Egyptian E-Learning University

Faculty of Computers & Information Technology

Development of a Skin Disease Detection System Using Neural Network

By

Marina Emad Mored Habeb	**2101013**
Mera Refat Gaballah	**2101090**
Martina William Takawy	**2101031**
Sara Raafat Rasmy	**2101569**
Nardeen Maged Alfons	**2101354**
Mariam Sameh Fawzi	**2100979**
Marina Ashraf Farouk	**2101014**

Supervised by

Dr. Amany Magdy Mohamed

professor, Faculty of Computers and Information at Egyptian E-Learning University

Assistant

Eng.Mohamed Mostafa Saad

Demonstrator, Faculty of Computers and Information at Egyptian E-Learning University

Sohag-2025

Abstract

Skin diseases are among the most common health issues worldwide, affecting millions of people of different ages. These conditions can range from mild irritations to severe and chronic infections that significantly impact a person's quality of life. Therefore, early and accurate diagnosis is crucial to prevent complications, reduce the burden on healthcare systems, and ensure timely and effective treatment. This graduation project focuses on developing an intelligent system capable of automatically detecting and classifying different types of skin diseases using advanced deep learning techniques. The system is trained on a comprehensive and well-organized dataset of labeled skin images that include examples of common dermatological conditions. Through this training process, the model learns to extract visual features and patterns that help distinguish between various diseases. The core technology used in this project is convolutional neural network (CNNs), which have proven highly effective in image analysis and medical diagnostics. The primary goal is to assist dermatologists in making faster and more accurate diagnoses, while also offering patients a preliminary self-assessment tool.This system has the potential to significantly improve access to dermatological care, especially in rural or underserved areas that lack access to specialists, thereby contributing to better overall public health outcomes.

Acknowledgments

First and foremost, praises and thanks to God, the Almighty, for his blessing throughout our years in college and our graduation project to complete this stage of our life successfully.

We have made efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

Thanks to Egyptian E-Learning University Sohag center for helping us to reach this level of awareness.

We would like to express our deep and sincere gratitude to our project supervisor **DR. Amany Magdy Mohamed** for giving us the opportunity to lead us and providing invaluable guidance throughout this project. It was a great privilege and honor to work and study under his guidance. We are extremely grateful for what he has offered us.

We are especially indebted to say many thanks to **Eng. Mohamed Mostafa Saad** for guidance and constant supervision as well as for his patience, friendship, empathy, and great sense of humor. His dynamism, vision, sincerity and motivation have deeply inspired us.

Finally, we would like to express our gratitude to everyone who helped us during the graduation project.

Contents

List of Figures:

List of Tables:

List of Acronyms/Abbreviations:

(AI) Artificial Intelligence

(CNN)Convolutional Neural Networks

(SVM) Support Vector Machine

(KNN) K-Nearest Neighbor

(ReLU) Rectified Linear Unit

(VGG16) Visual Geometry Group 16

Chapter 1

Introduction

1.1 General

Skin diseases are among the most widespread health problems affecting people across the globe. They vary greatly in type and severity, ranging from mild conditions such as acne and eczema to more serious diseases like melanoma and psoriasis. Due to the visible nature of skin disorders, they often cause not only physical discomfort but also psychological stress and social anxiety for affected individuals. Early detection and appropriate intervention are therefore essential to prevent complications and improve patients' quality of life.

However, diagnosing skin diseases can be a complex task, even for experienced dermatologists. This complexity arises from the fact that many skin conditions share similar visual symptoms, such as rashes, discoloration, or lesions, which can make accurate diagnosis challenging without professional expertise and tools.

To address this challenge, this project aims to develop an intelligent system based on CNNs that can automatically detect and classify various skin diseases from image data. By leveraging the power of deep learning and computer vision, the proposed system can assist in identifying potential skin conditions with high accuracy.

The goal is not to replace medical professionals but to provide an initial assessment tool that offers preventive advice and encourages users to seek medical attention promptly. This can be especially beneficial in areas with limited access to dermatologists, helping bridge the gap between early detection and proper treatment . [1]

1.2 Background

Skin diseases are among the most common health issues worldwide, affecting people of all ages and backgrounds. These conditions include acne, eczema, psoriasis, and other non-cancerous dermatological problems. With the rapid development of Artificial Intelligence (AI), especially deep learning, there has been growing interest in using these technologies to support the diagnosis of skin diseases.

The idea of using computers for dermatological diagnosis began in the late 1990s, when researchers started applying image processing techniques to detect patterns in skin images. However, due to limitations in computational power and the lack of large, labeled datasets, early attempts faced significant challenges.

In recent years, the emergence of powerful deep learning models such as CNNs , along with access to large and well-annotated datasets like ISIC, has enabled significant progress in automated skin disease classification. These models can now identify complex features in medical images and offer accurate diagnostic support.

The main motivation behind developing AI-based tools for skin disease diagnosis is to provide faster, more accessible, and more consistent diagnostic assistance—especially in regions where dermatologists are not readily available. These tools can support general practitioners and improve the early detection and management of skin conditions.

Our project builds on these advancements by developing a system that uses deep learning techniques to classify common skin diseases based on image data. By leveraging the power of AI, we aim to contribute to the field of smart healthcare and offer an efficient, automated solution for dermatological diagnosis. [1]

1.3 Motivation

Dermatological diseases are a significant global health challenge, affecting millions of individuals worldwide. Despite their widespread occurrence, diagnosing these conditions often presents a considerable challenge due to the diverse nature of skin diseases and the overlapping symptoms that many share.

As a result, accurate diagnosis frequently requires the expertise of dermatologists and specialized medical equipment. This can create barriers to timely diagnosis and treatment, especially in areas with limited access to healthcare professionals.

The rising prevalence of dermatological conditions, combined with the shortage of dermatologists in many regions, makes it increasingly critical to develop efficient diagnostic tools. This growing demand for more accurate and accessible diagnostic methods is vital not only to improve healthcare delivery but also to reduce the burden on healthcare systems. The ability to diagnose skin diseases early can significantly reduce the risk of complications and lead to more effective treatments, ultimately improving patient outcomes and quality of life.

In recent years, the application of AI and deep learning techniques has shown great promise in transforming the healthcare industry, particularly in the field of medical imaging. AI-driven models, such as CNNs , have the potential to automate the diagnostic process by analyzing medical images with a level of precision and speed that surpasses traditional methods. By leveraging these technologies, it is possible to offer innovative solutions that not only enhance diagnostic accuracy but also make the process more efficient, which is crucial in managing the growing number of dermatological cases.

The use of medical image analysis for dermatological diseases, specifically through deep learning models, offers significant advantages over traditional diagnostic approaches. These AI-based models can process and analyze large datasets of skin images rapidly, leading to quicker and more accurate diagnoses. This not only enhances the overall efficiency of healthcare delivery but also

facilitates early detection of diseases, which is critical for improving patient outcomes. By providing early assessments and preventive advice based on the analysis of images and data, this system can assist patients in seeking medical attention sooner, thereby reducing the risk of severe health complications.
This project seeks to develop an intelligent model that analyzes and classifies dermatological conditions from skin images, offering a valuable tool for early detection. The system will provide essential preventive advice and guidance, encouraging patients to visit a healthcare professional for further evaluation, ensuring timely and accurate medical intervention.

1.4 Problem Statement

Skin diseases are among the most widespread health conditions, and early diagnosis plays a crucial role in effective treatment. However, many individuals face significant delays in diagnosis, which can lead to the worsening of their condition—especially in cases where skin diseases progress rapidly if left untreated.

One of the major challenges is the general lack of awareness regarding skin conditions, their symptoms, and the preventive measures needed to manage them. This limited understanding often leads to unnecessary anxiety and hesitation in seeking medical attention. Furthermore, people frequently rely on online images for self-diagnosis, which can be misleading due to the visual similarities between different dermatological conditions. Such confusion increases the risk of misdiagnosis or neglect.

Another critical issue is the difficulty in accessing reliable medical services. Many individuals are unaware of nearby hospitals or specialized dermatologists who can provide accurate and timely care for skin-related concerns. This gap in accessibility and awareness further contributes to delayed treatment and poor health outcomes.

Therefore, there is a growing need for an intelligent, accessible system that can assist in the preliminary identification of common skin diseases, raise awareness, and guide users toward appropriate medical resources.

1.6 Objectives

1.6.1 Main Objective

The primary goal of this project is to develop an AI-powered mobile application capable of providing fast, accurate, and accessible diagnosis of common skin diseases through image analysis. The app aims to improve early detection, increase user awareness, and facilitate access to specialized medical care.

1.6.2 Specific Objectives

- To design and implement an image processing module that uses deep learning techniques for accurate classification of skin diseases.
- To create a reliable information database within the app that educates users about various skin conditions, preventive measures, and care guidelines.
- To integrate an AI-driven diagnostic system that provides instant feedback and recommendations based on analyzed images.
- To develop a location-based service feature that helps users find nearby hospitals and specialized dermatology clinics for professional medical assistance.
- To ensure the app has a user-friendly interface to encourage widespread adoption and ease of use for individuals with varying levels of technical expertise.

Chapter 2

Related Work

2.1Related Work

paper	Published Date	Author	Algorithms
Skin Disease Detection using Image Processing and Machine Learning	May17,2023	SamalJahnavi, Nallamilli Gayatri, Thota Sravani.Publication, ...elt.	Naive Bayes (97.5%) accuracy, SVM (90%) and KNN (93.75%)
Advance Sy Of Skin Diseases Detection Using Image Processing Methods	in 2022	Dr.Sandra Paola , Dr. María A ,Dr Jhony A, ...elt .	K-Nearest Neighbor (KNN) (91.2%) with Symlet Analysis, and (SVM) lower accuracy
A Method of Skin Disease Detection Using Image Processing and Machine Learning	in 2019	Nawal Soliman ALKolifi ALEnezi .	Utilizes AlexNet (pretrained CNN) for feature extraction and Support Vector Machine (SVM) achieving (100%) accuracy

Table 2.1: Related Work

2.2 Limitation

In previous studies related to dermatological disease classification, it was observed that the accuracy rates were not as high as they could have been.

Additionally, most of these studies focused on classifying only a limited number of diseases, which restricted the model's practical application. Expanding the scope to include a wider range of skin diseases would make the model more robust and versatile, addressing a critical gap in the existing research.

2.3 Summary

Recent studies on skin disease detection using image processing and machine learning have explored various algorithms to enhance classification accuracy. For instance, Naive Bayes, SVM, and KNN were applied in one study, achieving accuracies of 97.5%, 90%, and 93.75%, respectively. Another approach utilized KNN with Symlet analysis and SVM, resulting in a 91.2% accuracy, while a deep learning-based method employed AlexNet for feature extraction and SVM for classification, reporting 100% accuracy. However, these studies faced notable limitations: many did not achieve consistently high accuracy across different cases, and most focused on a narrow set of diseases. This limited the models' generalizability and practical use. Expanding future models to cover a broader range of skin conditions would improve robustness and increase their real-world effectiveness. [1] [2] [3]

Chapter 3

Proposed system

3.1 Approach used to solve the problem

3.1.1 CNN (Convolutional Neural Network)

CNNs (CNNs) are a class of deep learning models primarily used for image classification and computer vision tasks. The architecture is inspired by how the human visual system works — learning features gradually from low-level (edges, colors) to high-level (shapes, objects).

Components of CNN Architecture

1. Input Layer

The input layer of a CNN receives the raw image data, which is usually represented as a 3D matrix of height, width, and channels. The height and width define the image size, while the channels represent colors like red, green, and blue. This format helps the network analyze both spatial and color information.

2. Convolutional Layer

In the convolutional layer, small filters or kernels are applied to the input image. These filters, typically sized 3×3 or 5×5, slide over the image performing element-wise multiplication at each position. This process produces a feature map that highlights important patterns in the image, such as edges, corners, and textures, which help to distinguish key details within the image.

3. Activation Function (ReLU)

After each convolutional layer, an activation function is applied to introduce non-linearity into the model. The most commonly used activation function is ReLU (Rectified Linear Unit), which replaces all negative values with zero while keeping positive values unchanged. This process allows the model to learn complex patterns by enabling it to capture non-linear relationships within the data.

4. Pooling Layer

The pooling layer serves to reduce the dimensions of the feature maps generated by the convolutional layers. The most commonly used pooling method is Max Pooling, which works by selecting the maximum value within a specified window of the feature map. This reduction helps decrease computational load and controls overfitting while preserving important features.

5. Flatten Layer

After completing the convolution and pooling operations, the resulting two-dimensional feature maps are flattened into a one-dimensional vector. This flattened vector is then passed into the fully connected (dense) layers, which process the information for final classification or prediction tasks.

6. Fully Connected (Dense) Layers

Fully connected (dense) layers function similarly to traditional neural networks. They learn to combine the extracted features from previous layers and make decisions based on them. The final dense layer typically uses a Softmax activation function for multi-class classification tasks, or a Sigmoid activation function for binary classification.

7. Output Layer

The output layer produces the final prediction, such as the class of a skin disease. In classification tasks, this layer typically provides a probability distribution over all possible classes, indicating the likelihood of the input belonging to each class. [4]

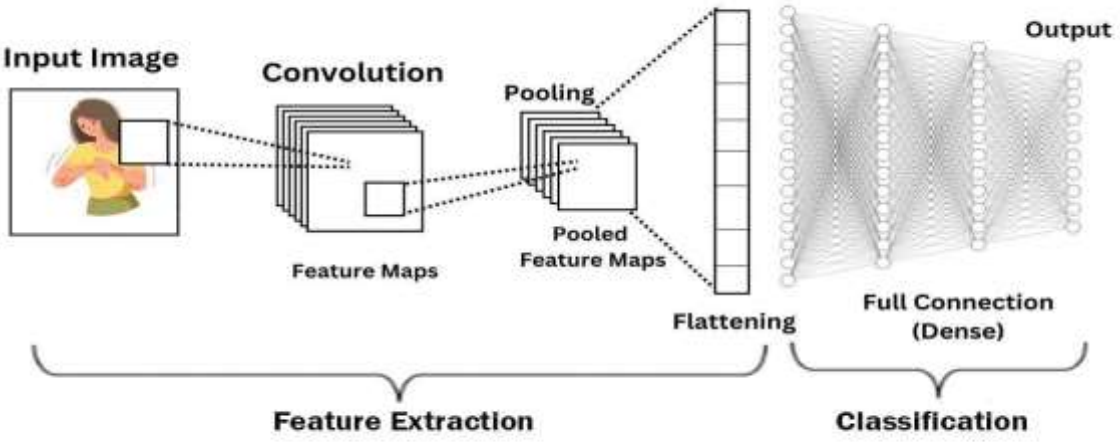


Figure 3.1 : CNN Architecture

3.2 System architecture

3.2.1 Flowchart

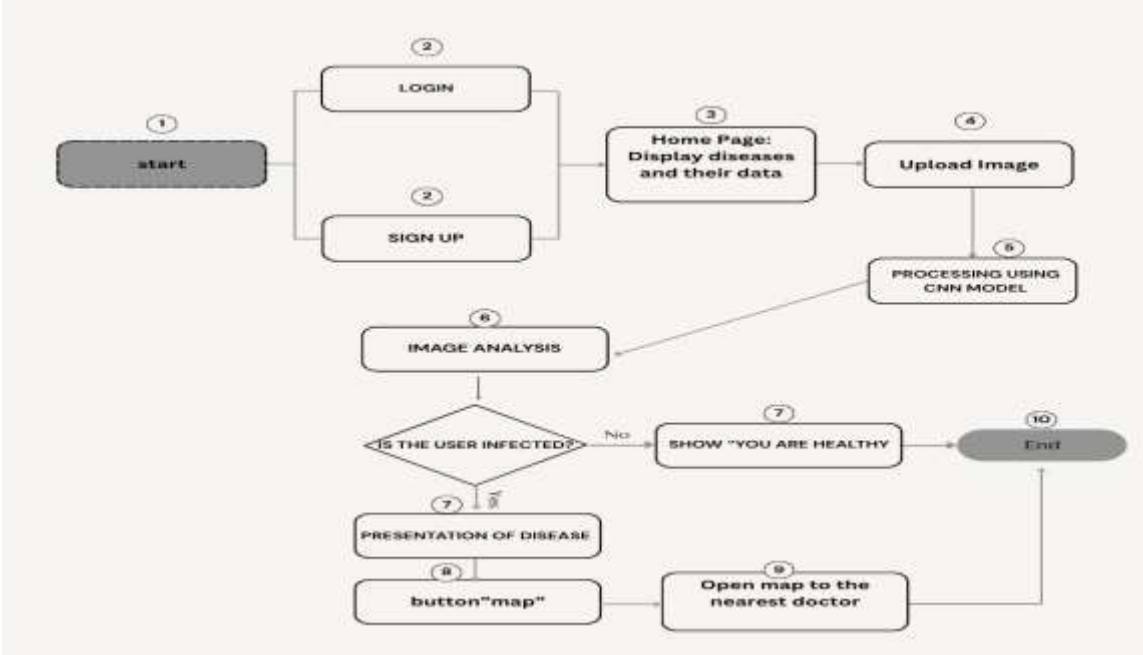


Figure 3.2: Flowchart

3.2.2 Use Case

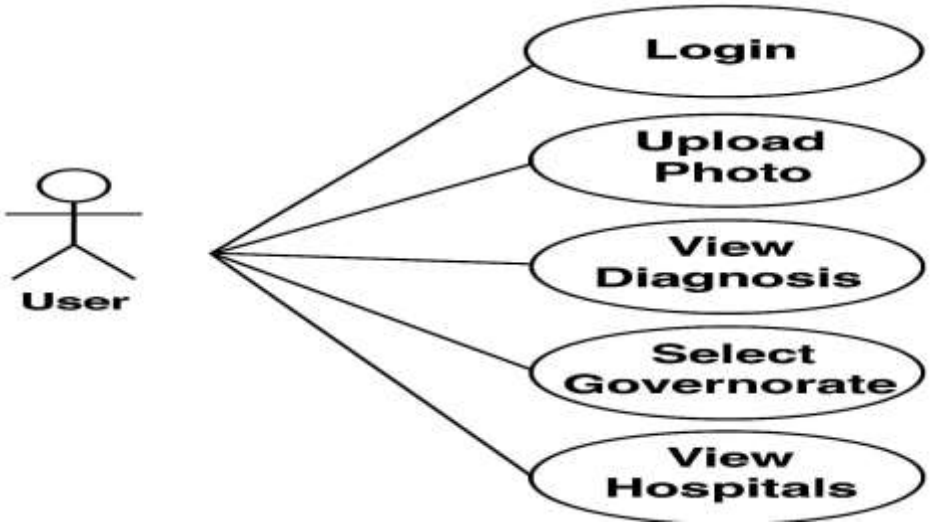


Figure 3.3 : Use Case

3.2.3 ER diagrams

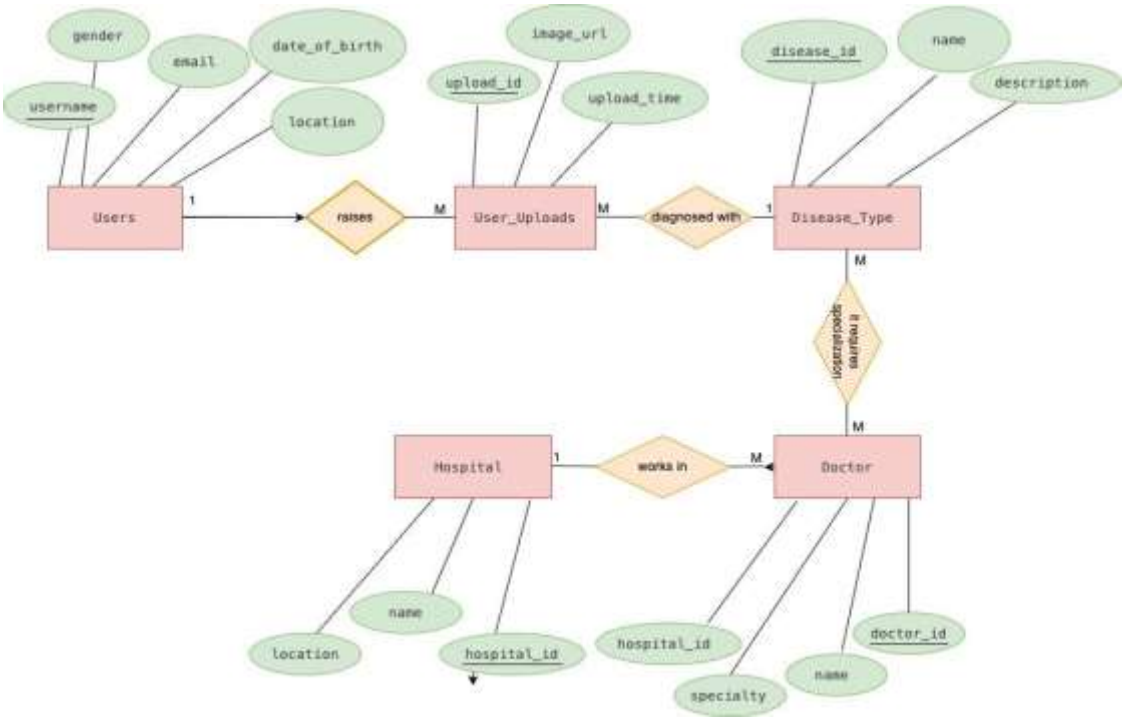


Figure 3.4 : ER diagrams

3.3 Algorithms used

3.3.1 EfficientNetB0

EfficientNetB0 is a deep learning model from the EfficientNet family, specifically designed for image classification tasks. It is a type of CNN that introduces a *compound scaling* method, which efficiently balances the network's depth, width, and input resolution. Unlike traditional models that scale only one dimension, EfficientNetB0 scales all three dimensions simultaneously in a structured and coordinated way, resulting in better accuracy and lower computational cost.

