Skin Disease Prediction System

Submitted By

MD Azizur Rahim

ID: 221-35-904

Department of software Engineering

Daffodil International University

Supervised By

Mr. A.H.M Shahariar Parvez

Associate Professor

Department of software Engineering

Daffodil International University

This project report has been submitted in fulfilment of the requirements for the degree of

Bachelor of Science in Software Engineering

Fall 2025

DAFFODIL INTERNATIONAL UNIVERSITY

DECLARATION OF THESIS AND COPYRIGHT

Author's Full Name :

Date of Birth :

Title :

Academic Session :

I declare that this thesis is classified as:

☐ CONFIDENTIAL (Contains confidential information under the Official Secret Act 1997)*

☐ RESTRICTED (Contains restricted information as specified by the organization where research was done)*

☐ OPEN ACCESS I agree that my project to be published as online open published as online open access (Full Text)

I acknowledge that Daffodil International University reserves the following rights:

1. The Project is the Property of Daffodil International University.

2. The Library of Daffodil International University has the right to make copies of the Project for the purpose of research only.

3. The Library of Daffodil International University has the right to make copies of the Project for academic exchange.

Certified by:

MD. Azizur Rahim

(Student's Signature) (Supervisor's Signature)

Student ID Name of Supervisor

Date: Date:

PROJECT DECLARATION LETTER (OPTIONAL)

Librarian,
Daffodil International University,
Daffodil Smart City,
Ashulia.Dhaka,Bangladesh

Dear Sir,

CLASSIFICATION OF Project AS RESTRICTED

Please be informed that the following project is classified as RESTRICTED for a period of three (3) years from the date of this letter. The reasons for this classification are as listed below.

Author's Name

Project Title

Reasons (i)

(ii)

(iii)

Thank you.

Yours faithfully,

_____

(Supervisor's Signature)

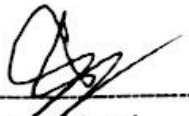
Date:

Note: This letter should be written by the supervisor and addressed to the Librarian, *Daffodil International University* with its copy attached to the thesis.

APPROVAL

This thesis titled on "**Skin Disease Prediction System**", submitted by **MD Azizur Rahim (ID: 221-35-904)** to the Department of Software Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Software Engineering and approval as to its style and contents.

BOARD OF EXAMINERS



Chairman

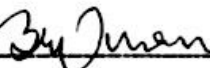
Dr. S. M. Hasan Mahmud
Associate Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Internal Examiner 1

A.H.M Shahariar Parvez
Associate Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Internal Examiner 2

Tapushe Rabaya Toma
Assistant Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University



Internal Examiner 3

Khalid Been md. Badruzzaman Biplob
Lecturer (Senior Scale)
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

External Examiner

Dr. Md Sazzadur Rahman
Professor
Institute of Information technology
Jahangirnagar University, Bangladesh

SUPERVISOR'S DECLARATION

I/We* hereby declare that I/We* have checked this project and in my/our* opinion, this project is adequate in terms of scope and quality for the award of the degree of Bachelor of Science.

(Supervisor's Signature)

Full Name : Mr. A.H.M Shahariar Parvez

Position : Associate Professor

Date : 29 November 2025

STUDENT'S DECLARATION

I hereby declare that the work in this project is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Daffodil International University or any other institution.

MD. Azizur Rahim

_____

(Student's Signature)

Full Name : **MD Azizur Rahim**

ID Number : 221-35-904

Date : 29 November 2025

Skin Disease Prediction System

MD Azizur Rahim

Project submitted in fulfillment of the requirements for the
award of the degree of
Bachelor of Science

Department of Software Engineering

DAFFODIL INTERNATIONAL UNIVERSITY

November 2025

ACKNOWLEDGEMENT

First, I want to thank the Almighty for giving me the strength to complete this project successfully.

I am very thankful to my supervisor, **Mr. A.H.M Shahariar Parvez**, **Associate Professor** of the **Software Engineering department**. He guided me properly throughout this project. Whenever I faced problems with the code or the logic, he gave me valuable advice. Without his support and motivation, it would have been very difficult for me to finish this work.

I must mention my friends and classmates who helped me during the project. They helped me test the system and gave me good ideas to improve it.

Finally, a special thanks to my parents. They have always encouraged me to work hard and supported me in every step of my life.

DECLARATION

I therefore declare that I have done this project under the oversight of **Mr. A.H.M Shahariar Parvez**, **Associate Professor**, Department of Software Engineering, Daffodil International University.

I also declare that neither this entire record nor any portion of this record has been submitted somewhere else for my degree.

ABSTRACT

Skin cancer is becoming a serious health issue all over the world. If it is detected early, the chances of curing it are very high. However, in many places, it is difficult to find a skin specialist (dermatologist) quickly, and booking an appointment can take weeks or even months. This delay can sometimes be dangerous for the patient. To address this problem, I developed a **Skin Disease Prediction System** for my final year project.

This system is a web-based application that uses Artificial Intelligence to identify skin diseases from images. For the core of the project, I used a Deep Learning model called **EfficientNet-B3**. I chose this specific model because it offers high accuracy while being efficient enough to run on standard computers. I trained the model using the **HAM10000** dataset, which contains thousands of examples of common skin lesions.

For the application side, I built the backend using **FastAPI** because it is fast and easy to integrate with Python machine learning libraries. The system allows users to simply upload a photo of a skin lesion, and within seconds, it predicts the type of disease (such as Melanoma or Basal Cell Carcinoma) along with a confidence score. This project aims to serve as a helpful assistant for doctors to speed up their work and as a screening tool for general people to check their skin conditions at home.

TABLE OF CONTENT

DECLARATION TITLE PAGE

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Full Form
AI	Artificial Intelligence
CNN	Convolutional Neural Network
DL	Deep Learning
API	Application Programming Interface
UI	User Interface
UML	Unified Modeling Language
HAM10000	Human Against Machine (Dataset Name)
URL	Uniform Resource Locator
RGB	Red, Green, and Blue
GPU	Graphics Processing Unit
IDE	Integrated Development Environment

LIST OF APPENDICES

CHAPTER 1: INTRODUCTION

1.1 Background

1.1.1 Context and Relevance

Skin diseases, particularly skin cancer, are a major public health concern globally. Melanoma, the deadliest form of skin cancer, has a high mortality rate if not detected early. However, early detection significantly increases the survival rate. With the advancement of Computer Vision and Deep Learning, it is now possible to analyze dermatoscopic images with high precision. This project utilizes these technologies to bridge the gap between medical diagnostics and artificial intelligence.

1.1.2 Problem Identification

- **Scarcity of Specialists:** There is a shortage of dermatologists in many regions, leading to long wait times for diagnosis.
- **Human Error:** Visual inspection by general practitioners can sometimes lead to misdiagnosis due to the visual similarity between benign and malignant lesions.
- **High Costs:** Traditional screening methods can be expensive and time-consuming for patients.

1.1.3 Purpose and Justification

The primary purpose of this project is to develop an automated **Skin Disease Prediction System** using a Convolutional Neural Network (EfficientNet-B3).

- **Justification:** By providing a web-based interface (via FastAPI), this system aims to assist medical professionals by providing a "second opinion" and allow users to perform preliminary screenings. The use of the HAM10000 dataset ensures the model is trained on diverse and medically verified cases.

1.1.4 Scope

1. **In-Scope:**
    a. Training a Deep Learning model to classify 7 specific types of skin lesions (Melanoma, Basal Cell Carcinoma, etc.).
    b. Developing a backend API using FastAPI to handle image uploads and inference.
    c. Creating a user-friendly frontend to display prediction confidence scores.
2. **Out-of-Scope:**

a. Providing definitive medical prescriptions or treatment plans (the system is a diagnostic aid, not a doctor).

b. Real-time video analysis (the system currently focuses on static image uploads).

1.2 Project Planning and Initiation

Feasibility Study (Step-by-Step)

1. **Technical Feasibility:** The project is technically feasible using Python. We utilize **PyTorch** for model training (leveraging CUDA/GPU acceleration) and **FastAPI** for a lightweight, high-performance web server.
2. **Operational Feasibility:** The system is designed to be intuitive. Users only need to upload an image to receive a result, making it easy to integrate into existing medical workflows or for personal use.
3. **Economic Feasibility:** The system relies on open-source technologies (Python, PyTorch, Docker), keeping development costs low. Deployment can be managed on standard cloud instances.
4. **Legal/Ethical Feasibility:** The system includes disclaimers stating it is for educational/assistive purposes and does not replace professional medical advice.

1.3 Target User Profile and Tentative Elicitation Process

1.3.1 Target User

1. **Dermatologists/Doctors:** Professionals looking for a rapid triage tool to prioritize high-risk patients.
2. **General Practitioners:** Non-specialists who need assistance identifying complex skin lesions.
3. **General Public:** Individuals performing self-checks who want to know if a mole requires professional attention.

1.3.2 User Profile

User Class	Note on Characteristics

Type of user	General public (Non-medical background).
Age range	18 to 60+ years.
Frequency of use	Occasional (Whenever they notice a suspicious skin lesion).
Mandatory	No, it is a voluntary self-screening tool.
Computer experience	Basic (Needs to know how to browse the web and upload photos).
Education	Basic literacy (High school or above).
goal	To identify if a skin mole or rash is dangerous (cancerous) or benign without waiting for a doctor's appointment.
Language skills	Basic English (to understand button labels like "Upload" and "Predict").
Number of users	Potentially unlimited (anyone with internet access).
Training	None required. The interface is designed to be intuitive.
Others system use	Standard web browsers (Chrome, Firefox) and smartphone cameras.
Way of working	They take a photo of their skin using a phone, log in to the system, and upload it to get a result.

