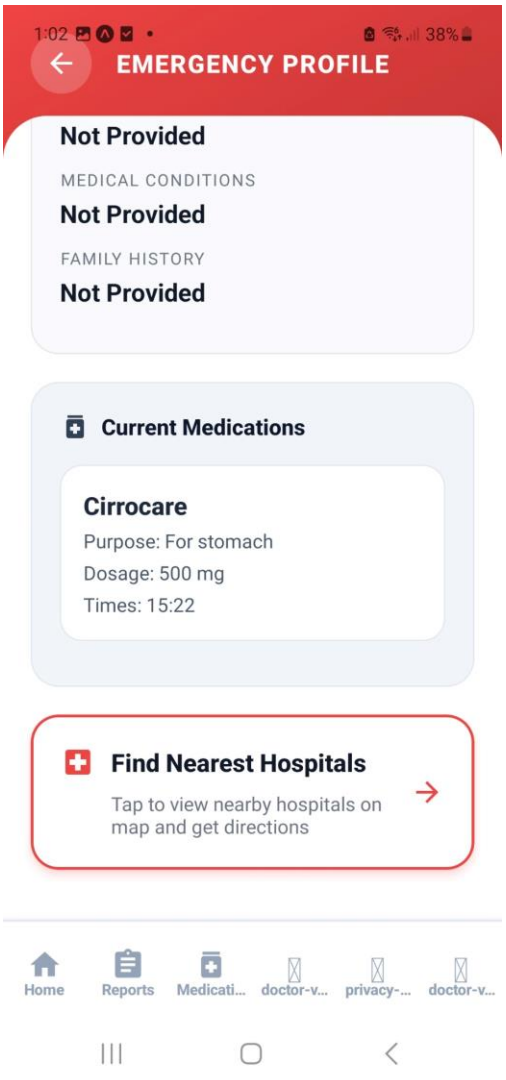

Figure 27 Emergency Profile details with current medications and nearest-hospital shortcut

Scrolling further down reveals the Current Medications card, which highlights active medications that may affect treatment decisions (for example, antibiotics, anticoagulants, or chronic prescriptions). Each medication entry is formatted with the medication name, purpose, dosage, and scheduled time(s) to provide a quick clinical snapshot.

Below the medication list, a large call-to-action card ("Find Nearest Hospitals") guides the user toward immediate help. The strong border and iconography are intentional: in a stressful moment, the UI should make the next action obvious without requiring reading long text.

This section depends on device location permissions. If permissions are denied or GPS is unavailable, the application should handle it gracefully by explaining the issue and offering a retry or manual search option (rather than failing silently).

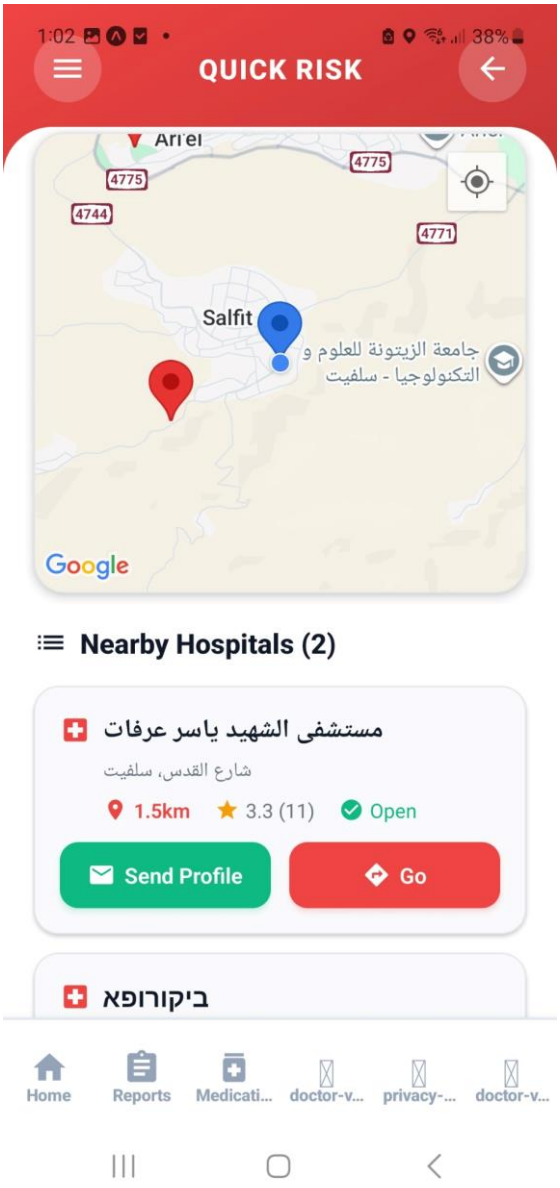

Figure 28 Nearby Hospitals map view showing user location and hospital markers

This screen presents a map-based view of nearby hospitals using Google Maps. The blue marker represents the user's current location, while additional markers represent detected hospitals within the configured search radius.

The combination of map + list supports two user styles: a visual overview for orientation (map) and a structured comparison for decision-making (list). The list header indicates the number of hospitals found, which helps the user confirm that results were successfully retrieved.

The map also includes standard controls (for example, the recenter/target button) so the user can quickly return to their location if the map has been panned or zoomed.

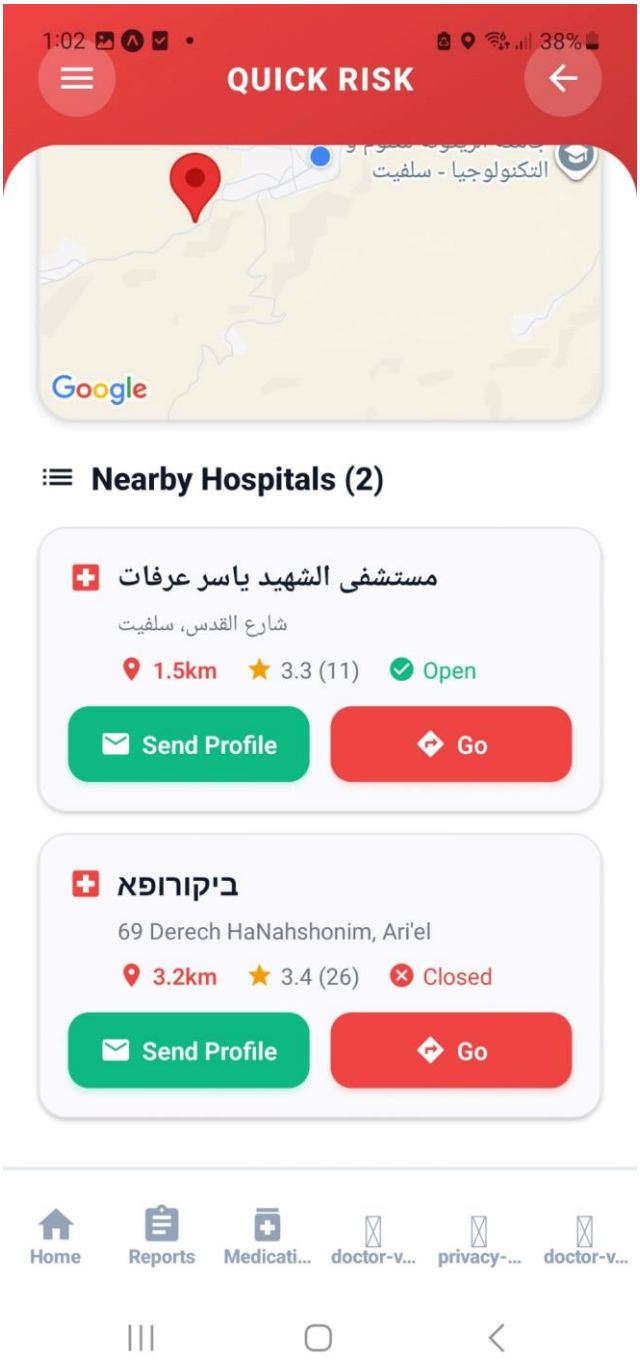

Figure 29 Nearby Hospitals list view with actionable hospital cards (Send Profile and Go)

Below the map, the application displays hospitals as cards containing the facility name, address, distance from the user, rating, and an availability indicator (Open/Closed). These fields come from Places/Maps data and are presented in a compact, scannable layout.

Two primary actions are supported for each hospital: Send Profile and Go. “Send Profile” is intended to share the user’s emergency profile (and optional medication/condition

summary) with the selected hospital or an emergency contact workflow, while “Go” opens turn-by-turn navigation in the device’s mapping application.

The second card in this example demonstrates multilingual compatibility (Arabic/Hebrew/English names). Supporting mixed-language results is important in real-world map data, where place names are often returned in the local language.

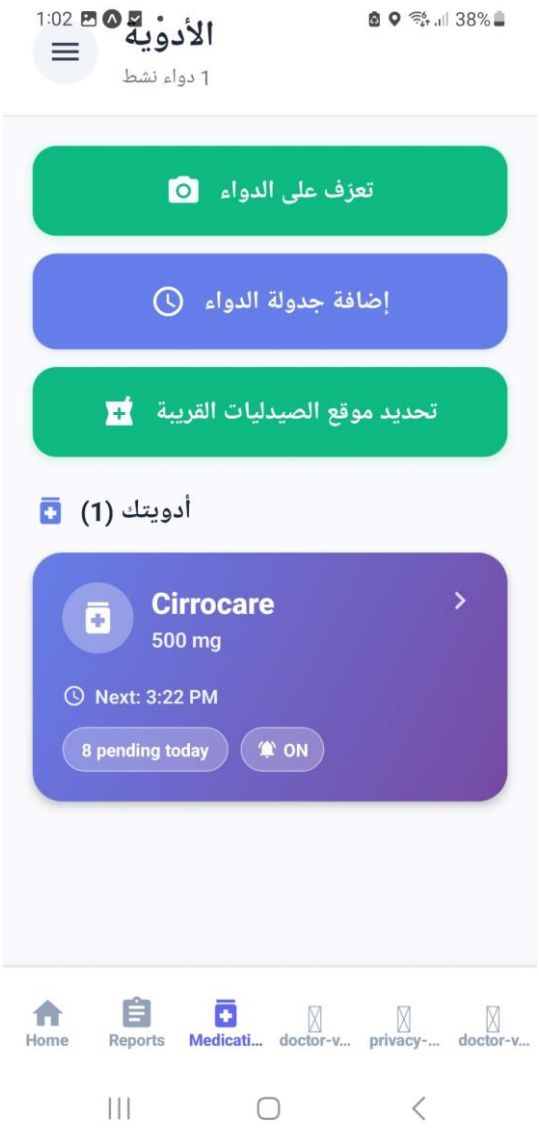

Figure 30 Medications hub in Arabic with quick actions and active medication card

This screen acts as the main entry point for medication management. The interface is localized in Arabic and shows the count of active medications at the top, making the user's current workload immediately visible.

Three quick-action buttons provide a clear workflow: (1) AI-assisted medication recognition (camera-based), (2) adding a medication schedule, and (3) locating nearby pharmacies for refills or urgent access. These shortcuts reduce the number of steps needed for common tasks and improve usability for non-technical users.

The active medication card displays the medication name and dosage, the next scheduled dose time, the number of pending doses for today, and a notification toggle. This design connects scheduling and adherence in one place: the user can review what is due next and confirm that reminders are enabled.

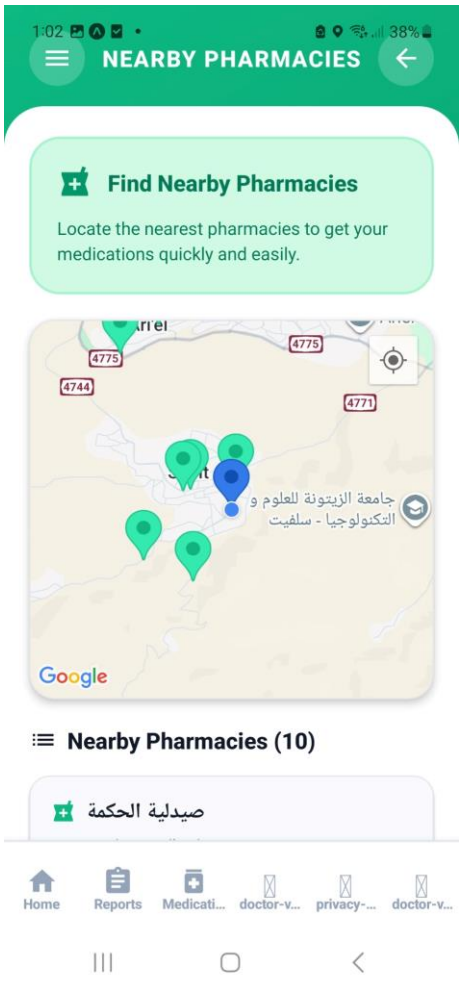

Figure 31 Nearby Pharmacies map view with clustered pharmacy markers and list count

This page provides pharmacy discovery using location services and map rendering. Green markers represent nearby pharmacies returned by the Places search, while the blue marker indicates the user's current position.

A short description explains the feature's purpose ("Locate the nearest pharmacies…"), which is helpful for first-time users. The "Nearby Pharmacies (10)" label confirms the number of results and provides feedback that the search has completed successfully.

From an implementation perspective, this feature must handle common edge cases such as no results in the area, limited network connectivity, or location permission denial. In each case, the UI should provide an informative message and an option to retry.

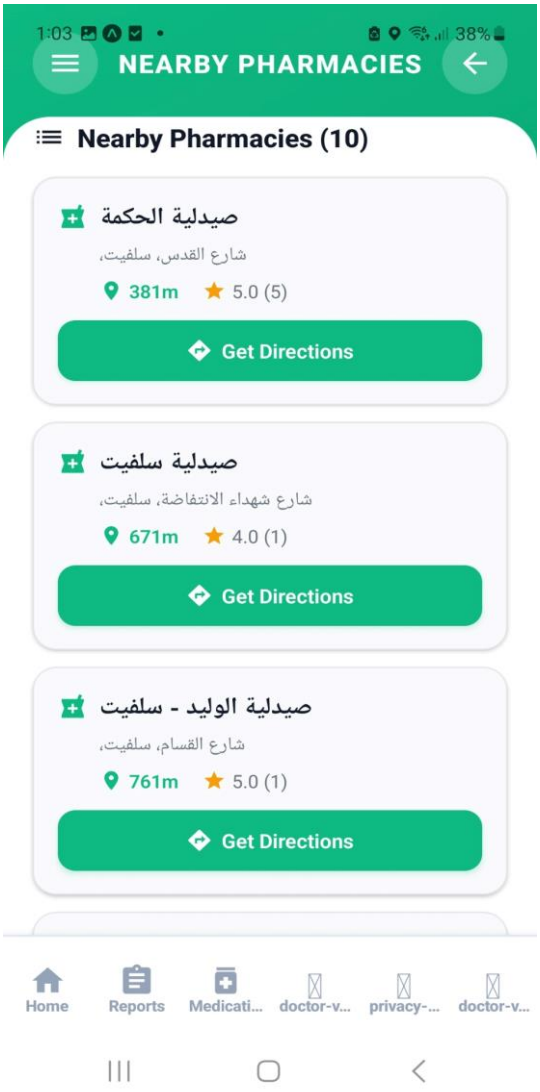

Figure 32 Nearby Pharmacies list view showing distance, ratings, and "Get Directions" action

When the user scrolls, pharmacies are presented as individual cards with their name, address, distance (meters), and rating. This information supports quick comparison and helps the user choose the most convenient option.

