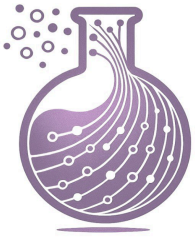
Kingdom of Saudi Arabia
Ministry of Education
College of Computer
Computer Science Department

Qassim University جامعة القصيم

المملكة العربية السعودية
وزارة التعليم
كلية الحاسب
قسم علوم الحاسب



SmartLab

A mobile application AI-powered medical tests analyzer.

Students:

Amjad Al-Mutiri	**391202735**
Elham Al-Suyyan	**411202269**
Ghzlan Al-Rashidi	**421202118**

Supervisor:

Ghaida Abdalaziz Al-Hamidi.

A project report submitted to Qassim University in partial fulfillment of the requirements for the degree of B.Sc. in Computer Science

Qassim - Saudi Arabia

1446/1447 (2024/2025)

CERTIFICATE

It is certified that this project report has been prepared and written under my direct supervision and guidance. This project report is approved for submission for its evaluation.

Teacher: Ghaida Alhamidi

DEDICATION

This project is dedicated to those who strive to make healthcare accessible and understandable to all individuals.

Amjad Al-Mutiri

Elham Al-Suyyan

Ghzlan Al-Rashidi

ACKNOWLEDGEMENT

To those with whom we shared a common dream and a path of shared learning, we dedicate this project with appreciation and gratitude. To the amazing team, who spared no effort in achieving this accomplishment, the moments of collaboration were filled with challenges and creativity, and together, we overcame every obstacle. Thank you to every member of the team for your determination, persistence, and dedication; the success we celebrate today is the result of everyone's effort and sincere cooperation.

To our dear families, who supported and encouraged us throughout our academic journey, we could not have reached this day without your continuous support and constant motivation. To our esteemed teachers, who provided us with knowledge and guidance, we extend our deepest thanks and appreciation for your invaluable efforts.

We also dedicate this project to our colleagues and friends, who were always by our side throughout this journey, and to everyone who played a role, big or small, in our success.

Lastly, we dedicate this work to all who believe in the importance of collaboration and teamwork in achieving great successes, and to everyone who supported and encouraged us along this special journey.

Amjad Al-Mutiri

Elham Al-Suyyan

Ghzlan Al-Rashidi

Contents

List of Tables

List of Figures

ABSTRACT

SmartLab is an AI-powered mobile application developed using Flutter and Dart, designed to simplify and interpret medical laboratory test results for general users. The application leverages advanced OCR (Optical Character Recognition) technologies, including integration with the OCR.space API and planned enhancements using Google Vision AI, to accurately extract text from medical reports. Extracted data is then analyzed using the Gemini AI service to generate clear, concise summaries and provide actionable health improvement tips. SmartLab offers features such as historical test result storage and bilingual support (English and Arabic) through dynamic language switching. By combining machine learning with user-friendly interfaces, SmartLab empowers users to better understand their health data, identify potential risks, and track medical progress effectively. The project highlights a modular code structure, secure local database integration, and a scalable architecture that facilitates future expansions, including disease prediction based on lab results.The final system achieved an overall text extraction and interpretation accuracy of approximately 92%, ensuring reliable and user-friendly health insights.

CHAPTER ONE

INTRODUCTION

1.1 Introduction

Artificial Intelligence (AI) is increasingly being utilized across various fields to simplify complex tasks, particularly in data analysis. AI's ability to process and interpret large amounts of data quickly has transformed many aspects of everyday life, including healthcare. One significant area where AI has made an impact is in the analysis of medical data, helping individuals understand and use health information more effectively [1]. However, many people—especially older adults and those without a medical background—still struggle to comprehend medical reports due to the presence of complicated terminology and symbols [2].

This project aims to address this challenge by developing a mobile application, SmartLab, that leverages AI to simplify the analysis of medical reports. The app allows users to upload or capture an image of their medical report, then uses OCR and AI algorithms to extract and interpret the data. The analysis is presented in plain, user-friendly language along with personalized health tips derived from the report.

In addition to AI-driven interpretation, the application includes key features such as user registration and login, with the option to continue as a guest. It also supports sending the analysis results via email, giving users the flexibility to save or share their reports. All user data and reports are securely stored using an integrated SQLite database, allowing users to track changes over time.

1.2 Project Scope

The project will focus on building a mobile app using Flutter. Flutter is a tool used to create apps that can work on Android and iOS. The app will use Gemini API to analyze medical reports and provide easy-to-understand explanations. Additionally, the system includes a database to store user information and their previously analyzed reports, allowing users to view and track their medical history over time. The user can upload a file or take a picture of their medical report. Following that, the app will process the medical report and give them a simple explanation of what it means with basic health advice based on the report, And send the test analysis via Email according to the users desire. Finally, SmartLab application will support two languages Arabic and English to reach more users.

1.3 Aim and Objectives

The main goal of this project is to create an easy-to-use app that helps users understand and analyze their medical reports.
Thus, the main objectives of this research are to:

- Build a cross-platform mobile application using Flutter, compatible with both Android and iOS.
- Integrate the OCR.space API to extract text from medical reports in image or PDF formats.
- Utilize the Gemini API to analyze and simplify the extracted medical data.
- Provide personalized health advice based on the interpreted results.
- Design an intuitive user interface that supports both Arabic and English.
- Implement secure user registration, login, and guest access.
- Enable users to send analysis results via email and store previous reports using a local SQLite database.

1.4 Motivation

The motivation for this project comes from the difficulties that many people face when trying to understand their medical reports. For people who are not familiar with medical terms, it can be confusing and stressful. This is especially true for older individuals, who may not always have someone to explain the report to them.

This app will provide an easy way for users to understand what their medical reports are saying, making healthcare information more accessible. By giving users clear, simple explanations and advice, the app helps them manage their health better and feel more in control.

1.5 Project Plan and Schedule

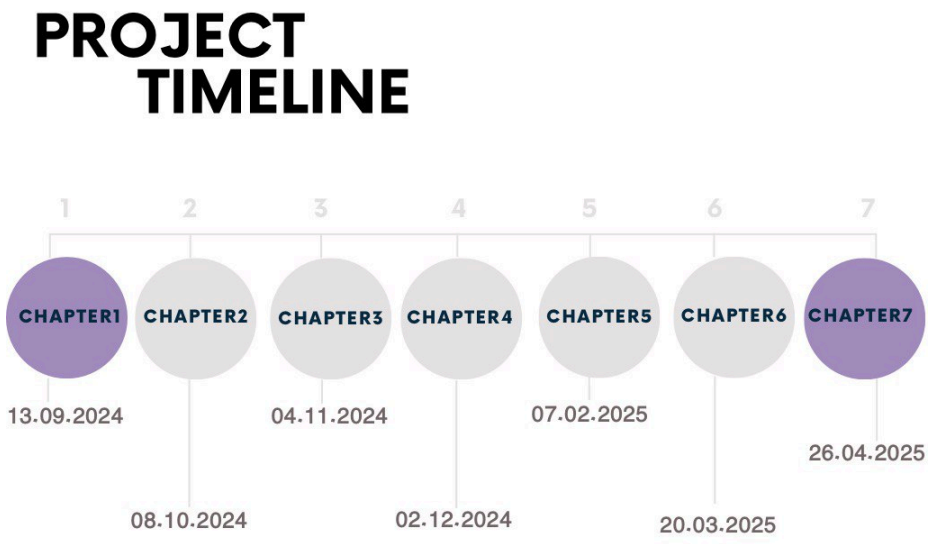


Figure 1: Project Plan and Schedule.

1.6 Outline of the Report

The report is organized into the following chapters:

- *Chapter One (Introduction):* Provides an overview of the project, including its scope, objectives, motivation, and the project plan.
- *Chapter Two (Literature Review and Methodology):* Reviews existing technologies and similar applications. Discusses the proposed system and the methodologies employed.
- *Chapter Three (System Analysis):* Covers the problems users face with medical report comprehension and details how the app addresses these issues.
- *Chapter Four (System Design):* Describes the technical structure of the app, including system architecture, interfaces, and data management.
- Chapter Five(System Implementation): This chapter explains the implementation phase of the project. It details the development environment, tools, and programming languages used. It also includes the steps taken to build the system's core functionalities, such as user interface development, backend integration, and database connection.
- Chapter Six(System Testing and Result Discussion): This chapter focuses on the testing strategies used to validate the system's functionality, performance, and usability. It covers unit testing, integration testing, and user acceptance testing. The results of the tests are discussed in detail, highlighting any issues discovered and how they were resolved.
- Chapter seven(Conclusion and future work): The final chapter summarizes the key achievements of the project and reflects on the challenges encountered during the development process. It also discusses the limitations of the current system and suggests potential enhancements and features that could be added in the future to improve the system’s usability and efficiency.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

The healthcare industry has seen an increase in digital tools and mobile applications aimed at helping individuals better understand their health. Many apps provide users with access to medical information, track their health, or offer advice from professionals. However, very few apps specifically focus on simplifying complex medical reports, especially for users without a medical background [3]. This chapter reviews some of the existing applications that aim to bridge the gap between complex medical data and the general public and highlights the unique features of the proposed app.

2.2 Background

In recent years, healthcare has moved more into the digital world, with many providers now offering online access to medical records, lab results, and other health documents. Although this has made it easier for people to access their medical information, it has also created new challenges. Many patients, especially those who don't have a medical background, find the medical language, symbols, and terms difficult to understand. This is particularly hard for older adults, who may not be familiar with medical terms or comfortable using digital tools to view their health information [4].

2.2.1 Optical Character Recognition (OCR) Process

Optical Character Recognition (OCR) technology plays a critical role in enhancing digital healthcare systems by enabling automated text extraction from medical documents [5]. In the SmartLab application, OCR functionality is implemented using the OCR.space API, a powerful cloud-based service that streamlines the recognition process while offering high accuracy and support for multiple languages, including Arabic and English[9].

Unlike traditional OCR approaches that rely on multiple local steps—such as manual scanning, preprocessing, segmentation, and post-processing—the OCR.space API handles the entire OCR pipeline remotely, significantly reducing implementation complexity and processing time[6]. Users simply upload a file, and the API returns the extracted text in a structured format, ready for AI-based analysis[7].

The main internal stages handled by OCR.space include:

1- **Image Analysis and Preprocessing:** OCR.space automatically adjusts contrast, removes noise, corrects skew, and converts documents to a suitable format for accurate recognition.
2- **Text Detection and Segmentation:** It detects lines, words, and characters without the need for manual segmentation logic[5].
3- **AI-Powered Recognition:** The API uses advanced recognition models optimized for printed and typed documents, including support for Arabic script, which is essential for medical reports in regional contexts[9].
4- **Error Handling and Confidence Scoring:** The output includes text along with confidence levels, helping developers filter out low-quality results or trigger resubmission if needed.
5- **Format Conversion:** The recognized text can be returned in plain text, searchable PDF, or JSON formats, making it suitable for integration and storage.

By relying on OCR.space, SmartLab ensures that the document digitization process is fast, efficient, and accurate—serving as the foundation for further AI-driven analysis and personalized health recommendations**.**

2.2.2 Technologies Powering OCR Space API

1. **Deep Neural Networks (DNN):** Power the backend recognition engine for learning from large datasets and improving accuracy over time.
2. **Convolutional Neural Networks (CNN**): Handle image feature extraction and are especially effective for interpreting visual content in medical documents.
3. **Cloud-Based Scalability:** As a cloud API, OCR Space leverages scalable architecture to process large volumes of documents without local hardware dependencies.
4. **Multi-language and Multi-format Support**: Supports multiple languages and input formats, making it ideal for global healthcare use cases.

