الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي و البحث العلمي

Université Abou Bekr Belkaid
Tlemcen Algérie

جامعة أبي بكر بلقايد
تلمسان الجزائر

Peoples Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

FACULTY OF TECHNOLOGY

DEPARTMENT OF BIOMEDICAL ENGINEERING

Thesis

For obtaining the degree of

MASTER in Biomedical Engineering

Specialty: Biomedical and Hospital Informatics

Presented by:

Miss LANEG Dyma
Miss MEKKAOUI Takoua

Theme

MedGarde: Mobile application for assisting caregivers and optimizing mobile medical records

Jury members

Mrs BENSMAIL Ilham	President	M. C.B University of Tlemcen
Mrs KAZI TANI Lamia Fatiha	Examiner	M. C.A University of Tlemcen
Mr. TALEB Tariq	Supervisor	M. C.B University of Tlemcen

Academic Year: 2024 – 2025

Dedication

I dedicate this work to my family, especially my father, Mekkaoui Aoun and my mother, Zebdi Naziha. Their guidance and love have always helped me find the right path.

I also dedicate it to my sisters (Maroua and Nour El-Houda) and brothers: Oussama, Mohamed and youcef, who have supported me every step of the way.

A deep gratitude to my best friend Briki Nada Amina for always encouraging me and caring for me.To my thesis partner Dyma Laneg, your teamwork made this journey more meaningful and possible.To everyone who supported me during the challenging times while making this work.

Thank you from the bottom of my heart.

-MEKKAOUI TAKOUA-

I would like to dedicate this work to my wonderful parents, LANEG Said and BOUDIEB Fadila. For inspiring me to dream big and teaching me that consistency is the key, to my amazing siblings, Bouchra and Yacine, I appreciate how you always see me as your role model and always be by my side. For my Meriem, who has been there for me through all of life's ups and downs. And most of all, I dedicate this to you my dear partner MEKKAOUI TAKOUA with love and appreciation for all that you are and all that you do, for your patient and the moments we shared along together in this road.

Finally, to everyone who have supported me and believed in me along the road.

-LANEG Dyma-

Acknowledgment:

First and foremost, I express my sincere gratitude to Allah for granting us the strength, patience, and willingness to undertake and complete this study.
We would like to express our gratitude to all who contributed to this thesis.
We thank our supervisor Mr.TALEB Tariq for his guidance, support, and encouragement throughout this journey. As we are grateful to Tlemcen University and its Professors for shaping our technical skills our critical thinking and more than .all the moral lessons they patiently passed to us
A special thanks to our dear companion Malek through this journey for her support and valuable advice, and to the student Mr.Ghalmi for his assistance with the Flutter .framework
I also appreciate Alanya University for their help in identifying our project problem .finding the optimal solution
Finally, I thank the jury members for their time and effort in reviewing this work.
This thesis wouldn't have been possible without everyone's support.

Abstract:

Healthcare delivery in developing nations, particularly Algeria, faces significant challenges due to fragmented medical record systems, inadequate digital infrastructure, and limited access to suitable mobile health solutions. This thesis presents the design, development, and implementation of MedGarde, a comprehensive mobile health application built with Flutter as the front-end framework and Firebase as the backend platform and database storage, specifically tailored to address Algeria's healthcare challenges while considering local language, culture, and technology constraints. **Keywords:** Mobile Application, Patient Monitoring, Caregiver, Flutter, Firebase, Medical Records, Healthcare App.

Résumé :

La prestation de soins de santé dans les pays en voie de développement, en particulier en Algérie, est confrontée à des défis importants en raison de systèmes de dossiers médicaux fragmentés, d'une infrastructure numérique inadéquate et d'un accès limité à des solutions de santé mobile appropriées. Cette thèse présente la conception, le développement et la mise en œuvre de MedGarde, une application de santé mobile complète construite avec Flutter comme framework frontal et Firebase en tant que plateforme backend et solution de stockage et base de données. Elle est spécifiquement conçue pour répondre aux défis du système de santé en Algérie, tout en tenant compte de la langue, de la culture et des contraintes technologiques locales. **Mots-clés:** Application mobile, suivi des patients, garde malade, flutter, firebase, dossies médicaux, application de santé.

ملخص :

يواجه تقديم الرعاية الصحية في الدول النامية، ولا سيما الجزائر، تحديات كبيرة بسبب أنظمة السجلات الطبية المجزأة، وعدم كفاية البنية التحتية الرقمية، ومحدودية الوصول إلى الحلول الصحية المتنقلة المناسبة. تقدم هذه الأطروحة تصميم وتطوير وتنفيذ تطبيقMedGarde ، وهو تطبيق شامل للرعاية الصحية على الهاتف المحمول تم تصميمه باستخدام فلاتر كإطار عمل للواجهة الأمامية وفاير بايز كمنصة خلفية وتخزين قاعدة البيانات، وهو مصمم خصيصًا لمواجهة تحديات الرعاية الصحية في الجزائر مع مراعاة القيود اللغوية والثقافية والتقنية المحلية. **الكلمات المفتاحية:** تطبيق الهاتف المحمول, متابعة المريض, مقدم الرعاية, فلاتر, فايربايز, السجلات الطبية, تطبيق الرعاية الصحية.

List of Contents

CHAPTER 2 Design and Modelling

CHAPTER 3 Technology Selection and Justification for MedGarde

List of tables

List of figures

General introduction

General introduction

Healthcare in countries like Algeria still faces many problems, especially in areas with weak digital systems. Patients often have paper records kept at different clinics, which makes it hard for doctors to see the full medical history. This can lead to medication mistakes, late diagnoses, and more work for family caregivers who are not trained.

At the same time, few mobile health apps work in Arabic or are easy to use for people with low digital skills. Many patients, especially older adults, still depend on paper documents. These problems show the need for a new solution that fits the local context.

A 5W1H and Fishbone analysis reveals that the root causes of poor healthcare delivery include: (1) disjointed medical records due to a lack of centralized systems with heavy reliance on paper-based documentation , (2) minimal policy enforcement or funding for digital infrastructure, (3) uncoordinated medication management, (4) Few user-friendly mobile health applications exist in Arabic, healthcare centers lack digital infrastructure for patient data management, and (5) societal hesitation and limited literacy toward digital health tools. As shown in the root cause analysis, these issues not only delay accurate diagnosis and treatment but also place undue stress on caregivers, particularly in rural and digitally underserved communities.

What?	Poor Healthcare Delivery.
Where?	Nations with underfunded digital health infrastructure (e.g., Algeria).
When?	Ongoing, worsened by rising demand for data-driven healthcare tools.
Why?	Insufficient policy prioritization/funding for integrated electronic health records.
How?	Medication errors, delayed diagnoses, caregiver burnout.
Who?	Patients, caregivers, and healthcare providers.

Table 01: *5W1H analysis Poor Healthcare Delivery.*

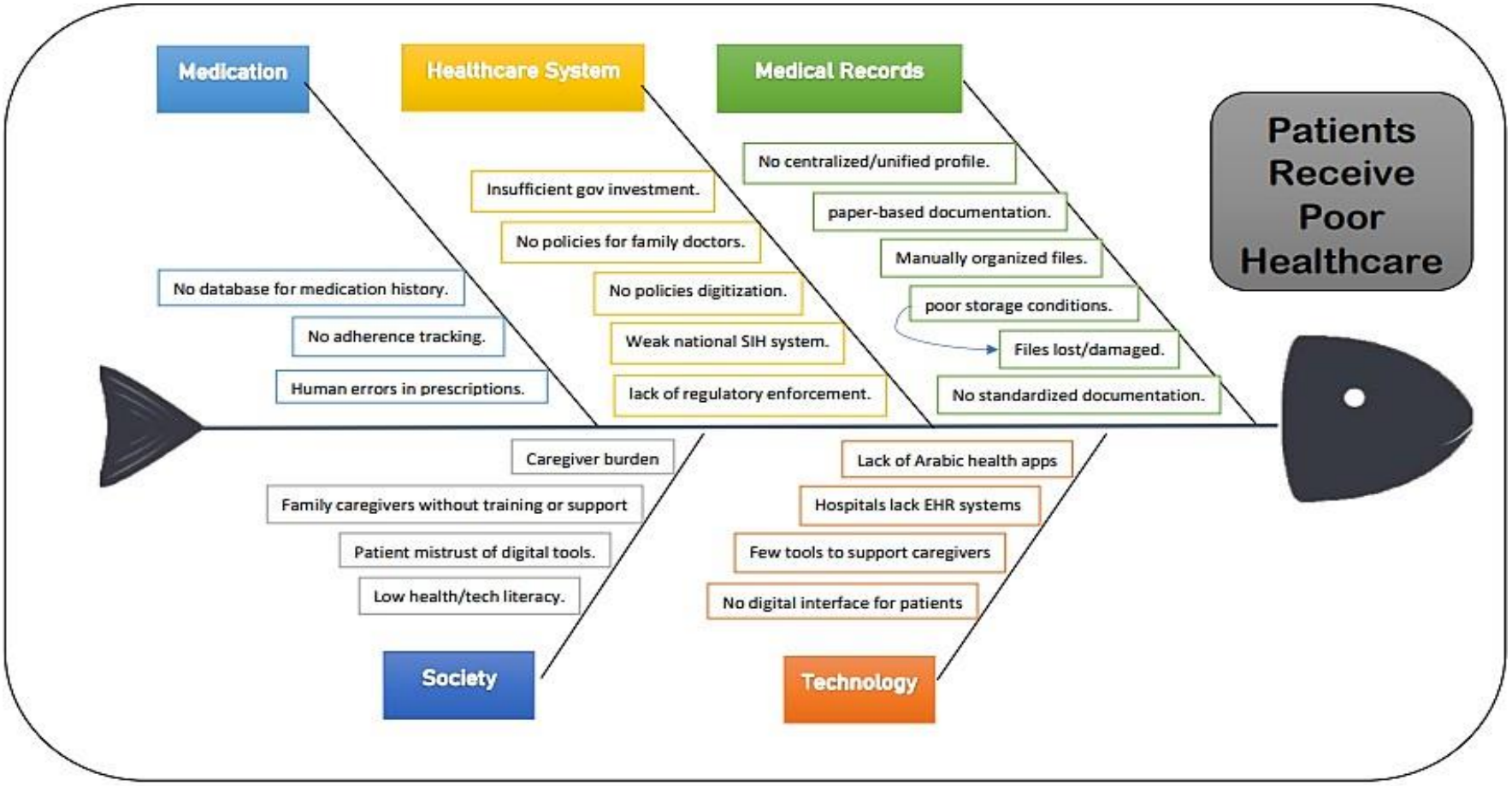


Figure 1: *Fishbone diagram – Root causes of poor patient healthcare*

This research proposes a new smartphone app called MedGarde, specifically designed to solve the healthcare problems we identified in Algeria. MedGarde combines several features into one easy-to-use app that helps patients, families, and doctors manage healthcare better.

What MedGarde Aims to Do:

1. Keep all medical information in one place: Instead of having medical papers scattered across different hospitals, MedGarde will store everything digitally on your phone:
    - Personal health information
    - Doctor appointment dates and times
    - All medications and dosages
    - Test results and medical reports
    - Everything in one secure, easy-to-find location

2. Help family caregivers: the app includes special features for family members who take care of sick relatives:
    - Simple tools to manage care for multiple family members
    - Easy ways to track if patients are taking their medicines correctly
    - Help coordinate appointments and treatments
    - Lower stress and confusion for family caregivers

3. Smart medication management: MedGarde uses artificial intelligence to help with medicines:
    - Automatic reminders to take medications on time
    - Warnings if two medicines might be dangerous together
    - Tracking to see if patients are following their treatment plans
    - Reduces mistakes and improves health outcomes

4. Works with existing hospitals: The app is designed to connect with hospital computer systems:
    - Can share information with hospitals that have digital systems
    - Can scan and store paper documents from hospitals still using paper files
    - Makes it easier for doctors to access patient information

5. Made for Algerian and Arabic users:
    - Available in the Arabic language first
    - Uses pictures and symbols that are easy to understand
    - Works for people who aren't very familiar with smartphones

6. Works without good internet: The app is designed to work even when the internet connection is poor:
    - Most features work offline
    - Works on basic smartphones, not just expensive ones
    - Accessible to people from all economic backgrounds

Secondary Objectives:

- Enable predictive analytics capabilities for early intervention and improved chronic disease management
- Create a platform for healthcare provider networking and appointment scheduling
- Implement gamification elements to encourage consistent app usage and healthy behaviors
- Establish secure data sharing mechanisms that maintain patient privacy while facilitating care coordination

This thesis is structured into five main chapters, each addressing specific aspects of the MedGarde development process:

Chapter 1: Literature and state of art

This chapter provides a comprehensive review of the current healthcare landscape, examining medical record management systems, the role of family caregivers in Algeria, and existing mobile health applications.

Chapter 2: Design and modelling

Presents the application architectural design, including requirements specification, use case modelling, database design, and sequence diagrams.

Chapter 3: Technology Selection and Justification

Provides detailed analysis and justification for the chosen technology stack, including the selection of Flutter for cross-platform development, Firebase for backend services, and supporting tools and frameworks. The chapter includes comparative analysis of alternative technologies and risk assessment strategies.

Chapter 4: Implementation and Development

This chapter presents the comprehensive development process of the MedGarde application, detailing the translation of design specifications into a functional mobile health solution

General conclusion: Results and Discussion

The thesis concludes with a comprehensive summary of contributions, current limitations, and perspectives for future development and expansion of the MedGarde platform.
This research contributes to the growing field of mobile health applications by demonstrating how culturally adapted, technically robust solutions can address specific healthcare challenges in developing nations while providing a foundation for broader digital health transformation initiatives.

Chapter 1:
Literature and state of art

1 Introduction

Healthcare systems worldwide struggle with patient information management. This problem is worse in developing countries due to limited resources and poor infrastructure.
Algeria, like many countries in the Middle East and North Africa (MENA region), struggles with a fragmented medical record system and relies heavily on family carers to support patient care.
This chapter aims to:

1. Define medical record types and management standards
2. Analyse the current state of healthcare documentation in Algeria
3. Examine the role of family caregivers in the Algerian healthcare system
4. Evaluate existing mobile health applications and identify gaps
5. Establish the foundation for developing MedGarde as a comprehensive solution

2 Medical Records in Healthcare Systems

2.1 Types and Definitions of Medical Records

A medical record is a collection of written or digital documents that include notes, reports, and any information related to the patient's health status, treatment plan, and clinical interventions. [2]

According to ISO/TR 14639-1:2012(en), an "*electronic medical record*" (EMR) is defined as an "*electronic record of an individual in a physician's office or clinic, which is typically in one setting and is provider-centric*", whereas an "*electronic patient record*" (EPR) is defined as an "*electronic record of an individual in a hospital or health care facility, which is typically in one organization and is facility-centric*". Given the previous two definitions, an electronic health record (EHR) is defined as follows: *"Information relevant to the wellness, health and healthcare of an individual, in computer-processable form and represented according to a standardized information model, or the longitudinal electronic record of an individual that contains or virtually interlines to data in multiple EMRs and EPRs, which is to be shared and/or interoperable across healthcare settings and is patient-centric."*

Furthermore, a personal health record (PHR) is defined by ISO/TR 14292:2012(en) as *"…a representation of information regarding, or relevant to, the health, including wellness, development and welfare of that individual, which may be stand-alone or may integrate health information from multiple sources, and for which the individual, or the representative to whom the individual delegated his or her rights, manages and controls the PHR content and grants permissions for access by, and/or sharing with, other parties."*

Based on these ISO definitions [8]-[20], the main types of medical records are:
Electronic Health Record (EHR): An EHR is a patient's medical history in digital form, replacing traditional paper charts. It includes detailed health data such as past and current medical conditions, prescribed medications, vaccination history, test results, doctor's notes, treatment strategies, and

follow-up reports. It helps the healthcare professionals securely store and share patient information throughout the larger facilities network. [1]

Electronic Medical Record (EMR): just like the EHR with the same components but only used within a single clinic or hospital, and the patient health data is not shared outside the facility. [1]

Personal Health Record (PHR): also, a documentation tool, but it is managed by the patient himself, either digitally or on paper. It includes personal details, prescribed drugs, vaccination records, family health background, and important medical reports like lab tests or imaging results. By maintaining a PHR, patients can actively manage their health information and share it with their doctors as needed. [1]

2.2 Professional Standards and Best Practices

Once a patient record is created, whether in a paper or digital form, the health provider professional is responsible for ensuring the record is stored securely and maintained appropriately.

Maintain accessibility: providers ensure patients can access their charts.
Secure Transfer: medical records should be shared securely, and both the date and the method of transfer must be documented.
Retention and Destruction: medical records must be kept for a minimum of 10 years for minors, starting when the patient turns 19; for other patients, records should be retained for 10 years from the last entry; once the retention period has ended, records must be destroyed securely.
Storage and Security: Providers must store medical records safely to protect their integrity and confidentiality. They should take precautions against theft, loss and unauthorised access by using locked cabinets and restricted access areas. Electronic records require regular backups that are stored securely and must be easily accessible when needed.
EHR: In EHR system and service provider selections, health providers should evaluate their options and choose secure, powerful systems which allow patient access via name and health number, require passwords for protection, maintain an audit trail and be available in a printable format. Additionally, EPR systems should automatically back up files and protect against data loss.

2.3 Key Components of Medical Records

Patient identification information: such as the patient's full name, date of birth, gender, contact details (phone number, address, email), and unique identifiers (e.g., health number, national ID number).
Medical history: includes past surgeries, allergies, and chronic conditions.
Medication information: All the current medications or treatments taken by the patient
Family medical history: helps anticipate potential inherited risks
Treatment history: past treatments, surgeries, and hospitalizations.
Diagnostic test results: laboratory reports (e.g., blood work, urinalysis) and imaging studies (e.g., X-rays, CT/MRI scans).

Singed consent forms: Legal documents confirming the patient's approval for specific treatments, surgeries, or participation in research trials

Clinical progress notes: Regular updates from healthcare providers detailing observations, treatments administered, and the patient's response.

Immunization records: all vaccines a patient has received

Insurance and billing details: Details about a patient's insurance coverage, billing history, and payment agreements.

2.4 Impact on Healthcare Continuity

Good clinical notes are essential for recording patient history, ensuring continuity of care, aiding healthcare decisions, and improving patient outcomes while balancing information sharing and confidentiality

Good clinical records lead to:

- Enable secure medical data sharing and facilitate interdisciplinary collaboration.
- Support consistent patient care delivery without any interruptions
- Make better clinical decisions through comprehensive patient data
- Help risk evaluation through better data accessibility
- Support root cause analysis for serious incident investigations
- Reliable documentation for legal proceedings
- Enhanced diagnostic and therapeutic planning and reducing redundant tests
- Increase efficiency through better time utilization

Where poor clinical records end up with:

- Miscommunication among healthcare teams
- Raises of medical errors with potential legal consequences
- Redundant and duplicate diagnostic tests and treatments
- Prolonged inpatient care – hospital stays
- Higher risk of critical errors.

3 Healthcare Documentation in Algeria

3.1 Current State of Medical Record Management

The Algerian healthcare system faces real challenges in medical record management, consistent reliance on paper-based systems, fragmentation problems across different levels of care, and a top-down administrative system that causes delays and creates obstacles to smooth collaboration and data sharing. Simplicity

3.1.1 Paper-based system mostly

Many developing countries have healthcare systems that still rely on paper records for managing patients' information. This use of paper creates challenges for providing continuous care and sharing vital information between different levels of healthcare:

- Healthcare providers have limited access to share patient information with each other.

- There is a higher risk of losing, damaging, or misplacing patient records because of poor archiving practices.
- ineffective workflows and time-consuming administrative processes
- Barriers to retrieving actionable insights from population health data for surveillance and planning purposes

3.1.2 Fragmentation issues

Fragmented medical records cause incomplete patient histories and medication errors. This breaks the continuity of care. Additionally, it hinders connection and collaboration among healthcare teams because patients may receive care from multiple providers, with each healthcare facility maintaining separate patient records that ultimately create multiple disconnected profiles for individuals. This fragmentation is one of the characteristics of Algeria's healthcare system, since it is a mix of public and private services, with the public sector providing most of the care.

This significant fragmentation in medical record management across its three-tier structure:

1. Primary care: Local health centres (Polycliniques, EPSP).
2. Secondary care: Regional and specialised hospitals.
3. Tertiary care: University Hospital Centres (CHU) in major cities.

This fragmentation shows up in several ways:

- **Administrative fragmentation:** Algerian health law establishes a hybrid administrative structure with shared central-local competencies but does not encourage local health institutions to seek funding and administrative independence [7]. This hybrid approach creates inconsistencies in record management practices across different healthcare levels.
- **Information silos:** even where digital systems exist, they often lack interoperability.
- **Infrastructure challenges:** inadequate infrastructure and unreliable systems make the fragmentation problem worse, making it difficult to establish a unified record management system.

3.2 Challenges in Information Sharing

Algeria's healthcare system relies on paper-based medical records and manual information-sharing practices, which lead to significant challenges in coordination across various levels.

Patient-initiated information sharing

- **Physical document carrying:** patients are frequently required to manually transport their physical medical records between healthcare providers.
- **Manual record keeping:** Patients often maintain personal copies of important medical documents. This patient-driven approach is necessary because there are no integrated electronic systems.

For the healthcare provider's side

- **Paper-based referral system:** When patients are referred from primary to specialised care, their medical information is often transferred using handwritten notes, printed diagnostic reports, and physical medical records.
- **Manual consultation processes:** healthcare providers must rely on verbal communication with patients to gather medical histories; the responsibility of an accurate medical history recall depends on the patients and their families

3.3 Paper-Based vs. Digital Systems Analysis

3.3.1 Paper base

Advantages:

❖ **Implementation & Initial Costs**
- Low initial implementation costs
- No reliance on technology infrastructure
- Familiar workflow for staff
- No software licensing fees
- Minimal training requirements

❖ **Physical Control & Reliability**
- Complete control over physical access
- No system crashes or technical failures
- Always available regardless of power outages
- No internet connectivity requirements

❖ **Documentation Depth & Quality**
- Personalized patient stories and narratives
- Subtle patient behaviors documented in free text
- Comprehensive contextual information
- Flexible documentation space for complex cases

❖ **Simple Data Management**
- No software updates or system maintenance required
- No vendor dependencies or lock-in concerns
- Simple filing and organization systems
- No data migration issues

Disadvantages

❖ **Operational Inefficiencies**
- Manual processes for search and retrieval is time-consuming
- Only one person can access records at a time
- Time-consuming filing and administrative tasks
- Difficulty finding specific information quickly
- Cannot integrate with other hospital systems
- Limited analytical capabilities for research

❖ **Storage & Space Issues**
- Requires significant physical storage space

- Storage costs increase exponentially with patient volume
- Risk of documents being lost, damaged, or destroyed
- No backup in case of natural disasters (fires, floods)

❖ **Security Vulnerabilities**

- Weak physical security - documents can be easily altered
- No tracking of who accesses or modifies records
- Easy to remove documents without detection
- Limited audit capabilities for regulatory compliance

❖ **Documentation Quality Issues**

- Handwriting often illegible leading to medical errors
- Inconsistent documentation standards across providers
- Missing information and incomplete records common
- No automated alerts for drug interactions or allergies
- Prone to transcription errors when sharing information
- No clinical decision support capabilities

❖ **Limited Collaboration & Sharing**

- Access and sharing of documents is severely limited
- Requires photocopying and physical delivery for sharing
- Delays in information transfer between providers
- Limited care coordination capabilities
- Cannot be accessed during off-site emergencies
- No patient portal access for self-service

3.3.2 Digital records

Advantages:

❖ **Long-term Cost Efficiency**

- Reduced paper and supply costs
- Lower administrative overhead long-term
- Elimination of physical storage costs
- Reduced staff time on manual processes

❖ **Advanced Data Capabilities**

- Advanced search functionality and instant retrieval
- Big data analytics for population health insights
- Research capabilities and trend analysis
- Seamless integration with lab, imaging, and billing systems
- Automated backup and disaster recovery
- Scalable storage for growing patient populations

❖ **Advanced Security & Access**

- Multi-factor authentication systems
- Comprehensive audit trails and tracking
- Role-based access permissions
- Advanced encryption protocols
- Remote access capabilities for authorized users

- Real-time access from multiple locations simultaneously

❖ **Accuracy & Clinical Decision Support**

- Improved accuracy and legibility
- Automated clinical alerts
- Evidence-based decision support tools
- Standardized workflows and consistent formatting
- Built-in prompts for required fields
- Real-time clinical guidance and reminders

❖ **Enhanced Patient Care & Safety**

- Reduced medication errors through automated alerts
- Better care coordination between providers
- Quick access to patient data in emergencies
- Patient portal access for engagement
- Telemedicine integration capabilities

Disadvantages

❖ **Implementation & Technical Challenges**

- High initial implementation and ongoing maintenance costs
- Dependence on technology and potential system failures
- Need for extensive staff training and change management
- Steep learning curve, especially for older staff
- Regular software updates and system maintenance required
- Vendor lock-in concerns and dependency issues

❖ **Technology Dependencies**

- System downtime can affect patient care
- Requires reliable internet connectivity and power
- Need for robust IT infrastructure and support
- Regular hardware upgrades and replacements needed
- Potential for technical glitches during critical moments

❖ **Cybersecurity & Privacy Risks**

- Vulnerable to cyber-attacks and data breaches
- Risk of unauthorized digital access and hacking
- Privacy concerns with cloud storage solutions
- Complex regulatory compliance requirements
- Potential for large-scale data theft affecting thousands
- Identity theft risks from digital data exposure

❖ **User Experience & Workflow Issues**

- Can reduce face-to-face patient interaction time
- Potential for over-reliance on templates reducing personalization
- Learning curve may temporarily reduce productivity
- Alert fatigue from too many automated notifications

❖ **Training & Adaptation Challenges**

- Requires extensive and ongoing staff training
- Resistance to change from staff comfortable with paper

- Generational gaps in technology adoption
- Need for continuous education on system updates
- Temporary productivity loss during transition period

3.3.3 Comparison Summary

Comparing paper-based medical records to digital medical records shows a clear advancement in how we handle healthcare documentation. Paper records are simple and cheaper to start with, but digital systems perform better in most key areas. Electronic medical records can be retrieved faster. Which significantly enhances efficiency in clinical work and improves the quality of patient care.