Each card provides a single high-confidence action: Get Directions. Selecting it should launch navigation to the pharmacy in Google Maps (or the user's default navigation app), minimizing friction during time-sensitive situations.

The consistent card layout supports long lists by keeping the most important attributes in the same positions, reducing cognitive load when scanning multiple options.

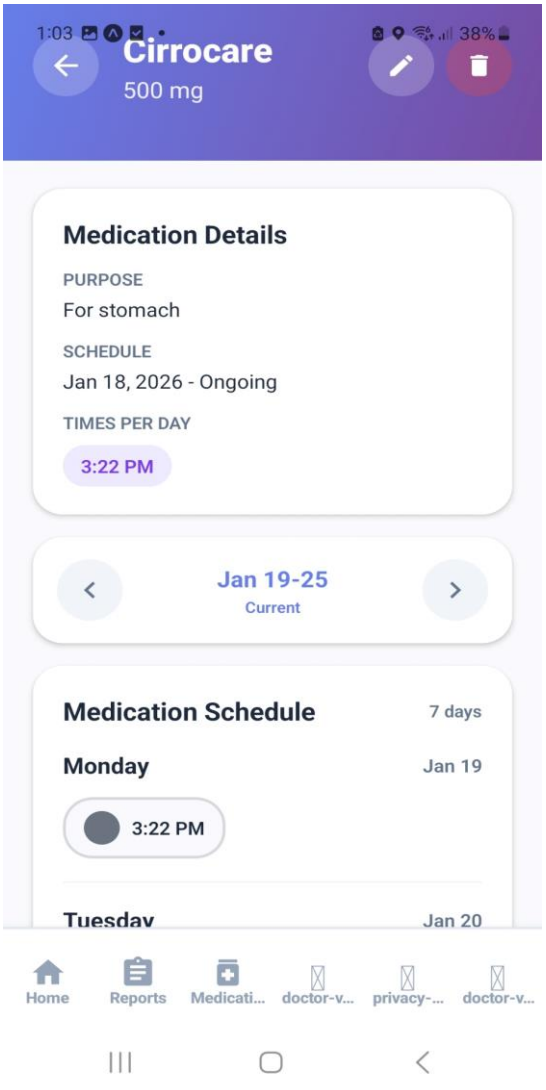

Figure 33 Medication details page with schedule metadata and 7-day plan navigation

This screen provides a detailed view for a specific medication. The header includes quick actions (edit and delete), allowing the user to update incorrect entries or remove medications that are no longer needed.

The Medication Details card summarizes the purpose, the overall schedule range (start date to ongoing), and the configured times per day. Showing the time as a distinct pill/label improves readability and reduces the chance of misreading the dose time.

Below, the weekly navigator (e.g., “Jan 19–25”) allows the user to move through weeks and view their planned intake over a 7-day horizon. This supports adherence planning and makes it easier to recognize patterns such as missed days or repeated reminders.

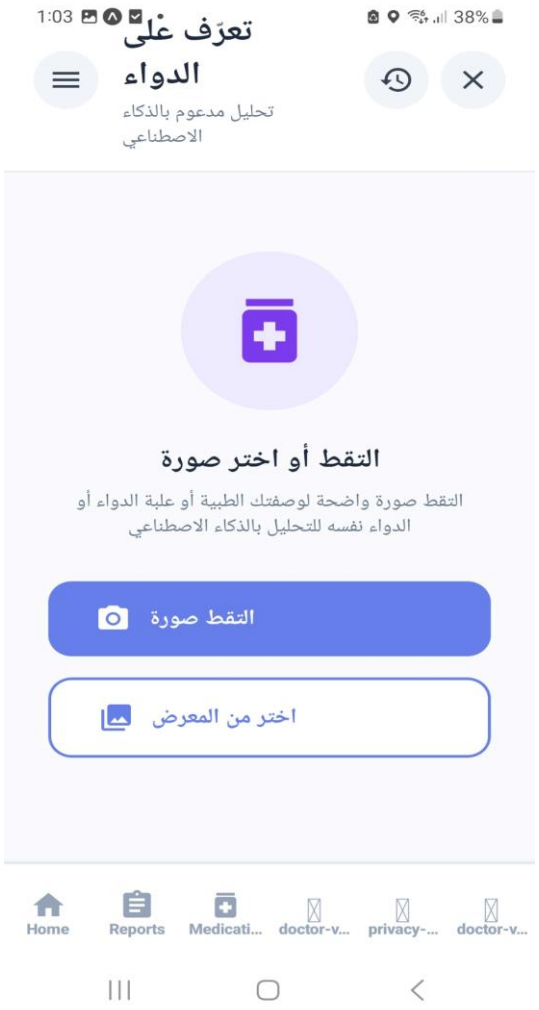

Figure 34 AI medication recognition screen allowing camera capture or gallery selection

This screen enables the AI-supported workflow for identifying a medication from a photo. The instructions emphasize capturing a clear image of a prescription, medication box, or the medication itself, which improves the accuracy of OCR (text extraction) and visual recognition.

Two input options are provided: taking a new photo with the camera or choosing an existing image from the device gallery. Offering both options supports real-world usage where patients may have screenshots, WhatsApp images, or previously saved prescription photos.

After an image is selected, the application can send it to the AI analysis service to extract structured fields such as medication name, dosage, and usage instructions. The user should always be able to review and confirm the extracted data before it is saved to their medication list.

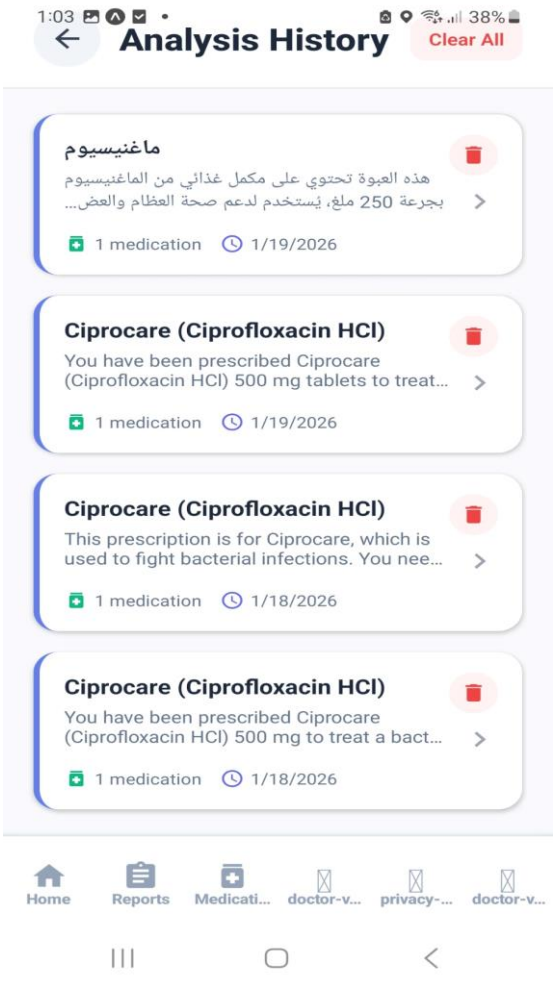

Figure 35 Analysis History page storing AI-generated medication analyses with privacy controls

The Analysis History screen lists previous AI analysis results so the user can revisit them without re-uploading images. Each entry shows a title, a short summary preview, the number of medications identified, and the analysis date.

Per-item delete icons allow the user to remove individual records, while the “Clear All” action provides a fast way to wipe the full history. These controls are important for privacy and data minimization, especially when the device is shared within a family.

The mixed Arabic/English entries demonstrate multilingual support and also highlight that the system stores the user-facing explanation text returned by the AI, not only raw extracted values. This makes the history useful as a “medical notes” reference, not just a technical log.

Chapter 5: Discussion

5.1 Interpretation of Results

The delivered system demonstrates that AI-driven document understanding can be integrated into a mobile health workflow using a secure backend and cloud storage. The separation into layers (client/API/data/AI services) reduces coupling and enables future replacement of models or storage providers without major UI redesign.

5.2 Contribution of This Work

- End-to-end lab report analysis pipeline from upload to structured JSON and mobile visualization.
- AI Doctor module that supports both cloud AI (Gemini) and local AI (Ollama) through a unified API.
- Full medication management workflow (recognition, confirmation, scheduling, logging, and reminders).
- Practical security baseline using token authentication and secure file storage patterns.
- Doctor-Patient Connection module with relationship lifecycle, consent-based permission levels, protected patient data access, and integrated care workflows (chat, prescriptions, appointments, visits).
- Administrative Dashboard & Analytics — role-based user management, usage insights, system monitoring, and AI utilization metrics for system administrators.

5.3 Limitations

- AI outputs may contain inaccuracies; the system is designed for assistance and education rather than diagnosis.

- External AI services introduce latency and cost variability depending on network and provider.
- Clinical validation and regulatory compliance are outside the scope of this academic project but would be required for real-world deployment.

Appendices

Appendix A: Disclaimer Statement Format (Reference)

The disclaimer statement included in this report follows the official format provided by the Faculty of Engineering & Information Technology requirements document.

Appendix B: Sample API Endpoints (Summary)

Table 2 provides a concise overview of representative API endpoints by module.

Module	Endpoint (Example)	Purpose
Auth	POST /api/register, POST /api/login	Create account and obtain access token
OTP	POST /api/send-otp, POST /api/verify-otp	Email verification and password flows
Lab Reports	POST /api/lab-reports/upload	Upload report, store in S3, trigger AI analysis
Medications	POST /api/medications/recognize	Analyze medication image and extract details
Medical Chat	POST /api/medical-chat/message	Send message to Gemini-based AI Doctor
Local AI	POST /api/ai-doctor/message	Send message to local/Ollama model
Sharing	POST /api/share-conversation	Generate shareable link and summary report

Table 2: Representative API endpoints used by Health Tracker.

UI Screenshots – Batch 4 (Medication, Profile, Security, AI Doctor, Doctor Visit)

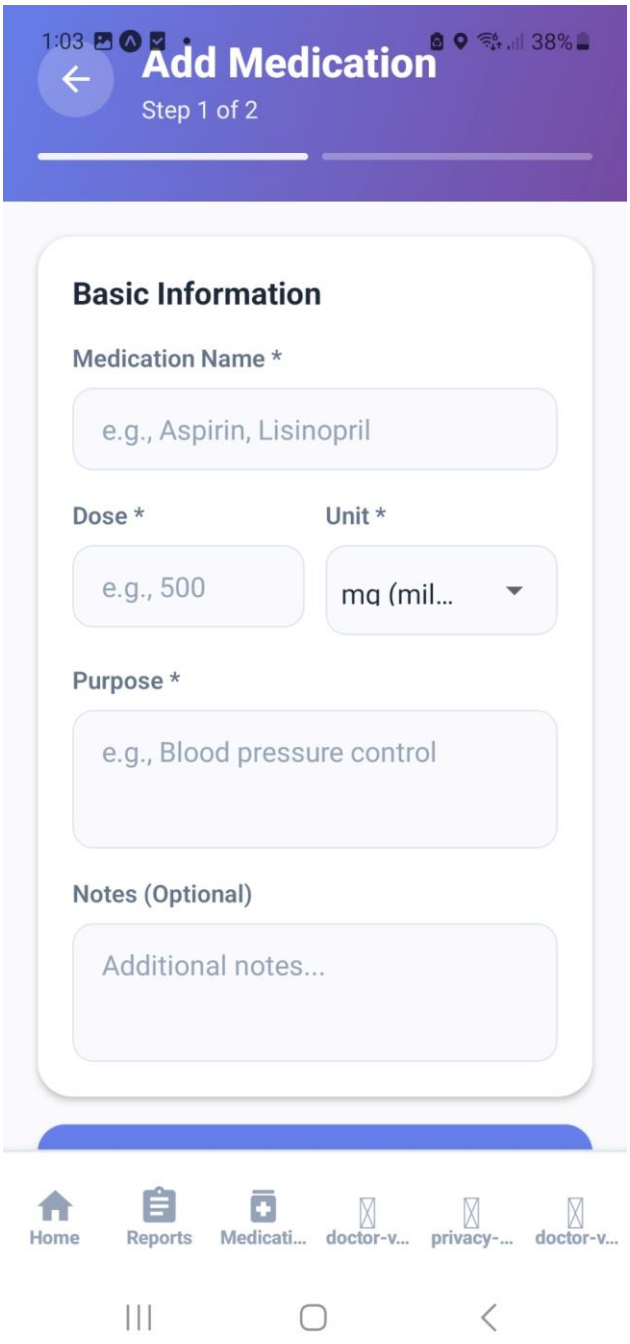

Figure 36 Add Medication – Step 1 (Basic Information)

This screen is the first step of the medication creation workflow. It focuses on collecting the essential medication metadata that will later drive scheduling, reminders, and tracking. The form captures the medication name (with clear examples), dose value, and dose unit (dropdown), which together define a precise dosage such as “500 mg”.

In addition, the user records the medication purpose (e.g., blood pressure control) to provide context in the medication list and in reminder notifications. An optional notes field supports free-text instructions such as “take after food” or “avoid driving”, which helps users personalize their plan. The step indicator (“Step 1 of 2”) provides progress feedback, while the back navigation maintains a standard, low-friction mobile flow.