The model begins with a lightweight and efficient base architecture and processes input images through a series of MBConv blocks—which include convolutional layers, batch normalization, activation functions, and sometimes skip connections. These blocks help preserve feature information throughout the network. After passing through all layers, the model applies global average pooling to summarize the features, followed by a fully connected layer and a Softmax function to produce the final classification output.

Thanks to its efficiency and strong performance, EfficientNetB0 is widely used in medical image analysis, such as skin disease classification, where it helps in accurately identifying disease categories with minimal resource usage.

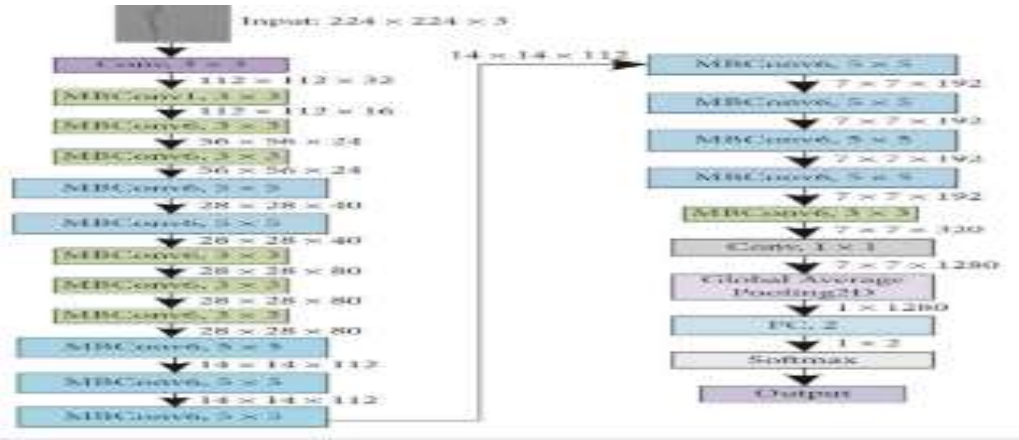


Figure 3.5 :EfficientNetB0 model Architecture

3.3.2 ResNet50

ResNet50 is a deep CNN with 50 layers, known for introducing the concept of residual learning. In very deep networks, accuracy may decrease due to the vanishing gradient problem. ResNet50 addresses this by using shortcut (skip) connections that allow gradients to flow directly through the network without degradation.

The architecture consists of multiple stacked blocks, where each block includes convolutional layers followed by batch normalization and activation functions. The key feature is that each block also adds its input directly to its output, forming a *residual connection*. This helps the model learn identity mappings, making it easier to train very deep networks.

ResNet50 is highly effective in medical image classification, including skin disease detection, because of its ability to capture complex features while maintaining training stability and accuracy.

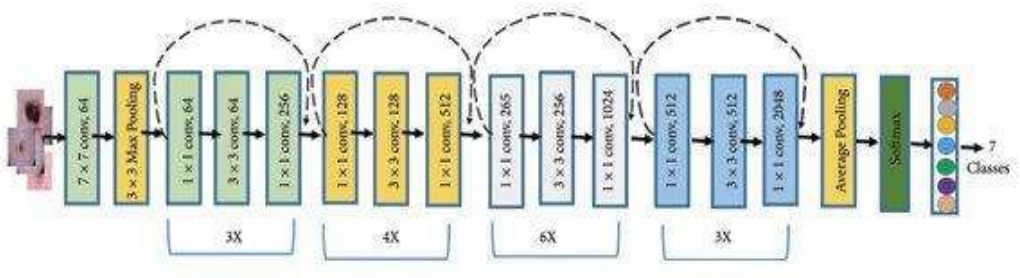


Figure 3.6 : ResNet50 model Architecture

3.3.3 VGG16

VGG16 is a classical CNN architecture that contains 16 layers, mostly composed of 3x3 convolutional filters followed by max pooling and fully connected layers. It is known for its simple and uniform architecture, which makes it easy to implement and understand.

While VGG16 has a large number of parameters and is computationally intensive, it delivers strong performance in many image classification tasks. The network progresses through increasing depth, extracting richer features at each level before reaching the final softmax classification layer.

VGG16 is often used in academic research and medical imaging tasks like skin disease classification, especially when clarity and explainability of the architecture are important.

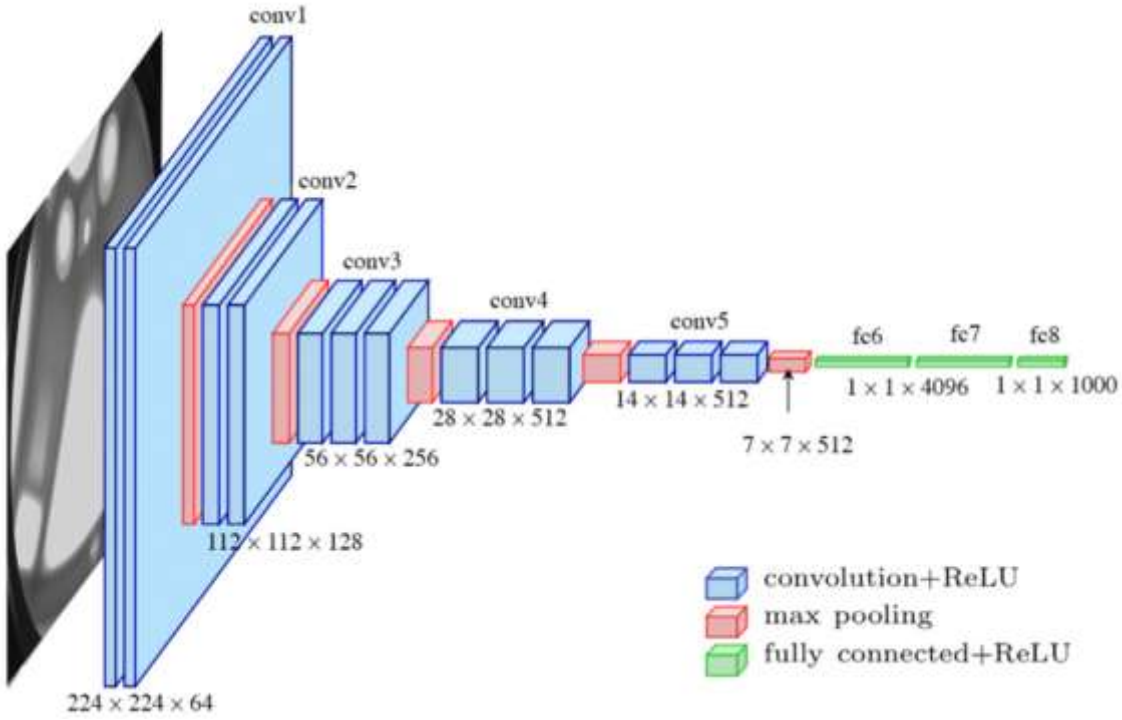


Figure 3.7 : VGG16 model Architecture

3.3.4 InceptionV3

InceptionV3 is an advanced CNN architecture designed for efficiency and depth. It utilizes Inception modules, which perform several convolutions (1x1, 3x3, 5x5) in parallel within the same layer. This allows the network to extract features at multiple scales simultaneously.

The model incorporates techniques like factorized convolutions, auxiliary classifiers, and label smoothing, making it both accurate and computationally efficient. It is deeper and more optimized than earlier versions of the Inception family.

InceptionV3 is well-suited for complex medical image classification tasks, such as diagnosing skin conditions, because of its ability to learn fine-grained features with high precision.

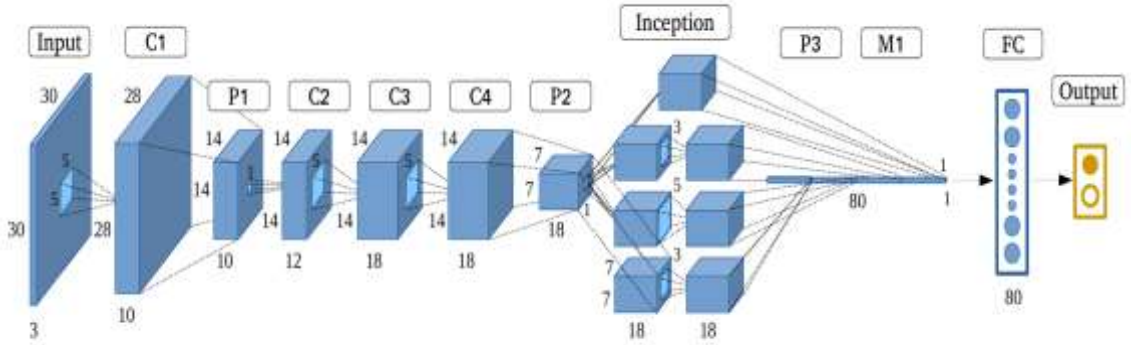


Figure 3.8 : InceptionV3 model Architecture

3.3.5 MobileNetV2

MobileNetV2 is a lightweight CNN architecture optimized for mobile and embedded devices. It introduces depthwise separable convolutions to reduce computational cost and inverted residual blocks to preserve important features while minimizing parameters.

The architecture is designed to maintain a good balance between speed, efficiency, and accuracy, making it ideal for real-time applications. Despite being smaller and faster than traditional networks, MobileNetV2 still performs well in image classification tasks.

In medical applications like skin disease detection, MobileNetV2 is a preferred choice when deploying on smartphones or resource-limited devices, allowing fast and accurate predictions on the go**.** [5] [6] [7] [8]

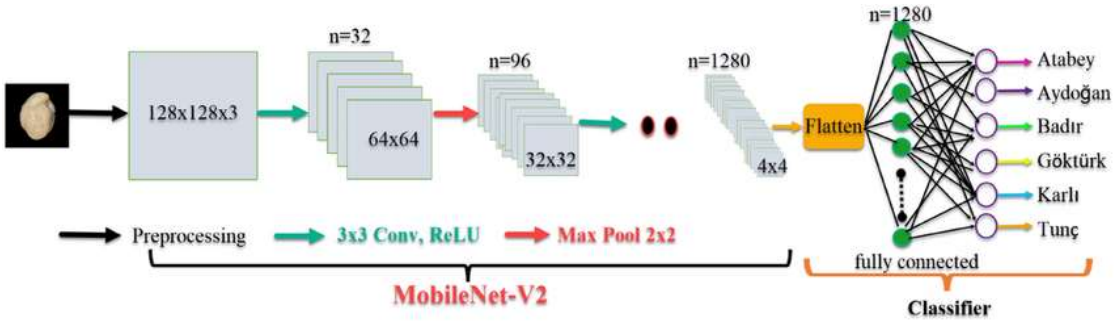


Figure 3.9: MobileNetV2 model Architecture

3.6 Frameworks Used

3.6.1 TensorFlow

TensorFlow is an open-source deep learning framework developed by Google. It is widely used for building and training neural network models. TensorFlow supports running models on various hardware such as CPUs, GPUs, and TPUs, making it ideal for medical image classification tasks like skin disease detection. It also provides the high-level Keras API, which simplifies the process of designing and training neural networks.

3.6.2 Flask

Flask is a lightweight web framework written in Python used to build web applications and backend servers. In your project, Flask is used to create a simple backend server that receives images from the mobile app, sends them to the AI model for analysis, and returns the diagnosis results to the app. Flask is easy to set up and helps connect your AI model with the external Flutter application.

3.6.3 Flutter

Flutter is an open-source UI framework developed by Google for building cross-platform mobile and web applications with attractive and fast user interfaces. In your project, Flutter is used to develop the mobile app that allows users to upload skin images, receive diagnosis results, and view nearby hospitals based on their location.

3.6.4 Firebase

Firebase is a cloud-based development platform provided by Google that offers a suite of ready-to-use services to simplify app building without the need to create servers or databases from scratch. In your project, Firebase can be used for user data storage, authentication (sign-in), image storage, and app usage analytics. Firebase integrates smoothly with Flutter apps through dedicated libraries that allow easy access to its services. [9] [11]

Chapter 4

Implementation

4.1 Technologies, tools, and programming languages used.

The system was developed using the following technologies and tools:

- **Programming Language:** Python , Dart
- **Framework:** TensorFlow , Kera , Flutter SDK , image_picker , Flask , Firebase
- **Development Environment:** Jupyter Notebook , Google Colab , Android Studio , VS Code
- **Dataset Structure:** Images organized into folders by class (train/test split)
- **File Storage:** Local directories (path-based image loading)

4.2 Key components/modules of the system.

The system consists of the following main components:

4.2.1 Data Loading and Preprocessing Module

- Loads images from structured directories
- Resizes images to (224x224)
- Converts them to RGB
- Applies EfficientNet-specific normalization
- Encodes class labels into numerical and one-hot formats

4.2.1 Model Building Module

- Utilizes the pretrained EfficientNetB0 as the base model
- Freezes base model weights for transfer learning
- Adds custom dense layers with ReLU and softmax activations
- Compiles the model using the Adam optimizer and categorical cross-entropy loss

4.2.2 Training Module

- Trains the model using 80% of the dataset
- Uses batch size of 64 and 20 epochs
- Validates performance on the remaining 20% of the dataset

4.2.3 Evaluation Module

- Predicts labels on validation and test sets
- Calculates performance metrics: Accuracy, Precision, Recall, F1-Score, and AUC
- Visualizes confusion matrix and training/validation accuracy and loss curves

4.2.4 Prediction Module

- Loads the trained model
- Applies it on new unseen test images
- Outputs predicted class labels for each image

4.2.5 Splash Screen Component

- Displays the app highlights key features..
- Automatically navigates to the Login or Signup screen.

4.2.6 Authentication Component

- Allows users to log in or sign up via the Flutter app.
- Sends authentication requests (login/signup) to the Firebase backend for verification.

4.2.7 User Interface Component

- Shows a list of 8 skin diseases with images, names, and brief descriptions.
- Provides a search bar for easy disease lookup.
- Displays user information (avatar based on gender, username, email) fetched from Firestore .

4.2.8 Disease Information Component

- Presents detailed information about each disease, including symptoms, causes, and prevention.

4.2.9 Image Upload & Diagnosis Component

- Enables users to upload images by taking a photo or selecting from the gallery.
- Uploads images to the Flask backend for analysis.
- Receives and displays diagnosis results returned from the backend.

4.2.10 Hospitals Locator Component

- Shows a “Hospitals” button after diagnosis if a condition is detected.
- Allows users to select their governorate.
- Retrieves and displays a list of hospitals with addresses from Firestore.
- Shows the selected hospital location on a map.

4.2.11 Settings Component

- Supports switching between Light and Dark themes.
- Supports switching between English and Arabic languages.
- Allows users to log out.

4.2.12 Backend Component

- Provides RESTful API endpoints for:

    Authentication (login/signup).

    Image upload and analysis.

    Fetching disease data.

    Fetching hospital data.

- Integrates with a database (Firestore or other) to store user, disease, and hospital data.
- Hosts the Machine Learning model for image diagnosis.
- Processes image input and returns diagnosis results to the Flutter app.

4.3Challenges faced and how they were resolved.

During the development of the AI model, we encountered several key challenges that required strategic solutions:

- **Limited Dataset Size**:

  Initially, the dataset contained only 1,159 images, which was insufficient for training a robust deep learning model.
  Solution: We applied data augmentation techniques to artificially expand the dataset, increasing the total number of images to 4,285. This helped improve the model's ability to generalize and reduced the risk of overfitting.

- **Random Parameter Selection**:

  At first, the model's hyperparameters (like learning rate, batch size, etc.) were chosen randomly without a structured method, leading to inconsistent performance.
  Solution: We implemented **Grid Search**, which systematically explores combinations of hyperparameters to find the best configuration. This approach helped us stabilize and optimize the training process.

- **Hardware Limitations with Increasing Dataset and Classes**:

  We considered expanding the number of skin disease classes from 8 to 10. However, due to hardware limitations and the heavy computational cost of running Grid Search on a larger dataset, this was not feasible.
  Solution:We decided to maintain the original 8-class setup, balancing between model complexity and system feasibility.
  Throughout the process, we worked on optimizing the model to achieve the best possible performance.
  This focused and well-structured approach allowed us to reach a **validation accuracy of 96%**, demonstrating the model's effectiveness and our success in addressing the hardware and complexity limitations.

During the development of the Flutter application, we encountered several practical challenges that required creative and technical solutions to ensure a smooth user experience

- **Integrating Flutter with Flask Backend**

  One of the main challenges was setting up smooth communication between the Flutter frontend and the Flask backend, especially handling image uploads and receiving diagnosis results.
  Solution: We used HTTP requests with proper headers and multipart form data in Flutter to send images. On the Flask side, we ensured correct API routes to accept and process these images, then return results in JSON format.[16][18]

Chapter 5

Testing & Evaluation

5.1 Testing strategies

To ensure the quality, reliability, and user satisfaction of the skin disease diagnosis application, a comprehensive testing strategy was employed throughout the development process. This strategy includes three main types of testing:

5.1.1 Unit Testing

Unit testing focuses on verifying the correctness of individual components or functions independently. This allows early detection of bugs and ensures each module behaves as expected.

- Validating input fields such as email and password in the Flutter app.
- Testing theme switching logic between light and dark modes.
- Ensuring the language toggle updates the UI correctly between English and Arabic.
- Verifying backend Flask functions, including image preprocessing and model inference accuracy.

5.1.2 Integration Testing

Integration testing verifies the interactions between different modules to confirm they work together properly, ensuring the system functions as a whole.

- End-to-end testing of image upload from Flutter app to Flask backend and receiving diagnosis.
- Full user authentication flow from sign-up to login and data retrieval.
- Checking UI updates propagation, such as language changes across multiple screens.

5.1.3 User Testing

User testing assesses usability and user experience by involving real users to perform typical tasks and providing feedback for improvements

- Tasks included account registration, image uploading, diagnosis review, and hospital search.
- Feedback was collected on UI clarity, button labeling, and upload responsiveness.
- Resulted in improvements to button labels, and image upload performance.

5.2 Performance metrics

We evaluated our model using several key performance metrics to ensure effectiveness:

```
26/26 ━━━━━━━━━━━━━━━━━━━━ 12s 406ms/step
26/26 ━━━━━━━━━━━━━━━━━━━━ 11s 414ms/step
Accuracy: 0.9642416769420469
Precision: 0.9643905624490097
Recall: 0.9642416769420469
F1-score: 0.9640347375777337
AUC-ROC curve: 0.9991392810044847
```

Figure 3.10 : Accuracy of model

These metrics indicate a highly accurate and reliable model for skin disease classification. We also optimized the training pipeline for speed and resource efficiency to enable smooth training on limited hardware. Scalability was considered, but constrained by hardware limitations, leading us to focus on 8 classes instead of expanding to more.

5.3 Comparison with existing solutions

Our approach focuses on diagnosing 8 specific skin diseases with high accuracy, achieving results that surpass many existing studies that either limit themselves to fewer diseases or only perform binary classification (disease presence or absence) rather than precise diagnosis. While some prior works have focused on limited classifications, this model reaches a balanced and effective solution suitable for real-world applications.

Chapter 6

Results & Discussion

6.1 Introduction

This chapter presents a comprehensive overview of the results obtained from implementing the AI-powered skin disease classification system and the mobile application built with Flutter. The project aimed not only to develop an accurate machine learning model but also to integrate it into a practical, user-friendly mobile app that allows users to upload an image, receive a diagnosis, and access useful medical information and nearby hospital recommendations.

The system was developed in two interconnected phases:

- **Phase 1:** Training a deep learning model using the EfficientNetB0 architecture to classify 8 common skin diseases.
- **Phase 2:** Building a cross-platform Flutter app that enables user interaction, disease information browsing, image uploading, and diagnosis retrieval. [4]

6.2 Summary of Findings

6.2.1 AI Model Results:

- The dataset was expanded from 1,159 to 4,285 images using data augmentation techniques to improve generalization.
- After fine-tuning the EfficientNetB0 model with grid search, the following results were achieved:
    - **Accuracy:** 96.42%
    - **Precision:** 96.44%
    - **Recall:** 96.42%
    - **F1-Score:** 96.40%
    - **AUC-ROC:** 0.9991

These results indicate a highly accurate and dependable model suitable for real-world classification of skin diseases.

6.2.2 Flutter App Results:

- The app was developed using Flutter with a bilingual interface (Arabic and English).
- Key features include:
  - Login and registration system
  - Image upload via camera or gallery
  - Real-time diagnosis using the Flask backend
  - Detailed disease information (symptoms, causes, prevention)
  - Hospital locator based on governorate selection
  - Dark/light mode toggle and multi-language support
- The app delivers a smooth and intuitive experience for users of all ages.

6.3 Interpretation of Results (Did the project meet its objectives?)

Yes, the project successfully achieved its main objectives:

- Developed an accurate and robust AI model for diagnosing 8 skin diseases.
- Integrated the model into a user-friendly mobile application with real-world functionality.
- Achieved strong performance metrics (over 96% accuracy).
- Provided educational content and location-based support within the app.

The solution not only delivers accurate medical insights but also guides users toward further medical assistance—making it a complete diagnostic and support system.

6.4 Limitations of the proposed solution

Despite the promising results achieved by the overall skin disease diagnosis application, several limitations should be considered:

- **Limited Disease Coverage**
  The AI model currently supports only eight skin conditions, which limits its effectiveness in diagnosing rare or complex dermatological cases.
- **Sensitivity to Image Quality**
  The accuracy of the diagnosis may be affected by unclear, low-resolution, or poorly lit images captured in uncontrolled environments.
- **No Real-Time Medical Consultation**
  The system does not provide direct communication with dermatologists for follow-up or second opinions.
- **Static Hospital Information**
  Hospital data is stored statically in Firestore and is not dynamically updated from official health sources.

Chapter 7

Conclusion & Future Work

7.1Conclusion

This project is primarily designed to improve the early detection and accurate diagnosis of various dermatological (skin-related) diseases by leveraging advanced technological methods, especially deep learning techniques. Early diagnosis is critical because it increases the chances of effective treatment and can significantly minimize the risks associated with delayed medical intervention.