Digital records are better at being accurate, secure, and able to work with different systems, which helps improve patient care and cut down on medical errors. However, switching to digital records needs a lot of money and technical know-how upfront, which can be difficult for smaller healthcare facilities [9]. The long-term benefits include better efficiency, reduced healthcare costs through preventative care, and improved coordination of health services. [10]
Digital systems are very important for modern healthcare. They make it easier to access patient information and ensure that records are complete and useful for analysis. Although there are some challenges in putting these systems in place, evidence shows they help provide better and more coordinated care for patients today [11]. Healthcare organizations should focus on digital transformation. They also need to tackle cost and training challenges to fully benefit from this change. [12]

4 Family Caregivers in Algerian Healthcare

4.1 Role definition and responsibilities

4.1.1 Definition

Also known as "family caregivers," these individuals are often family members or friends who regularly provide consistent assistance and care to their relatives who are sick or have disabilities. They help with activities of daily living and typically do this work on an unpaid basis, without formal training to provide these services.
Key characteristics of modern family caregiving:

- Taking care of family members now requires more time, effort, and skill than in the past.
- Caregivers often do not get enough training for this role [13].
- Most care work happens at home.
- Some caregivers work at least 80 hours of care per month [14].
- Caregiving for a long period of time
- Women do it more than men
- Cuts into personal time: Unpaid caregivers' effort that positive activities in their daily lives are reduced by 27.2% [15]
- Dealing with serious health problems and requiring ongoing healthcare attention Lack of training and support

4.1.2 Responsibilities

As the key person providing daily help, caregivers must handle multiple responsibilities:

- Daily Medication Management: Caregivers are often responsible for organising medication schedules, ensuring treatment adherence, managing prescriptions, providing reminders, and monitoring for side effects [16].
- Household management duties: Caregivers often take on responsibilities such as light housekeeping, laundry, and maintaining a clean, safe, and comfortable living environment.
- Transportation entails driving to medical appointments.
- Coordinating between healthcare providers.
- Maintain and preserve the personal record of the patient.

The difficulty of these tasks depends on the patient's condition. Patients with chronic conditions need more help managing their medications and following specific lifestyles and diets. Older individuals also present different challenges for caregivers based on their specific situations,

4.1.3 Learning through hands-on-experience

Family carers typically develop their skills by trying things out and making mistakes. Over time, they become more familiar with managing medications and following treatment plans.

This learning process helps build confidence in important tasks; they learn to operate medical equipment, identifying health changes and responding effectively to emergency situations.

Caregivers often learn by watching healthcare professionals during hospital visits. They ask questions during medical appointments and adapt techniques based on what works best for their family member. Education helps care partners perform tasks better. The more knowledge they have, the better they can do their responsibilities.

This informal way of learning can be stressful and risky. Caregivers may not get clear training on the right procedures or safety rules.

4.2 Challenges and Constraints

4.2.1 Primary Challenges Faced by Family Caregivers

During their caregiving duties, carers often experience a significant burden, accompanied by anxiety, stress, and sometimes depression. This burden manifests across several key areas.

a) **Physical and Health Challenges:** The physical demands of caregiving—particularly for patients who are highly dependent or suffer from multiple chronic conditions—can significantly impact caregivers' own health and well-being [17].

b) **Lack of Choice and Autonomy:** Nearly half of all caregivers report having no choice in taking on the caregiving role. This involuntary assumption of responsibilities often leads to psychological distress and frustration [13].

c) Behavioural and Cognitive Challenges: Caregivers experience increased stress when patients require extensive physical and mental support. Issues such as verbal aggression and sleep disturbances further intensify the caregiver's burden [18].

d) **Limited access to support services:** In many developing countries there is no system in place to address informal caregivers' concerns, needs, gaps, and barriers.

4.2.2 Economic Constraints and Caregiving Capacity

Family caregivers often face serious financial challenges. They need to provide support while dealing with their own economic issues; they have to manage both the direct costs and the opportunity costs from reduced employment participation. Lower-income caregivers struggle to give quality care while meeting their own basic needs. They are often forced to choose between delivering proper care and maintaining their financial stability.

4.2.3 Physical, Emotional, and Social Consequences

The burden on caregivers affects many areas of their lives: [19]

a) **Mental health consequences:** Providing care for individuals with medical conditions—especially mental illnesses—can lead to a range of negative emotions, including anger, frustration, low self-esteem, persistent worry, and feelings of helplessness.
b) **Social isolation:** Caregiving responsibilities often lead to reduced social interactions and activities, as carers must sacrifice personal time to meet the needs of the patient.
c) **Psychological distress:** the combination of physical demands, emotional stress and social isolation creates psychological challenges that can lead to depression, anxiety and burnout.

4.2.4 Conclusion

Family caregivers face many challenges that come from individual, cultural and system factors. To help them, we need to use techniques that understand the many sides of caregivers' stress. Effective support should meet their immediate needs and also push for long-term changes to make caregiving easier and sustainable.

5 Mobile Health Technology Landscape

The global mobile health market is growing quickly due to more people using smartphones and the potential of digital health solutions. Mobile apps provide promising ways to help people stick to their medication plans, which is a major challenge in healthcare worldwide.

This study looks at popular health apps that support caregivers, track medications, and manage medical records. We aim to identify useful features and existing gaps, especially for developing healthcare systems like those in Algeria, where digital health is still developing.

To meet this goal, we evaluated eleven popular apps based on six important criteria: integration with electronic personal medical records, tools for caregiver support, medication tracking, telehealth options, support for multiple languages, and offline access. We chose these features because they relate to common medication adherence problems in healthcare settings with limited resources.
Methodology:
Apps were chosen if they:

- Were updated after 2020.
- Support caregivers or track health/medication.
- Use modern tools like AI or electronic personal medical record integration.
- Have good user reviews (4 stars or more).

5.1 Existing Applications Review

5.1.1 Comprehensive Health Management Platforms

Mon Espace Santé [21]

Mon Espace Santé is a French national digital health platform that consolidates patients' health records, including prescriptions, test results, vaccination histories, and consultation reports. It provides access control features for secure information sharing between patients and health professionals. The platform supports emergency data availability and appointment scheduling.

Figure 1.1: *Mon espace sante app*

Practo [22]

Practo is a digital health platform that offers a combination of services, including appointment booking with doctors, online consultations, and access to digital health records. It supports uploading medical documents, tracking past consultations, and maintaining personal health histories. The app is designed to help users navigate the healthcare system efficiently.

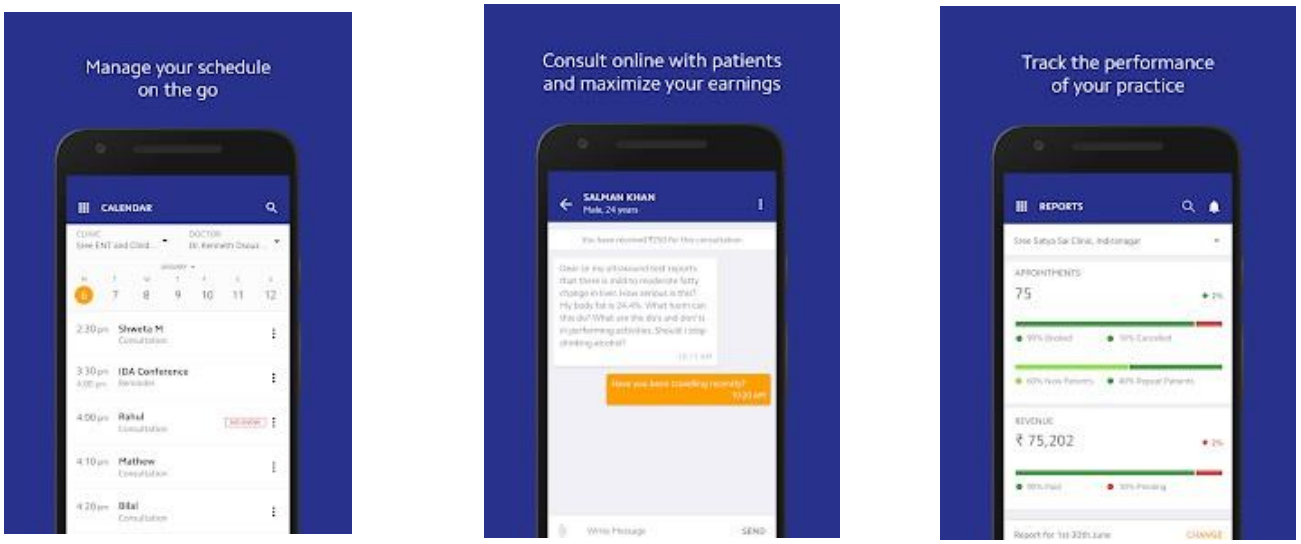


Figure 1.2: *Practo app*

CareClinic [23]

CareClinic is a health management application that offers an integrated platform for tracking medication, symptoms, activities, nutrition, and therapy. It includes features such as customizable health journals, mood tracking, pill reminders, and reporting tools. Users can generate logs and share progress reports with healthcare professionals to improve clinical oversight.

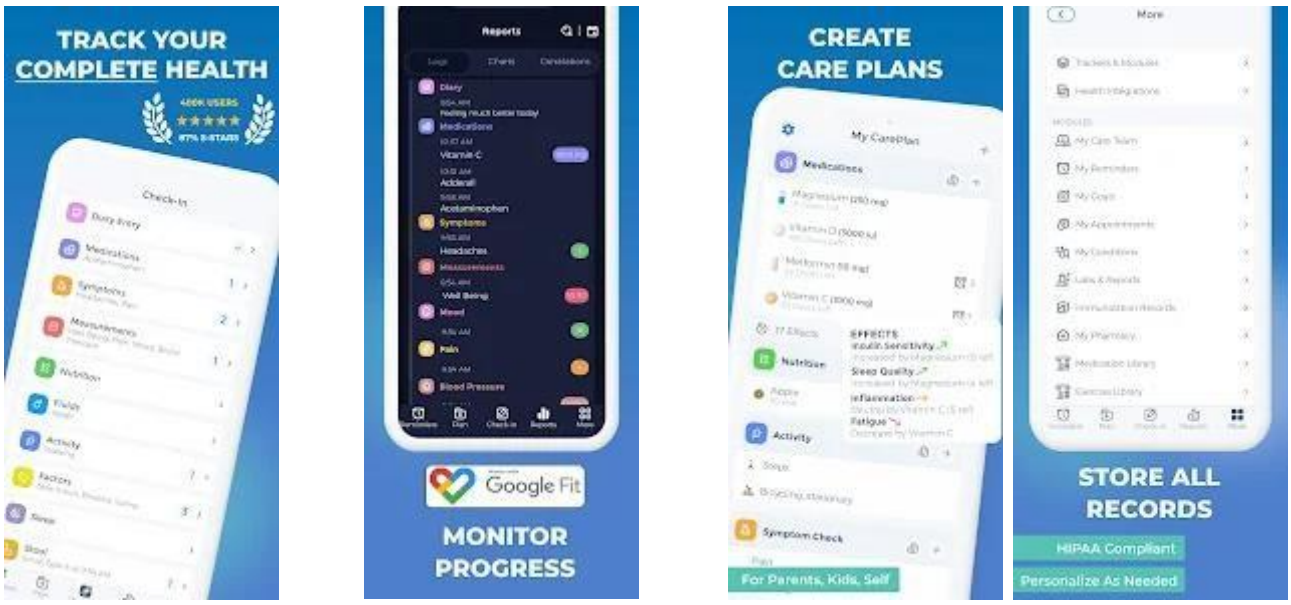


Figure 1.3: *CareClinic app*

5.1.2 Medication Management Applications

MediSafe [24]

MediSafe is a mobile application designed for medication management. It provides features such as dose reminders, interaction alerts, refill notifications, and a platform to allow users to log their medication intake. The app also enables caregiver integration, where designated individuals can monitor adherence in real time, thus promoting treatment continuity.

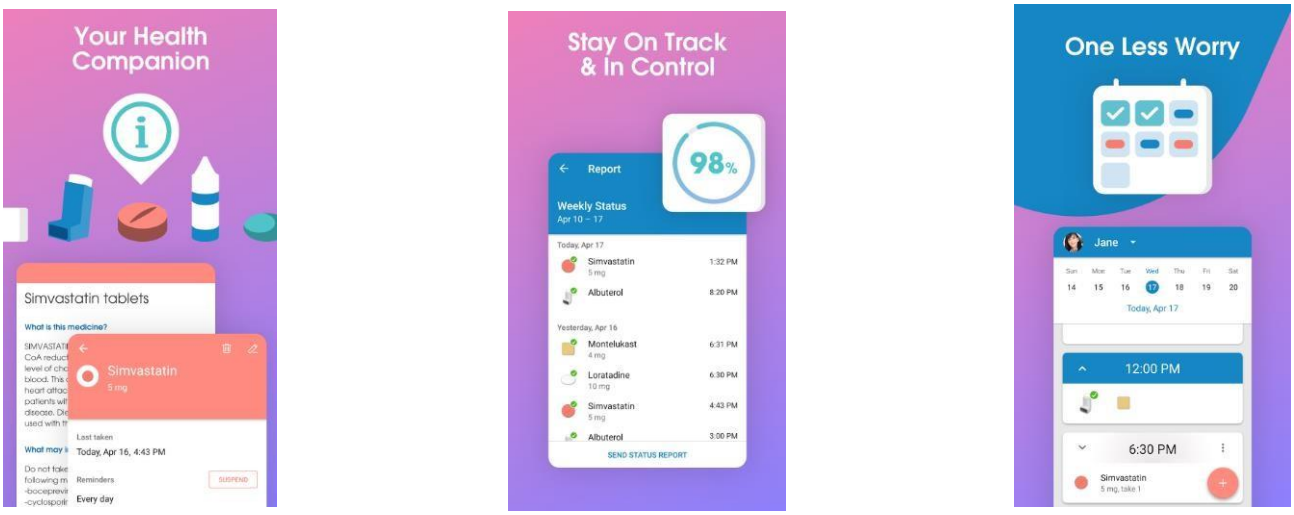


Figure 1.4: *Medisafe app*

CareZone[25]

CareZone is a medication management application that enables users to scan prescriptions, organize medication lists, set reminders, and track dosage intake. It also includes a journaling feature to monitor symptoms and a calendar for scheduling doctor appointments and refills

.

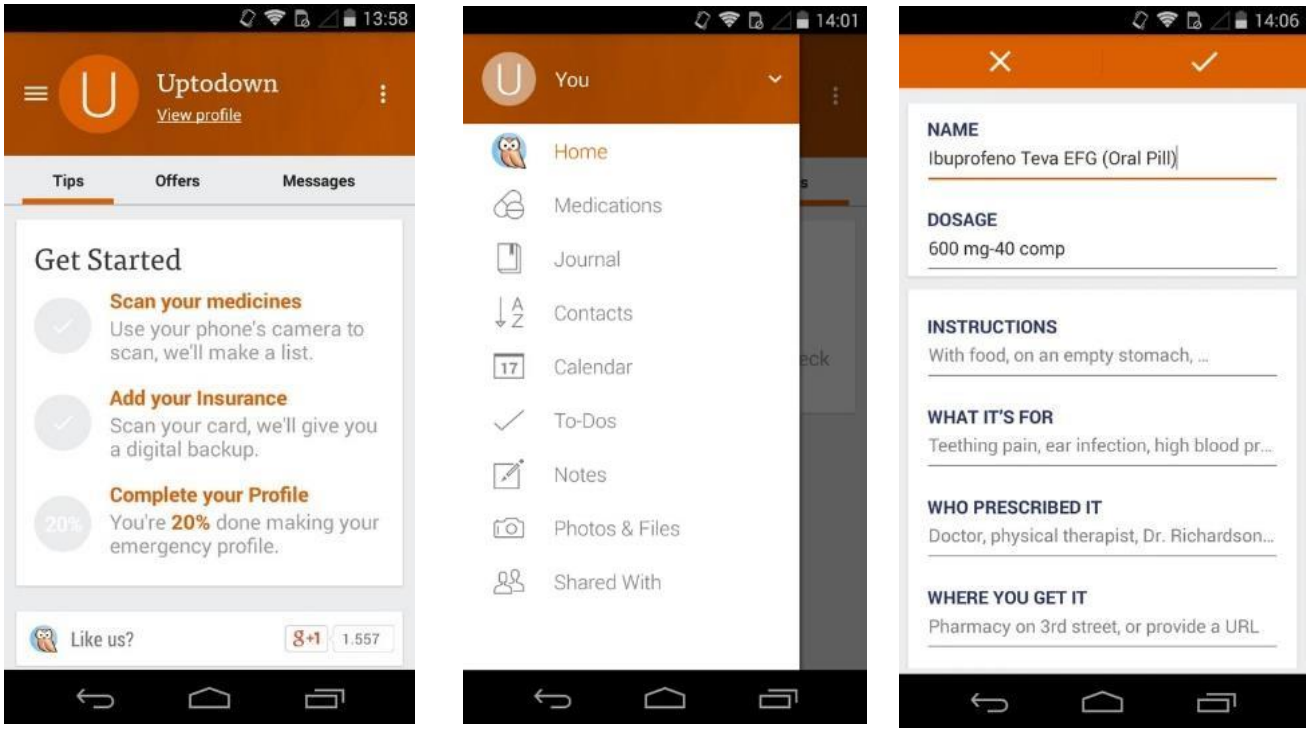
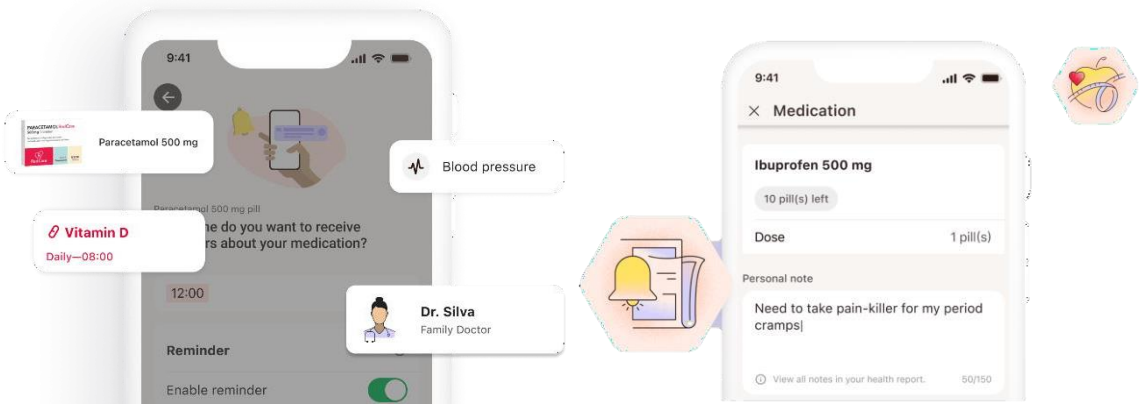


Figure 1.5 : *CareZone app*

MyTherapy [26]

MyTherapy is a medication reminder and health tracker app. It supports tracking of pill intake, vital parameters, and exercise routines. The app includes features for setting up custom alarms, maintaining a therapy logbook, and sharing health summaries with physicians or caregivers.



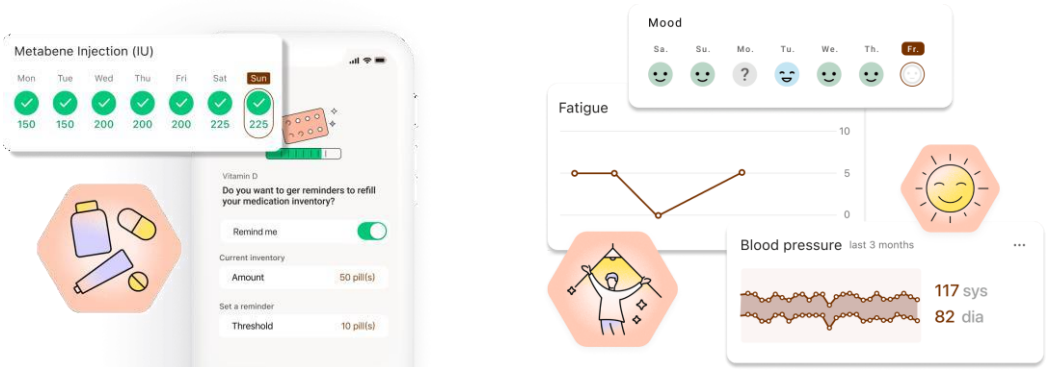


Figure 1.6 : *MyTherapy app*

MedHelper [27]

MedHelper is a comprehensive medical management app that allows users to schedule reminders for medication, doctor appointments, and tests. It supports health parameter tracking and offers a detailed log of medication history. The app includes the ability to manage prescriptions and access health-related notes.

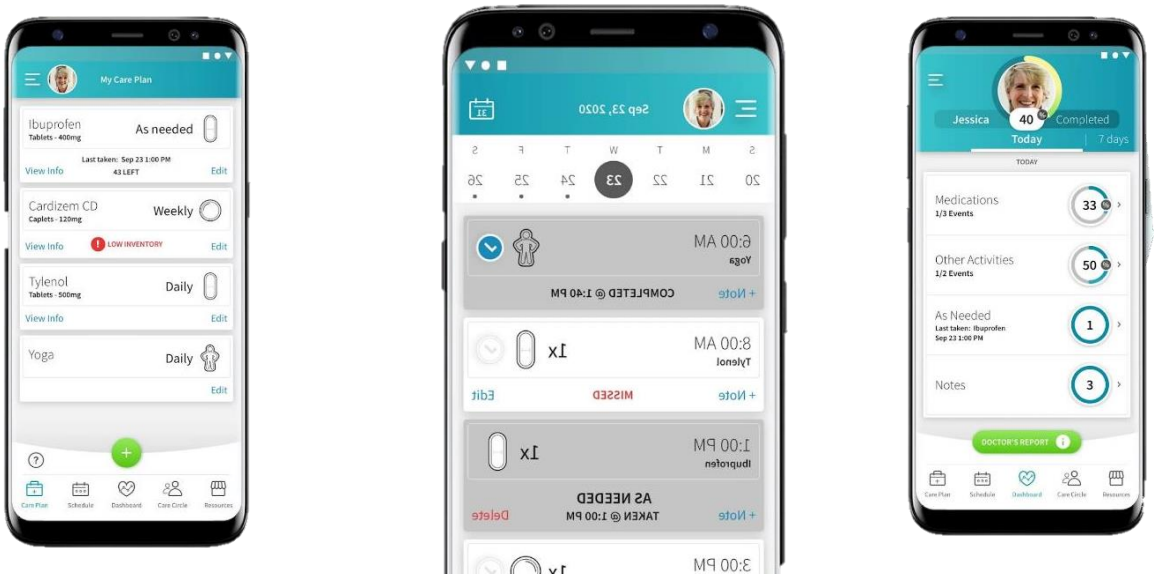


Figure 1.7: *MedHelper app*

5.1.3 Smart Medication Dispensing Solutions

Pillo [28]

Pillo is a smart pill dispenser supported by an application interface. It assists users in adhering to their medication schedules through reminders, voice commands, and automated dispensing of medication. The system includes user authentication features and can notify caregivers or family members of missed doses.

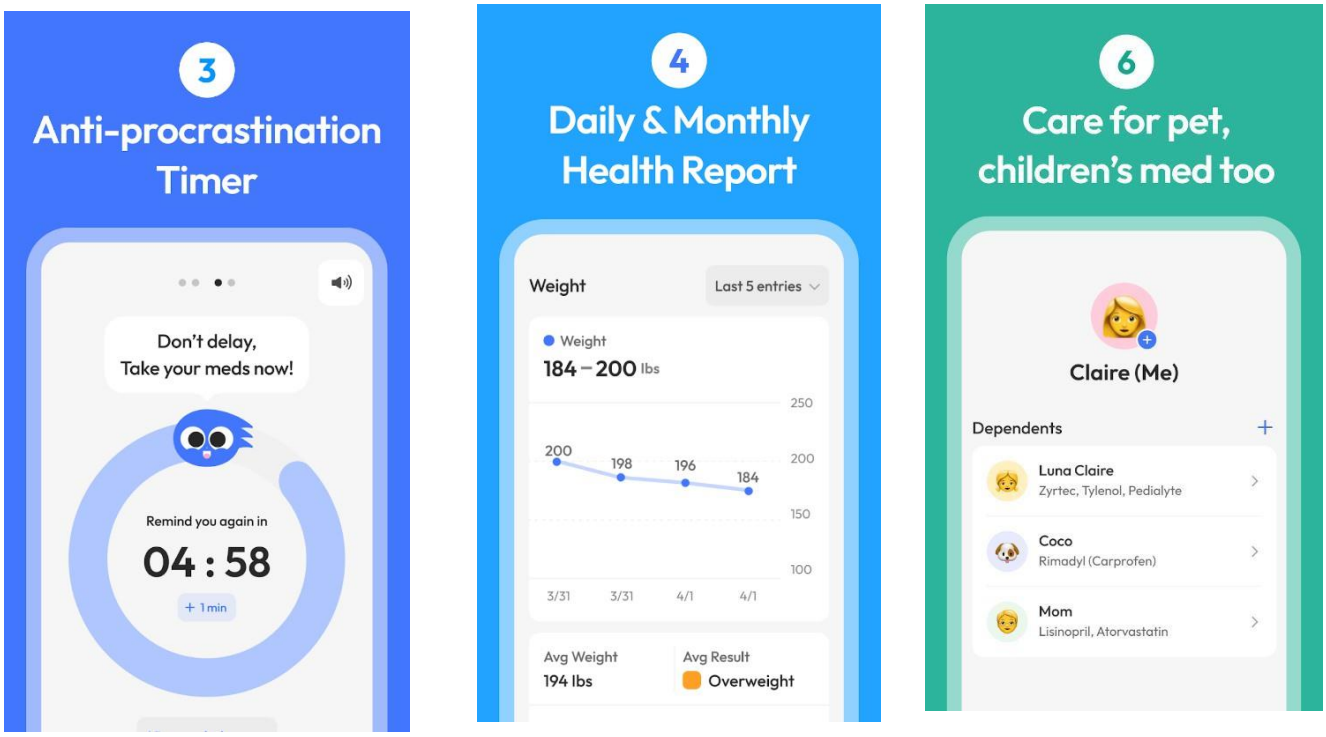


Figure 18: *Pillo app*

5.1.4 Specialized Care and Support Platforms

iCompanion [29]

iCompanion is a digital health assistant aimed at supporting people with Alzheimer's and related conditions. It allows tracking of daily health indicators, moods, symptoms, and cognitive function. The app also offers calendar features, journaling tools, and generates health reports that can be shared with medical professionals.