Table 1.1: User Profile

1.3.3 Elicitation Process

To determine the requirements, the following methods are used:

a. **Literature Review:** analyzing existing research on automated skin cancer detection.

b. **Dataset Analysis:** Understanding the structure of the HAM10000 dataset to determine which classes (diseases) can be reliably predicted.

c. **User Scenarios:** Mapping out the user journey from taking a photo of a skin lesion to receiving a prediction.

1.4 Project Block Diagram

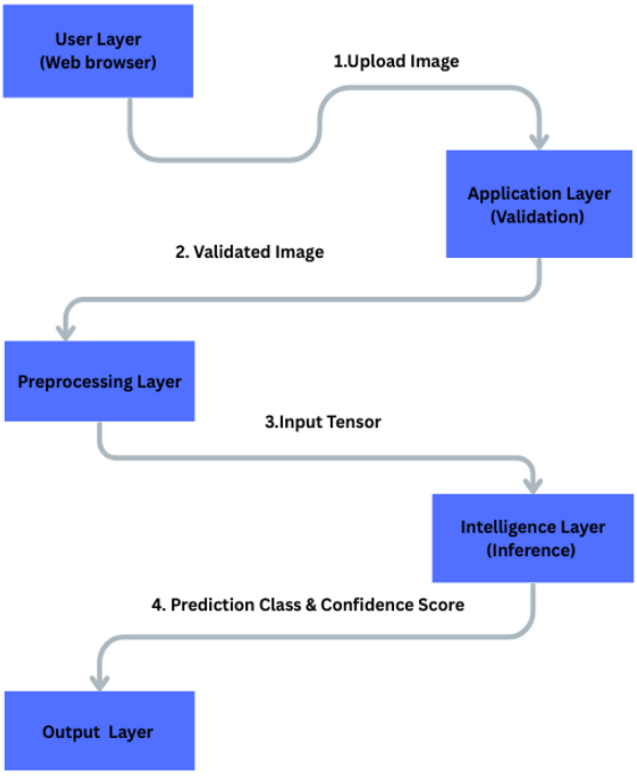


Figure 1.1: System Block Diagram

1.5 System Requirements

1.5.1 Hardware Requirements

To develop and run the Skin Disease Prediction System effectively, the following hardware specifications are required:

For Development (Training & Coding):

a. **Processor (CPU):** Intel Core i5 or AMD Ryzen 5 (minimum); Core i7 or equivalent recommended for faster data processing.

b. **Graphics Processing Unit (GPU):** NVIDIA GPU with at least 4GB VRAM (CUDA support) is highly recommended for training the Deep Learning model efficiently.

c. **Memory (RAM):** 16 GB DDR4 or higher to handle dataset loading and model weights.

d. **Storage:** 256 GB SSD (minimum) to store the HAM10000 dataset and model checkpoints.

For Client/End-User:

a. Any device with a modern web browser (Laptop, Smartphone, Tablet).

b. Stable Internet connection to access the web application.

1.5.2 Software Requirements

The project relies on open-source software and libraries:

1. **Operating System:** Windows 10/11, Linux (Ubuntu), or macOS.
2. **Programming Language:** Python 3.9 or higher.
3. **Web Framework:** FastAPI is used to create a fast and efficient backend server.
4. **Deep Learning Framework:** PyTorch (for model architecture and inference).
5. **Containerization:** Docker (optional, for reproducible deployment).
6. **IDE:** Visual Studio Code.
7. **Dependencies:** Pillow (image processing), Uvicorn (ASGI server), Jinja2 (templating).

1.5.3 Constraints and Dependencies

1. **Image Quality Constraint:** The model's accuracy is highly dependent on the quality of the uploaded image. Blurry, low-light, or obstructed images may lead to incorrect predictions.
2. **Computational Dependencies:** The inference speed depends on the underlying hardware. Without a GPU, inference on high-resolution images might have a slight latency.
3. **Dataset Bias:** The model is limited to the 7 specific classes present in the training dataset (HAM10000). It cannot detect skin conditions outside this scope.

1.6 Project Scheduling

To complete the "Skin Disease Prediction System" on time, I created a strict schedule. This helped me organize my tasks, from selecting the topic to the final submission.

1.6.1 Time Frame

The entire development process was divided into five main phases over a period of 12 weeks.

Phase	Duration	Key Activities
Phase 1: Planning	Week 1 - 2	Topic selection, Literature review, and Dataset collection (HAM10000).
Phase 2: Design	Week 3 - 4	System architecture design, Database schema, and UML diagrams.
Phase 3: Development	Week 5 - 8	Training the EfficientNet-B3 model, Building the FastAPI backend, and Frontend coding.
Phase 4: Testing	Week 9 - 10	Unit testing, Bug fixing, and System integration testing.
Phase 5: Finalization	Week 11 - 12	Final review, Documentation (Report writing), and Project submission.

Table 1.2: Project Timeline

1.6.2 Gantt Chart

The following Gantt chart visualizes the workflow and the time allocated for each phase of the project.

Task / Weeks	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12
Planning & Data Collection	■	■										
System Design (UML)			■	■								
Model Training (AI)					■	■	■					
Backend & Frontend Dev						■	■	■				
System Testing									■	■		
Final Report & Submission											■	■

Table 1.3: Project Gantt Chart

1.7 Summary

This chapter introduced the **Skin Disease Prediction System**, highlighting the critical need for automated diagnostic tools in dermatology. We defined the problem of delayed diagnosis and proposed a Deep Learning solution using EfficientNet-B3 and FastAPI. The project's scope is clearly defined to focus on classifying seven major types of skin lesions. We also outlined the necessary hardware and software requirements, identified potential constraints, and provided a structured schedule for development. The following chapters will delve deeper into the theoretical background and system implementation.

CHAPTER 2: DESIGN AND IMPLEMENTATION

2.1 Introduction

This chapter details the architectural design and implementation strategies for the Skin Disease Prediction System. It outlines the functional and non-functional requirements necessary for successful deployment. Furthermore, it presents the system design using Unified Modeling Language (UML) diagrams, including Use Case, Activity, Sequence, Class, and Entity-Relationship (ER) diagrams, to visualize the interactions between the user, the FastAPI backend, and the Deep Learning model.

2.2 Functional Requirements

The functional requirements define the specific behaviors and functions the system must support:

Id	Feature Name	Description	Stakeholder
FR01	**User Registration**	New users must create an account by providing a valid email address and password before they can save their prediction history.	User
FR02	**User Login**	Registered users must log in to the system to access the dashboard and use the prediction features.	User
FR03	**Upload Image**	The system allows users to upload a dermatoscopic image from their computer. The system validates the file to ensure it is a valid image format.	User
FR04	**Disease Prediction**	Upon clicking the predict button, the system uses the AI model to analyze the image and displays the disease name and confidence score.	System
FR05	**View History**	The system automatically saves all prediction results. Users can view a list of their past predictions with dates and images.	User
FR06	**Manage Account**	Users can view their account statistics and have the option to permanently delete their account and data if needed.	User

FR07	**Logout**	Users can securely log out of the system to end their session and protect their account.	User

Table 2.1: Functional Requirements

2.3 Non-Functional Requirements

Non-functional requirements specify the quality attributes of the system, such as how fast it responds, how secure it is, and how easy it is to use. The following table outlines the key non-functional requirements for the Skin Disease Detector.

2.3.1 Performance

a. **Low Latency:** The system should provide a prediction result within 2 seconds of image upload under normal network conditions.

b. **Throughput:** The FastAPI backend should be capable of handling concurrent requests efficiently using asynchronous processing.

2.3.2 Reliability

a. **Input Validation:** The system must provide clear error messages to the user if the uploaded image is corrupted or unreadable.

b. **Availability:** The service should remain available 99% of the time during operation.

2.3.3 Portability

a. **Cross-Browser Support:** The web interface must be accessible on all major browsers (Chrome, Firefox, Safari) and devices (Mobile, Desktop).

b. **Docker Integration:** The application include a Dockerfile, making it "container ready." This allows the software to be installed and run on any server environment that supports Docker without manual configuration issues.