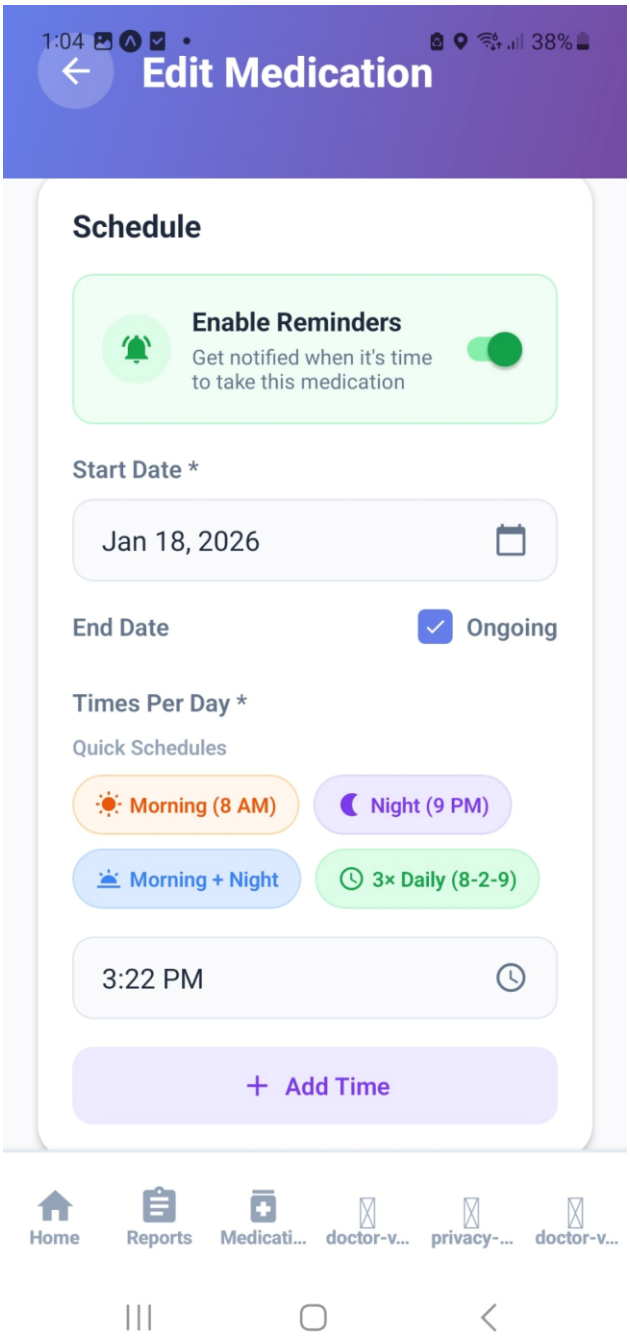

Figure 37 Edit Medication – Scheduling and Reminder Configuration

This screen manages the medication schedule, turning the basic medication definition into an actionable daily plan. A dedicated “Enable Reminders” toggle allows users to switch notifications on or off without deleting the medication, supporting real-world use cases such as temporary pauses or physician-directed changes.

The user selects a start date and (optionally) an end date, with an “Ongoing” option for chronic medications. “Times Per Day” is simplified through Quick Schedules (morning, night, morning+night, or 3× daily), which pre-populate times to reduce user effort and input errors. Users can still customize times using the time picker and the “Add Time” action, enabling flexible regimens such as irregular dosing or non-standard schedules. These time entries are later used by the notification system to generate scheduled reminders and by adherence tracking to log taken/missed doses.

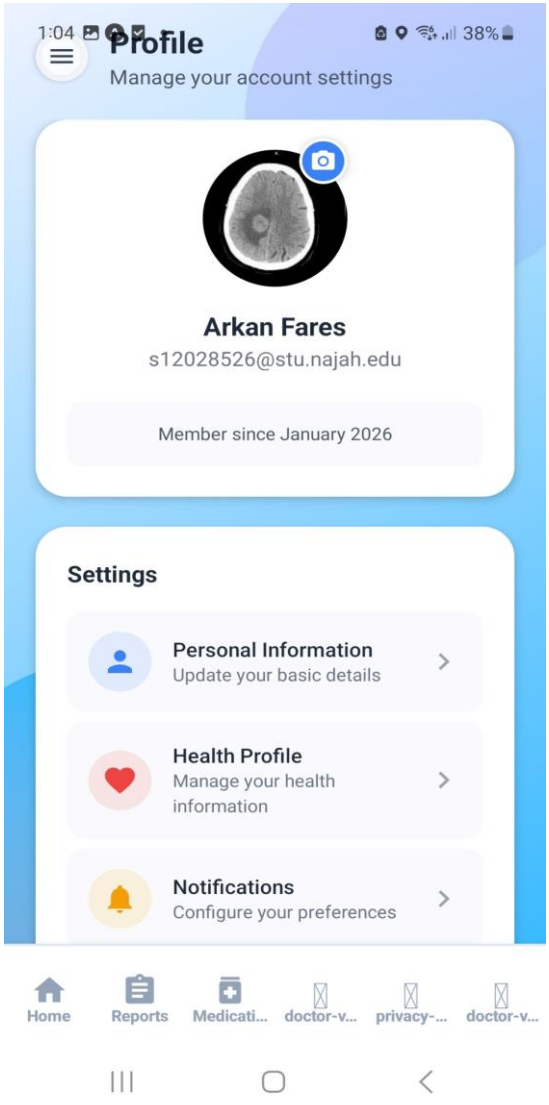

Figure 38 Profile – Account Hub and Settings Navigation

The Profile screen acts as the central hub for account identity and configuration. It displays the user's avatar, name, email address, and membership information, providing quick confirmation that the user is signed into the correct account.

A camera icon on the profile image indicates that the avatar can be updated, supporting personalization and better user recognition. Below the account card, the Settings area provides clear entry points to key modules: Personal Information (basic account details), Health Profile (medical context used across the app), and Notifications (reminder configuration and history). This structure keeps high-impact settings easy to find while maintaining a clean and consistent navigation model.

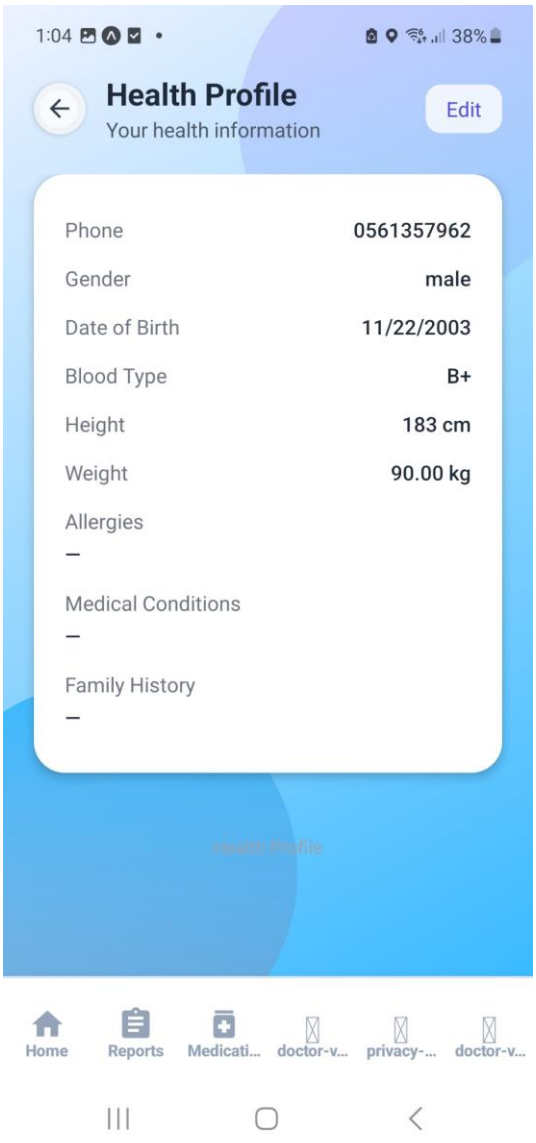

Figure 39 Health Profile – Stored Patient Context (View Mode)

This screen presents the user's health profile in a readable, structured format. It includes contact and demographic information (phone, gender, date of birth) as well as clinical context such as blood type, height, weight, allergies, medical conditions, and family history.

An "Edit" action allows controlled updates to these fields. Capturing this information is important because it can enrich multiple app features: medication management (e.g., allergy awareness), doctor-visit summaries (age and relevant history), and AI-assisted explanations that can be contextualized to the patient profile. Separating view and edit modes also reduces accidental changes while keeping the data accessible for review.

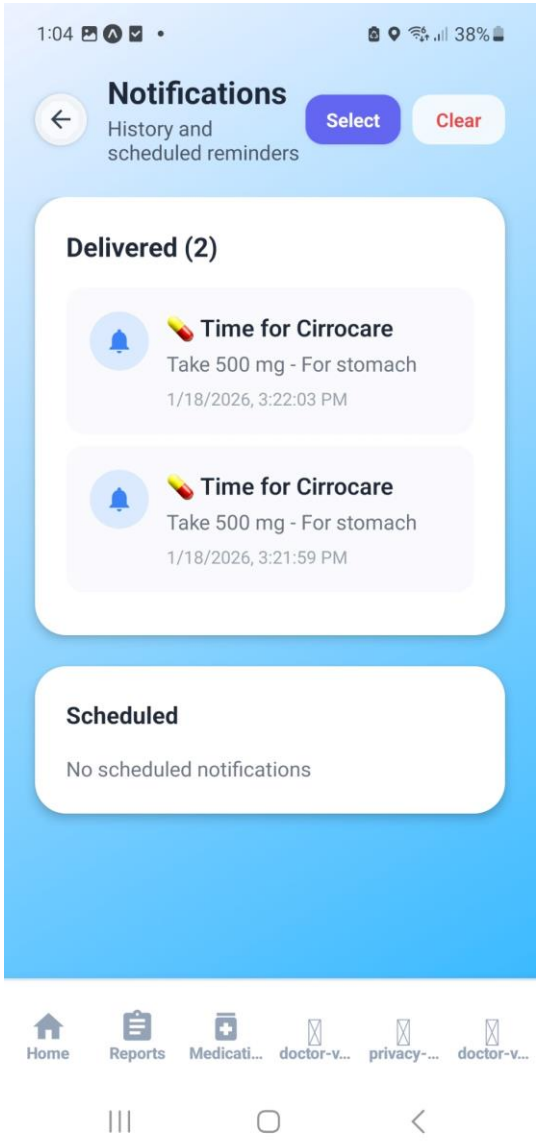

Figure 40 Notifications

The Notifications module provides visibility into reminder activity and helps users manage notification clutter. Delivered notifications are listed with clear titles and supporting details (dose and purpose), enabling the user to confirm what reminder fired and when.

The screen also distinguishes between Delivered and Scheduled notifications. If the schedule is empty or reminders are disabled, the Scheduled section communicates that there are no pending reminders. The “Select” and “Clear” controls support batch management (e.g., selecting multiple delivered notifications and clearing them) so the user can keep the history readable without impacting the underlying medication schedule.

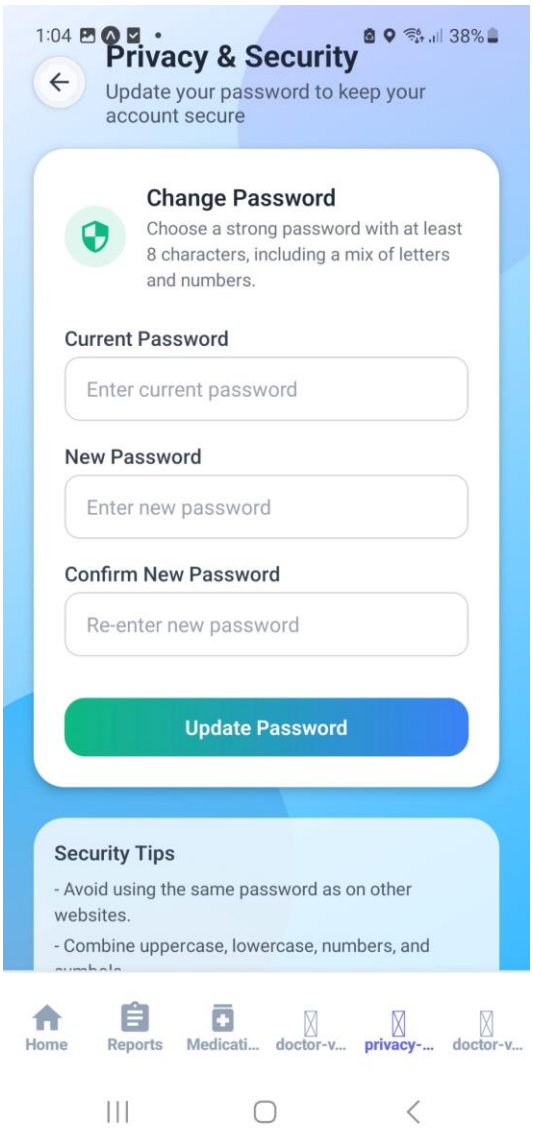

Figure 41 Privacy & Security – Change Password

This screen supports secure account maintenance by allowing the user to change their password. It provides a short, user-friendly guideline for password strength and includes the standard three-field pattern: current password, new password, and confirmation of the new password.

The “Update Password” action is designed to trigger both client-side validation (e.g., minimum length, confirmation match) and server-side validation (verifying the current password, enforcing policy, and securely updating the stored credential). Including “Security Tips” directly on the screen reinforces best practices and reduces weak-password reuse, which is particularly important in a health application that may store sensitive personal and medical data.

Figure 42 Dr. MedGemma – AI Medical Assistant (Start Screen)

Dr. MedGemma is an AI assistant module designed to provide preliminary guidance and patient-friendly explanations. The welcome card introduces the assistant in Arabic and includes an explicit disclaimer that the information is advisory and does not replace a licensed clinician's diagnosis—an important safety and trust feature for any medical AI interface.

The screen provides quick-start actions ("لدي بعض الأعراض" / "سؤال عن صحتي") to guide users toward common intents and reduce the cognitive load of a blank chat. A unified input area supports either typing symptoms or attaching an image (e.g., a report photo) for analysis, enabling multimodal interactions. In the overall system, responses can be generated either via local models (e.g., MedGemma on-device/server) or cloud models depending on the configured pipeline and privacy requirements.

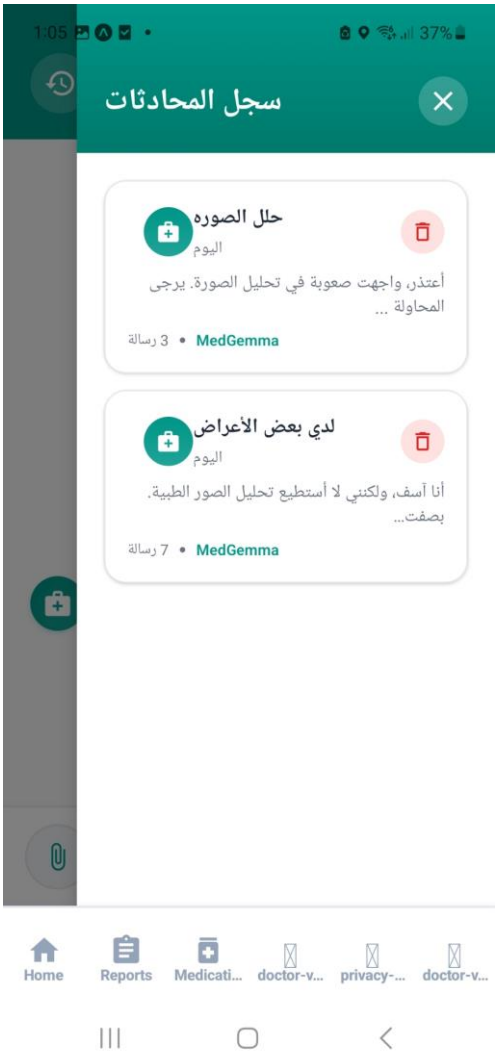

Figure 43 Conversation History – Saved AI Sessions

This modal lists previous AI conversations (“سجل المحادثات”), allowing users to revisit past discussions without losing context. Each card shows the conversation topic (e.g., image analysis or symptoms), the date label, a short preview of the last message, and a message count for quick orientation.