The project's scope extends beyond merely identifying skin conditions. It also emphasizes offering actionable preventive advice. This means that once a condition is detected, users are also provided with practical tips and recommendations on how to manage or prevent the worsening or spread of the disease. This proactive approach is vital for controlling contagious skin diseases and promoting healthier practices among individuals.

By providing a reliable, easy-to-use diagnostic tool, users can quickly get a preliminary diagnosis and take timely actions to seek professional care if necessary. This can lead to faster intervention, better management of symptoms, and an overall reduction in healthcare costs, until visit the nearest doctor.

Moreover, the integration of deep learning models highlights how modern AI (AI) can be effectively applied in the healthcare sector. Deep learning allows the system to learn from a vast amount of skin disease images, improving its diagnostic accuracy over time. By doing so, the project not only offers more accessible healthcare solutions but also contributes to the future vision of technology-driven preventive healthcare. [4]

7.2 Future Work:

In the previous section, the implemented features were stated. In this section the new features that could be added to application, will be introduced, plus additional points.
The main system for applications is related to future work :

1. Expanding the Disease Categories

Adding more skin diseases to the current list to cover a wider range of conditions and improve diagnostic capabilities.

2. Enhancing and Exploring Model Performance

Improving the overall classification performance by refining the training dataset, optimizing preprocessing techniques, and experimenting with alternative model architectures such as Vision Transformers, InceptionV3, and MixNet to identify the most effective solution.

3. Integrating Doctor Consultation Feature

Allowing users to book appointments with dermatologists directly through the app after receiving a diagnosis.

4. Diagnosis History Log

Storing each user's diagnosis history for future reference and follow-up, enabling better health tracking.

5. Offline Mode

Providing access to essential features and previously loaded data even when the user is not connected to the internet.

6. Multilingual Support

Adding support for additional languages such as French and German to make the app accessible to a broader audience.

References

[1]Jagdish, M. J., Gualán Guamangate, S. P., García López, M. A., De La Cruz-Vargas, J. A., & Roque Camacho, M. E. (2022). Advance study of skin diseases detection using image processing methods. *Natural Volatiles & Essential Oils*, *9*(1), 997–1007.

[2] Vakalopoulou, M., Christodoulidis, S., Burgos, N., Colliot, O., & Lepetit, V. (2025). *Deep learning: Basics and convolutional neural networks (CNN).* In O. Colliot (Ed.), *Machine Learning for Brain Disorders* (Chapter 3). Springer.

[3] GVP College of Engineering for Women. (2023). Skin detection using image processing and machine learning. *Research Square*.

[4] Raj, A., & Sharma, P. (2020). An overview of CNN architectures and applications. *International Journal of Computer Science and Information Security*, *18*(3), 55–65

[5] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet https://www.kaggle.com/datasets/subirbiswas19/skin-disease-dataset/data

[6] Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations (ICLR)*.

[7] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). MobileNets: Efficient (SNNs) for mobile vision

applications. *arXiv preprint arXiv:1704.04861*.

[8] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

[9] Flutter Documentation. (n.d.). Retrieved from https://docs.flutter.dev/

[10] Biswas, S. (n.d.). Skin disease dataset. Retrieved from

[11] Meehan, T. (2019). *Pragmatic Flutter*.

[12] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

[13] Chollet, F. (2017). *Deep learning with Python*. Manning Publications.

(14) Biswas, S. (n.d.). Skin disease dataset. Retrieved from https://www.kaggle.com/datasets/subirbiswas19/skin-disease-dataset/data

[15] AlEnezi, N. S. A. (2019). A method of skin detection using image processing and machine learning. *Procedia Computer Science*.

[16] Miola, A. (2020). *Flutter complete reference*.

[17] Flutter Community. (n.d.). Flutter widgets catalog. Retrieved from https://flutter.dev/docs/development/ui/widgets

[18] Firebase. (n.d.). Firebase features and capabilities. Retrieved from https://firebase.google.com/products

[19] Firebase Flutter. (n.d.). Retrieved from https://firebase.flutter.dev/

Chapter 8

Appendices

8.1 Introduction to the AI Model

The AI model used in this project is based on a CNN, specifically the well-known EfficientNetB0 architecture, which is widely recognized for its high performance in image classification tasks. The model was trained on a dataset of labeled skin disease images to learn how to distinguish between eight different dermatological conditions. It includes multiple layers that extract important visual features from images, which are resized to a uniform dimension of 128×128 pixels for consistent input processing. After training, the model can analyze a new image and provide the most likely diagnosis along with the confidence level for each possible class. The model has demonstrated promising results in identifying the targeted skin diseases, making it a useful tool for initial diagnostic support.

8.1.1 Explain Dataset

As previously mentioned, one of the key challenges we initially faced was the limited size of the dataset, which originally consisted of only 1,159 images distributed across 8 skin disease categories. This small dataset size posed a significant obstacle to effectively training a deep learning model, as such models typically require large volumes of data to generalize well and avoid overfitting.

To overcome this limitation, we applied data augmentation techniques—such as image rotation, flipping, zooming, and color adjustments—to artificially expand the dataset while preserving the medical relevance of each image. As a result, we successfully increased the dataset size to 4,285 images.

- In this dataset image collected .
- The dataset consists of 4285 images categorized into 8 skin diseases.
- The data is divided into two subsets: 80% (4052 images) for training and 20% (233 images) for testing.

Disease	Training Images	Testing Images	Total Images
Cellulitis	879	33	912
Impetigo	440	20	460
Athlete's Foot	418	32	450
Nail Fungus	516	33	549
Ringworm	450	23	473
Cutaneous Larva Migrans	378	30	408
Chickenpox	493	34	527
Shingles	478	33	511
Total	4052	233	4285

Table 8.1 :Dataset

Figure 8.1 : Sample of Dataset

```python
import cv2
import numpy as np
import imgaug.augmenters as iaa
import os
from glob import glob
input_folder = r"C:\Users\CS\Downloads\skin-disease-datasaet\train_set\FU-ringworm"
output_folder = r"C:\Users\CS\Downloads\skin-disease-datasaet\train_set\New folder"
os.makedirs(output_folder, exist_ok=True)
image_paths = glob(os.path.join(input_folder, "*.jpg"))
augmenters = iaa.Sequential([
    iaa.Fliplr(0.5),
    iaa.Flipud(0.3),
    iaa.Affine(rotate=(-30, 30), scale=(0.8, 1.2)),
    iaa.Crop(percent=(0, 0.2)),
    iaa.Grayscale(alpha=(0.0, 1.0)),
    iaa.AddToHueAndSaturation((-20, 20)),
    iaa.LinearContrast((0.8, 1.5)),
    iaa.Affine(rotate=(-20, 20)),
    iaa.GaussianBlur(sigma=(0, 1.5)),
     iaa.Sharpen(alpha=(0, 1.0), lightness=(0.75, 1.5)),
    
])
for img_path in image_paths:
    img = cv2.imread(img_path)
    if img is None:
        print(f" NOT FOUND: {img_path}")
        continue
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    for i in range(3):
        augmented_img = augmenters.augment_image(img)
        img_name = os.path.basename(img_path)
        save_path = os.path.join(output_folder, f"aug_{i}_" + img_name)
        cv2.imwrite(save_path, cv2.cvtColor(augmented_img, cv2.COLOR_RGB2BGR))

print("DONE:", output_folder)
```

Figure 8.2 : Data Augmentation

```python
def build_model(learning_rate=0.001):
    base_model = EfficientNetB0(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
    base_model.trainable = False  # تجميد الطبقات الأساسية
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    predictions = Dense(num_classes, activation='softmax')(x)
    model = Model(inputs=base_model.input, outputs=predictions)
    model.compile(optimizer=Adam(learning_rate=learning_rate), loss='categorical_crossentropy', metrics=['accuracy'])
    return model
    model = KerasClassifier(
    model=build_model,
    verbose=1
)

param_grid = {
    'model__learning_rate': [0.001, 0.0001, 0.002, 0.0002],
    'batch_size': [32, 64, 128],
    'epochs': [5, 10, 15, 20]
}
best_score = 0
best_params = {}

print("\nStarting Grid Search with the following parameters:\n")

for params in ParameterGrid(param_grid):
    print(f"Training with: learning_rate={params['model__learning_rate']}, batch_size={params['batch_size']}, epochs={params['epochs']}")
    model.set_params(**params)
    model.fit(X_train, y_train_one_hot)
    score = model.score(X_val, y_val_one_hot)
    print(f"Accuracy: {score:.4f}\n")
    if score > best_score:
        best_score = score
        best_params = params
print("\nBest Parameters:", best_params)
print("Best Accuracy Score:", best_score)
```

Figure 8.3: Grid Search

Grid Search is a technique used to find the best combination of hyperparameters for a model , It works by Defining a set of values for each hyperparameter (like learning rate, batch size, number of layers, etc.)

```
Best Parameters: {'batch_size': 64, 'epochs': 20, 'model__learning_rate': 0.0002}
Best Accuracy Score: 0.967940813810111
```

Figure 8.4 : Best Parameters of Grid Search

```
def build_model():
    base_model = EfficientNetB0(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
    base_model.trainable = False

    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    predictions = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=base_model.input, outputs=predictions)
    model.compile(optimizer=Adam(learning_rate=0.0002), loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

Figure 8.5 : Build The Model using EfficientNetB0

```
model = build_model()
history = model.fit(
    X_train, y_train_one_hot,
    validation_data=(X_val, y_val_one_hot),
    epochs=20,
    batch_size=64,
    verbose=1
)
```

Figure 8.6 : Training Model

```
val_loss, val_accuracy = model.evaluate(X_val, y_val_one_hot)
print(f"Validation Accuracy: {val_accuracy:.4f}")
```

Figure 8.7: Evaluating Model

After training, we check the model's performance on the validation set to measure accuracy and see how well it generalizes to unseen data.

```
Final Validation Accuracy: 96.42%
Final Validation Loss: 0.1001
```

Figure 8.8 :Final Performance

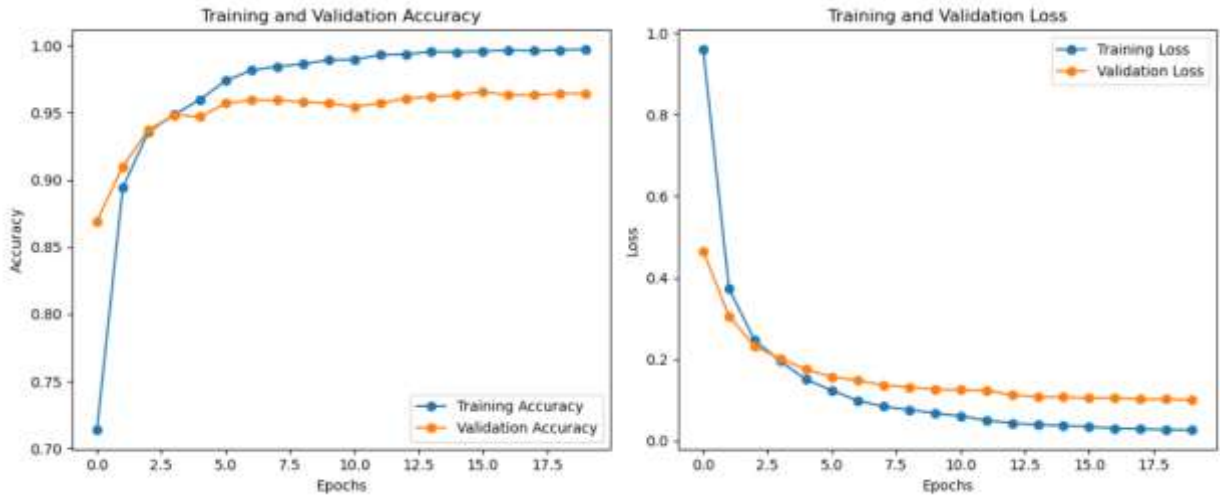


Figure 8,9 : Visualizing Training Performance

The plots show the change in **accuracy** and **loss** over epochs, confirming the model is learning effectively without overfitting.

```python
import cv2
import numpy as np
from tensorflow.keras.applications.efficientnet import preprocess_input
from sklearn.preprocessing import LabelEncoder

test_path = r"C:\Users\pc\Desktop\skin-disease-datasaet\test_set"

real_label = []
predicted_class = []

le = LabelEncoder()
le.fit(y_train)

for folder in sorted(os.listdir(test_path)):
    folder_path = os.path.join(test_path, folder)
    for file in os.listdir(folder_path):
        file_path = os.path.join(folder_path, file)

        img = cv2.imread(file_path)
        img = cv2.resize(img, (224, 224))
        img = np.array([img])
        img = preprocess_input(img)


        predictions = model.predict(img)

        real_label.append(folder)

        predicted_class_index = np.argmax(predictions)
        predicted_class.append(le.classes_[predicted_class_index])  # ح

for real, predicted in zip(real_label[:5], predicted_class[:5]):
    print(f"Real: {real}, Predicted: {predicted}")
```

Figure 8.10 : Prediction on Test Set

This code loads test images, preprocesses them, uses the trained model to predict their classes, and compares predictions with the real labels

```python
y_pred = model.predict(X_val)
y_pred_classes = np.argmax(y_pred, axis=1)
```

Figure 8.11 :Generate Prediction

```
accuracy = accuracy_score(y_val_encoded, y_pred_classes)
precision = precision_score(y_val_encoded, y_pred_classes, average='weighted')
recall = recall_score(y_val_encoded, y_pred_classes, average='weighted')
f1 = f1_score(y_val_encoded, y_pred_classes, average='weighted')


    y_pred_proba = model.predict(X_val)
    y_pred_proba = y_pred_proba
    auc = roc_auc_score(y_val_encoded, y_pred_proba, multi_class='ovr', average='weighted')
```

Figure 8.12: calculate Evaluation Metrics

We calculate accuracy, precision, recall, and F1-score to measure how well the model correctly classifies the validation data.The AUC-ROC score reflects the model's ability to distinguish between classes (the higher, the better).

```
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("AUC-ROC curve:", auc)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=categories, yticklabels=categories)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show
```

Figure 8.13 : code of Matric Results & Confusion Matrix

This section prints the main evaluation metrics (accuracy, precision, recall, F1, and AUC), then visualizes the confusion matrix to show how well the model classified each class.

```
26/26 ━━━━━━━━━━━━━━━━━━━━ 12s 406ms/step
26/26 ━━━━━━━━━━━━━━━━━━━━ 11s 414ms/step
Accuracy: 0.9642416769420469
Precision: 0.9643905624490097
Recall: 0.9642416769420469
F1-score: 0.9640347375777337
AUC-ROC curve: 0.9991392810044847
```

Figure 8.14 : Final Evaluation Metrics on Validation Set

These results show that the model performs very well on the validation set, with high accuracy, precision, recall, F1-score, and an almost perfect AUC-ROC score.

Figure 8.15 :Confusion Matrix Visualization

8.2 Introduction to Flutter

In the following section, I will present an overview of Flutter, the mobile development framework used to build the user interface of the application. This section will cover the main structure of the Flutter project, key functionalities implemented in the app, and how Flutter was used to connect with the backend and provide a smooth user experience. Flutter played a central role in building the app's interactive features, multilingual support, dark/light theme toggling, and seamless communication with Firebase and the diagnostic server.

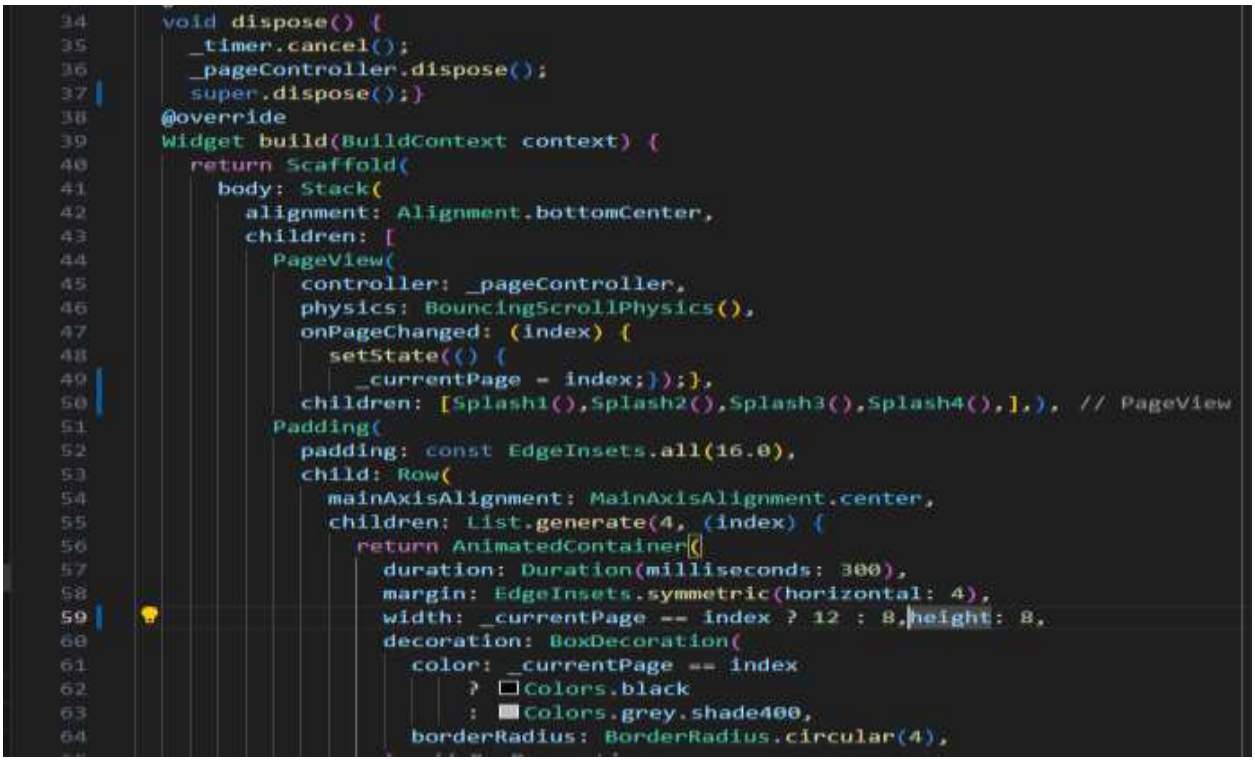
8.2.1 Splash Screen

The application opens with an animated Splash Screen that automatically cycles through four introduction pages. These pages highlight the main features of the app, and after the final screen, the user is directed to the Home Page.

Figure 8.16 : Splash Screen

```dart
import 'dart:async';
import 'package:flutter/material.dart';
import 'package:flutter_application_1/Splach1.dart';
import 'package:flutter_application_1/Splach2.dart';
import 'package:flutter_application_1/Splach3.dart';
import 'package:flutter_application_1/Splach4.dart';
import 'package:flutter_application_1/home.dart';
class PageViewExample extends StatefulWidget {
  @override
  _PageViewExampleState createState() => _PageViewExampleState();
}
class _PageViewExampleState extends State<PageViewExample> {
  PageController _pageController = PageController();
  int _currentPage = 0;
  late Timer _timer;
  @override
  void initState() {
    super.initState();
    _timer = Timer.periodic(Duration(seconds: 3), (Timer timer) {
      if (_currentPage < 4) {
        _currentPage++;
      } else {
        _currentPage = 0;}
      _pageController.animateToPage(
        _currentPage,
        duration: Duration(milliseconds: 300),
        curve: Curves.easeIn,);
      if (_currentPage == 4) {
        _timer.cancel();
        Navigator.pushReplacement(
          context,
          MaterialPageRoute(builder: (context) => SplashScreen()),);} });} // Timer.periodic
  @override
```

Figure 8.17 : Code Splash Screen P1



```dart
  void dispose() {
    _timer.cancel();
    _pageController.dispose();
    super.dispose();}
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Stack(
        alignment: Alignment.bottomCenter,
        children: [
          PageView(
            controller: _pageController,
            physics: BouncingScrollPhysics(),
            onPageChanged: (index) {
              setState(() {
                _currentPage = index;});},
            children: [Splash1(),Splash2(),Splash3(),Splash4(),],), // PageView
          Padding(
            padding: const EdgeInsets.all(16.0),
            child: Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: List.generate(4, (index) {
                return AnimatedContainer(
                  duration: Duration(milliseconds: 300),
                  margin: EdgeInsets.symmetric(horizontal: 4),
                  width: _currentPage == index ? 12 : 8,height: 8,
                  decoration: BoxDecoration(
                    color: _currentPage == index
                        ? Colors.black
                        : Colors.grey.shade400,
                    borderRadius: BorderRadius.circular(4),
```

Figure 8.18 : Code Splash Screen P2

8.2.2 Home Screen(After Splash Screen)

The final Splash Screen displays the app title, logo, and two main options: login and create account. It allows users to proceed to authentication and supports localization using easy_local

Figure 8.19 : Home Screen

```dart
class SplashScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
        body: Stack(
      children: [
        Positioned.fill(
          child: Image.asset(
            'C:/Users/CS/Desktop/flutter_application_1/lib/Image/f.jpg',
            fit: BoxFit.cover,
          ), // Image.asset
        ), // Positioned.fill
        Column(
          mainAxisAlignment: MainAxisAlignment.start,
          children: [
            Padding(
              padding: const EdgeInsets.only(top: 100.0),
              child: Column(
                children: [
                  CircleAvatar(
                    backgroundImage: AssetImage('C:/Users/CS/Desktop/flutter_application_1/lib/Image/t.jpg'),
                    radius: 100,
                  ), // CircleAvatar
                  SizedBox(height: 30),
                  Text(
                    tr('skin_disease'),
                    style: TextStyle(
                      color: Color.fromARGB(255, 194, 83, 83),
                      fontSize: 45,
                      fontWeight: FontWeight.bold, ),),],),), // TextStyle // Text // Column // Padding
            SizedBox(height: 100),
            Expanded(
              child: Padding(
```

Figure 8.20 : Code Home Screen P1