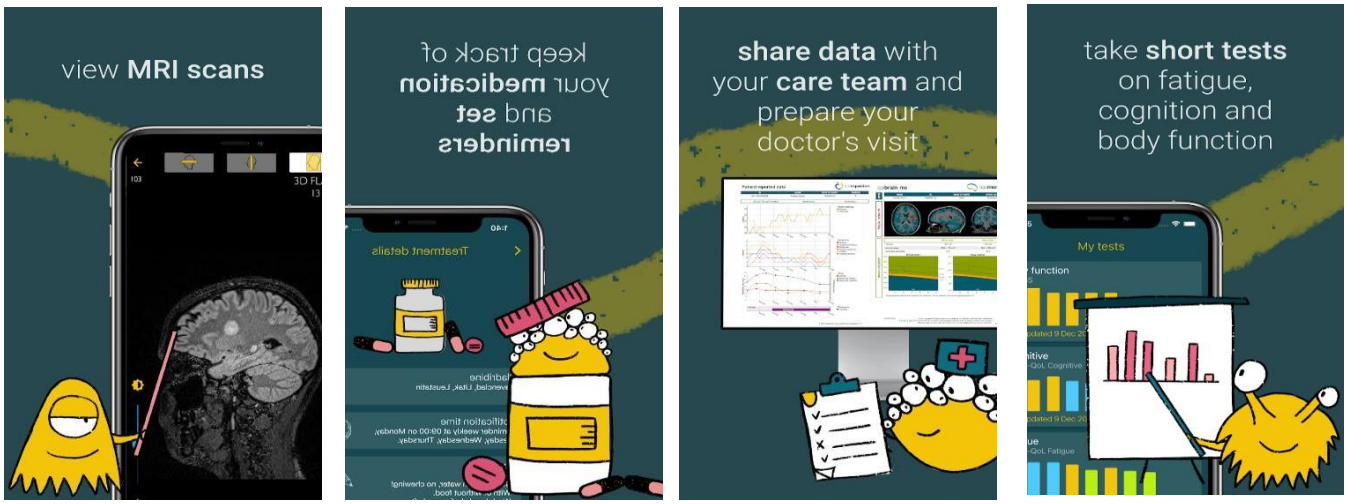


Figure 1.9: *iCompanion app*

E-aidants [30]

E-aidants is a caregiver-oriented web platform developed in France to support informal caregivers of elderly and dependent individuals. It centralizes access to health services, informational content, and social resources. The platform facilitates task management, communication with professionals, and navigation through administrative procedures.

5.1.5 Patient Community and Support Networks

Carenity [31]

Carenity is an online social platform dedicated to patients with chronic conditions and their caregivers. It provides a space for sharing experiences, discussing disease management strategies, and accessing research-backed information. The app fosters peer support and community-based interaction among individuals with similar health conditions.

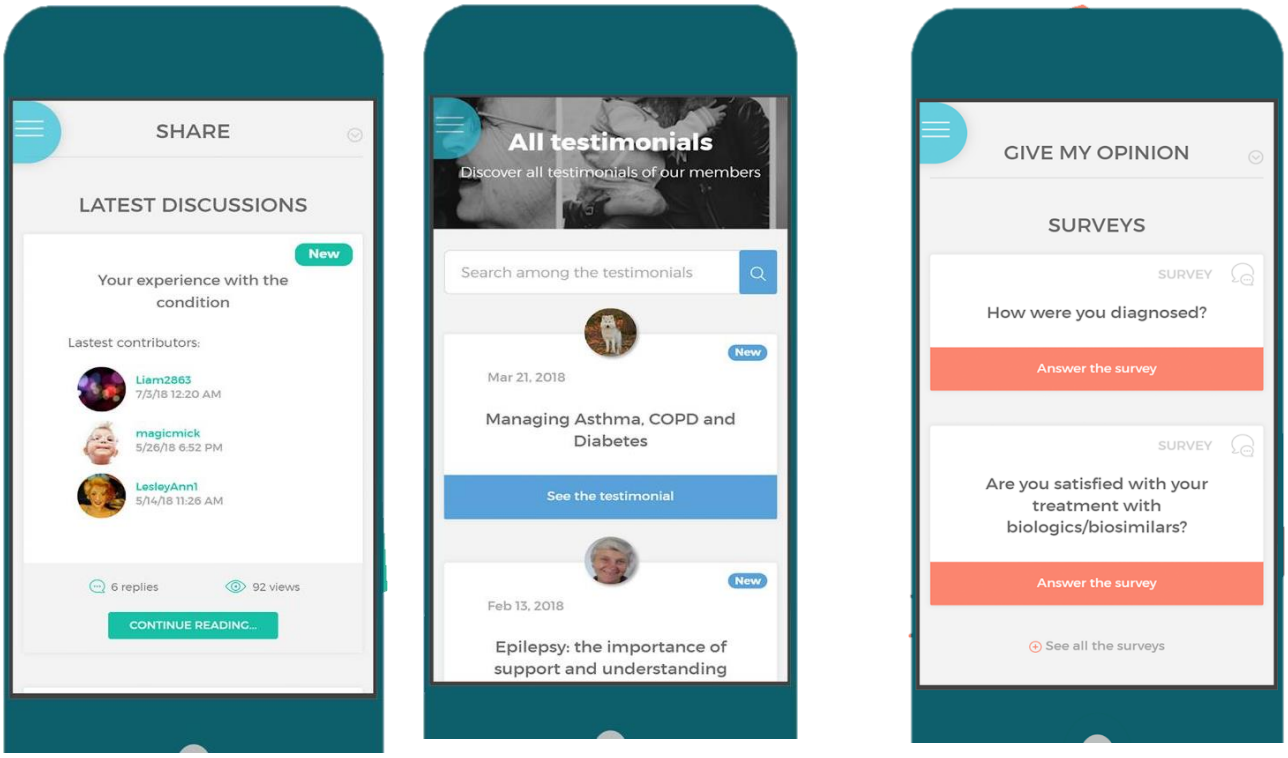


Figure 10: *Carenity app*

5.2 Critical analysis

5.2.1 Medication Management

Several mobile applications effectively support medication adherence through reminders and logs. **MediSafe** and **MyTherapy**, for example, provide intuitive interfaces for tracking doses, issuing alerts, and generating adherence reports. but both apps lack EPR features and local language options.

CareZone adds convenience with medication scanning features but also suffers from a lack of interoperability with healthcare systems and does not support Arabic or French. Similarly, **Pillo** and **MedHelper** focus exclusively on individual medication tracking, overlooking caregiver

collaboration or broader treatment coordination—critical features in multi-patient household settings.

5.2.2 Caregiver Collaboration

Apps like **Carenity** and **E-aidants** lacks EPR features and local language options.
iCompanion and **CareClinic** help with symptom tracking but miss key features like offline mode or Arabic menus.

5.2.3 EPR and Telehealth Integration

Mon Espace Santé is strong in EPR management but only works in France and is hard to use for people with low tech skills.
Practo includes digital records and doctor booking, but it's not designed for low-resource settings.

5.2.4 Regional Adaptability

Most global apps are designed for high-connectivity regions and tech-literate users. **CareClinic**, **MyTherapy**, and **Practo** do not offer Arabic or French interfaces, and few apps support offline usage.

App Name	EMR Integration	Multilingual (Arabic/French)	Offline Support	Caregiver Tools	Medication Management	Telehealth
MediSafe	✗	✗	✗	✓	✓	✗
MyTherapy	✗	✗	✗	Limited	✓	✗
Mon Espace Santé	✓	✗	✗	Limited	✓	✓
Practo	✓	✗	✗	✗	✓	✓
CareClinic	✗	✗	✗	Limited	✓	✗
Carenity	✗	✗	✗	✓	Limited	✗
MedHelper	✗	✗	✗	✗	✓	✗
CareZone	✗	✗	✗	✓	✓	✗

Table 1.1: *Healthcare apps feature comparison*

5.3 Key Gaps and Opportunities

From the analysis, we see that:

- No app offers electronic personal record + caregiver tools + Arabic + offline mode.
- Current applications do not leverage AI for predictive healthcare analytics or intelligent medication management, representing a significant missed opportunity for improving care outcomes.
- Few applications offer robust offline functionality, despite the reality of intermittent internet connectivity in many developing regions.
- Most applications are designed for developed healthcare systems and lack the cultural sensitivity and language support required for the markets of developing countries.

MedGarde aims to address these gaps by combining EPR integration, multilingual interfaces, medication tracking, and offline capability—all within a caregiver-centric design.

5.4 Regional Adaptation Requirements

In Algeria, healthcare is still mostly on paper. Most people don't use digital health tools because they don't exist in Arabic, are hard to use, or need internet

6 Conclusion and Bridge to Technical Development

Key findings:

- Algeria's healthcare system relies heavily on paper-based records, causing fragmentation across care levels
- Family caregivers play a crucial role but lack proper support and training
- Current mobile health apps don't meet the specific needs of Arabic-speaking users in resource-limited settings
- No existing solution combines EPR integration, caregiver support, and offline functionality

Identified Gaps:

- Lack of integrated digital health solutions for developing countries
- Missing Arabic/ language support in health apps
- Limited offline functionality for areas with poor connectivity
- Insufficient caregiver-focused features in existing applications

Implications for MedGarde Development:

These findings justify the need for a comprehensive, culturally-adapted mobile health solution that addresses the unique challenges of the Algerian healthcare context.

This literature review establishes the foundation for developing MedGarde by identifying critical gaps in current healthcare documentation and mobile health solutions. The analysis reveals that Algeria's healthcare system requires a solution that combines:

Technical Requirements

- Electronic personal record integration capabilities
- Offline functionality for limited connectivity
- Multi-language support (Arabic/French/English)
- Caregiver collaboration features

User-Centred Design Needs

- Simple interfaces for low-tech literacy users
- Cultural adaptation for MENA region
- Family-centred care workflows
- Resource-constrained environment optimization

Chapter 2 will translate these identified needs into concrete technical specifications, system architecture, and design requirements that address the gaps identified in this literature review. The architectural design will demonstrate how MedGarde can bridge the current limitations while providing a practical, culturally-appropriate solution for Algerian healthcare stakeholders.

Chapter 2:
Design and modelling

1 Introduction and Development Strategy

The design and modeling phase of the MedGarde application establishes the blueprint for development, translating the identified healthcare challenges in Algeria into concrete system specifications. This chapter outlines the systematic approach to requirements gathering, architectural design decisions, and development methodology.

1.1 Modular Development Approach

While the MedGarde vision encompasses a comprehensive healthcare management system as described in Chapter 1, this thesis implements a **modular development strategy**. Rather than attempting to build the entire system at once—which would require extensive resources and time beyond the scope of this master's project—we focus on developing core functional modules that provide immediate value to users while establishing a foundation for future expansion.

The initial implementation prioritizes the following key modules:

- User and patient account management
- Medical file management and sharing
- Doctor directory and appointment scheduling

This phased approach offers several advantages:

- It allows for faster delivery of a functional minimum viable product (**MVP**)
- It enables early user testing and feedback collection
- It establishes a solid architectural foundation for subsequent modules
- It aligns with available development resources and project timeline constraints

The requirements specified in this chapter reflect this modular approach, with clear differentiation between core features targeted for initial implementation and future modules planned for subsequent development phases.

2 Requirements Engineering

This section details the functional and non-functional requirements for the MedGarde application, with focus on the initial implementation's core modules: user and patient account management, medical file management and sharing, and doctor directory and appointment scheduling.

2.1 Functional requirements

User Registration and Authentication

- The system shall provide a secure registration form for health caregivers using email and password
- The system shall validate email format and password strength during registration
- The system shall implement secure authentication mechanisms
- The system shall provide a "Forgot Password" feature that sends a secure reset link to the user's email
- The system shall allow users to logout

User Profile Management

- The system shall allow users to create and edit their profiles with personal information
- The system shall store and display user profile information including: Full name, Date of birth, Gender, Blood type, Address, Phone number and Email

- The system shall enable a single caregiver to manage multiple patient profiles
- The system shall display a dashboard showing all patients linked to the caregiver's account

Patient Profile Creation and Management

- The system shall allow caregivers to create multiple patient profiles
- The system shall allow linking multiple caregivers to a single patient profile with appropriate permissions
- The system shall store patient personal information (name, age, gender, blood type, address, contact details, relationship to caregiver)
- The system shall allow caregivers to update patient profile information
- The system shall support sharing responsibility of a patient to another caregiver
- The system shall allow caregivers to remove their responsibility for a patient with appropriate confirmation

Health Record Management

- The system shall allow manual entry of basic health records with structured data fields
- The system shall permit uploading digital medical files in common formats (PDF, JPG, PNG)
- The system shall support capturing and digitizing paper medical documents
- The system shall allow updating metadata of stored medical files
- The system shall categorize by specialty and tag medical documents by type
- The system shall support searching within stored documents using multiple criteria: document name, date range, document type, medical speciality, and doctor's name.
- The system shall enable secure sharing of medical files with specified doctors
- The system shall require confirmation before deletion of any medical file

Appointment Scheduling

- The system shall allow creating new appointments with required details (doctor, location, date, time, additional notes)
- The system shall provide a calendar view of all scheduled appointments
- The system shall visually distinguish between upcoming and past appointments
- The system shall send notifications for upcoming appointments (reminders)
- The system shall allow editing or cancellation of upcoming appointments

Doctor Directory

- The system shall maintain a list of healthcare providers (doctors)
- The system shall store doctor details: name, speciality, contact information and address
- The system shall allow updating doctor details
- The system shall enable adding new doctors to the directory
- The system shall support searching the doctor directory by name, specialty, or location
- The system shall enable removing a doctor from the directory with confirmation
- The system shall link doctors to associated medical files and appointments for quick reference

2.2 Non-Functional Requirements

Usability

- The application shall provide an Arabic-first interface design
- The application shall support right-to-left text direction
- The application shall use icon-based navigation for users with low digital literacy
- The system shall feature an intuitive and user-friendly interface
- The application shall implement a consistent and intuitive user interface
- The application shall provide clear error messages and recovery options
- The application shall reduce the number of steps required to complete common tasks
- The application shall include contextual help and tooltips for complex features

Performance

- The application shall load initial screens within 3 seconds on target devices
- The application shall load patient data and medical reports within 3 seconds.
- The application shall display search results within 2 seconds
- The application shall complete document scanning within 5 seconds
- The application shall operate efficiently on devices with minimum 2GB RAM
- The application shall require less than 100MB of initial storage space
- The application shall function on devices with Android 9.0 (API level 28) and above
- The application shall process user inputs with no perceptible delay
- The application shall maintain responsiveness during synchronization operations
- The application shall optimize battery consumption during background operations
- The application shall minimize network data usage for key functions

Reliability and Availability

- The application shall be available for use 24/7, excluding maintenance periods
- The application shall recover gracefully from crashes, preserving user data
- The application shall provide offline access to previously loaded patient data
- The application shall synchronize local changes with the server
- The application shall preserve user data in case of unexpected termination
- The application shall implement data validation to prevent inconsistent states

Security/Data Protection

- The application shall encrypt all sensitive data at rest and in transit.
- The application shall implement secure authentication with industry-standards
- The application shall request minimal device permissions necessary for operation
- The application shall provide transparent privacy settings and data usage information
- The application shall allow users to delete their account and associated data
- The application shall comply with relevant Algerian data protection regulations
- The application shall implement secure file sharing with end-to-end encryption
- The application shall anonymize data used for analytics
- The application shall limit data collection to essential information

Maintainability

- The application shall follow Flutter best practices for code organization
- The application shall maintain consistent coding standards
- The application shall include appropriate documentation
- The application shall implement modular architecture

- The application shall separate business logic from presentation layer
- The application shall include comprehensive documentation for code and APIs
- The application shall implement automated testing with minimum 80% code coverage
- The application shall undergo user acceptance testing with target audience
- The application shall be tested on multiple device configurations
- The application shall implement automated testing for critical paths
- The application shall log errors and exceptions

2.3 Technical Constraints

Development Constraints

- The application shall be developed with cross-platform compatibility
- The application shall implement a modular architecture to support future expansion

Integration Constraints

- The system shall be designed with standardized APIs to allow future integration with EMR systems
- The system shall implement file format validation for uploaded medical documents
- The system shall support standard calendar integration protocols

Deployment Constraints

- The initial version shall be deployed on Android platform only
- The application must function in environments with intermittent internet connectivity
- The server infrastructure must comply with Algerian data residency requirements

3 System Analysis and Design

3.1 Use Case Analysis

3.1.1 User Role Definition

The system supports two user types:

- **User:** A general authenticated user who can manage their own account.
- **Guard:** A user with extended privileges to manage patients, appointments, doctors, and medical records.

Actor	Description/Role
User	A general person creating an account or logging in
Guard	A more privileged user who manages patients, doctors, and files
Patient	Has an account managed by a Guard , doesn't initiate use cases
Doctor	Added/edited/deleted by the Guard , not directly using the system

Table 2.1: *App actors and their roles*

Notes:

- All Guards are Users, but not all Users are Guards.
- Patient login is planned for future use (e.g., when a patient becomes responsible for their own care).

3.1.2 Use Case Specifications

Each use case be ow follows the format:

- Name
- Goal
- Actor
- Precondition
- Steps
- Alternative
- Outcome

a) User Account Management Use Cases

❖ Use Case: Create User Account	
Goal:	Register a new user in the system.
Actor:	User.
Precondition:	User has a valid email.
Steps:	1. Enter name, birthday, gender, blood type, phone, and address. 2. Enter email, password. 3. Confirm password. 4. Submit form.
Alternative:	– Email already used → error message. – Password mismatch → error message. – Missing fields → error message.
Outcome:	➢ User account is created ➢ Redirected to login with success message
❖ Use Case: Login	
Goal:	Allow access to the app.
Actor:	User.
Precondition:	User has a valid account.
Steps:	1. Enter email and password. 2. Submit login form.
Alternative:	– Invalid credentials → show error.
Outcome:	➢ Logged in and redirected to dashboard
❖ Use Case: Update User Account	
Goal:	Modify personal profile data.
Actor:	User.
Precondition:	User is authenticated.
Steps:	1. Open profile. 2. Edit fields. 3. Confirm password (if changed). 4. Save changes.
Alternative:	– Password mismatch → error. – Missing fields → error.
Outcome:	➢ Account updated with success message.
❖ Use Case: Logout	

Goal:	Securely exit the app.
Actor:	User.
Precondition:	User is authenticated.
Steps:	1. Navigate to settings. 2. Click “Logout”.
Alternative:	None.
Outcome:	➢ Session ends, redirected to login.

b) Patient Management Use Cases

❖ Use Case: Create Patient Account	
Goal:	Register a patient under guard’s care
Actor:	Guard.
Precondition:	Patient is not yet in the system.
Steps:	1. Go to Patients page. 2. Click “Add Patient”. 3. Enter name, birthday, gender, etc. 4. Enter email, password, confirm password. 5. Save.
Alternative:	– Email already used → error message. – Password mismatch → error message. – Missing fields → error message.
Outcome:	➢ Patient added with confirmation.
❖ Use Case: Update Patient Account	
Goal:	Edit patient info including relation name.
Actor:	Guard.
Precondition:	Patient exists under the guard.
Steps:	1. Select patient from list. 2. Edit profile fields. 3. Save changes.
Alternative:	– Incomplete fields → error.
Outcome:	➢ Profile updated and confirmed.
❖ Use Case: Relieve Patient Responsibility	
Goal:	Unassigned a patient.
Actor:	Guard.
Precondition:	Patient exists under guard’s list.
Steps:	1. Select patient. 2. Click “Remove”. 3. Confirm.
Alternative:	Cancel → no change.
Outcome:	➢ Patient removed, confirmation shown.

c) Doctor Management Use Cases

❖ Use Case: Add Doctor

Goal:	Add new doctor information
Actor:	Guard.
Precondition:	None.
Steps:	1. Go to Doctors section. 2. Click “Add Doctor”. 3. Enter details. 4. Save.
Alternative:	Incomplete data → error.
Outcome:	Doctor added with message.
❖ Use Case: Update Doctor	
Goal:	Modify existing doctor information.
Actor:	Guard.
Precondition:	Doctor exists.
Steps:	1. Select doctor. 2. Click “Update”. 3. Change fields. 4. Save.
Alternative:	– Incomplete fields → error.
Outcome:	➢ Doctor updated.
❖ Use Case: Delete Doctor	
Goal:	Remove a doctor profile.
Actor:	Guard.
Precondition:	Doctor exists.
Steps:	1. Select doctor. 2. Click “Delete”. 3. Confirm.
Alternative:	Cancel → no change.
Outcome:	➢ Doctor removed.
❖ Use Case: Add Appointment.	
Goal:	Schedule a patient appointment
Actor:	Guard
Precondition:	Doctor exists.
Steps:	1. Go to Appointments 2. Click “Add Appointment” 3. Enter data 4. Save
Alternative:	– No doctor → prompt to add. – Incomplete fields → error.
Outcome:	➢ Appointment created, reminder set.

d) Medical File Management Use Cases

❖ Use Case: Add medical file.

Goal:	Upload a patient's medical history.
Actor:	Guard.
Precondition:	None.
Steps:	1. Open EMR. 2. Click "Add File". 3. Fill required fields. 4. Attach file. 5. Save.
Alternative:	Missing info → error.
Outcome:	File stored with confirmation.
❖ Use Case: Update medical file.	
Goal:	Modify an existing file.
Actor:	Guard.
Precondition:	File exists in EMR.
Steps:	1. Select file. 2. Click "Delete" then confirm
Alternative:	– Incomplete info → error
Outcome:	➢ File updated.
❖ Use Case: Delete medical file.	
Goal:	Permanently delete a file.
Actor:	Guard.
Precondition:	Doctor exists.
Steps:	4. Select doctor. 5. Click "Delete" then confirm
Alternative:	Cancel → no change.
Outcome:	➢ File deleted.
❖ Use Case: Download medical file.	
Goal:	Save file locally.
Actor:	Guard.
Precondition:	File exists in EMR.
Steps:	1. Select file. 2. Click "Download".
Alternative:	– None.
Outcome:	➢ File saved to device, success message shown.
❖ Use case: Share medical file.	
Goal:	Share medical file with doctor
Actor:	Guard.
Precondition:	– File exists in EMR & Doctor exists.
Steps:	1. Select file. 2. Click "Share". 3. Select doctor. 4. Sent.
Alternative:	– No doctor → prompt to add.
Outcome:	➢ File shared with the doctor.