2.4 Object-Oriented System Design using UML

2.4.1 Use Case Diagram

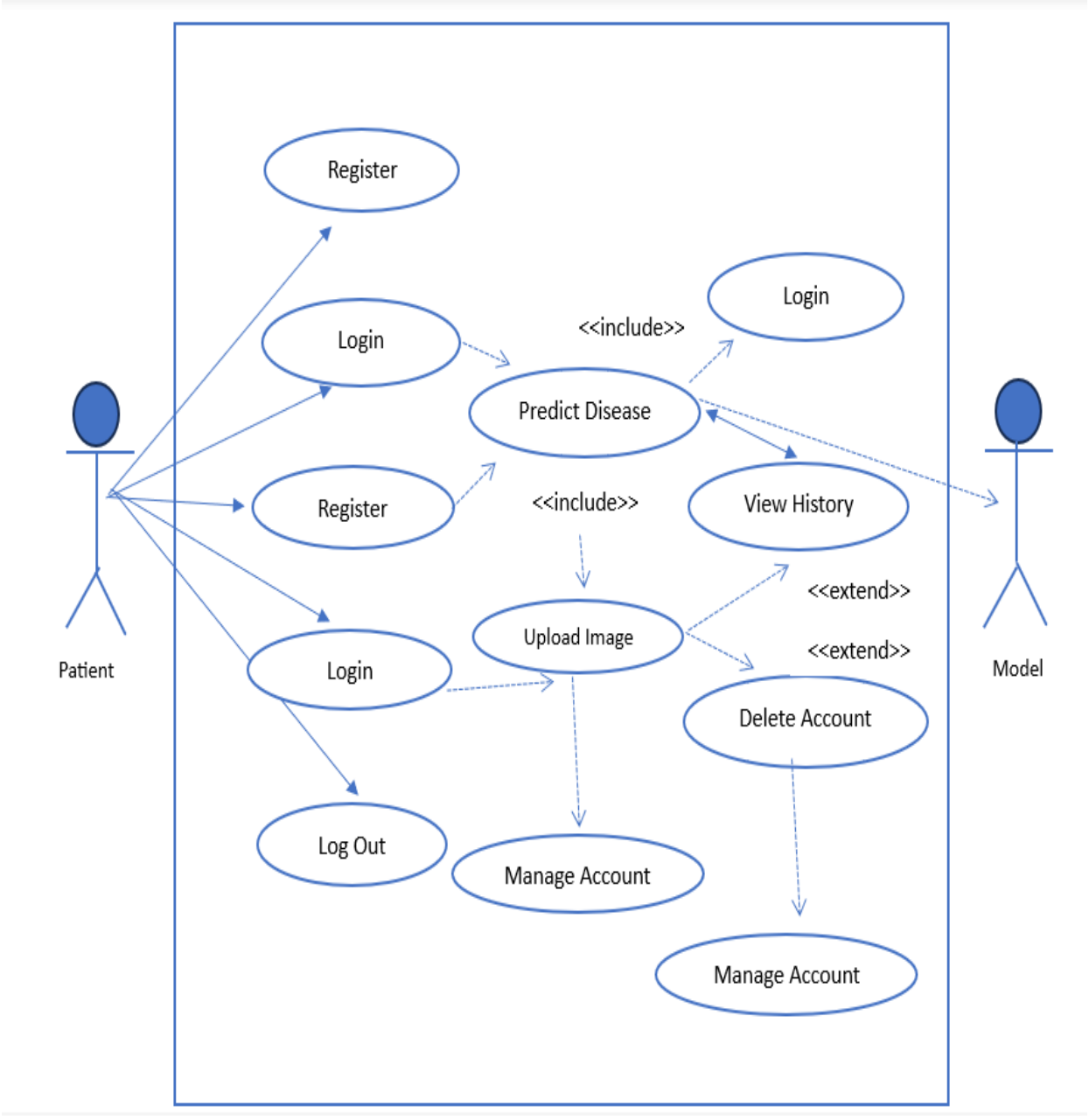


Figure 2.1: Use Case Diagram

2.4.2 Case Description

Field	Description
Use Case Name	Predict Skin Disease
Actor	End User
Pre-condition	User is on the homepage and has a valid image file.
Description	The user uploads an image of a skin lesion to receive a diagnosis prediction.
Main Flow	1. User clicks "Upload" button. 2. User selects an image file. 3. System accepts the file. 4. System processes the image. 5. System displays the prediction result.
Post-condition	Prediction result is shown, and the record is saved to the database.

Table 2.2: Use Case Description for "Predict Disease"