```
Expanded(
  child: Padding(
    padding: const EdgeInsets.all(10.0),
    child: Column(
      mainAxisAlignment: MainAxisAlignment.start,
      children: [
        Center(
          child: Container(
            width: 220,
            height: 50,
            margin: EdgeInsets.symmetric(vertical: 10),
            child: MaterialButton(
              onPressed: () {
                Navigator.push(
                  context,
                  MaterialPageRoute(
                    builder: (context) => login(),),);}, // MaterialPageRoute
              child: Text(
                tr('login'),
                style: TextStyle(
                  color: Colors.white,
                  fontWeight: FontWeight.bold,
                ),), // TextStyle // Text
              color: Color.fromARGB(255, 18, 146, 125),
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(30),),),), // RoundedRecta
        Row(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(
              tr('no_account'),
```

Figure 8.21 : Code Home Screen P2

```
        children: [
          Text(
            tr('no_account'),
            style: TextStyle(
              color: Color.fromRGBO(0, 6, 6, 1),
            ), // TextStyle
          ), // Text
          TextButton(
            onPressed: () {
              Navigator.push(
                context,
                MaterialPageRoute(
                  builder: (context) => SignUp(),
                ), // MaterialPageRoute
              );
            },
            child: Text(
              tr('create_account'),
              style: TextStyle(
                color: Color.fromARGB(255, 11, 170, 143),
                fontWeight: FontWeight.bold,
              ), // TextStyle
            ), // Text
          ), // TextButton
        ],
      ), // Row
```

Figure 8.22 : Code Home Screen P3

8.2.3 Login Screen

The login screen allows users to securely sign in using their email and password. It includes input validation, password visibility toggle, and error handling for incorrect credentials.

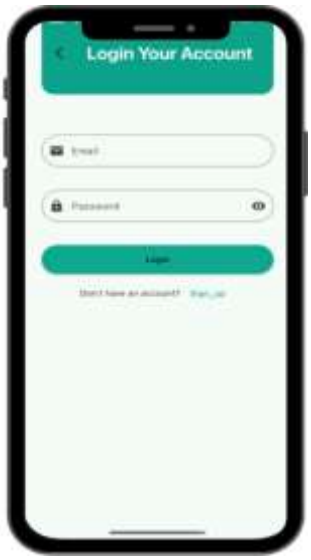


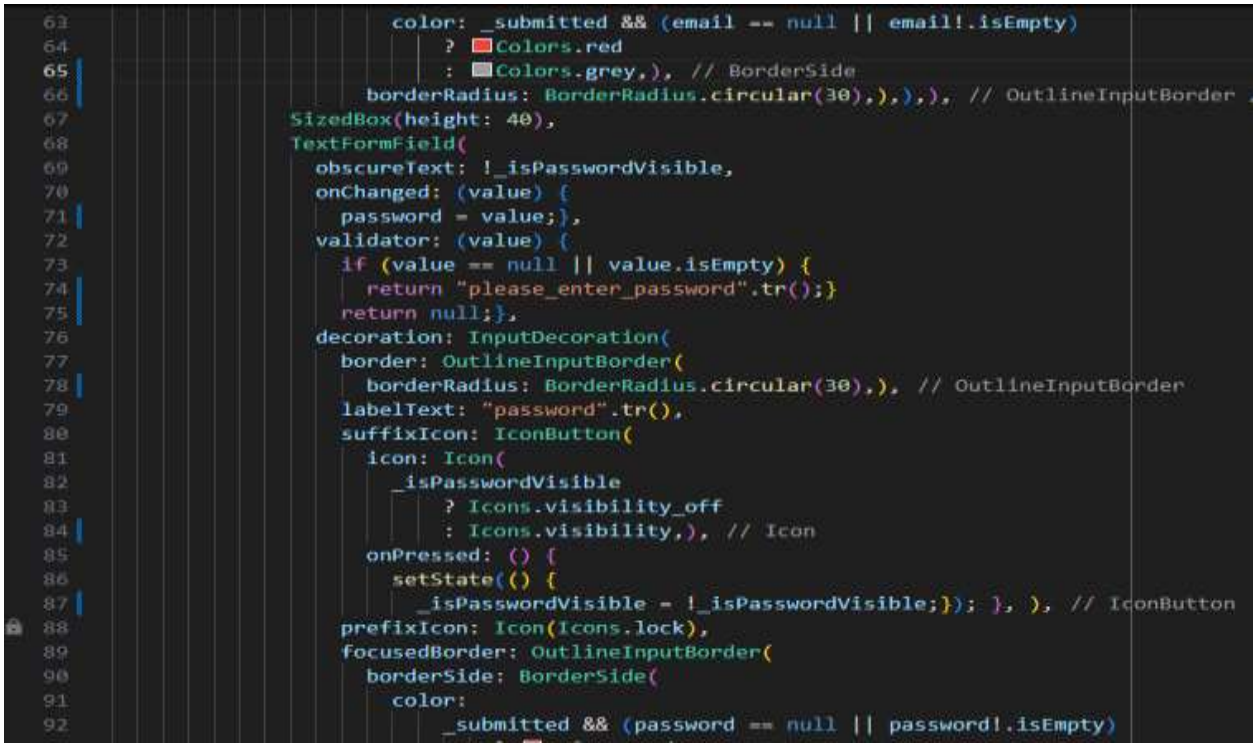
Figure 8.23 : Login Screen

```dart
class login extends StatefulWidget {
  const login({super.key});
  @override
  State<login> createState() => _loginState();}
class _loginState extends State<login> {
  String? email;
  String? password;
  var formKey = GlobalKey<FormState>();
  bool _isPasswordVisible = false;
  bool _submitted = false;
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Theme.of(context).colorScheme.background,
      appBar: PreferredSize(
        preferredSize: Size.fromHeight(100),
        child: Padding(
          padding: const EdgeInsets.symmetric(horizontal: 30),
          child: AppBar(
            title: Text(
              "login_your_account".tr(),
              style: TextStyle(
                fontSize: 30,
                fontWeight: FontWeight.bold,
                color: Colors.white,),), // TextStyle // Text
            centerTitle: true,
            backgroundColor: Color.fromARGB(255, 11, 170, 143),
            shape: RoundedRectangleBorder(
```

Figure 8.24 : Code Login Screen P1

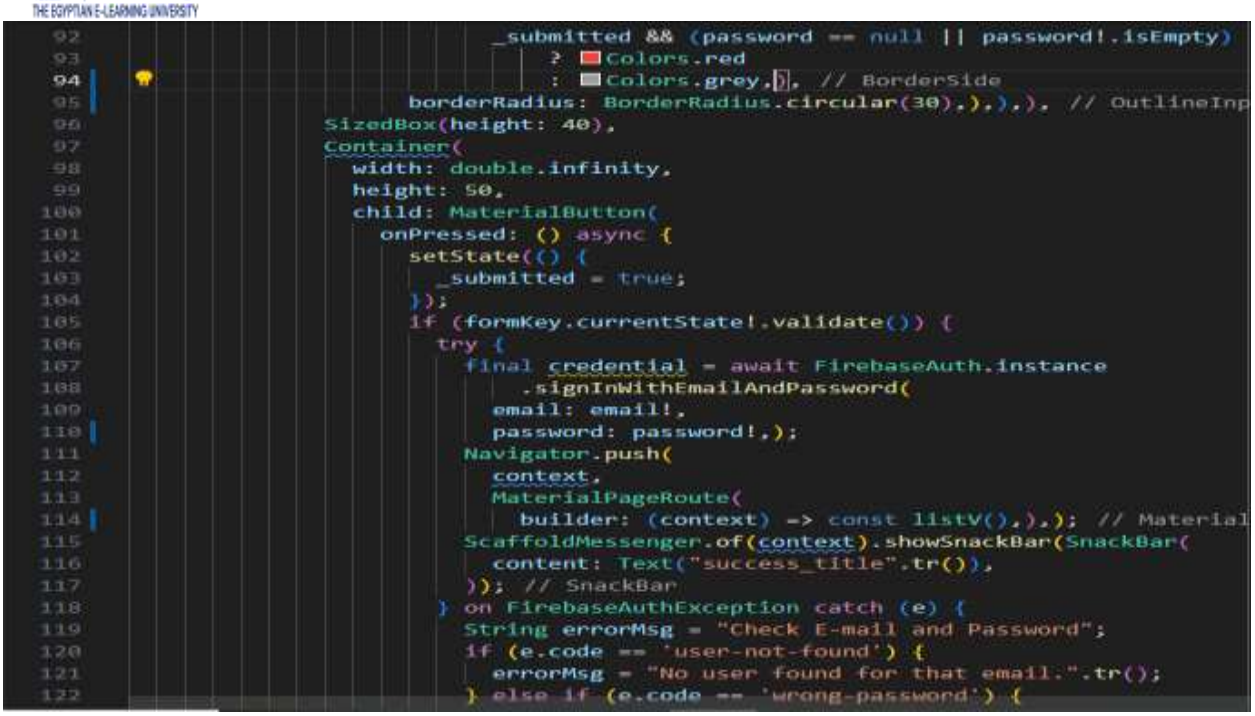
```
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.vertical(bottom: Radius.circular(20)),),
            elevation: 4,),),), // AppBar // Padding // PreferredSize
        body: Form(
          key: formKey,
          autovalidateMode: _submitted
              ? AutovalidateMode.onUserInteraction
              : AutovalidateMode.disabled,
          child: Padding(
            padding: const EdgeInsets.all(30.0),
            child: Column(
              mainAxisAlignment: MainAxisAlignment.start,
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                SizedBox(height: 40),
                TextFormField(
                  onChanged: (value) {
                    email = value;},
                  validator: (value) {
                    if (value == null || value.isEmpty) {
                      return "please_enter_email".tr();}
                    return null; },
                  decoration: InputDecoration(
                    border: OutlineInputBorder(
                      borderRadius: BorderRadius.circular(30),
                    ), // OutlineInputBorder
                    labelText: "email".tr(),
                    prefixIcon: Icon(Icons.email),
                    focusedBorder: OutlineInputBorder(
                      borderSide: BorderSide(
                        color: _submitted && (email == null || email!.isEmpty)
```

Figure 8.25 : Code Login Screen P2



```
                        color: _submitted && (email == null || email!.isEmpty)
                            ? Colors.red
                            : Colors.grey,), // BorderSide
                      borderRadius: BorderRadius.circular(30),),),), // OutlineInputBorder
                SizedBox(height: 40),
                TextFormField(
                  obscureText: !_isPasswordVisible,
                  onChanged: (value) {
                    password = value;},
                  validator: (value) {
                    if (value == null || value.isEmpty) {
                      return "please_enter_password".tr();}
                    return null;},
                  decoration: InputDecoration(
                    border: OutlineInputBorder(
                      borderRadius: BorderRadius.circular(30),), // OutlineInputBorder
                    labelText: "password".tr(),
                    suffixIcon: IconButton(
                      icon: Icon(
                        _isPasswordVisible
                            ? Icons.visibility_off
                            : Icons.visibility,), // Icon
                      onPressed: () {
                        setState(() {
                          _isPasswordVisible = !_isPasswordVisible;}); }, ), // IconButton
                    prefixIcon: Icon(Icons.lock),
                    focusedBorder: OutlineInputBorder(
                      borderSide: BorderSide(
                        color:
                            _submitted && (password == null || password!.isEmpty)
```

Figure 8.26 : Code Login Screen P3



```
                                _submitted && (password == null || password!.isEmpty)
                                    ? Colors.red
                                    : Colors.grey,), // BorderSide
                            borderRadius: BorderRadius.circular(30),),), // OutlineInp
                      SizedBox(height: 40),
                      Container(
                        width: double.infinity,
                        height: 50,
                        child: MaterialButton(
                          onPressed: () async {
                            setState(() {
                              _submitted = true;
                            });
                            if (formKey.currentState!.validate()) {
                              try {
                                final credential = await FirebaseAuth.instance
                                    .signInWithEmailAndPassword(
                                  email: email!,
                                  password: password!,);
                                Navigator.push(
                                  context,
                                  MaterialPageRoute(
                                    builder: (context) => const ListV(),),); // Material
                                ScaffoldMessenger.of(context).showSnackBar(SnackBar(
                                  content: Text("success_title".tr()),
                                )); // SnackBar
                              } on FirebaseAuthException catch (e) {
                                String errorMsg = "Check E-mail and Password";
                                if (e.code == 'user-not-found') {
                                  errorMsg = "No user found for that email.".tr();
                                } else if (e.code == 'wrong-password') {
```

Figure 8.27 : Code Login Screen P4

```
                              )); // SnackBar
                            } on FirebaseAuthException catch (e) {
                              String errorMsg = "Check E-mail and Password";
                              if (e.code == 'user-not-found') {
                                errorMsg = "No user found for that email.".tr();
                              } else if (e.code == 'wrong-password') {
                                errorMsg =
                                    "Wrong password provided for that user.".tr();}
                              ScaffoldMessenger.of(context).showSnackBar(
                                SnackBar(content: Text(errorMsg)),);}) },
                          child: Text(
                            "login".tr(),
                            style: TextStyle(color: Colors.black),), // Text
                          color: Color.fromARGB(255, 11, 170, 143),
                          shape: RoundedRectangleBorder(
                            borderRadius: BorderRadius.circular(30),),), // RoundedRectangleBorde
                      SizedBox(height: 10),
                      Row(
                        mainAxisAlignment: MainAxisAlignment.center,
                        children: [
                          Text("dont_have_account".tr()),
                          TextButton(
                            onPressed: () {
                              Navigator.push(
                                context,
                                MaterialPageRoute(builder: (context) => SignUp()),); },
                            child: Text(
                              "sign_up".tr(),
                              style: TextStyle(
                                color: Color.fromARGB(255, 11, 170, 143),),),),],),),),);}}
```

Figure 8.28 : Code Login Screen P5

8.2.4 Sign Up Screen

This is a "Sign Up" interface built using Flutter and Firebase. When a user enters their details such as username, email, password, date of birth, and gender, the app first validates that all fields are filled in correctly. Once the validation is successful, Firebase Authentication is used to create a new account with the provided email and password. Additional user information like name, date of birth, and gender is then saved in Firestore. If the registration is successful, the user is redirected to the login screen. In case of errors, such as a weak password or an already registered email, an appropriate error message is displayed to the user.

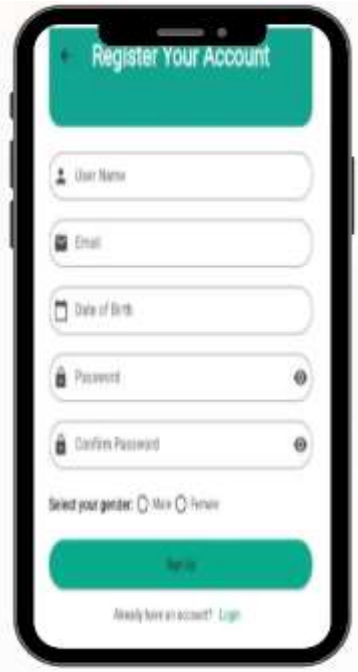


Figure 8.29 : Sign Up Screen

```dart
class SignUp extends StatefulWidget {
  const SignUp({super.key});
  @override
  State<SignUp> createState() => _SignUpState();}
class _SignUpState extends State<SignUp> {
  final TextEditingController _dateController = TextEditingControll
  final GlobalKey<FormState> formkey = GlobalKey<FormState>();
  String? _selectedGender;
  String? email;
  String? password;
  String? confirmPassword;
  String? username;
  bool _showGenderError = false;
  bool _isPasswordVisible = false;
  bool _isConfirmPasswordVisible = false;
  bool _submitted = false;
  @override
  void dispose() {
    _dateController.dispose();
    super.dispose();}
  Future<void> _selectDate(BuildContext context) async {
    final DateTime? pickedDate = await showDatePicker(
      context: context,
      initialDate: DateTime.now(),
      firstDate: DateTime(1900),
      lastDate: DateTime.now(), );
    if (pickedDate != null) {
      setState(() {
```

Figure 8.30 : Code Sign Up Screen P1

```dart
      setState(() {
        _dateController.text =
            "${pickedDate.day}/${pickedDate.month}/${pickedDate.year}";});}}
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: PreferredSize(
        preferredSize: Size.fromHeight(100),
        child: Padding(
          padding: const EdgeInsets.symmetric(horizontal: 30),
          child: AppBar(
            title: Text(
              tr('register_your_account'),
              style: TextStyle(
                fontSize: 30,
                fontWeight: FontWeight.bold,
                color: Colors.white,),), // TextStyle // Text
            backgroundColor: Color.fromARGB(255, 11, 170, 143),
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.vertical(bottom: Radius.circular(20)),),
            elevation: 4,), ),), // AppBar // Padding // PreferredSize
      body: SingleChildScrollView(
        child: Form(
          key: formkey,
          autovalidateMode: _submitted
              ? AutovalidateMode.onUserInteraction
              : AutovalidateMode.disabled,
          child: Padding(
            padding: const EdgeInsets.all(30.0),
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
```

Figure 8.31 : Code Sign Up Screen P2

```
children: [
  SizedBox(height: 20),
  TextFormField(
    onChanged: (value) {
      username = value;},
    validator: (value) {
      if (value!.isEmpty) return tr('please_enter_username');
      return null;},
    decoration: InputDecoration(
      border: OutlineInputBorder(
          borderRadius: BorderRadius.circular(30)), // OutlineInputBorder
      labelText: tr('user_name'),
      prefixIcon: Icon(Icons.person),),), // InputDecoration // TextFormField
  SizedBox(height: 20),
  TextFormField(
    onChanged: (value) {
      email = value;},
    validator: (value) {
      if (value!.isEmpty) return tr('please_enter_email');
      return null;},
    decoration: InputDecoration(
      border: OutlineInputBorder(
          borderRadius: BorderRadius.circular(30)), // OutlineInputBorder
      labelText: tr('email'),
      prefixIcon: Icon(Icons.email), ), ), // InputDecoration // TextFormField
  SizedBox(height: 20),
  TextFormField(
    controller: _dateController,
    readOnly: true,
    validator: (value) {
      if (value!.isEmpty) return tr('please_enter_date_of_birth');
```

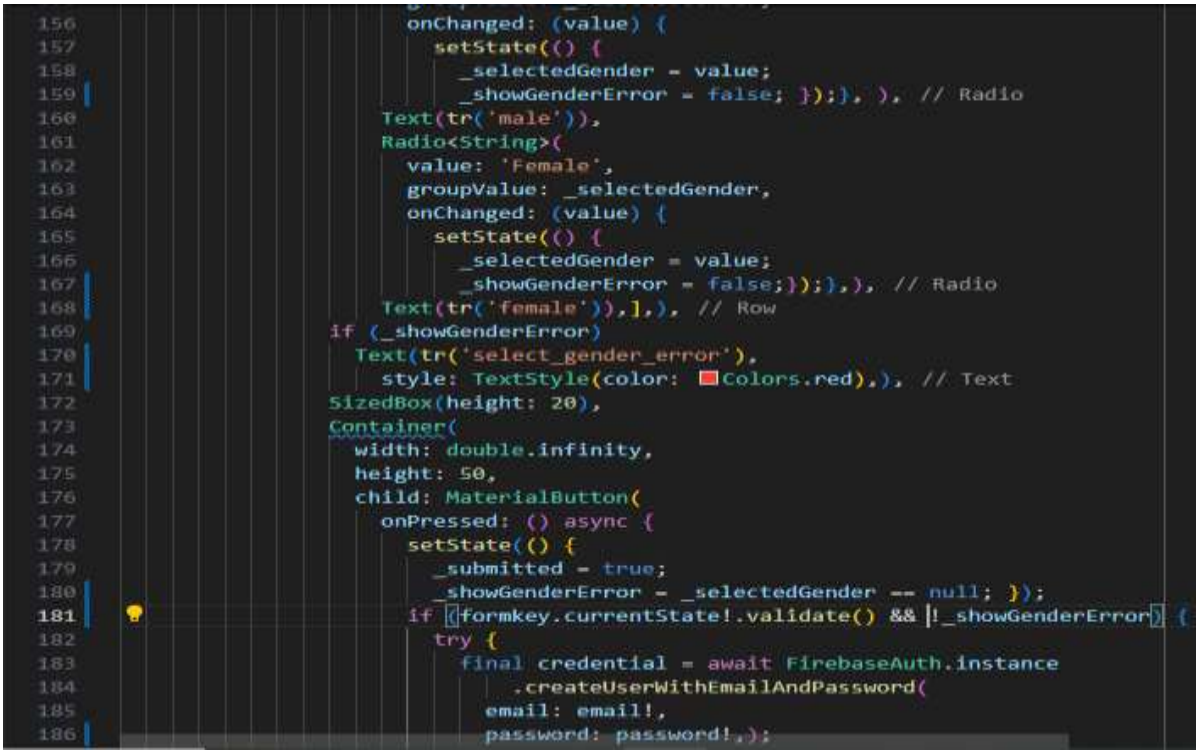
Figure 8.32 : Code Sign Up Screen P3