Table 2.2: *User Cases description*

3.1.3 Use Case Diagram

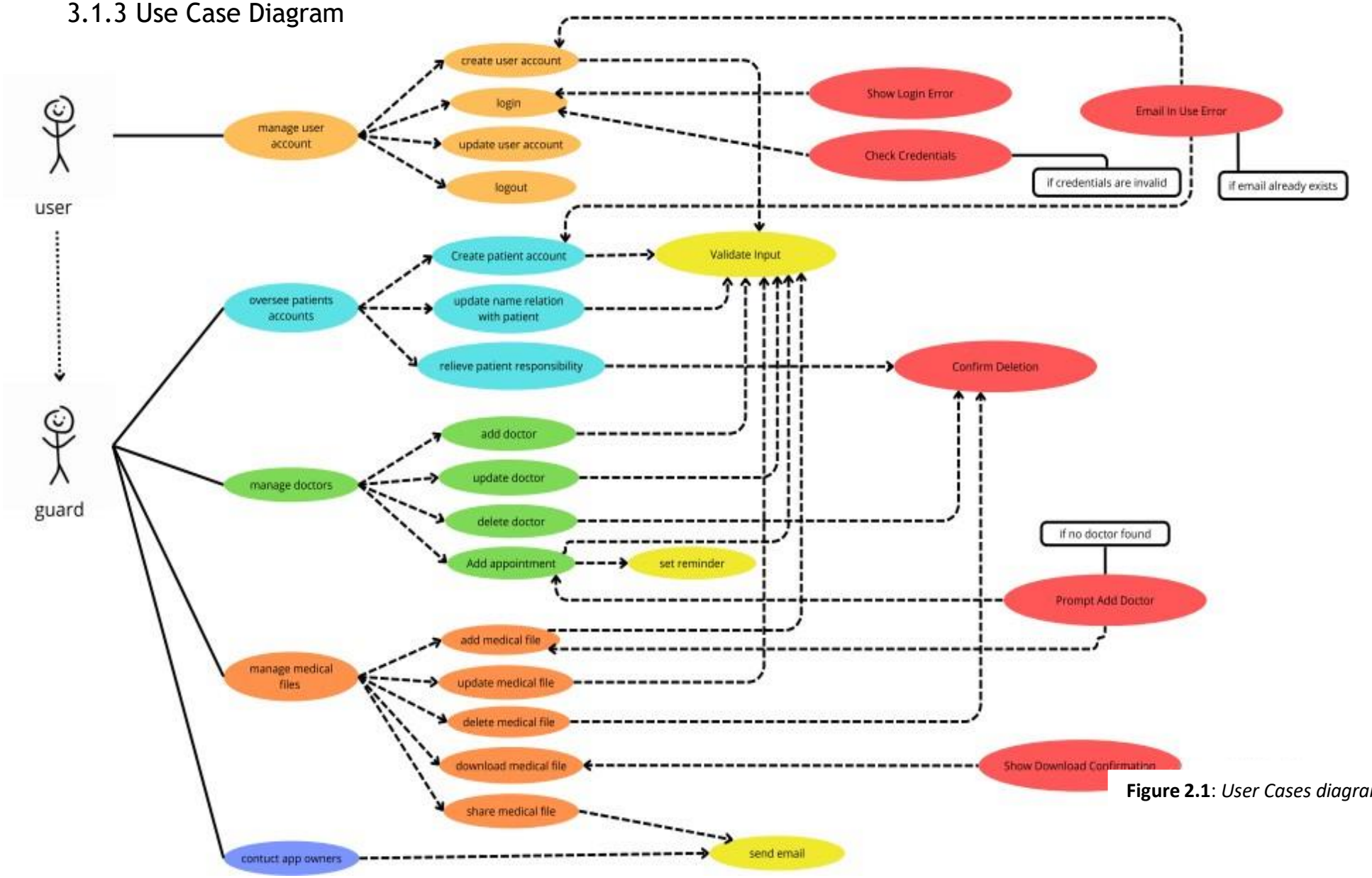


Figure 2.1: *User Cases diagram*

3.2 Data Modelling

3.2.1 Conceptual Data Model (MCD)

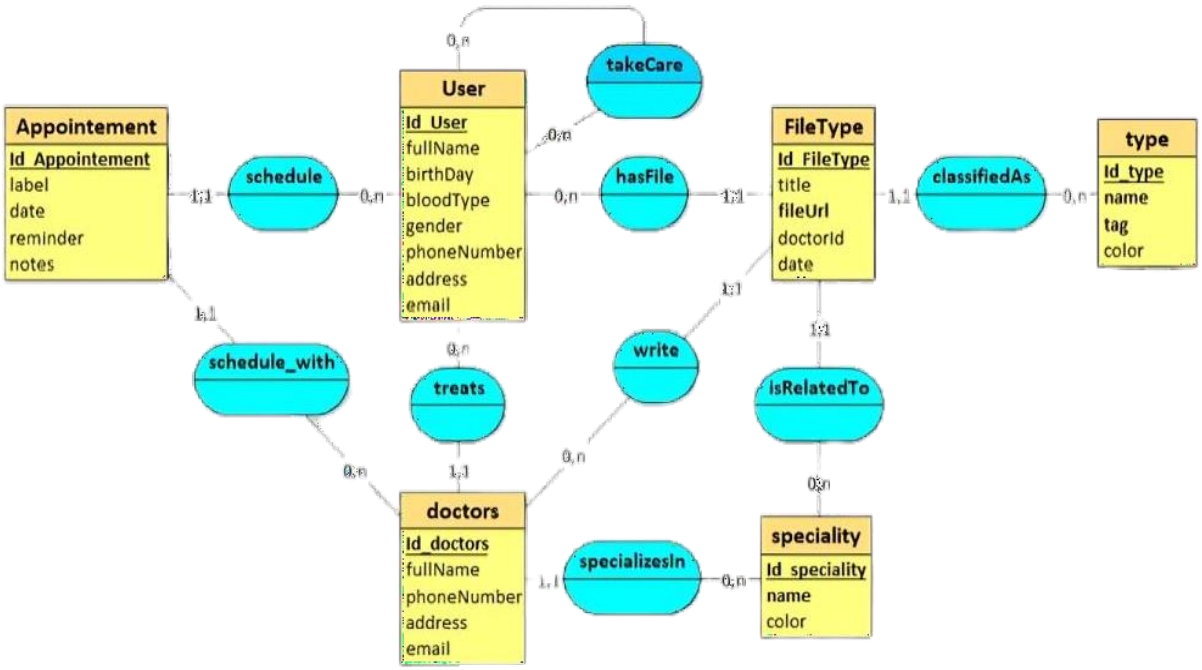


Figure 2.2: *MCD diagram*

3.2.2 Logical Data Model (MLD)

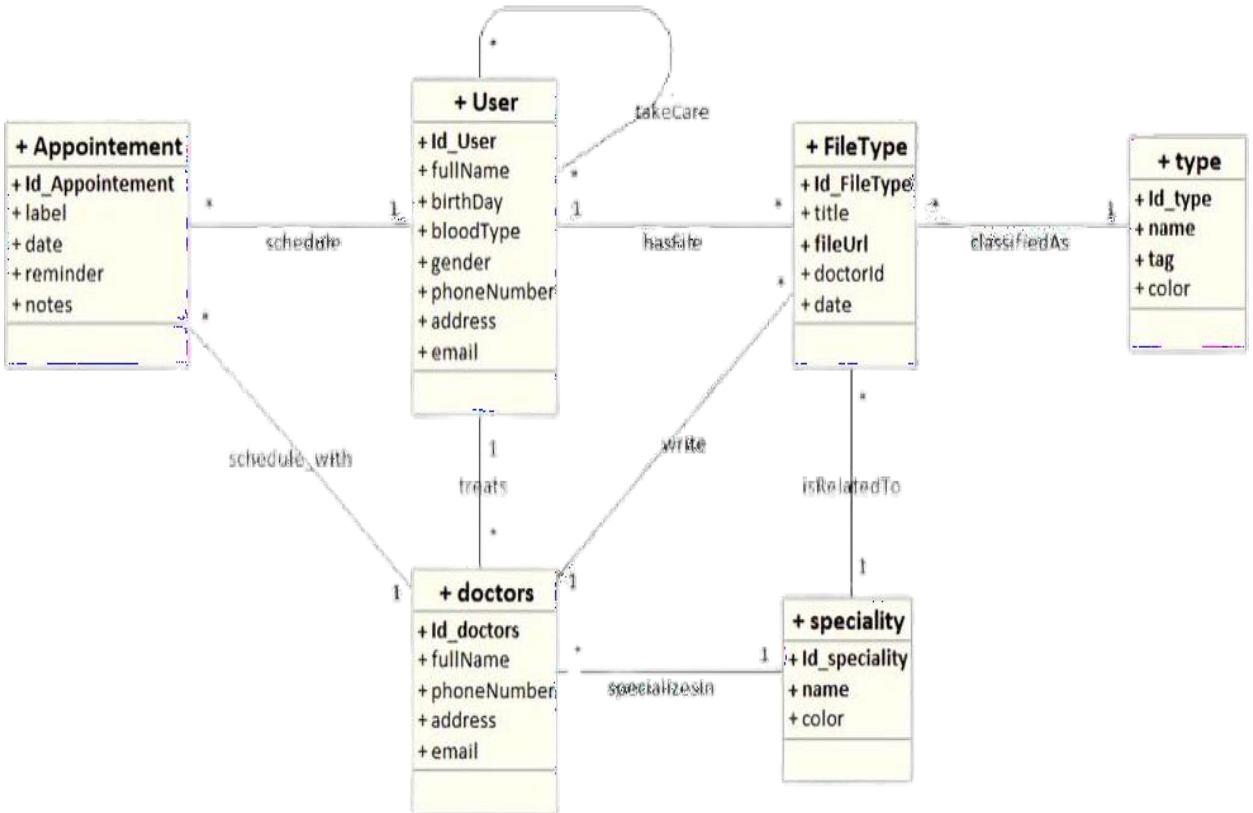


Figure 2.3: *MLD diagram*

3.2.3 Class Diagram

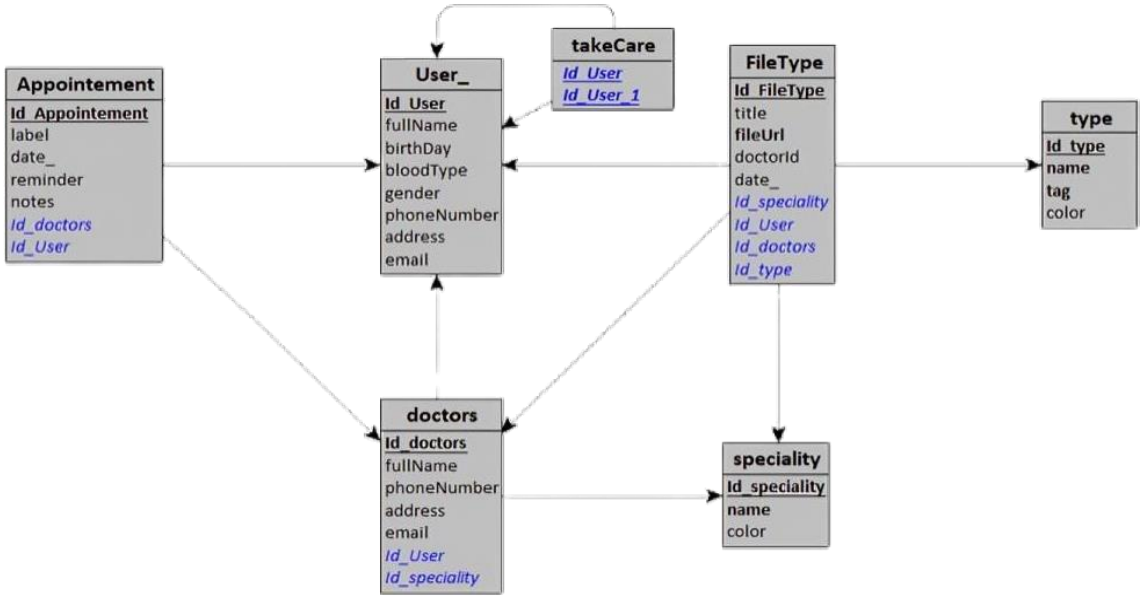


Figure 2.4: Class diagram

3.2.4 Sequence diagram

a) Authentication

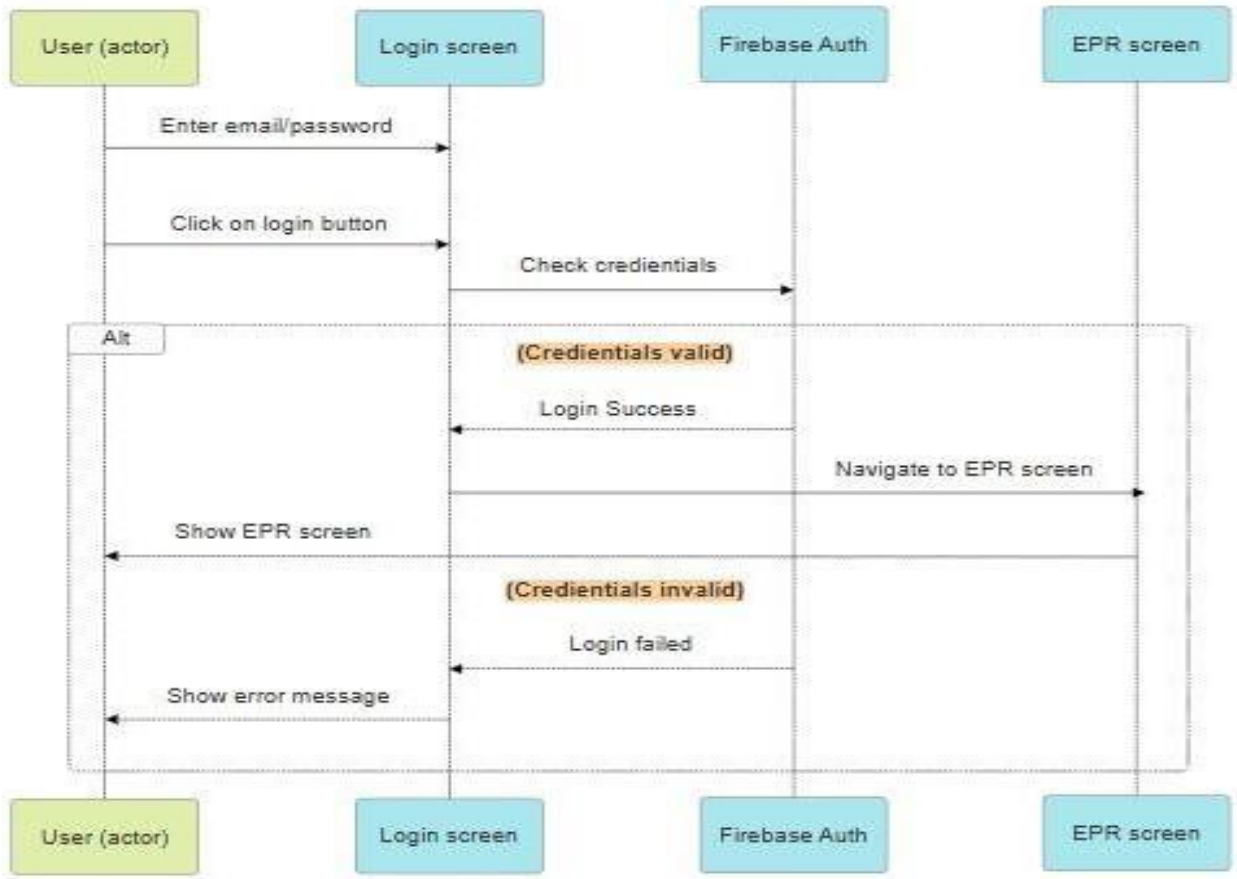


Figure 2.5: *Authentication sequence diagram*

b) Forget password

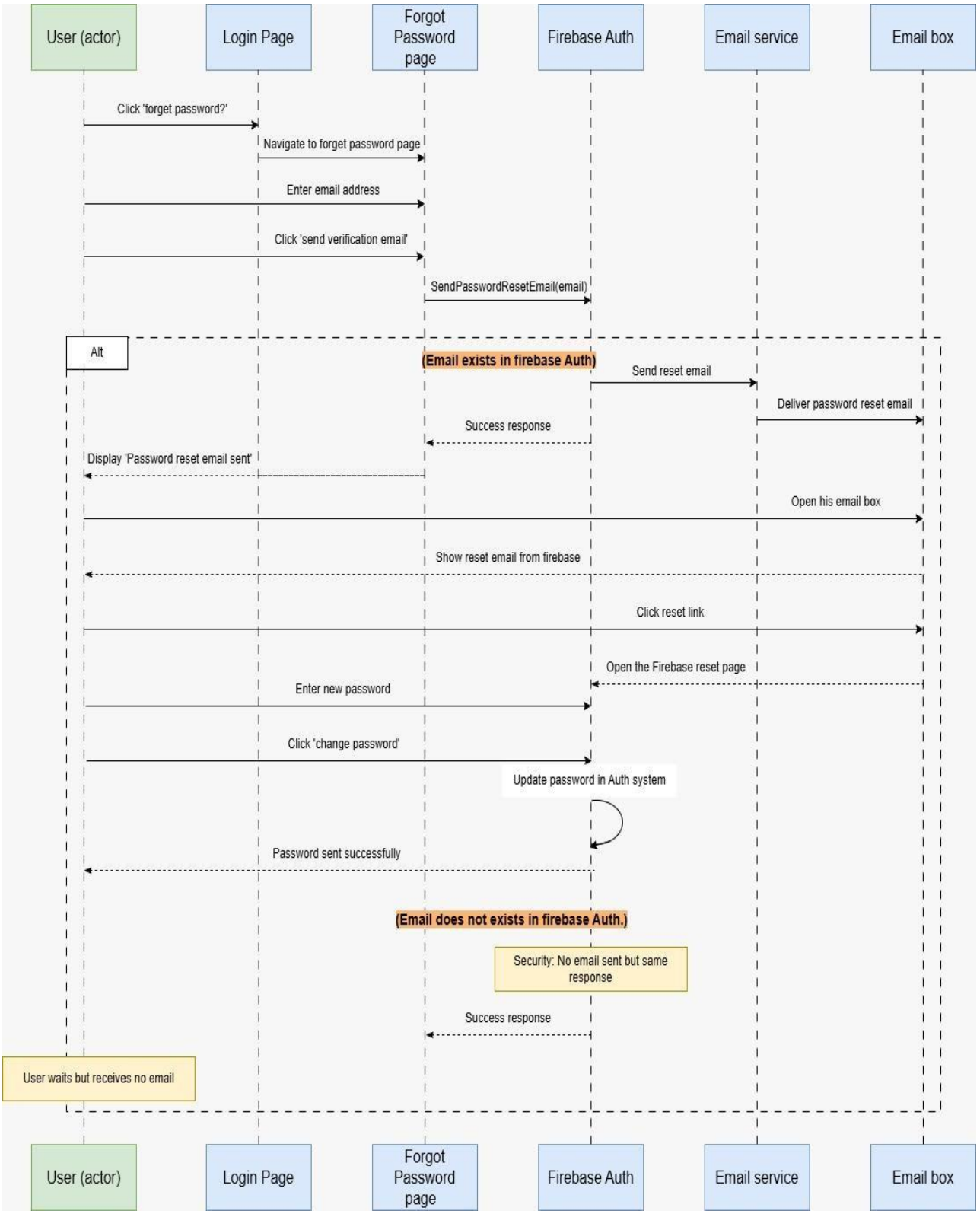


Figure 2.6: *Forget password sequence diagram*

c) Add document file

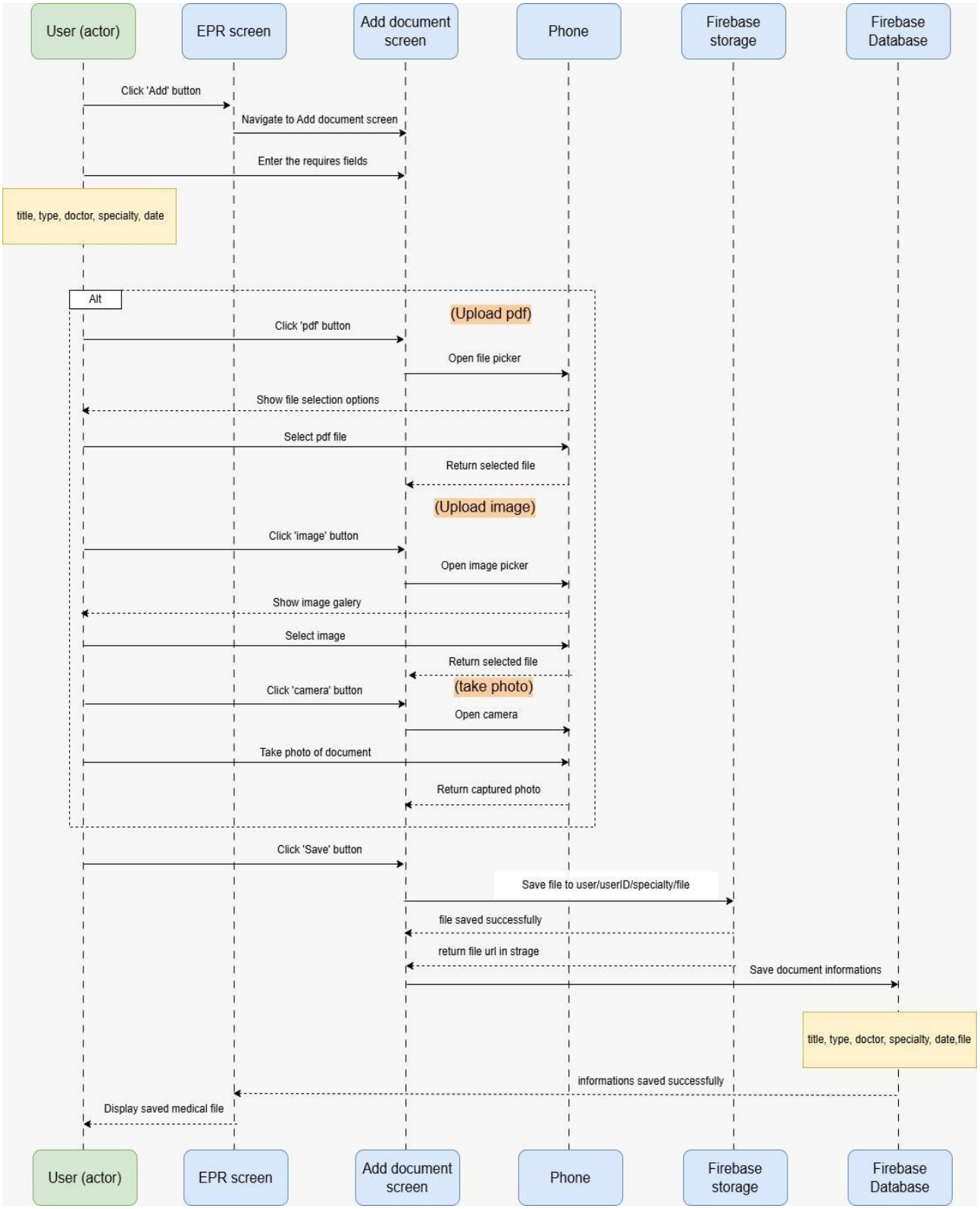


Figure 2.7: *Add document sequence diagram*

d) Add appointment

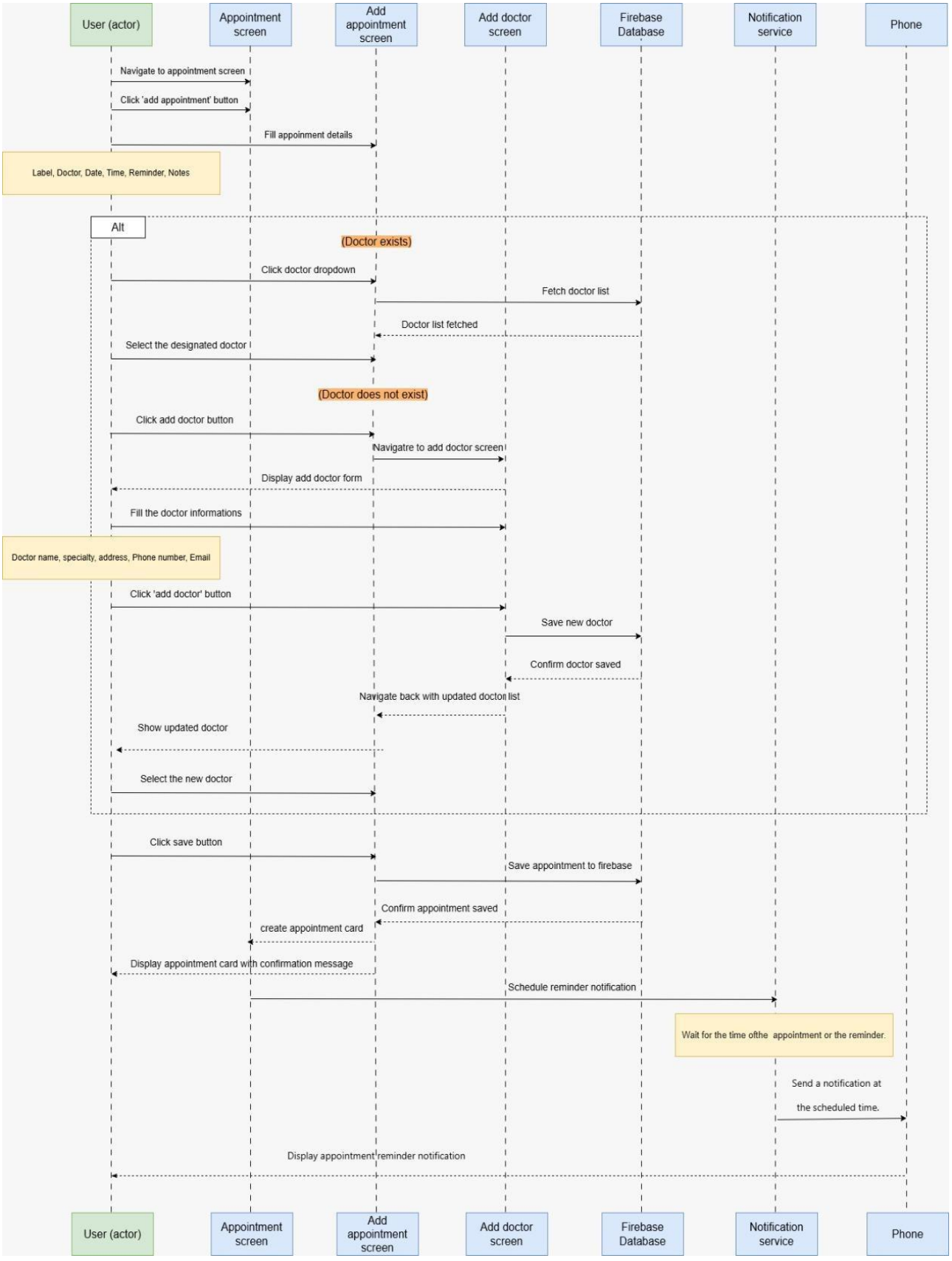


Figure 2.8: *Add appointment sequence diagram*

4 Conclusion

This chapter presented the design and modeling framework for the MedGarde application, it identified critical functional requirements across four core domains: user authentication, patient management, health record management, and appointment scheduling. The non-functional requirements emphasize usability for Arabic speakers, strong security for healthcare, and reliable performance on target devices.

The modular development approach focuses on three main modules: user and patient account management, medical file management and sharing, and doctor directory with appointment scheduling. This setup allows for quick deployment and lays the groundwork for future growth. The detailed use case analysis and system design models offer clear technical plans for implementation.

Having established this design framework, the next chapter examines the technology selection process, evaluating development platforms and tools that can effectively implement these requirements while addressing the specific constraints of this academic project and the Algerian healthcare context.