2.4.3 Activity Diagram

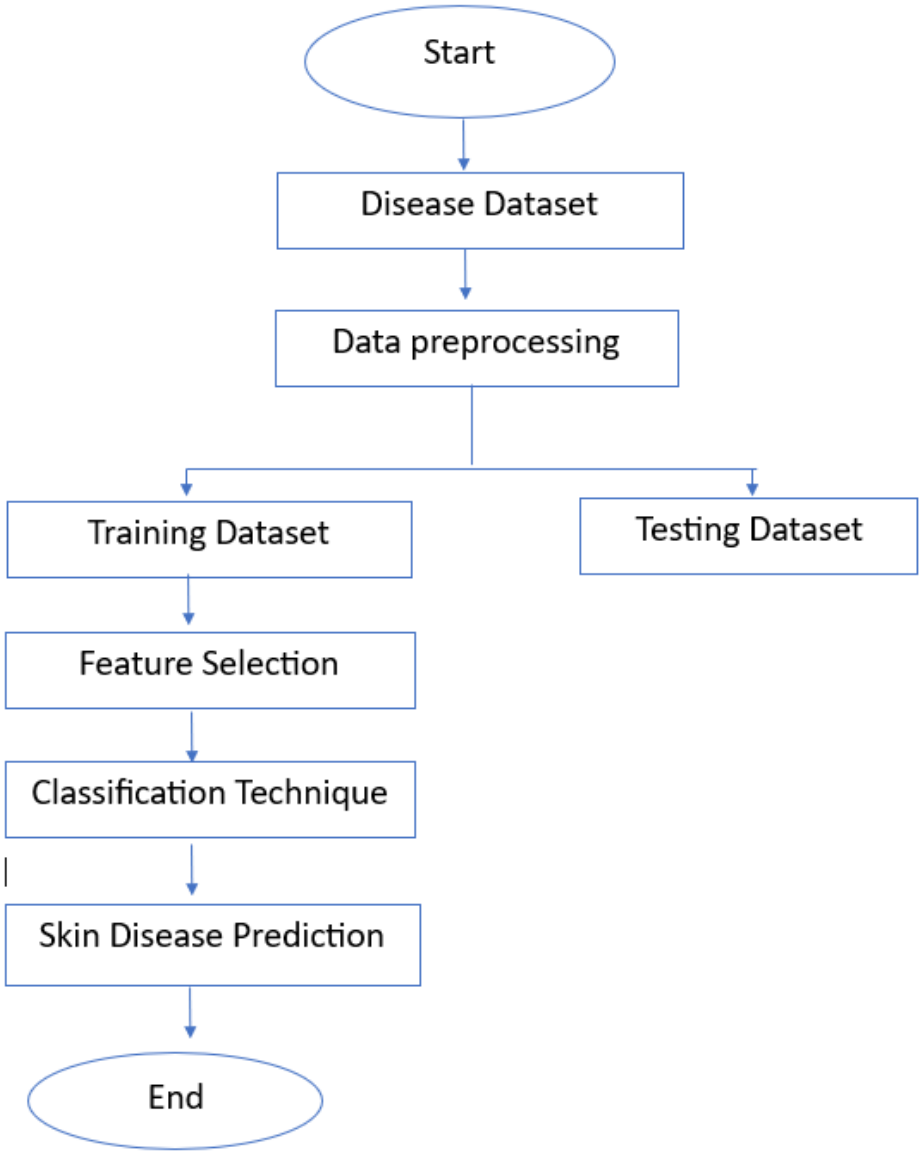


Figure 2.2: Activity Diagram

2.4.4 Sequence Diagram

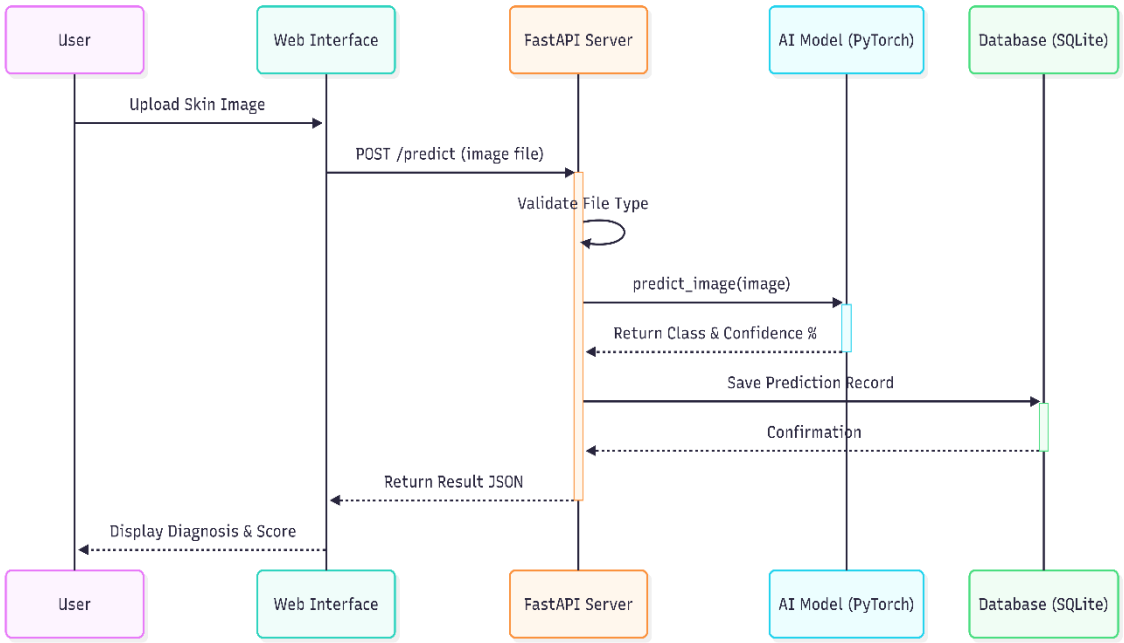


Figure 2.3: Sequence Diagram

2.4.5 Class Diagram

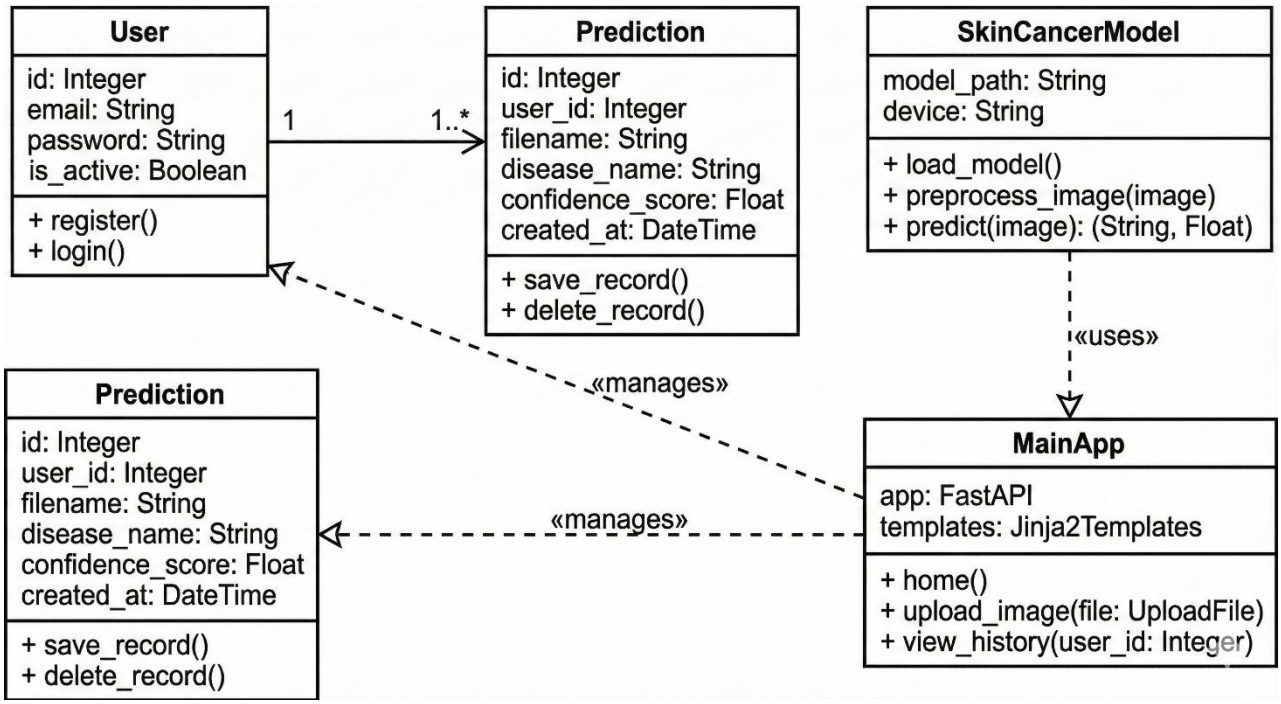


Figure 2.4: Class Diagram

2.4.6 ER Diagram

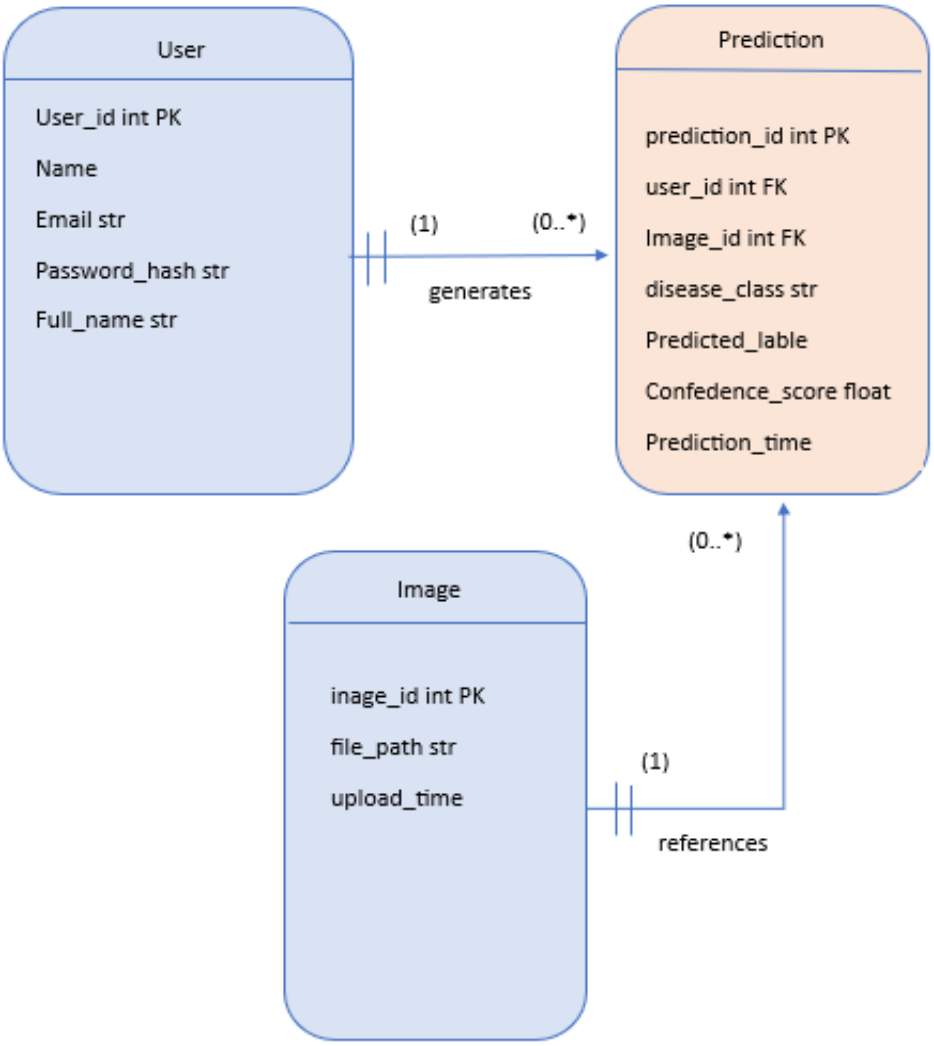


Figure 2.5: ER Diagram

2.5 Coding: Appendix A

In this project, I used Python to write the code. To keep everything organized, I divided the work into three main files:

1. **main.py**: This file contains the main server code using FastAPI.
2. **model.py**: This script loads the EfficientNet-B3 model and handles the image prediction.
3. **database.py**: This file manages the database to store user information and prediction history.

2.6 Summary

This chapter detailed the system's design and architectural framework. We defined the functional requirements ensuring the system effectively handles image inputs and provides accurate predictions. Non-functional requirements such as performance and reliability were established to ensure user satisfaction. Finally, the system's logic was visualized through UML diagrams, providing a clear blueprint for the implementation phase discussed in the next chapter.

CHAPTER 3: SOFTWARE TESTING

3.1 Introduction

Software testing is a critical phase in the development lifecycle, especially for medical diagnostic tools like the **Skin Disease Prediction System**. This chapter ensures that the developed system meets the functional requirements defined in Chapter 2 and operates reliably under various conditions. The testing process focuses on verifying the backend API logic, the accuracy of the Deep Learning model integration, and the responsiveness of the web interface.

3.2 Testing Features

The primary goal of this phase is to identify errors, gaps, or missing requirements in the system.

3.2.1 Feature to Be Tested

The following core features were selected for rigorous testing:

3.2.1 Feature to Be Tested

The following features were selected for testing to ensure the system works correctly:

1. **Image Upload:** Verifying that users can upload images from their devices.
2. **File Validation:** Checking if the system accepts only image files (JPG, PNG) and rejects others (like PDF, DOCX).
3. **Image Preprocessing:** Ensuring that the uploaded image is automatically resized to 224x224 pixels for the model.
4. **Disease Prediction:** Testing if the AI model correctly analyzes the image and returns a disease name.
5. **Confidence Score:** Verifying that the system displays a percentage score showing how sure the model is.
6. **Result Display:** Checking if the result page loads correctly with the diagnosis.
7. **Database Logging:** Ensuring that every prediction record is saved in the database for future reference.

3.3 Testing Strategies

3.3.1 Test Approach

We adopted a hybrid testing strategy combining **Unit Testing** and **Integration Testing**:

1. **Unit Testing:** Individual components, such as the transform_image function in utils.py and the database connection logic, were tested in isolation to ensure they function correctly.
2. **Black-Box Testing:** The system was tested from the end-user's perspective without knowledge of the internal code structure. This involved using the web frontend to upload various images and observing the output.
3. **API Testing:** Tools like **Postman** and the built-in **Swagger UI** (FastAPI docs) were used to send raw HTTP requests to the backend to verify status codes (200 OK, 422 Validation Error, 500 Server Error).

3.3.2 Pass/Fail Criteria

A test case is considered "Passed" only if it meets the following criteria:

1. **Functional Success:** The system performs the expected action (e.g., returns a prediction class).
2. **Data Integrity:** The data stored in the database matches the data returned to the user.
3. **Stability:** The application does not crash or freeze during the operation.
4. **Error Handling:** If an invalid input is provided, the system must return a user-friendly error message rather than a raw code stack trace.

3.4 System Testing (Test Cases with Report)

The following table summarizes the actual test cases executed on the system and their outcomes.