A delete icon is available per conversation to support privacy control and storage management. Persisting sessions enables continuity—users can return to earlier advice, compare changes in symptoms, or reuse prior context when asking follow-up questions. From a technical perspective, this view typically reads from a local database (or backend store) that records session metadata, timestamps, and message history keyed by user account.

Figure 44 Doctor Visit – Recording Setup (Metadata + Microphone)

The Doctor Visit screen is the entry point for recording a doctor–patient consultation and later generating an AI-assisted summary. It first collects visit metadata: visit date, doctor name (optional), and clinic name (optional). This metadata helps organize visits in the history view and improves downstream search and retrieval.

A prominent microphone button indicates the primary action: start/stop recording. Once recorded, the audio can be transcribed (e.g., using Whisper/WhisperX) and analyzed to produce structured outputs such as key complaints, diagnosis hints, medications prescribed, follow-up recommendations, and patient instructions. The screen design intentionally keeps distractions minimal so the user can start recording quickly during real appointments.

Figure 45 Visit History

Visit History provides a chronological list of past visits and their processing status. Entries can appear as “Recording” (actively being captured), or “Completed” once transcription and analysis have finished. Each completed card includes a short preview of the generated summary so users can quickly recall the visit’s outcome without opening the full detail page.

A delete icon allows users to remove visits, which is important for privacy and for cleaning test data during development. Showing the total number of visits and using clear status badges makes the feature feel reliable: users can immediately see whether a visit is still in progress or fully processed, and they can manage their personal medical timeline in one consistent place.

UI Screenshots – Batch 5 (Authentication, Registration, and Doctor Dashboard)

This batch highlights the authentication and onboarding flow (welcome screen, role selection, patient and doctor registration, and email verification), followed by the doctor home dashboard in Arabic (RTL) including overview metrics, quick actions, and profile verification status.

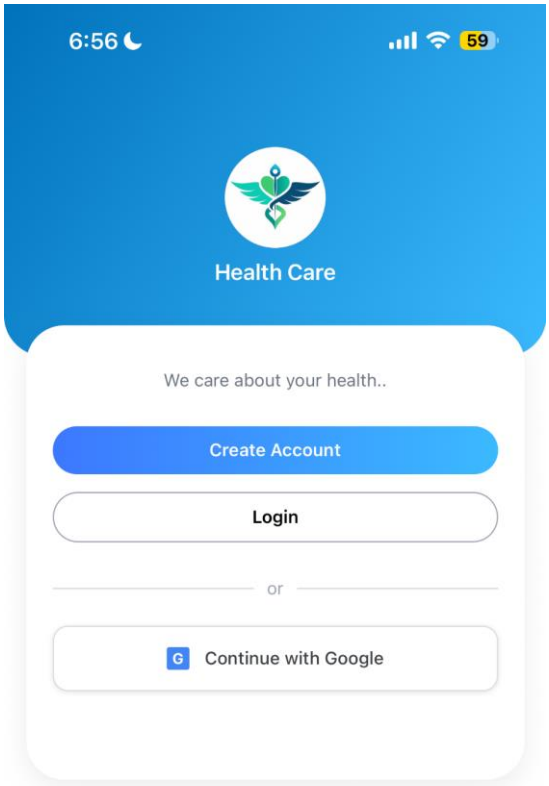

Figure 46 Welcome Screen – Registration and Login

The welcome screen presents the application brand identity with the Health Care logo and a short supportive tagline (“We care about your health..”). Primary actions are offered as large, high-contrast buttons: “Create Account” (primary) and “Login” (secondary), supporting first-time and returning users.

A divider separates traditional email/password flows from the federated option ("Continue with Google"), which reduces friction for onboarding. The layout uses a clean card over a blue gradient background to keep the user's attention on the call-to-action.

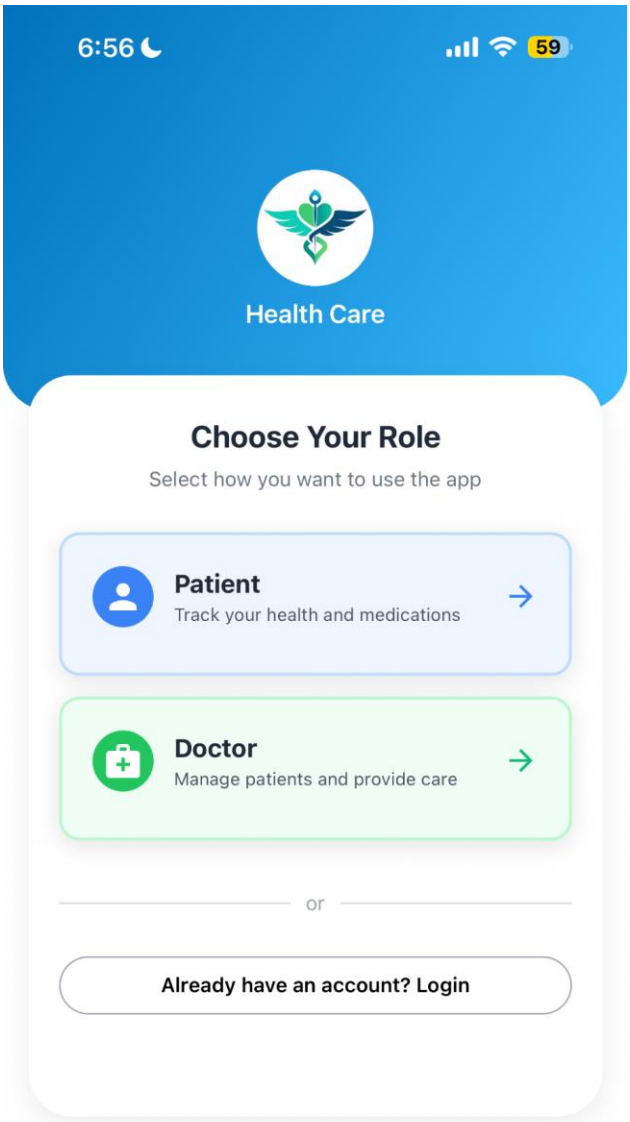

Figure 47 Role Selection – Patient vs. Doctor

After choosing to create an account, the user selects how they will use the app. Two large selectable cards are provided: “Patient” (track health and medications) and “Doctor” (manage patients and provide care).

Each card includes an icon, short description, and a directional arrow indicating navigation to the corresponding registration flow. A secondary button at the bottom (“Already have an account? Login”) enables quick switching for existing users.

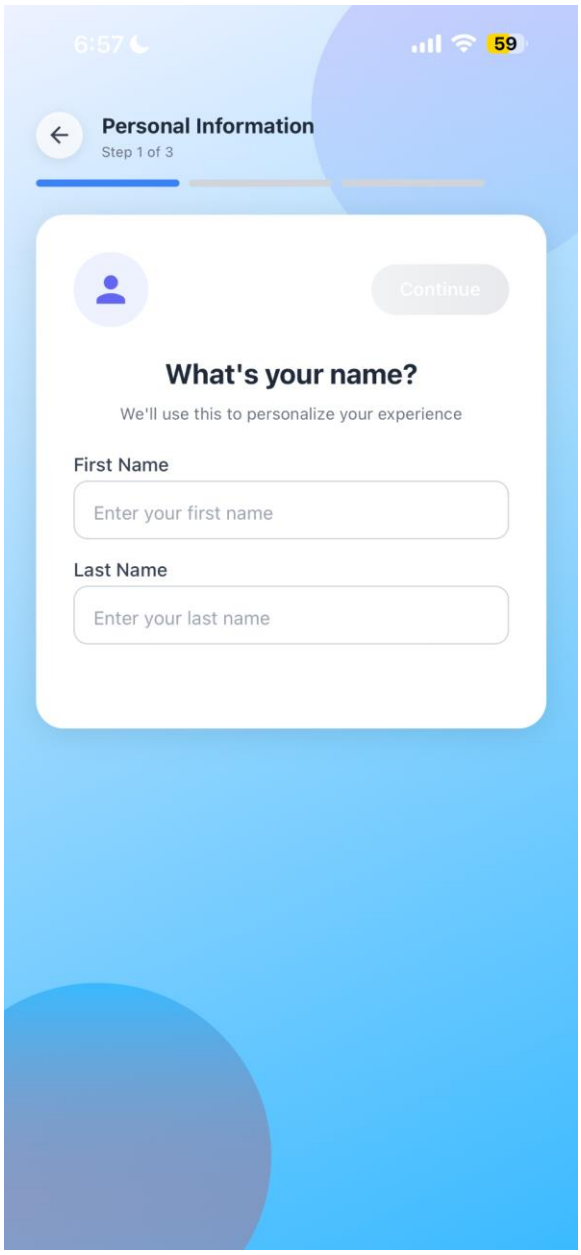

Figure 48 Patient Registration – Personal Information (Step 1 of 3)

This screen collects the patient's basic identity data as the first step of a multi-step onboarding process. A progress indicator ("Step 1 of 3") visually communicates completion status to reduce drop-off.

Two input fields are provided for first and last name. The "Continue" button remains disabled until required fields are completed, enforcing mandatory data entry and minimizing incomplete registrations.

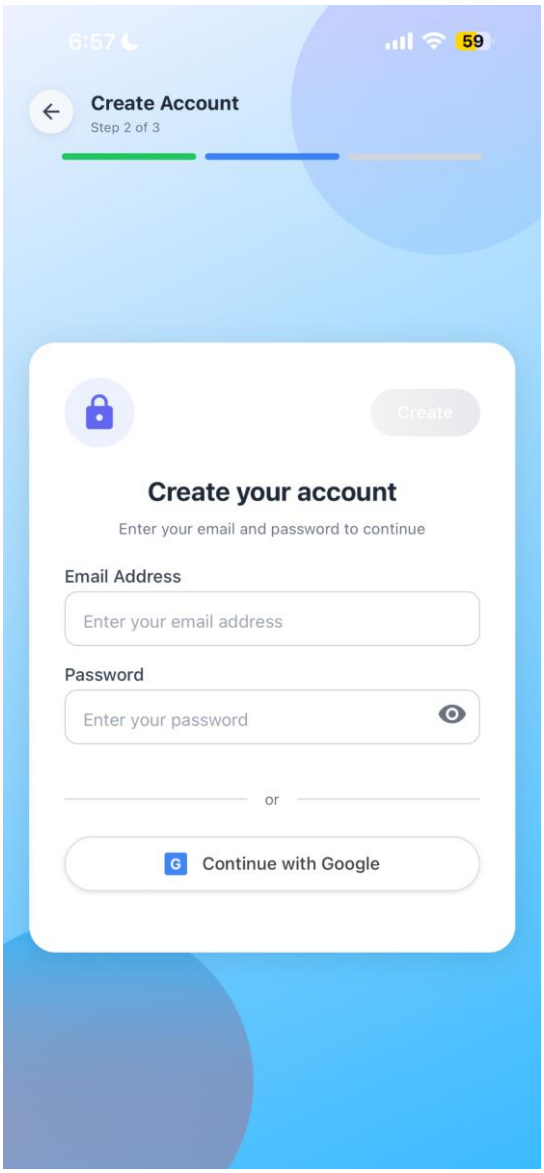

Figure 49 Patient Registration – Account Credentials (Step 2 of 3)

The second step creates the patient's credentials using an email address and password. A password visibility toggle (eye icon) supports usability by allowing users to confirm their input.

A Google sign-in option is also offered as an alternative to manual credential creation. The primary action ("Create") is visually disabled until valid inputs are provided, supporting basic client-side validation.

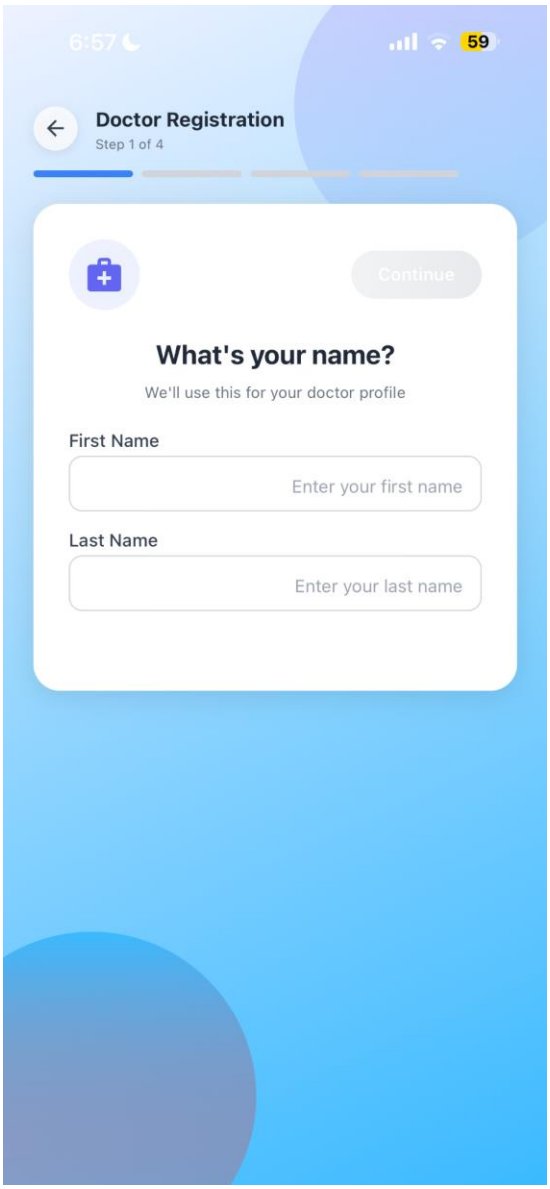

Figure 50 Doctor Registration – Personal Information (Step 1 of 4)

Doctor onboarding follows a similar structured approach, starting with name collection for the professional profile. The progress bar ("Step 1 of 4") indicates a longer registration workflow tailored to doctor-specific requirements.

First and last name fields are provided and the "Continue" button remains disabled until completion, ensuring required profile identity data is captured before advancing to subsequent steps (e.g., additional profile or verification details).