Chapter 3:

Technology selection and justification

1 Introduction

1.1 Project Context and Constraints

1.1.1 Application Requirements Analysis

MedGarde is designed as a comprehensive healthcare management system with three core modules:

- **User and Patient Account Management:** Multi-user architecture supporting caregivers managing multiple patients
- **Medical File Management:** Secure storage, categorization, and sharing of medical documents
- **Doctor Directory and Appointment Scheduling:** Healthcare provider management and appointment coordination

Technical Requirements:

- Arabic-first interface with RTL text support
- Offline functionality for intermittent connectivity scenarios
- Cross-platform deployment capability (Android primary, iOS future consideration)
- Healthcare-grade data security and privacy compliance
- Integration with document scanning and file management capabilities

1.1.2 Project constraints

- Hardware Limitations: Windows 10 workstation with the following specifications:
  - **Processor**: Intel(R) Core(TM) i3-4005U CPU @ 1.70GHz 1.70 GHz.
  - **Memory**: 8, 00 Go.
  - **Storage**: 476GB SSD with 270GB available space.
  - **Display**: 1360 x 786 resolution.
- Hardware constraints: Development on systems with 2-4GB RAM
- Budget constraints: Preference for open-source and free-tier solutions
- No dedicated backend development team or server infrastructure management capability

1.2 Chapter objectives:

This chapter presents the systematic analysis and justification for the technological foundations of MedGarde

The technology selection process was driven by the application's core requirements:

- multilingual support (Arabic-first design),
- cross-platform compatibility,
- data security compliance

Given the project's scope as a master's thesis with inherent limitations in time, budget, and team size, the technology selection prioritizes solutions that maximize development efficiency while maintaining professional-grade security and performance standards essential for healthcare applications.

2 Mobile Development Approach Decision

2.1 Native vs Cross-Platform Analysis

Decision Point: Should we build native (separate iOS/Android apps) or cross-platform (single codebase)?

Criteria	Native développement	Cross-Platform developpement
Development Time	2x longer (separate codebases)	50% faster (single codebase)
Team Resources	Need iOS + Android specialists	Single team can handle both
Budget constraints	Higher cost (dual development)	Lower cost (shared codebase)
Performance	optimal	Near-native (95% performance)
Maintenance	Separate update needed	Single update for both platforms

Table 3.1: *Native Vs Cross-Platform development*

Conclusion: Given our resource constraints and academic project timeline, cross-platform development is the logical choice.

3 Cross-Platform Framework Selection

Decision Point: Which cross-platform framework best meets our requirements? Flutter, React native or Xamarin.

3.1 Flutter

Flutter is a portable UI software development kit (SDK) created by Google in 2017 for building natively compiled applications across mobile, web, and desktop platforms from a single codebase. Its growing use by companies like Google Ads and Alibaba shows its capability for developing complex, high-performance applications.

Figure 3.1: *Flutter logo*

Key advantages of Flutter include:

- **Rapid development:** the Hot Reload feature allows developers to see code changes in real time, streamlining the development process. Direct machine code compilation enhances execution speed and performance.
- **Cross platform support:** developers can maintain a single codebase for both iOS and Android, ensuring consistency across platforms.
- **Rich UI and Performance:** with a customizable layered architecture, the framework supports expressive user interfaces and quick rendering, along with a variety of ready to use Widgets for visually appealing applications.
- **Strong documentation and community:** comprehensive documentation acts as a central resource, making on boarding and troubleshooting easier.

3.2 React Native:

Figure 3.2: *React Native logo*

React Native is a cross-platform mobile development framework created by Facebook that uses JavaScript and renders native components on both iOS and Android platforms.

Key advantages

- **Cross-Platform efficiency:** Single codebase deployment across both Android and iOS platforms, reducing development time and costs while maintaining native performance.
- **JavaScript foundation:** Uses widely-adopted JavaScript language with flexible syntax and extensive library ecosystem, making it accessible to web developers.
- **Component-Based Architecture: Promotes** code reusability and modular development approach for easier maintenance and updates.
- **Rich ecosystem:** Provides numerous pre-built components and libraries, accelerating development without building features from scratch.
- **Strong community support**: Large developer community offers extensive documentation, quick problem-solving resources, and regular framework improvements.

3.3 Xamarin Framework

Figure 3: *Xamarin logo*

Xamarin is a Microsoft-owned open-source platform that enables development of modern, high-performance applications for iOS, Android, and Windows using .NET and C#.

Key advantages

- **High Code Reusability**: Developers can share up to 90% of application code across platforms, writing business logic once in C# while maintaining native performance and user experience on each platform.
- **.NET Integration**: Built on Microsoft's .NET framework, providing managed environment benefits including automatic memory management and garbage collection.
- **Native Performance**: Acts as an abstraction layer that manages communication between shared code and platform-specific code, delivering native app performance and appearance.
- **Cross-Platform Compilation**: Applications can be developed on PC or Mac and compiled into native packages (.apk for Android, .ipa for iOS) without code modification.
- **Enterprise Focus**: Strong integration with Microsoft ecosystem and enterprise development tools, making it suitable for large-scale business applications.

3.4 Comparaison table

Criteria	Flutter	React Native	Xamarin
Arabic/RTL Support	Excellent built-in	Good with libraries	Limited
Performance	Near-native (60fps)	Good (varies)	Native performance
Learning Curve	Moderate (Dart)	Easy (JavaScript)	Steep (C#/.NET)
Development Speed	Fast (Hot Reload)	Fast	Moderate
Community Support	Growing rapidly	Mature	Microsoft-focused
Future-proofing	Google backing	Facebook backing	Microsoft backing
Healthcare Apps	Alibaba Health, etc.	Limited examples	Some entreprise

Table 3.2: *Cross platform framework comparison*

Key Decision Factors:

- Arabic-first requirement: Flutter's superior RTL and internationalization support
- Performance on low-end devices: Flutter's compilation to native code
- Healthcare app precedents: Proven in medical applications
- Single codebase efficiency: 95% code reuse across platforms

Conclusion: Flutter emerges as the optimal choice for our Arabic-first healthcare application.

4 Environment Selection for Flutter Development

Decision point: Which IDE best supports Flutter development given our constraints?

4.1 Visual Studio Code

Figure 3.4: *Visual studio code logo*

Visual Studio Code (VS Code) is a lightweight, open source code editor developed by Microsoft. It supports a wide range of programming languages and is designed for cross platform development on Windows, macOS, and Linux. The main features of vs code consist of:

- **Extensibility**: A robust ecosystem of extensions allows for extensive customization to meet diverse development needs.
- **Integrated Terminal**: The built-in terminal enables command line operations without the need to leave the editor.
- **IntelliSense:** Offers intelligent code completions based on variable types, function definitions, and imported modules.
- **Debugging Tools:** Features a powerful debugger that allows users to step through code, inspect variables, and view call stacks.
- **Version Control Integration:** Provides seamless integration with Git for effective source control management.

4.2 Android Studio

The official Integrated Development Environment (IDE) for Android app development. Based on the powerful code editor and developer tools from IntelliJ IDEA, Android Studio offers a range of features designed to enhance productivity in building Android applications, such as:

- A flexible Gradle based build system.
- A fast and feature rich emulator.
- A unified environment for developing across all Android devices.

Figure 3.5: *Android studio logo*

- Live edit for updating composables in emulators and physical devices in real time.
- Lint tools for identifying performance, usability, version compatibility, and other issues.

4.3 Visual Studio Code Vs Android Studio

We compare Visual Studio Code and Android Studio to understand their differences and help choose the appropriate development environment.

Overview:

Visual Studio Code is a general-purpose code editor that supports many programming languages, while Android Studio is specifically designed for Android application development.

Resource factor	Visual studio code	Android studio
RAM	2-4GB optimal	8GB+ recommended
Loading time	30 seconds	2-3 minutes
Processor speed	1.6 GHz or faster processor	x86_64 CPU architecture; 2nd generation Intel Core or newer, or AMD CPU with support for a Windows Hypervisor
Disk space	disk footprint of < 500 MB	8 GB of available disk space minimum (IDE + Android SDK + Android Emulator)
Platform	Windows 10 and 11 (64-bit)	Windows 8, 10 and 11 (64-bit)
Cross platform	optimized	Android focused
Flutter Support	Excellent (via extensions)	Good (plugin required)

Visual Studio	Android studio
Resource usage	
Lightweight	Heavy
Customization Options	
Offers extensive customization through extensions and themes. Users can install from over 40,000 available extensions to add new features and change the appearance.	Has limited customization options. It focuses mainly on Android development tools rather than allowing users to modify the interface extensively.
Programming Language Support	
Supports multiple programming languages including JavaScript, Python, Dart, C#, and many others. This makes it suitable for various types of software development projects.	Primarily supports Java and Kotlin, which are the main languages for Android development. It provides advanced features specifically for these languages.
Android Development Features	
Requires additional extensions to develop Android applications. Developers need to install Flutter, React Native, or other plugins and manually configure the Android SDK.	Includes all necessary tools for Android development. It comes with built-in emulators, visual layout editors, and debugging tools specifically designed for Android applications.
Performance	

Runs faster and uses less memory. It opens files quickly and responds immediately to user actions, even on computers with limited resources.	Runs slower due to its comprehensive features. It may become sluggish on older computers but provides more advanced capabilities for Android development.
Testing and Debugging	
Provides basic debugging tools. Advanced testing and debugging features require installing additional extensions.	Includes comprehensive testing and debugging tools designed specifically for Android applications. It offers memory profilers, performance analysers, and advanced debugging features without requiring additional installations.
Cost	
Free	Free

Table 3.3: *Visual studio code Vs Android studio*

Testing Results on Target Hardware (4GB RAM):

- VS Code: Responsive throughout development
- Android Studio: Frequent freezing during UI design
- Productivity impact: 15-20% improvement with VS Code

Conclusion: VS Code provides the optimal development experience within our hardware constraints.

4.4 Visual Studio Code: Optimal Choice

So we choose Visual Studio Code because it is better for developers who work with multiple programming languages-, have limited computer resources - in our case-, or develop web and cross-platform applications. It is lightweight and highly customizable.
Android Studio is more suitable for developers who focus exclusively on Android application development. It provides all necessary tools in one package but requires more system resources.
The choice between these two development environments depends on the specific requirements of the project and the available system resources.

5 Backend selection

Decision Point: Build custom backend vs use Backend-as-a-Service (BaaS)?

5.1 Firebase Platform:

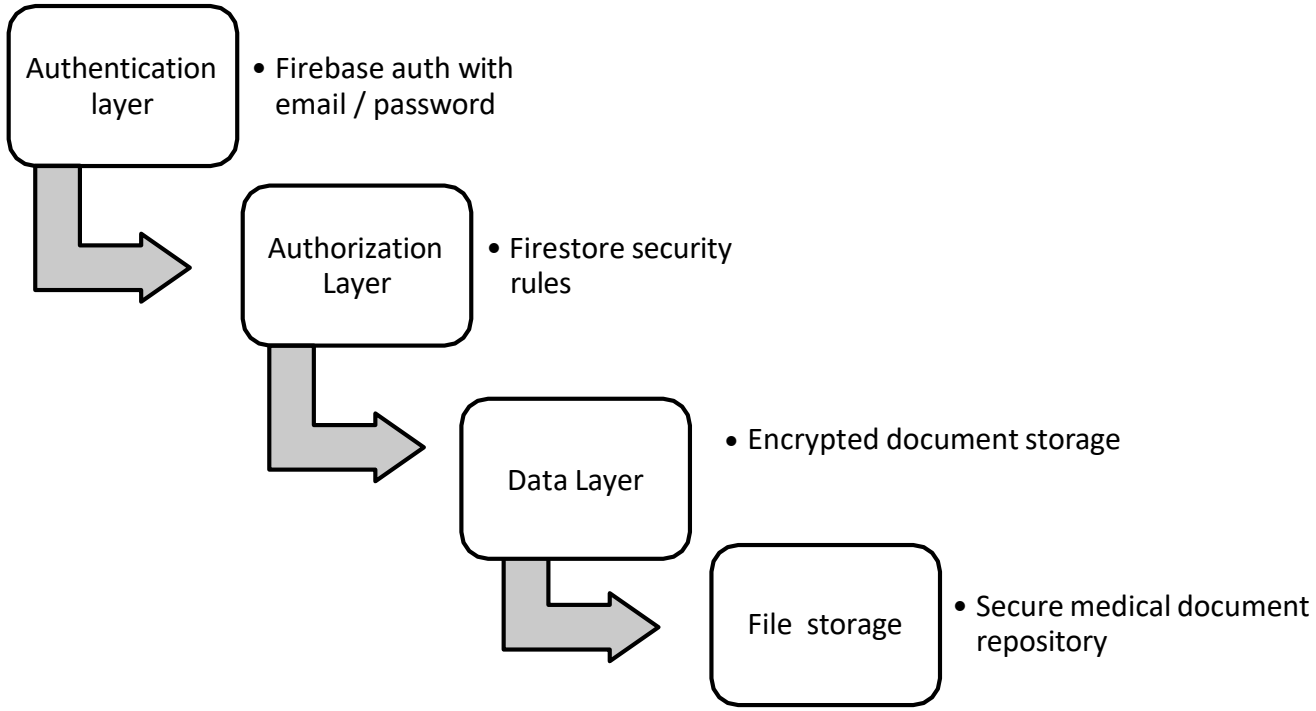
Figure 3.6: *Firebase logo*

Firebase is a Google backend app development platform that streamlines the process of building and deploying applications for both mobile and web. It provides a comprehensive suite of tools for various backend tasks, such as database management, user authentication, storage, and analytics.

This enables developers to concentrate on the front end of their applications without the need to create and manage a complex backend infrastructure.

The principal advantages of Firebase are as follows:

- **Authentication:** Supports passwords, phone numbers, and various social logins.
- **Real time Database:** Data syncs across all clients and remains available offline and NoSQL database support.
- Hosting: Fast content delivery worldwide.
- **Testing Services:** Virtual and physical device testing in Google's data centers.
- **Push Notifications:** Easy implementation without -additional coding.
- **Data Security Implementation:**



5.2 Firebase Vs Custom backend

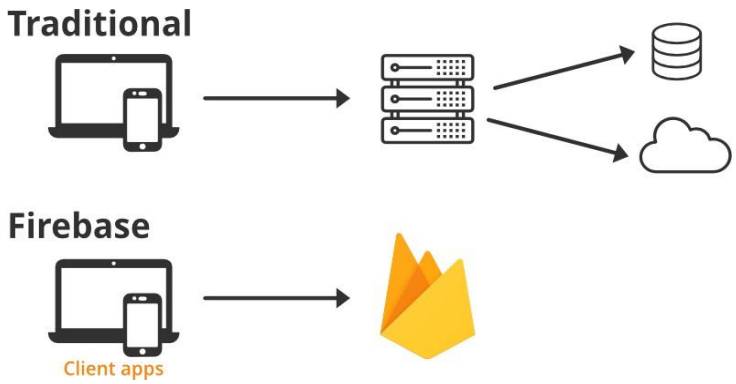


Figure 3.7: *Traditional Vs Firebase architecture*

Criteria	Custom backend	Firebase (BaaS)
Development time	Months	Weeks
Team requirements	Backend developer	Frontend team sufficient
Infrastructure management	Server setup/maintenance	Full managed
Security Implementation	Custom security	Enterprise grade built-in
Scalability	Manual scaling	Auto-scaling
Cost(MVP stage)	High upfront	Free tier available

Table 3.4: *Traditional Vs Firebase*

Conclusion: Firebase eliminates the need for dedicated backend development, server management, and database administration -backend complexity- while providing enterprise-grade security essential for healthcare data.

6 Programming Language Justification

6.1 Dart Language

Context: Flutter requires Dart programming language

Dart is a programming language created by Lars Back and Kasper Lund, with development led by Google. It is suitable for building web and mobile applications, along with server and desktop software.

Figure 3.8: *dart logo*

The key advantages of Dart include:

- Java like, object-oriented syntax for easy learning.
- Excellent official documentation and community resources.
- Real time testing via DartPad.
- Clean and simple development ecosystem.
- High performance across platforms.

Advantages for Our Project:

- Object-oriented syntax (familiar coming from Java/C#)
- Null safety (reduces runtime errors in healthcare context)
- Asynchronous programming (essential for Firebase operations)
- Strong typing (reduces bugs in medical data handling)
- Excellent documentation (important for academic project)

Learning Curve Assessment:

- Coming from Java background: 2-3 weeks proficiency
- Comprehensive documentation available
- Strong community support

7 Design and Development Tools

7.1 Figma

Figma is an advanced design tool utilized for the creation and collaboration on user interfaces and prototypes for websites, applications, and more. Unlike conventional design software that necessitates installation, Figma functions entirely within a web browser, ensuring accessibility across all major platforms, including Windows, macOS, and Linux. This platform independence facilitates easy access and broad usability for both individuals and teams.

Figure 3.9: *Figma logo*

Figma offers a wide range of features and benefits like:

- **Real Time Collaboration:** As a cloud based tool, Figma allows multiple users to work on the same design in real time.
- **Responsive Design:** Its auto layout and constraint features help in building responsive designs that adapt to different screen sizes.
- **Prototyping Tools**: the user can create interactive prototypes with animations, transitions, and gestures, allowing stakeholders to experience how the final product will feel and function.
- **Creating Mobile App Interfaces**: Figma is widely used for designing mobile app interfaces, allowing the creation of multiple screens, interactive prototypes, and user experience testing prior to development.

7.2 Looping:

Figure 3.10: *Looping logo*

Looping is a conceptual data modelling tool designed to help users organize and structure data through ordered diagrams and graphical representations. The primary features of Looping consist of:

- **Free and Open Source**: Completely accessible without licensing restrictions.
- **Entity and Association Modelling:** Supports creation of entities, associations, and reflexive relationships.
- **UML Class Diagram Generation**: Enables visual modelling through class diagrams.
- **SQL Integration:** Allows SQL query execution and table creation.
- **Real Time Display:** Shows logical modules in a text-based format instantly.

7.3 Draw.io

Figure 3.11: *Draw.io logo*

Draw.io and its online editor are top choices for creating sketches and diagrams on the web. You can use the online editor with different storage platforms, and there's also a standalone desktop application for offline use.
It has many features like:

- Security-first diagramming app for teams
- Users choose where to keep their diagram data
- Integrates Google apps, GitHub, Microsoft apps, Notion, and more
- Supports real-time collaboration with shared cursors
- Easy-to-use diagram editor
- Advanced tools for various diagram types

7.4 TeamViewer

TeamViewer is a widely used remote assistance and remote desktop control software developed by the German company TeamViewer GmbH in 2005. It enables technicians and users to access and control devices remotely, making it highly practical in both professional and personal settings.

Figure 3.12: *TeamViewer logo*

The primary characteristics of TeamViewer include:

- **Remote Control Access**: Enables full remote control of another device for troubleshooting.
- **Remote Software Installation:** Allows installation of applications on distant devices without physical access.
- **Telework Support:** Lets employees access their office setups from home with internet connected devices.
- **Cross Platform Compatibility:** Suitable for both professional and personal use.
- **Free for Personal Use:** TeamViewer is available at no cost for non-commercial purposes.

7.5 Git

Git is a version control system designed to efficiently track changes in files. It is particularly beneficial in collaborative settings where multiple contributors are simultaneously modifying the same files.

In a typical Git workflow:

1. A new branch is created from the main version of the collaboratively developed files.
2. Changes are made independently and securely on individual branches.
3. Git then merges these specific changes back into the main branch, ensuring that individual updates do not conflict with one another.

Furthermore, Git enables the use of remote repositories, facilitating seamless code sharing across various locations. By integrating with tools such as Visual Studio Code, developers can push and pull code directly within the editor, which enhances the development workflow. These functionalities contribute to maintaining a synchronized and current project environment while simultaneously minimizing the risk of conflicts and data loss

7.6 Tool Selection Rationale

Each tool selected based on:

- Free/open-source aligned with budget constraints
- Minimal training required
- Working well with Flutter/VS Code ecosystem
- Supporting documentation and presentation needs

- Figma ======➔ Industry-standard UI/UX design
- Git ========➔ Version control essential for academic project documentation
- Looping ====➔ Entity-relationship modeling for database design
- Drow.io ====➔ Used for sequence diagram
- Team Viewer ➔ Team distance collaboration

8 Technology Stack Validation

8.1 Risk Assessment and Mitigation Strategies

Identified Risks	Cause	mitigation
Flutter Ecosystem Maturity	Relatively newer framework compared to React Native	Google's continued investment and growing enterprise adoption
Dart Language Adoption	Limited use outside Flutter ecosystem	Comprehensive documentation and strong community support
Firebase Vendor Lock-in	Dependency on Google's platform	Abstraction layers designed for future migration flexibility

Table 3.5: *Technology Risk Analysis*

8.2 Performance Validation Strategy

Planned Testing:

- Performance benchmarking on target devices (Android 9.0+, 2GB RAM)
- User experience testing with Arabic-speaking healthcare workers
- Offline functionality validation in low-connectivity environments
- Security assessment of data handling procedures

8.3 Technology Stack Alignment with Project Goals

8.3.1 Academic Project Suitability

The technology selection follows a logical decision tree:

- Mobile app requirement → Cross-platform approach (resource efficiency)
- Cross-platform need → Flutter framework (Arabic support + performance)
- Flutter development → VS Code IDE (hardware constraints)
- Backend requirements → Firebase BaaS (single-developer efficiency)
- Supporting tools → Open-source ecosystem (budget alignment)

This systematic approach ensures each technology choice is justified by objective criteria rather than preference, creating a foundation that directly addresses the constraints and requirements of a master's thesis project:

- Resource Efficiency: Enables professional-grade development within student budget constraints Learning Value: Provides exposure to modern, industry-relevant technologies
- Documentation: Strong community resources support academic documentation requirements
- Scalability: Architecture supports future research and commercial development

8.3.2 Requirements Alignment Check

Requirement	Technology solution	Validation
Arabic first interface	flutter internationalization	Proven RTL support
Cross platform deployment	Flutter framework	Single codebase for Android/ IOS
Heathcare data security	Firebase security + encryption	Enterprise grade protection
Low resource development	Vs code + flutter	Efficient on 2-4GB RAM
Rapid development	Firebase BaaS	No backend development needed
Academic constraints	Open-source stack	Zero licensing costs

Table 3.6: *Requirement Alignment check*

9 Conclusion

The technology stack for MedGarde includes Flutter and Visual Studio Code for the frontend. Firebase acts as the backend. This setup provides a reliable solution for the healthcare application. It enables cross-platform development from one codebase. It also supports the Arabic language, guarantees healthcare-grade security, and delivers strong performance within our project limits.

The evaluation process shows that these technology choices are based on objective criteria rather than preference. The next chapter moves from choosing technology to developing practical applications. It covers the entire implementation process, from the initial setup to final testing. This chapter shares the real development experience, highlighting the challenges faced and the solutions used. It also displays the finished functional application with detailed screenshots and technical documentation.

Chapter 4:

Implementation and development

1 Introduction

This chapter explains how to set up the MedGarde application. It covers the system architecture design, and a step-by-step guide to implementing key features like user authentication, managing medical records, and creating a bilingual interface.

The chapter also discusses real-world challenges faced during the project and how the team addressed them. With detailed technical documents and application screenshots, it shows how the chosen technologies resulted in an effective healthcare management solution.

2 System Architecture

2.1 Architectural Overview

2.1.1 Client-Server Architecture with Firebase

The MedGarde application implements a modern client-server architecture leveraging Firebase as the Backend-as-a-Service (BaaS) platform. This architectural approach provides several advantages aligned with the project's requirements for rapid development, scalability, and reliable data management.

2.1.2 Three-Tier Architecture Implementation

The system follows a three-tier architecture pattern:

- Presentation Layer (Client): Flutter mobile application running on Android devices
- Business Logic Layer: Firebase Cloud Functions and client-side business logic
- Data Layer: Firebase Firestore NoSQL database with Firebase Storage for file management

2.1.3 Data Flow Architecture

The system implements a unidirectional data flow pattern:

- User Interaction: Users interact with the Flutter UI components
- State Management: Application state is managed using Provider
- Service Layer: Business logic services communicate with Firebase APIs
- Data Persistence: Firebase Firestore stores structured data while Firebase Storage handles file uploads
- Real-time Updates: Changes are propagated back to all connected clients through Firebase listeners

2.1.4 Security Architecture

The architecture implements multiple security layers:

- Authentication Layer: Firebase Authentication with email/password and secure token management
- Authorization Layer: Firestore security rules control data access based on user roles and relationships
- File Security: Firebase Storage rules ensure only authorized users can access medical documents
- Data Encryption: All data is encrypted in transit (TLS) and at rest
  - **Encryption at Rest - Fully Automatic :**
  - **Firestore Database**: All documents automatically encrypted with AES-256
  - **Firebase Storage**: All uploaded files automatically encrypted
  - **Firebase Authentication**: User credentials automatically encrypted
  - **No developer action required** - happens behind the scenes
  - **Encryption in Transit - Fully Automatic:**
  - **TLS/HTTPS**: All communication automatically uses TLS encryption

- **API calls**: Encrypted by default between app and Firebase servers
- **No configuration needed** - Firebase enforces secure connections

2.2 Architecture Diagram and Components

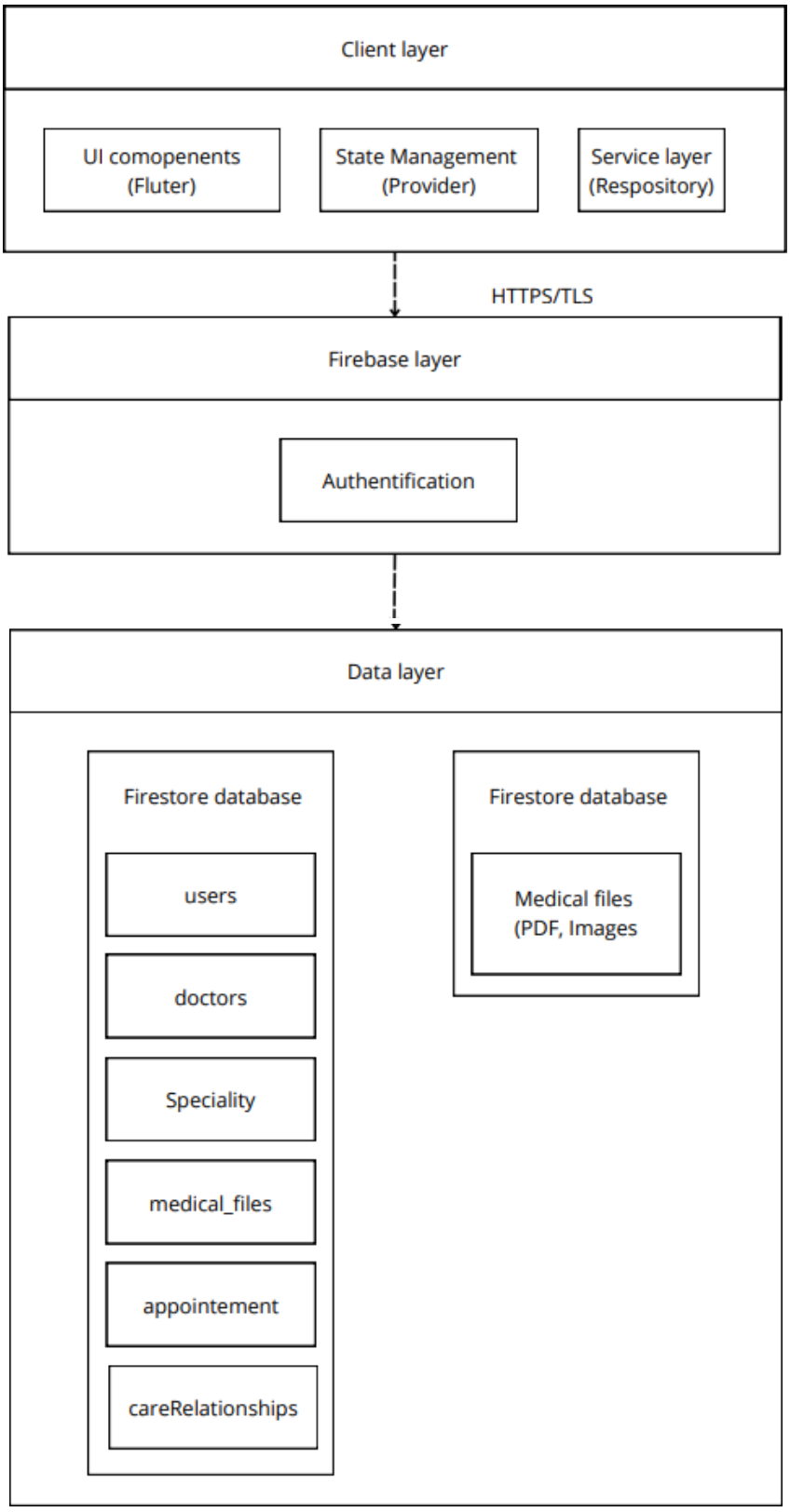


Figure 4.1: *Architecture diagram*

2.3 Database Design and Data Model

The data model is designed to support the modular architecture while maintaining data consistency and enabling efficient queries.