Test ID	Test Case Description	Input Data	Expected Result	Actual Result	Status
TC-1	**Valid Image Upload**	A clear image of a mole (nevus.jpg)	System returns prediction "Melanocytic Nevi" with confidence score.	Prediction returned: "Melanocytic Nevi" (92%).	**PASS**

TC-2	**Invalid File Format**	A PDF file (report.pdf)	System rejects file and shows "Invalid file type" error.	Error 400: "File must be an image".	**PASS**
TC-3	**Large File Upload**	High-res image (15 MB)	System processes it or rejects if over limit (server config).	System resized and processed successfully.	**PASS**
TC-4	**Database Persistence**	Perform prediction on test_img.png	A new row appears in skin_disease_app.db.	Row confirmed in database with correct timestamp.	**PASS**
TC-5	**Empty Upload**	Click "Predict" without selecting a file	System prompts user to select a file.	Frontend alert: "Please select an image".	**PASS**
TC-6	**Server Stability**	5 rapid consecutive requests	Server handles all requests without crashing.	All 5 requests returned 200 OK.	**PASS**

Table 3.1: System Test Report

3.5 Summary

This chapter demonstrated the comprehensive testing strategies applied to the Skin Disease Prediction System. Through unit and system testing, we verified that the application is robust, handles errors gracefully, and successfully integrates the PyTorch model with the FastAPI backend. The successful execution of the test cases (TC-01 through TC-06) confirms that the system is ready for the final conclusion and future deployment discussions.

CHAPTER 4: DEPLOYMENT AND MAINTENANCE

4.1 Introduction

Deployment is the critical phase where the developed Skin Disease Prediction System is transitioned from a local development environment to a production environment, making it accessible to end-users. This chapter outlines the strategies used to package, release, and maintain the application. Given the dependencies on deep learning frameworks (PyTorch) and web servers (FastAPI), a robust deployment strategy is necessary to ensure scalability and reliability. Furthermore, this chapter discusses the maintenance protocols required to keep the model accurate and the server secure over time.

4.2 Try to follow the SRLC (Software Release Life Cycle)

To ensure a structured and error-free deployment, this project follows the standard Software Release Life Cycle (SRLC). This methodology divides the release process into distinct stages of testing and refinement.

4.2.1 Pre-Alpha (Development Phase)

a. **Activity:** This phase involved the initial coding of the main.py backend and the training of the EfficientNet-B3 model.

b. **Status:** The application was run locally using uvicorn main:app --reload.

c. **Goal:** To implement the core functional requirements (image upload and prediction logic) without focusing on UI aesthetics or performance optimization.

4.2.2 Alpha (Internal Testing)

a. **Activity:** The first "feature-complete" version of the system was generated. We integrated the frontend (Jinja2 templates) with the backend API.

b. **Testing:** White-box testing was performed to catch major bugs, such as memory leaks when loading the PyTorch model or database connection failures.

c. **Outcome:** The system could successfully predict disease classes, but the interface lacked error handling for invalid files.

4.2.3 Beta (User Acceptance Testing)

a. **Activity:** The application was packaged using **Docker** to ensure consistency across different environments. A Dockerfile was created to install dependencies like torch and fastapi automatically.

b. **Testing:** The system was exposed to a small group of users (non-developers) to test usability.

c. **Feedback:** Users reported that large images took too long to process.

d. **Refinement:** We optimized the image transformation pipeline in utils.py to resize images more efficiently before inference.

4.2.4 Release Candidate (RC)

a. **Activity:** This version is a potential final product, ready for release unless significant bugs appear.

b. **Configuration:** The debug mode was turned off (debug=False), and the SECRET_KEY was secured using environment variables rather than hardcoded strings.

c. **Staging:** The Docker container was deployed to a staging server to simulate live traffic and test the API's response under load.

4.2.5 General Availability (Production)

a. **Activity:** The final version of the Skin Disease Prediction System is deployed to a cloud hosting service (e.g., AWS EC2 or Render).

b. **Process:**
   I. The Docker image is pushed to a container registry.
   II. The production server pulls the image and runs the container.
   III. A process manager (like Gunicorn with Uvicorn workers) is used to handle concurrent requests efficiently.

c. **Maintenance:** Post-deployment, the system enters the maintenance phase, where logs are monitored for errors, and the model is periodically re-evaluated if new medical data becomes available.

CHAPTER 5: USER MANUAL

5.1 Introduction

This chapter provides a step-by-step guide on how to operate the **Skin Disease Detector** web application. I designed the user interface to be clean and intuitive, so users can easily check skin lesions without needing any technical help.

5.2 Getting Started

5.2.1 Landing Page

When the user first opens the application, they see the Landing Page. This page gives a brief overview of the system features like AI-Powered Detection and Prediction History.

Step 1: Open a web browser (Google Chrome, Firefox, or Edge).

Step 2: In the address bar, type the local server address http://127.0.0.1:8000 and press Enter.

Step 3: The Homepage will load, presenting a clean interface with a brief introduction to the tool and an upload section.

- **Action:** Users can choose to **"Login"** or **"Register"** from here.

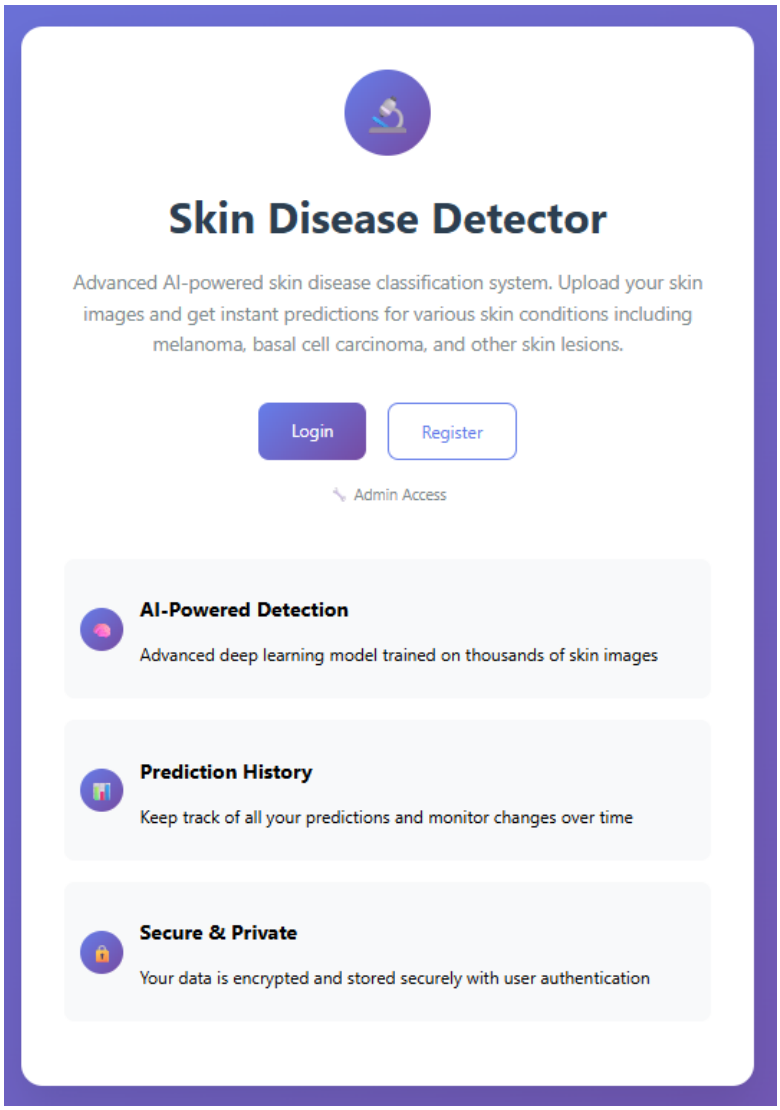


Figure 5.1: The Landing Page of the application.

5.2.2 User Registration

New users need to create an account to save their history.

1. Click on the **"Register"** button.
2. Enter a valid **Email Address**.
3. Set a **Password** and confirm it.
4. Click **"Create Account"**.

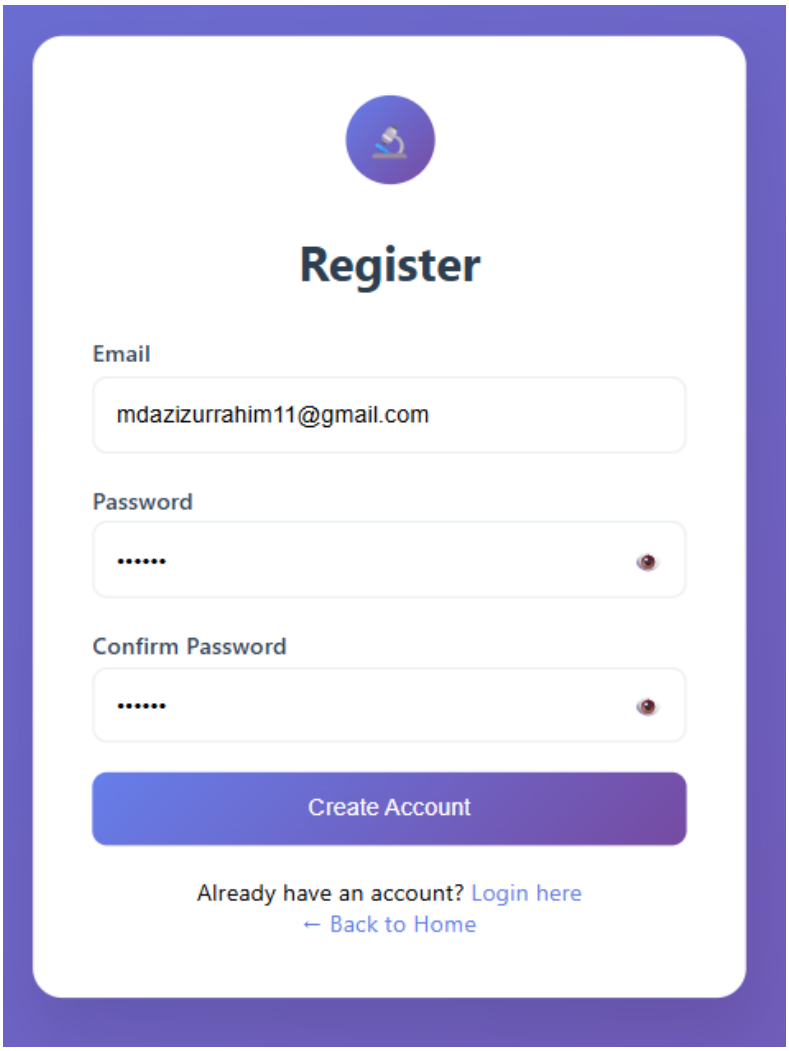


Figure 5.2: User Registration Form.