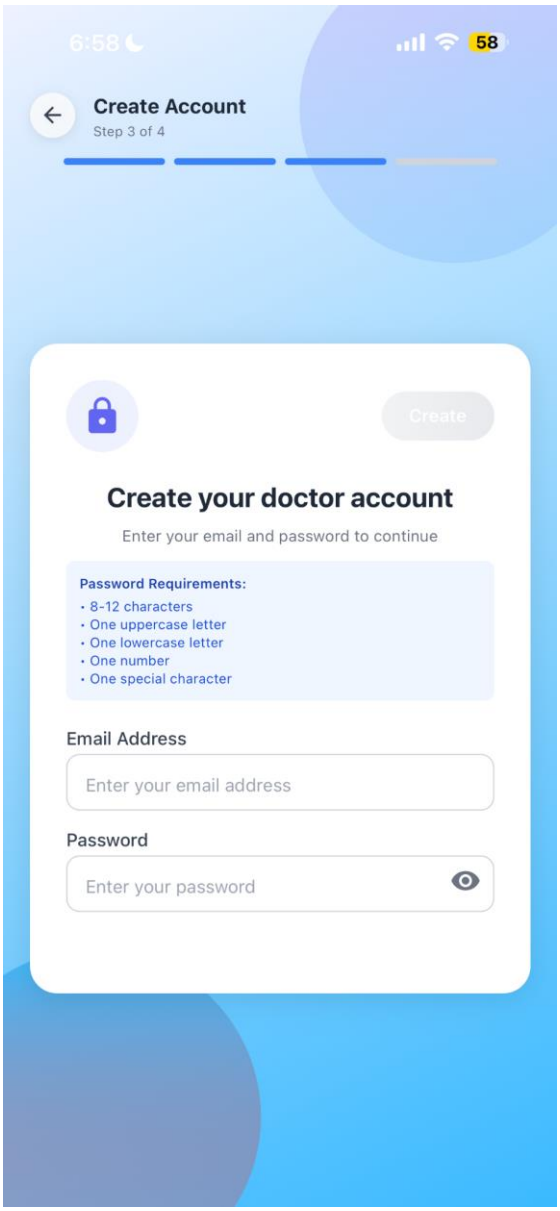

Figure 51 Doctor Registration – Create Doctor Account with Password Policy (Step 3 of 4)

This screen captures doctor login credentials and explicitly displays password requirements (length, uppercase, lowercase, number, and special character). Presenting the policy inline helps users create compliant passwords on the first attempt.

Email and password fields are shown with a visibility toggle, and the “Create” button remains inactive until the password meets validation rules. This reduces backend validation failures and improves registration success rates.

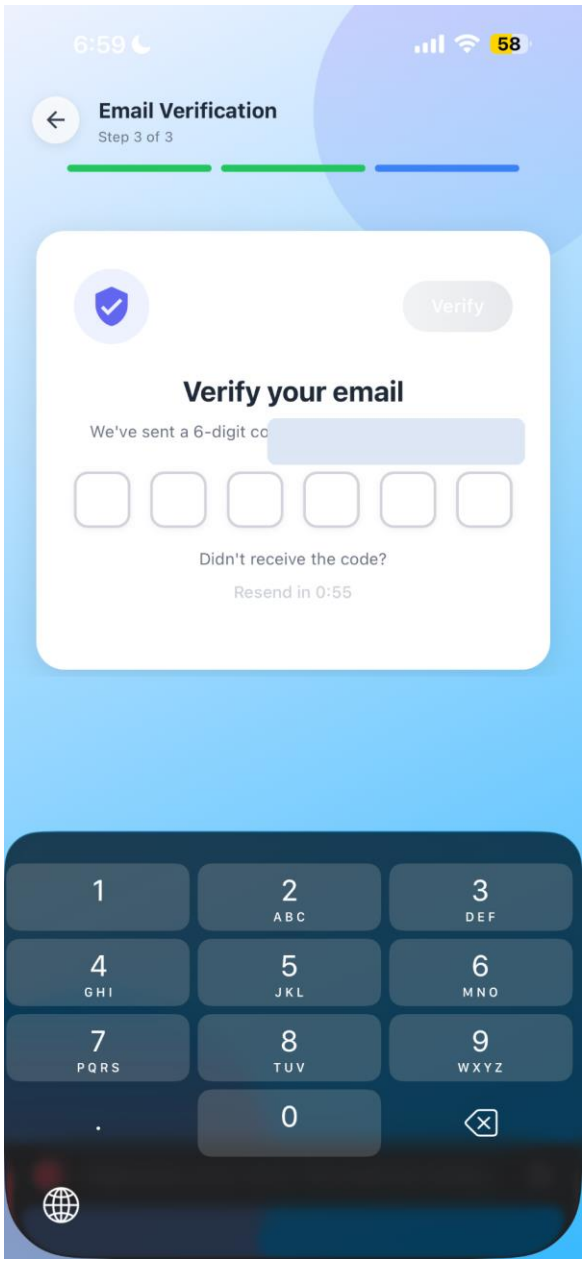

Figure 52 Email Verification – OTP Code Entry (Step 3 of 3)

Following registration, the app enforces email verification using a 6-digit one-time password (OTP). The user enters the code into segmented input boxes designed for fast, error-resistant entry on mobile devices.

A “Resend” timer prevents rapid repeated requests, helping rate-limit OTP delivery. The “Verify” button stays disabled until all digits are entered, ensuring the verification request is well-formed before submission.

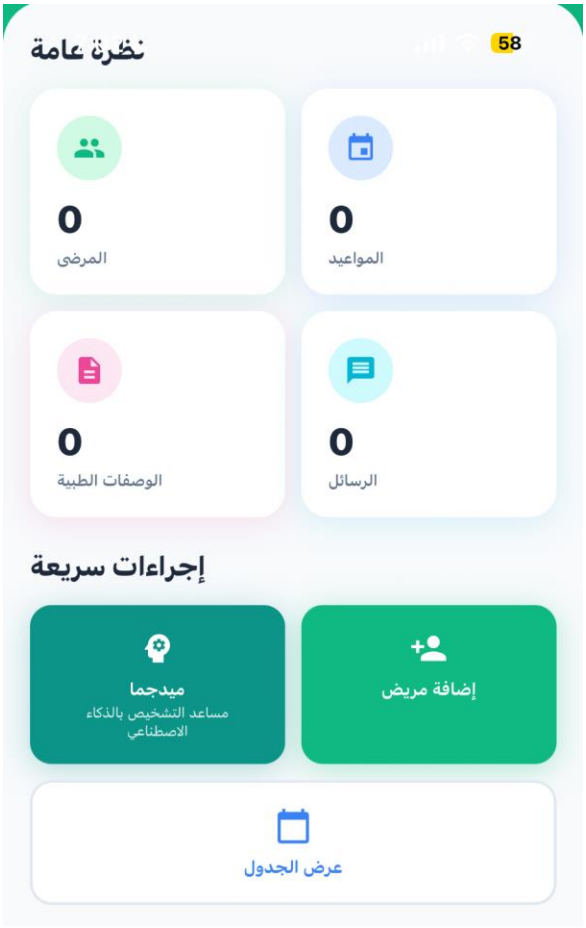

Figure 53 Doctor Home Dashboard

After login, the doctor dashboard displays a personalized greeting (“مساء الخير”) and the doctor name, along with a status note that the profile is under review (“الملف الشخصي قيد المراجعة”). Navigation is available via a hamburger menu and a notification bell with an unread badge.

This screenshot shows the Arabic doctor dashboard (RTL layout) with a high-level summary section presented as four statistic cards (Patients, Appointments, Prescriptions, Messages). Below the summary, a “Quick Actions” area provides large shortcut buttons for launching the MedGemma AI diagnostic assistant and for adding a new patient, followed by a direct shortcut to the schedule view.

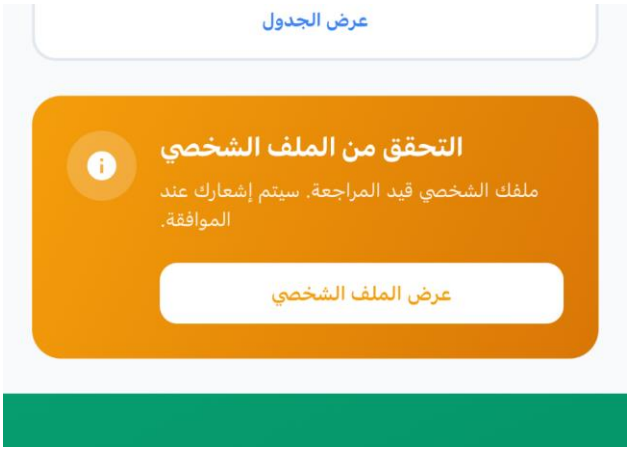

Figure 54 Doctor Home Dashboard – Schedule Access and Profile Verification

Scrolling the dashboard reveals additional utilities, including a prominent "عرض الجدول" (View Schedule) button for quickly accessing the doctor's calendar or appointment timetable.

This screenshot highlights the account verification banner displayed on the doctor dashboard. The banner indicates that the doctor's profile is pending review and approval, and provides a clear call-to-action button ("View Profile") to open the profile page and review submitted details.

UI Screenshots – Batch 6 (Doctor Portal Navigation and Core Modules)

This batch documents additional doctor-side UI screens that support navigation and day-to-day operations, including the side navigation drawer and the core modules for patient management, appointments, prescriptions, and messaging. All screens use Arabic (RTL) typography, consistent iconography, and color-coded module headers to help users recognize context quickly.

Figure 55 Doctor side navigation drawer (module shortcuts and notification indicator).

The screenshot shows the slide-out side navigation drawer opened from the hamburger menu. A profile header card (with avatar placeholder and role badge) appears at the top, along with a close (X) button. The menu lists the main doctor modules with icons: Dashboard, MedGemma, Patients, Appointments, Prescriptions, and Messages. A secondary “Account” section provides access to Profile and Notifications, where an unread badge/indicator is visible to signal pending alerts. The currently selected menu item is visually emphasized to confirm the active page.

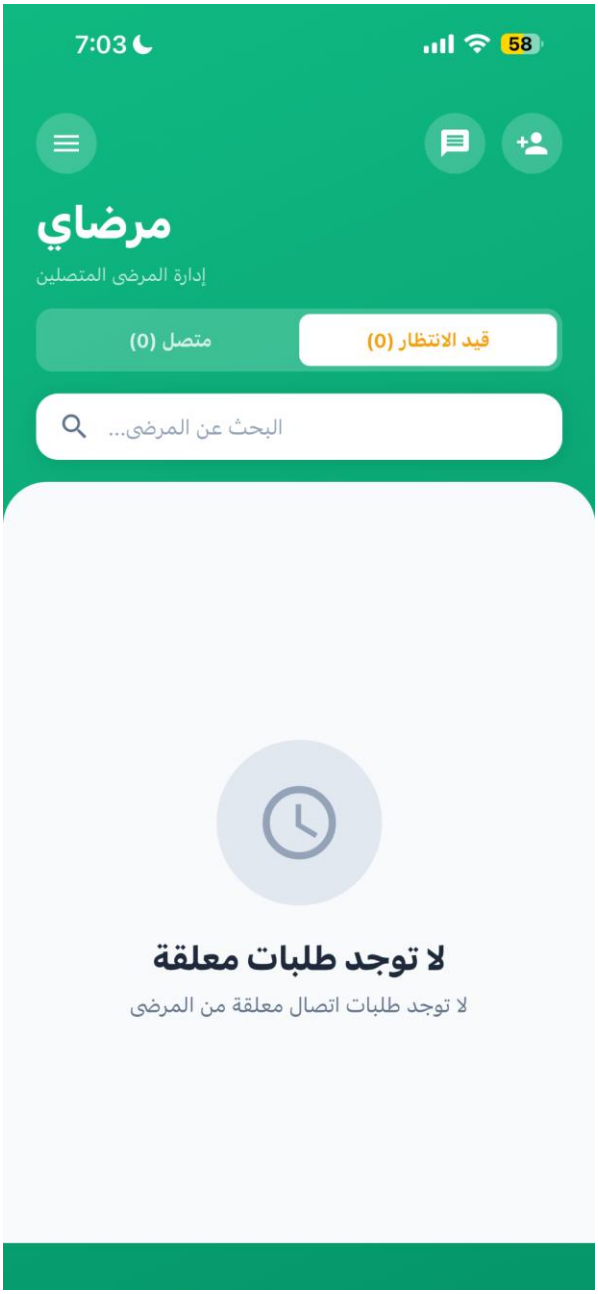

Figure 56 Patients module

This screen represents the doctor's patient management page ("My Patients"). A two-tab segmented control separates connected patients (متصل) from pending connection requests (قيد الانتظار). A search bar enables quick lookup by patient name or identifier. When no pending requests exist, the UI displays a centered empty-state illustration and message ("No pending requests"), ensuring the page still communicates status clearly without data. Top-right actions provide fast access to messaging and to adding a new patient.

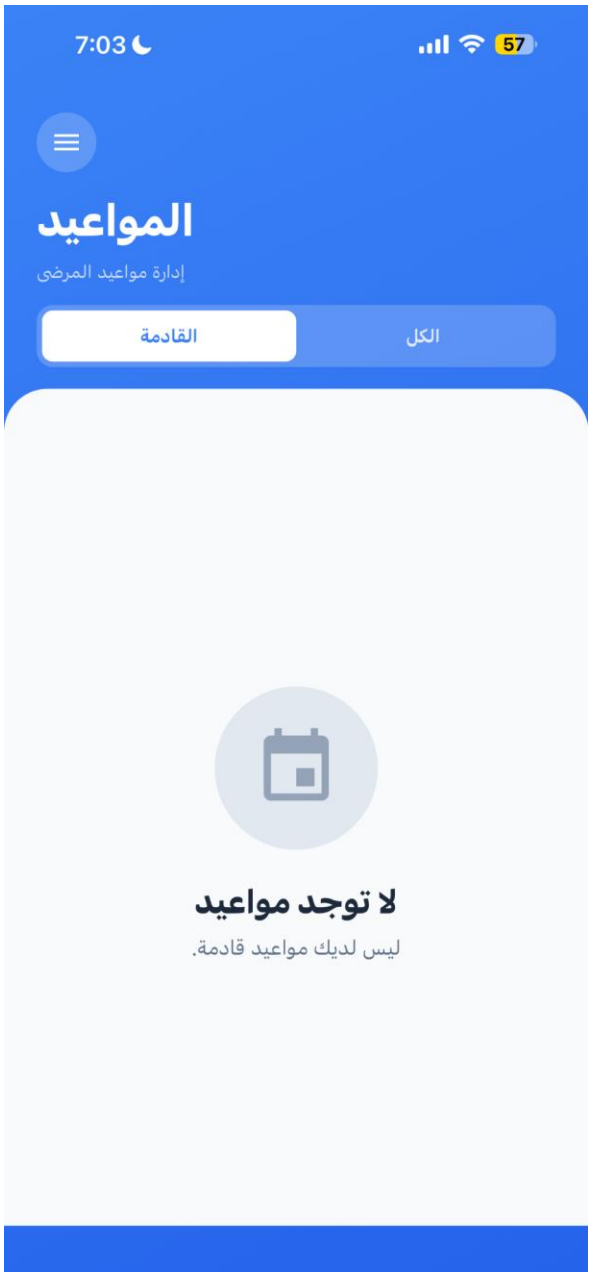

Figure 57 Appointments module

The appointments management screen uses a blue header to distinguish the module context and includes a segmented filter for upcoming appointments (القادمة) versus the full appointment list (الكل). When there are no upcoming appointments, an empty-state illustration and explanatory text are shown, confirming that the system is functioning but currently has no scheduled items. The layout keeps the main content area uncluttered while preserving the filter controls for future use.