2.3.1 Firebase Collections Structure

Collection	Collection parameters	Role

Users Collection	{
	"uid": "user identifier",
	"fullName": "user full name",
	"phoneNumber": "phone number",
	"address": "user address",
	"birthDay": "DD/MM/YYYY",
	"gender": "Male/Female",
	"email": user@example.com,
	"bloodType":
	"A+/B+/O+/AB+/A-/B-/O-/AB-",
	}
		The user's collection stores caregiver and patient profile information. Each document represents a registered caregiver who can manage multiple patients through the careRelationships collection, and a patient that can't manage his account

Doctors Collection	{
	"id": "doctor identifier",
	"fullName": "Dr. Doctor Name",
	"speciality": "Medical speciality",
	"email": "doctor@example.com",
	"phoneNumber": "doctor phone",
	"address": "clinic address",
	"userId": "patient's uid",
	}
		The doctor's collection maintains a directory of healthcare providers. Each caregiver can add and manage their own doctor entries, supporting the requirement for a searchable doctor directory.

Medical Files Collection	{
	"id": "file identifier",
	"userId": "patient's uid",
	"doctorId": "associated doctor id",
	"title": "document title",
	"type": "type name",
	"speciality": "speciality name",
	"date": "dd mm yy",
	"file Type": "pdf/image",
	"fileUrl": "firebase_storage_url",
	}
		Medical files store metadata about uploaded documents, with actual files stored in Firebase Storage. This separation allows for efficient querying while supporting large file uploads.

Appointments Collection	{ "id": "appointment identifier", "patientId": "patient's uid", "doctorId": "associated doctor id ", "date": "dd mm yy at 00:00:00", "label": "Appointment Label", "notes": "Additional Notes",	Appointments link patients with doctors for scheduled visits, supporting calendar functionality and reminder notifications.

Care Relationships Collection	"reminder": "reminder setting", }	

	{
	"id": "relationship identifier",
	"caregiverId": "caregiver's uid",
	"patientId": "patient's uid",
	"relationshipTitle": "ex: Mother",
	}
		Care relationships enable the multi-caregiver functionality, allowing multiple users to manage a single patient's medical information with appropriate permissions.

File Types Collection	{
	"id": "type identifier",
	"name": "file type name",
	"tag": "category tag",
	"color": "hexadecimal code",
	}
		File types provide categorization for medical documents, supporting the search and filtering requirements.

Speciality Collection	{
	"name": "speciality Name",
	"color": "hexadecimal code",
	}
		Medical speciality provide categorization for medical documents and doctors, supporting the search and filtering requirements.

Table 4.1: *Collections Structure*

2.3.2 Data Relationships and Integrity

The data model implements logical relationships between collections:

- **User-Centric Design**: All data is associated with user accounts, ensuring proper access control and data ownership.
- **Flexible Patient Management**: The careRelationships collection enables many-to-many relationships between caregivers and patients.
- **Document Linking**: Medical files are linked to both users and doctors, enabling quick access to patient history and doctor-specific documents.
- **Appointment Integration**: Appointments connect patients with doctors

2.3.3 Scalability Considerations

The NoSQL document structure supports horizontal scaling:

- **Denormalization**: Some data is intentionally duplicated to reduce query complexity
- **Indexing Strategy**: Firestore composite indexes support complex queries for searching and filtering
- **Subcollections**: Future expansion can utilize Firestore subcollections for hierarchical data organization

2.3.4 Data Validation and Security Rules

Firestore security rules enforce data validation and access control:

- **Authentication Requirements**: All operations require authenticated users
- **Ownership Validation**: Users can only access their own data and shared patient data
- **Relationship Verification**: Care relationships are validated before allowing patient data access
- **Data Integrity**: Rules prevent inconsistent data states and enforce required fields

This data model design supports the current modular implementation while providing flexibility for future feature expansion, ensuring the MedGarde application can grow to meet the comprehensive healthcare management vision outlined in Chapter 1.

3 Flutter Project Structure

This flutter project uses a feature based and layered separation architecture the code is organized by business areas, such as patients, doctors, and appointments. While keeping shared resources (constants, widgets, services) in separate layers. The design follows modular principles, allowing for reusable components and centralized state management with providers. And built-in internationalization - different languages- support.
This method allows us to work on features independently while keeping the code reusable and scalable for complex healthcare applications.
To improve the readability and simplicity of code, we used refactoring strategy which is the process of restructuring code, while not changing its original functionality.

```
lib/
├── constants/
│   ├── app_colors.dart
│   ├── app_strings.dart
│   ├── app_text_style.dart
│   ├── export_constants.dart
│   └── global_variables.dart
│
├── firebase_files/
│   ├── auth_service.dart
│   └── firebase_options.dart
│
├── generated/
│   ├── intl/
│   │   ├── messages_all.dart
│   │   ├── messages_ar.dart
│   │   └── messages_en.dart
│   └── l10n.dart
│
├── global_widgets/
│   ├── bloodtype.dart
│   ├── custom_dropdown.dart
│   ├── custom_search_bar.dart
│   ├── custom_text_field.dart
│   ├── genderselection.dart
│   └── search_header.dart
│
├── l10n/
│   ├── intl_ar.dart
│   └── intl_en.dart
│
├── models/
│   ├── care_relation.dart
│   ├── doctor.dart
│   ├── medical_file.dart
│   ├── patient_doctor.dart
│   └── user.dart
│
├── providers/
│   ├── local_provider.dart
│   └── patient_context_provider.dart
```

```
├── screens/
│   ├── appointment/
│   │   ├── add_appointment_screen.dart
│   │   └── appointments_screen.dart
│   │
│   ├── authentication/
│   │   └── widgets/
│   │       └── forgot_password.dart
│   │
│   │   ├── login.dart
│   │   └── signup.dart
│   │
│   ├── doctor/
│   │   ├── add_doctor_screen.dart
│   │   ├── doctor_card.dart
│   │   ├── doctor_form.dart
│   │   ├── doctor_screen.dart
│   │   └── edit_doctor_screen.dart
│   │
│   ├── drawer/
│   │   ├── widgets/
│   │   │   ├── about_us.dart
│   │   │   └── email_service.dart
│   │   │
│   │   └── app_drawer.dart
│   │
│   ├── epr/
│   │   ├── add_document_screen.dart
│   │   ├── edit_document.dart
│   │   ├── epr_screen.dart
│   │   └── speciality_screen.dart
│   │
│   ├── home/
│   │   └── home_screen.dart
│   │
│   ├── patients/
│   │   └── widgets/
│   │       └── alert_dialog.dart
│   │
│   │   ├── login_form.dart
│   │   ├── patient_card.dart
│   │   ├── patients_screen.dart
│   │   └── signup_patient.dart
│   │
│   └── profile/
│       ├── widgets/
│       │   ├── edit_button.dart
│       │   ├── password_field.dart
│       │   └── password_toggle.dart
│       │
│       └── profile_screen.dart
│
├── services/
│   ├── account_service.dart
│   ├── file_upload_service.dart
│   ├── notification_service.dart
│   └── storage_service.dart
│
├── theme/
│   └── app_theme.dart
```

```
│
├── utils/
│   └── firebase_helper.dart
│
├── widgets/
│   ├── language_selection.dart
│   ├── notification_overlay.dart
│   ├── notification_test_button.dart
│   └── onboarding.dart
│
└── main.dart
```

4 Navigation Flow Architecture

4.1 Authentication and On boarding Flow

App Launch Sequence:

```
App Launch
├── Splash Screen (2-3 seconds)
│   ├── Check Authentication Status
│   ├── Check Onboarding Status
│   └── Route to Appropriate Screen
└── Navigate to Next Screen
```

First-Time User Flow:

```
Onboarding Flow (New users only)
├── Welcome Screen
│   └── App Overview & Introduction
├── Features Walkthrough
│   └── Key App Capabilities
└── Get Started Button
    └── Navigate to Registration
```

Authentication Screen:

- **Login Screen**
  - **Email Input** - User email address
  - **Password Input** - User password
  - **Remember Me Toggle** - Stay logged in option
  - **Login Button** → Navigate to EMR (Medical Guard dashboard)
  - **Forgot Password Link** → Password Reset Flow
  - **Sign Up Link** → Navigate to Registration

- **Registration Screen**
  - **Personal Information Form** - User details collection
  - **Account Creation Process** → Email Verification Required
  - **Sign Up Button** → Navigate to EMR (Medical Guard dashboard)

4.2 Main Navigation Structure

Bottom Tab Navigation (4 Primary Tabs)

1. **Profile** - User account and settings
2. **EMR/Medical Files** - Home tab, medical records management
3. **Patients** - Patient relationship management
4. **Doctors** - Healthcare provider directory

4.3 Patient Management Module Flow

Patients Tab Main View

```
Patients List Interface
├── Search Bar - Find specific patients
├── Patient Cards (Enhanced Display)
│   ├── Patient Name or Relationship Label
│   ├── Tap Action → Open Patient's EMR
│   └── Long Press → Context Menu
│       └── Remove Responsibility (with confirmation)
└── Add Patient FAB (Floating Action Button)
```

Add Patient Options

```
Add Patient Flow
├── Link Existing Patient
│   ├── Patient Email Input
│   ├── Patient Password Input
│   └── Add Patient Button
└── Create New Patient
    ├── Personal Information Form
    ├── Account Creation → Email Verification
    └── Add Patient Button
```

4.4 Medical Files Management (EMR)

EMR Tab Structure

```
EMR Main Interface
├── Medical Specialties Grid View
├── Global Search Functionality
└── Specialty Management
```

Specialty Card Interface

```
Specialty Card
├── Specialty Name Display
├── File Count Badge
└── Tap Action → View Specialty Files
    ├── Medical Files List View
    ├── Search & Filter Options
    │   ├── Filter by File Type
    │   ├── Filter by Doctor
    │   └── Filter by Date Range
    └── File Cards Display
```

File Card Details & Actions

```
Medical File Card
├── File Information Display
│   ├── Document Title
│   ├── File Type Tag
│   ├── Doctor Name
│   ├── Date Created/Modified
│   └── File Preview
└── Tap Action → Context Menu
    ├── View File - Open document viewer
    ├── Edit File
    │   ├── Update File Details
    │   └── Replace Attached Document
    ├── Share with Doctor - Send to healthcare provider
    ├── Download - Save to device
    └── Delete File - Remove with confirmation
```

Add New File Process

```
Add Medical File FAB
├── File Title Input
├── File Type Selection
```

```
├── Doctor Selection (with specialty)
├── Date Input
├── Document Source Options
│   ├── Scan Document (Camera)
│   ├── Upload from Gallery
│   └── Import from Files
└── Save Button
```

4.5 Doctor Directory Module Flow

Doctors Directory Interface

```
Doctors Management
├── Search Functionality
├── Appointments Tab Integration
├── Doctors List View
└── Doctor Card Interface
    ├── Doctor Name Display
    ├── Tap Action → Doctor Details View
    │   ├── Medical Specialty
    │   ├── Contact Information (Phone/Email)
    │   └── Office Address
    └── Long Press → Delete Doctor (with confirmation)
```

Add Doctor Process

```
Add Doctor FAB
├── Doctor Name Input
├── Medical Specialty Input
├── Email Address Input
├── Phone Number Input
└── Office Address Input
```

4.6 Appointment Management Flow

Appointments Tab Features

```
Appointments Interface
├── Calendar View Options
│   ├── Month/Week/Day Toggle
│   ├── Appointment Indicators
│   └── Tap Date → Day Detail View
├── Appointment Cards
│   ├── Doctor Name
│   ├── Date & Time
│   └── Status Indicator
└── Add Appointment FAB
```

Create Appointment Process

```
Add Appointment Flow
├── Appointment Label Input
├── Doctor Selection
├── Date & Time Picker
├── Notes & Instructions
└── Reminder Settings Configuration
```

4.7 User Profile Management Flow

Profile Tab Information

```
User Profile Display
├── Relationship Name (if in patient view mode)
├── Full Name
├── Date of Birth
├── Gender
├── Blood Type
```

```
├── Phone Number
├── Email Address
└── Edit Profile Button
```

Profile Editing Interface

```
Edit Profile Form
├── Relationship Name (patient view mode)
├── Full Name Input
├── Date of Birth Picker
├── Gender Selection
├── Blood Type Selection
├── Phone Number Input
├── Email Address Input
├── Change Password Option
│   ├── Current Password Input
│   └── New Password Input
└── Save Changes Button
```

4.8 Settings and Configuration Flow

Settings Menu Options

```
Settings Tab
├── Language Selection
├── Contact Us - Support information
├── About Us - App information
└── Logout - End user session
```

4.9 Key Interface Features Summary

- **Multi-role Support**: Medical guard and patient views
- **Comprehensive EMR**: Specialty-based file organization
- **Patient Relationship Management**: Link and manage multiple patients
- **Doctor Directory**: Maintain healthcare provider contacts
- **Appointment Scheduling**: Calendar-based appointment management
- **Document Management**: Scan, upload, and organize medical documents
- **Enhanced Security**: Authentication with password reset capabilities

5 Core Features Implementation

5.1 Internationalization: Arabic & English Implementation

The MedGarde app supports multiple languages, specifically English and Arabic, to serve diverse users with cultural sensitivity and correct text direction.

a) App Localization Delegate

The application uses a custom AppLocalizationDelegate to support English and Arabic, validating locales, loading language resources, and optimizing performance by managing reloads.

b) Message Lookup Architecture

The system has different MessageLookup classes for each language, with Arabic having over 1000 localized messages for features like medical records and appointments, supporting dynamic content insertion.

c) Language Provider Integration

The app uses a LanguageProvider with Flutter's Provider pattern for managing language preferences and current locale state, storing settings with SharedPreferences for consistency.

d) Directional Layout Support

The system automatically detects language, using Flutter's Directionality widget for correct text direction: LTR for English, RTL for Arabic.

e) Application Structure Integration

The main application class (MyApp) uses MultiProvider for PatientContextProvider and LanguageProvider, and configures localization delegates in MaterialApp.

f) Initialization and Persistence

The application starts by loading saved language preferences in the LaunchScreen widget for a smooth user experience, checking for first-time users.

The Arabic language support ensures correct text direction, font rendering, and translations, providing a scalable solution for multilingual functionality with good performance and user experience.

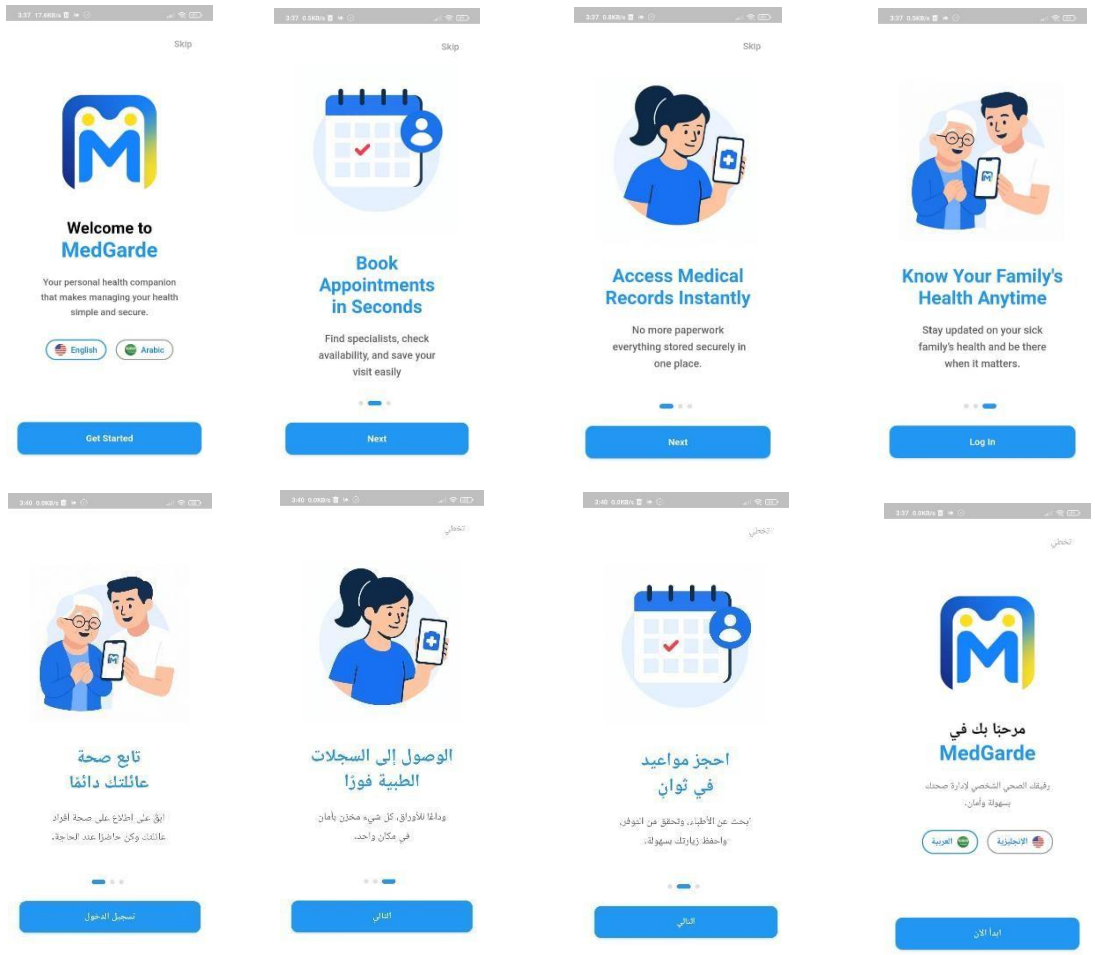


Figure 4.2: *On boarding*

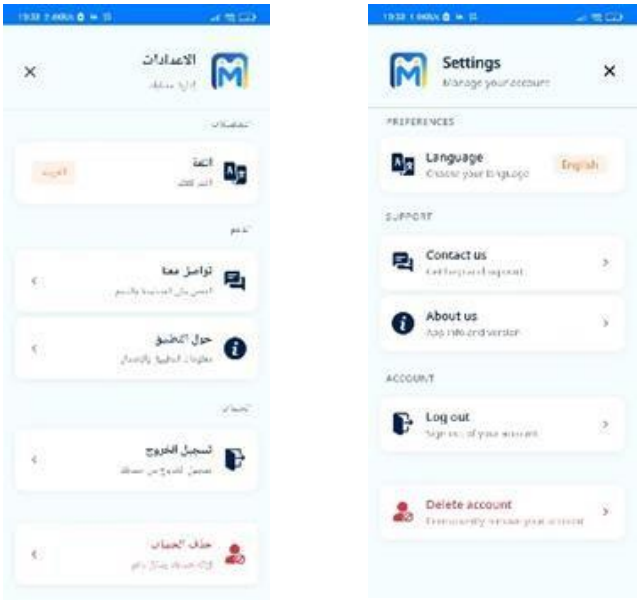


Figure 4.3: *Settings*

5.2 User Authentication System

The MedGarde app uses Firebase Authentication and Firestore for secure user registration, login, password recovery, and profile management with strong error handling and improved user experience.

The centralized AuthService class manages authentication operations and connects the application to Firebase services and user profile storage.

The service uses FirebaseAuth for authentication and FirebaseFirestore for storing user data, ensuring secure credential verification and data management.

User Registration Process

The user registration system collects and checks essential information including username, email, password, phone number, birth date, gender, blood type, and address information before creating an account.

The system creates a new user account using Firebase Authentication's createUserWithEmailAndPassword method after successful validation and retrieves the UID from the returned UserCredential object.

The registration process stores detailed user profiles in a Firestore collection called 'users', identified by Firebase UID, containing the other personal information fields.

The system uses SharedPreferences for local data storage, aiding automatic login and user experience, with error handling during registration.

User Authentication and Login

When users submit their credentials, the system utilizes Firebase Authentication's signInWithEmailAndPassword method to verify the provided credentials against the stored user database.
The login has a "Remember Me" feature that leverages SharedPreferences to save user emails locally for easier future logins.
Error handling during login gives feedback on issues like user not found, wrong passwords, and invalid emails.
Users are directed to the main interface (EPR screen) after successful login.

Password Recovery System

The forgot password function helps users securely recover their accounts using Firebase Authentication's password reset system with guidance.
The password recovery process validates the email and sends a password reset email using Firebase Authentication's sendPasswordResetEmail method.

Security and Error Handling

The system ensures security and error handling, using Firebase Authentication which provides built-in security features including secure password hashing, session management, and protection against common authentication vulnerabilities.
Error handling occurs at various levels, offering clear messages for issues like weak passwords, existing emails, network problems, and invalid inputs to aid users.
Firebase Authentication automatically keeps users logged in until they log out or sessions expire

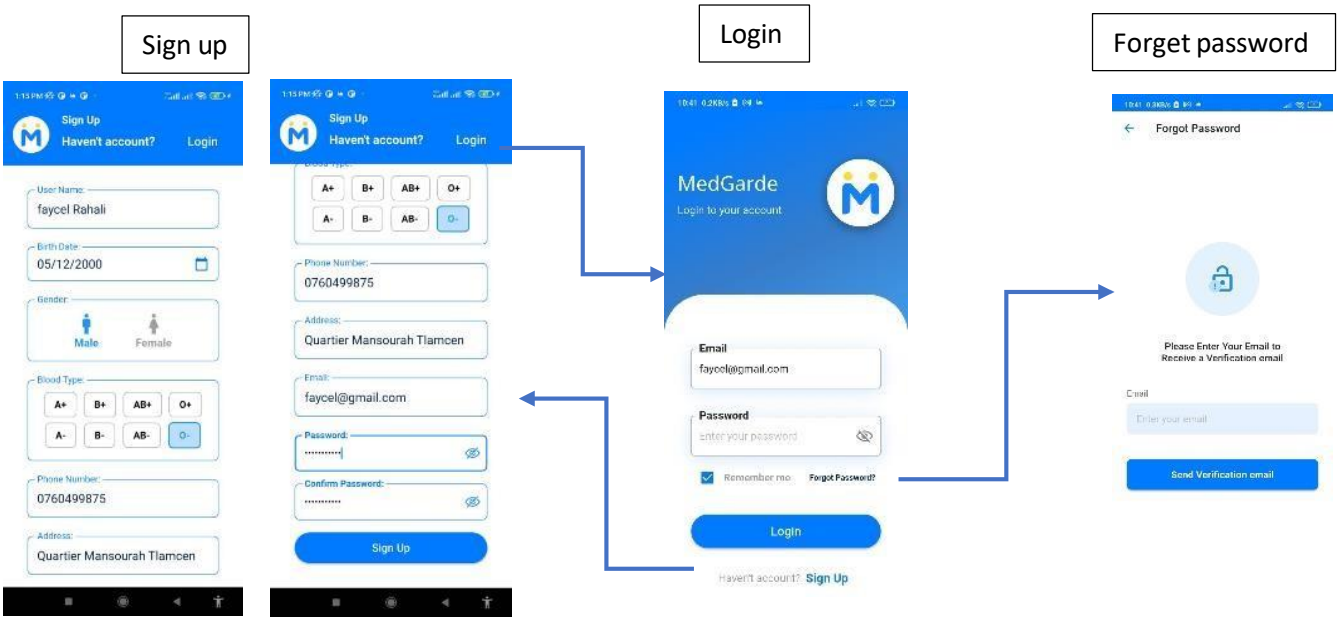


Figure 4.4: *Authentication and registration interface*

5.3 Patient Management System

Core Architecture and Data Flow

The system uses Firebase Authentication for user management and Cloud Firestore for data storage, focusing on caregiver-patient connections. It uses PatientContextProvider for state management.

Patient List Management

The main patient screen allows users to manage both their own and their patients' medical information through a self-referential 'Me' card, where they can act as both caregiver and patient making care management easier.