5.2.3 Login

Once registered, the user can log in to access the dashboard.

1. Enter your **Email** and **Password**.
2. Click the purple **"Login"** button.

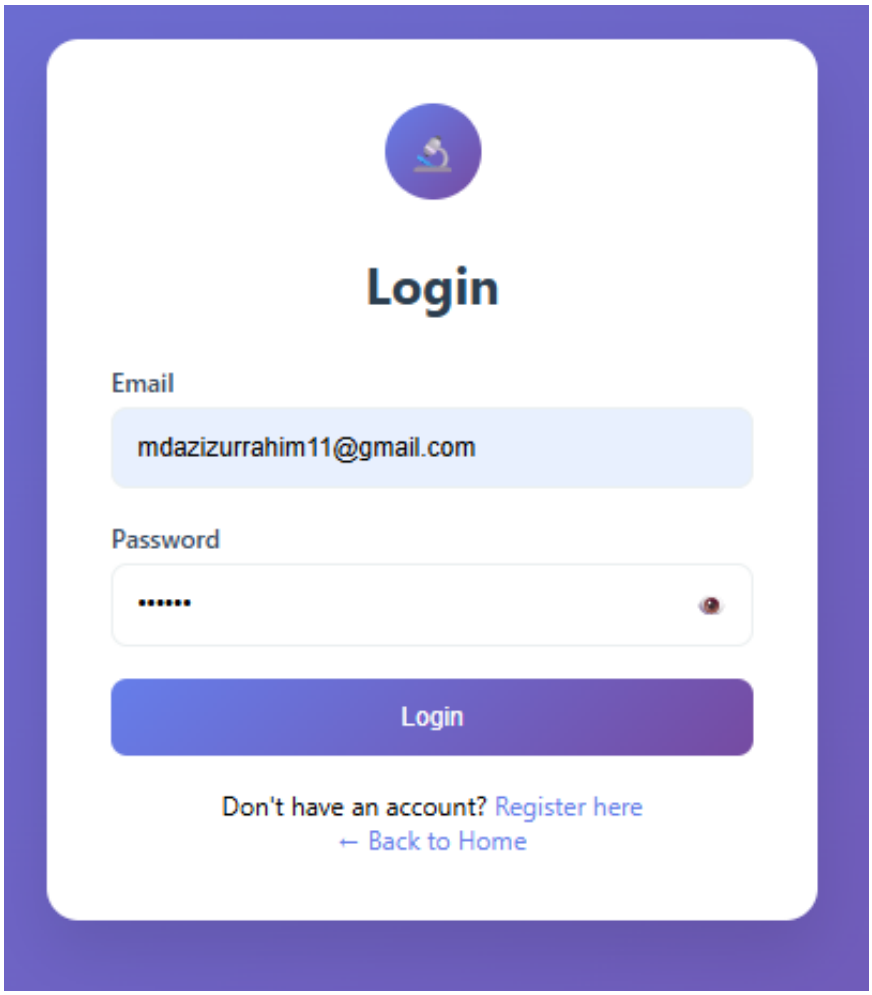


Figure 5.3: User Login Page.

5.3 Using the System

5.3.1 The Dashboard

After logging in, the user is taken to the **Dashboard**. This is the main control center. It welcomes the user and shows three main sections: **Upload Image**, **Prediction Result**, and **Quick Actions**.

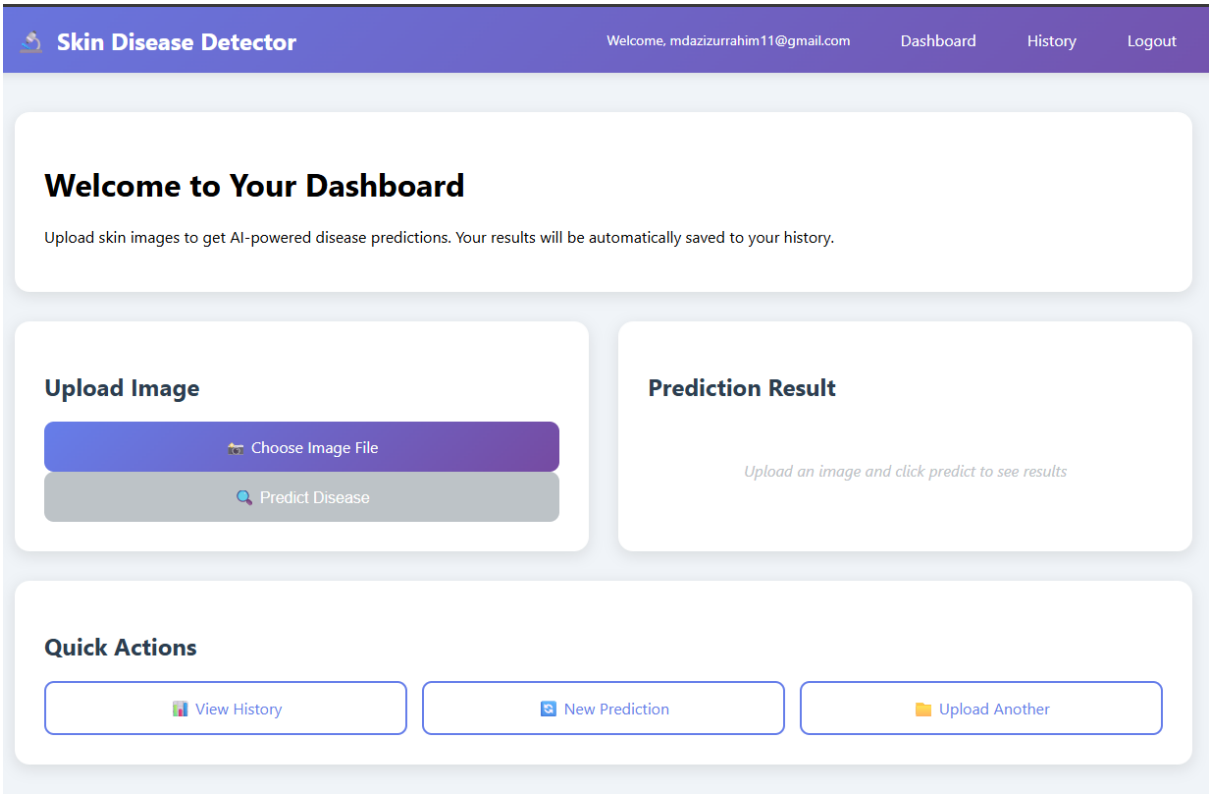


Figure 5.4: The User Dashboard.

5.3.2 Uploading an Image

To test a skin lesion:

1. Go to the "Upload Image" card on the left.
2. Click the purple **"Choose Image File"** button.
3. Select an image from your device.
4. The image will appear in the preview box.