2.2.3 Advantages and Disadvantages of Using OCR space API

• **Advantages :**

1- **Speed and Automation:** Instantly extracts data from documents without manual input.
2- **Accessibility:** Converts static medical data into editable, searchable formats.
3- **Accuracy:** High precision in printed and handwritten recognition with built-in AI improvements.

• **Disadvantages :**

1- **Limited Handwriting support :** OCR.space performs best with typed or printed text, handwritten reports may yield partial results.
2- **Internet Dependency :** Being cloud-based, the service requires a stable internet connection to function.
3- **Output Control :** Less flexibility in customizing internal OCR behavior compared to local open-source solutions.

2.2.4 Applications of OCR.space in Daily Life

OCR technology, particularly via cloud platforms like OCR.space, has broad applications in everyday life and digital transformation efforts:

• **Healthcare:** Digitizing and extracting data from lab reports, prescriptions, and medical forms.
• **Finance:** Processing invoices, receipts, and checks for record-keeping.
• **Education:** Converting printed textbooks and documents into editable formats.
• **Transportation:** License plate recognition and smart ticketing systems.
• **Government and Legal:** Digitization of official paper documents and archives[21].

In SmartLab, OCR serves as a gateway that transforms medical report images into readable and analyzable text—forming the basis for AI-driven health insights that help users understand their health status more clearly.

2.3 Existing Related Systems

2.3.1 MyChart

MyChart is a widely used health management app that allows users to view their medical records, schedule appointments, and communicate with their healthcare providers. While it provides access to medical information, it does not offer simplified explanations of medical reports. Users can view their lab results, but they need to interpret the data themselves, which can be challenging for non-medical users [20].

Limitations:

- Lacks simplified explanations of medical data.
- Does not provide personalized health advice.
- Focuses more on appointment scheduling and communication with healthcare providers than on explaining reports.

Figure 2: MyChart app [10].

2.3.2 WebMD

WebMD is a popular health information platform that offers a symptom checker, general health advice, and information about diseases, medications, and treatments. It provides users with a broad range of health content but does not specifically simplify medical reports. It requires users to actively search for information, and it provides general knowledge rather than personalized insights [22].

Limitations:

- Does not provide a simplified interpretation of specific medical reports, making it difficult for users to understand detailed health data.
- Provides generic health information, not tailored to individual users based on their reports.
- Requires users to actively search for information, which can be challenging and overwhelming, particularly for elderly individuals

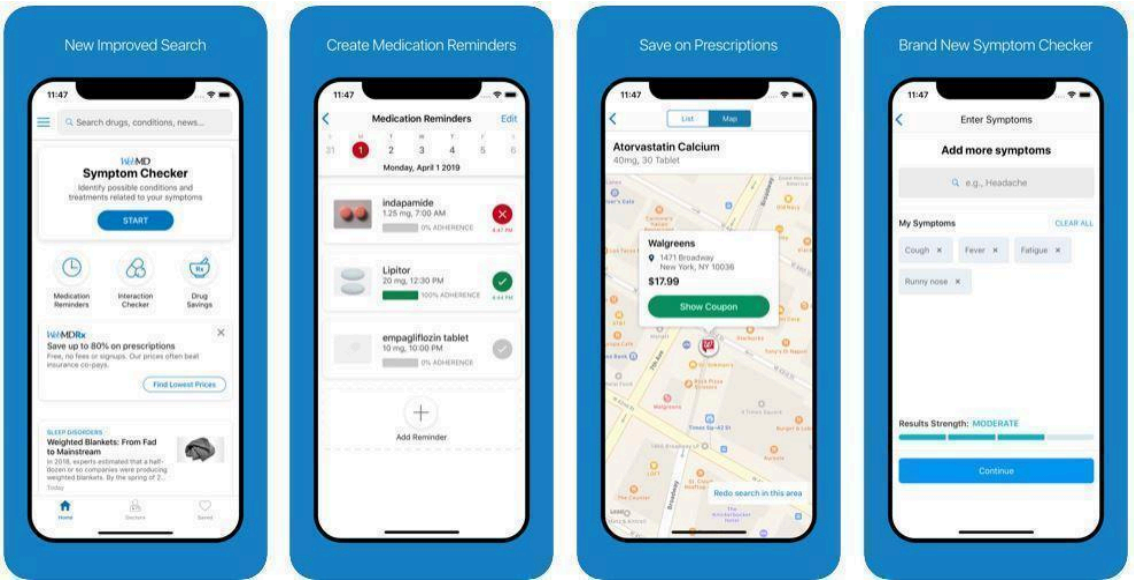


Figure 3: WebMD [11] .

2.3.3 Ada Health

Ada Health is an AI-driven health app that allows users to input their symptoms and receive potential diagnoses. It uses a symptom-checker model where users answer questions about their symptoms, and the app suggests possible health conditions. While Ada provides personalized health advice, it focuses on symptoms rather than interpreting medical reports like lab results or diagnostic documents [23].

Limitations:

- Focuses on symptom checking, not on simplifying medical reports.
- Does not allow users to upload or scan actual medical documents.
- Provides potential diagnoses but lacks an explanation of medical analysis results.

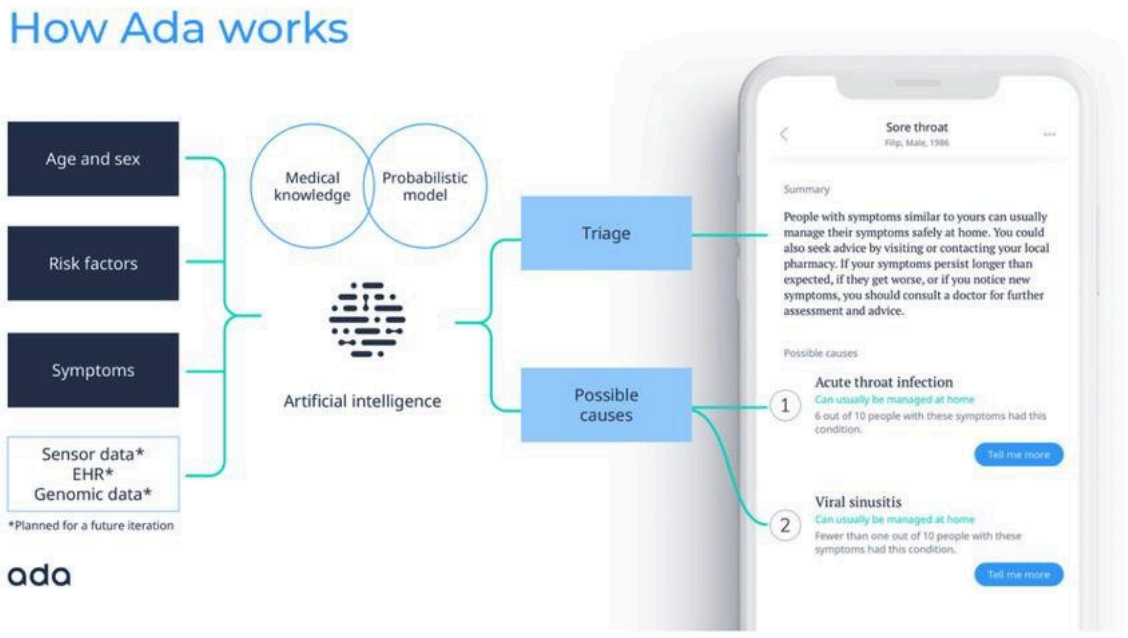


Figure 4: Ada Health [12].

2.3.4 LabCorp | Patient

LabCorp's app allows users to access lab test results directly from their smartphones. While this app enables users to view their lab results, it does not offer simplified explanations or personalized advice based on the data. The medical information provided is often presented in complex terms, leaving users to either research the terms or consult with their healthcare provider for clarity [24].

Limitations:

- Offers access to lab results but no simplification of medical terms.
- Does not offer any advice or insights on what to do next based on the results.
- Not designed with elderly or non-medical users in mind, which limits its accessibility for those groups.

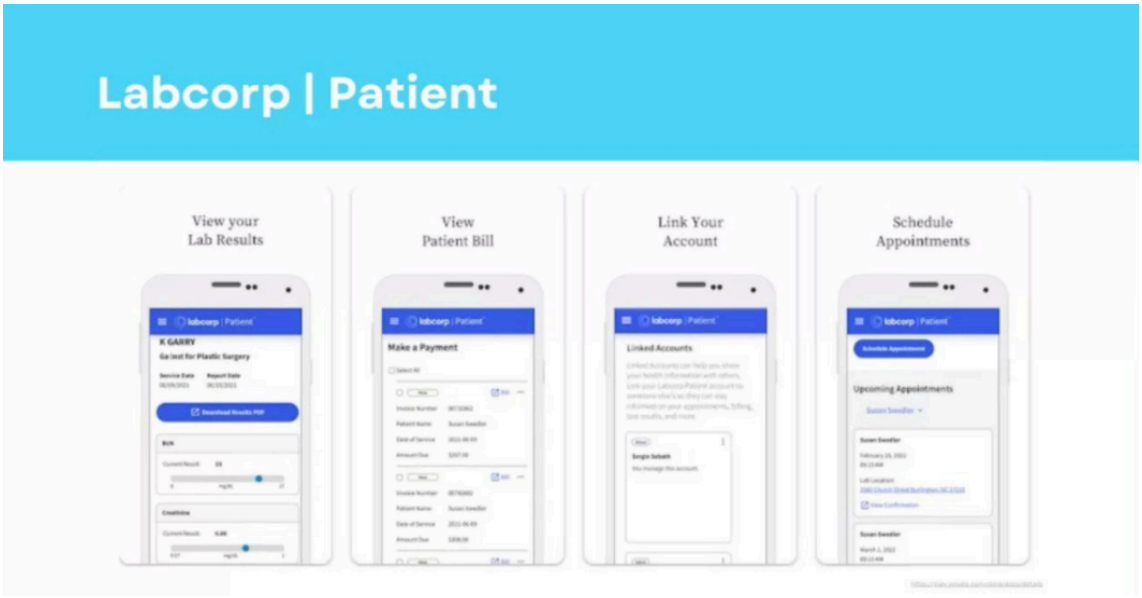


Figure 5: LabCorp | Patient [13] .

2.4 Contribution

The proposed SmartLab app makes a significant contribution to healthcare and technology by offering an innovative solution to a real-world challenge – helping people understand their medical reports more easily. AI plays a key role in this process by using natural language processing to simplify complex medical language, extract medical test results in a clear and simplified manner, and provide advice based on the analysis results. This enables users to make informed health decisions, especially for elderly users and non-native speakers through its multilingual support. By making healthcare information more accessible and understandable, SmartLab aims to reduce the anxiety often associated with medical reports, and encourage users to take proactive control of their health. Its user-friendly design, fast performance, and personalized health advice make it an effective tool for bridging the gap between medical professionals and patients, especially those from underrepresented groups.

Comparison of existing healthcare applications

Feature	MyChart	WebMD	Ada Health	LabCrop Patient	SmartLab (Proposed)
Simplified Medical Reports	No	No	No	No	Yes
Personalized Health Advice	No	General Advice	Yes(Symptoms)	No	Yes(Reports-Based)
Symptom Checker	No	Yes	Yes	No	No
AI Analysis of Medical Reports	No	No	No	No	Yes
Multilingual Support	No	No	No	No	Yes
Upload Medical Reports	Yes	No	No	Yes	Yes

Table 1: comparison of existing healthcare applications.