```
      return null; },
    decoration: InputDecoration(
        border: OutlineInputBorder(
            borderRadius: BorderRadius.circular(30)), // OutlineInputBorde
        labelText: tr('date_of_birth'),
        prefixIcon: Icon(Icons.calendar_today)), // InputDecoration
    onTap: () => _selectDate(context),), // TextFormField
  SizedBox(height: 20),
  TextFormField(
    obscureText: !_isPasswordVisible,
    onChanged: (value) {
      password = value; },
    validator: (value) {
      if (value!.isEmpty) return tr('please_enter_password');
      return null;},
    decoration: InputDecoration(
      border: OutlineInputBorder(
          borderRadius: BorderRadius.circular(30)), // OutlineInputBorder
      labelText: tr('password'),
      suffixIcon: IconButton(
        icon: Icon(
          _isPasswordVisible
              ? Icons.visibility_off
              : Icons.visibility,), // Icon
        onPressed: () {
          setState(() {
            _isPasswordVisible = !_isPasswordVisible;
          });},), // IconButton
      prefixIcon: Icon(Icons.lock),),), // InputDecoration // TextFormFie
  SizedBox(height: 20),
  TextFormField(
```

Figure 8.33 : Code Sign Up Screen P4

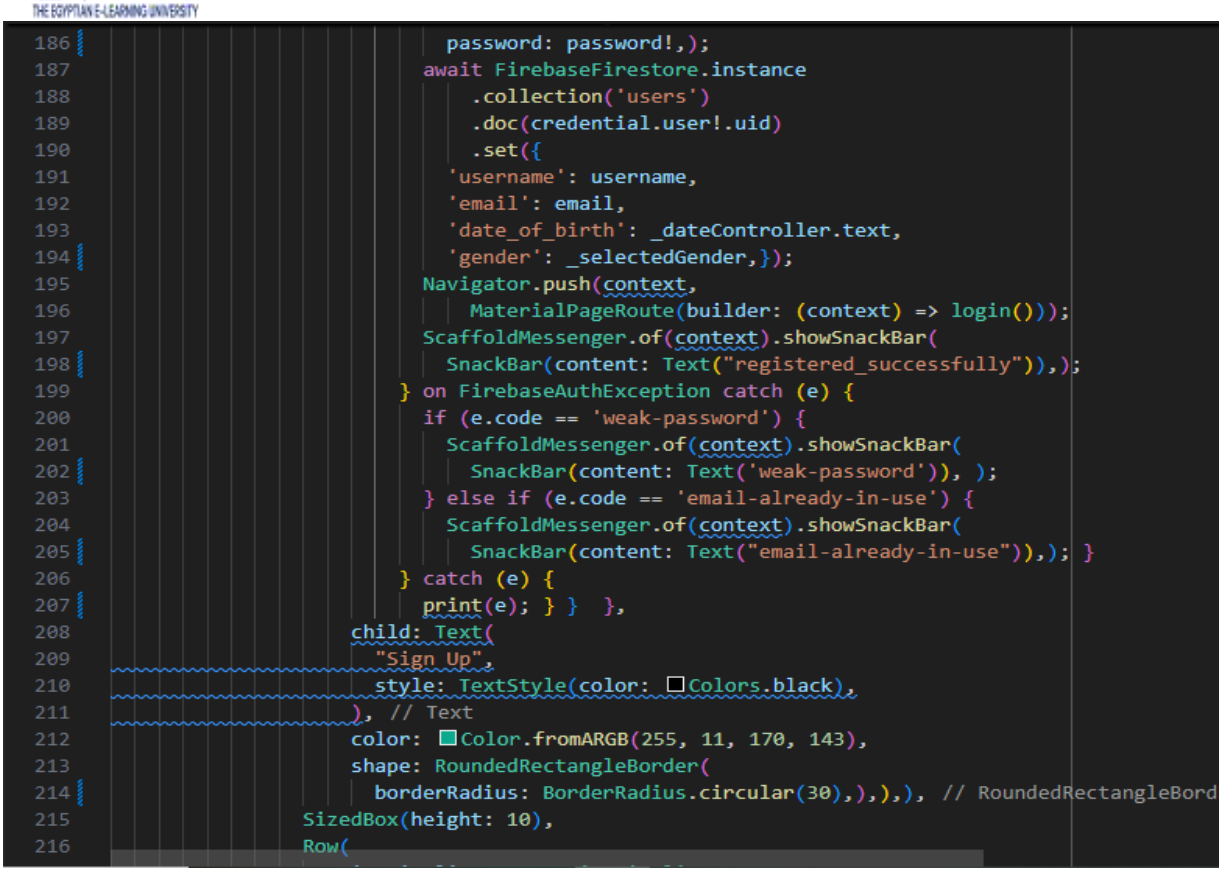
```
TextFormField(
  obscureText: !_isConfirmPasswordVisible,
  onChanged: (value) {
    confirmPassword = value;}, validator: (value) {
    if (value!.isEmpty)return tr('please_enter_confirm_password');
    if (value != password) return tr('passwords_do_not_match');
    return null;},
  decoration: InputDecoration(
    border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(30)), // OutlineInputBorder
    labelText: tr('confirm_password'),
    suffixIcon: IconButton(
      icon: Icon(
        _isConfirmPasswordVisible
            ? Icons.visibility_off
            : Icons.visibility,), // Icon
      onPressed: () {
        setState(() {
          _isConfirmPasswordVisible =
              !_isConfirmPasswordVisible;});},), // IconButton
    prefixIcon: Icon(Icons.lock),),), // InputDecoration // TextFormField
SizedBox(height: 20),
Row(
  children: [
    Text(
      tr('select_gender'),
      style:
          TextStyle(fontSize: 16, fontWeight: FontWeight.bold),), // Text
    Radio<String>(
      value: 'Male',
      groupValue: _selectedGender,
```

Figure 8.34 : Code Sign Up Screen P5



```
      onChanged: (value) {
        setState(() {
          _selectedGender = value;
          _showGenderError = false; });}, ), // Radio
    Text(tr('male')),
    Radio<String>(
      value: 'Female',
      groupValue: _selectedGender,
      onChanged: (value) {
        setState(() {
          _selectedGender = value;
          _showGenderError = false;});},), // Radio
    Text(tr('female')),],), // Row
  if (_showGenderError)
    Text(tr('select_gender_error'),
      style: TextStyle(color: Colors.red),), // Text
  SizedBox(height: 20),
  Container(
    width: double.infinity,
    height: 50,
    child: MaterialButton(
      onPressed: () async {
        setState(() {
          _submitted = true;
          _showGenderError = _selectedGender == null; });
        if (formkey.currentState!.validate() && !_showGenderError) {
          try {
            final credential = await FirebaseAuth.instance
                .createUserWithEmailAndPassword(
              email: email!,
              password: password!,);
```

Figure 8.35 : Code Sign Up Screen P6



```
                password: password!,);
              await FirebaseFirestore.instance
                  .collection('users')
                  .doc(credential.user!.uid)
                  .set({
                'username': username,
                'email': email,
                'date_of_birth': _dateController.text,
                'gender': _selectedGender,});
              Navigator.push(context,
                  MaterialPageRoute(builder: (context) => login()));
              ScaffoldMessenger.of(context).showSnackBar(
                SnackBar(content: Text("registered_successfully")),);
            } on FirebaseAuthException catch (e) {
              if (e.code == 'weak-password') {
                ScaffoldMessenger.of(context).showSnackBar(
                  SnackBar(content: Text('weak-password')), );
              } else if (e.code == 'email-already-in-use') {
                ScaffoldMessenger.of(context).showSnackBar(
                  SnackBar(content: Text("email-already-in-use")),); }
            } catch (e) {
              print(e); } }  },
          child: Text(
            "Sign Up",
            style: TextStyle(color: Colors.black),
          ), // Text
          color: Color.fromARGB(255, 11, 170, 143),
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(30),),),), // RoundedRectangleBord
        SizedBox(height: 10),
        Row(
```

Figure 8.36 : Code Sign Up Screen P7

```
        SizedBox(height: 10),
        Row(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(tr('already_have_account')),
            TextButton(
              onPressed: () {
                Navigator.push(context,
                    MaterialPageRoute(builder: (context) => login()));},
              child: Text(
                tr('login'),
                style: TextStyle(
                  color: Color.fromARGB(255, 11, 170, 143), ),),),],),],),),), ),);}} /
```

Figure 8.37 : Code Sign Up Screen P8

8.2.4 The Diseases Screen

This page is accessible after the user successfully logs in or signs in to the application. It provides an overview of eight common skin conditions, each displayed with an image, name, and brief description. Users can search for diseases by name, making it easy to find specific conditions. Tapping on "Read More" takes the user to a detailed page for each disease

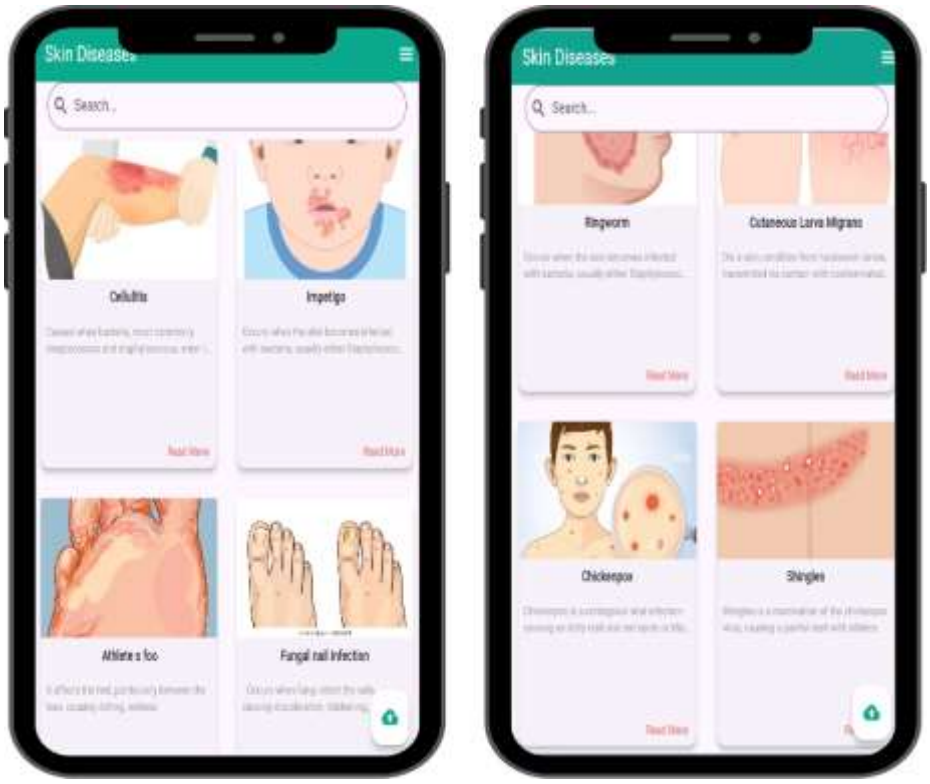


Figure 8.38 : The Diseases Screen

```dart
class listV extends StatefulWidget {
  const listV({super.key});
  @override
  State<listV> createState() => _listVState();}
class _listVState extends State<listV> {
  List<ProductModel> diseases = [
    ProductModel(
      productName: 'cellulitis_name',
      image: 'C:/Users/CS/Desktop/flutter_application_1/lib/Image/2.3.jpg',
      description: 'cellulitis_description',),
    ProductModel(
      productName: 'impetigo_name',
      image: 'C:/Users/CS/Desktop/flutter_application_1/lib/Image/2.webp',
      description: 'impetigo_description',),
    ProductModel(
      productName: 'athletesfoot_name',
      image: 'C:/Users/CS/Desktop/flutter_application_1/lib/Image/4.jpg',
      description: 'athletesfoot_description',),
    ProductModel(
      productName: 'fungalnail_name',
```

Figure 8.39 : Code Diseases Screen P1

```dart
    ProductModel(
      productName: 'fungalnail_name',
      image: 'C:/Users/CS/Desktop/flutter_application_1/lib/Image/35.webp',
      description: 'nail_fungus_description',),
    ProductModel(
      productName: 'ringworm_name',
      image: 'C:/Users/CS/Desktop/flutter_application_1/lib/Image/5.webp',
      description: 'ringworm_description', ),
    ProductModel(
      productName: 'cutaneouslarva_name',
      image: 'C:/Users/CS/Desktop/flutter_application_1/lib/Image/5.PNG',
      description: 'cutaneouslarva_description',),
    ProductModel(
      productName: 'chickenpox_name',
      image: 'C:/Users/CS/Desktop/flutter_application_1/lib/Image/6.PNG',
      description: 'chickenpox_description',),
    ProductModel(
      productName: 'shingles_name',
      image: 'C:/Users/CS/Desktop/flutter_application_1/lib/Image/7.PNG',
      description: 'shingles_description',),];
  String searchQuery = '';
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Theme.of(context).scaffoldBackgroundColor,
      appBar: AppBar(
        title: Text(
          'skin_diseases_title'.tr(),
          style: TextStyle(
            color: Theme.of(context).brightness == Brightness.dark
                ? Colors.white
```

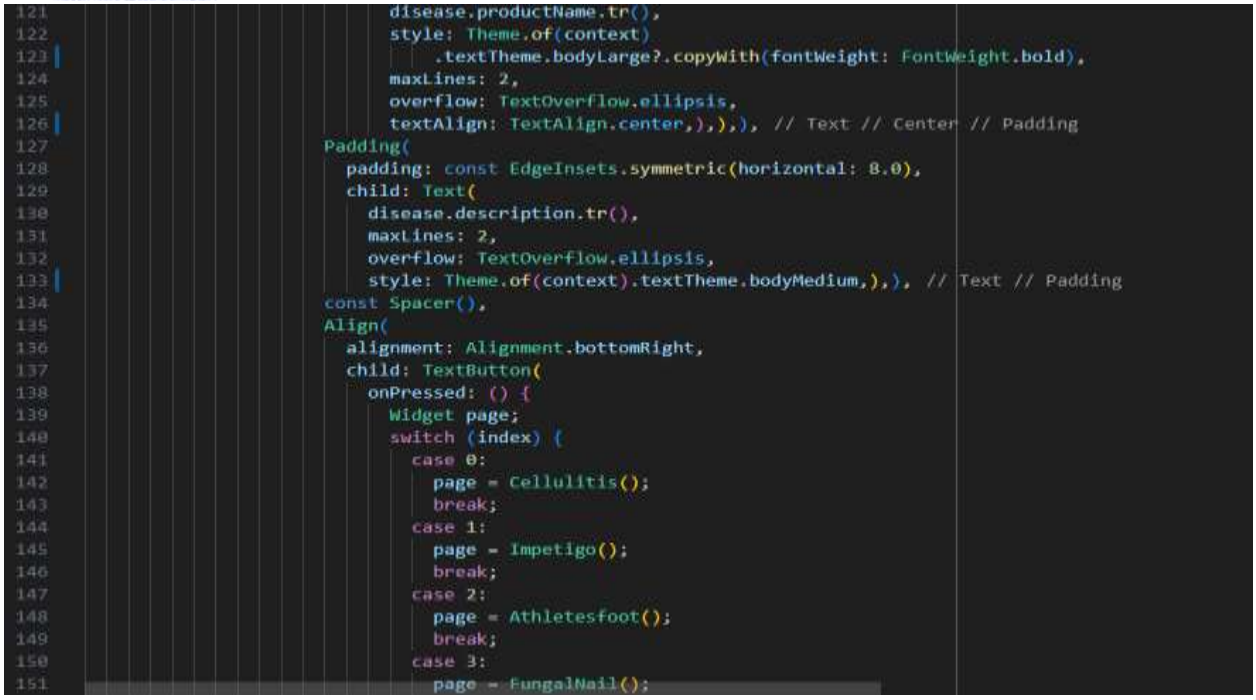
Figure 8.40 : Code Diseases Screen P2

```
            color: Theme.of(context).brightness == Brightness.dark
                ? Colors.white
                : Colors.black,),), // TextStyle // Text
        iconTheme: IconThemeData(
          color: Theme.of(context).brightness == Brightness.dark
              ? Colors.white
              : Colors.black,), // IconThemeData
        backgroundColor: const Color.fromARGB(255, 11, 170, 143),), // AppBar
      endDrawer: const Menu(),
      body: Column(
        children: [
          Padding(
            padding: const EdgeInsets.symmetric(horizontal: 20.0, vertical: 10),
            child: TextField(
              decoration: InputDecoration(
                prefixIcon: const Icon(Icons.search),
                hintText: 'search_hint'.tr(),
                hintStyle: TextStyle(color: Theme.of(context).hintColor),
                border: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(25),
                  borderSide: BorderSide(color: Theme.of(context).dividerColor),),), //
              onChanged: (value) {
                setState(() {
                  searchQuery = value.toLowerCase();});},),), // TextField // Padding
          Expanded(
            child: GridView.builder(
              padding: EdgeInsets.all(10),
              gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
                crossAxisCount: 2,
                crossAxisSpacing: 10,
```

Figure 8.41 : Code Diseases Screen P3

```
                crossAxisSpacing: 10,
                mainAxisSpacing: 10,
                childAspectRatio: 0.8,), // SliverGridDelegateWithFixedCrossAxisCount
              itemCount: diseases
                  .where((disease) => disease.productName .tr() .toLowerCase()
                      .contains(searchQuery)).toList().length,
              itemBuilder: (context, index) {
                var filteredDiseases = diseases
                    .where((disease) => disease.productName.tr().toLowerCase()
                    .contains(searchQuery)).toList();
                ProductModel disease = filteredDiseases[index];
                return Card(
                  color: Theme.of(context).cardColor,
                  elevation: 4,
                  shape: RoundedRectangleBorder(
                    borderRadius: BorderRadius.circular(15), ), // RoundedRectangleBorder
                  child: Column(
                    crossAxisAlignment: CrossAxisAlignment.start,
                    children: [
                      ClipRRect(
                        borderRadius:
                            BorderRadius.vertical(top: Radius.circular(15)),
                        child: Image.asset(
                          disease.image,
                          fit: BoxFit.cover,
                          height: 140,
                          width: double.infinity, ),), // Image.asset // ClipRRect
                      Padding(
                        padding: const EdgeInsets.all(8.0),
                        child: Center(
                          child: Text(
```

Figure 8.42 : Code Diseases Screen P4



```
                        disease.productName.tr(),
                        style: Theme.of(context)
                            .textTheme.bodyLarge?.copyWith(fontWeight: FontWeight.bold),
                        maxLines: 2,
                        overflow: TextOverflow.ellipsis,
                        textAlign: TextAlign.center,),),), // Text // Center // Padding
                    Padding(
                      padding: const EdgeInsets.symmetric(horizontal: 8.0),
                      child: Text(
                        disease.description.tr(),
                        maxLines: 2,
                        overflow: TextOverflow.ellipsis,
                        style: Theme.of(context).textTheme.bodyMedium,),), // Text // Padding
                    const Spacer(),
                    Align(
                      alignment: Alignment.bottomRight,
                      child: TextButton(
                        onPressed: () {
                          Widget page;
                          switch (index) {
                            case 0:
                              page = Cellulitis();
                              break;
                            case 1:
                              page = Impetigo();
                              break;
                            case 2:
                              page = Athletesfoot();
                              break;
                            case 3:
                              page = FungalNail();
```

Figure 8.43 : Code Diseases Screen P5

```
                              page = FungalNail();
                              break;
                            case 4:
                              page = Ringworm();
                              break;
                            case 5:
                              page = Cutaneous();
                              break;
                            case 6:
                              page = Chickenpox();
                              break;
                            case 7:
                              page = Shingles();
                              break;
                            default:
                              page = Scaffold(
                                appBar: AppBar(title: Text("Details")),
                                body: Center(
                                    child: Text("No details available.")),);} // Center // Scaffold
                          Navigator.push(
                            context,
                            MaterialPageRoute(builder: (context) => page),);},
                        child: Text(
                          'read_more'.tr(),
                          style: TextStyle(
                            fontSize: 12.5,
                            color: Color.fromARGB(255, 225, 102, 102),),),),),),],), // T
          floatingActionButton: FloatingActionButton(
            backgroundColor: Color.fromARGB(255, 11, 170, 143),
            onPressed: () {
```

Figure 8.44 : Code Diseases Screen P6

```
onPressed: () {
  Navigator.push(
    context,
    MaterialPageRoute(builder: (context) => upload()),); },
child: const Icon(Icons.cloud_upload, color: Colors.white),),);}}
```

Figure 8.45 : Code Diseases Screen P7

8.2.5 Disease Detail Screen

A unified template was developed using a custom widget called DiseasePageTemplate to display information about skin diseases. This template presents the disease name, image, and distinct sections for symptoms, causes, and prevention. It supports both dark and light themes, as well as multilingual translations. The template is applied to individual disease pages, by passing specific keys, image paths, and descriptions for each condition. This approach was implemented for 8 different skin diseases to ensure a consistent, organized, and user-friendly presentation.

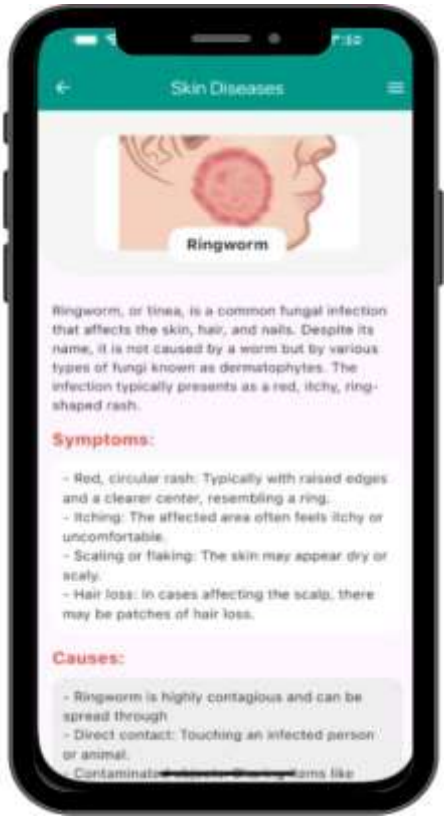


Figure 8.46 : Disease Detail Screen

```dart
class DiseasePageTemplate extends StatelessWidget {
  final String titleKey;
  final String imagePath;
  final String symptomsKey;
  final String causesKey;
  final String preventionKey;
  const DiseasePageTemplate({
    super.key,
    required this.titleKey,
    required this.imagePath,
    required this.symptomsKey,
    required this.causesKey,
    required this.preventionKey,
  });
  @override
  Widget build(BuildContext context) {
    final isDark = Theme.of(context).brightness == Brightness.dark;
    final gradient =
        isDark ? DarkThemeColors.cardGradient : LightThemeColors.cardGradient;
    final textColor =
        isDark ? DarkThemeColors.textColor : LightThemeColors.textColor;
    final surfaceColor = Theme.of(context).colorScheme.surface;
    return Scaffold(
      appBar: AppBar(
        title: Text(titleKey.tr()),
        backgroundColor: isDark ? null : Colors.teal,
        flexibleSpace: isDark
            ? Container(decoration: BoxDecoration(gradient: gradient))
```

Figure 8.46 : Code DiseasePageTemplate p1