The screen uses Firebase Firestore for real-time data synchronization, updating patient relationships based on the caregiver ID, allowing users to see only authorized patients.

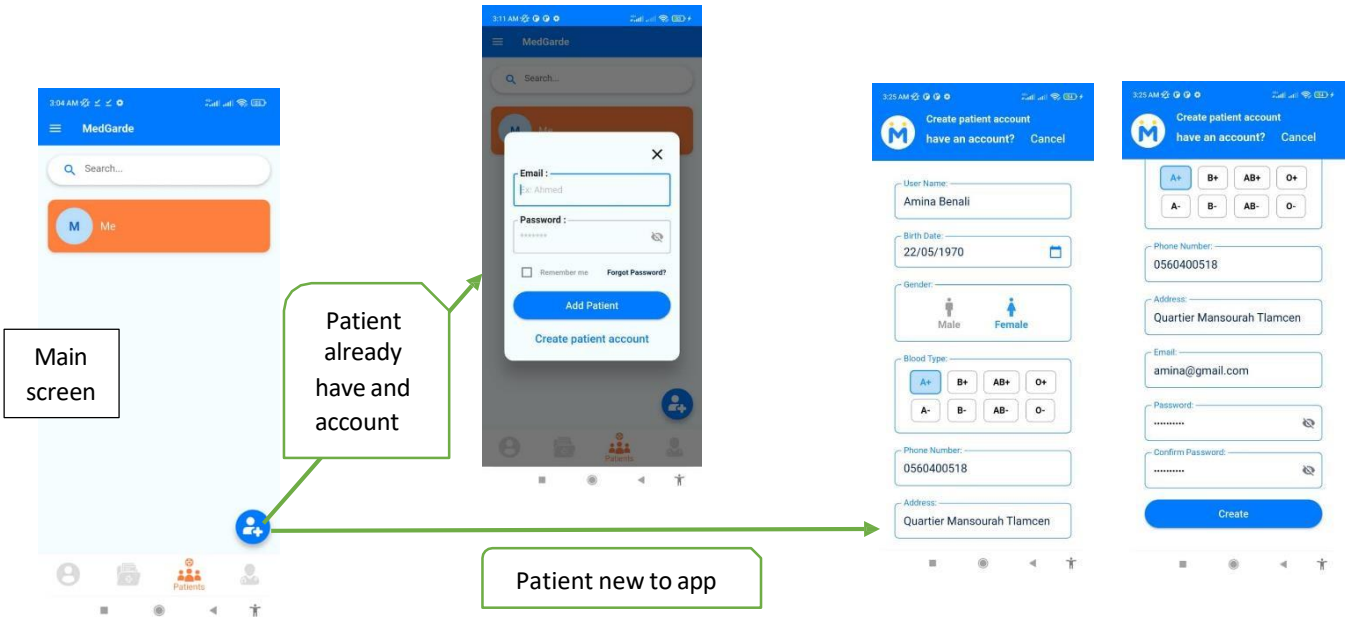
Patient Addition Workflow

The system allows two ways to add new patients. One method is linking existing accounts by verifying credentials using a patient's email and password, retrieving their profile upon successful authentication. The second pathway allows caregivers to create new patient accounts by collecting detailed patient information. It establishes a Firebase Authentication account and sets up care relationships between the caregiver and patient.

Patient Context Management

The system has a context-aware interface that switches between personal mode for users to manage their own medical data and caregiver mode for managing another patient's information.

The patient removal process has a two-step confirmation to avoid accidental deletions. Users receive a general confirmation and a specific warning, protecting their personal records.



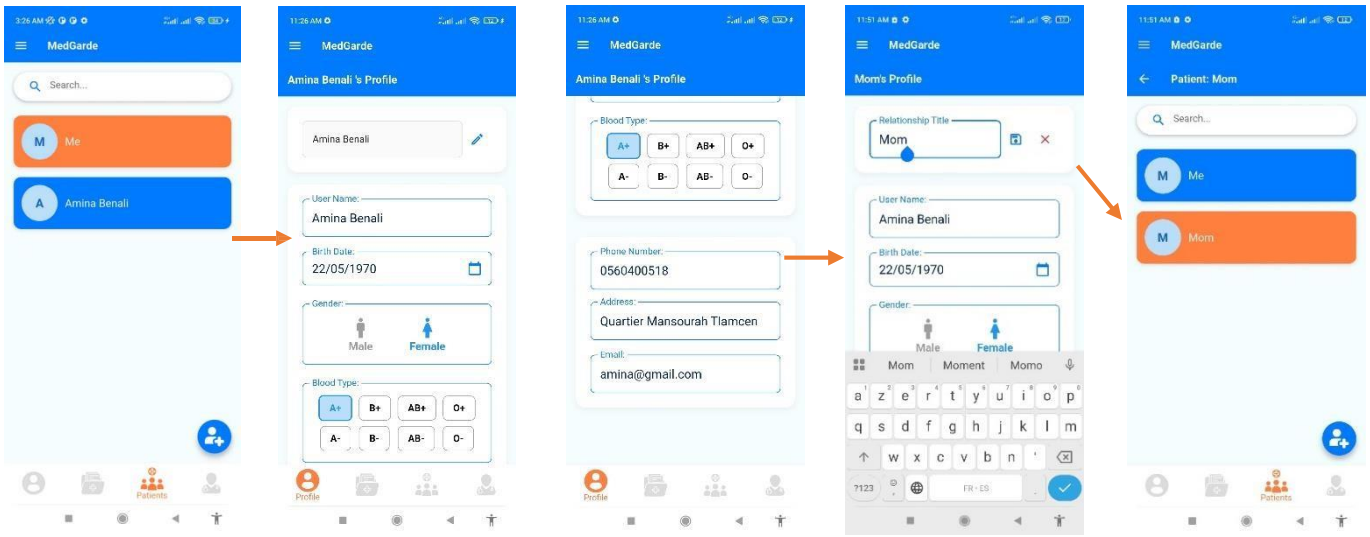


Figure 4.5: *Patient management interface*

5.4 Profile Management System

Core Architecture and State Management

The ProfileScreen class uses StatefulWidget for mutable state and the Provider pattern with PatientContextProvider. It enables dual modes for caregivers to view and edit patient profiles while maintaining distinct permissions and interface behaviors.

Data Controllers and Form Management

The system uses several TextEditingController instances for user information fields and a GlobalKey to validate the form. The implementation maintains separate boolean flags for editing states.

User Context Detection

The application checks the PatientContextProvider state to load either a selected patient's profile or the current user's profile for data safety and user experience.

Data Loading and Synchronization

The _loadUserData method implements asynchronous data retrieval from Firestore, handling both user profile information and care relationship details.

Profile Data Persistence

The saveUserData method checks required fields before saving user profile changes to Firestore, ensuring all necessary information is complete and secure. Password updates require reauthentication using the current password before applying new credentials, maintaining security best practices

.

Relationship Management

The _saveRelationshipTitle method updates relationship metadata and synchronizes changes with the PatientContextProvider to maintain a consistent state across the application. This ensures relationship titles remain current throughout the user session.

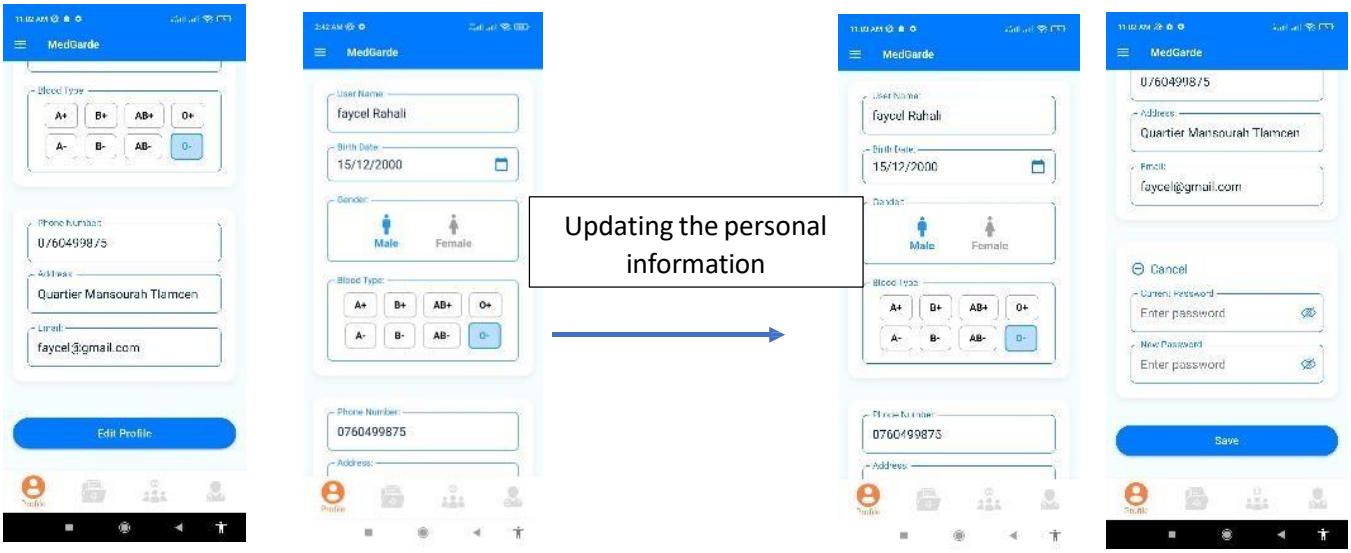


Figure 4.6: *Profile (save/edit) interface*

5.5 Medical Records Management System

Electronic Personal Records (EPR) Screen

The EPR screen uses a StatefulWidget to manage medical data with Firebase, allowing easy switching between personal and patient records while tracking specialty counts and search functions.

Add Document Screen

The Add Document screen features a form with multi-layered validation, checking required fields and offering real-time feedback with error messages.

The system uses two data loading methods for doctors and file types, retrieving info from different Firestore collections while considering patient context for doctor records.

A file handling system supports PDF selection, image gallery selection, and camera capture. It uses a StorageService class for consistent file operations and allows file previewing and removal.

The system allows easy addition of doctors during document creation, auto-filling their specialty

The system constructs MedicalFile objects with all necessary metadata including user context, file URLs from cloud storage, and formatted dates.

Both screens adjust based on viewing personal or patient records, ensuring data isolation and consistent interface patterns.

Edit Document Screen

The edit screen connects to Firebase services, retrieves medical file data, file types, and doctor information to set up the editing interface.

The file management system allows users to upload various file types like PDFs and images. Users can select files from their gallery, capture images, or choose PDFs. It updates the interface based on the file type and deletes old files during uploads.

The system uses a patient context provider to identify if the user edits their own files or a patient's files.

The system creates an updated medical file with form data including storage URL after validation, using Firestore to update documents consistently and securely.

Specialty Screen

The specialty screen fetches file type settings, retrieves medical files for the user or patient, and gets doctor information to minimize queries and ensure complete data availability.

The system allows detailed file searches using filters for file types, doctor assignments, dates, and formats, with real-time matching for precise results.

The screen allows complete file management with a menu for opening, editing, sharing, downloading, and deleting files. It adapts to file types for efficient access and deletion.

The system allows easy sharing of medical files via email, automatically filling in details for the user.

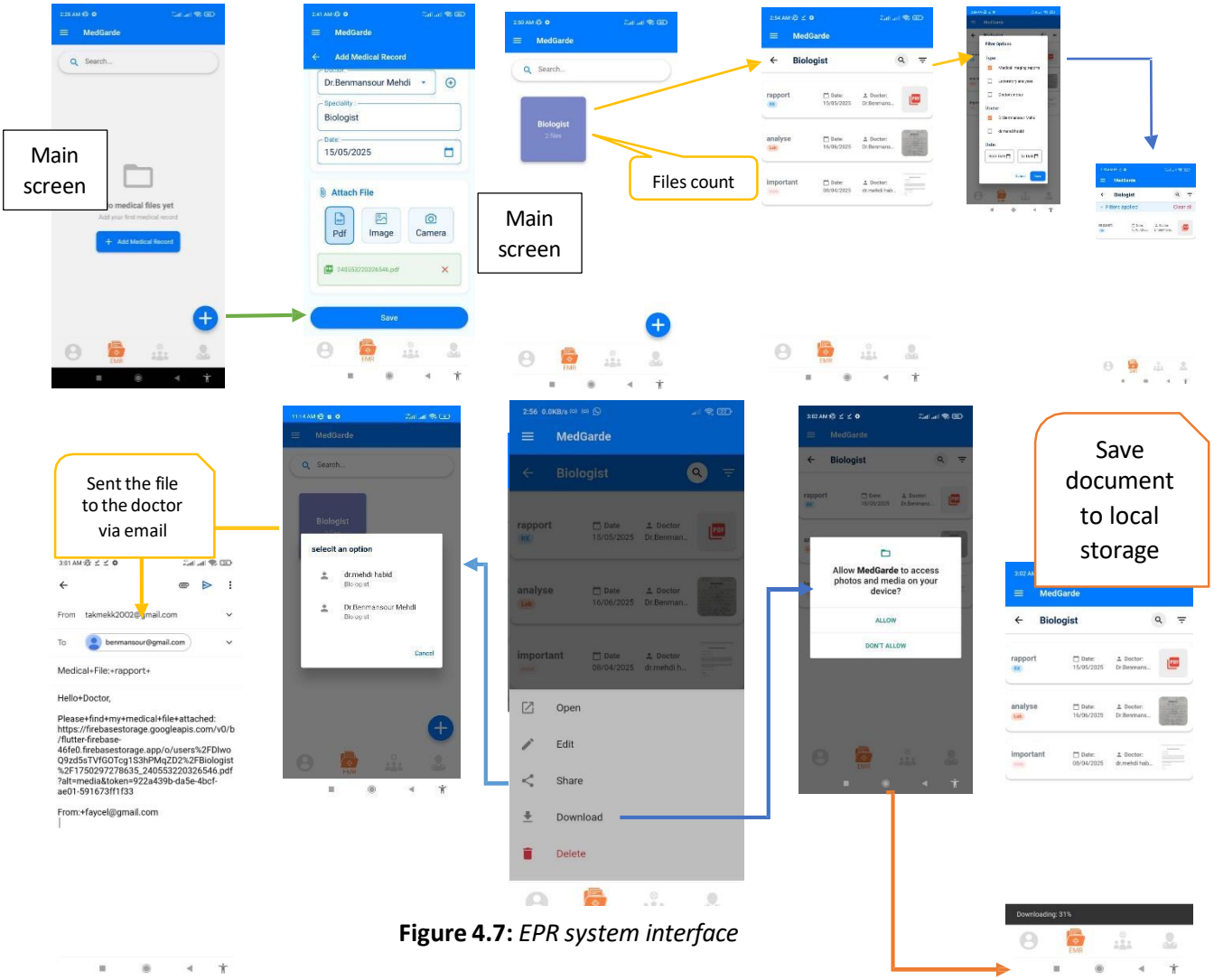


Figure 4.7: *EPR system interface*

5.6 Doctor Management System

The doctor management system is implemented as a comprehensive CRUD (Create, Read, Update, Delete) application that allows users to manage doctor information within a medical application.

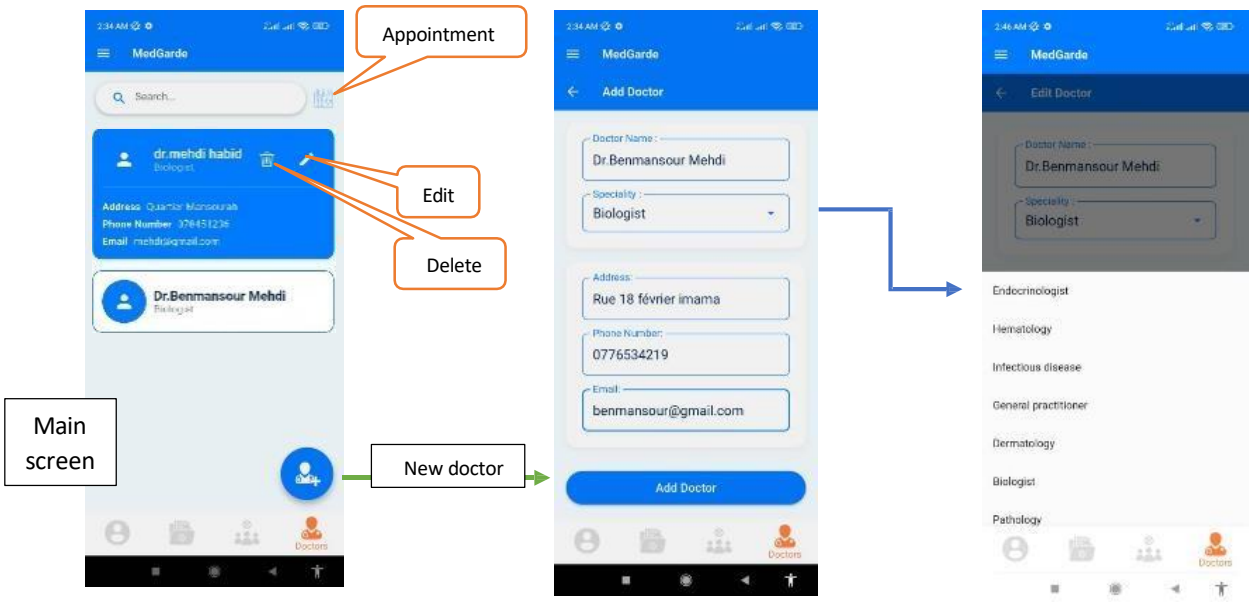


Figure 4.8: *Doctor Interface*

5.7 Appointment System

Add Appointment Screen

The Add Appointment Screen allows users to create medical appointments by entering titles, notes, selecting date and time, setting reminders, and choosing doctors from a dynamic list.

Date and time selection uses Flutter's pickers, preventing past date selection and limiting future appointments to one year ahead.

The reminder system provides options for no reminder, 30 minutes, 1 hour, or 1 day before an appointment, using a modal selection interface like the doctor picker.

Save Appointment Functionality

The appointment saving process checks required fields like title, doctor, date, and time, showing error messages if incomplete.

The system creates a complete appointment object with details like label, doctor ID, date, notes, patient ID, and reminder preferences, stored in Firestore.

The system schedules notifications based on user preferences, uses a unique appointment ID for management, and provides instant confirmation and success messages.

The app shows the appointment list and refreshes to confirm the new appointment created.

Appointments Display Screen

The Appointments Display Screen has a calendar to view and manage medical appointments, starting with the current date and fetching doctor details from the Firestore database.

The calendar interface offers a custom weekly view with navigation buttons, interactive days for date selection, and visual indicators for the current and past dates.

Real-time Data Synchronization

The appointment listing uses Firebase Firestore's real-time streaming with StreamBuilder to sync and instantly update filtered appointments by date and patient context.

The appointment display uses current time to show past appointments in muted colors, indicating their completed status.

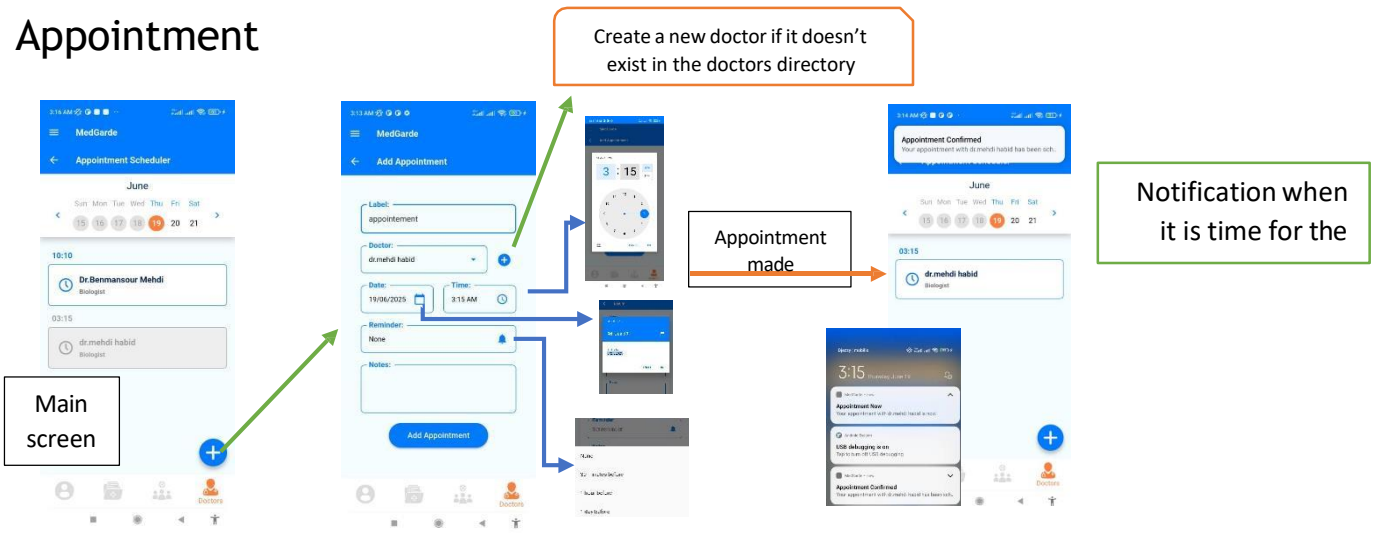


Figure 4. 9: *Appointment system interface*

6 Implementation Challenges and Solutions

6.1 Technical challenges:

❖ **Emulator Setup Problems:**

To run the code, an emulator was required. First, we tried using an online tool called Genymotion, but we couldn't configure it correctly with the Flutter project in VS Code. After that, we attempted to set up the Android Studio emulator, but also it didn't work and was too resource-intensive for the personal computer. Finally, we used a Flutter package called Device Preview (Version 1.2.0). While this package was functional, it wasn't very helpful and took a considerable amount of time to display the execution results.

❖ **Firebase Limitations**

- Security rules for Cloud Storage in Firebase may require a paid subscription
- Limited regions for Cloud Functions; North Africa areas not directly supported
- Selected Europe-west1 (Belgium) may increase latency for real-time applications
- Limited rule validation when offline affects patient profile editing

❖ **Authentication Issues**

- Difficulties in switching accounts between caregivers and patients
- Firebase error messages often lack clarity during authentication

❖ **Notification Challenges**

- Presenting notifications outside the app poses additional challenges
- Modifications needed to activate notifications at the correct local time for appointments

6.2 Adaptations made

❖ **Development and Testing**

• Real devices connected to Visual Studio Code

• Provides better performance evaluation than emulators

❖ Billing Plan

• Chose the Spark Plan for payment based on actual usage

• Included $300 promotional credit from Google for Firebase storage

6.3 Solved issues

• Code updates were implemented to address and resolve Firebase security rules conflicts.

• Integrated Provider package for better state management during account transitions

• Set up with Algerian location settings for accurate local time notifications.

7 Conclusion

This chapter explained how we developed the MedGarde application from start to finish, resulting in a functional mobile app. We chose Flutter for the interface and Firebase for data storage, which suited our project requirements. The app now includes all the key features we intended: users can register and log in securely, manage medical files, add patients, and schedule appointments. We also made it work in both Arabic and English, which is important for Algerian users. Even though we faced some problems with setting up the development tools and Firebase rules, we found solutions by testing on real phones instead of emulators. The final application proves that we can create a useful healthcare app within the limits of a master's thesis project.

Conclusion and Perspectives

Summary of work completed

This analysis looks at how well the MedGarde mobile health app meets its original goals in three main areas: managing user and patient accounts, handling and sharing medical files, and organizing a doctor directory along with appointment scheduling.

User and Patient Account Management

- Comprehensive User Registration: Complete profile creation with username, email, password, phone number, birth date, gender, blood type, and address information
- Secure Authentication System: Firebase Authentication with email/password verification, "Remember Me" functionality, and password recovery
- referential "Me" card allowing users to act as both caregiver and patient
- Caregiver-Patient Relationships: Two-pathway patient addition system (linking existing accounts or creating new ones)
- Profile Management: Context-aware editing for both personal and patient profiles with proper validation
- Arabic Language Support: Full RTL (Right-to-Left) text direction with over 1000 localized messages
- Patient Context Switching: Seamless switching between personal and caregiver modes
- Relationship Metadata: Advanced relationship title management and synchronization
- Security Enhancements: Two-step confirmation for patient relief responsibility, reauthentication for password changes
- Failed to edit patient profile personal information
- Failed to delete user account permanently
- No communication or collaboration between the caregivers of the single patient

Medical File Management

- Centralized Storage: Firebase Cloud Storage with 10MB file limit per upload,
- Multi-format Support: supporting images (JPG, PNG) and documents (PDF)preview and management
- A medical file is defined by the doctor and his speciality, date, type
- Document digitisation: The File handling system supports PDF selection, image gallery selection, and camera capture
- Comprehensive File Management: Complete CRUD operations (Create, Read, Update, Delete) for medical files
- Search and Filtering: by file types, doctor, dates, and formats
- Secure Access Control: only caregivers who they are responsible on the care recipient can manipulate his electronic personal medical record
- File Sharing with doctor: Email sharing functionality with automatic detail filling
- Real-time Data Synchronization: Firebase Firestore integration for instant updates
- Storage Service Integration: Consistent file operations through dedicated StorageService class

- Specialty-Based Organization: Normalized medical specialties collection for advanced search and filtering

Doctor Directory and Appointment Scheduling

- Comprehensive Doctor Management: Full CRUD system for doctor profiles with specialty management
- Dynamic Doctor Selection: Integration of doctor information in appointment and medical file systems
- Advanced Appointment System: Complete scheduling with title, notes, date/time selection, and reminder options
- Calendar Interface: Custom weekly view with navigation, interactive date selection, and visual indicators
- Reminder System: Multiple reminder options (30 minutes, 1 hour, 1 day or none) with +3.0
- Visual Calendar System: Interactive calendar with past/present date indicators/ inclear past. Present

Current limitations

- Failed to edit patient profile personal information
- Failed to delete user account permanently
- No communication or collaboration between the caregivers of the single patient
- No Offline Functionality
- Failed to implement scanning function
- Multi selection files to operate one option (delete, download or share)
- Healthcare Provider Networking: Basic doctor directory exists but no networking features for coordinate communication
- Appointment can’t be updated, deleted.
- Appointment details of the appointment can’t be read or updated

Improvement perspectives and future work

The foundation established by this thesis opens numerous pathways for expansion and enhancement

Better Medication Management

The current MedGarde system has basic appointment scheduling and file storage. We want to add smarter medication tracking with AI help. This will remind patients to refill their medicines and warn them about dangerous drug combinations. These features will help patients follow their treatment plans better.