Chapter Three

Problem Analysis

3.1 Introduction

Problem analysis is a key step in understanding the challenges that need to be addressed in a project **[25]**. It involves identifying the difficulties users face and analyzing the system's requirements to ensure it solves those problems effectively. In this chapter, we will examine the issues people encounter when trying to understand their medical test reports, especially the complexity of medical terms and data. Misunderstanding medical reports can lead to serious consequences, such as mismanaging a health condition, delaying necessary treatments, or ignoring warning signs of potential health issues. For example, if a patient misinterprets abnormal test results or doesn't understand the significance of certain medical terms, they might fail to seek timely medical intervention or follow prescribed treatment plans. This can exacerbate their condition and lead to unnecessary health risks. The inability to understand such reports can create confusion and anxiety, leaving patients feeling helpless and disengaged from their own healthcare decisions.

To address these challenges, we will break down the problem into key components and identify the specific functional and non-functional requirements that the proposed AI-driven mobile app must meet. This analysis also covers the user needs and the app's interactions with the user, ensuring it is designed to be simple, clear, and effective, ultimately enabling individuals to take informed actions regarding their health.

3.2 Problem Specification

Research has shown that improving patients' understanding of their medical reports significantly enhances health outcomes and reduces fear and anxiety. For instance, a study by the American College of Radiology found that simplified radiology reports, written in patient-friendly language, increase patient engagement in healthcare decisions and promote adherence to treatments. When patients understand medical terminology, they communicate more effectively with healthcare providers and become more proactive in managing their health [26].

However, many medical reports contain complex terms or symbols that are difficult for the average person to understand. Terms like "hematocrit levels," "glucose intolerance," or symbols indicating abnormal values often lead to confusion and misinterpretation. This lack of clarity can prevent patients from fully understanding the significance of their health conditions and recommended treatments.

SmartLab addresses this challenge by leveraging AI to simplify medical reports. Users can upload their medical reports, and the app generates easy-to-understand explanations. For example, instead of using the term "hypertension," it simplifies it to "high blood pressure," along with personalized health advice. This approach not only reduces anxiety but also empowers users, helping them take an active role in their health management and make informed decisions about their well-being.

3.3 System Analysis

3.3.1 Requirement Analysis

The SmartLab app has both functional and non-functional requirements to provide a smooth, intuitive experience.

Functional Requirements:

1. **User Registration and Authentication via Email:**Users must be able to create an account and log in using a valid email address, which serves as the primary access point to all application functionalities. Additionally, the system shall support guest access, allowing users to explore the application and utilize limited features without the need for account creation.
2. **OCR and AI-based Analysis:** Use OCR.space to extract text from uploaded reports and then analyze it using Gemini AI to provide simplified explanations of lab results.
3. **Data input :** Allow users to upload medical test reports as images(JPEG,JPG,PNG,PDF)via the device gallery or camera using the file_picker plugin.
4. **Health Tips:** Generate personalized health improvement suggestions based on test metrics (e.g., high/low values).
5. **Email Report Summary:** Enable users to send a summarized interpretation of the report to their registered email address.
6. **Multilingual Support:** Support Arabic and English using a toggle with persistent language settings via SharedPreferences.

Non-functional Requirements:

1. **Usability:** Intuitive design with clear instructions and large text for elderly users.
2. **Performance:** Real-time analysis with minimal waiting.
3. **Scalability:** Handle increased demand as the user base grows.

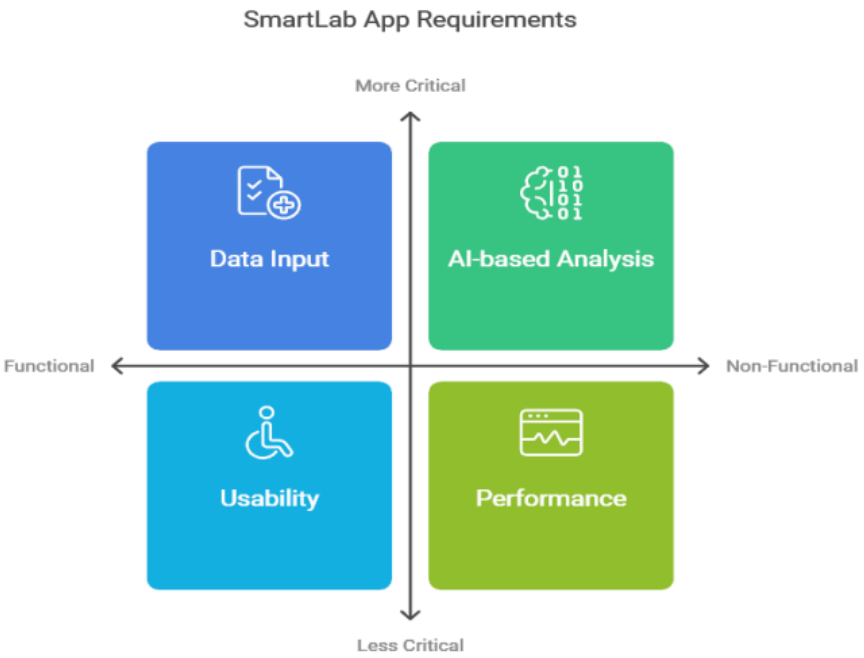


Figure 6: SmartLab App Requirements [16] .

3.4 Implementation and Evaluation Plan

3.4.1 Technical Tools and Programming Languages

SmartLab will be developed using **Flutter**, which is a framework based on the **Dart** programming language developed by **Google**. Dart is primarily used for building Flutter applications that run across multiple platforms such as **iOS**, **Android**, and web or desktop applications. Flutter is known for its fast

development and consistent user experience, making it the ideal choice for building comprehensive and integrated applications [27] .

Figure 7: Flutter [18]

The *SmartLab* application utilizes the Gemini API to analyze medical data by leveraging advanced artificial intelligence techniques. Gemini's powerful natural language processing capabilities have enabled effective interpretation of complex medical information. Its compatibility with multiple programming environments, such as Python and web-based platforms, has facilitated the flexible and efficient implementation of the application. This service has contributed to generating simplified and accurate explanations of medical data, enhancing users' understanding of medical content [28].

The OCR.space API will scan the text on paper and convert it into a digital format that can be processed by AI. In a Flutter application, this process is supported by several essential libraries. The http package is used to send images to the OCR.space server via HTTP POST requests. The image_picker package allows users to capture or select images from their device. To handle file paths and access temporary directories, path_provider and path are employed. The dart:convert library is also crucial for encoding images in Base64 format and parsing the JSON response received from the API. These libraries work together seamlessly to enable OCR functionality and integration with AI processing in a mobile environment[29].

Figure 8:OCR.space [20]

To ensure a seamless user experience that supports multiple languages, the **SharedPreferences** library will be used in SmartLab. This library facilitates the storage of simple key-value pairs locally on the device, making it ideal for saving user preferences such as selected language (e.g., Arabic or English). By storing the language setting persistently, SharedPreferences ensures that the app maintains the user's language choice across sessions. This contributes to a consistent and personalized user interface experience, including proper support for right-to-left (RTL) layout when Arabic is selected. Additionally, SharedPreferences enables smooth toggling between languages without requiring complex state management or server-side handling, enhancing the overall usability and accessibility of the application [30].

Role of OCR.space in OCR

• **OCR.space**: is a cloud-based OCR API that provides accurate text extraction from images and PDF documents. It supports multiple languages and does not require complex on-device configuration, making it a convenient solution for mobile applications like SmartLab. The API allows the SmartLab app to send medical test result images to the cloud for processing and receive the extracted text, which is then used for further analysis. Its high accuracy and support for various image formats make it a suitable tool for interpreting medical documents within the app. **[31]**.

Importance of OCR in SmartLab

The OCR technology in SmartLab plays a crucial role in enabling the app to handle written medical reports and convert them into digital text for further analysis.
SmartLab will be developed using Flutter, which is a framework based on This helps speed up the process of analyzing complex medical data and providing accurate insights to users, which is vital in the healthcare field, where quick and precise information access is essential for both patients and medical staff **[32]**.

Development Phases of the App

Ultimately, the app will be improved over multiple phases. The first phase focuses on implementing core features such as uploading medical reports and analyzing them using AI. The second phase will involve adding advanced features like multilingual support and enhancements based on user feedback.

CHAPTER FOUR

SYSTEM DESIGN

4.1 Introduction

The system design process is all about turning requirements into a real, working solution. This chapter describes the different system components, data management strategies, and uses diagrams to show how everything interacts and flows together. The purpose of this chapter is to give a clear picture of how the system is structured and how it functions, helping readers understand the decisions made to meet user needs. By focusing on both the architecture and interfaces, we ensure that the system design helps make medical report analysis simpler and more user-friendly with the help of AI.

4.2 System Design Specification

4.2.1 User Interface (UI)

The SmartLab application offers a series of user interfaces designed to enhance user experience and simplify the interaction process. Below is a detailed list and description of each user interface:

1. **Home Interface** : The home screen is the starting point for users. It provides quick access to the key features of the app, such as uploading medical reports, viewing recent analyses, and tracking health trends. The design is minimalistic, with large buttons for easy access, making it user-friendly for people of all ages.

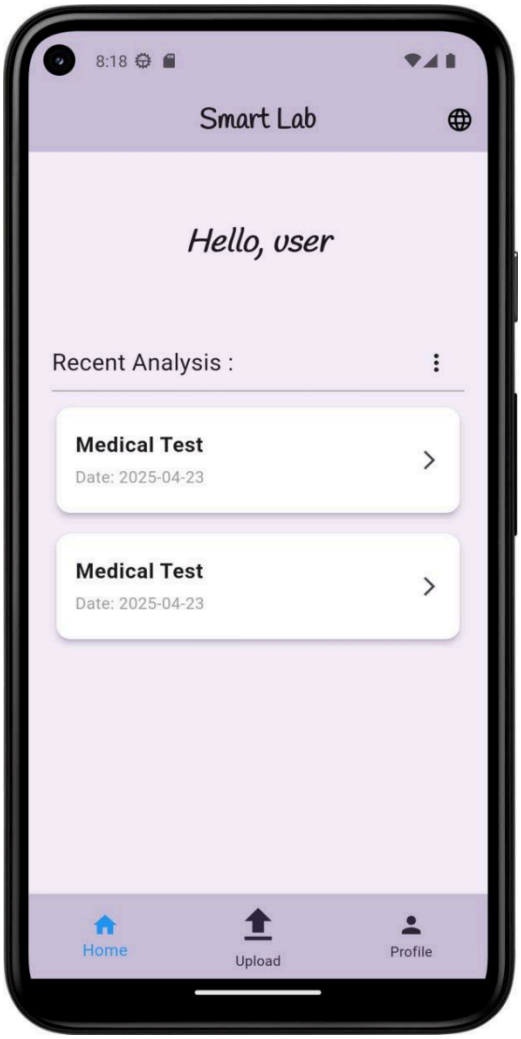


Figure 9 : home screen interface.

2. **Login Interface:** The login screen serves as the gateway for users to access the SmartLab application. It features two primary input fields: one for the username and another for the password, enabling registered users to securely log into their accounts. For new users, the interface includes a clear and accessible option to create a new account, guiding them through a simple registration process. Additionally, the screen offers a convenient "Continue as Guest" feature, allowing users to explore the app's basic functionalities without the need for registration. This flexible design ensures that both returning users and first-time visitors can quickly access and benefit from the app's features**.**

Figure 10: Login interface.