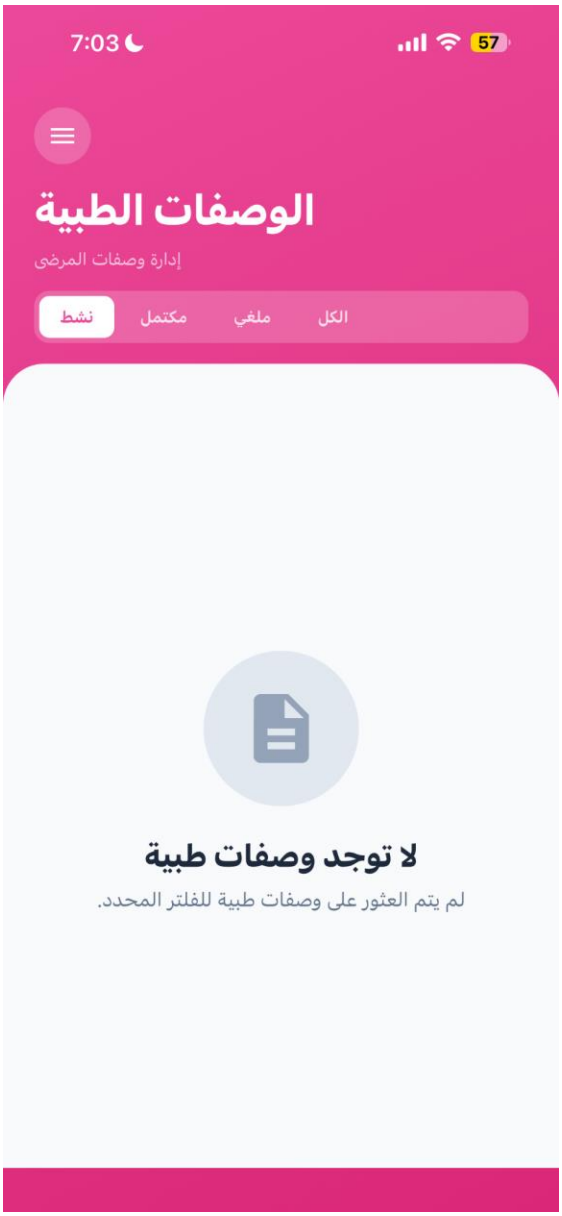

Figure 58 Prescriptions module

This screenshot shows the prescriptions management page (“Medical Prescriptions”) with a pink header theme. A horizontal set of status filters (e.g., Active, Completed, Cancelled, All) allows the doctor to quickly narrow results. When the selected filter returns no records, the UI presents a consistent empty-state message (“No prescriptions found for the selected filter”), which helps users understand that the filter is applied correctly and that there is simply no data in that category.

Figure 59 Messages module

The messages screen (“Messages”) is designed for doctor–patient communication. It features a dedicated header color and a large content area intended for the conversation list. In the absence of active conversations, an empty-state illustration and prompt (“No messages – start a conversation with your patients”) guide the doctor toward the next action, maintaining usability even for first-time users.

UI Screenshots – Batch 7 (Doctor Profile, Settings, Notifications, AI Assistant, Admin Panel)

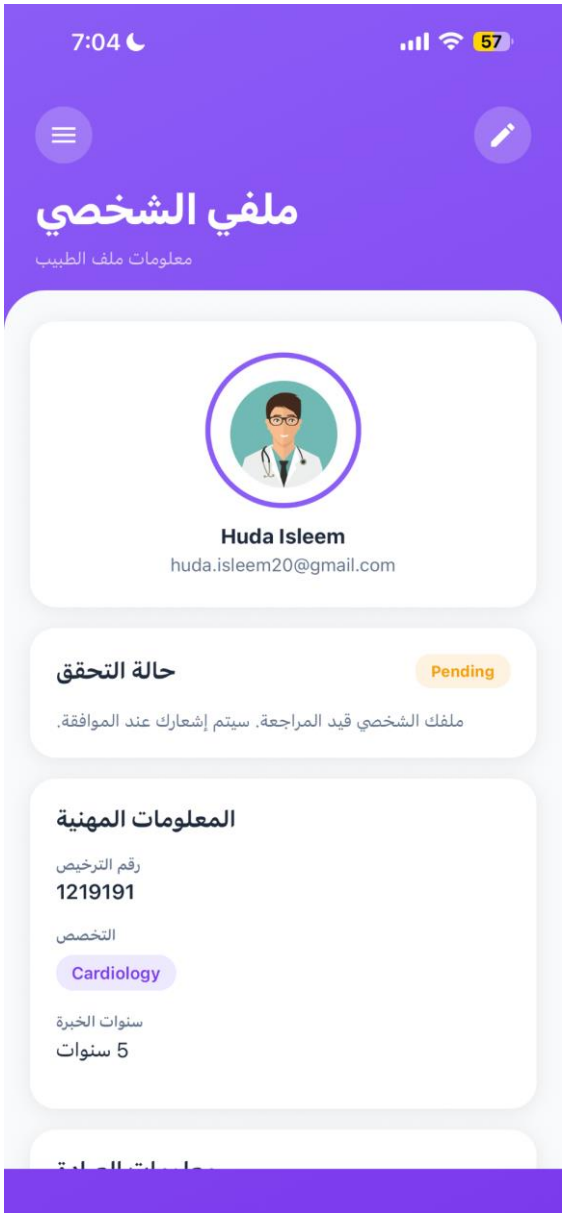

Figure 60 Doctor Profile – Personal File, Verification Status, and Professional Information

This screen presents the doctor's profile summary ("ملفي الشخصي") in a card-based layout. It shows the profile avatar, the doctor's name and email, followed by a dedicated verification-status card marked as Pending, indicating the profile is under review and the user will be notified upon approval. A professional information section below lists key credentials such as license number, specialty (shown as a pill chip), and years of experience.

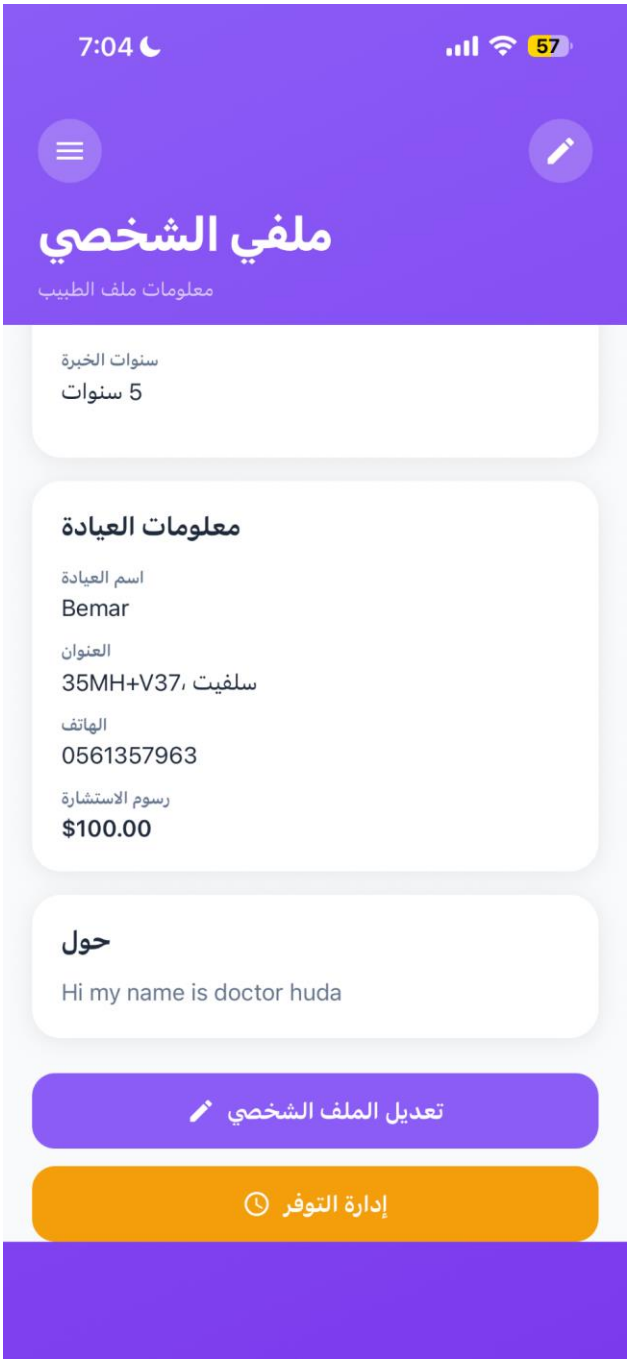

Figure 61 Doctor Profile – Clinic Details, About Section, and Primary Actions

The lower part of the profile page focuses on clinic-related information and public-facing details. It includes the clinic name, address, phone number, and consultation fee displayed in a structured form-like card. An "About" field provides a short biography, while two prominent call-to-action buttons enable editing the profile and managing availability ("إدارة التوفر"), supporting the doctor onboarding and scheduling workflow.

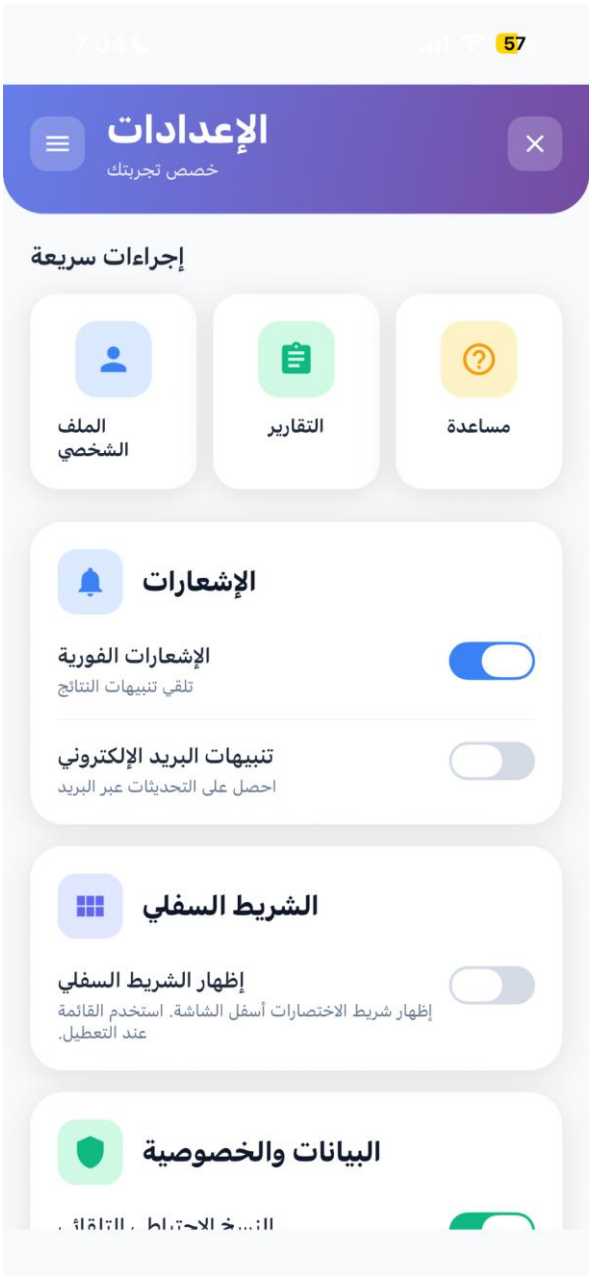

Figure 62 Settings

The settings hub (“الإعدادات”) uses quick-action tiles to jump to common areas (profile, reports, help). Below, the notifications panel offers separate toggles for instant notifications and email notifications, allowing granular communication preferences. A bottom-navigation section provides a master switch to show or hide the bottom shortcut bar, with additional privacy controls available further down the page.

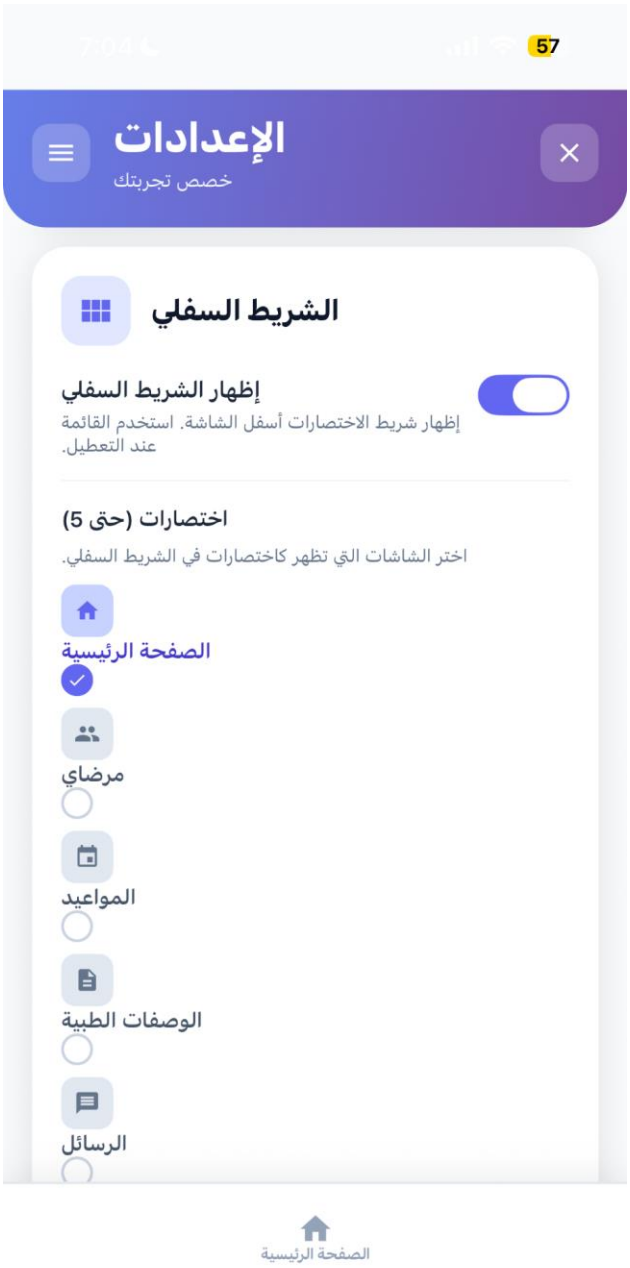

Figure 63 Bottom Navigation Customization

When the bottom shortcut bar is enabled, the user can choose which screens appear as navigation shortcuts (up to five). The interface lists candidate destinations with icons and selection controls, highlighting the currently selected items (e.g., Home). A live preview at the bottom reflects the chosen configuration, ensuring users understand the resulting navigation layout before leaving the settings screen.