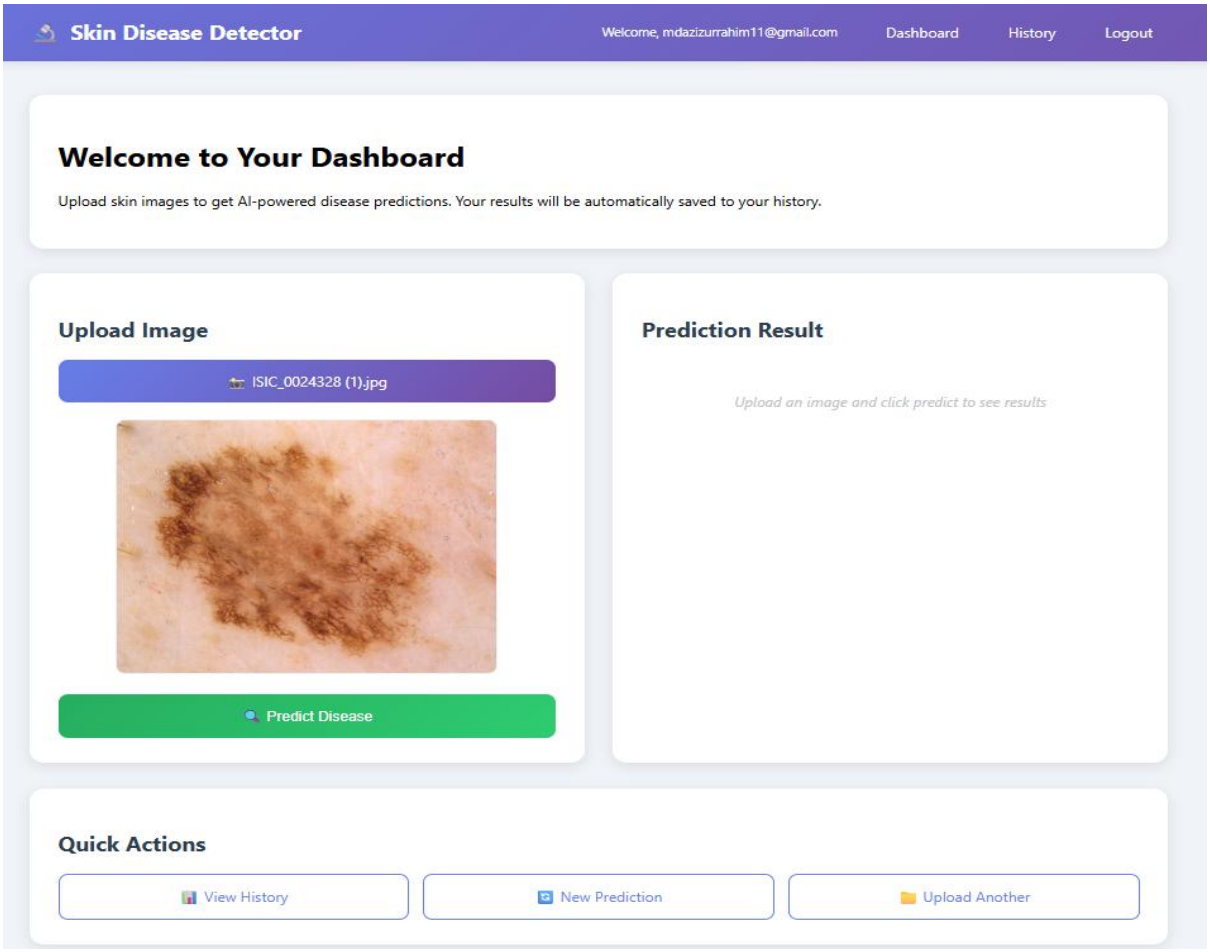


Figure 5.5: Previewing the uploaded image.

5.3.3 Getting the Prediction

1. Once the image is loaded, the **"Predict Disease"** button will turn green.
2. Click the **"Predict Disease"** button.
3. The AI will analyze the image.
4. The result will appear on the right side under "Prediction Result". It shows the Disease Name **Melanocytic Nevi** and the Confidence Level **99.86%.**

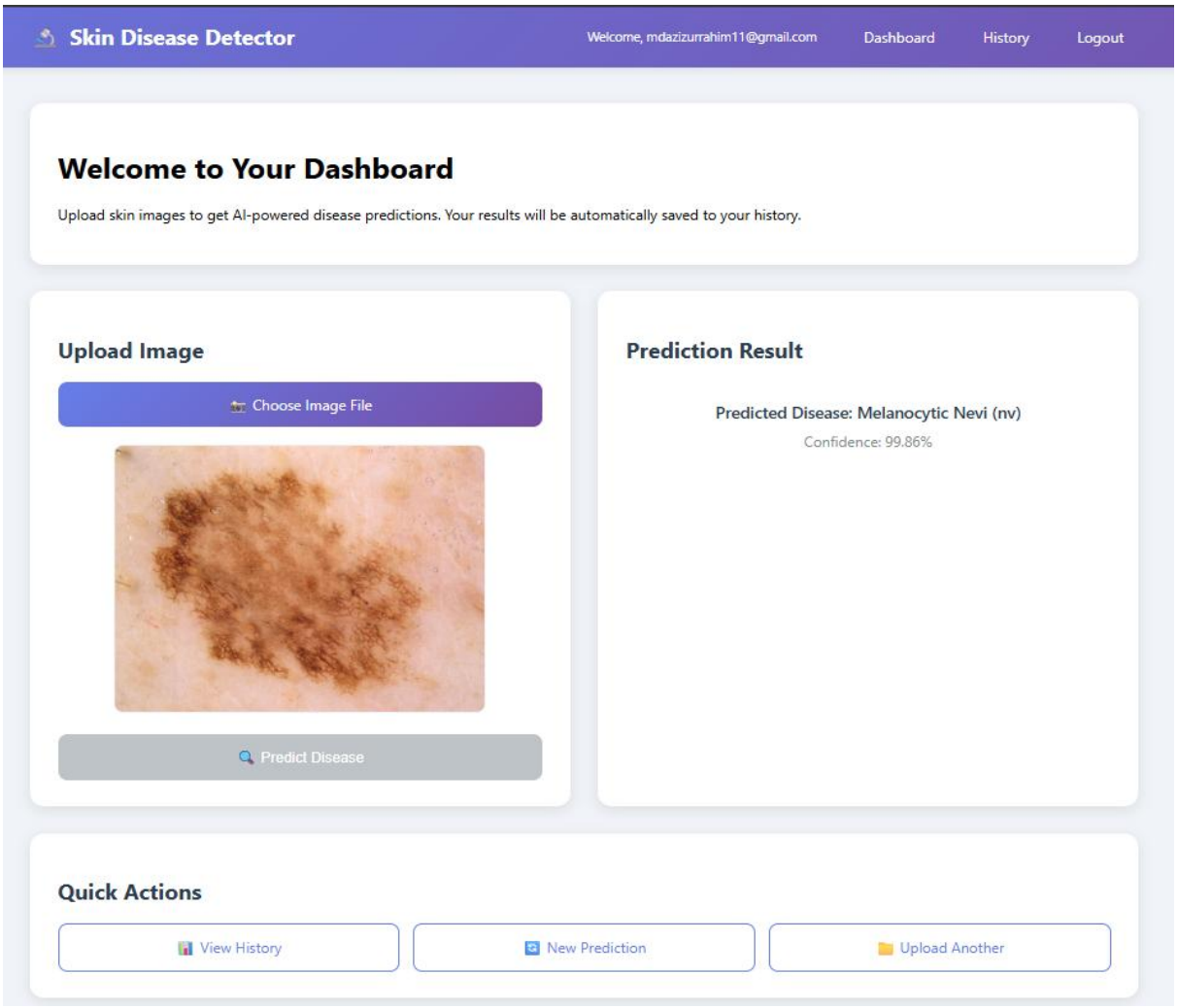


Figure 5.6: Prediction Result showing disease name and confidence.

5.4 Managing Data

5.4.1 Prediction History

The application automatically saves every test result.

1. Click **"History"** in the top menu or **"View History"** in Quick Actions.
2. This page shows statistics (Total Predictions, Average Confidence).
3. Below that, it lists all past predictions with their images and dates. Users can delete old records here.

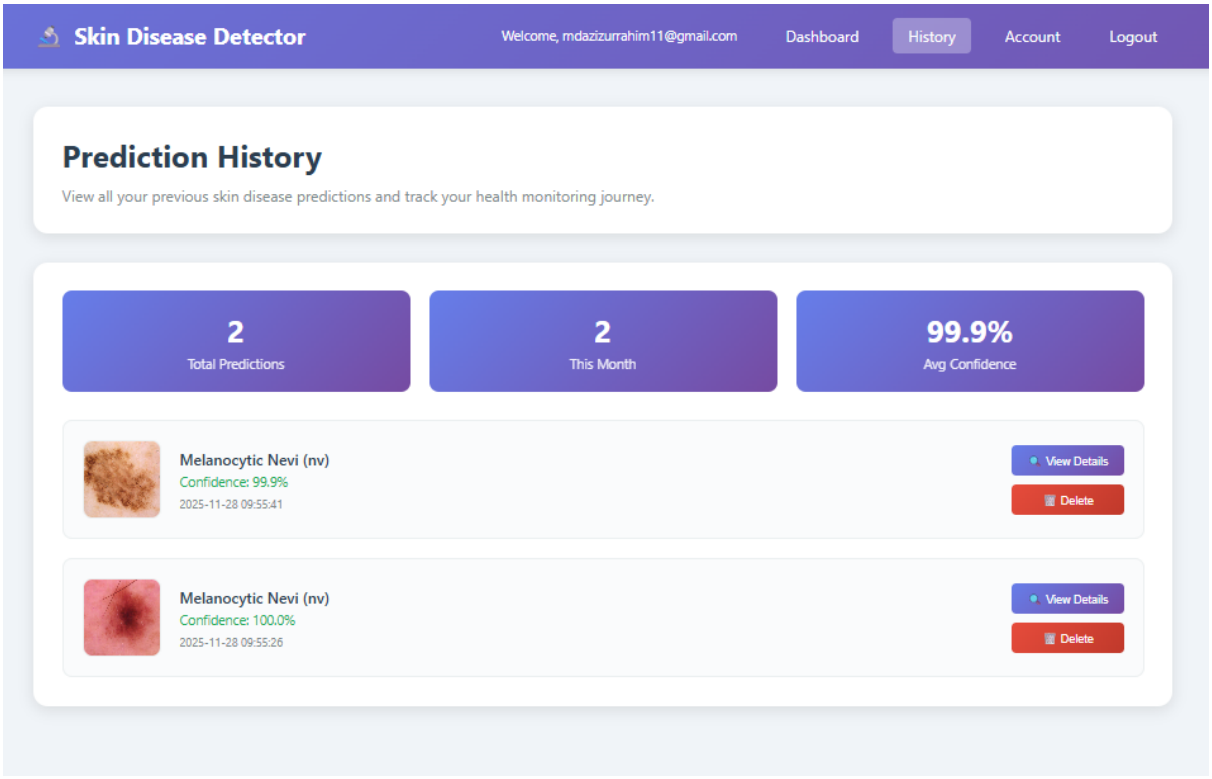


Figure 5.7: Prediction History Page.

5.4.2 Account Settings

Users can manage their profile by clicking the **"Account"** button in the top right corner.

- This page displays account information and usage statistics.
- It also includes a "Danger Zone" where users can permanently delete their account.

