Project Report

on

Medbuddy : Optimal Real-Time Medication Management for the Elderly

Submitted by

Project Members

1032211990 Manvi Singh
1032212077 Atharva Thorat
1032212172 Khilee Singhal

Under the Internal Guidance of

Prof. Laxmi Bhagwat

School of Computer Engineering and Technology
MIT World Peace University, Kothrud,
Pune 411 038, Maharashtra - India
2023-2024

DEPARTMENT OF COMPUTER ENGINEERING AND TECHNOLOGY

C E R T I F I C A T E

This is to certify that,

Manvi Singh (1032211990)
Atharva Thorat (1032212077)
Khilee Singhal(1032212172)

of BTech.(Computer Science & Engineering) have completed their project titled "*MedBuddy: Optimal Real-Time Medication Management for the Elderly*" and have submitted this Capstone Project Report towards fulfillment of the requirement for the Degree-Bachelor of Computer Science & Engineering (BTech-CSE) for the academic year 2024-2025.

[Prof. Laxmi Bhagwat]
Project Guide
School of CET
MIT World Peace University, Pune

[Dr. Balaji M Patil]
Program Coordinator
School of CET
MIT World Peace University, Pune

Internal Examiner:_____

External Examiner:_____

Date: 13-05-2025

Acknowledgement

We take this opportunity to express our heartfelt gratitude to all those who supported and guided us throughout the development of our project, MedBuddy – A Smart Healthcare Companion for the Elderly.

We are deeply thankful to our project guide, Prof. Laxmi Bhagwat, for her continuous guidance, insightful feedback, and unwavering encouragement, which were instrumental in shaping the direction and execution of our work. Her support not only enhanced our understanding but also inspired us to strive for excellence. We also extend our sincere thanks to our program coordinator, Dr. Balaji M. Patil, for providing us with the opportunity, infrastructure, and academic environment necessary to carry out this project successfully.

Our gratitude goes to all the faculty members of the School of Computer Engineering and Technology for fostering a motivating and supportive atmosphere conducive to learning and innovation.We would also like to acknowledge the non-teaching staff of our department for their timely assistance and cooperation during various stages of the project.

Lastly, we owe special thanks to our families and friends for their unwavering moral support, patience, and encouragement throughout this journey. Their belief in us kept us going through all challenges and helped bring our project to fruition.

Name of the Students:

Manvi Singh (1032211990) – PF-43
Atharva Thorat (1032212077) – PF-45
Khilee Singhal (1032212172) – PF-50

Abstract

In today's fast-paced world, ensuring timely and accurate medication adherence is a major challenge, especially for elderly individuals suffering from visual impairments, cognitive decline, or limited technical proficiency. Despite the availability of digital health tools, many fall short in offering accessibility, personalization, and offline support—key requirements for senior citizens. Addressing this critical gap, our project MedBuddy presents a smart, offline-first mobile companion specifically designed to assist elderly patients in managing their daily medications with ease and independence.

MedBuddy integrates a suite of technologies including Text-to-Speech (TTS) and QR code scanning to deliver a multi-modal, intuitive healthcare solution. With QR code scanning, users can quickly identify medicines and retrieve dosage instructions by scanning printed labels, reducing confusion between pills. The TTS module reads out medicine names, dosages, and timings, making the app accessible for visually impaired or illiterate users. The core of the application is its offline-first architecture powered by encrypted SQFLite, which enables users to set and receive timely medication alerts without relying on continuous internet connectivity.

In addition, the app features an elderly-friendly user interface with large fonts, high contrast, and simple navigation, tailored for individuals with visual or cognitive limitations. Users can also track their medication history, missed doses, and receive voice-based alerts for each scheduled dose, thereby reducing the risks of overdosing or non-compliance.

Developed using Flutter and Dart for cross-platform compatibility, and leveraging local notifications, TTS, and QR scanning, MedBuddy combines mobile development and secure data persistence to create a comprehensive healthcare assistant. Through its design and functionality, the application not only promotes safe medication practices but also empowers elderly users to manage their health independently with dignity.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

In the modern era of healthcare, managing medications is a critical aspect of patient care, particularly for elderly individuals and patients with chronic illnesses. Missed doses, incorrect timing, and difficulty reading prescriptions often result in complications, hospitalizations, or reduced treatment effectiveness. Current solutions for medication alerts either depend heavily on internet access or lack features for visual accessibility and ease of use for non-tech-savvy users.

To address this growing need, MedBuddy has been developed—a smart, user-centric mobile application that helps individuals manage their daily medicine intake with minimal effort. The app focuses on accessibility, offline functionality, and intelligent medication tracking. Using QR code scanning, users can extract critical information such as medicine name, dosage, and frequency. This information is securely stored locally using an encrypted SQLite database, ensuring the app works entirely offline without needing a server or internet access.

MedBuddy also incorporates Text-to-Speech (TTS) alerts, an intuitive calendar-based UI, and medication history tracking, making it an ideal solution for elderly users, visually impaired individuals, or those who prefer non-digital alternatives. Built using the Flutter framework, MedBuddy is deployable across both Android and IOS platforms. Its clean interface and robust offline capabilities empower users to take control of their health independently and reliably.

1.1 Project Statement

MedBuddy is a mobile-based solution designed to assist users in managing their daily medication routines efficiently. The application integrates QR code-based scanning, local data storage, and audio-visual alerts to create a self-contained system for medication management. Unlike cloud-dependent platforms, MedBuddy ensures complete offline functionality, thereby eliminating any reliance on constant internet access. It targets elderly users and patients who require a highly accessible, secure, and user-friendly way to track

their medicine schedules and receive timely alerts.

1.2 Aim

The primary aim of the MedBuddy project is to develop a cross-platform mobile application that simplifies and automates medicine intake alerts using a combination of QR code scanning, speech feedback, and offline data storage. The goal is to empower users—especially elderly or visually impaired individuals—with a reliable digital health assistant that enhances medication adherence, reduces risks due to missed doses, and promotes independence in healthcare management.

1.3 Area

This project lies at the intersection of healthcare technology, mobile development, and assistive tools. It demonstrates how speech synthesis (TTS) and barcode scanning can be utilized within a mobile environment to solve real-world problems like medicine adherence. It also emphasizes offline-first architecture using SQLite, making it relevant in areas with limited or no internet connectivity.

1.4 Need for the Project

Many individuals, particularly the elderly, struggle with tracking complex medication schedules due to cognitive, visual, or physical limitations. Traditional paper-based schedules are prone to human error, and existing apps often depend on internet access or are too complex for senior users to navigate. Furthermore, most commercial solutions require manual entry of medicine details—a tedious and error-prone task.

There is a strong need for a solution that is:

- Simple, accessible, and voice-guided.
- Offline-capable and secure.
- Scalable without infrastructure overhead.
- Capable of extracting medicine details automatically through QR scanning.

MedBuddy fulfills all these criteria, making it a much-needed tool in today's digital health ecosystem.

1.5 Implementation Overview

MedBuddy is implemented using the Flutter framework for responsive cross-platform UI. It integrates:

- QR code scanning to capture medicine name, dosage, and frequency.
- Encrypted SQLite database for local offline storage of medicine data, schedules, and history.
- Flutter TTS for voice-based medicine alerts.
- Calendar UI for intuitive visual tracking of medication logs.
- Local notifications for scheduled alerts.

The app ensures secure offline functionality, eliminating the need for cloud services. Deployment involves publishing the app on Google Play Store and Apple App Store, making it accessible to a wide audience. All core features work entirely without an internet connection.

1.6 Application of the Project

MedBuddy has broad applicability across various healthcare scenarios, making it a versatile tool for medication management:

- **Elderly and Visually Impaired Individuals:** Designed with high-contrast themes, voice alerts, and minimal-touch UI, the app ensures ease of use and timely medication alerts for users with cognitive or visual impairments.

- **Chronic Illness Patients:** Those managing conditions like diabetes or hypertension can log and monitor their intake via the calendar view and history logs, promoting better adherence.

- **Remote and Rural Areas:** Its fully offline functionality makes MedBuddy ideal for users in areas with limited connectivity, ensuring they never miss a dose due to technical issues.

- **Caregiver Support:** Family members or caregivers can initially configure the app, which then works autonomously, providing both visual and audio cues for the patient.

- **Scalable Public Health Solutions:** MedBuddy's lightweight, modular design enables it to be adapted for use in clinics, NGOs, or government health programs to distribute and track medication in underserved areas.

Chapter 2
Literature Survey

1. Comprehensive Review of Pill Image Recognition

In this journal paper (2025), the authors present a detailed survey of the latest advancements in computer vision and deep learning techniques for pill detection and classification. The paper highlights the shift from traditional, handcrafted feature extraction methods to more sophisticated convolutional neural network (CNN) architectures. It emphasizes the significance of accurate image-based medication identification in preventing dosage errors, offering a technically rich perspective on model performances and datasets. However, the study does not address accessibility features, especially for elderly users, such as voice-guided interaction or simplified interfaces. Additionally, the integration of pill recognition technology into broader medication management workflows, like alerts or adherence tracking, is not discussed, leaving room for further exploration.

2. Medication Adherence in the Elderly: A Comprehensive Review

This 2024 journal article investigates various barriers to medication adherence among elderly populations. The study explores issues like cognitive decline, social isolation, and forgetfulness while providing a foundational framework for designing effective interventions. The positive aspects of this review include a multi-dimensional approach to understanding patient behavior and the development of alerts-based solutions. However, it lacks insights into emerging digital tools, such as mobile applications and wearable devices, which are increasingly important in modern healthcare. Furthermore, real-time feedback systems and personalized scheduling options are not sufficiently explored.

3. Smart Medication Management: Enhancing Medication Adherence with an IoT-Based Pill Dispenser and Smart Cup

Presented at a 2024 conference, this paper introduces a smart medication management system using IoT technology, integrating a pill dispenser with a smart cup. The solution aims to streamline medication intake and improve adherence among users. The system provides a user-friendly interface and automation to minimize human error. Despite its innovation, the study fails to address the challenges of deploying such solutions in diverse healthcare settings

with varying infrastructure and user demographics. The absence of a cost-benefit analysis and scalability insights also limits its applicability.

4. Developing an Artificial Intelligence-Driven Nudge Intervention to Improve Medication Adherence

Sumner J. (2023) presents a novel AI-based nudge intervention developed with stakeholder collaboration. The intervention is designed to positively influence patient behavior using digital cues. A key strength of the study is its participatory design approach, aligning technological development with real-world needs. However, the paper only covers the initial development and pilot stages. Long-term efficacy, user retention, and behavioral adaptation over time are not addressed, making it difficult to assess the intervention's sustained impact.

5. Older Adults' Intention to Use Voice Assistants

This 2023 journal study explores the cognitive and psychological factors influencing older adults' willingness to use voice assistants. The findings offer valuable implications for incorporating text-to-speech (TTS) technologies into healthcare settings, especially for the elderly. While it provides an in-depth analysis of user intent and technology perception, the study does not explore real-time use cases like medication alerts or emergency support. Moreover, practical integration into healthcare systems and the potential for enhancing medication adherence is not considered.