3. **Registration Interface** : The registration screen is designed to facilitate a smooth and user-friendly account creation process for new users. It includes multiple input fields to collect essential user information, such as Full Name, Email, Username, Password, and Age. Each field is clearly labeled to ensure ease of use and reduce user input errors.

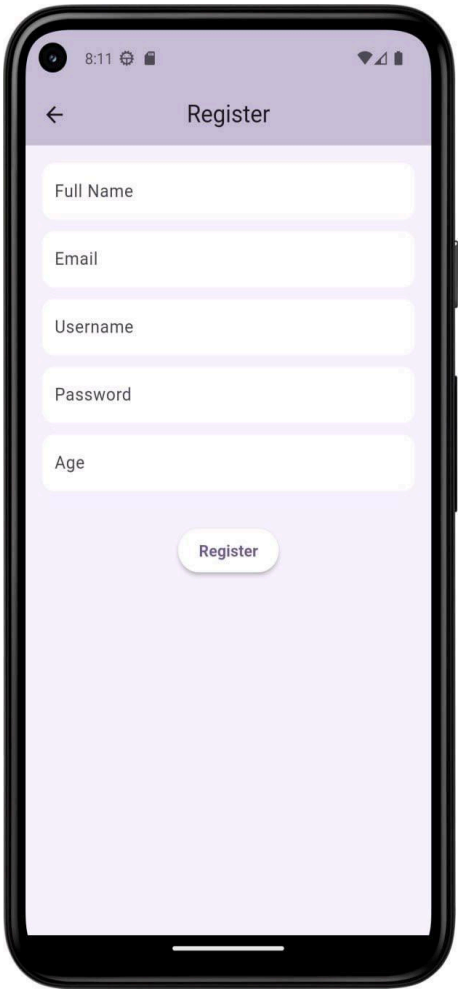


Figure 11 : Register interface.

4. **Report Upload Interface**: This interface allows users to upload their medical reports either by taking a photo or by selecting an existing image file from their device. The upload screen includes clear instructions on how to properly capture the medical report for optimal OCR (Optical Character Recognition) results. The interface is streamlined to make the uploading process as simple as possible.

Figure 1 : **report upload interface.**

5. **Analysis Results Interface**: Once a medical report is processed, users can view the analysis results through this interface. The screen displays a simplified explanation of the report data, including key health metrics and any identified issues. The results are presented in a user-friendly format, using plain language and visual aids (e.g., icons or charts) to help users easily understand the information.

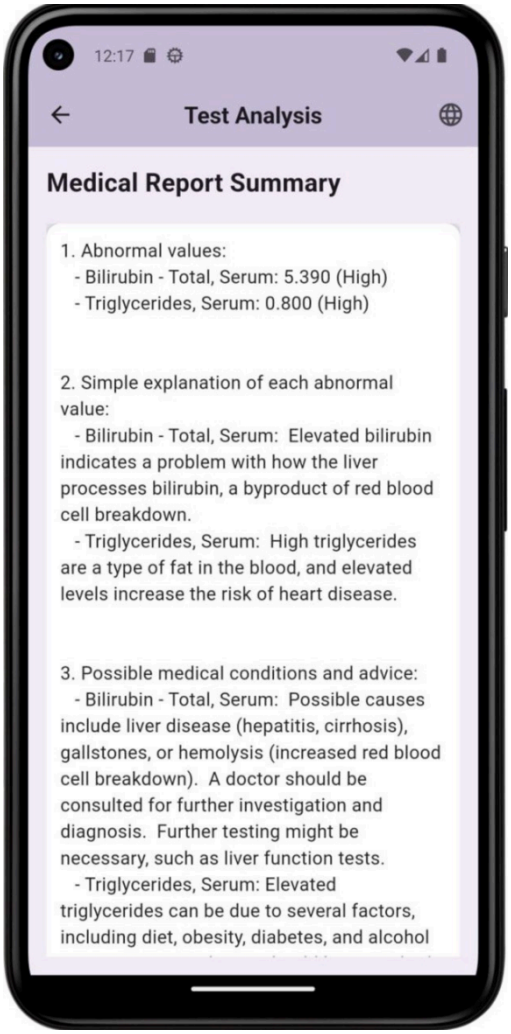


Figure 13 : **Test Analysis interface**

5. **Profile interface** : The Profile interface displays the user's personal information, including full name, age, email address, and password. It provides a simple and secure way for users to view and manage their account details. If needed, users can update any of their information directly through this screen, ensuring their profile remains accurate and up to date.

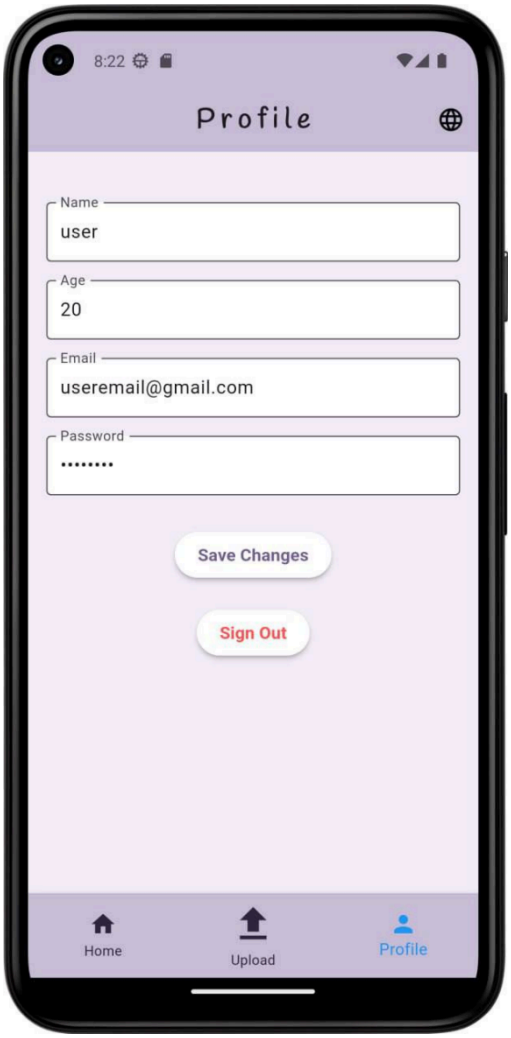


Figure 14: Profile interface.

6. **Email Result:** After the medical report is analyzed, SmartLab provides users with the option to send the results directly to their registered email. This feature ensures that users can easily access, review, and save their health analysis outside the app for future reference or to share with healthcare professionals. The email templates are designed to support both English and Arabic languages, ensuring that users receive the analysis in their preferred language for maximum clarity and accessibility.

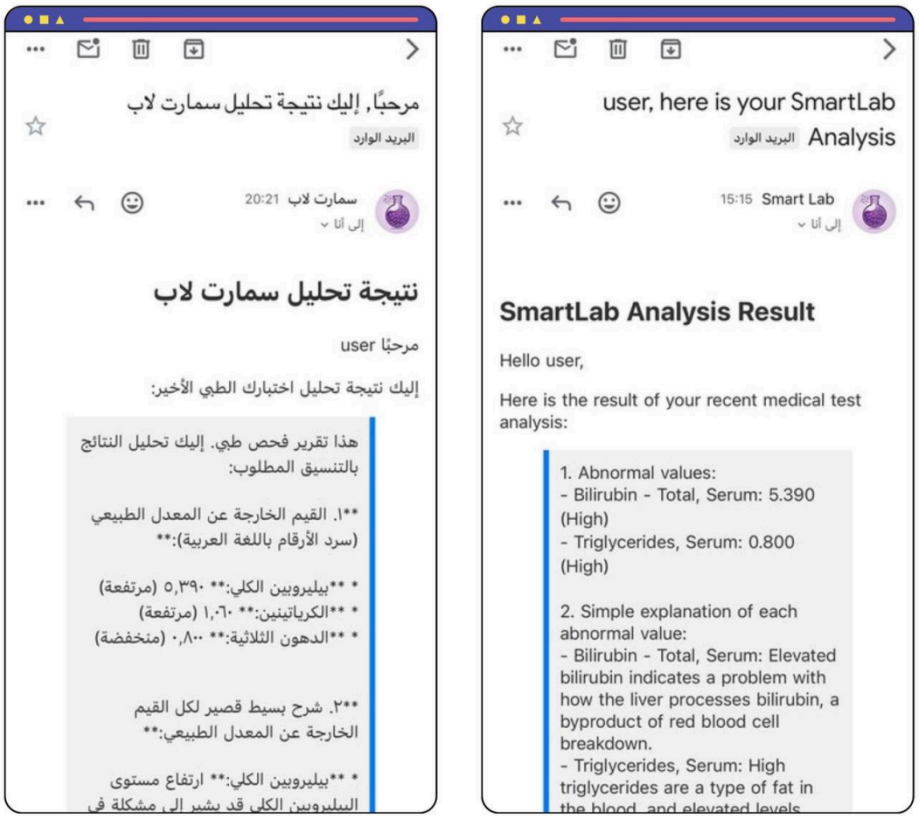


Figure 15: Test analysis via Email

4.2.2 Backend Architecture

The backend architecture of the SmartLab app is designed to support AI-powered medical report analysis, local data persistence, and user-centric interactions while maintaining performance and simplicity. SmartLab currently employs a local SQLite database to store user data and analyzed medical reports directly on the device, ensuring both privacy and offline accessibility. The use of SQLite is ideal for lightweight mobile applications and helps maintain a fast, responsive user experience.

Key architectural components include:

Authentication Module (Auth): Handles user registration and login functionalities, storing credentials locally in the SQLite database. It ensures that users have secure and persistent access to their accounts.

Report Management Module: Manages the storage of uploaded medical reports and their corresponding AI-generated analyses. Each record is saved locally and can be retrieved later for review, enabling users to track their health over time.

AI Integration Service: Connects with cloud-based services such as OCR.space for text extraction and Gemini AI for interpreting medical results. After a report file is uploaded, it is sent to the OCR service, and the extracted text is analyzed by Gemini to generate user-friendly explanations and health tips.

SharedPreferences Layer: While not part of the database, this layer is responsible for storing non-sensitive user preferences like language settings to ensure a personalized experience across sessions.

At present, the backend logic is locally embedded within the Flutter application, which means there is no external API Gateway or server infrastructure. However, the design leaves room for future scalability where cloud storage and remote user authentication could be introduced to support multi-device sync and cross-platform functionality.

4.2.3 AI Integration

AI integration lies at the core of SmartLab, enabling the transformation of complex medical test results into clear, user-friendly summaries. This AI-driven capability is made possible through the integration of both OCR and large language model (LLM) services, which work together to extract and interpret medical data for general users.

The AI services in SmartLab are divided into two primary components:

OCR Service (OCR.space API): After a user uploads a medical test report file, the OCR.space API is used to extract structured text from the image. This cloud-based OCR engine offers high accuracy in recognizing medical terminology and supports various formats, making it a reliable solution for preprocessing report data.

Gemini AI (Text Analysis Service): Once the text is extracted, it is forwarded to the Gemini AI API for advanced interpretation. This LLM-powered service analyzes the medical content and simplifies it into digestible insights, highlighting abnormalities (e.g., high or low test values) and providing personalized health advice.

These services are orchestrated by the business logic layer within the SmartLab Flutter app, ensuring a seamless flow from image upload to final analysis. The entire process runs asynchronously to maintain app responsiveness and user satisfaction. This local-first architecture—with remote AI augmentation—enables SmartLab to deliver powerful features without compromising on performance or usability.

4.3 Design Architecture

The system architecture illustrated in figure **16** describes the conceptual model that defines the structure, behavior, and different components of the system. The SmartLab application uses a client-server architecture that comprises a front-end and a back-end.