```dart
            : null,
        leading: BackButton(),), // AppBar
      endDrawer: Menu(),
      body: SingleChildScrollView(
        child: Column(
          children: [
            Stack(
              alignment: Alignment.bottomCenter,
              children: [
                Container(
                  height: 200,
                  decoration: BoxDecoration(
                    color: Theme.of(context).cardColor,
                    borderRadius: const BorderRadius.only(
                      bottomLeft: Radius.circular(50),
                      bottomRight: Radius.circular(50),),), // BorderRadius.only
                  child: Center(
                    child: ClipRRect(
                      borderRadius: BorderRadius.circular(16),
                      child: Image.asset(
                        imagePath,
                        height: 140,
                        width: 300,
                        fit: BoxFit.cover,),),),), // Image.asset // ClipRRect /
                Positioned(
                  bottom: 16,
                  child: Container(
                    padding:
                        const EdgeInsets.symmetric(horizontal: 20, vertical: 8),
                    decoration: BoxDecoration(
                      gradient: gradient,
```

Figure 8.47 : Code DiseasePageTemplate p2

```
                gradient: gradient,
                borderRadius: BorderRadius.circular(16), ), // BoxDecoration
              child: Text(
                titleKey.tr(),
                style: TextStyle(
                  color: textColor,
                  fontSize: 18,
                  fontWeight: FontWeight.bold,),),),)],), // TextStyle // Text //
const SizedBox(height: 16),
Padding(
  padding: const EdgeInsets.all(16.0),
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(
        tr('Symptoms'),
        style: const TextStyle(
          fontSize: 20,
          fontWeight: FontWeight.bold,
          color: Colors.red, ),), // TextStyle // Text
      const SizedBox(height: 8),
      Container(
        padding: const EdgeInsets.all(12),
        decoration: BoxDecoration(
          color: surfaceColor,
          borderRadius: BorderRadius.circular(16),), // BoxDecoration
        child: Text(
          symptomsKey.tr(),
          style: const TextStyle(fontSize: 16),),), // Text // Container
      const SizedBox(height: 16),
      Text(
```

Figure 8.48 : Code DiseasePageTemplate p3

```
        tr('Causes'),
        style: const TextStyle(
          fontSize: 20,
          fontWeight: FontWeight.bold,
          color: Colors.red,),), // TextStyle // Text
      const SizedBox(height: 8),
      Container(
        padding: const EdgeInsets.all(12),
        decoration: BoxDecoration(
          color: surfaceColor,
          borderRadius: BorderRadius.circular(16), ), // BoxDecoration
        child: Text(
          causesKey.tr(),
          style: const TextStyle(fontSize: 16),),), // Text // Contain
      const SizedBox(height: 16),
      Text(
        tr('Prevention'),
        style: const TextStyle(
          fontSize: 20,
          fontWeight: FontWeight.bold,
          color: Colors.red,),), // TextStyle // Text
      const SizedBox(height: 8),
      Container(
        padding: const EdgeInsets.all(12),
        decoration: BoxDecoration(
          color: surfaceColor,
          borderRadius: BorderRadius.circular(16),), // BoxDecoration
        child: Text(
          preventionKey.tr(),
          style: const TextStyle(fontSize: 16),),),],), )],), ),);}}
```

Figure 8.49 : Code DiseasePageTemplate p4

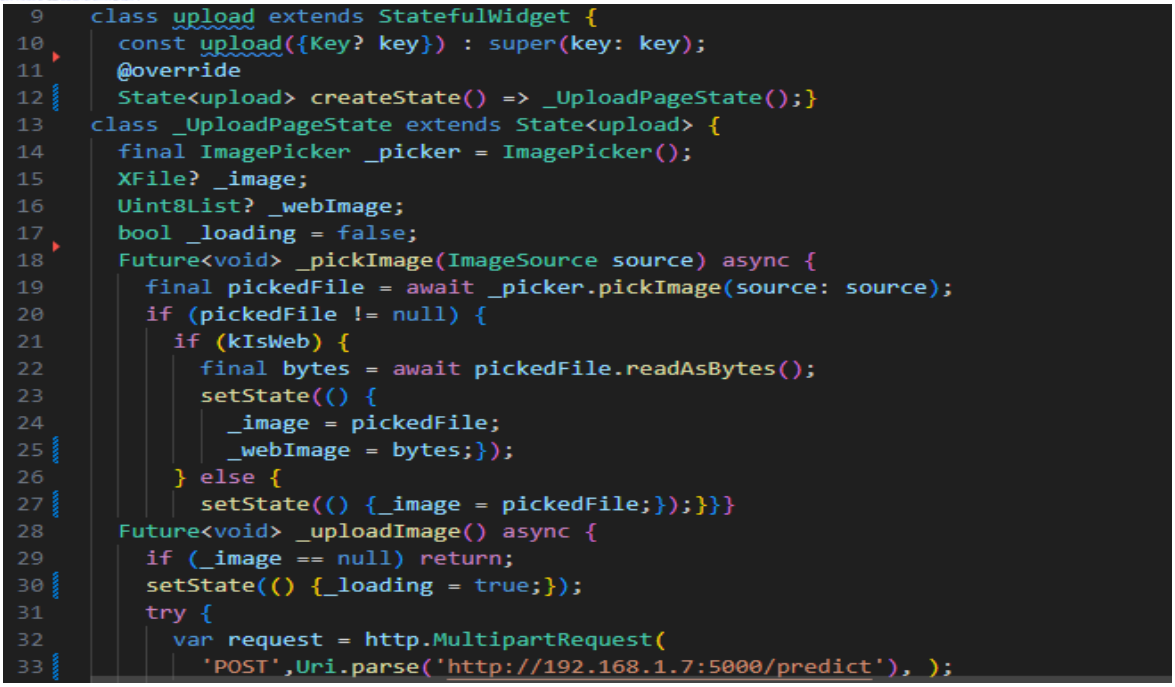
```dart
import 'package:flutter/material.dart';
import 'package:flutter_application_1/disease_template.dart';

class Cellulitis extends StatelessWidget {
  const Cellulitis({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return DiseasePageTemplate(
      titleKey: 'cellulitis_name',
      imagePath: 'C:/Users/CS/Desktop/flutter_application_1/lib/Image/2.3.jpg',
      symptomsKey: 'cellulitis_symptoms',
      causesKey: 'cellulitis_causes',
      preventionKey: 'cellulitis_prevention',
    );
  }
```

Figure 8.50 :Code Disease Detail Screen

8.2.6 Image Upload Screen

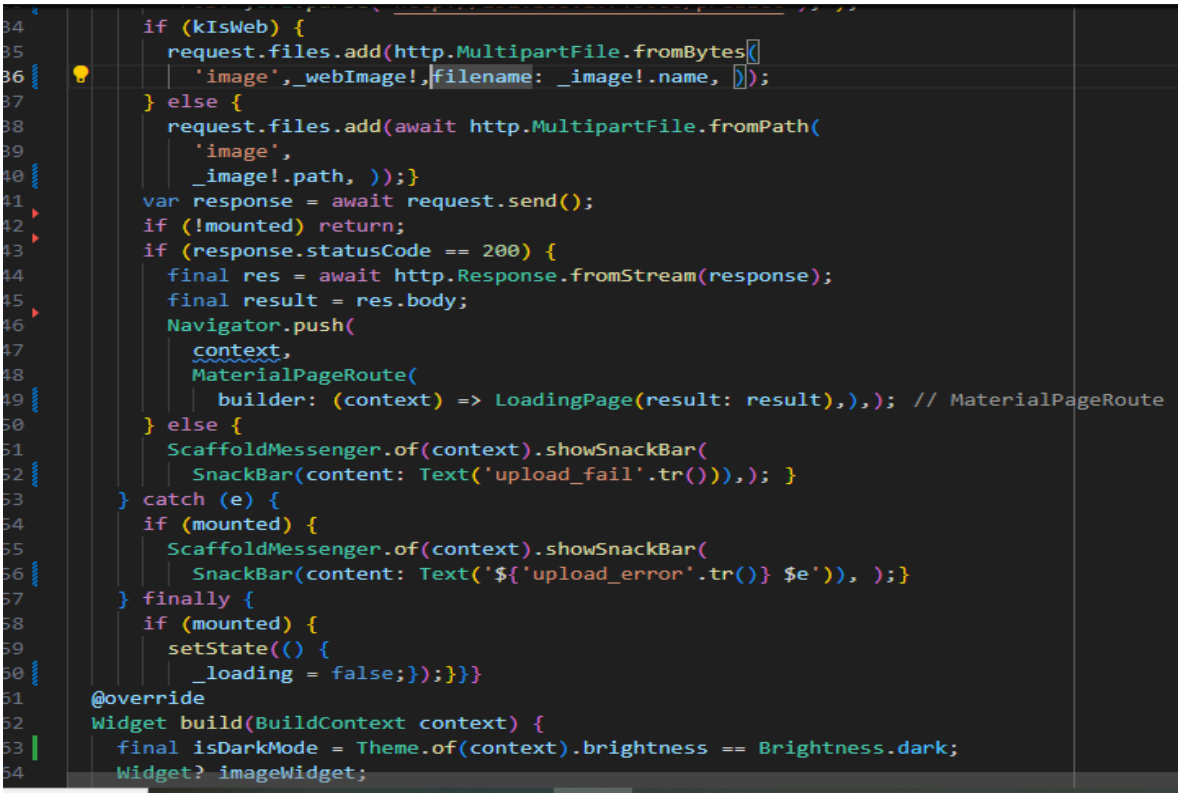
This page allows users to upload an image of a skin condition using either the camera or gallery. The uploaded image is then sent to a Flask-based backend to initiate an AI-powered diagnosis that identifies skin conditions

Figure 8.51 : Image Upload Screen



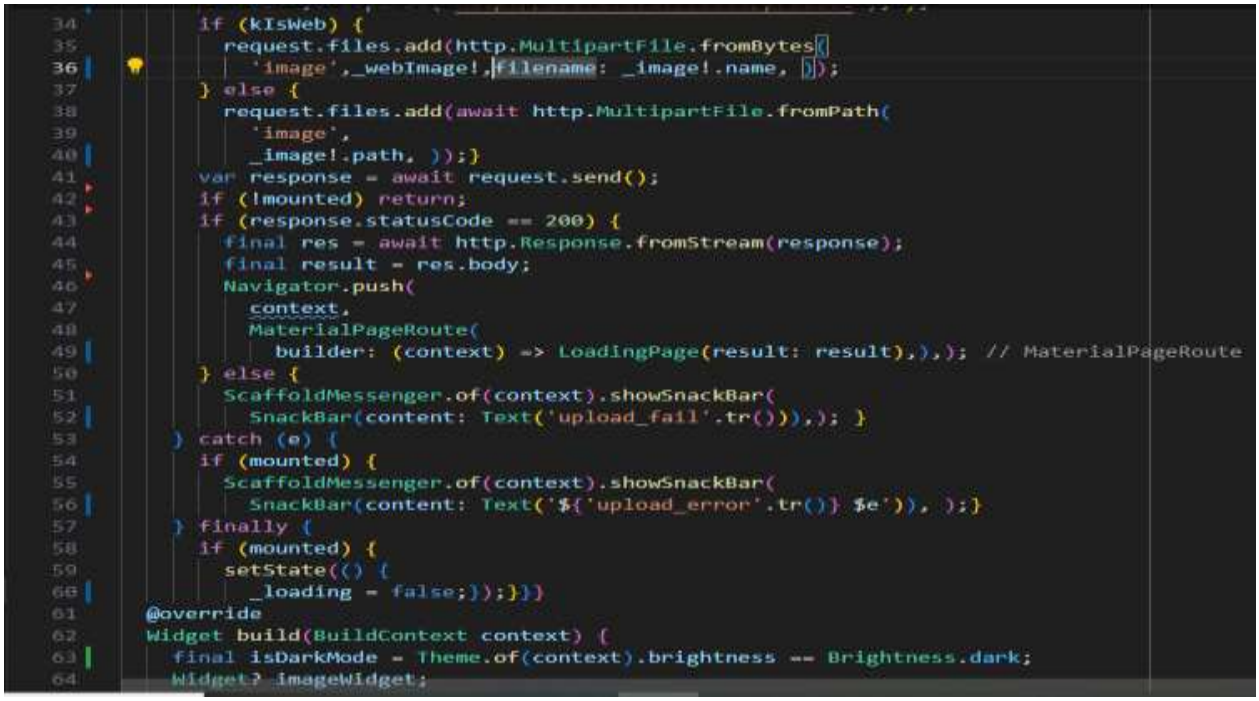
```dart
class upload extends StatefulWidget {
  const upload({Key? key}) : super(key: key);
  @override
  State<upload> createState() => _UploadPageState();}
class _UploadPageState extends State<upload> {
  final ImagePicker _picker = ImagePicker();
  XFile? _image;
  Uint8List? _webImage;
  bool _loading = false;
  Future<void> _pickImage(ImageSource source) async {
    final pickedFile = await _picker.pickImage(source: source);
    if (pickedFile != null) {
      if (kIsWeb) {
        final bytes = await pickedFile.readAsBytes();
        setState(() {
          _image = pickedFile;
          _webImage = bytes;});
      } else {
        setState(() {_image = pickedFile;});}}}
  Future<void> _uploadImage() async {
    if (_image == null) return;
    setState(() {_loading = true;});
    try {
      var request = http.MultipartRequest(
        'POST',Uri.parse('http://192.168.1.7:5000/predict'), );
```

Figure 8.52 : Code Image Upload Screen p1



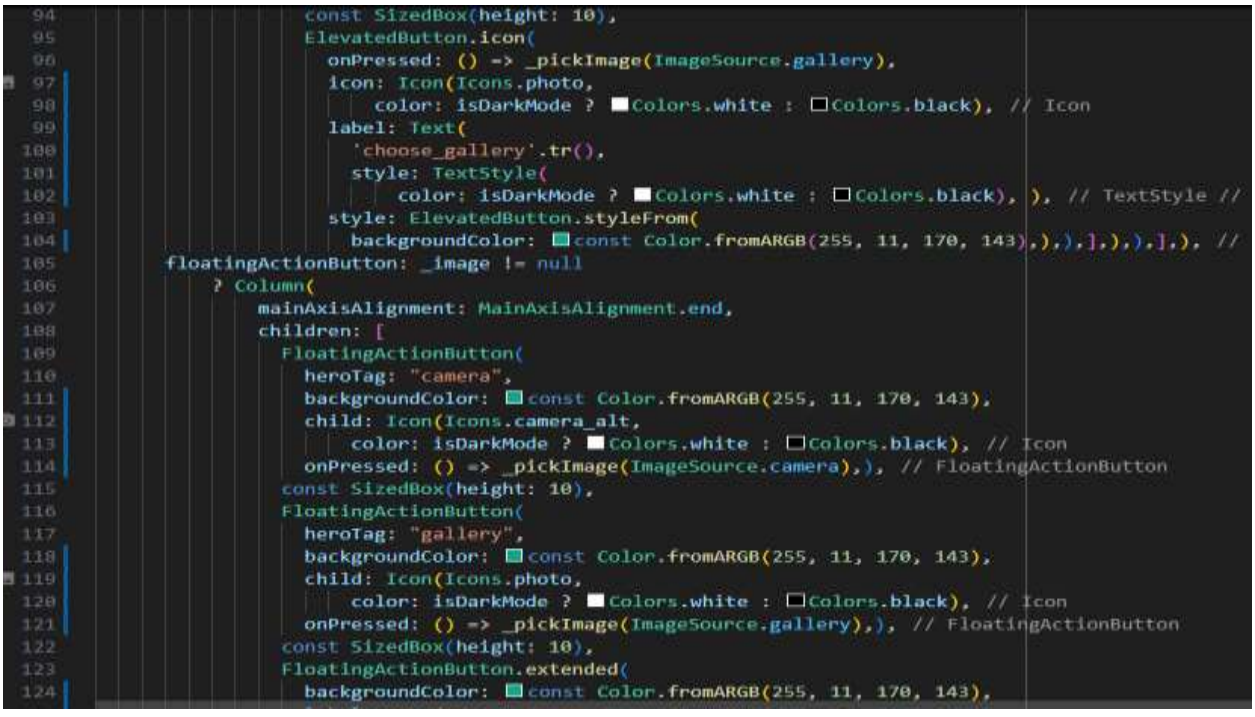
```dart
    if (kIsWeb) {
      request.files.add(http.MultipartFile.fromBytes(
        'image',_webImage!,filename: _image!.name, ));
    } else {
      request.files.add(await http.MultipartFile.fromPath(
        'image',
        _image!.path, ));}
    var response = await request.send();
    if (!mounted) return;
    if (response.statusCode == 200) {
      final res = await http.Response.fromStream(response);
      final result = res.body;
      Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) => LoadingPage(result: result),),); // MaterialPageRoute
    } else {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('upload_fail'.tr())),); }
  } catch (e) {
    if (mounted) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('${'upload_error'.tr()} $e')), );}
  } finally {
    if (mounted) {
      setState(() {
        _loading = false;});}}}
@override
Widget build(BuildContext context) {
  final isDarkMode = Theme.of(context).brightness == Brightness.dark;
  Widget? imageWidget;
```

Figure 8.53 : Code Image Upload Screen p2



```
      if (kIsWeb) {
        request.files.add(http.MultipartFile.fromBytes(
            'image',_webImage!,filename: _image!.name, ));
      } else {
        request.files.add(await http.MultipartFile.fromPath(
          'image',
          _image!.path, ));}
      var response = await request.send();
      if (!mounted) return;
      if (response.statusCode == 200) {
        final res = await http.Response.fromStream(response);
        final result = res.body;
        Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) => LoadingPage(result: result),),); // MaterialPageRoute
      } else {
        ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(content: Text('upload_fail'.tr())),); }
      } catch (e) {
        if (mounted) {
          ScaffoldMessenger.of(context).showSnackBar(
            SnackBar(content: Text('${'upload_error'.tr()} $e')), );}
      } finally {
        if (mounted) {
          setState(() {
            _loading = false;});}}}
  @override
  Widget build(BuildContext context) {
    final isDarkMode = Theme.of(context).brightness == Brightness.dark;
    Widget? imageWidget;
```

Figure 8.54 : Code Image Upload Screen p3



```
                const SizedBox(height: 10),
                ElevatedButton.icon(
                  onPressed: () => _pickImage(ImageSource.gallery),
                  icon: Icon(Icons.photo,
                      color: isDarkMode ? Colors.white : Colors.black), // Icon
                  label: Text(
                    'choose_gallery'.tr(),
                    style: TextStyle(
                        color: isDarkMode ? Colors.white : Colors.black), ), // TextStyle //
                  style: ElevatedButton.styleFrom(
                    backgroundColor: const Color.fromARGB(255, 11, 170, 143),),),],),),],), //
    floatingActionButton: _image != null
        ? Column(
            mainAxisAlignment: MainAxisAlignment.end,
            children: [
              FloatingActionButton(
                heroTag: "camera",
                backgroundColor: const Color.fromARGB(255, 11, 170, 143),
                child: Icon(Icons.camera_alt,
                    color: isDarkMode ? Colors.white : Colors.black), // Icon
                onPressed: () => _pickImage(ImageSource.camera),), // FloatingActionButton
              const SizedBox(height: 10),
              FloatingActionButton(
                heroTag: "gallery",
                backgroundColor: const Color.fromARGB(255, 11, 170, 143),
                child: Icon(Icons.photo,
                    color: isDarkMode ? Colors.white : Colors.black), // Icon
                onPressed: () => _pickImage(ImageSource.gallery),), // FloatingActionButton
              const SizedBox(height: 10),
              FloatingActionButton.extended(
                backgroundColor: const Color.fromARGB(255, 11, 170, 143),
```

Figure 8.55 : Code Image Upload Screen p4