Health Monitoring Features

MedGarde should connect with smartwatches and health devices to track vital signs, such as blood pressure, heart rate, and blood sugar. The app can then check this data to identify health issues early, before they become serious. This way, doctors can stop problems instead of just treating them later.

With these new features, MedGarde will become more than just a record-keeping app. It will be a full health monitoring system that offers personalized advice based on each patient's data.

Community Platform

We originally planned to build a community feature for MedGarde. This would be a safe social network where patients with the same diseases can connect and share experiences. Caregivers could give advice, and doctors could share educational content.

This community would help patients who feel lonely because of their health conditions.When people share knowledge and support one another, it leads to better treatment outcomes and a higher quality of life.

AI Integration

Artificial intelligence can achieve much more in MedGarde's future. AI can help doctors make better diagnoses by looking at symptoms and medical history together. Voice recognition can let users speak to the app instead of typing, making it simpler for those who aren’t comfortable with technology.

AI can also look at health data from many users to spot disease patterns and contribute to better public health. However, we must be very careful to protect patient privacy and keep their data secure.

Better User Interface

The app's design needs to be better. Future versions should include improved accessibility features like voice navigation for blind users and simpler interfaces for older people. We also need support for more languages besides Arabic and English.

The app should learn from each user's behavior and adjust to their needs. Young people who are comfortable with technology require different features than older patients who have difficulties with smartphones.

Following Healthcare Rules

To make MedGarde a complete healthcare platform, we must follow all healthcare laws and regulations. This includes telemedicine rules, data protection laws, and medical device approval processes. We need to work closely with government agencies to make sure our features are legal and beneficial for patients.

Long-term Goals

MedGarde can help improve healthcare in Algeria and other developing countries. The project can support the transition to digital healthcare and aid in creating national health information systems.

If MedGarde succeeds, it will show that healthcare technology works better when local teams develop it and adapt it to their culture. This may motivate similar projects in the region and prove that effective healthcare technology can be accessible to everyone, no matter their technical skills or financial situation.

References

References:

[1] PatientBetter, "Medical Record," PatientBetter Glossary. [Online]. Available: https://patientbetter.com/glossary/medical-record/. [Accessed: 16-Jun-2025].
[2] Meridiq, "What is a patient record?," Meridiq Blog. [Online]. Available: https://meridiq.com/en/what-is-a-patient-record/. [Accessed: 16-Jun-2025].
[3] College of Physicians and Surgeons of Nova Scotia, "Management of Medical Records," CPSNS Resources. [Online]. Available: https://cpsns.ns.ca/resource/management-of-medical-records/. [Accessed: 16-Jun-2025].
[4] Ditstek, "10 Components of Medical Records," Ditstek Blog. [Online]. Available: https://www.ditstek.com/blog/10-components-of-medical-records. [Accessed: 16-Jun-2025].
[5] "How to keep good clinical records," *PMC*, National Center for Biotechnology Information. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC5297955/. [Accessed: 16-Jun-2025].
[6] S. Boufissiou, "Analytical sight on the Algerian digital health strategy 2023–2027," *Akofena*, vol. 2, no. 14, pp. 7–18, 2024.
[7] "Open Access Macedonian Journal of Medical Sciences," *OAMJMS*. [Online]. Available: https://oamjms.eu/index.php/mjms/article/view/9509. [Accessed: 16-Jun-2025].
[8] *ISO/TR 14639-1:2012, "Health informatics — Capacity-based eHealth architecture roadmap — Part 1: Overview of national eHealth initiatives," International Organization for Standardization, 2012.*
[9] Park University, "Digital Health Records: Improving Efficiency and Patient Care," Park University Blog, 2024. [Online]. Available: https://www.park.edu/blog/digital-health-records-improving-efficiency-and-patient-care/. [Accessed: 16-Jun-2025].
[10] HealthIT.gov, "What are the advantages of electronic health records?" [Online]. Available: https://www.healthit.gov/faq/what-are-advantages-electronic-health-records. [Accessed: 16-Jun-2025].
[11] HealthIT.gov, "What are the differences between electronic medical records, electronic health records, and personal health records?" [Online]. Available: https://www.healthit.gov/faq/what-are-differences-between-electronic-medical-records-electronic-health-records-and-personal. [Accessed: 16-Jun-2025].

[12] StudyCorgi, "Electronic Health Records: Advantages and Disadvantages," 2022. [Online]. Available: https://studycorgi.com/electronic-health-records-advantages-and-disadvantages/. [Accessed: 16-Jun-2025].
[13] National Academies of Sciences, Engineering, and Medicine, "Family Caregiving," in *Families Caring for an Aging America*, Washington, DC: The National Academies Press, 2016. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK396398/. [Accessed: 16-Jun-2025].
[14] "Caregiver burden and health outcomes," *PMC*, National Center for Biotechnology Information. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC9122646/. [Accessed: 16-Jun-2025].
[15] National Alliance for Caregiving, "Caregiver Statistics: Demographics," Caregiver.org. [Online]. Available: https://www.caregiver.org/resource/caregiver-statistics-demographics/. [Accessed: 16-Jun-2025].
[16] "Medication adherence and family caregivers," *PMC*, National Center for Biotechnology Information. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC5690891/. [Accessed: 16-Jun-2025].
[17] "Psychosocial factors and caregiver burden," *BMC Primary Care*, vol. 24, article 65, 2023. [Online]. Available: https://bmcprimcare.biomedcentral.com/articles/10.1186/s12875-023-01985-y. [Accessed: 16-Jun-2025].
[18] "Burden and associated factors for caregivers of the elderly in a developing country," *Eastern Mediterranean Health Journal*, vol. 22, no. 6, 2016. [Online]. Available: https://www.emro.who.int/emhj-volume-22-2016/volume-22-issue-6/burden-and-associated-factors-for-caregivers-of-the-elderly-in-a-developing-country.html. [Accessed: 16-Jun-2025].
[19] "Impact of caring for family members with mental illnesses," *Health Promotion International*, vol. 38, no. 3, article daac049, 2022. [Online]. Available: https://academic.oup.com/heapro/article/38/3/daac049/6574394. [Accessed: 16-Jun-2025].
[20] ISO/TR 14292:2012, "Health informatics — Personal health records — Definition, scope and context," International Organization for Standardization, 2012.

App references :

[21] "Mon espace santé," Ministère de la Santé et de la Prévention, France. [Online]. Available: https://www.monespacesante.fr/. [Accessed: Jun. 17, 2025].
[22] "Consult top doctors online for any health concern," Practo. [Online]. Available: https://www.practo.com/. [Accessed: Jun. 17, 2025].
[23] "CareClinic: Personalized Symptom & Wellness Tracker," CareClinic Inc. [Online]. Available: https://careclinic.io/. [Accessed: Jun. 17, 2025].
[24] "Medisafe Pill & Med Reminder," Medisafe Inc. [Online]. Available: https://www.medisafe.com/. [Accessed: Jun. 17, 2025].
[25] "CareZone for Android - Download the APK from Uptodown," Uptodown Technologies S.L. [Online]. Available: https://carezone.en.uptodown.com/android. [Accessed: Jun. 17, 2025].
[26] "MyTherapy Pill Reminder & Dosage Tracker," smartpatient GmbH. [Online]. Available: https://www.mytherapyapp.com/. [Accessed: Jun. 17, 2025].
[27] "Dosecast - Medication Reminder," Google Play Store. [Online]. Available: https://play.google.com/store/apps/details?id=com.medhelper&hl=ar. [Accessed: Jun. 17, 2025].
[28] "Pillo Health," Pillo Inc. [Online]. Available: https://www.pillo.care/. [Accessed: Jun. 17, 2025].
[29] "iCompanion," Microsoft Corporation. [Online]. Available: https://icompanion.ms/. [Accessed: Jun. 17, 2025].
[30] "Aidant familial - Tous les produits pro," Dynseo. [Online]. Available: https://www.dynseo.com/tous-les-produits-pro/aidant-familial/. [Accessed: Jun. 17, 2025].
[31] "Carenity - The social network for patients and caregivers," Carenity. [Online]. Available: https://www.carenity.com/. [Accessed: Jun. 17, 2025].

Annexes

Annex1: Development Environment Setup

1 Hardware Specifications and System Requirements

The development environment was established on a Windows 10 workstation with the following specifications:

- Processor: Intel(R) Core(TM) i3-4005U CPU @ 1.70GHz 1.70 GHz.
- Memory: 8.00 GB.
- Storage: 476GB SSD with 270GB available space.
- Display: 1360 x 786 resolution.

2 Flutter Environment Configuration

The Flutter environment setup provided the foundation for the mobile application development process. This section details the systematic installation, configuration, and optimization of the Flutter development environment, including SDK installation, IDE configuration, and device setup.

2.1 Flutter SDK Installation

The Flutter SDK installation followed the official documentation methodology with additional verification steps:

❖ **Step 1: SDK Download and Extraction:**

The Flutter SDK was downloaded from the official repository (https://flutter.dev/docs/get-started/install/windows) as a ZIP archive. The archive was extracted to c:/flutter to ensure path compatibility and avoid space-related issues.

❖ **Step 2: Environment Variables Configuration:**

System PATH environment variable was modified to include the Flutter binary directory:

```
Path Variable: C:\flutter\bin
```

❖ **Step 3: Flutter Doctor Diagnostic:**

The initial system compatibility check was performed using the Flutter doctor command:

```
flutter doctor –v.
```

The flutter doctor output identified several missing components:

- Android toolchain - develop for Android devices
- Chrome - develop for the web
- VS Code
- Connected device

❖ **Step 4: Dependency Resolution:**

Each identified dependency was systematically addressed through Android toolchain installation:

- Android SDK Command-line Tools (latest) - Version 16.0
- Android SDK Build-Tools 35.0.0
- Android SDK Platform-Tools 35.0.0

❖ **Step 5: License Acceptance**

Android SDK licenses were accepted using the command:

```
flutter doctor --android-licenses
```

❖ **Final Verification:**

Post-installation flutter doctor output confirmed successful setup with all components properly configured.

```
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 3.29.0, on Microsoft Windows [version 10.0.19045.5854], locale fr-FR)
[√] Windows Version (10 Professionnel 64-bit, 22H2, 2009)
[√] Android toolchain - develop for Android devices (Android SDK version 34.0.0)
[√] Chrome - develop for the web
[!] Visual Studio - develop Windows apps (Visual Studio Build Tools 2019 16.11.44)
    X The current Visual Studio installation is incomplete.
      Please use Visual Studio Installer to complete the installation or reinstall Visual Studio.
[!] Android Studio (version 2024.2)
    X Unable to determine bundled Java version.
[√] VS Code (version 1.100.2)
[√] Connected device (3 available)
[√] Network resources
```

2.2 IDE Configuration: Visual Studio Code

Visual Studio Code version 1.100.1 (user setup)- updated in May 2025- was configured with essential extensions optimized for Flutter development productivity:

- Flutter Extension (v3.111.0):
  - Publisher: Dart-Code
  - Features Enabled:
    - Flutter widget inspector integration
    - Hot reload and hot restart functionality
    - Flutter outline view for widget hierarchy
    - Automatic imports and code completion
    - Integrated terminal for Flutter commands

- Dart Extension (v3.111.0):
  - Publisher: Dart-Code
  - features:
    - Automatic hot reloads for Flutter
    - Quickly switch between devices for Flutter
    - Automatically finds SDKs from PATH
    - Notification of new stable Dart SDK releases
    - Real time errors and warnings
    - Documentation in hovers and tooltips

These extensions are regularly updated to ensure compatibility with the latest VS Code version, streamlining workflows across debugging, version control, and project execution.

2.3 Testing Device Configuration

The application was primarily tested on a physical Android device to ensure real-world performance and user experience validation.

2.3.1 Device Specifications

- Device Model: Xiaomi Redmi 10 5G
- Brand: Xiaomi
- Android Version: Android 11 (MIUI 12.5.6)
- CPU: Octa-core (Max 2.2 GHz)
- RAM: 4.00 GB
- Internal Storage: 128 GB
- Screen Resolution: 1080 x 2408 pixels

The Xiaomi Redmi 10 5G was selected as the primary testing device for several reasons:

- **Target User Representation**: This mid-range Android device reflects a significant segment of the target user base
- **Performance Assessment**: The device provides adequate specifications to evaluate application performance under typical user conditions
- **UI Validation**: With standard HD+ resolution and 90Hz refresh rate, it effectively evaluates UI responsiveness and visual performance
- **Accessibility**: Available as a student device for consistent testing throughout development

2.3.2 Device Preparation and Connection Protocol:

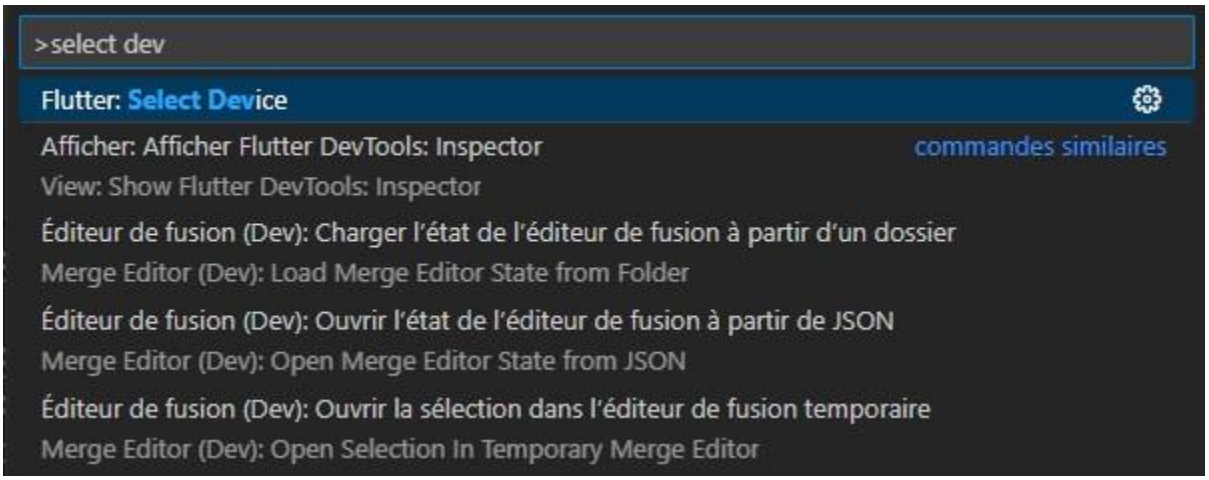
This process has several steps that help connect the development environment to the target hardware smoothly.

- **Step 1: Developer Mode Activation:**
  - Navigate to the device's settings.
  - Go to the "About Device" section.
  - Locate the device's build number and tap on it several times until you see the confirmation message for developer mode.
- **Step 2: USB Debugging Configuration:**
  - The USB debugging option can be used after accessing developer mode:
  - In the Settings menu, go to Developer Options.
  - Scroll down to find the Debugging section and then activate the USB Debugging toggle.
- **Step3 Device Recognition Verification:**
  - Connect the Android device to the computer using a USB cable, ensuring that the cable supports data transmission.
  - A prompt will appear on the phone asking for permission to allow USB debugging. Accept the permission to proceed.
  - Open the VSCode terminal and run the command flutter devices. This will display all the devices available in the system.

```
PS C:\Users\user\Desktop\MedGarde\FlutterProjects\first_pages> flutter devices
Found 4 connected devices:
  M2103K19C (mobile) • n7mrbyv8xw6h79z9 • android-arm64  • Android 11 (API 30)
  Windows (desktop)  • windows          • windows-x64    • Microsoft Windows [version 10.0.19045.5965]
  Chrome (web)       • chrome           • web-javascript • Google Chrome 137.0.7151.104
  Edge (web)         • edge             • web-javascript • Microsoft Edge 137.0.3296.68
```

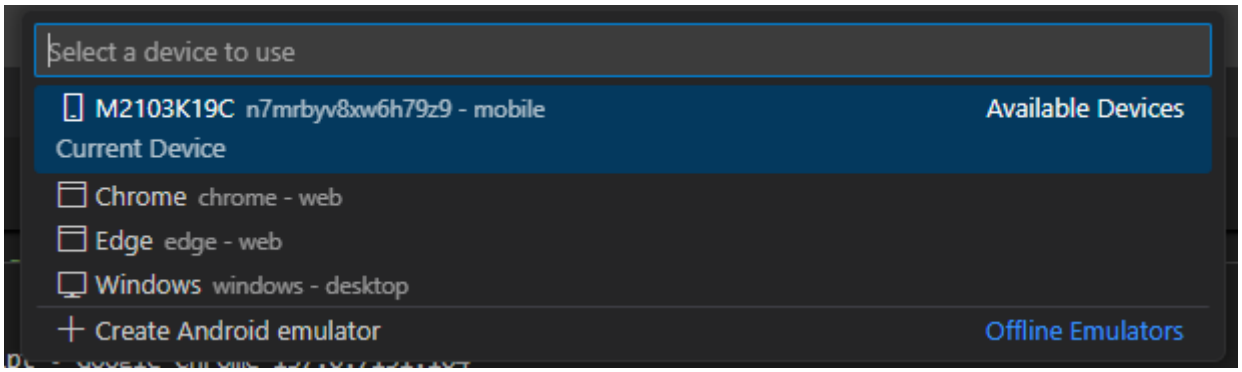
- **Step 4: Integrated Development Environment Configuration:**

- Open the Flutter project root in VSCode. Press Ctrl+Shift+P to select the real device, then click Enter.



❖ **Step 5: Application Deployment and Execution:**

- Select your device as an emulator to run your Flutter project.



- Press F5 or go to "Debug > Start Debugging" in the VSCode menu.
- The Flutter project will be built and installed on the physical device.
- It can then view the output and interact with the app on the device's screen.

2.4 Flutter Package Dependencies

The project utilized a curated selection of Flutter packages, each chosen for specific functionality and maintained compatibility. The complete dependency configuration was defined in pubspec.yaml:

```
dependencies:
  flutter:
    sdk: flutter
  flutter_localizations:
    sdk: flutter
```

localizator version 0.3.2

Core Dependencies	
firebase_core	^3.13.0
cloud_firestore	^5.6.6
firebase_authfireb	^5.5.2
firebase_storage	^12.4.5
UI and User Experience Dependencies	
flutter_native_splash	^2.4.6
intl_phone_field	3.2.0
flutter_holo_date_picker	^2.0.0
drop_down_list	2.0.0
animate_do	^4.2.0
cupertino_icons	^1.0.8
flutter_localizations	
Development Dependencies	
flutterfire_cli	^1.2.0

Utility and Functionality Dependencies	
shared_preferences	^2.5.3
email_validator	3.0.0
intl	^0.19.0
provider	^6.1.4
http	^1.3.0
image_picker	^1.1.2
file_picker	^10.1.2
path	^1.9.1
permission_handler	12.0.0+1
open_file	^3.5.10
path_provider	^2.1.5
dio	^5.8.0+1
clipboard	^0.1.3
flutter_email_sender	^7.0.0
url_launcher	^6.2.4
cached_network_image	^3.3.1
flutter_local_notifications	^19.1.0
timezone	^0.10.1

3 Firebase Configuration

Firebase was selected as the Backend-as-a-Service (BaaS) solution to provide scalable cloud infrastructure for user authentication, Cloud Firestore database with appropriate security rules, and file storage capabilities.

3.1 Firebase Project Setup

The Firebase project setup involved the following steps:

1. **Create Firebase project**: A new project was established through Firebase Console (console.firebase.google.com) to serve as the central backend infrastructure.
2. **Initialize with Google account**: The project was set up using a personal Google account for quick setup and direct ownership.
3. **Choose europe-west1 region**: This region was selected for geographic proximity and data residency compliance.
4. **Enable Google Analytics**: Google Analytics for Firebase was activated to collect user engagement metrics for data-driven decision-making.

3.2 Firebase CLI Installation and Authentication

The Firebase CLI setup for Visual Studio Code and Flutter involved the following configuration steps:

1. **Configure Node.js** - Node.js was installed as a prerequisite for Firebase CLI functionality.

2. **Install Firebase CLI globally** - The CLI was installed globally using npm to provide command access across the system.
3. **Integrate with Visual Studio Code** - The CLI was accessed via the integrated terminal to manage Firebase features including authentication, Firestore, and cloud functions.
4. **Configure for Flutter projects** - The CLI was used to deploy and configure backend resources according to the pubspec.yaml file specifications.
5. **Link Firebase project** - Authentication and initialization commands were executed to save configuration files in the project directory for seamless development access.

3.3 Firebase Services Configuration

3.3.1 Authentication Service Setup

The project utilizes Email/Password Authentication as the primary method for managing user identities. This approach allows users to register and log in using a unique email address coupled with a secure password. Firebase Authentication manages the backend processes, providing a secure and scalable solution that integrates seamlessly with other Firebase services.

Password Policy Configuration:

- Minimum Length: 8 characters
- Complexity Requirements: Mixed case letters, numbers, and special characters
- Email Verification: Required for new user registrations
- Password Reset: Enabled through email verification process

3.3.2 Cloud Firestore Database

Cloud Firestore was configured as the primary NoSQL database solution for real-time data storage and synchronization.

Data base Setup Parameters:

- Database Mode: Native mode for optimal performance
- Location: europe-west1 (matching project location)
- Database ID: (default) primary database instance
- Billing Mode: Pay-as-you-go for flexible resource scaling

Data Modelling Strategy:

Collection	Purpose
Users Collection	Stores user profiles, personal information and contact details.
Doctors Collection	Contains professional details for doctors.
Care Relationships Collection	Manages the relational mapping between patients and caregiver.
Appointments Collection	Handles scheduling data for medical visits.
Medical Files Collection	Acts as a storage reference for uploaded healthcare documents such as reports, prescriptions, and imaging.

File Types Collection	Defines classification types for uploaded medical files (e.g., lab report, prescription, and scan), ensuring structured filtering and validation during file upload.
Specialties Collection	Maintains a normalized list of medical specialties linked to doctor profiles, supporting advanced search and filtering capabilities

3.3.3 Cloud Storage

Firebase Cloud Storage was configured for file upload, storage, and retrieval capabilities.

Storage Bucket Configuration:

- Default Bucket: thesis-mobile-app-2024.appspot.com
- Storage Location: europe-west1 (matching database location)
- Access Control: Authenticated users only with custom rules
- File Size Limits: 10MB maximum per individual file upload
- Supported File Types: Images (JPG, PNG), Documents (PDF)

3.3.4 Security Rules Configuration

Authentication Enforcement: Every request must originate from an authenticated user through Firebase Authentication (request.auth!= null), ensuring that only verified users can access application resources.
Role-Based Access Control: Access levels (e.g., doctor, patient, admin) are defined using user metadata or custom claims, providing granular permission management based on user roles.
Document Ownership Verification: Users are granted access only to documents where their UID corresponds to a designated field (e.g., userId == request.auth.uid), ensuring data privacy and user-specific access control.
Operation Separation: Detailed rules establish distinct conditions for reading, creating, updating, and deleting documents, providing fine-grained control over data manipulation operations.
Data Validation: Rules ensure that incoming data adheres to the expected format, preventing empty appointment fields or invalid timestamps, thereby maintaining data integrity throughout the application.

The comprehensive Firebase configuration provided a secure, scalable backend infrastructure supporting authentication, data storage, and file management capabilities essential for the mobile application's functionality and user experience.

Abstract:

Healthcare delivery in developing nations, particularly Algeria, faces significant challenges due to fragmented medical record systems, inadequate digital infrastructure, and limited access to suitable mobile health solutions. This thesis presents the design, development, and implementation of MedGarde, a comprehensive mobile health application built with Flutter as the front-end framework and Firebase as the backend platform and database storage, specifically tailored to address Algeria's healthcare challenges while considering local language, culture, and technology constraints.**Keywords:** Mobile Application, Patient Monitoring, Caregiver, Flutter, Firebase, Medical Records, Healthcare App.

Résumé:

La prestation de soins de santé dans les pays en voie de développement, en particulier en Algérie, est confrontée à des défis importants en raison de systèmes de dossiers médicaux fragmentés, d'une infrastructure numérique inadéquate et d'un accès limité à des solutions de santé mobile appropriées. Cette thèse présente la conception, le développement et la mise en œuvre de MedGarde, une application de santé mobile complète construite avec Flutter comme framework frontal et Firebase en tant que plateforme backend et solution de stockage et base de données. Elle est spécifiquement conçue pour répondre aux défis du système de santé en Algérie, tout en tenant compte de la langue, de la culture et des contraintes technologiques locales.**Mots-clés:** Application mobile, suivi des patients, garde malade, flutter, firebase, dossies médicaux, application de santé.

ملخص :

يواجه تقديم الرعاية الصحية في الدول النامية، ولا سيما الجزائر، تحديات كبيرة بسبب أنظمة السجلات الطبية المجزأة، وعدم كفاية البنية التحتية الرقمية، ومحدودية الوصول إلى الحلول الصحية المتنقلة المناسبة. تقدم هذه الأطروحة تصميم وتطوير وتنفيذ تطبيقMedGarde ، وهو تطبيق شامل للرعاية الصحية على الهاتف المحمول تم تصميمه باستخدام فلاتر كإطار عمل للواجهة الأمامية وفاير بايز كمنصة خلفية وتخزين قاعدة البيانات، وهو مصمم خصيصًا لمواجهة تحديات الرعاية الصحية في الجزائر مع مراعاة القيود اللغوية والثقافية والتقنية المحلية. **الكلمات المفتاحية:** تطبيق الهاتف المحمول, متابعة المريض, مقدم الرعاية, فلاتر, فايربايز, السجلات الطبية, تطبيق الرعاية الصحية.