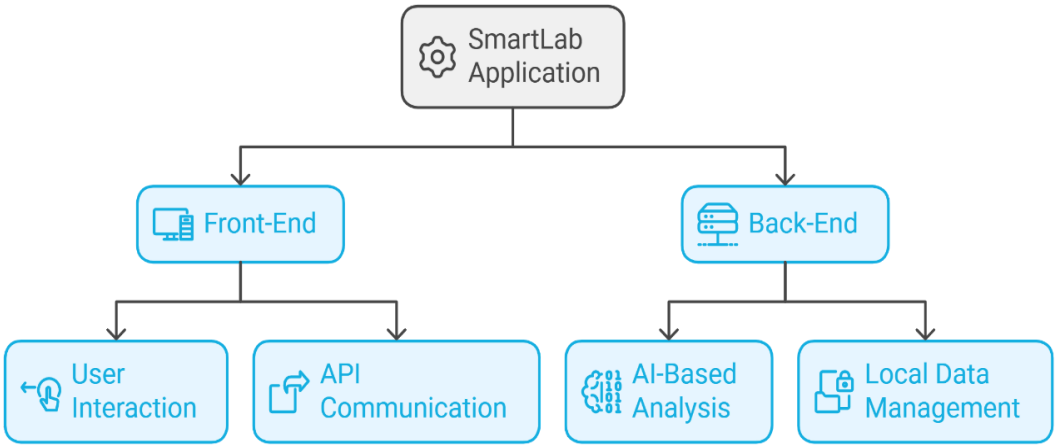


Figure 16: Design Architecture of SmartLab app.

- **Front-End:** The user interface of the application, built using Flutter, is responsible for interacting with users and collecting their input (such as uploading medical reports). The front-end uses APIs to communicate with the back-end.

- **Back-End:** The back-end server processes user requests and performs AI-based analysis using Gemini API. The system now integrates database functionality using SQLite. This includes a user database that manages user registration, login, and profile details, and a report database that stores analyzed medical reports. This ensures that user data and their report history are securely stored and can be retrieved at any time. This addition enhances data management and supports future scalability.

- **Data Management:** At this stage, we are not using a database server yet. Instead, user data and medical reports are handled locally within the application, with limited storage functionality. In future phases, we plan to introduce a full database solution to securely store user information and medical reports, ensuring scalability and robust security for sensitive health information.

4.4 System UML Diagrams

4.4.1 Class diagram

A class diagram is a type of static structure diagram in UML (Unified Modeling Language) that describes the structure of a system by showing its classes, their attributes, methods, and the relationships between the classes. It is commonly used in software development to visually represent the object-oriented design of a system. Class diagrams include elements such as associations, generalizations, and dependencies, helping developers and stakeholders understand the system's architecture and its components.[33]

The class diagram for the SmartLab application provides an overview of the system's structure by detailing its classes, attributes, and relationships. The primary classes are User, MedicalReportFile, MedicalReport, AnalysisResult, and HealthAdvice. The User class represents general attributes and methods common to all users.

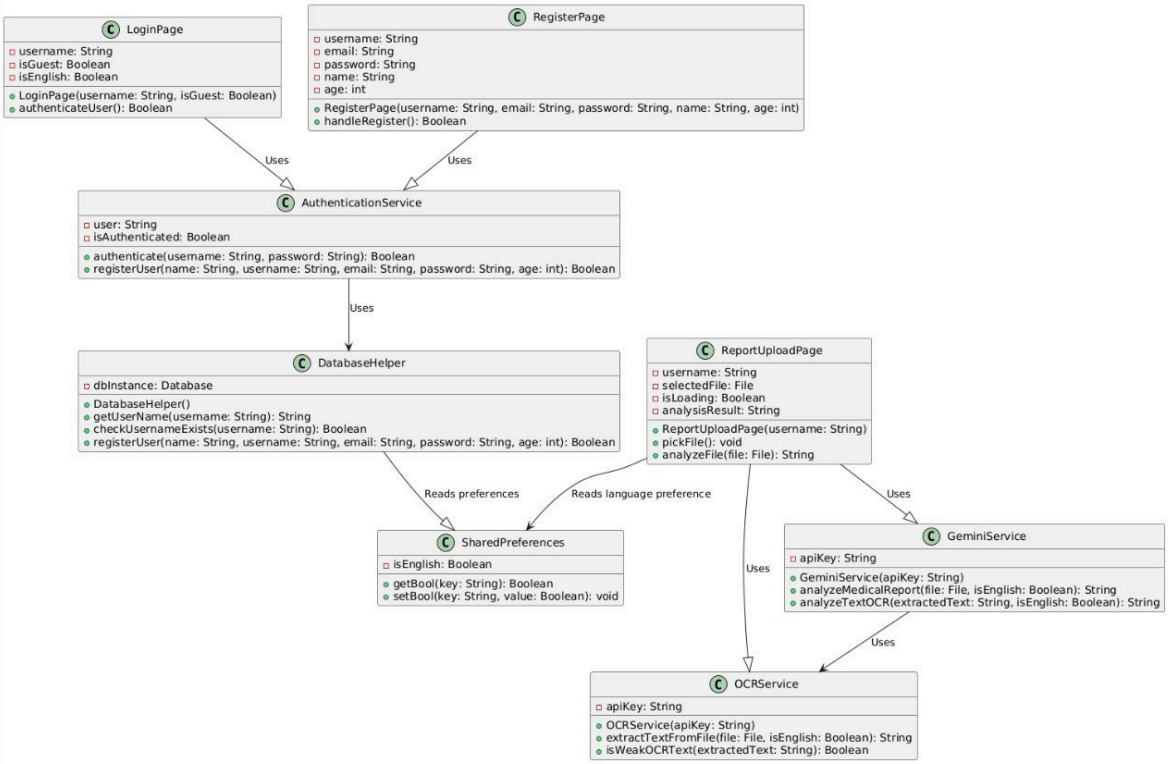


Figure 17 : Class Diagram.

The MedicalReportFile class handles report uploads, while MedicalReport manages the analysis workflow. The AnalysisResult class captures the analysis output, and HealthAdvice is used to generate personalized health tips. Relationships between these classes demonstrate how data flows through the system—for example, how a Patient uploads a MedicalReportFile, which then generates an AnalysisResult that in turn produces HealthAdvice.

4.4.2 Sequence Diagrams

A sequence diagram is a type of interaction diagram in UML that shows the order of messages exchanged between objects over time to accomplish a specific task. It includes lifelines representing objects and horizontal arrows for messages or method calls. Sequence diagrams are useful for visualizing process flows and analyzing system behavior during design.[34]

The sequence diagrams illustrate the interactions between users and the system components over time for various use cases, they can upload medical reports, view health trends, and analyze their reports. This flow involves the SmartLab System and AI Service, data analyzed and presented to the patient in a logical, step-by-step manner.

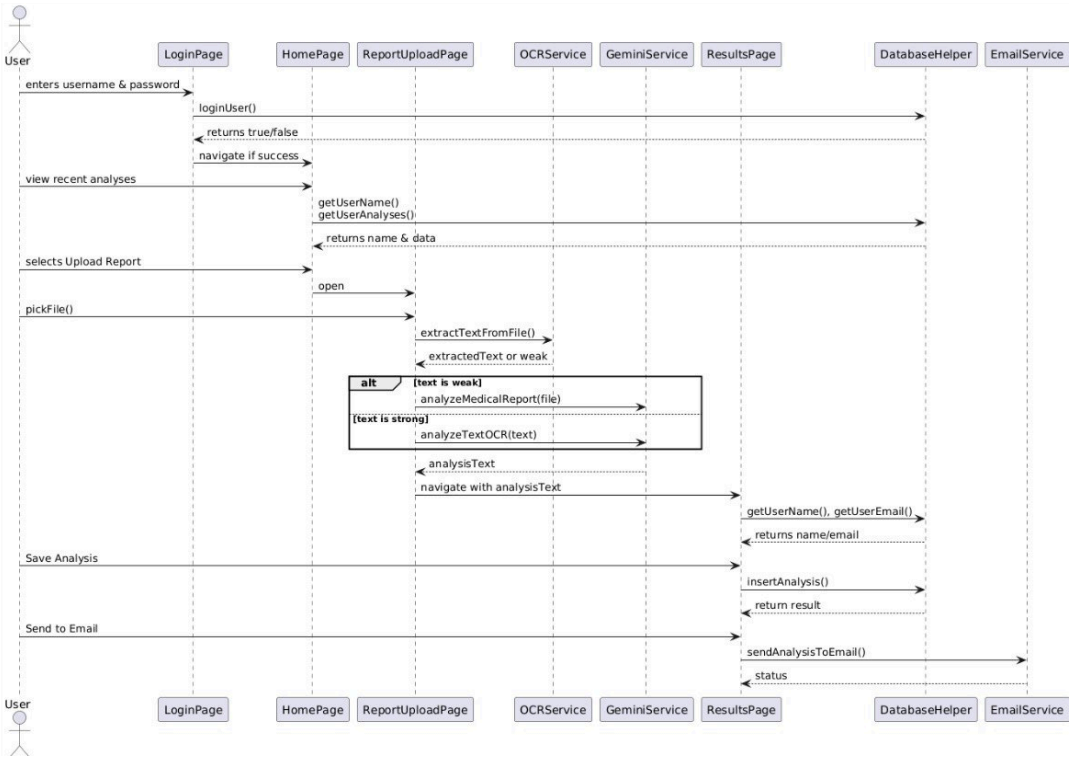


Figure 18 : Sequence Diagram.

4.5 pseudocode

4.5.1 Medical Report Processing

The medical report processing functionality includes the steps for uploading and analyzing reports. Below is the pseudocode outlining these steps:

```
pseudocode FilePickingAndAnalysis

 Input: None (Relies on user interaction to select a file)

 Output: AnalysisText

 Step 1: Check if user is a guest

 Step 2: If user is a guest, prompt for login and exit

 Step 3: Display file picker for user to select a file

 Step 4: If no file is selected, exit

 Step 5: If file is selected, get the file path

 Step 6: Check if the selected file is a PDF

 Step 7: Start loading process (show loading screen)

 Step 8: Try to analyze the selected file

    - If the file is a PDF, call the appropriate method to analyze the file

    - If the file is not a PDF, process the file accordingly (e.g., convert to text for analysis)

 Step 9: Navigate to the results page with the analysis result

 Step 10: Catch any errors that may occur during analysis

    - Handle analysis failure gracefully (e.g., show error message)

 Step 11: Stop the loading process (hide loading screen)

End Algorithm
```

CHAPTER FIVE

SYSTEM IMPLEMENTATION

5.1 Introduction

This chapter presents the development journey of the SmartLab application, highlighting the transformation from initial design concepts to a fully functional system. It also discusses the technologies utilized, the system's architecture, and how the application operates in a real-world environment.

5.2 Implementation and Evaluation Plan

The implementation and evaluation phase of the *SmartLab* application represents a critical stage in the project lifecycle. This phase required careful planning to ensure the successful delivery of a functional, scalable, and user-friendly system. The following sections outline the implementation strategy, divided into two main phases, and describe the evaluation approach used to validate the system.

The project adopts an **Agile Development Framework**, enabling incremental feature delivery, continuous testing, and iterative improvements based on early feedback. This approach was chosen to accommodate evolving technical challenges and prioritize essential functionalities.

The implementation proceeded in two sequential phases:

Phase 1: Core Feature Development
The first phase focused on establishing the fundamental capabilities of the SmartLab system. Key activities during this phase included:

- Developing the functionality for users to upload medical reports, supporting both image and PDF formats.