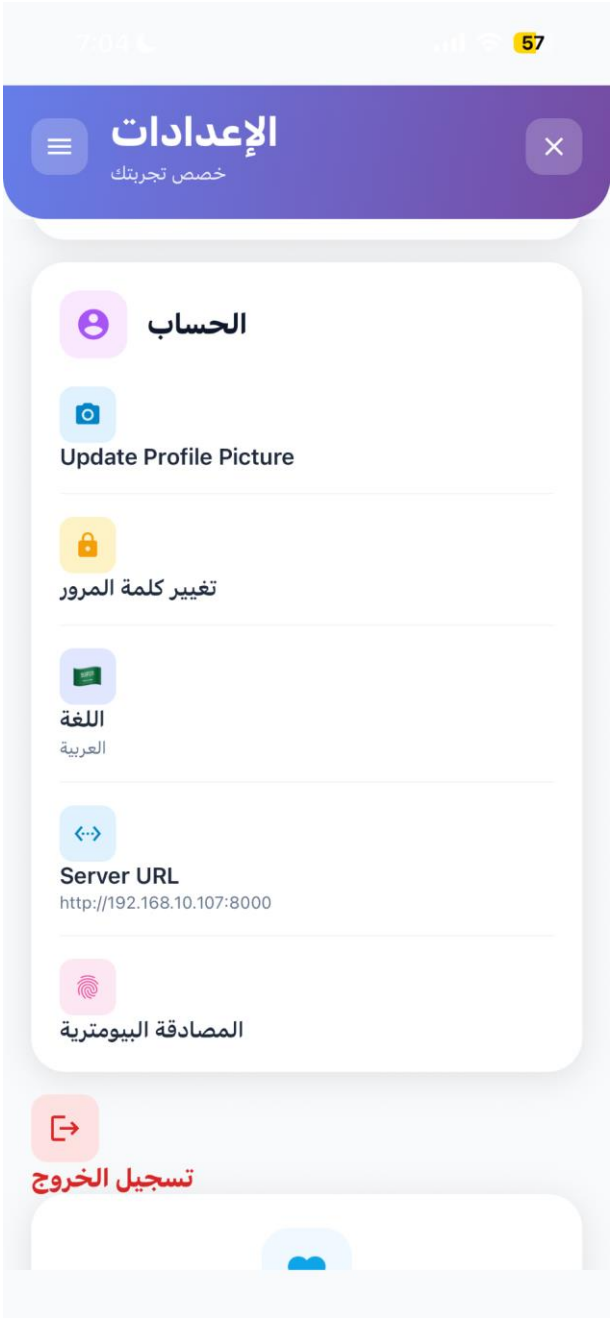

Figure 64 Account Settings

This account section groups core account controls into clear list items: updating the profile picture, changing the password, and selecting the app language. It also exposes a configurable Server URL field for development/testing environments and includes an option for biometric authentication. A distinct sign-out control is placed at the bottom to reduce accidental logout and improve usability.

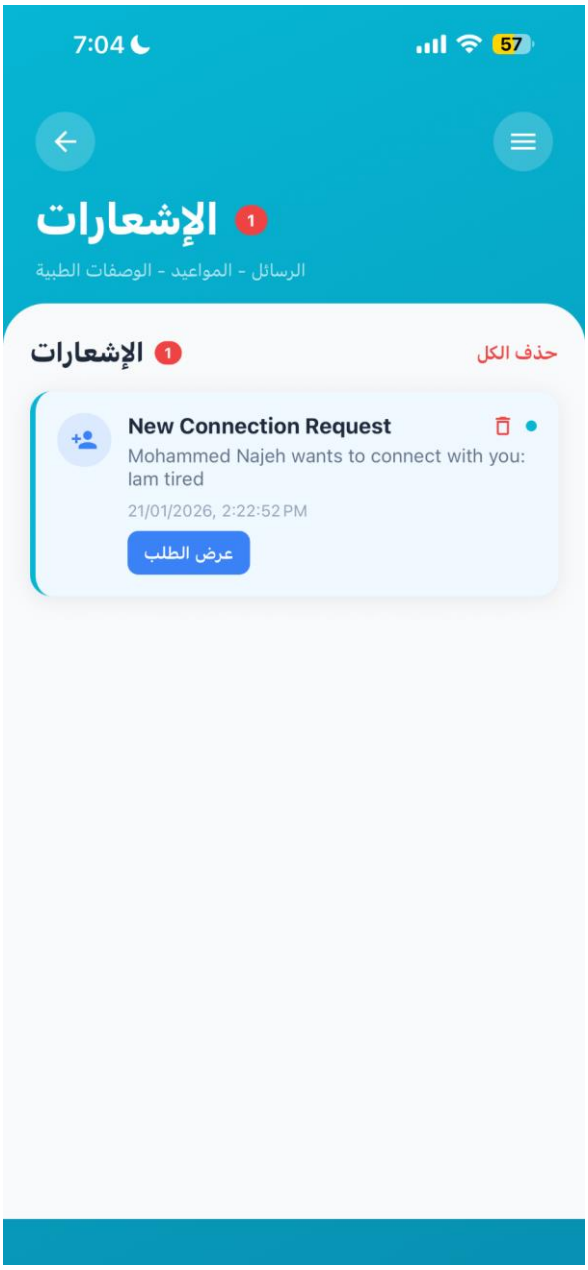

Figure 65 Notifications Feed

The notifications page displays recent events related to messages, appointments, and prescriptions. In this example, a new connection request is shown with a short message snippet, timestamp, and a button to view the request details. Users can delete individual notifications via a trash icon and clear all notifications through a “Delete All” action, while an unread indicator helps distinguish new items.

Figure 66 MedGemma Clinical Assistant

The MedGemma assistant screen (“ميدجما”) provides a structured chat interface for clinical decision support. It prompts the clinician to submit a standardized case summary (age, sex, symptoms, duration, vital signs, medications, allergies, and chronic conditions) and offers one-tap shortcuts for key outputs such as differential diagnosis, recommended tests, and red flags/when to refer. The input area supports voice entry and sending messages, and top-level buttons provide quick access to treatment options and drug–drug interaction checks.

Figure 67 Administrator Dashboard

The administrator dashboard ("لوحة تحكم المسؤول") summarizes system-level KPIs using compact metric cards, including active users (last 30 days), total users, suspended accounts, and new registrations. A feature-usage panel aggregates counts for major modules (lab reports, medications, doctor visits, chats), enabling quick health checks of adoption. Administrative actions provide entry points to user management and analytics/insights.

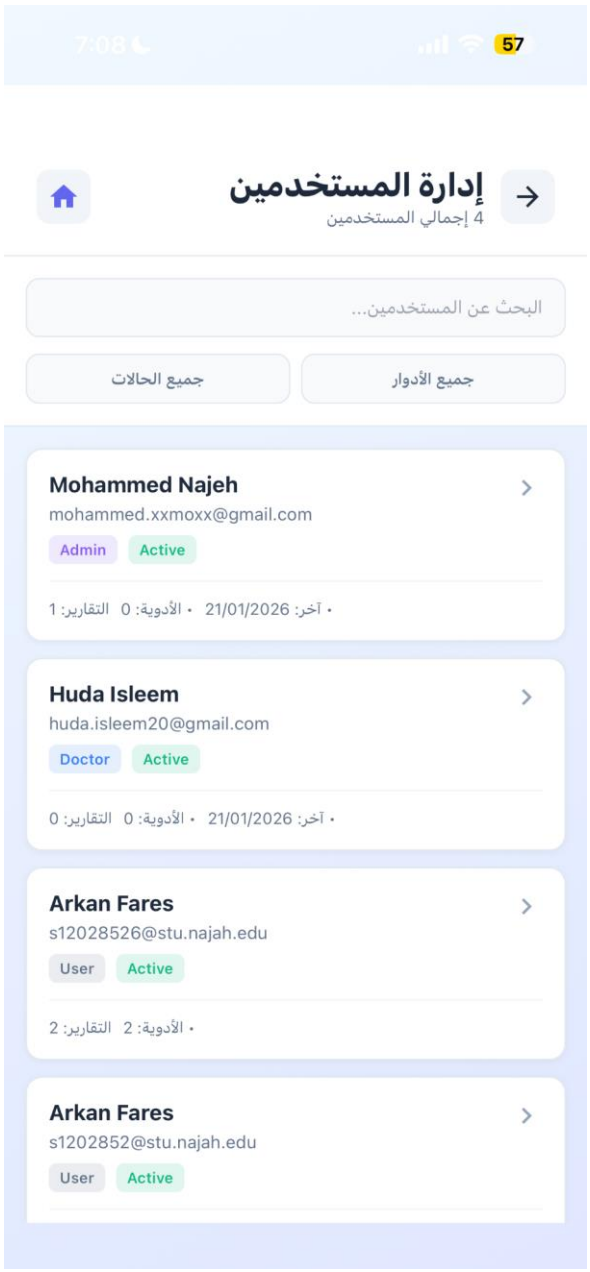

Figure 68 User Management

The user management screen supports operational administration through search and filter controls (by status and role). Each user is presented as a card showing name, email, role badge (e.g., Admin/Doctor/User), account status, and lightweight usage indicators such as counts of medications and reports, along with last activity date. This design enables rapid triage of accounts for support, verification, or auditing tasks.

Figure 69 User Management – Filtered View (Active Doctors)

This view demonstrates role-based filtering in the user management module. With the role filter set to Doctor and status set to Active, the list shows only matching accounts, confirming that the filtering logic and UI state (selected chips) correctly restrict the dataset and simplify administrative review.

UI Screenshots – Batch 8 (Analytics & Insights Dashboard and Connect Doctor Module)

Figure 70 User Management — filtered view (Active Admin accounts)

This screenshot shows the admin user management screen with active status and the Admin role filters applied. The header indicates a single matching user, and the user card surfaces key audit fields (name, email, role badge, status badge, last activity date) to support quick administrative review and account governance.

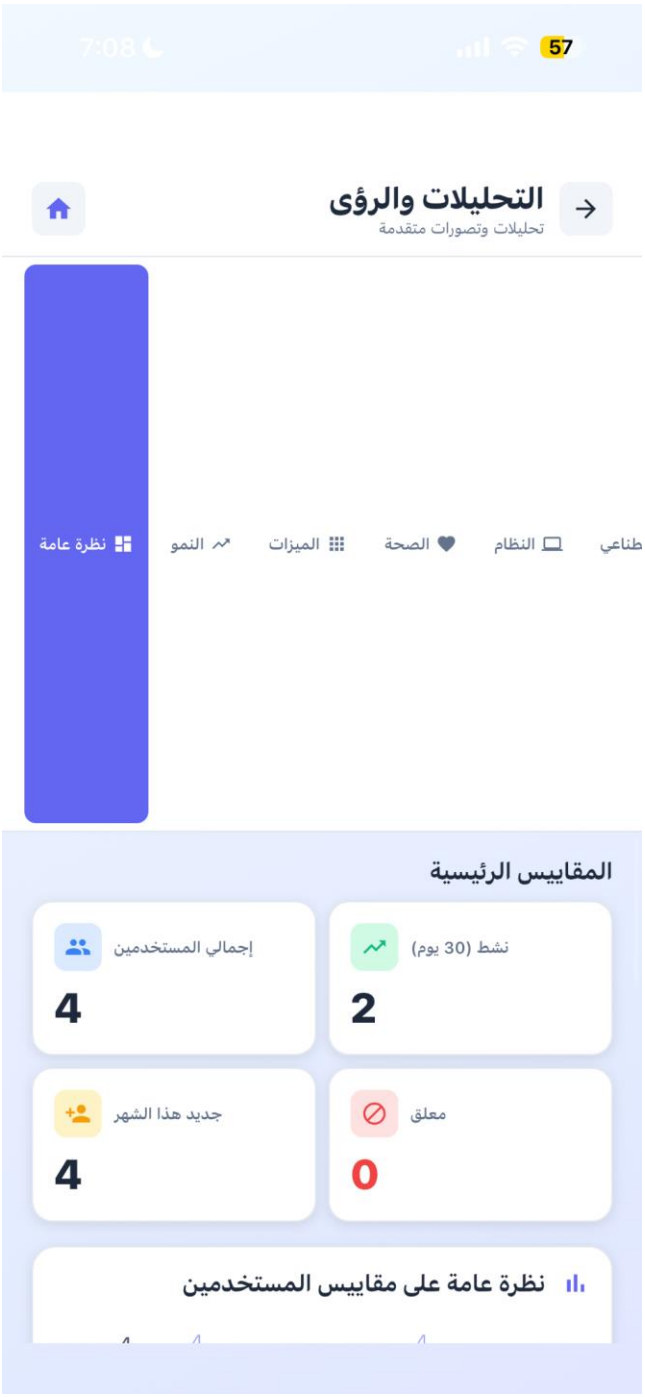

Figure 71 Analytics & Insights — overview dashboard

The overview tab summarizes key user metrics in compact cards: total users, active users over the last 30 days, new users this month, and suspended accounts. A simple visualization beneath the cards provides a quick at-a-glance trend view for the administrator.

Figure 72 Analytics & Insights — growth view (new user trend by date)

The growth tab visualizes account creation over time using a line chart and a corresponding daily breakdown list. This supports monitoring onboarding activity and validating user growth during testing and deployment phases.

Figure 73 Analytics & Insights — feature usage comparison (this week vs last week)

This view compares weekly feature usage for core modules such as Lab Reports, Medications, Doctor Visits, and Conversations. The grouped bars (this week vs last week) help detect spikes or drop-offs and validate that telemetry is being captured per feature.

Figure 74 Analytics & Insights — feature breakdown card

A detailed feature card provides multi-window statistics for a single module (Lab Reports), including this week vs last week counts, this month vs last month counts, and the calculated percentage change. This format supports quick diagnostics of engagement changes.