```
124                backgroundColor: const Color.fromARGB(255, 11, 170, 143),
125                label: Text(
126                  'image_analysis'.tr(),
127                  style: TextStyle(
128                      color: isDarkMode ? Colors.white : Colors.black),), //
129                icon: Icon(Icons.send,
130                    color: isDarkMode ? Colors.white : Colors.black), // Icon
131                onPressed: _uploadImage,),],): null, );}} // FloatingActionButton.
132
```

Figure 8.56 : Code Image Upload Screen p5

8.2.7 Loading Screen

This page appears immediately after the user uploads a skin condition image. It simulates an AI diagnosis process by showing a loading indicator until the analysis is completed.

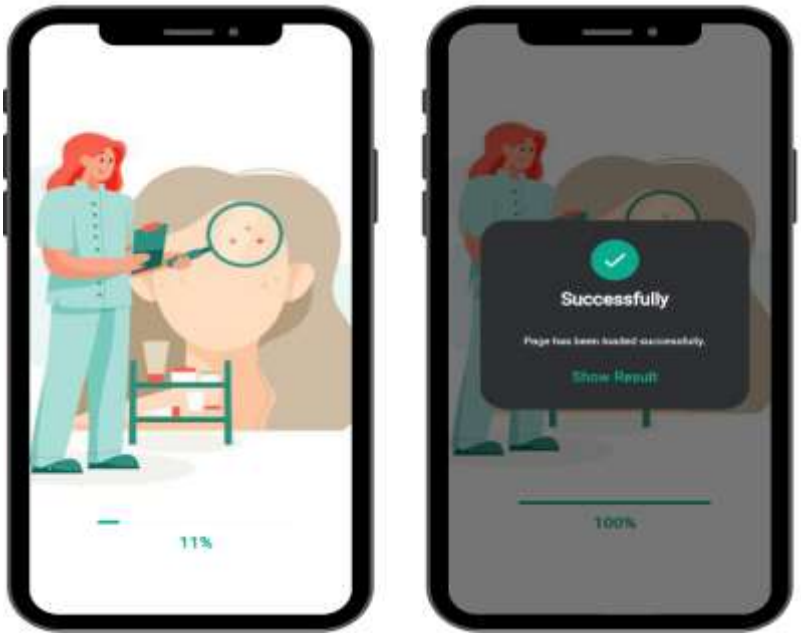


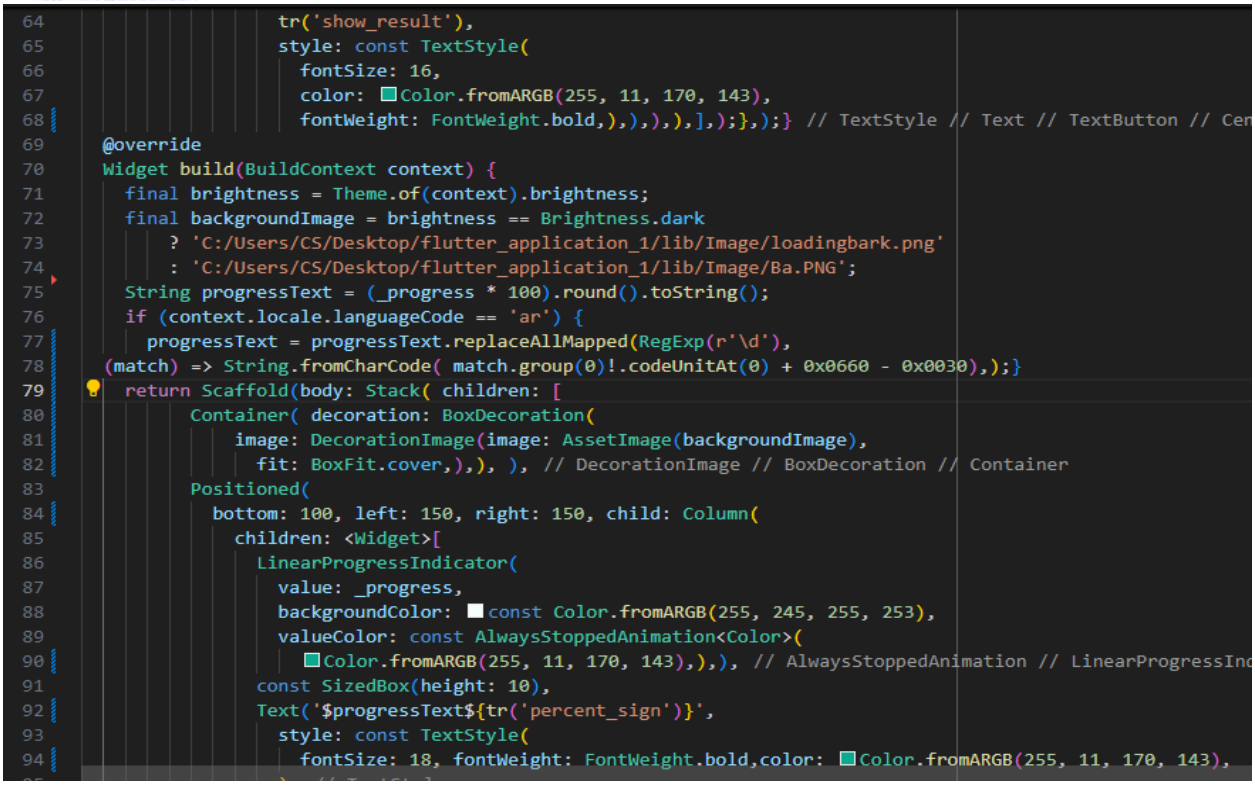
Figure 8.57 : Loading Screen

```dart
class LoadingPage extends StatefulWidget {
  final String result;
  LoadingPage({required this.result});
  @override
  _LoadingPageState createState() => _LoadingPageState();}
class _LoadingPageState extends State<LoadingPage> {
  double _progress = 0;
  @override
  void initState() {
    super.initState();
    _simulateLoading();}
  Future<void> _simulateLoading() async {
    for (int i = 1; i <= 100; i++) {
      await Future.delayed(Duration(milliseconds: 50));
      setState(() {
        _progress = i / 100;});};
    _showCompletionDialog();}
  void _showCompletionDialog() {
    final isDarkMode = Theme.of(context).brightness == Brightness.dark;
    showDialog(
      context: context,
      barrierDismissible: false,
      builder: (BuildContext context) {
        return AlertDialog(
          backgroundColor: isDarkMode ? Colors.black : Colors.white,
          title: Center(
            child: Column(
              mainAxisSize: MainAxisSize.min,
              children: [
                Container(
```

Figure 8.58 : Code Loading Screen p1

```dart
                  width: 50,
                  height: 50,
                  decoration: BoxDecoration(
                    color: const Color.fromARGB(255, 11, 170, 143),
                    shape: BoxShape.circle,), // BoxDecoration
                  child: Icon(
                    Icons.check,
                    color: isDarkMode ? Colors.black : Colors.white,
                    size: 30,),), // Icon // Container
                const SizedBox(height: 10),
                Text( tr('success_title'),
                  style: TextStyle(
                    fontSize: 20,
                    color: isDarkMode ? Colors.white : Colors.black,
                    fontWeight: FontWeight.bold,),), // TextStyle // Text
                const SizedBox(height: 30),
                Text(tr('success_message'),
                  style: TextStyle(
                    fontSize: 12,
                    color: isDarkMode ? Colors.white : Colors.black,),),)],),),),
          actions: [
            Center(
              child: TextButton(
                onPressed: () {
                  Navigator.of(context).pop();
                  Navigator.push(
                    context,
                    MaterialPageRoute(
                      builder: (context) => health(result: widget.result),),);},
                child: Text(
                  tr('show_result'),
```

Figure 8.59 : Code Loading Screen p2



```
                    tr('show_result'),
                    style: const TextStyle(
                      fontSize: 16,
                      color: Color.fromARGB(255, 11, 170, 143),
                      fontWeight: FontWeight.bold,),),),),],);},);} // TextStyle // Text // TextButton // Cen
  @override
  Widget build(BuildContext context) {
    final brightness = Theme.of(context).brightness;
    final backgroundImage = brightness == Brightness.dark
        ? 'C:/Users/CS/Desktop/flutter_application_1/lib/Image/loadingbark.png'
        : 'C:/Users/CS/Desktop/flutter_application_1/lib/Image/Ba.PNG';
    String progressText = (_progress * 100).round().toString();
    if (context.locale.languageCode == 'ar') {
      progressText = progressText.replaceAllMapped(RegExp(r'\d'),
  (match) => String.fromCharCode( match.group(0)!.codeUnitAt(0) + 0x0660 - 0x0030),);}
    return Scaffold(body: Stack( children: [
          Container( decoration: BoxDecoration(
              image: DecorationImage(image: AssetImage(backgroundImage),
                fit: BoxFit.cover,),), ), // DecorationImage // BoxDecoration // Container
          Positioned(
            bottom: 100, left: 150, right: 150, child: Column(
              children: <Widget>[
                LinearProgressIndicator(
                  value: _progress,
                  backgroundColor: const Color.fromARGB(255, 245, 255, 253),
                  valueColor: const AlwaysStoppedAnimation<Color>(
                    Color.fromARGB(255, 11, 170, 143),),), // AlwaysStoppedAnimation // LinearProgressInd
                const SizedBox(height: 10),
                Text('$progressText${tr('percent_sign')}',
                  style: const TextStyle(
                    fontSize: 18, fontWeight: FontWeight.bold,color: Color.fromARGB(255, 11, 170, 143),
```

Figure 8.60 : Code Loading Screen p3

8.2.8 Diagnosis Result Screen

This page displays the AI-generated diagnosis result based on the uploaded image. It shows the predicted skin condition, the confidence percentage, and gives a recommendation along with a navigation option to hospitals. If the confidence level is low, it informs the user that no disease was detected.

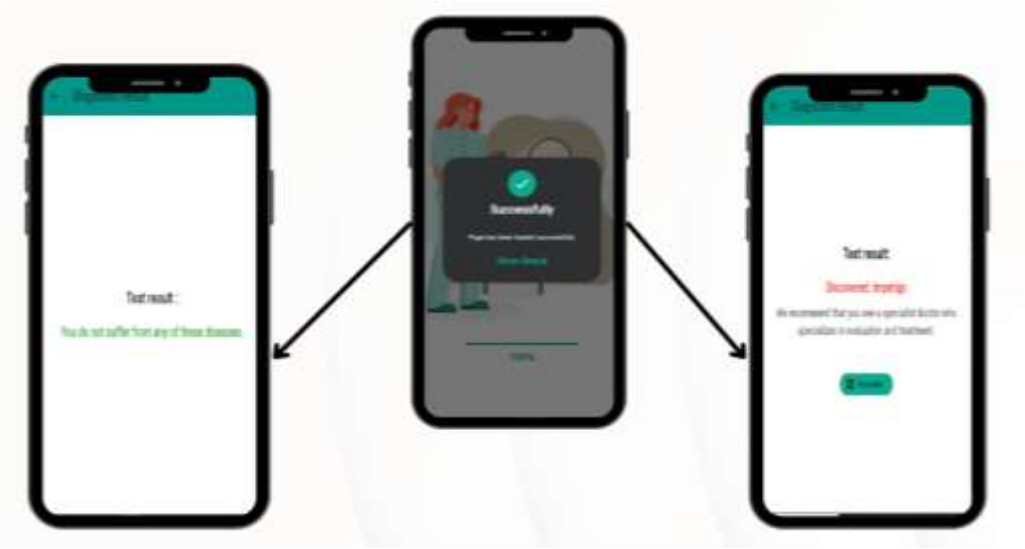
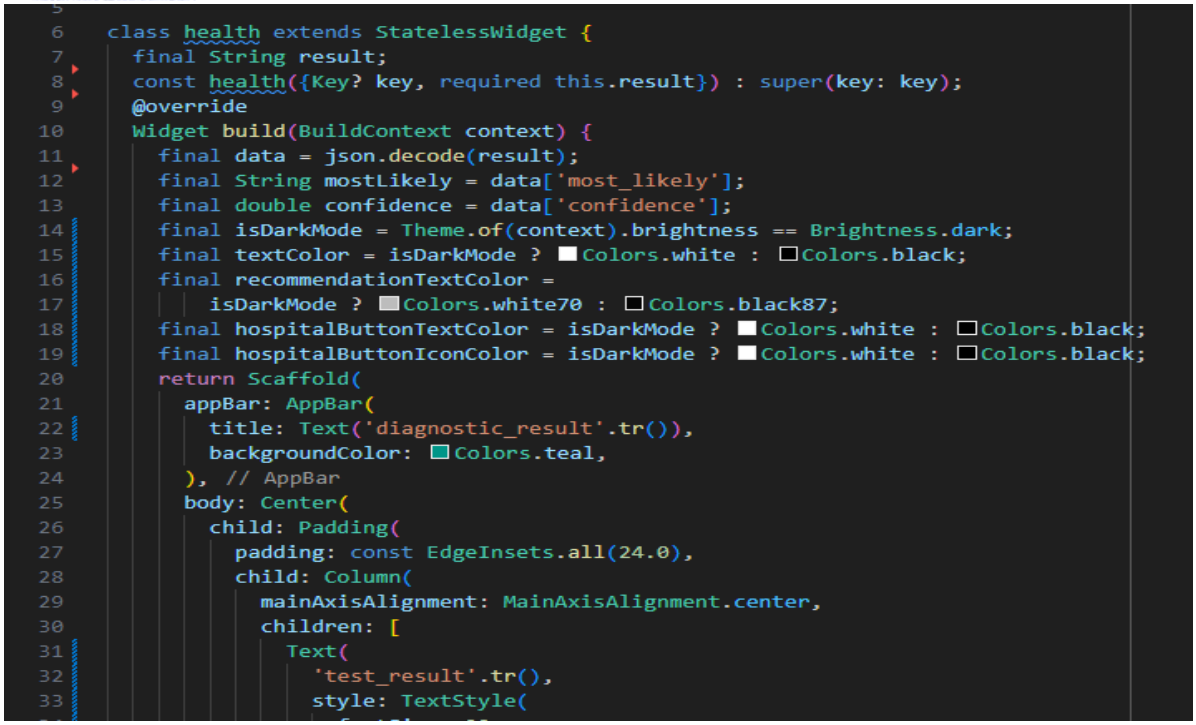
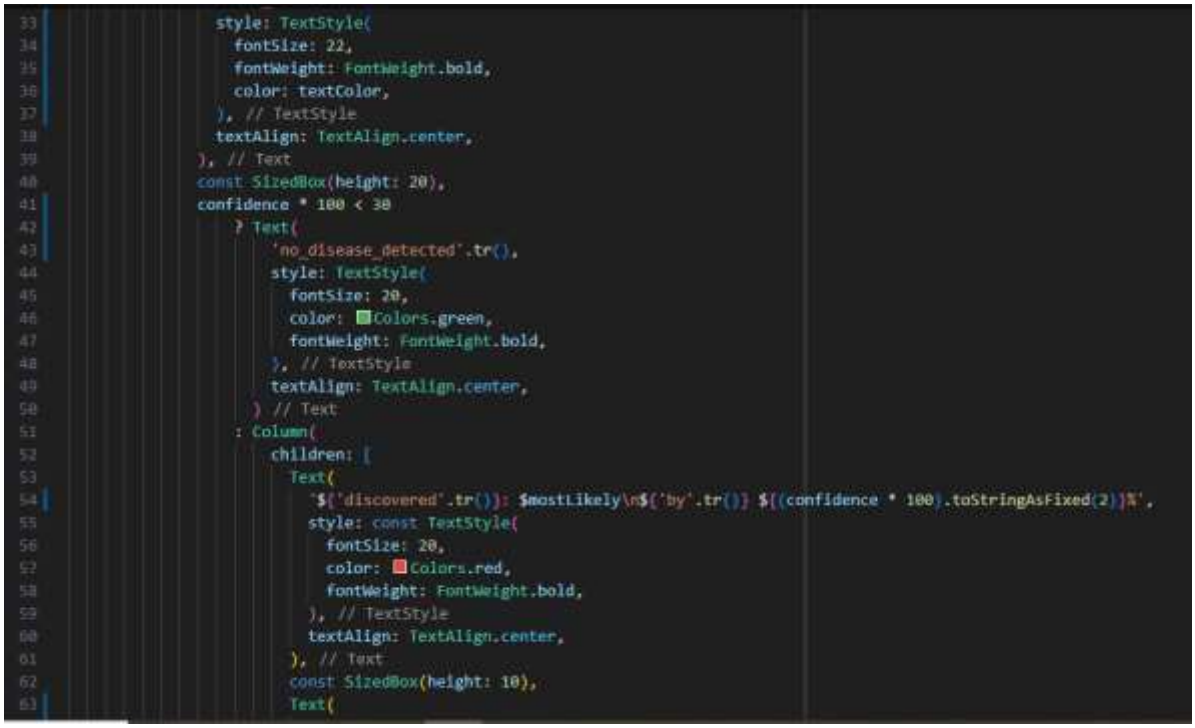


Figure 8.61: Diagnosis Result Screen



```dart
class health extends StatelessWidget {
  final String result;
  const health({Key? key, required this.result}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    final data = json.decode(result);
    final String mostLikely = data['most_likely'];
    final double confidence = data['confidence'];
    final isDarkMode = Theme.of(context).brightness == Brightness.dark;
    final textColor = isDarkMode ? Colors.white : Colors.black;
    final recommendationTextColor =
        isDarkMode ? Colors.white70 : Colors.black87;
    final hospitalButtonTextColor = isDarkMode ? Colors.white : Colors.black;
    final hospitalButtonIconColor = isDarkMode ? Colors.white : Colors.black;
    return Scaffold(
      appBar: AppBar(
        title: Text('diagnostic_result'.tr()),
        backgroundColor: Colors.teal,
      ), // AppBar
      body: Center(
        child: Padding(
          padding: const EdgeInsets.all(24.0),
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Text(
                'test_result'.tr(),
                style: TextStyle(
```

Figure 8.62 : Code Diagnosis Result Screen p1



```dart
            style: TextStyle(
              fontSize: 22,
              fontWeight: FontWeight.bold,
              color: textColor,
            ), // TextStyle
            textAlign: TextAlign.center,
          ), // Text
          const SizedBox(height: 20),
          confidence * 100 < 30
              ? Text(
                  'no_disease_detected'.tr(),
                  style: TextStyle(
                    fontSize: 20,
                    color: Colors.green,
                    fontWeight: FontWeight.bold,
                  ), // TextStyle
                  textAlign: TextAlign.center,
                ) // Text
              : Column(
                  children: [
                    Text(
                      '${'discovered'.tr()}: $mostLikely\n${'by'.tr()} ${(confidence * 100).toStringAsFixed(2)}%',
                      style: const TextStyle(
                        fontSize: 20,
                        color: Colors.red,
                        fontWeight: FontWeight.bold,
                      ), // TextStyle
                      textAlign: TextAlign.center,
                    ), // Text
                    const SizedBox(height: 10),
                    Text(
```

Figure 8.63 : Code Diagnosis Result Screen p2

```
                textAlign: TextAlign.center,
              ), // Text
              const SizedBox(height: 10),
              Text(
                'recommendation'.tr(),
                style: TextStyle(
                  fontSize: 18,
                  color: recommendationTextColor,
                ), // TextStyle
                textAlign: TextAlign.center,
              ), // Text
              const SizedBox(height: 50),
              ElevatedButton.icon(
                icon: Icon(Icons.local_hospital,
                    color: hospitalButtonIconColor), // Icon
                label: Text(
                  "hospitals".tr(),
                  style: TextStyle(color: hospitalButtonTextColor),
                ), // Text
                style: ElevatedButton.styleFrom(
                  backgroundColor:
                      const Color.fromARGB(255, 11, 170, 143),
                ),
                onPressed: () {
                  Navigator.push(
                    context,
                    MaterialPageRoute(
                        builder: (_) => SelectGovernorateScreen()), // MaterialPageRoute
                  );},), ], ),  ],), ),   ),); }} // ElevatedButton.icon // Column // Column
```

Figure 8.64 : Code Diagnosis Result Screen p3

8.2.9 Select Governorate Screen

This page allows the user to choose their governorate to display a list of nearby hospitals. It serves as a location-based filter after receiving a diagnosis result, helping users find relevant medical facilities.

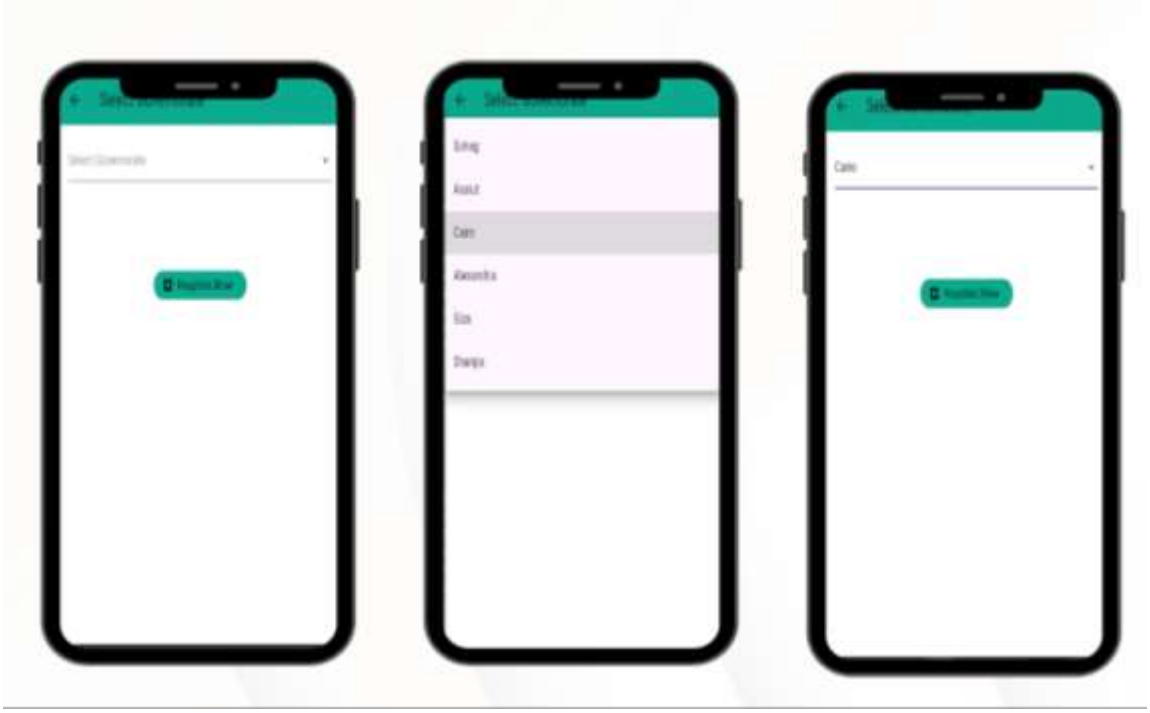


Figure 8.65 : Select Governorate Screen

```dart
class SelectGovernorateScreen extends StatefulWidget {
const SelectGovernorateScreen({Key? key}) : super(key: key);
  @override
  State<SelectGovernorateScreen> createState() =>
      _SelectGovernorateScreenState();}
class _SelectGovernorateScreenState extends State<SelectGovernorateScreen> {
  String? selectedGovernorate;
  final List<String> governorates = ['Sohag','Assiut', 'Cairo','Alexandria','Giza','Sharqia',];
  @override
  Widget build(BuildContext context) {
    bool isDarkMode = Theme.of(context).brightness == Brightness.dark;
    Color textColor = isDarkMode ? Colors.white : Colors.black;
    return Scaffold(
      appBar: AppBar(
        backgroundColor: const Color.fromARGB(255, 11, 170, 143),
        title: Text(
          'select_governorate'.tr(),
          style: TextStyle(color: textColor),
        ), // Text
        iconTheme: IconThemeData(color: textColor),
      ), // AppBar
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          children: [
            DropdownButtonFormField<String>(
              value: selectedGovernorate,
              hint: Text(
                'select_governorate'.tr(),
                style: TextStyle(color: textColor),), // Text
```

Figure 8.65 : Code Select Governorate p1