- Integrating OCR (Optical Character Recognition) technology to accurately extract text from the uploaded reports.

- Implementing an AI-driven analysis module to interpret the extracted medical information and present it in a simplified and user-friendly format.

This phase laid the foundational infrastructure necessary for the system's primary purpose: facilitating the understanding of complex medical test results.

Phase 2: Backend Enhancement and AI Integration

The second phase introduced critical backend improvements to enhance system performance, reliability, and feature richness. Major activities included:

- Incorporating a local database solution using **SQLite** (managed through DB Browser for SQLite) to store user accounts and historical medical reports securely on the device.

- Integrating the **Gemini API** to perform more advanced AI-based analysis, delivering deeper insights and personalized medical report explanations.

- Extending the system's language capabilities to support both Arabic and English, ensuring accessibility for a wider audience.

These enhancements significantly strengthened the application's ability to offer continuous service even offline and deliver sophisticated, multilingual medical interpretations.

Evaluation Strategy

System evaluation was conducted through iterative testing across all major components. Specifically:

- **OCR Accuracy Testing**: Various types of medical reports were used to assess the effectiveness of text extraction, with adjustments made to optimize recognition accuracy.
- **AI Analysis Validation**: The AI module's output was evaluated for clarity, medical relevance, and language quality in both Arabic and English.
- **Database Performance Assessment**: The SQLite database was tested for data integrity, retrieval speed, and the ability to handle multiple reports per user without loss or corruption.

Through continuous testing and refinement, SmartLab’s functional and non-functional requirements were thoroughly validated, ensuring a robust and user-centered final product.

5.3 Key Development Technologies and APIs

- **Flutter:** Used for cross-platform development on Android and iOS.
- **Dart:** The main programming language used in Flutter.
- **OCR.space API :** For cloud-based OCR with high accuracy and Arabic/English support.
- **Gemini API (Google):** For AI-powered language processing and personalized health advice.
- **DB Browser for SQLite :** Used to store user data and medical reports securely.
-**EmailJS API :** Used to send analysis to user's email, email send based on user's preferred language

These tools were chosen for their compatibility with mobile development, lightweight performance, and support for the required functionalities.

5.4 Code Structure

5.4.1 Overview

The SmartLab mobile application adopts a modular and layered software architecture to promote clean code organization, scalability, and ease of maintenance. Developed using the Flutter framework (Dart), the application integrates AI services and optical character recognition (OCR) to process medical test reports and deliver simplified analyses to end-users. The application structure is divided into functional components, each handling a distinct responsibility such as user interface, business logic, data storage, and third-party API integration.

5.4.2 Project Directory Structure

The core files of the SmartLab application are organized under the lib/ directory. Below is a high-level overview of the project structure:

```
lib/
├── screens/          # Contains all UI screen files
│   ├── login_page.dart
│   ├── register_page.dart
│   ├── home_page.dart
│   ├── report_upload_page.dart
│   ├── results_page.dart
│   └── profile_page.dart
├── services/         # Includes API services
│   ├── gemini_service.dart
│   ├── ocr_service.dart
```

```
│   └── email_service.dart
├── database/           # SQLite database helper
│   └── database_helper.dart
├── widgets/            # Shared UI components
│   ├── base.dart
│   ├── bottom_nav_bar.dart
│   └── top_app_bar.dart
└── main.dart           # Application entry point
```

This structural separation ensures a clear division of functionality, simplifies navigation within the codebase, and supports effective teamwork and feature isolation.

5.4.3 Main Components

a) User Authentication

Files: login_page.dart, register_page.dart

This component allows users to sign in or register an account. Navigation is implemented using Navigator.push, and input validation ensures accurate credential management. Upon successful authentication, users are redirected to the Home Page.

This is a private asynchronous function responsible for handling the user login process within the SmartLab app. It first validates the input form using _formKey. If both the username and password fields are filled correctly, the function proceeds to check user credentials against the local SQLite database via the DatabaseHelper class. Upon successful authentication, it shows a success message and navigates to the HomePage. In case of failure or exception, it displays an appropriate error message using a localized SnackBar based on the

selected language. This function supports bilingual feedback (English and Arabic) and ensures secure and user-friendly login flow.

```dart
void _handleLogin() async {
  if (_formKey.currentState!.validate()) {
    String username = _usernameController.text.trim();
    String password = _passwordController.text;

    if (username.isEmpty || password.isEmpty) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
            content: Text(isEnglish
                ? 'Please fill in all fields.'
                : 'يرجى تعبئة جميع الحقول.')),  // Text, SnackBar
      );
      return;
    }
    try {
      bool userExists =
          await DatabaseHelper.instance.loginUser(username, password);
      if (userExists) {
        ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(
              content: Text(isEnglish
                  ? 'Login successful!'
                  : 'تم تسجيل الدخول بنجاح!')),  // Text, SnackBar
        );
        Navigator.pushReplacement(...);
      } else {
        ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(
              content: Text(isEnglish
                  ? 'Invalid username or password.'
                  : 'اسم المستخدم أو كلمة المرور غير صحيحة.')),  // Text, SnackBar
        );
      }
    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
            content: Text(isEnglish
                ? 'Login failed. Try again later.'
                : 'فشل تسجيل الدخول. حاول مرة أخرى لاحقًا.')),  // Text, SnackBar
      );
    }
  }
```

Figure 19 : _handleLogin() function.

b) Report Upload and OCR Processing

File: report_upload_page.dart

Users upload an image or PDF of their medical report through this interface. The application utilizes the OCR.space API to extract text from the file. A validation step then determines whether the extracted text is sufficient or if the original file should be passed directly to the AI for processing.

To illustrate this process briefly, the _pickFile() function from report_upload_page.dart is shown below:

This function manages the file selection and analysis workflow within the SmartLab app. It checks if the user is logged in, prompts guests to log in, and allows authenticated users to pick a file (image or PDF) from their device. Upon selection, it triggers a loading state, sends the file for medical analysis, and navigates to the results page with the generated report. If an error occurs, it handles it gracefully and stops the loading indicator.

```dart
Future<void> _pickFile() async {
  if (widget.isGuest) {
    await _showLoginPrompt();
    return;
  }

  FilePickerResult? result = await _pickFileFromDevice();
  if (result == null || result.files.single.path == null) {
    return;
  }

  File selected = File(result.files.single.path!);
  bool isPDF = _isPDFFile(selected);

  _startLoading(isPDF);

  try {
    String analysisText = await _analyzeFile(selected, isPDF);
    _navigateToResultsPage(analysisText);
  } catch (e) {
    _handleAnalysisFailure(e);
  } finally {
    _stopLoading();
  }
}
```

Figure 20 : _pichFile() function.

c) AI-Based Analysis

File: gemini_service.dart

The extracted or original input is analyzed using the Gemini API, which generates an interpretation of the lab test results in layman's terms. This AI-enhanced explanation is then presented to the user on the Results Page.

This asynchronous function sends a medical report file (image or PDF) to the Google Gemini API for analysis. It encodes the file in Base64, attaches it to a structured prompt (in English or Arabic based on the user's language preference), and retrieves a clear summary of abnormal test values, explanations, and possible medical advice. The result is formatted for easy understanding by non-specialists, making the medical data more accessible to general users.

```
Future<String> analyzeMedicalReport(File file, bool isEnglish) async {
  String apiUrl =
      "https://generativelanguage.googleapis.com/v1/models/gemini-1.5-flash:generateContent?key=$apiKey";

  List<int> fileBytes = await file.readAsBytes();
  String base64File = base64Encode(fileBytes);

  String fileExtension = file.path
      .split('.')
      .last
      .toLowerCase();
  String mimeType =
  (fileExtension == 'pdf') ? 'application/pdf' : 'image/jpeg';

  String prompt = isEnglish
      ? """...."""
      : """...""";

  var requestBody = {...};

  var response = await http.post(
    Uri.parse(apiUrl),
    headers: {"Content-Type": "application/json"},
    body: jsonEncode(requestBody),
  );

  if (response.statusCode == 200) {
    var jsonResponse = jsonDecode(response.body);
    return jsonResponse["candidates"][0]["content"]["parts"][0]["text"] ??
        "No explanation found.";
  } else {
    return "Error: ${response.body}";
  }
}
```

Figure 21 : analyzeMedicalReport() function.

d) Local Database Storage

File: database_helper.dart

This component manages local data persistence using the SQLite package. Analysis results are stored locally with metadata for future retrieval, enabling longitudinal data tracking and visualization.

```dart
// Initialize database
Future<Database> _initDB() async {
  String path = await getDatabasesPath();
  String dbPath = join(path, 'users.db');
  print("Database Path: $dbPath");

  if (await databaseExists(dbPath)) {
    print("Old database found, deleting it to recreate tables.");
    await deleteDatabase(dbPath);
  }

  return await openDatabase(
    dbPath,
    version: 5, // Database version
    onCreate: _createDB,
    onUpgrade: _upgradeDB,
  );
}
```

Figure 22 : _initDB() function.

e) Email Notification

File: email_service.dart

SmartLab integrates with EmailJS to allow users to send analysis summaries via email. The function is optional and triggered on demand from the Results Page and Home Page.

This static asynchronous function is responsible for sending the user's medical report analysis results via email using the EmailJS API. It accepts user-specific data (name, email, and analysis result) along with the app's language setting to select the appropriate email template (English or Arabic). It sends a POST request with the formatted data and provides real-time feedback to the user via a SnackBar indicating whether the email was sent successfully or if an error occurred. This function helps integrate a personalized and multilingual communication layer into the SmartLab app.

```dart
class EmailService {
  static Future<bool> sendAnalysisToEmail(...) async {
    const serviceId = 'service_nn17u95';
    const templateIdEnglish = 'template_rdni558';
    const templateIdArabic = 'template_qw27dfe';
    const publicKey = 'aLAIXXjDD9n873wZb';
    final urlString = 'https://api.emailjs.com/api/v1.0/email/send';
    final url = Uri.parse(Uri.encodeFull(urlString));

    // Use the provided client or default to http.Client()
    client ??= http.Client();

    print("Sending POST request to: $url");

    try {
      final response = await client.post(
        url,
        headers: {...},
        body: json.encode({...}),
      );

      final success = response.statusCode == 200;

      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(...),  // SnackBar
      );

      return success;
    } catch (e) {
      print("Error while sending email: $e");

      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text(
            isEnglish
                ? "An error occurred while sending the email."
                : "حدث خطأ أثناء إرسال البريد.",
          ),  // Text
        ),  // SnackBar
      );

      return false;
```

Figure 23 : *sendAnalysisToEmail()* function.

f) Reusable UI Components

Files: base.dart, bottom_nav_bar.dart, top_app_bar.dart

These files contain customizable UI widgets for consistent design and layout across the application. The bottom navigation bar and top app bar are reused throughout various screens to enhance the user experience.

(*Note: For the full list of related functions and detailed implementation, please refer to the Appendix.*)

5.4.4 Component Interaction Flow

1. The user authenticates via the login or registration screen.

2. After logging in, the user uploads a medical report file.

3. The file is processed through the OCR service to extract text.

4. Based on the extracted content, the application either sends the text or the file to the Gemini AI service.

5. The AI response is shown on the Results Page.

6. The user may optionally send the analysis to their email or just save it to the Homepage on the recent analysis section.