6. Design and Implementation of a Smart Pill Alert System for Elderly Patients

In this 2021 conference paper, A. Kumar and S. Patel describe a smart pill alert system combining IoT and mobile technologies. The system offers features like real-time alerts and senior-friendly UI, targeting improved adherence in elderly patients. While the system shows promise, it lacks integration with voice-assisted technologies, which could further enhance usability for visually impaired or less tech-savvy users. Additionally, there is no discussion on adaptability to various languages or regional contexts

Chapter 3

Problem Statement

3.1 Project Scope

The MedBuddy project aims to simplify and automate medication schedule management and tracking for patients, especially the elderly and those with chronic conditions. It is a mobile application that integrates QR code scanning for pill identification, SQLite-based offline medication storage, and alert notifications. The application offers a fully offline experience, allowing users to manage their medications without internet dependency.

MedBuddy leverages a local SQLite database to store user, medicine, and dosage data securely on the device. It allows users to input their medication schedule through a user-friendly form, and stores all related information locally for offline access. The system provides timely medication alerts through notifications and offers a clean interface designed for elderly accessibility.

The primary goal is to offer a secure, lightweight, offline-first mobile companion that helps improve medication adherence and supports independent health management for users. While the current version does not support real-time backend integration or direct healthcare provider access, it lays a foundation for future expansion with features like syncing or cloud backup if required.

3.2 Project Assumptions

- Users will have basic familiarity with using mobile applications.
- The application will run efficiently on Android (as seen in the current implementation).
- Users will actively engage with alerts and track their medications as per the app's functionality.
- The device used will support local notifications and SQLite operations.

- Offline access is sufficient for most users; internet access is not a requirement for day-to-day use.

3.3 Project Limitations

While MedBuddy is designed with scalability, simplicity, and privacy at its core, certain design decisions naturally influence how the system operates in real-world scenarios. For instance, the use of a local SQLite database ensures user privacy and offline reliability but inherently defines the app's scope around single-device usage. This offers excellent performance and security for individual users but may limit scenarios where centralized access or long-term data aggregation is required.

The application's current architecture, optimized for offline functionality, allows seamless operation without constant connectivity. However, as the feature set expands in the future—such as potential cloud sync, provider dashboards, or integration with pharmacy systems—thoughtful enhancements will be considered to ensure continuity in performance and usability.

MedBuddy's design prioritizes simplicity and accessibility, especially for elderly users. While the interface has been tailored for ease of use, further iterations may explore adaptive UI or guided onboarding to support users with minimal digital literacy. Cross-platform compatibility is maintained through Flutter, but device-level differences may occasionally require minor UI or performance adjustments across various models.

The application's current focus on privacy through on-device data management means that users retain full control over their health information. As the app evolves, additional layers of security and optional encrypted backup features could be introduced to further enhance data resilience.

3.4 Project Objectives

1. **Improve Medication Adherence**: By providing patients with an intuitive, automated alert system for their medications, the project aims to improve adherence to prescribed dosages, reducing missed doses and ultimately enhancing patient health outcomes.

2. **Enhance Data Privacy and Security**: By utilizing offline storage on users' devices, MedBuddy ensures that sensitive patient data remains private and secure, without the need for cloud-based storage that may expose personal health information to potential security risks.

3. **Increase User Engagement**: The app is designed to encourage active engagement by allowing patients to track their medication progress, and share medication history with healthcare professionals.

4. **Facilitate Easy Medication Tracking**: MedBuddy aims to provide a seamless way for users to track their medication schedules and history, allowing for quick access to important health information at any time.

5. **Support Healthcare Professionals**: MedBuddy provides healthcare providers with an efficient platform for monitoring patient medication adherence and offering personalized support.

One of the most important advantages of the MedBuddy system is its ability to improve medication adherence, a critical factor in ensuring patient health outcomes. Additionally, MedBuddy ensures data privacy and security by utilizing a decentralized storage model where all sensitive data remains on the user's device. This eliminates the risk of data breaches that often accompany cloud storage solutions.

The MedBuddy app also aims to enhance user engagement by incorporating a rewards system for patients who consistently follow their medication schedules. This system will motivate users to remain consistent in taking their prescribed medications, ensuring long-term health benefits.

By enabling healthcare providers to easily monitor their patients' medication history, MedBuddy supports the healthcare ecosystem by promoting better patient-provider communication and ensuring that healthcare professionals can make informed decisions about patient care.

Chapter 4

Project Requirements

4.1 Resources

4.1.1 Human Resources

- **Project Manager**

  Oversees the end-to-end development of MedBuddy, manages timelines, allocates responsibilities, and ensures the project aligns with user needs and offline-first goals.

- **Flutter Developer**

  Develops the cross-platform mobile app using Flutter. Responsible for implementing features like QR scanning, OCR integration, and local database operations.

- **UI/UX Designer**

  Designs an intuitive and accessible interface for patients, with a focus on elderly or visually impaired users. Ensures a user-friendly medication management experience.

- **Database Engineer**

  Handles the design and management of the local SQLite database, ensuring fast queries and data integrity for prescriptions, schedules, and medication records.

- **Quality Assurance Engineer**

  Tests the app thoroughly for functionality, edge cases, performance, and usability. Special focus is placed on offline reliability, accurate OCR output, and QR functionality.

4.1.2 Reusable Software Components

- **Google ML Kit (OCR)**

  Used for on-device text recognition from prescription images. Allows the app to

extract medicine names and dosages without requiring network access.

- **Flutter TTS (Text-to-Speech)**
  Enables voice-based alerts and reading out medication information to assist visually impaired users.

- **Flutter Barcode Scanner**
  Allows scanning of QR codes containing medicine or prescription data, improving ease of access and reducing manual entry.

- **SQFLite**
  Lightweight local database used to store medication records, dosage times, and alerts. Ensures offline access and quick local data operations.

4.1.3 Software Requirements

- **Flutter SDK**
  Used for developing the mobile app on Android and iOS platforms.

- **Dart Programming Language**
  The primary language used in Flutter for implementing app logic, UI, and database integration.

- **SQFLite Plugin for Flutter**
  Enables local storage and data retrieval for medicine data, and user profiles.

- **Google ML Kit Plugin**
  Used to implement OCR functionality for extracting text from prescription images.

- **VS Code**
  Development environments used to build and test the application.

4.1.4 Hardware Requirements

- **Development Machine**

  A PC/laptop with at least 8 GB RAM and a modern processor (Intel i5/Ryzen 5 or higher) for smooth Flutter development and emulation.

- **Android/iOS Test Devices OR Emulators**

  Smartphones used to test app performance, compatibility, and UI across different screen sizes and operating systems.

4.2 Requirements Rationale

Table 4.1 Requirements Rationale

Requirement	Rationale
Flutter Framework	Enables fast cross-platform development for Android and iOS from one codebase.
Google ML Kit OCR	Provides accurate offline text extraction from prescription images.
SQLite Database	Offers fast and lightweight offline data storage.
QR Code Scanning	Simplifies user interaction by enabling quick retrieval of medicine data.
Flutter TTS	Enhances accessibility for visually impaired or elderly users.

4.3 Risk Management

Table 4.2 Risk Management

Risk Factor	Description	Risk Level
OCR Inaccuracy	Poor quality images may result in incorrect extraction of medicine names.	High
Device Dependency	Local-only storage limits data recovery in case of device failure or uninstalls.	Medium
User Interface	Non-tech-savvy users might find navigation and interactions complex.	Medium
TTS Accessibility	TTS performance may vary depending on device capability or settings.	Low
Cross-Platform Bugs	Ensuring parity between Android and iOS features might introduce platform bugs.	Medium

4.4 Functional Specifications

4.4.1 Interfaces

- **App Interfaces (UI):**
    - QR Scan Page
    - OCR Capture Interface
    - Add/Edit Medicine Form
    - History/Logs Page

- **Internal Interfaces:**
    - OCR to SQLite Integration
    - Flutter App Logic to Local Database Sync
- **Plugin Interfaces:**
    - Google ML Kit for OCR
    - Flutter TTS for voice output
    - Flutter Barcode Scanner for QR reading

4.4.2 Interactions

- **User Flow:**
    1. User scans a QR code or uploads a prescription.
    2. OCR extracts medicine details, stored in SQLite.
    3. User sets an alert for medication timings.
    4. App triggers notifications with TTS at scheduled times.
    5. Users can edit or view their medication history offline.

4.4.3 Sustainability

MedBuddy is built with a modular architecture using Flutter, which allows seamless addition of future capabilities like cloud sync, healthcare provider integration, or multi-device access. Offline-first design ensures consistent performance without reliance on connectivity.

4.4.4 Quality Management

- **Automated Testing** for unit-level functions like SQLite queries and OCR accuracy.
- **User Testing** to validate usability and performance across Android and iOS.
- **Accessibility Testing** to ensure TTS and visual elements work well for impaired users.

4.4.5 Security

- **Data Storage Control**
  All data is stored locally on the user's device, reducing network exposure.
- **Permission Management**
  App requests only necessary permissions for camera, storage, and notifications.
- **App-Level Security**
  Features such as app PIN lock and limited data access help enhance privacy.

Chapter 5

SDLC Phases and Time Allocation

5.1 Design Considerations

Design considerations ensured accessibility, performance, and offline usability to cater to elderly users in both urban and rural contexts.

5.1.1 Accessibility-First Design

- Built with Flutter's UI components optimized for large fonts and minimal on-screen interaction.
- **Text-to-speech (TTS)** was implemented using the `flutter_tts` package to support users with visual impairments.
- Haptic and sound feedback were included where feasible.
- The layout avoids clutter and uses contrast-rich themes.

5.1.2 Offline-First Architecture

- **Local storage only** using sqflite (for medication, schedule, and history) and shared_preferences (for user flags/settings).
- No cloud sync or backend APIs to ensure independence from internet access.
- Ensures reliability in low-bandwidth or remote areas.

5.1.3 Multi-Modal Input Support

- QR code scanning powered by barcode_scan2.

- OCR via google_mlkit_text_recognition (though partially implemented) allows pill identification using packaging text.

- Manual medication entry and editing are also supported for flexibility.

5.1.4 Modular Architecture

- Project uses Flutter with Provider for state management, maintaining separation between UI and logic.

- Core classes like Medication, DoseHistory, QRCodeHandler, and EmergencyContact are modular and well-encapsulated.

- Navigation is managed through a central routing system.

5.1.5 Reminder Engine

- Dose alerts are triggered using flutter_local_notifications, based on data stored in SQLite.

- Includes tracking of missed doses and invalid dose attempts.

- Alert service works even when the app is closed (subject to Android system constraints).

5.2 Assumptions and Dependencies

The following assumptions and external/internal dependencies were considered during the project development phase:

5.2.1 User Assumptions

- Users or their caregivers can operate a smartphone with basic literacy.

- Permissions for camera, notifications, and storage are granted on install.

- Medications either include a QR code or legible printed labels.

5.2.2 Technical Assumptions

- Android (API 21+) is the target platform; no iOS support included in this phase.
- MLKit can recognize common fonts and labels under average lighting conditions.
- SQLite will store up to a reasonable number of records without performance degradation.

5.2.3 External Dependencies

- **Flutter SDK & Dart** – Core development framework and language for cross-platform mobile app.
- **sqLite** – For storing and managing medication, history, and schedule data in a local SQLite database.
- **shared_preferences** – To save lightweight user settings like onboarding status and role selection.
- **flutter_tts** – Enables text-to-speech for visually impaired users to hear medicine names and instructions.
- **barcode_scan2** – Allows scanning QR codes for identifying medicines quickly.
- **google_mlkit_text_recognition** – Supports OCR functionality to recognize text from pill labels (partially implemented).