Figure 75 Analytics & Insights — health demographics (age distribution)

The health tab presents demographic distribution derived from user health profiles. In this example, the age histogram highlights the current dataset distribution, enabling future extensions such as population-level trends and targeted health analytics.

Figure 76 Analytics & Insights — system performance and storage metrics

System-level operational metrics are surfaced for administrators, including API error count over the last 24 hours, average response time, used storage, and database size. The section supports early detection of performance regressions and capacity issues.

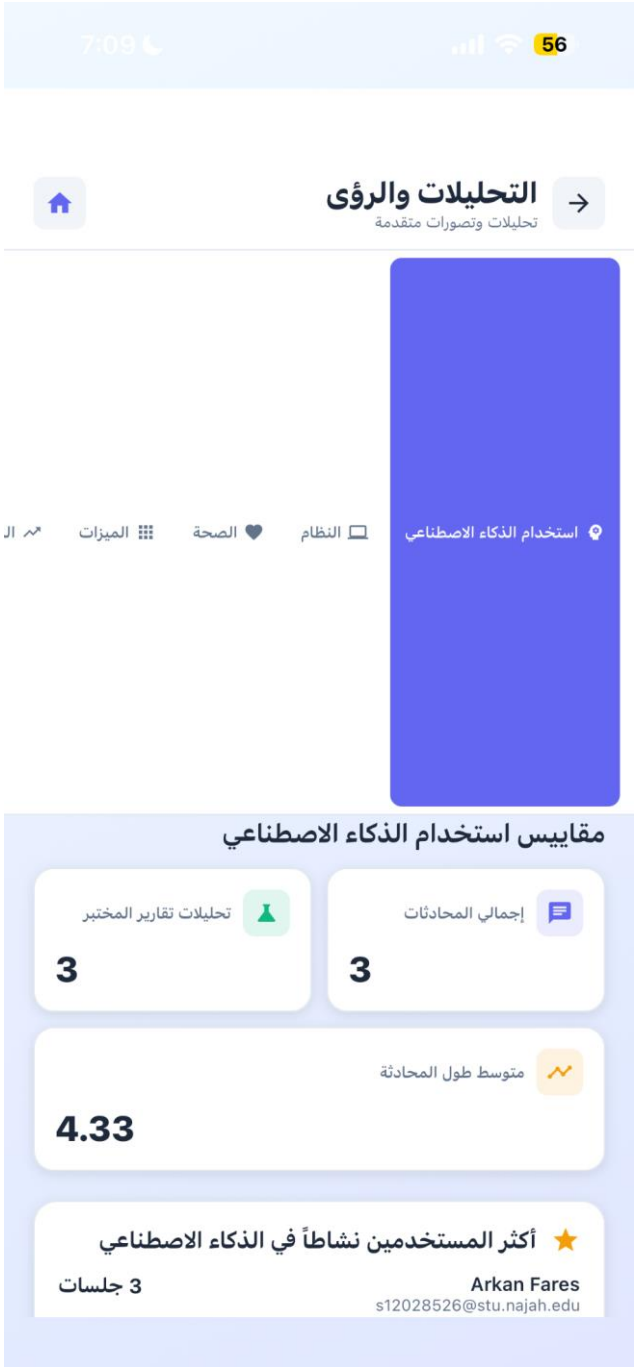

Figure 77 Analytics & Insights — AI usage metrics (lab analyses and conversations)

This tab tracks AI utilization across the platform, including the number of lab report analyses, total AI conversations, and average conversation length. A 'most active user' card helps validate engagement and supports auditing during evaluation.

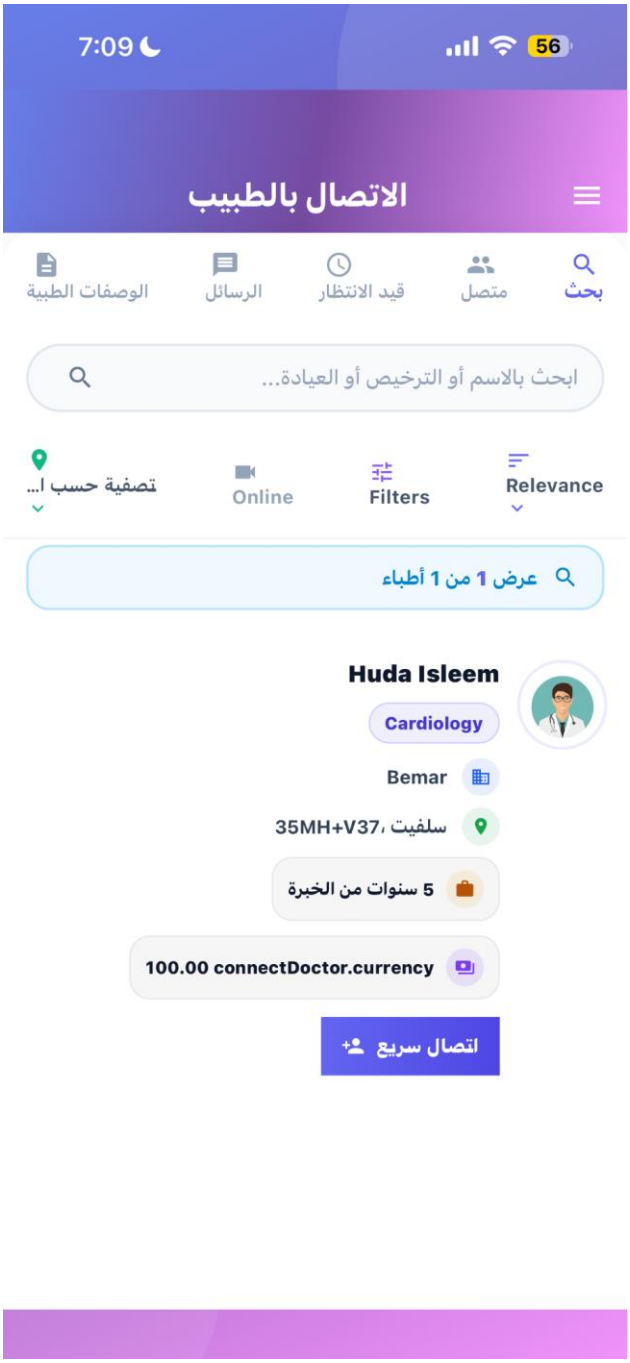

Figure 78 Connect Doctor — patient search and doctor listing

The patient-facing Connect Doctor screen provides search, quick filters, and sorting controls (e.g., city, online status, relevance). Each doctor card displays specialization, clinic name, location, experience, and consultation fee, with a primary 'Quick Connect' action. This screenshot also highlights a localization placeholder issue (currency label) that can be addressed via i18n translation keys.

Figure 79 Connect Doctor — doctor profile preview (modal details)

Selecting a doctor opens a profile preview showing the doctor's name, license number, specialization tags, clinic information (name, address/plus code, phone), and consultation fee. This supports informed patient selection prior to sending a connection request.

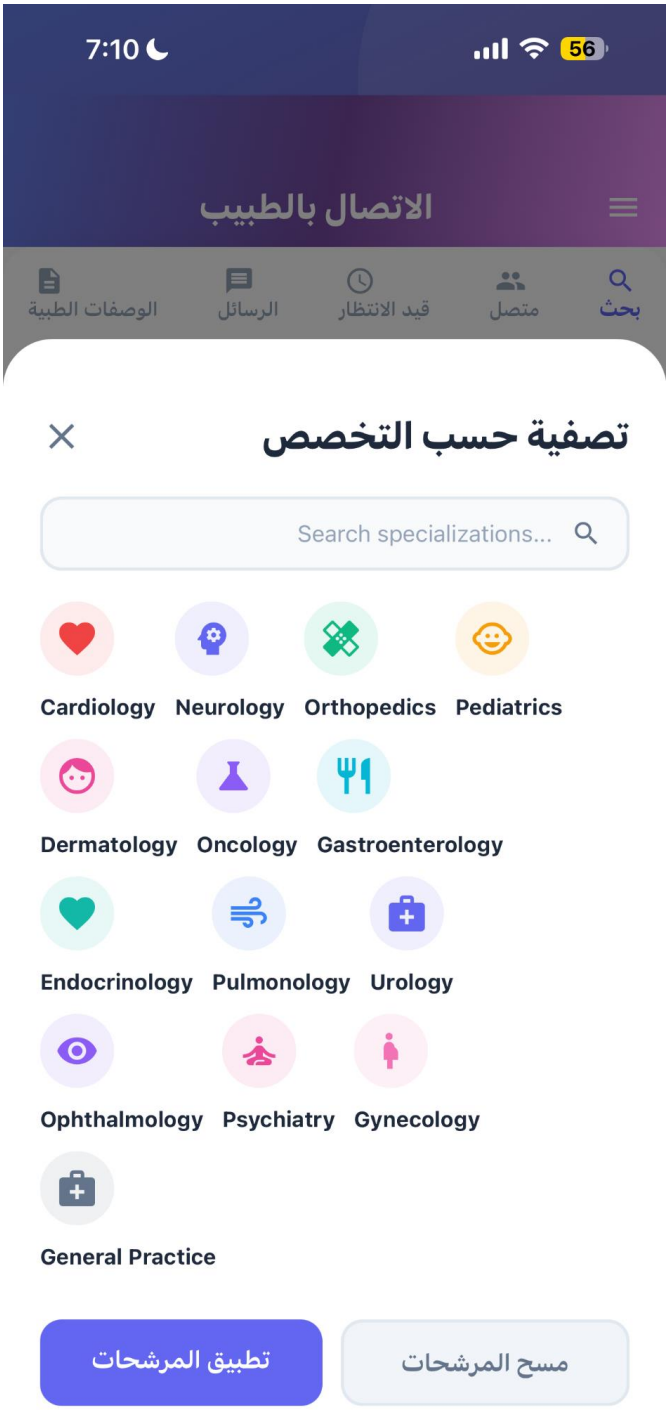

Figure 80 Connect Doctor — specialization filter modal

The specialization filter modal offers searchable medical specialties with icon-based categories, allowing users to narrow results quickly. Action buttons at the bottom enable applying filters or resetting to defaults for rapid exploration.

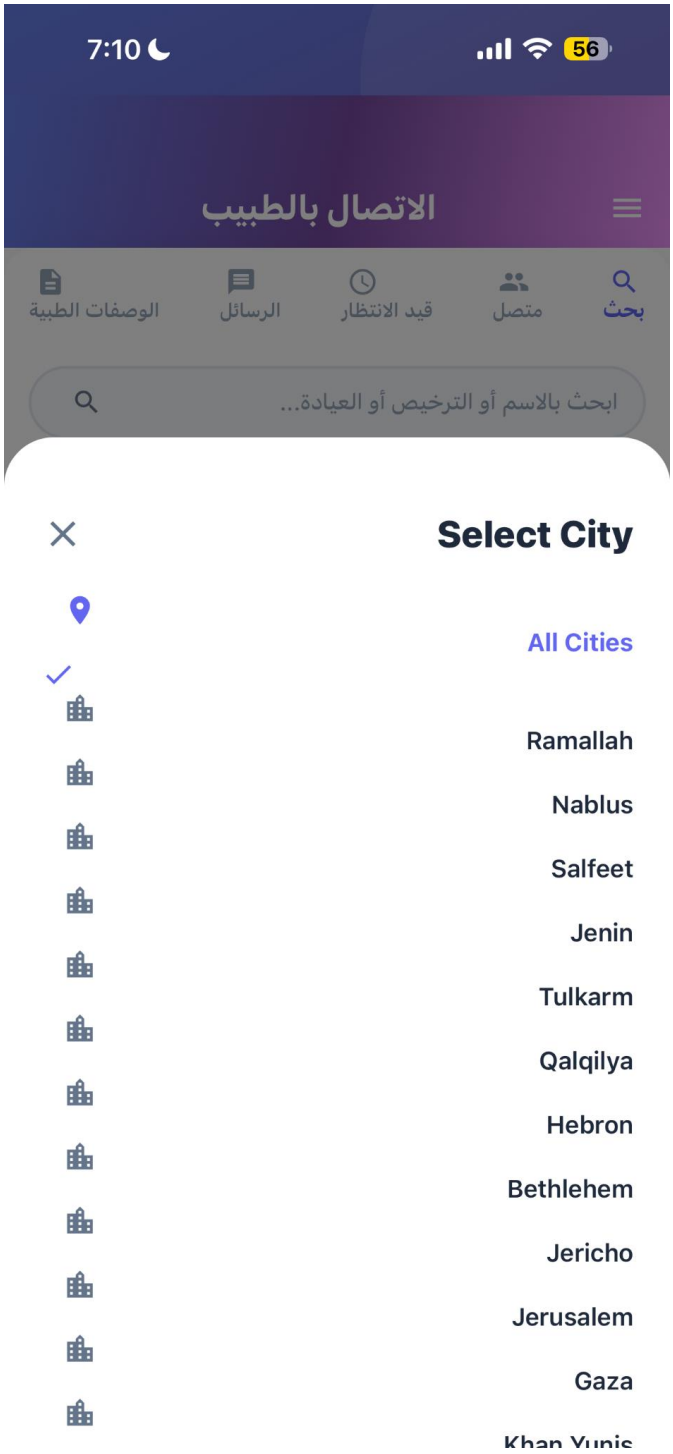

Figure 81 Connect Doctor — city selection modal

The city selector modal allows users to filter available doctors by Palestinian cities (e.g., Ramallah, Nablus, Salfit, Jenin, Tulkarm). An 'All Cities' option supports broad discovery, while the list structure keeps filtering fast and mobile-friendly.

Chapter 6: Conclusions and Recommendations

6.1 Conclusions

Health Tracker delivers a full-stack healthcare management platform that combines secure data storage, medication adherence tools, and AI-assisted understanding of laboratory reports. The architecture demonstrates a maintainable approach to integrating multiple AI providers into a single product while preserving clear boundaries between UI, API, data, and external services.

6.2 Recommendations and Future Work

- Add a FHIR export mode for lab values and summaries to support interoperability with hospital systems.
- Introduce role-based access (e.g., clinician accounts) and consent-based sharing workflows.
- Implement offline-first caching for key screens and queued uploads for unstable connections.
- Add a rules-based validation layer for abnormal lab values to trigger safety recommendations and alerts.
- Expand evaluation with clinician review and user studies to measure comprehension and adherence improvements.

References

Amazon Web Services. (2024). Amazon Simple Storage Service (Amazon S3) Documentation.

Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures (Doctoral dissertation, University of California, Irvine).

HL7 International. (2023). FHIR Release 4 (R4) Specification.

Laravel. (2025). Laravel Documentation (Version 12).

OWASP Foundation. (2021). OWASP Top 10 - The Ten Most Critical Web Application Security Risks.

React Native. (2024). React Native Documentation.