7. Language preference is maintained across the application using SharedPreferences.

5.5 Discussion

The system implementation effectively translated the design into a functional, user-friendly application. Challenges such as Arabic text OCR and AI limitations were addressed through appropriate tool selection. The integration of Gemini provided reliable, contextual analysis for diverse medical content. Overall, the implementation phase validated the technical feasibility of the proposed solution.

CHAPTER SIX

SYSTEM TESTING AND RESULT DISCUSSION

6.1 Introduction

This chapter presents the testing approach used to evaluate SmartLab's functionality, accuracy, and usability. It includes the test cases applied, unit tests performed on core functionalities, the results obtained, and the implications of those results for system performance.

6.2 Testing Description and Test Cases

Unit Testing :is a software testing technique that involves testing individual units or components of a software application in isolation. These units, typically functions or methods, represent the smallest pieces of code. The primary goal of unit testing is to ensure that each unit performs as expected [35]. Targeted unit tests were conducted to verify the correct functionality of key services :

A. analyzeMedicalReport() in the Gemini service file was tested to verify its ability to correctly interpret and simplify extracted medical text, and return structured health advice.

```
PS C:\Users      \StudioProjects\SmartLab> flutter pub run build_runner build
Deprecated. Use `dart run` instead.
[INFO] Generating build script completed, took 747ms
[INFO] Precompiling build script... completed, took 12.3s
[INFO] Building new asset graph completed, took 3.1s
[INFO] Checking for unexpected pre-existing outputs. completed, took 2ms
[WARNING] No actions completed for 23.7s, waiting on:
  - mockito:mockBuilder on test/widget_test.dart

[INFO] Running build completed, took 34.5s
[INFO] Caching finalized dependency graph completed, took 154ms
[INFO] Succeeded after 34.8s with 83 outputs (166 actions)
PS C:\Users\eellh\StudioProjects\SmartLab> flutter test test/gemini_service_test.dart
00:06 +2: All tests passed!
```

Figure 24 : Unit test results on analyzeMedicalReport() function.

B. extractTextFromFile() in the OCR service file was tested for its ability to process various image files and reliably extract textual content under different input conditions.

```
PS C:\Users\eellh\StudioProjects\SmartLab> flutter test test/ocr_service_test.dart
00:06 +1: All tests passed!
```

Figure 25 : Unit test results on extractTextFromFile() function.

C. sendAnalysisToEmail() in the Email service file was tested to confirm the accurate formatting and successful dispatch of analysis results to user email addresses.

```
PS C:\Users\eellh\StudioProjects\SmartLab> flutter test test/email_service_test.dart
00:05 +0: EmailService sendAnalysisToEmail returns true on successful request
Sending POST request to: https://api.emailjs.com/api/v1.0/email/send
00:05 +1: All tests passed!
```

Figure 26 : Unit test results on sendAnalysisToEmail() function.

OCR Accuracy Test:

Different medical reports—typed and handwritten, in Arabic and English—were uploaded to assess the accuracy of text extraction.

AI Output Test:

Extracted text was analyzed using the Gemini API to evaluate the medical explanation quality and the relevance of health recommendations.

User Interaction Test:

Focused on the user experience in terms of navigation, language switching, screen clarity, and responsiveness.

Data Storage Test:

Ensured that user-uploaded reports were correctly stored and retrieved from the local SQLite database.

6.3 Results and Discussions

SmartLab was assessed across its critical functional components and yielded the following outcomes:

Unit Testing Outcomes

Unit tests on core service functions confirmed the following:

- **analyzeMedicalReport()** functioned reliably, returning coherent and accurate interpretations of varied medical report texts. Edge cases involving rare tests were handled gracefully, though some responses leaned toward general advice. To validate this behavior, unit tests were performed using a mock service, test code is provided below.

```
@GenerateMocks([GeminiService])
void main() {
 late MockGeminiService mockGeminiService;

 setUp(() {
  mockGeminiService = MockGeminiService();
 });

 group('GeminiService Tests', () {
  test('analyzeMedicalReport returns expected result', () async {
   final fakeFile = File('test/assets/fake_report.jpg');
   const expectedResult = 'Analysis of medical report in English.';
   const isEnglish = true;

   when(mockGeminiService.analyzeMedicalReport(fakeFile, isEnglish))
      .thenAnswer((_) async => expectedResult);

   final result =
      await mockGeminiService.analyzeMedicalReport(fakeFile, isEnglish);

   expect(result, expectedResult);
   verify(mockGeminiService.analyzeMedicalReport(fakeFile, isEnglish)).called(1);
  });
```

- **extractTextFromFile()** provided consistent extraction performance for typed documents, with reduced accuracy for handwritten or low-resolution inputs—pointing to opportunities for enhanced image preprocessing. To demonstrate this functionality, the relevant service implementation code is presented below.

```
Future<String> extractTextFromFile(File file, bool isEnglish) async {
  final multipartFile = await http.MultipartFile.fromPath('file', file.path);
  return extractTextFromMultipartFile(multipartFile, isEnglish);
}

Future<String> extractTextFromMultipartFile(http.MultipartFile file, bool isEnglish) async {
  final uri = Uri.parse("https://api.ocr.space/parse/image");
  final request = http.MultipartRequest('POST', uri)
    ..fields['apikey'] = apiKey
    ..fields['language'] = isEnglish ? 'eng' : 'ara'
    ..files.add(file);

  final response = await request.send();
  final responseData = await response.stream.bytesToString();

  if (response.statusCode == 200) {
    final jsonResponse = jsonDecode(responseData);
    return jsonResponse['ParsedResults'][0]['ParsedText'];
  } else {
    throw Exception("OCR failed: $responseData");
  }
}
```

- sendAnalysisToEmail() successfully composed and sent analysis reports in all test cases, with proper formatting and delivery confirmation, validating its integration within the user workflow. To demonstrate this functionality, the following code snippet is provided:

```
static Future<bool> sendAnalysisToEmail({
  required BuildContext context,
  required String userName,
  required String userEmail,
  required String analysisResult,
  required bool isEnglish,
  http.Client? client,
}) async {
  const serviceId = 'service_nn17u95';
  const templateIdEnglish = 'template_rdni558';
  const templateIdArabic = 'template_qw27dfe';
  const publicKey = 'aLAIXXjDD9n873wZb';

  final url = Uri.parse('https://api.emailjs.com/api/v1.0/email/send');
  client ??= http.Client();

  final response = await client.post(
    url,
    headers: {
      'origin': 'http://localhost',
      'Content-Type': 'application/json',
    },
    body: json.encode({
      'service_id': serviceId,
      'template_id': isEnglish ? templateIdEnglish : templateIdArabic,
      'user_id': publicKey,
      'template_params': {
        'user_name': userName,
        'analysis_result': analysisResult,
        'to_email': userEmail,
      },
    }),
  );

  return response.statusCode == 200;
}
```

OCR Performance

The OCR.space API effectively extracted text from printed medical documents. Accuracy diminished with handwritten or poorly scanned inputs, suggesting the need for future image enhancement techniques.

AI Analysis

The Gemini API returned simplified and user-friendly explanations for most standard tests, consistently adapting the analysis to the user's preferred language. Whether the user selected Arabic or English, Gemini reliably generated coherent and accessible reports accordingly. Its guidance was affirmed by medical reviewers as being both accessible and contextually appropriate, despite some generalizations in less common cases. To illustrate this functionality, examples of the AI-generated analysis in both Arabic and English are provided below.

Figure 27 : AI-generated analysis in both Arabic and English.

User Experience

Usability tests showed participants could complete tasks without confusion. The bilingual interface (English/Arabic) and seamless navigation were especially well-received, accommodating diverse user preferences.

Data Management

The SQLite-based local database consistently stored and retrieved historical data with no performance issues. Users could view and compare past results efficiently, supporting long-term health tracking.

CHAPTER SEVEN

CONCLUSION AND FUTURE WORK

7.1 Conclusion

SmartLab successfully meets its goal of making medical lab reports understandable and actionable for general users. The integration of OCR, AI, and email services, supported by robust unit testing, ensures a dependable experience. Identified areas for improvement—such as enhancing OCR for poor-quality images and fine-tuning AI outputs—will shape the roadmap for future development.

7.2 Future Work

To enhance the functionality and accessibility of the SmartLab application, several improvements and new features are planned for future development:

- ICD-10 Code Integration: Incorporating the International Classification of Diseases (ICD-10) codes will allow the app to link medical terms with standardized diagnoses. This feature will provide users with more precise and medically recognized information, improving the reliability of the analysis results.
- Web-Based Version of SmartLab: In addition to the mobile application, a web-based version of SmartLab will be developed. This will allow users to access the platform through desktop browsers, making it more flexible and accessible across different devices.
- Support for Additional Languages: To serve a more diverse user base, future versions of SmartLab will include support for additional languages beyond Arabic and English. This will enhance usability for non-Arabic and non-English speakers, making the app globally inclusive.

- Graphical Representation of Health Data: Future updates will include graphical visualizations such as charts and graphs to represent health trends and analysis results. This will help users better understand their health data over time in a more intuitive and engaging way.

7.3 FYP Closing Remarks

Conducting this Final Year Project, SmartLab, was a valuable and enriching experience that significantly deepened our technical expertise and problem-solving capabilities. From exploring the landscape of healthcare applications to implementing AI-powered features for medical test interpretation, we gained hands-on experience in developing a real-world, user-centric mobile solution. This project strengthened our understanding of Flutter development, OCR integration, AI analysis with Gemini, and secure local data storage. It also taught us the importance of collaborative teamwork, adaptability, and designing accessible solutions that prioritize user clarity and well-being.

References

[1] N. I. o. H. (. N. i. H. Team, "Artificial Intelligence and Your Health".

[2] C. H. O. G. Dr. Karen DeSalvo, "How AI is Connecting People to Helpful Health ıformation," 2024.

[3] L. Sirtonski, "Demystifying Radiology Reports with ChatGPT," Radiological Socie f North America (RSNA), 2024.

[4] B. J. o. G. Practice, "Unintended consequences of patient online access to health :cords: a qualitative study in UK primary care," 2023.

[5] OCR technology in healthcare – general overview Plamondon, R., & Srihari, S. N. !000). On-line and off-line handwriting recognition: A comprehensive survey. IEEE ransactions on Pattern Analysis and Machine Intelligence, 22(1), 63–84.

[6] OCR.space API – Technical capabilities OCR.space. (n.d.). Free OCR API - Optica Character Recognition. Available at: https://ocr.space/ocrapi [Accessed: April 2025

[7] Cloud-based OCR advantages and limitations A. Paul. (2018). Cloud-based OCR /stems: Accuracy and scalability considerations. Journal of Cloud Applications, 12(: 5–59.

[8] Deep Learning in OCR systems Jaderberg, M., Simonyan, K., & Zisserman, A. !016). Reading Text in the Wild with Convolutional Neural Networks. International ɔurnal of Computer Vision, 116(1), 1–20.

[9] Arabic OCR challenges and multilingual support Zramdini, A., & Ingold, R. (1998) ptical font recognition using typographical features. IEEE Transactions on Pattern nalysis and Machine Intelligence, 20(8), 877–882

[10] E. S. Corporation, "MyChart Features," Epic Systems Corporation, August 2024. Online]. Available: https://www.mychart.com/features.

[11] WebMD, "How WebMD Can Help You Manage Your Health," 2023. [Online]. vailable: https://www.webmd.com/about-webmd.

[12] A. H. GmbH, "How Ada Helps with Health Assessment," Ada Health GmbH, 2023 Online]. Available: https://www.ada.com/how-ada-works.

[13] L. o. A. Holdings, "LabCorp Patient App Overview," LabCorp , 2022. [Online]. vailable: https://www.labcorp.com/patient.