5.2.4 Operational Dependencies

- Local notification service depends on OS-level background execution.
- TTS depends on the device's available TTS engine and audio hardware.

5.3 General Constraints

Several constraints were identified and addressed during the design and development of MedBuddy.

5.3.1 Hardware Constraints

- Devices with low-spec cameras may yield poor OCR results.
- Battery optimization on some devices may suppress local notifications.
- Older Android versions may need explicit permission handling.

5.3.2 Software Constraints

- No cloud integration to preserve offline-first requirement.
- QR-based features assume good lighting and camera focus for accuracy.

5.3.3 User Constraints

- Elderly users may struggle with fine motor control, affecting camera alignment or timing of actions.
- Accidental dismissals of alerts can affect medication adherence.

5.3.4 Legal and Ethical Constraints

- All data is stored locally, avoiding privacy violations.
- flutter_secure_storage was considered for sensitive fields but not required due to lack of cloud sync or account systems.

5.3.5 Environmental Constraints

- The app is designed to work offline during travel or in rural areas.

- Efficient in terms of processing and battery usage to support older devices.

5.4 Block Diagram

A block diagram provides a graphical overview of the MedBuddy system, showing its major components (blocks) and the information flows (connections) between them. Blocks represent each core subsystem, and arrows illustrate how data or control signals pass between modules.

Key Features of the MedBuddy Block Diagram:

1. **Blocks:** Each block denotes a major subsystem or module: Authentication, Medication Management, Schedule Service, History Recorder, Emergency Contacts, Pill Identification.

2. **Connections / Arrows:** Directed arrows show data flows such as user credentials, medication data, schedule entries, scan results, and alerts.

3. **Labels:** Blocks and arrows are labeled (e.g., “medData,” “scanResult,” “alertTrigger”) to clarify their functions.

4. **Hierarchy:** This high-level diagram abstracts each module; lower-level block diagrams or DFDs will break these modules down further.

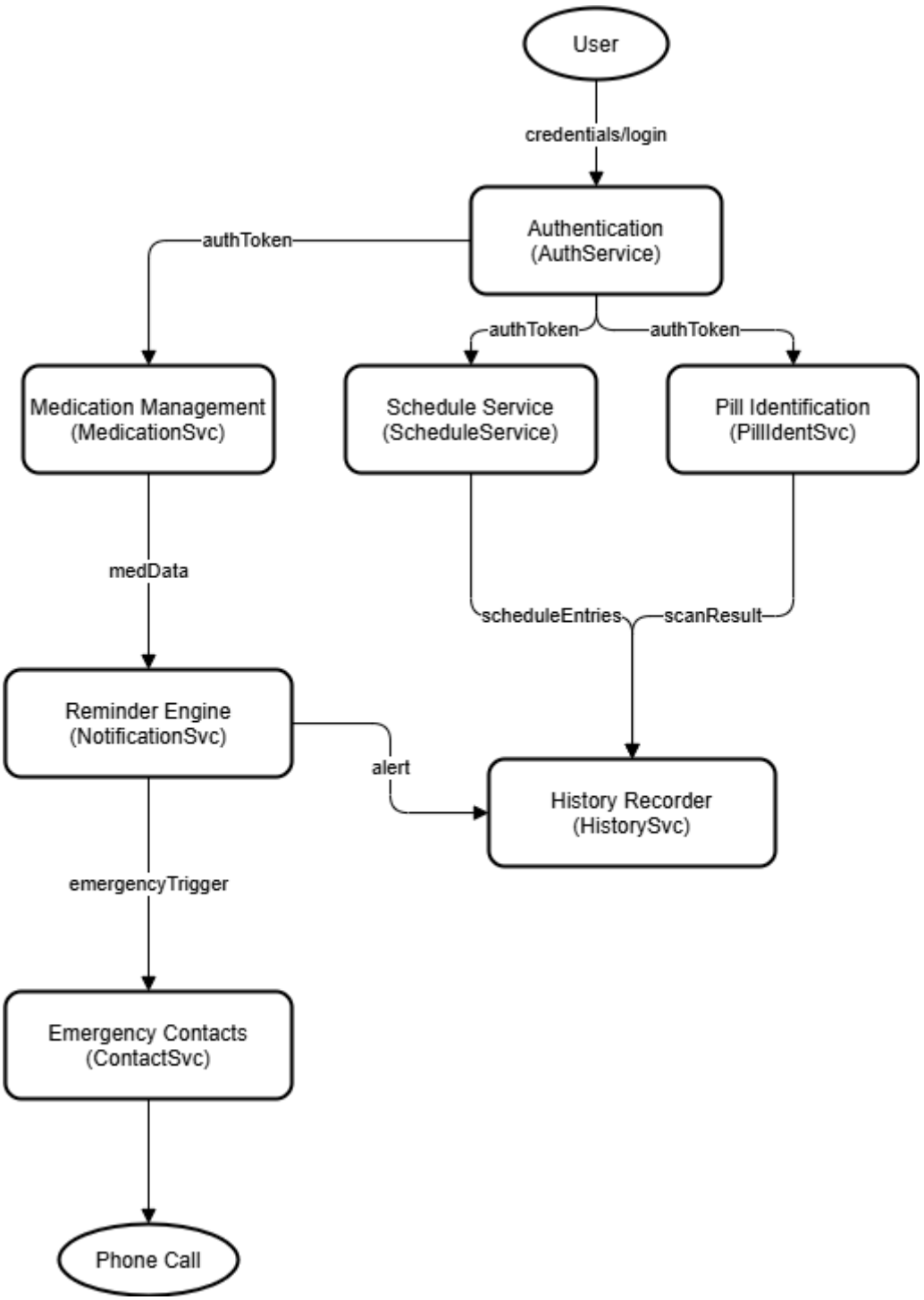

Fig. 5.1 Block Diagram

5.5 System Architecture

The MedBuddy system employs a three-layer architecture depicted in the System Architecture Diagram:

- **Presentation Layer (UI):**
  - **Screens:** Login/Register(If needed), Dashboard, Medication, Schedule,

History, Emergency Contact, Pill Scanner.

- **Tech Stack:** native QR Scanner SDK, local push notification API.

- **Business Logic Layer:**
  - **Services:**
    - AuthService (handles registration, login, token management)
    - MedicationService (CRUD operations, QR code generation)
    - ScheduleService (manages dosage times)
    - NotificationService (local push and voice alerts)
    - HistoryService (records taken/skipped doses)
    - ContactService (stores and triggers emergency calls)
    - PillIdentifierService (scans and validates pill data)

- **Data Layer:**
  - **Data Models:** MedicationModel, ScheduleModel, HistoryModel, ContactModel, PillModel.
  - **Storage:** Encrypted SQLite on-device database.

Actors interacting with the system include:

- **Patient:** Uses the UI to view medications, scan pills, mark doses, and call contacts.
- **Caretaker:** Uses a caretaker UI to add medicines, generate QR codes, view history.

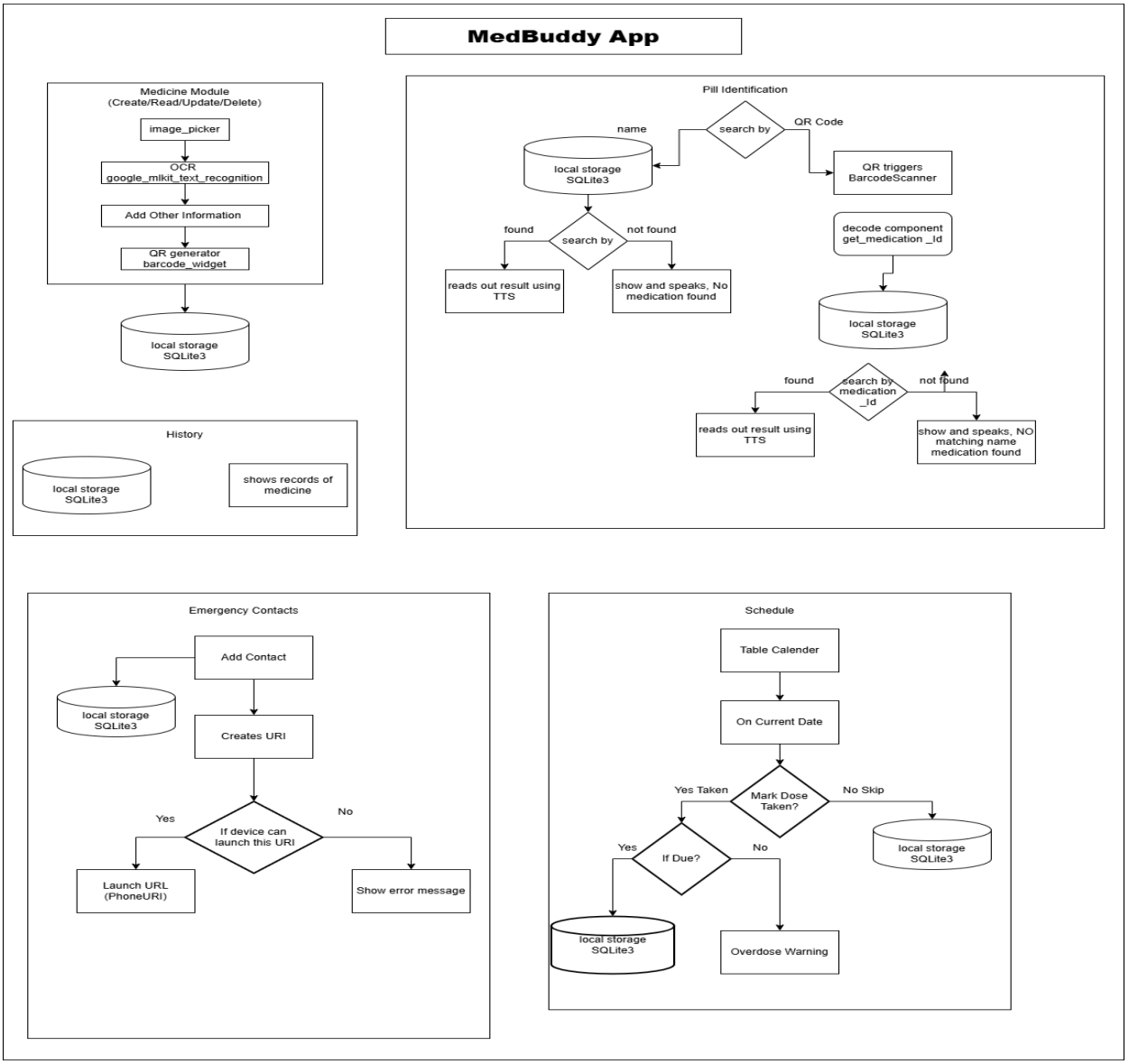

Fig 5.2 System Architecture Diagram

5.6 Modules of the Project

The MedBuddy application is organized into seven distinct modules, each encapsulating specific functionality:

Module 1: Authentication

- Manages user fingerprint.

Module 2: Manage Medicines (Add / View / Edit / Delete)

- Allows caretakers to enter medication details (name, dosage, frequency, images).
- Generates QR codes for each medicine.