```dart
            dropdownColor:
                isDarkMode ? const Color(0xFF1E1E1E) : Colors.white,
            items: governorates.map((gov) {
              return DropdownMenuItem(
                value: gov,
                child: Text(
                  'gov_$gov'.tr(),
                  style: TextStyle(color: textColor),),);}).toList(), // Text // DropdownMenuItem
            onChanged: (value) {
              setState(() {selectedGovernorate = value;});},), // DropdownButtonFormField
          const SizedBox(height: 100),
          ElevatedButton.icon(
            icon: Icon(
              Icons.local_hospital,
              color: textColor,), // Icon
            label: Text(
              'show_hospitals'.tr(),
              style: TextStyle(color: textColor),), // Text
            style: ElevatedButton.styleFrom(
              backgroundColor: const Color.fromARGB(255, 11, 170, 143),),
            onPressed: () {
              if (selectedGovernorate != null) {
                Navigator.push(
                  context,
                  MaterialPageRoute(
                    builder: (_) => HospitalsScreen(
                      governorate: selectedGovernorate!,),),); // HospitalsScreen // MaterialPageRoute
              } else {
                ScaffoldMessenger.of(context).showSnackBar(
                  SnackBar(
                    content: Text('please_select_governorate'.tr()),),);}},),],),),);} // SnackBar // ElevatedButton.ic
```

Figure 8.66 : Code Select Governorate p2

8.2.10 Hospitals Screen

This page displays a list of hospitals based on the governorate selected by the user. It retrieves hospital data from Firebase Firestore and allows users to open each hospital's location on Google Maps.

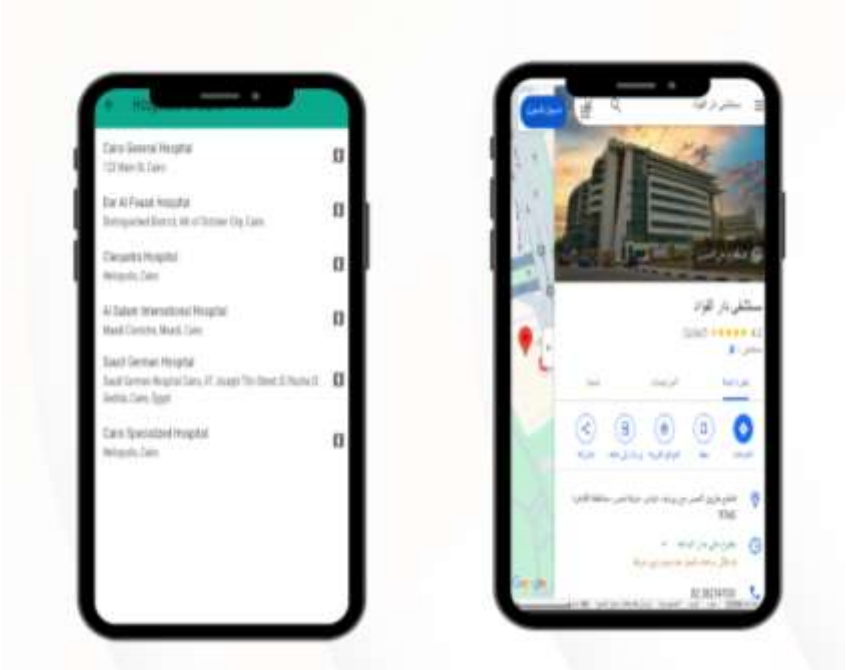


Figure 8.67 : Hospitals Screen

```dart
class HospitalsScreen extends StatelessWidget {
  final String governorate;
  const HospitalsScreen({Key? key, required this.governorate})
      : super(key: key);
  Future<void> _launchMapUrl(String url) async {
    final Uri uri = Uri.parse(url);
    if (await canLaunchUrl(uri)) {
      await launchUrl(uri, mode: LaunchMode.externalApplication);
    } else {
      throw 'Could not launch $url';}}
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: const Color.fromARGB(255, 11, 170, 143),
        title: Text('hospitals_in'.tr(args: [governorate])),
      ), // AppBar
      body: StreamBuilder<QuerySnapshot>(
        stream: FirebaseFirestore.instance
            .collection('hospitals')
            .where('governorate', isEqualTo: governorate)
            .snapshots(),
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.waiting) {
            return const Center(child: CircularProgressIndicator());
          }
          if (!snapshot.hasData || snapshot.data!.docs.isEmpty) {
            return Center(child: Text('no_hospitals_found'.tr()));
```

Figure 8.67 : Code Hospitals Screen p1

```dart
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.waiting) {
            return const Center(child: CircularProgressIndicator());
          }
          if (!snapshot.hasData || snapshot.data!.docs.isEmpty) {
            return Center(child: Text('no_hospitals_found'.tr()));
          }
          final docs = snapshot.data!.docs;
          return ListView.builder(
            itemCount: docs.length,
            itemBuilder: (context, index) {
              final data = docs[index].data() as Map<String, dynamic>;
              final name = data['name'] ?? 'no_name'.tr();
              final address = data['address'] ?? 'no_address'.tr();
              final mapUrl = data['map_url'];
              return ListTile(
                title: Text(name),
                subtitle: Text(address),
                trailing: const Icon(Icons.map),
                onTap: () {
                  if (mapUrl != null && mapUrl.toString().isNotEmpty) {
                    _launchMapUrl(mapUrl);
                  } else {
                    ScaffoldMessenger.of(context).showSnackBar(
                      SnackBar(content: Text('map_not_available'.tr())),
                    ); }},);},);}, // ListTile // ListView.builder
```

Figure 8.68 : Code Hospitals Screen p2

8.2.11 Menu Drawer

The Menu Drawer provides the user with easy navigation and settings, such as switching themes, changing language, viewing account information, and logging out.

Figure 8.69 : Menu Drawer

```
class Menu extends StatefulWidget {
  const Menu({super.key});
  @override
  State<Menu> createState() => _MenuState();}
class _MenuState extends State<Menu> {
  String? username;
  String? email;
  String? gender;
  @override
  void initState() {
    super.initState();
    fetchUserData();}
  Future<void> fetchUserData() async {
    final uid = FirebaseAuth.instance.currentUser?.uid;
    if (uid != null) {
      final doc =
          await FirebaseFirestore.instance.collection('users').doc(uid).get();
      if (doc.exists) {
        setState(() {
          username = doc['username'];
          email = doc['email'];
          gender = doc['gender'];});}}}
  Widget _getGenderIcon() {
    String imagePath;
    if (gender == 'Male') {
      imagePath =
```

Figure 8.70 : Code Menu Drawer p1

```
      imagePath =
          'C:/Users/CS/Desktop/flutter_application_1/lib/Image/icon1 (1).png';
    } else {
      imagePath =
          'C:/Users/CS/Desktop/flutter_application_1/lib/Image/icon1 (2).png';}
    return ClipOval(
      child: Image.asset(
        imagePath,
        width: 60,
        height: 60,
        fit: BoxFit.cover, ),);} // Image.asset // ClipOval
  @override
  Widget build(BuildContext context) {
    bool isDarkMode = Provider.of<ThemeNotifier>(context).isDarkMode;
    return Drawer(
      child: ListView(
        children: [
          DrawerHeader(
            decoration: const BoxDecoration(
              color: Color.fromARGB(255, 11, 170, 143),
            ), // BoxDecoration
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                _getGenderIcon(),
                const SizedBox(height: 10),
                Text(
                  username ?? '...loading',
                  style: const TextStyle(fontSize: 18, color: Colors.white),), // Text
                const SizedBox(height: 5),
                Text(
```

Figure 8.71 : Code Menu Drawer p2

```
                  Text( email ?? '',
                    style: const TextStyle(fontSize: 14, color: Colors.white70)),],),), // Text /
            ListTile(
              leading: const Icon(Icons.home),
              title: Text('home'.tr()),
              onTap: () => Navigator.pop(context), ), // ListTile
            ListTile(
              leading: const Icon(Icons.info),
              title: Text('about'.tr()),
              onTap: () => Navigator.pop(context),), // ListTile
            ListTile(
              leading: Icon(isDarkMode ? Icons.nights_stay : Icons.sunny),
              title: Text(isDarkMode ? 'dark'.tr() : 'light'.tr()),
              onTap: () {
                Provider.of<ThemeNotifier>(context, listen: false).toggleTheme();},), // ListTile
            ListTile(
              leading: const Icon(Icons.language),
              title:
                  Text(context.locale.languageCode == 'en' ? 'English' : 'عربي'),
              onTap: () {
                if (context.locale.languageCode == 'en') {
                  context.setLocale(const Locale('ar'));
                } else {
                  context.setLocale(const Locale('en'));}},), // ListTile
            ListTile(
              leading: const Icon(Icons.logout),
              title: Text("Logout".tr()),
              onTap: () {
                Navigator.push(context,
                  MaterialPageRoute(builder: (context) => login()),);},),],),);}} // ListTile // Li
```

Figure 8.72 : Code Menu Drawer p3

8.2.12 Translation Integration

This Flutter application features a complete translation system that allows users to seamlessly switch between Arabic and English. Implemented using the easy_localization package, all UI elements — including buttons, menus, forms, and medical content — are dynamically translated based on the selected language.

Users can change the language at any time through the Menu Drawer, and their preference is stored locally using SharedPreferences to ensure it persists across sessions.

This multilingual capability enhances accessibility, supports a diverse user base, and contributes to a globally inclusive and user-friendly design.

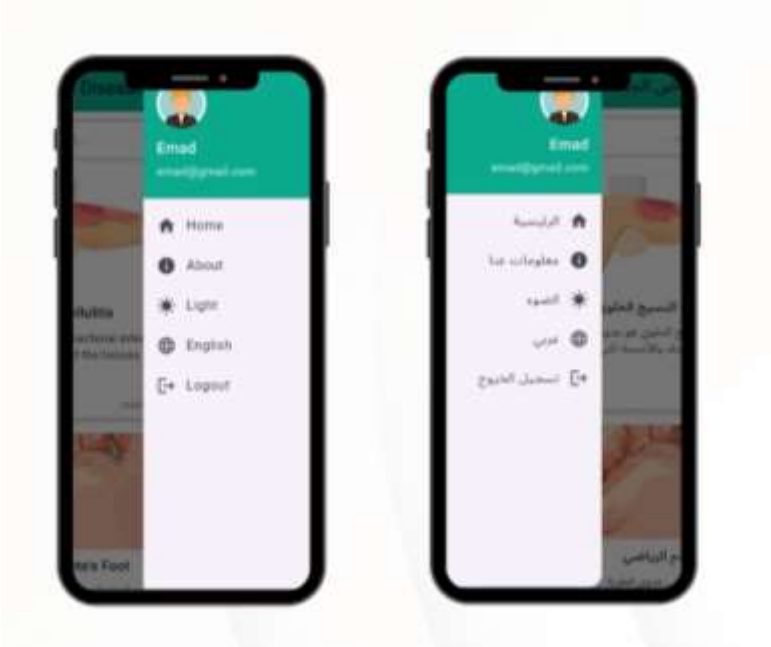


Figure 8.73 : Translation

```en.json
{
  "menu": "Menu",
  "home": "Home",
  "about": "About",
  "loading_page": "Loading Page",
  "skin_disease": "Skin Disease",
  "health": "Health",
  "create_account": "Create an Account",
  "dark": "Dark",
  "light": "Light",
  "change_language": "Change Language",
  "login_your_account": "Login Your Account",
  "email": "Email",
  "sign_up": "Sign Up",
  "Register Your Account": "Register Your Account",
  "User Name": "User Name",
  "Confirm Password": "Confirm Password",
  "Prevention": "Prevention",
  "Symptoms": "Symptoms",
  "Causes": "Causes",
```

```ar.json
{
  "menu": "القائمة",
  "home": "الرئيسية",
  "about": "معلومات عنا",
  "loading_page": "صفحة التحميل",
  "health": "الصحة",
  "skin_disease": "أمراض الجلد",
  "create_account": "إنشاء حساب",
  "signup": "تسجيل",
  "dark": "الظلام",
  "light": "الضوء",
  "change_language": "تغيير اللغة",
  "login_your_account": "سجّل دخولك",
  "email": "البريد الإلكتروني",
  "password": "كلمة المرور",
  "login": "تسجيل الدخول",
  "dont_have_account": "ليس لديك حساب؟",
  "Prevention": "الوقاية",
  "Symptoms": "الأعراض",
  "Causes": "الأسباب",
```

Figure 8.74: Code Translation

8.2.13 Theme Support

This Flutter project offers complete support for both dark and light modes. The theming system is managed through a custom ThemeNotifier class in combination with the Provider package, allowing dynamic theme switching at runtime.

Users can easily toggle between dark and light themes via the Menu Drawer, and their preference is saved locally using SharedPreferences, ensuring it persists across app sessions.

This feature enhances accessibility, improves usability in different lighting conditions, and contributes to a modern, user-friendly design.

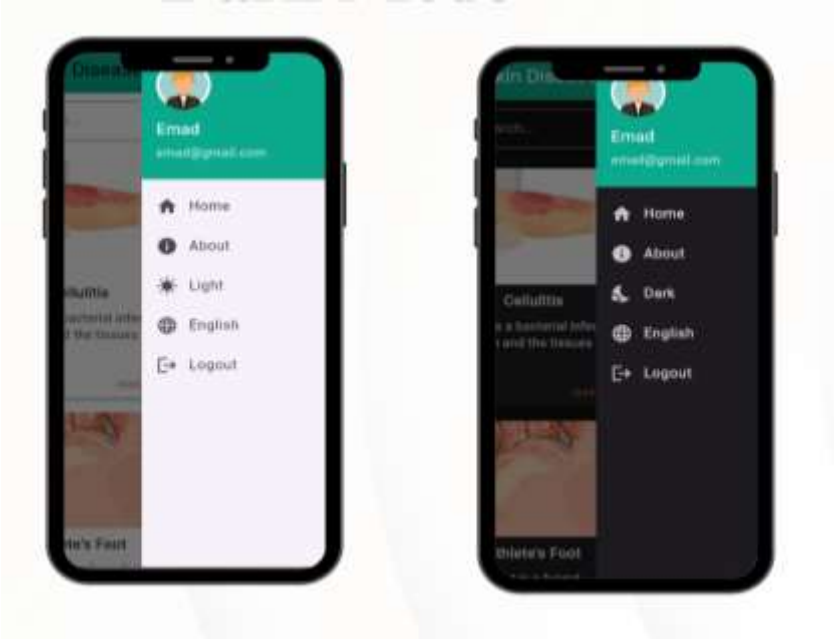


Figure 8.75 : Theme Support

8.2.14 Flask Backend

This project utilizes a Flask-based backend to handle image classification and diagnosis. When a user uploads an image through the Flutter application, it is sent to the Flask server via an HTTP POST request. The server processes the image using a pre-trained deep learning model (such as EfficientNetB0) to identify the most likely skin disease. The result, including the predicted condition and confidence score, is returned in JSON format to the Flutter app.

This integration between Flask and Flutter enables real-time, AI-powered diagnosis and ensures a smooth flow of data between the client and server, contributing to an efficient and responsive user experience.

```python
from flask import Flask, request, jsonify
from flask_cors import CORS
import numpy as np
import cv2
import tensorflow as tf
import os
app = Flask(__name__)
CORS(app)
model = tf.keras.models.load_model(r"C:/Users/CS/Efficientnet_model_skin_disease_model.keras")
classes = ['Cellulitis', 'Impetigo', 'Athlete-foot', 'Nail Fungus', 'Ringworm',
           'Cutaneous Larva Migrans', 'Chickenpox', 'Shingles']
def preprocess(img):
    img = cv2.resize(img, (128, 128))
    img = img.astype('float32') / 255.0
    return img
@app.route('/')
def home():
    return '''<h2>The diagnostic server is running successfully</h2> '''
@app.route('/predict', methods=['POST'])
def predict():
    if 'image' not in request.files:
        return jsonify({'error': 'No image file provided'}), 400
    file = request.files['image']
    img = cv2.imdecode(np.frombuffer(file.read(), np.uint8), cv2.IMREAD_COLOR)
    print(f"Image received with shape: {img.shape}")
    print(f"Image dtype: {img.dtype}")
    img = preprocess(img)
    print(f"Processed image shape: {img.shape}")
    pred = model.predict(np.expand_dims(img, axis=0))[0]
    predicted_index = np.argmax(pred)
    predicted_class = classes[predicted_index]
    prediction_details = [
        {"disease": classes[i], "probability": float(pred[i])}
```

Figure 8.76 : Code Flask p1

```python
        pred = model.predict(np.expand_dims(img, axis=0))[0]
        predicted_index = np.argmax(pred)
        predicted_class = classes[predicted_index]
        prediction_details = [
            {"disease": classes[i], "probability": float(pred[i])}
            for i in range(len(classes))]
        return jsonify({
            "most_likely": predicted_class,
            "confidence": float(pred[predicted_index]),
            "predictions": prediction_details
        })
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Figure 8.77 : Code Flask p2

8.2.15 Firebase

This project uses Firebase as the backend for both authentication and database management. User information — including username, email, and gender — is securely stored in a Firestore collection named "users". Upon login, the app retrieves the corresponding user data using the Firebase Authentication UID, and dynamically displays it in the Menu Drawer, including a gender-based profile icon.

Firebase Authentication also handles secure login and logout flows, ensuring proper session management. Additionally, hospital data is stored in Firebase, organized by governorate, allowing the app to efficiently query and display hospitals based on user location. Firebase's seamless integration with Flutter ensures real-time data syncing and a responsive user experience.[19]

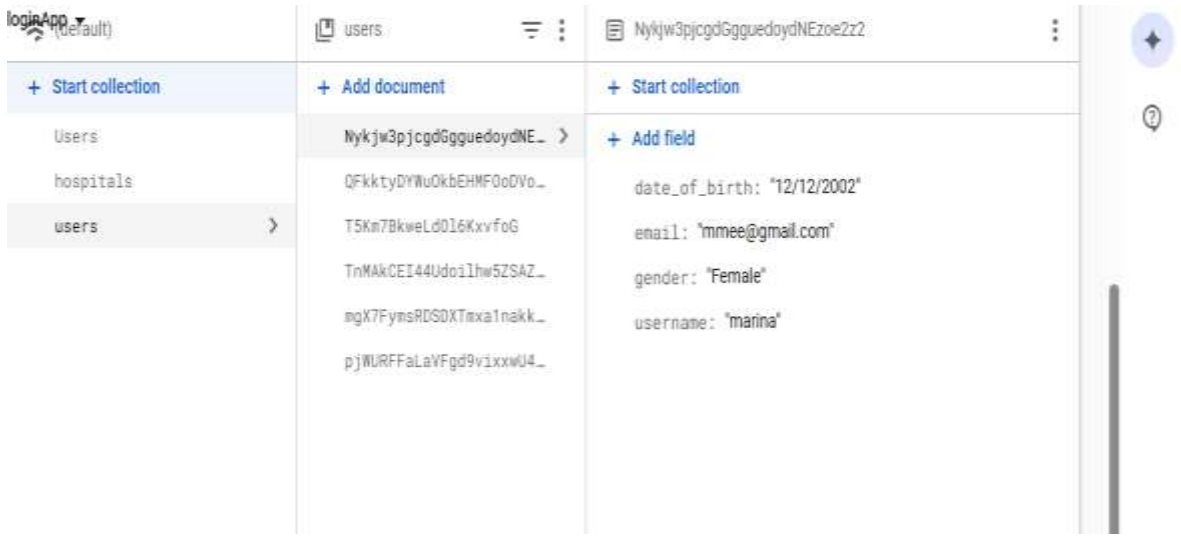


Figure 8.78 : Firebase 1

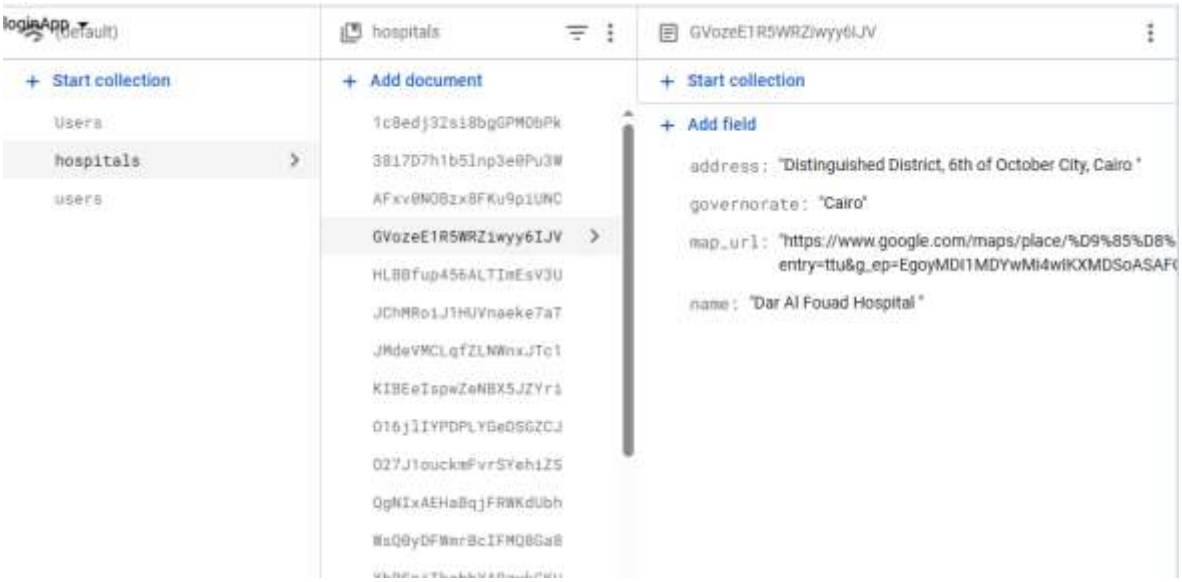


Figure 8.79 : Firebase 2