[14] CodeSansar, "Problem Analysis," CodeSansar, [Online]. Available: ttps://www.codesansar.com/computer-basics/problem-analysis.htm.

[15] A. C. o. R. (ACR), " Patient-Friendly Radiology Reports," ACR, [Online]. Availabl ttps://www.acr.org/Practice-Management-Quality-Informatics/Imaging-3/Case-Studi atient-Engagement/Patient-Friendly-Radiology-Reports.

[16] "Functional and Non-Functional Requirements Diagram for SmartLab App".

[17] A. Digital, "Flutter App Development," Appify Digital, [Online]. Available: ttps://www.appify.digital/post/flutter-app-development.

[18] Flutter.dev, "Flutter - Build Apps for Any Screen," Flutter.dev, [Online]. Available: ttps://flutter.dev.

[19] G. Developers, " Text Recognition with ML Kit," Google , [Online]. Available: ttps://developers.google.com/ml-kit/vision/text-recognition/v2/android.

[20]] General OCR applications Smith, R. (2007). An overview of the Tesseract OCR engine. In Proceedings of the Ninth International Conference on Document Analysi and Recognition (Vol. 2, pp. 629–633). IEEE.

[21] O. H. Center, "Data Analysis with ChatGPT," OpenAI , [Online]. Available: ttps://help.openai.com/en/articles/8437071-data-analysis-with-chatgpt.

[22] N. P. D. K. J. T. P. S. a. T. D. P. Batra, " OCR-MRD: Performance Analysis of ifferent Optical Character Recognition Engines for Medical Report Digitization," esearchGate, 2023.

[23] .H. o. A. a. A. Andrew Bird, "From OCR to AI: The Evolution of OCR Technology ffinda, 2024.

[24] R. R. Solutions, "What is Optical Character Recognition (OCR) and How It Will ffect Medical Records?," [Online]. Available: :tps://www.recordrs.com/blog/what-is-optical-character-recognition-ocr-and-how-it-affect-medical-records.

[25] CodeSansar, "Problem Analysis," CodeSansar, [Online]. Available: :tps://www.codesansar.com/computer-basics/problem-analysis.htm.

[26] I. J. a. G. B. James Rumbaugh, The Unified Modeling Language Reference Manual econd Edition, Addison-Wesley Professional.

[27] M. Fowler, UML Distilled: A Brief Guide to the Standard Object Modeling anguage., Addison-Wesley..

[28] Google. "Introducing Gemini: our largest and most capable AI model." Google Blo Dec. 2023. [Online]. Available: :tps://blog.google/technology/ai/google-gemini-ai/#introducing-gemini. [Accessed: 3-Apr-2025].

[29] OCR.space, "OCR API - Optical Character Recognition as a Service," OCR.space, 024. [Online]. Available: https://ocr.space/ocrapi. [Accessed: Apr. 24, 2025].

[30] [1] Google Developers, "shared_preferences | Flutter Package," 2024. [Online]. vailable: https://pub.dev/packages/shared_preferences. [Accessed: Apr. 24, 2025].

[31] OCR with OCR.space API in Flutter," Medium – Flutter Community. :tps://medium.com/flutter-community/flutter-ocr-using-ocr-space-api-2d7ac25852fb

[32] H. o. A. a. A. Andrew Bird, "From OCR to AI: The Evolution of OCR Technology, ffinda, 2024.

[33] I. J. a. G. B. James Rumbaugh, The Unified Modeling Language Reference Manual econd Edition, Addison-Wesley Professional.

[34] M. Fowler, UML Distilled: A Brief Guide to the Standard Object Modeling anguage., Addison-Wesley..

[35] Brightsec. "Unit Testing." *Brightsec Blog*. [Online]. Available: :tps://www.brightsec.com/blog/unit-testing/. [Accessed: Apr. 24, 2025].

Appendix

Appendix A: User Authentication Functions

Files: `login_page.dart`, `register_page.dart`

This appendix includes the major functions used for handling user login and registration.

- `_handleLogin()`

```
void _handleLogin() async {
 if (_formKey.currentState!.validate()) {
   String username = _usernameController.text.trim();
   String password = _passwordController.text;


   if (username.isEmpty || password.isEmpty) {
     ScaffoldMessenger.of(context).showSnackBar(
       SnackBar(
           content: Text(isEnglish
               ? 'Please fill in all fields.'
               : 'يرجى تعبئة جميع الحقول.')),
     );
     return;
   }


   try {
     bool userExists =
         await DatabaseHelper.instance.loginUser(username, password);


     if (userExists) {
       ScaffoldMessenger.of(context).showSnackBar(
         SnackBar(
             content: Text(isEnglish
                 ? 'Login successful!'
```

```
                : 'تم تسجيل الدخول بنجاح!')),
      );


      Navigator.pushReplacement(
        context,
        MaterialPageRoute(
            builder: (context) =>
                HomePage(username: username, isGuest: guest)),
      );
    } else {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
            content: Text(isEnglish
                ? 'Invalid username or password.'
                : 'اسم المستخدم أو كلمة المرور غير صحيحة.')),
      );
    }
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
          content: Text(isEnglish
              ? 'Login failed. Try again later.'
              : 'فشل تسجيل الدخول. حاول مرة أخرى لاحقًا.')),
    );
  }
 }
}
```

- *_handleRegister()*

```
//Registration function
void _handleRegister() async {
 if (_formKey.currentState!.validate()) {
   String name = _nameController.text.trim();
   String email = _emailController.text.trim();
   String username = _usernameController.text.trim();
   String password = _passwordController.text;
   int age = int.tryParse(_ageController.text.trim()) ?? 0;

   final emailRegex = RegExp(r'^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$');

   if (!emailRegex.hasMatch(email)) {
     ScaffoldMessenger.of(context).showSnackBar(
       SnackBar(
           content: Text(isEnglish
               ? 'Enter a valid email address!'
               : 'أدخل بريدًا إلكترونيًا صالحًا!')),
     );
     return;
   }

   if (password.length < 8) {
     ScaffoldMessenger.of(context).showSnackBar(
       SnackBar(
           content: Text(isEnglish
               ? 'Password must be at least 8 characters!'
               : 'يجب أن تكون كلمة المرور 8 أحرف على الأقل!')),
     );
     return;
   }
```

```
  if (age <= 0) {
    ScaffoldMmessenger.of(context).showSnackBar(
      SnackBar(
          content: Text(
              isEnglish ? 'Enter a valid age!' : 'أدخل عمرًا صالحًا!')),
    );
    return;
  }

  // Check if username already exists
  bool usernameExists =
      await DatabaseHelper.instance.checkUsernameExists(username);
  if (usernameExists) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
          content: Text(isEnglish
              ? 'Username already exists. Please choose another.'
              : 'اسم المستخدم موجود بالفعل. يرجى اختيار اسم آخر.')),
    );
    return;
  }

  try {
    await DatabaseHelper.instance
        .registerUser(name, username, email, password, age);

    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
          content:
              Text(isEnglish ? 'User Registered!' : 'تم تسجيل المستخدم!')),
```

```
    );


    Navigator.pop(context);
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
          content: Text(isEnglish
              ? 'Registration failed. Try again.'
              : 'فشل التسجيل. حاول مرة أخرى.')),
    );
  }
 }
}
```

Appendix B: Report Upload and OCR Processing Functions

File: `report_upload_page.dart`

Detailed functions related to uploading medical reports, handling files, and OCR extraction:

- `_pickFile()` *(already summarized in the main report but included here)*

```
Future<void> _pickFile() async {
 if (widget.isGuest) {
   await _showLoginPrompt();
   return;
 }


 FilePickerResult? result = await _pickFileFromDevice();
 if (result == null || result.files.single.path == null) {
   return;
```

```
 }


 File selected = File(result.files.single.path!);

 bool isPDF = _isPDFFile(selected);


 _startLoading(isPDF);


 try {

   String analysisText = await _analyzeFile(selected, isPDF);

   _navigateToResultsPage(analysisText);

 } catch (e) {

   _handleAnalysisFailure(e);

 } finally {

   _stopLoading();

 }

}
```

- *extractTextFromMultipartFile()*

```
Future<String> extractTextFromMultipartFile(http.MultipartFile file,
bool isEnglish) async {

 final uri = Uri.parse("https://api.ocr.space/parse/image");

 final request = multipartRequestFactory(uri)

   ..fields['apikey'] = apiKey

   ..fields['language'] = isEnglish ? 'eng' : 'ara'

   ..files.add(file);


 final response = await request.send();

 final responseData = await response.stream.bytesToString();


 if (response.statusCode == 200) {

   final jsonResponse = jsonDecode(responseData);
```

```
    final text = jsonResponse['ParsedResults'][0]['ParsedText'];
    return text;
  } else {
    throw Exception("OCR failed: $responseData");
  }
}
```

Appendix C: AI-Based Analysis Functions

File: `gemini_service.dart`

This section contains the function for communicating with the Gemini API, handling responses, and generating analysis :

- `analyzeMedicalReport()`

```
Future<String> analyzeMedicalReport(File file, bool isEnglish) async {
    String apiUrl =

"https://generativelanguage.googleapis.com/v1/models/gemini-1.5-flash:ge
nerateContent?key=$apiKey";

    List<int> fileBytes = await file.readAsBytes();
    String base64File = base64Encode(fileBytes);

    String fileExtension = file.path
        .split('.')
        .last
        .toLowerCase();
    String mimeType =
    (fileExtension == 'pdf') ? 'application/pdf' : 'image/jpeg';

    String prompt = isEnglish
        ? """
```

```
        This is a medical test report. Please analyze the attached file
(image or PDF) and return the result in the following clear format:



        1. Abnormal values:

           - Test name: value (High or Low)



        2. Simple explanation of each abnormal value:

           - Test name: short explanation



        3. Possible medical conditions and advice:

           - Test name: related conditions and medical advice (e.g., see
a doctor, or monitor)



        Please keep it short, informative, professional, and avoid using
symbols like ** or markdown formatting.

        """

       : """

هذا تقرير فحص طبي. يرجى تحليله وترتيب النتيجة بالتنسيق التالي:



        ١. القيم الخارجة عن المعدل الطبيعي ( سرد الارقام باللغه العربيه )          :

              - اسم التحليل: القيمة (مرتفعة أو منخفضة)



٢. شرح بسيط قصير لكل القيم الخارجة عن المعدل الطبيعي :

              - اسم التحليل: شرح طبي مختصر



٣. الحالات الصحية المحتملة والنصائح بشكل قصير ومختصر :

              - اسم التحليل: الأمراض المحتملة والنصيحة الطبية



! ملاحظات مهمة :

- لا تكتب أسماء التحاليل او الاختصارات أو الشروحات باللغة الإنجليزية إطلاقًا.

- استخدم فقط أسماء التحاليل الطبية العربية المعروفة.

- اجعل الإجابة منسقة وواضحة ومناسبة لفهم غير المختصين.

        """;
```

```
  var requestBody = {
    "contents": [
      {
        "parts": [
          {
            "inlineData": {"mimeType": mimeType, "data": base64File}
          },
          {"text": prompt}
        ]
      }
    ]
  };

  var response = await http.post(
    Uri.parse(apiUrl),
    headers: {"Content-Type": "application/json"},
    body: jsonEncode(requestBody),
  );

  if (response.statusCode == 200) {
    var jsonResponse = jsonDecode(response.body);
    return jsonResponse["candidates"][0]["content"]["parts"][0]["text"]
??
        "No explanation found.";
  } else {
    return "Error: ${response.body}";
  }
 }
```