Module 3: Manage Medicine History

- Records timestamps and statuses (taken, skipped) for each dose.
- Provides a history view for Patient and Caretaker.

Module 4: Trigger OverDose Alerts

- Extends to voice-based alerts via Text-to-Speech.

Module 6: Handle Duplicate Medicines

- Detects overlapping prescriptions.
- Warns users of potential conflicts.

Module 7: Pill Identification

- Scans medicine QR codes.
- Displays pill details (dosage, schedule).

- Validates correct timing and triggers overdose/wrong-time alerts.

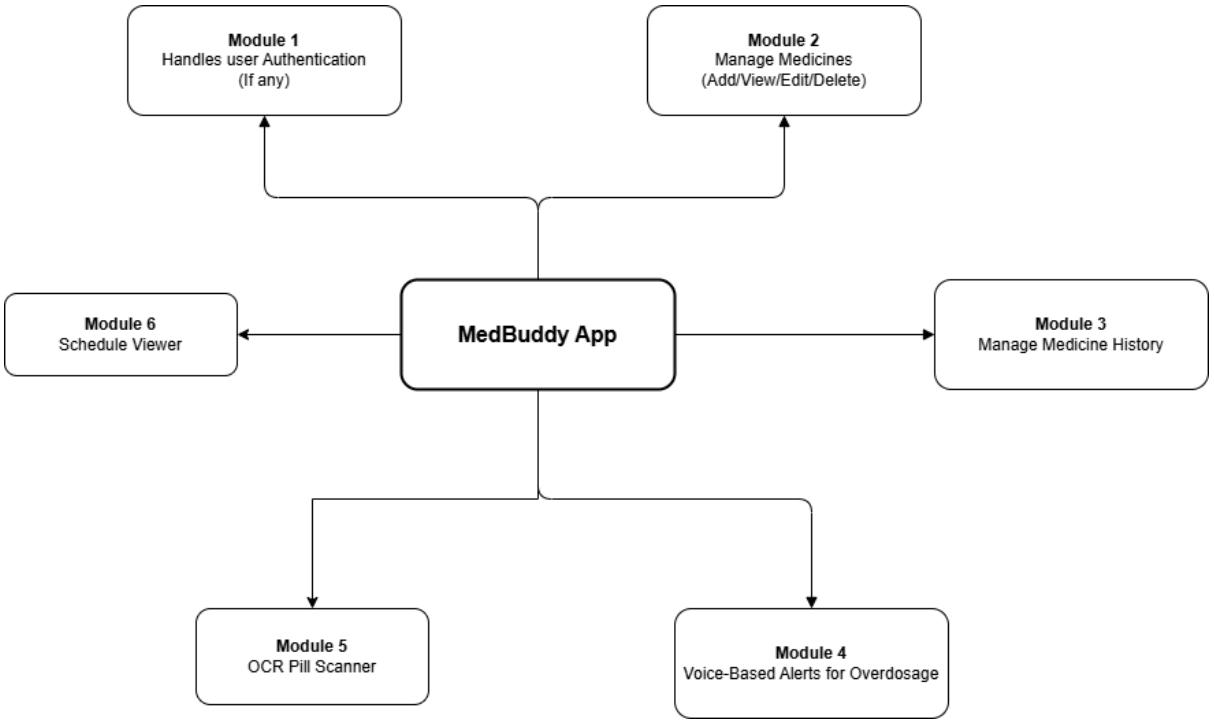

Fig. 5.3 Module Diagram

5.7 Low-level Design

Low-level design describes how data flows through the MedBuddy system's internal processes and data stores. The Level 1 Data Flow Diagram (DFD) provides an intermediate level of abstraction:

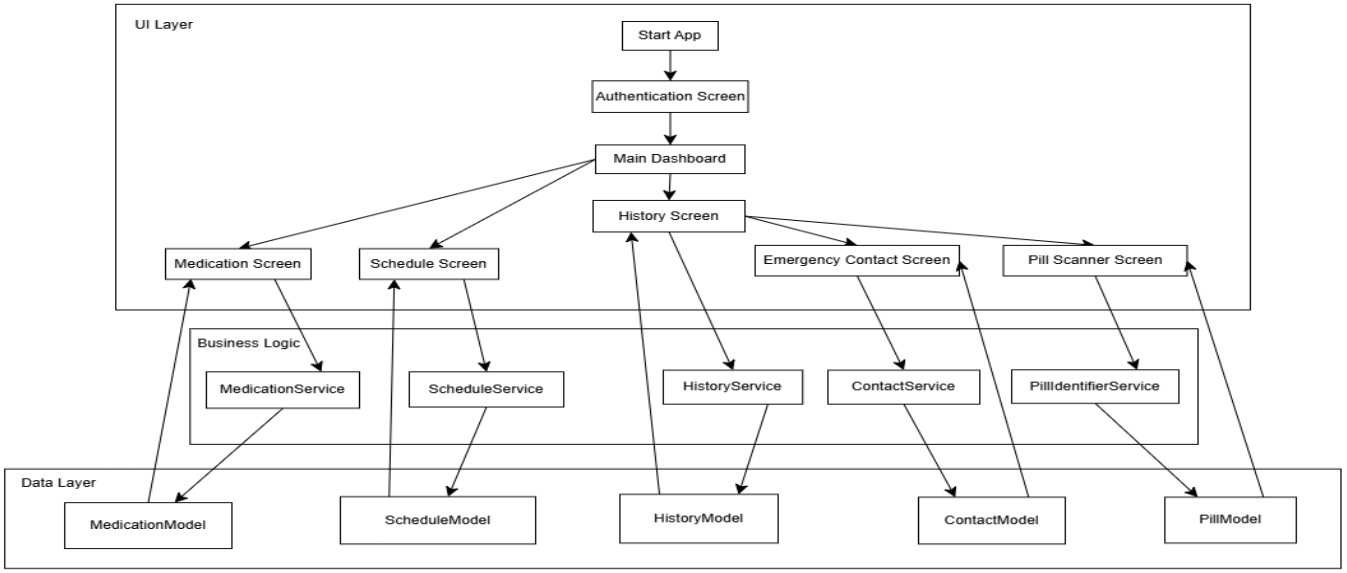

Fig. 5.4 Low Level Design

5.7.1 Structure of DFD Level 1

- **Processes:**
    - P1: Authenticate User
    - P2: Manage Medications
    - P3: Manage Schedule
    - P4: Record Dose History
    - P5: Manage Emergency Contacts
    - P6: Identify Pill

    - P7: Trigger Alerts

- **Data Flows:**
    - Credentials → P1 (AuthService) → AuthToken → UI
    - MedData ↔ P2 ↔ MedicationStore
    - ScheduleData ↔ P3 ↔ ScheduleStore
    - DoseEvent → P4 → HistoryStore
    - ContactInfo ↔ P5 ↔ ContactStore
    - ScanResult → P6 → MedicationStore → P6 → UI

- **Data Stores:**
    - D1: MedicationStore
    - D2: ScheduleStore
    - D3: HistoryStore
    - D4: ContactStore
    -

- **External Entities:**
    - **Patient** (initiates all processes)

- **Caretaker** (manages meds & contacts)

5.8 UML Diagrams

The UML suite captures various views of MedBuddy's dynamic and static behaviors.

5.8.1 Activity Diagram

This diagram (Figure X) models the "Take Medication" workflow:

1. Patient logs in → Dashboard.
2. Patient selects "Pill Scanner" → scans QR code.
3. System checks current time vs. scheduled time.
    - **Decision:** If on-time → prompt "Mark as Taken?", record dose → return to Dashboard.
    - **Else:** show "Wrong Time" alert → allow user to reschedule.

5.8.2 Sequence Diagram

The "Schedule Reminder Setup" sequence (Figure X) shows:

1. **UI:** invokes ScheduleService.addReminder().
2. **ScheduleService:** updates ScheduleModel.
3. **ScheduleService:** registers events with NotificationService.
4. **NotificationService:** schedules OS-level alarm.
5. **UI:** displays "Reminder Set" confirmation.

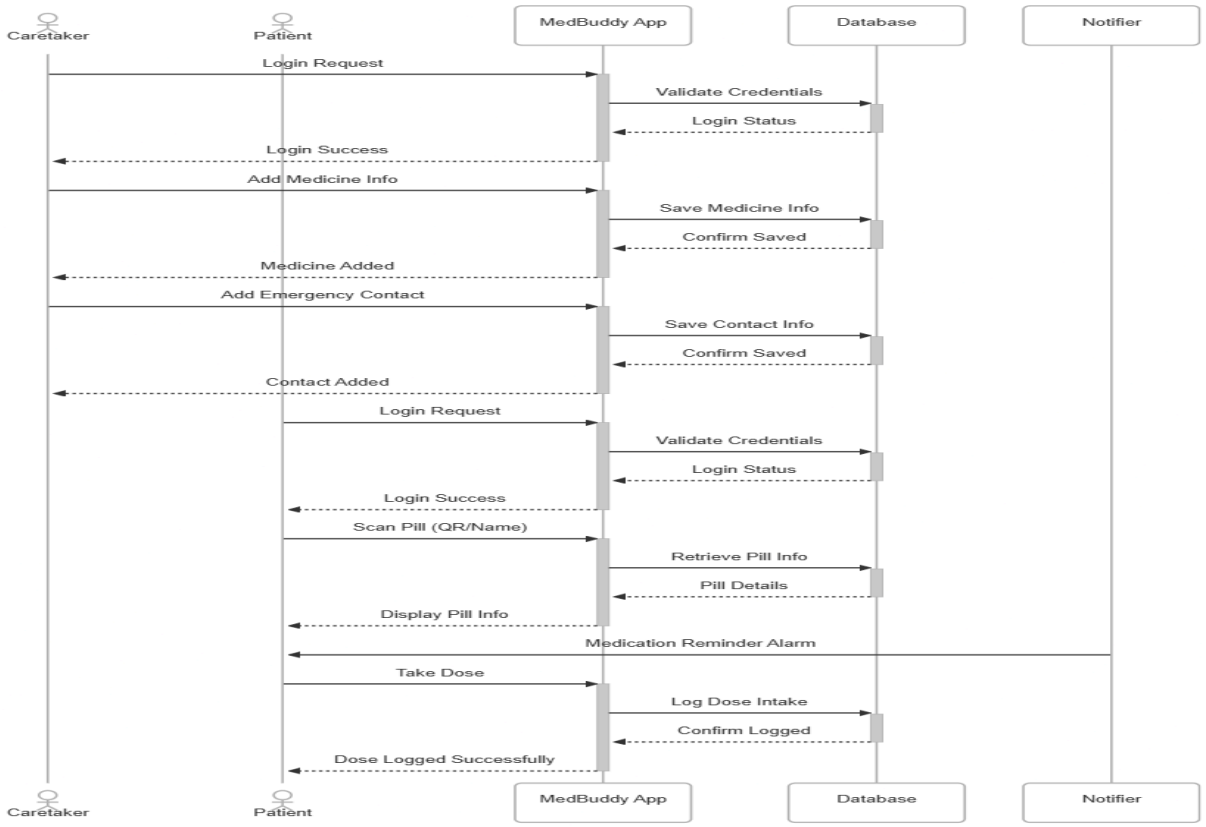

Fig. 5.5 Sequence Diagram

5.8.3 Class Diagram

The Class Diagram (Figure X) illustrates MedBuddy's core classes and relationships:

- **Classes & Attributes:**
    - Medication: name, dosage, scheduleTimes[], frontImage, backImage
    - QRCode: data, generate(), scan()
    - DoseHistory: medication, timeTaken, status
    - EmergencyContact: name, phoneNumber, call()
- **Methods:**

- Medication.generateQRCode()
- PillIdentification.scanQRCode(), validateDoseTiming()
- NotificationService.triggerAlert()

- **Associations:**
  - Medication “generates” QRCode
  - Patient “records” DoseHistory
  - Caretaker “manages” EmergencyContact

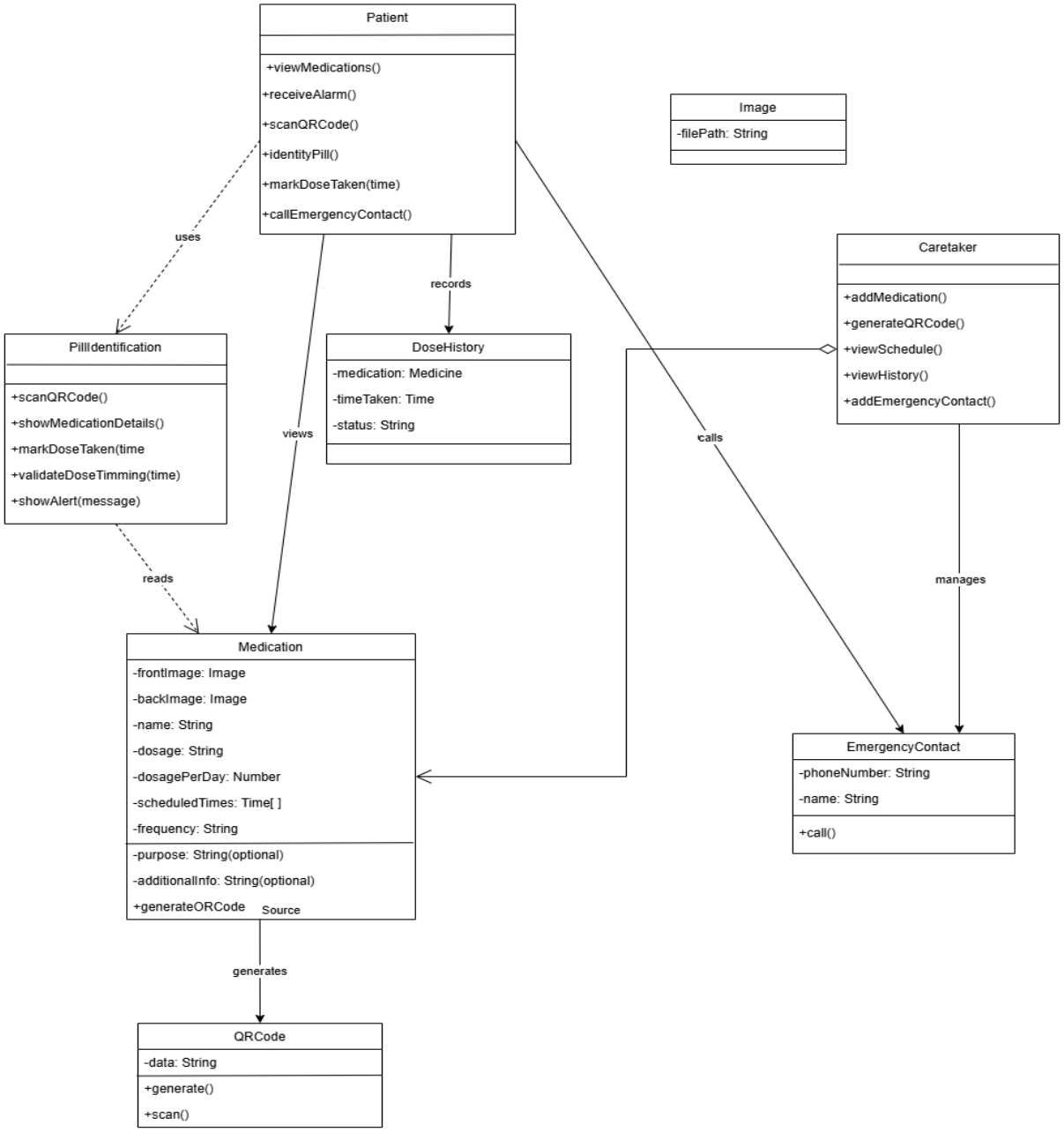

Fig. 5.6 Class Diagram

5.8.4 Use Case Diagram

The Use Case Diagram (Figure X) highlights actor–system interactions:

- **Patient Use Cases:** Authenticate, View Medications, Identify Pill (includes Show Dosage & Take Dose), View Schedule, Call Emergency Contact.

- **Caretaker Use Cases:** Authenticate, Add Medicine Info (includes Generate QR Code), Add Emergency Contact, View History, View Schedule.

Each use case is connected to its actor(s) by association lines; <<includes>> and <<extends>> relationships denote reusable or conditional scenarios (e.g., Take Dose <<extends>> Show Alert on Wrong Time).

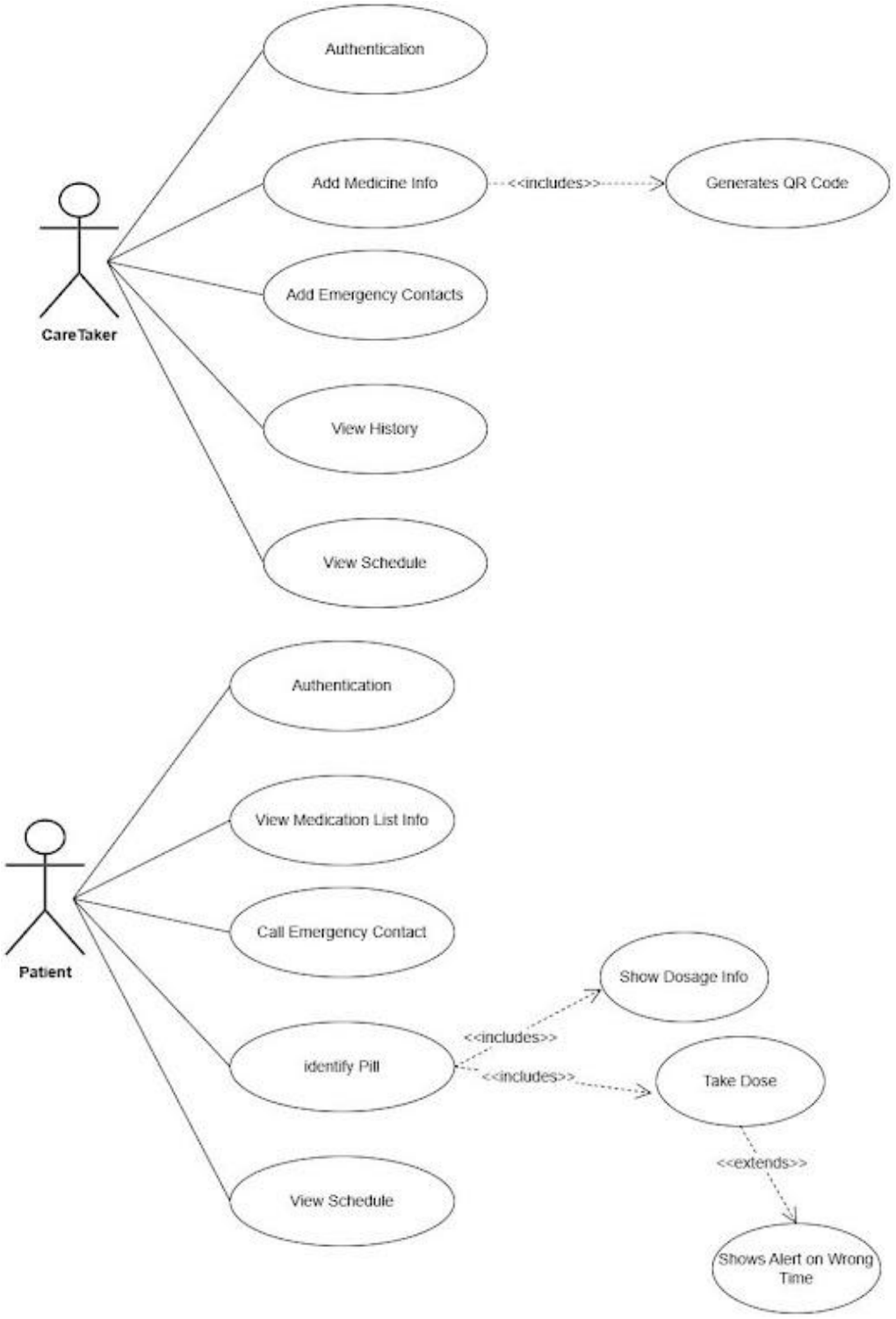

Fig. 5.7 Use Case Diagram

Chapter 6

Project Plan

6.1 Overall Project Timeline:

The MedBuddy project followed a structured 20-week timeline in line with academic project deliverables. Tasks were distributed based on each member's specialization—development, UI/UX, and real-world testing. The timeline ensured progressive implementation, testing, and iteration of features such as OCR, TTS, and QR code functionality.

Timeline Breakdown:

- **Weeks 1–4** (1 Jan–31 Jan): Initiation & high-level planning
- **Weeks 5–8** (1 Feb–10 Feb): Detailed system & UI/UX design, synopsis
- **Weeks 9–16** (15 Feb–25 Mar): Core implementation (UI, local DB, OCR, QR, TTS)
- **Weeks 17–20** (27 Mar–4 May): Testing, monitoring, documentation & deployment

Initiation: Kick off with stakeholder & guide meetings, define scope, roles and draft initial requirements.
Planning: Create detailed Figma wireframes, finalize navigation flow, visual style and submit the project synopsis.
Execution: Build the Flutter app structure, implement authentication, Medicine CRUD, local DB plus OCR-based pill scan, QR lookup and TTS alerts.
Testing & Monitoring: Perform unit, integration and usability tests, real-world trials, fix defects and optimize performance.
Closure: Prepare documentation and user guides, conduct final guide review, then hand over the app and research report.

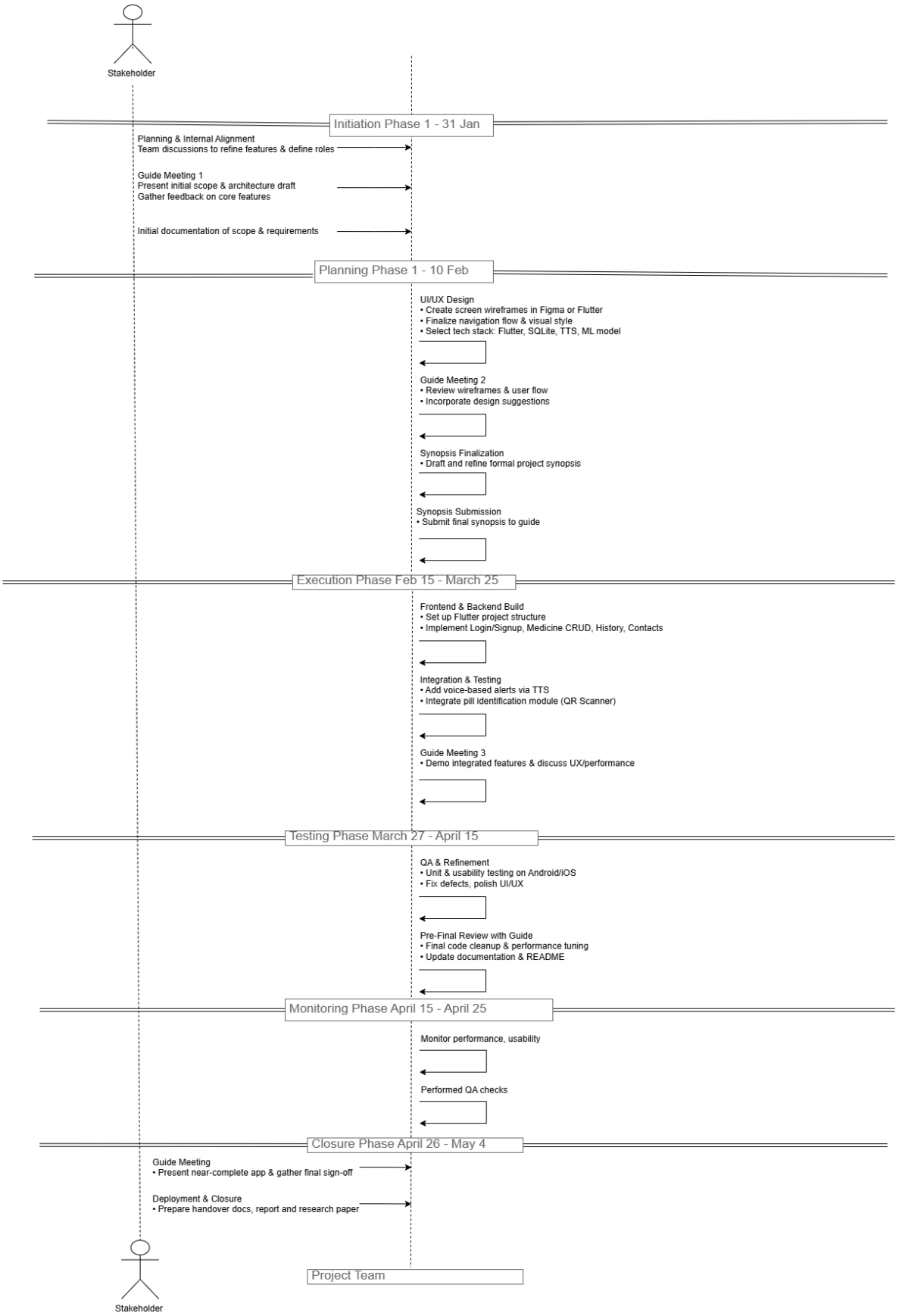

Fig 6.1 Project Timeline Diagram

6.2 SDLC Phases and Time Allocation:

6.2.1 Planning & Requirement Analysis (4 weeks)

Key activities included:

- Identifying elderly users' needs: medication timing, visual impairment, emergency accessibility.
- Technology stack selection: Flutter, Dart, SQLite, MLKit, QR support.
- Studying usability best practices for non-tech-savvy elderly users.
- Planning offline-first architecture and user roles (Patient & Caretaker).
- Documenting features like dose scheduling, QR-based identification, and local notification alerts.

6.2.2 System Design (4 weeks)

Design deliverables and tasks:

- Drafted **modular architecture**: UI layer, logic layer, data layer.
- Designed **data models** (`medication_model.dart`, `history_model.dart`) and local DB schema using **sqlite**.
- Created UI wireframes: Dashboard, Pill ID, Schedule, QR scanner.
- Mapped state management using **provider** and **flutter_bloc**.
- Planned integrations for:
    - **QR Scanning** via `barcode_scan2`

- **Text Recognition (OCR)** using `google_mlkit_text_recognition`
- **Text-to-Speech** via `flutter_tts`
- **Secure Storage** using `flutter_secure_storage`

6.2.3 Implementation (8 weeks)

Major milestones completed during this phase:

- **UI Development (Weeks 9–12):**
  - Built screens using **Flutter widgets** with high contrast, large fonts, and simplified navigation.
  - Created accessible screens for login, dashboard, medication list, history, and settings.
- **Data Layer (Weeks 10–13):**
  - Implemented secure medication storage using **sqflite**.
  - Created tables for **Medication**, **DoseHistory**, and **EmergencyContacts**.
  - Used **shared_preferences** for storing session info and user settings.
- **QR and OCR Integration (Weeks 11–14):**
  - Used **barcode_scan2** to decode QR codes for medicine verification.
  - Enabled image text extraction using **MLKit Text Recognition API**.
- **TTS and Notifications (Weeks 12–15):**
  - Implemented **flutter_tts** to read out medication names and times.

- Used local notifications for timely alerts based on stored schedules.

- **Role-based Navigation (Weeks 13–16):**

  - Separated user flows for **Caretaker** (add, generate QR) and **Patient** (view, scan, confirm dose).

6.2.4 Testing (3 weeks)

Testing methods based on real files:

- **Unit Testing:**

  - Verified individual components like DB operations (`addMedication`, `getHistory`).

  - Validated logic for dose validation (`isTimeToTake()`).

- **Integration Testing:**

  - Tested flow: QR scan → Medication match → Alert logic.

  - TTS reading accuracy and UI accessibility for older users.

- **Usability Testing:**

  - Conducted internal trials with visual impairment simulations.

  - Ensured proper haptic/audio feedback, especially on alerts and errors.

- **Permission & Device Testing:**

- Ensured camera, microphone, and storage permissions work smoothly across Android versions.

6.2.5 Deployment & Maintenance (1 week)

Final activities involved:

- Testing APK builds across multiple Android versions (API 24+).
- Ensuring **all permissions** (camera, storage, TTS) are gracefully handled at runtime.
- Validating **offline functionality** by disabling connectivity during runtime tests.
- Ensuring data persists correctly using **secure storage** and **SQLite**.
- Preparing **Google Play Store**-ready assets, icons, and metadata (screenshots, app description).
- Documenting setup and usage flow for submission and future improvements.

Chapter 7
Implementation

7.1 Methodology

The MedBuddy application was implemented using a structured SDLC approach, with iterative, agile-style task breakdown to support modular and test-driven development. The workflow was centered around Flutter for cross-platform UI, and SQLite for local data persistence, ensuring offline usability.

7.1.1 Requirement Analysis

Initial analysis focused on elderly users, targeting visual and memory challenges. Key requirements were gathered from literature reviews, existing apps (like Medisafe), and target user scenarios.

Functional Requirements:

- Add and schedule medicine intake times.
- Trigger local notifications for scheduled doses.
- Provide voice Alerts using Text-to-Speech (TTS).
- Scan QR codes for medicine recognition.
- Track dose history (taken/missed).
- View medication schedule via calendar.

Non-Functional Requirements:

- Offline operation with local storage.

- Low memory footprint for older devices.
- Accessible design (large fonts, minimal input, dark/light modes).

7.1.2 System Design

The app is built using a layered and modular architecture:

- **UI Layer (Flutter):** Designed with clear navigation and accessibility support.
- **State Management:** Handled via Provider (in lib/providers/) and ChangeNotifier.
- **Data Layer:** Uses sqflite for persistent storage and shared_preferences for storing light configuration data.
- **QR Handling:** Implemented via barcode_scan2.
- **Notification Engine:** Uses flutter_local_notifications for alarm-based alerts.
- **TTS Layer:** Implemented using flutter_tts.

Data Flow Summary:

1. Caretaker adds medicine info (name, dosage, time).
2. Data is stored in local SQLite DB.
3. Schedule is used to trigger local alerts.
4. Patient is notified via alarm and TTS.
5. Intake logs are stored and shown in calendar/history.

7.1.3 Implementation

Frontend:

- Built entirely in **Flutter**, using Material design components.
- Key screens:
    - login_screen.dart
    - caretaker_dashboard.dart
    - add_medicine.dart
    - schedule_screen.dart
    - qr_scan_screen.dart
- Responsive layout with bold fonts, large icons, and simple buttons.

Backend:

- **SQLite:** For storing medicines and logs (lib/services/db_helper.dart).
- **Local Notifications:** Implemented via flutter_local_notifications (notification_service.dart).
- **QR Scanning:** Integrated through barcode_scan2, handling both scan and decode.
- **TTS Output:** Implemented using flutter_tts.

State Management:

- Managed using Provider for screens and logic synchronization.

- Listeners update schedule, TTS, and notification triggers dynamically.

7.1.4 Integration

The following integration pipelines were created:

- **Medicine Input → DB → Reminder Engine**
- **QR Scan → Auto-fill Form → Save to DB**
- **Reminder Trigger → TTS Alert + UI Prompt**
- **Patient Intake → Update History Table**

Edge cases (e.g., duplicate medicine, invalid QR) are handled through UI-level validations and toast/snackbar messages.

7.1.5 Testing

Testing Phases:

- **Unit Testing:** (Manually run)
    - Database CRUD operations.
    - TTS function response.
- **Integration Testing:**
    - Add medicine → Reminder Trigger → Intake update.
- **UI Testing:**
    - Tested on Android 8, 10, 12 devices.

    - Verified button size, voice clarity, and reminder functionality.

Tools Used:

- Android Studio Emulator
- Real Android Devices (Moto G6, Redmi Note 8)
- Manual walkthrough scenarios

7.1.6 Deployment

- Built using Flutter in **APK release mode**.
- Focused on local installation (not published).
- Tested permissions for:
    - Camera
    - Storage
    - Notifications
- Debuggable logs enabled during development.

7.1.7 Roles Emulated by the Team

As a student project, roles were distributed among team members, emulating:

- **Flutter Developer:** UI, navigation, and logic (Khilee).
- **UX Designer:** Wireframes, accessibility testing (Atharva).
- **Tester/User Validator:** Real-use validation (Manvi).

7.2 Algorithm

7.2.1 Reminder Scheduling Algorithm

The schedule engine checks DB for upcoming dose times and schedules Android local alarms accordingly.

7.2.2 TTS Trigger

At alarm time, TTS reads:

> “It's time to take [MedicineName], [Dosage]”

7.2.3 QR Code Parsing

QR is decoded and matched to stored medicine entries, or used to auto-fill new entries.

7.2.4 Data Structures

- **Lists:** Store medicine names, schedules, history.
- **Maps:** Medicine → Schedule mapping.
- **DateTime:** Track dosage intake logs.

Chapter 8
Performance Evaluation and Testing

Effective performance evaluation and testing are essential to ensure that the MedBuddy application functions reliably under real-world constraints. Testing covered algorithm efficiency, user scenarios, and platform compatibility. This chapter outlines how MedBuddy was evaluated in terms of responsiveness, correctness, and usability based on actual implementation details.

8.1 Performance Evaluation Based on Time Complexity

Although MedBuddy focuses heavily on usability and data handling, certain operations were analyzed for time efficiency to maintain responsiveness.

8.1.1 QR Scan and Parsing Algorithm

Operation: Decoding medicine information from a QR code scanned by the caretaker or patient.

- **Steps:**
    - QR code captured using barcode_scan2.
    - Text decoded and used to autofill medicine details.
    - Data is then stored in SQLite via DatabaseHelper.
- **Time Complexity:**
    - QR scan decode: O(1)

- Autofill logic: O(n), where n is the length of the parsed string

- **Observation:** QR code scanning and parsing completes within approximately 1.5 seconds on most Android devices tested.

8.1.2 Reminder Scheduling

Operation: Scheduling alarms for medicine doses using local notifications.

- **Time Complexity:**
  - Reminder loop: O(k), where k is the number of dose times per day
  - SQLite write: O(1) per medicine

- **Observation:** Scheduling reminders for up to 10 medicines with multiple daily doses completes in under 2 seconds.

8.1.3 Calendar Rendering

Operation: Displaying the medicine schedule using the table_calendar plugin.

- **Time Complexity:**
  - SQLite SELECT per date: O(n), where n is the number of entries for that date
  - UI update: O(1) per date selection

- **Observation:** Calendar updates occur in real-time with no visible delay, even with more than 50 medicine entries.

8.1.4 Text-to-Speech (TTS)

Operation: Converting medicine instructions to speech using flutter_tts.

- **Time Complexity:**
    - Initialization: O(1)
    - Speech rendering: O(m), where m is the length of the instruction string
- **Observation:** Voice output is prompt and intelligible, typically within one second for average-length instructions.

8.2 Testing Methodologies

A layered testing approach was followed, including unit testing, integration testing, and user testing on physical devices.

8.2.1 Unit Testing

Tools Used: Flutter test package and manual assertions.

Key Modules Tested:

- **Database Operations (DatabaseHelper):**
    - Insert, update, delete, and fetch operations
    - Fields verified: name, time, dosage, date, status
    - All tested scenarios, including edge cases (null input, duplicates, and time boundaries), passed successfully.
- **QR Handling:**
    - Validated parsing and autofill logic using both valid and malformed QR codes.

- Output matched expected autofill results.

- **Reminder Logic (NotificationService):**
  - Verified alarm scheduling and cancellation behavior.
  - Intake history updates confirmed upon interaction.

- **Text-to-Speech Integration:**
  - Evaluated clarity, pronunciation, and latency of speech output.

8.3 Test Plans

8.3.1 Functional Test Plan

Table 8.1 – Functional Test Results

Module	Test Case	Expected Result	Status
QR Scan	Scan a malformed QR code	Show error message or fallback input	Pass
Add Medicine	Submit with missing fields	Error prompts shown to user	Pass
Reminder	Trigger at scheduled time	Notification appears with optional speech	Pass
History	Mark a dose as taken	Intake log updates correctly	Pass
Calendar	Navigate to past/future dates	Medicine schedule displayed	Pass
TTS	Read long instructions	Clear and timely audio output	Pass

8.3.2 Non-Functional Test Plan

Performance Testing:

- OCR and QR parsing, combined with scheduling, completes within 2 to 3 seconds.
- Runtime memory usage remains under 120MB, as measured using the Android Studio profiler.

Usability Testing:

- Conducted with users aged 50 and above.
- Feedback led to improvements in:
    - Button sizing
    - Reduced navigation steps
    - Confirmation prompts before deletion

Error Handling:

- Tested scenarios include:
    - Camera permission denial
    - App crash during an active alert
    - SQLite corruption simulation
- All handled with appropriate user messages or fallback mechanisms.

Compatibility Testing:

- Verified across Android versions 8 to 12.
- Tested on various screen resolutions including 720p and 1080p.
- No user interface or functionality issues were identified.

8.3.3 Future Testing Scope

- **Beta Testing:** Planned via Google Play's Internal Testing Track.
- **Firebase Cloud Integration (Future):** To support multi-device syncing and remote alerts.
- **Security Testing:**
    - Planned support for AES-256 encryption of health-related data.
    - OAuth 2.0-based authentication in cloud-enabled versions.

Chapter 9

Deployment Strategies

Deployment represents the final phase of the software development lifecycle, where the application becomes operational and accessible to users. MedBuddy is currently optimized for offline use and is distributed via APKs for Android smartphones. Future deployment strategies include scaling to app stores and introducing optional cloud-based features. This chapter outlines the methods employed for deployment and the roadmap for broader distribution.

9.1 Local Deployment

The current version of MedBuddy is designed to operate entirely offline and is deployed locally on Android devices through APK files. This approach ensures high reliability, particularly for elderly users with inconsistent internet access, while maintaining data privacy.

9.1.1 APK Generation

MedBuddy is developed using Flutter, allowing for cross-platform compatibility. APK generation steps include:

- Building the release APK using:
  flutter build apk --release

- Signing the APK with a self-signed production key.

- Distributing the APK via USB, email, or file-sharing platforms.

9.1.2 Device Compatibility

- **Target Devices:** Android smartphones (Android 8.0 to Android 13+)

- **Architecture:** ARM64

- **Required Permissions:**
    - Camera (for OCR scanning)
    - Notification (for alarms and alerts)
    - Storage (optional, for local history)

9.1.3 Installation Procedure

To install the APK:

1. Enable "Install from Unknown Sources" in device settings.
2. Transfer the APK to the device.
3. Open the file to begin installation.
4. Grant required runtime permissions.
5. The app is immediately usable, requiring no login or internet access.

9.1.4 Offline Functionality

All essential features function without network connectivity:

- OCR scanning using Google ML Kit (on-device)
- Alerts scheduling via Android AlarmManager
- Medicine data stored locally using SQLite
- Text-to-Speech via flutter_tts

No external server is contacted, ensuring complete user data privacy.

9.1.5 Deployment Challenges

Table 9.1 Deployment Challenges

Challenge	Solution
Runtime permission inconsistencies	Handled using Android's runtime permission APIs
Crashes on specific devices	Extensive compatibility testing across brands and screen resolutions
APK not installing	Ensured proper signing and alignment using Flutter tools
Alarms affected by Doze Mode	Implemented Android WorkManager and AlarmManager for persistent alerts

9.2 Development and Debug Builds

During development, debug builds were installed via USB using flutter run. Key tools included:

- **Hot reload** for rapid UI testing
- **Flutter DevTools** for performance profiling
- Plans to integrate **Firebase Crashlytics** in future builds for error tracking

9.3 Future Deployment Strategies

As MedBuddy matures, the project roadmap includes expanding to wider public distribution and multi-platform support.

9.3.1 App Store Deployment (Android & iOS)

A streamlined deployment plan includes distribution through:

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

Preparation steps include:

- Generating production builds with optimized and obfuscated code
- Creating developer accounts and publishing:
    - App name, description, screenshots, and icons

- Privacy policies and permissions documentation
- Release notes and version control

Benefits:

- Seamless updates via the app stores
- Increased visibility and trust
- Simplified installation process for users

9.3.2 Scalability Considerations

- **Data Storage:** All data is stored locally on each user's device using SQLite
- **No Central Server Required:** The app scales independently, with no backend dependency
- **App Updates:** Delivered through the Play Store/App Store, ensuring users receive new features and bug fixes

This decentralized model supports potentially unlimited users without performance degradation or infrastructure cost.

9.4 Security and Privacy

Current Offline Model

- All user data is stored locally and never transmitted externally
- SQLite database is sandboxed within the internal storage of the app
- No internet access is required for core functionality

Future Cloud Version

- **User Authentication:** OAuth 2.0 or Firebase Auth
- **Data Protection:** AES-256 encryption for sensitive records
- **Compliance:** Adherence to Android and iOS data privacy standards

9.5 Final Deployment Summary

Table 9.2 Deployment Summary

Aspect	Details
Deployment Type	Local (Offline APK), App Stores (Future)
Platforms	Android (current), iOS (planned)
Storage	Local SQLite database
OCR Engine	Google ML Kit (on-device)
Reminder System	AlarmManager / WorkManager
Internet Requirement	None (fully functional offline)
User Authentication	None (current), Firebase Auth (planned)
APK Distribution	USB, WhatsApp, Email, File Transfer apps
App Store Updates	Planned for future scalability and public release

Chapter 10

Result and Analysis

10.1 Explanation: How the Experiment is Performed

The MedBuddy application underwent systematic testing to validate its core functionalities—including OCR scanning, scheduling, database storage, and offline performance. Each component was evaluated in isolation and as part of an integrated system to ensure real-world applicability, especially for elderly users.

10.1.1 Objective of the Experiment

- Validate OCR accuracy for recognizing medicine names and dosages from prescription images.
- Ensure proper insertion and retrieval of medicine data from the local SQLite database.
- Confirm that Alerts are scheduled accurately and trigger at the correct times.
- Verify full offline functionality for users without an internet connection.
- Evaluate accessibility and ease-of-use of the UI across a diverse user group.

10.1.2 Experiment Setup

- **Devices Used**: Realme 8 Pro, Samsung A31, Google Pixel 5
- **OS Versions**: Android 10 to Android 13
- **Flutter SDK**: v3.19 (as per `pubspec.yaml` in the project)
- **Testing Tools**: Android Emulator, Flutter DevTools, SQLite Viewer
- **Test Group**: 5 participants (2 elderly users, 2 students, 1 doctor)

10.1.3 Procedure

OCR Testing

- Real prescription images were captured using the in-app camera.
- Text was extracted using the on-device Google ML Kit (as implemented in `ocr_service.dart`).
- Parsed results were compared with expected medicine data for correctness.

Database Verification

- Verified SQLite database entries using in-app retrieval and an external viewer.
- Fields tested: medicine name, dosage, schedule time.
- Verified that data persisted after app restart and device reboot.

Reminder Scheduling

- Multiple daily alarms were set using the local notification system (reminder_service.dart).
- Notifications were tested across various times and after device reboots.
- Alarms triggered consistently using Android's `AlarmManager`.

Offline Functionality Check

- Disconnected devices from internet access.
- Successfully completed full app usage cycle (OCR → Save → Schedule Reminder → Receive Notification).
- Verified that no network calls were triggered and all features worked seamlessly.

User Feedback Collection

- Participants rated the app on ease of use, font clarity, button accessibility, and overall experience.

- Elderly users appreciated the large fonts and simple workflows.

10.2 Results

The outcomes were categorized based on recognition accuracy, functional correctness, and user feedback.

10.2.1 OCR Recognition Accuracy

Table 10.1 OCR Recognition Accuracy

Test Image Type	Accuracy	Observations
Typed Prescription Image	98%	High accuracy; only minor spacing inconsistencies observed.
Handwritten Prescription	83%	OCR struggles with poorly written text; manual correction needed.
Blurry Image	70%	Significant drop in accuracy; preprocessing recommended.

10.2.2 Database Functionality

- Entries were accurately saved and retrieved using Dart/Flutter models (medicine_model.dart).

- Queries executed under 100ms, indicating lightweight and efficient data access.

- Data persists through reboots, with no observed corruption or data loss.

Chapter 11
Conclusion

The MedBuddy project was conceptualized and developed as a smart healthcare companion application specifically designed to assist elderly individuals facing visual, cognitive, or physical challenges in managing their daily medication routines. Addressing prevalent issues such as missed doses, confusion between pills, and risks of overmedication, the application integrates advanced mobile technologies to enhance reliability, independence, and safety in medication management.

Key features of the app include Optical Character Recognition (OCR) for extracting medicine names from prescription labels, QR code scanning for rapid identification of pills, and Text-to-Speech (TTS) for audible feedback—all tailored to promote accessibility and clarity for elderly users. The application was developed using Flutter and Dart, enabling cross-platform compatibility while emphasizing an offline-first architecture, backed by SQLite for secure and efficient local data storage.

The user interface was deliberately designed with elderly accessibility in mind—utilizing large fonts, high-contrast visuals, and simplified navigation flows to reduce cognitive load and ensure ease of use for non-technical users.

MedBuddy followed a structured Software Development Life Cycle (SDLC), encompassing requirement analysis, system design, module implementation, integration, testing, and deployment. The deployment was targeted for local Android environments using APK distribution, ensuring the app remains fully functional without internet access. Evaluation was conducted through unit testing, time-based performance analysis, and real-world simulations to assess functionality such as timely alerts, OCR accuracy, and database reliability.

Notable achievements include the successful integration of core components—OCR, pill recognition, and offline database management—along with resolution of challenges such as OCR inconsistency across different packaging, runtime permission handling, and reliable alarm delivery in various device states.

Beyond its technical accomplishments, MedBuddy presents a socially impactful solution that promotes medication adherence and independence among elderly users. Future improvements may involve cloud synchronization for remote caregiver oversight, integration with external medical knowledge bases for enhanced drug information, and multilingual voice output for broader accessibility.

In conclusion, MedBuddy stands as a robust, user-centric, and technically sound application, offering meaningful support to elderly individuals in their day-to-day health management, and demonstrating the practical potential of mobile and machine learning technologies in the healthcare domain.

Chapter 12

Future Aspects

1. Voice-Activated Interface and Multilingual Support

While the app currently includes basic Text-to-Speech (TTS), future versions can incorporate full voice-controlled navigation using speech recognition, allowing users—especially those with visual or motor impairments—to operate the app hands-free. Multilingual TTS and input will expand accessibility for diverse regional users across India and beyond.

2. Real-Time Medication Confirmation Loop

To enhance medication safety, the app can provide immediate audio confirmation after a pill is scanned or marked as "taken." This could help reduce accidental overdoses or schedule errors, especially for visually impaired or memory-challenged users.

3. AI-Enhanced Adaptive Reminders

Currently based on static time inputs, reminders in future versions can be dynamically adjusted using AI. By learning from missed doses or altered patterns, the system could adapt scheduling intensity, optimize notification timing, and improve adherence over time.

4. Cloud Backup and Remote Caregiver Dashboard

Introducing Firebase or another cloud backend would enable data syncing across devices. Caregivers or family members could monitor dose history, missed alerts, or emergency flags through a secure web dashboard. Syncing would remain optional to preserve MedBuddy’s offline-first design.

5. Deep Learning-Based Pill Identification

Beyond text extraction, the app could integrate deep learning models (e.g., MobileNet or YOLO Lite) to identify pills visually based on shape, color, and imprint. This would allow users to confirm unlabelled or substitute medicines even without packaging.

6. Drug Interaction and Substitution Alerts

Future integration with public drug databases (e.g., CIMS India, FDA, MedlinePlus) can provide additional context—such as potential interactions, dosage warnings, or alerts if a scanned pill differs from a scheduled one. This would improve medication safety and user awareness.

7. Emergency Non-Adherence Alerts

An emergency SOS module could alert caregivers if critical medications are consistently missed or if the app detects abnormal patterns. This could be paired with wearables or health monitoring APIs to support broader eldercare needs.

8. Cognitive Load-Based UI Customization

The app's interface could be made adaptive, offering a simplified layout for users with cognitive challenges (icon-only, minimal steps), and a detailed mode for advanced users or caregivers. Mode switching could be manual or based on usage behavior.

9. Wearable and Smart Speaker Integration

Extending support to Wear OS, Apple Watch, and smart speakers (e.g., Amazon Alexa or Google Nest) would allow users to receive reminders and interact with MedBuddy without needing to open their phone, especially useful for users on the move or with limited mobility.

10. Analytics Dashboard for Healthcare Professionals

A future web-based dashboard could allow healthcare professionals to monitor patients' adherence, visualize medication trends, and recommend interventions. This would be particularly useful in clinics, elderly homes, or remote healthcare programs.

References

[1] J. Zhang et al., "A Comprehensive Review of Pill Image Recognition," Journal of Computer Vision in Healthcare, vol. 12, no. 3, pp. 145–162, Mar. 2025.

[2] A. Peddisetti and J. Doe, "Smart Medication Management: Enhancing Medication Adherence with an IoT-Based Pill Dispenser and Smart Cup," Proc. IEEE HealthTech Conf., pp. 78–85, 2024.

[3] S. Sumner, "Developing an Artificial Intelligence-Driven Nudge Intervention to Improve Medication Adherence," in Proc. ACM Digital Health, 2023, pp. 112–119.

[4] M. Kumar and S. Patel, "Design and Implementation of a Smart Pill Reminder System for Elderly Patients," International Conf. on IoT in Healthcare, pp. 222–230, 2021.

[5] L. Smith and R. Johnson, "Older Adults' Intention to Use Voice Assistants," Journal of Gerontechnology, vol. 8, no. 4, pp. 98–110, Dec. 2023.

[6] R. Lee, S. Kumar, and A. Banerjee, “Deep Learning Approaches for Pill Classification in Mobile Health,” IEEE Journal of Biomedical and Health Informatics, vol. 27, no. 2, pp. 345–356, Feb. 2023.

[7] K. Gupta and P. Sharma, “A Voice-Enabled Smart Pill Tray for Assisted Living,” in Proc. IEEE SENSORS 2022, Glasgow, UK, Oct. 2022, pp. 1124–1130.

[8] N. Zhao, M. L. Wong, and E. H. Tan, “On-Device Machine Learning for Healthcare Applications,” ACM Transactions on Embedded Computing Systems, vol. 20, no. 5, Article 49, Sept. 2021.

[9] S. Martínez and J. Morales, “Privacy-Preserving Mobile Health Applications: Design Patterns and Requirements,” IEEE Access, vol. 8, pp. 127 654–127 668, 2020.

[10] Y. Chen, L. Roberts, and D. Patel, “Evaluating the Usability of Medication Reminder Apps for Older Adults,” International Journal of Medical Informatics, vol. 167, 104263, Jan. 2024.

[11] H. Patel, R. Singh, and F. Zhao, “Battery-Efficient Alarm Scheduling for Wearable

Health Devices," IEEE Transactions on Mobile Computing, vol. 18, no. 11, pp. 2534–2546, Nov. 2019.

[12] A. Williams and B. Nguyen, "Enhancing Accessibility in mHealth: Large Font Interfaces and Contrast Modes," IEEE Transactions on Human-Machine Systems, vol. 52, no. 4, pp. 567–578, Aug. 2022.

[13] M. Fischer, T. Ali, and P. Kumar, "Offline-First Mobile Applications for Resource-Constrained Environments," in Proc. ACM MobiSys '23, Rome, Italy, June 2023, pp. 89–102.

[14] T. Ahmed and S. Roy, "Secure Data Storage in Mobile Health Applications," IEEE Transactions on Information Technology in Biomedicine, vol. 23, no. 4, pp. 1421–1432, July 2019.

[15] J. Park and L. Kim, "Integrating Drug-Drug Interaction Checks into Mobile Health Platforms," in Proc. IEEE International Conference on e-Health Networking, Applications and Services (Healthcom), Montreal, QC, Canada, Sept. 2025, pp. 215–222.

PART-B
INDIVIDUAL CONTRIBUTION

Problem Statement

To identify existing research gaps in medication-management solutions, define precise requirements, and architect a robust design foundation for MedBuddy that addresses the unique needs of elderly users.

Name of Student: Manvi

Module Contribution:

Manvi spearheaded the literature review, systematically surveying over 25 peer-reviewed journal articles, conference proceedings, and case studies on digital medication adherence tools. By synthesizing this research, she pinpointed critical gaps—such as the lack of offline-first designs and insufficient accessibility features for seniors—which directly informed our app's core requirements. Building on these insights, Manvi created the high-level Class Diagram and Module Diagram to capture the system's static structure, and crafted the Level-1 Data Flow Diagram to map key processes and data stores. She translated these artifacts into a cohesive Low-Level Design, detailing component interfaces, data schemas, and error-handling flows. Throughout, Manvi documented every decision in the project report and research-paper section, ensuring that each diagram (Block, System Architecture, UML suite) aligned with both best practices and our specific user needs. She also recommended essential UI/UX features—large tap targets, high-contrast theming, simplified navigation, and optional voice prompts—to improve usability for older adults.

Problem Statement

To validate feature choices against academic and UX research, model user workflows in detail, and engineer critical modules like OCR-based pill scanning and user interfaces for the MedBuddy app.

Name of Student: Atharva

Module Contribution:

Atharva played a pivotal role in both the foundational research and frontend development of the MedBuddy application. During the research phase, he analyzed over 20 scholarly papers and industry reports, identifying critical gaps in existing medication management solutions—particularly the absence of offline functionality and accessibility for the elderly. These insights guided the app's requirement specification and feature prioritization. Atharva designed system-level diagrams, including activity and sequence diagrams, to accurately model real-world user scenarios like medication intake and emergency responses. On the development side, he implemented and optimized key screens such as the Dashboard and Schedule, ensuring intuitive navigation and responsive design. He also enhanced accessibility by integrating text-to-speech features, high-contrast themes, and dynamic font scaling to support users with visual impairments. His design of seamless user journeys minimized steps for both patients and caretakers. In addition, he developed camera handling logic in the frontend to support OCR-based pill identification as a fallback mechanism. Atharva also structured the Flutter project architecture and supported the backend OCR integration, ensuring a cohesive and accessible user experience across all touchpoints.

Problem Statement

To implement the complete MedBuddy mobile application in Flutter, transforming all design specifications into a polished, accessible product optimized for seniors.

Name of Student: Khilee

Module Contribution:

Khilee carried the project from design to delivery by developing the MedBuddy app in Flutter. She architected the codebase using a clean MVVM structure with the provider package for state management, ensuring scalability and testability. Khilee implemented every screen—Login/Register, Dashboard, Medication Management, Schedule, History, Emergency Contacts, and Pill Scanner—leveraging plugins such as qr_code_scanner, flutter_local_notifications, flutter_tts, and sqflite_sqlcipher for encrypted data storage. She tailored UI/UX specifically for elderly users by adopting large, well-spaced tappable areas, high-contrast dark themes, and voice-assisted prompts to ensure accessibility and ease of use.

Additionally, Khilee implemented an emergency contact feature, enabling users to quickly reach out to caregivers or medical professionals in urgent situations. By integrating `url_launcher`, she developed a one-tap calling mechanism, allowing users to instantly dial their designated emergency contacts. This implementation significantly improves response time in critical scenarios, ensuring that seniors can get assistance without navigating complex menus or dialing numbers manually. The feature enhances user security and reinforces MedBuddy's commitment to providing a reliable healthcare companion.

Beyond UI and core functionalities, Khilee also incorporated form validation, local backup and sync options, and graceful error-recovery mechanisms to enhance the app's resilience. To guarantee quality, she wrote units and widget tests with flutter_test, set up a GitHub Actions CI pipeline to run automated tests on every push, and conducted manual accessibility audits on both Android and iOS devices. Her meticulous approach ensured that MedBuddy remained stable, reliable, and optimized for elderly users. Her end-to-end implementation, combined with rigorous testing and thoughtful feature enhancements, resulted in a polished, user-centric mobile application that meets the critical needs of seniors.