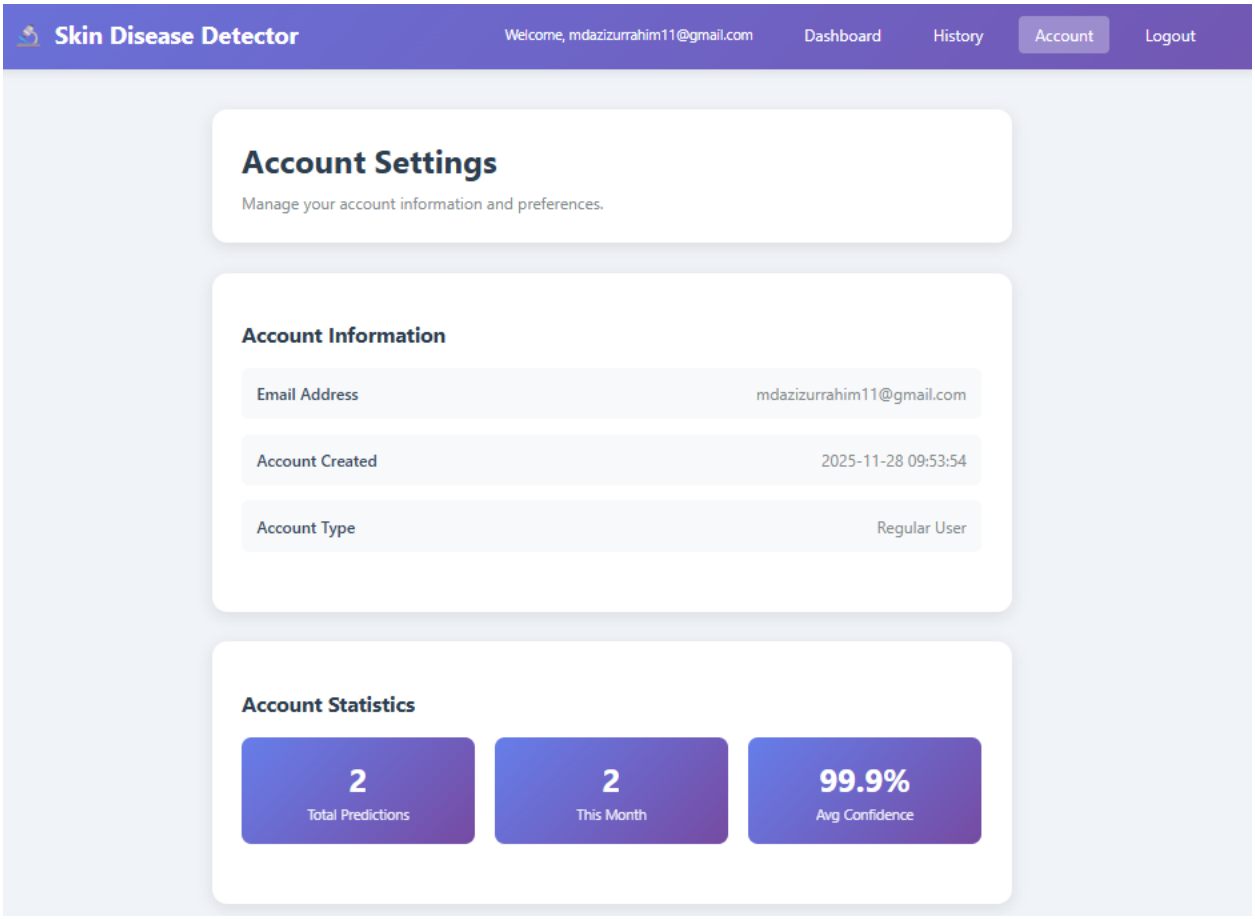


Figure 5.8: User Account Settings.

5.5 Summary

This chapter explained the complete workflow of the Skin Disease Detector. From creating an account to analyzing images and managing history, the system provides a smooth and secure experience for the user.

CHAPTER 6: PROJECT SUMMARY

6.1 Introduction

This final chapter synthesizes the outcomes of the **Skin Disease Prediction System** project. Throughout this documentation, we have explored the design, implementation, and testing of a Deep Learning solution aimed at assisting in the early detection of skin cancer. By leveraging the **EfficientNet-B3** architecture and **FastAPI**, we successfully created a web-based tool capable of classifying seven distinct types of skin lesions. This chapter summarizes the project's realized scope, identifies its inherent limitations, and proposes a roadmap for future enhancements.

6.2 Project Limitation

While the system operates successfully as a prototype, it is subject to several limitations that must be acknowledged:

1. **Dataset Limitations:** The model was trained on the **HAM10000** dataset. While extensive, this dataset is limited to 7 specific classes. The system cannot detect other common skin conditions (e.g., eczema, psoriasis, or acne) and will try to force them into one of the 7 known categories.
2. **Demographic Bias:** Most public dermatological datasets, including HAM10000, predominantly feature lighter skin tones. The model’s accuracy may decrease when analyzing lesions on darker skin tones due to a lack of diverse representation in the training data.
3. **Dependency on Image Quality:** The system relies heavily on the quality of the user-uploaded image. Poor lighting, blurriness, or low resolution significantly reduces the confidence score and accuracy of the prediction.
4. **Not a Medical Device:** This system provides a statistical probability based on patterns. It is a decision-support tool, not a diagnostic device, and cannot replace the clinical judgment of a certified dermatologist.

6.3 Scope

The project successfully achieved the following objectives within the defined scope:

1. **Model Performance:** A Convolutional Neural Network (EfficientNet-B3) was trained and fine-tuned, achieving a satisfactory accuracy rate for the classification of pigmented skin lesions.
2. **Backend Architecture:** A robust, asynchronous API was developed using **FastAPI**, capable of handling concurrent image upload requests efficiently.
3. **User Interface:** A functional, user-friendly web interface was implemented using **Jinja2** templates, allowing non-technical users to interact with the model easily.
4. **Integration:** The end-to-end pipeline—from image upload to preprocessing, inference, and result display—was fully integrated and tested.

6.4 Future Work

To evolve this project from an academic prototype into a viable commercial or medical product, the following future enhancements are proposed:

1. **Mobile Application Development:** Converting the web interface into a native mobile app (using Flutter or React Native) to allow users to take photos and analyze them directly from their smartphones.
2. **Explainable AI (Grad-CAM):** Implementing Gradient-weighted Class Activation Mapping (Grad-CAM) to visualize a "heatmap" on the image. This would show users *where* the model is looking (e.g., the irregular border of a mole), increasing trust in the AI's decision.
3. **Expanded Dataset:** Retraining the model on a larger, more diverse dataset (such as the ISIC 2019 or 2020 challenge datasets) to include more skin diseases and a wider range of skin tones.
4. **Real-Time Video Analysis:** Upgrading the backend to process video streams, allowing doctors to scan a patient's skin in real-time rather than taking static photos.
5. **Doctor-in-the-Loop:** Adding a feature where ambiguous or low-confidence predictions are automatically flagged and sent to a dermatologist for manual review.

6.5 Conclusion

The **Skin Disease Prediction System** demonstrates the transformative potential of Artificial Intelligence in healthcare. By automating the preliminary screening of skin lesions, this tool addresses the critical need for faster, accessible diagnostic support.

Over the course of this project, we successfully navigated the challenges of Deep Learning model integration and web development. Although limitations regarding dataset diversity and legal diagnostic status exist, the core technical framework is solid. This project serves as a foundational step toward a future where AI and medical professionals work in tandem to improve patient outcomes and save lives through early detection.

REFERENCES

Below is the list of resources, datasets, and libraries used in the development of the Skin Disease Prediction System:

1. Dataset (HAM10000)

We obtained the dataset from Kaggle, which originates from the ISIC (International Skin Imaging Collaboration) archive.

- **Source:** Kaggle (Skin Cancer MNIST: HAM10000)
- **Link:** https://www.kaggle.com/datasets/kmader/skin-cancer-mnist-ham10000
- **Original Paper:** Tschandl, P., Rosendahl, C., & Kittler, H. (2018). *The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions*. Nature Scientific Data, 5, 180161.

2. Model Architecture (EfficientNet)

The core deep learning model used in this project is EfficientNet-B3.

- **Paper:** Tan, M., & Le, Q. V. (2019). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. International Conference on Machine Learning (ICML).
- **Link:** https://arxiv.org/abs/1905.11946

3. Backend Framework (FastAPI)

FastAPI was used to build the high-performance web server.

- **Official Documentation:** https://fastapi.tiangolo.com/
- **Citation:** Ramírez, S. (2018). *FastAPI: High performance, easy to learn, fast to code, ready for production*.

4. Deep Learning Library (PyTorch)

PyTorch was used for loading the model and performing inference.

- **Official Documentation:** https://pytorch.org/docs/stable/index.html
- **Citation:** Paszke, A., Gross, S., Massa, F., et al. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Advances in Neural Information Processing Systems (NeurIPS).

5. Frontend Templating (Jinja2)

Jinja2 was used to render the HTML templates for the user interface.

- **Official Documentation:** https://jinja.palletsprojects.com/

6. Related Medical Research

This research paper inspired the idea of using AI for skin cancer detection.

- **Paper:** Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., & Thrun, S. (2017). *Dermatologist-level classification of skin cancer with deep neural networks*. Nature, 542(7639), 115–118.
- **Link:** https://www.nature.com/articles/nature21056

APPENDICES

Appendix A.1: Backend Logic (main.py)

This section contains the core FastAPI implementation for handling image uploads and inference.

```
from fastapi import FastAPI, File, UploadFile, Request
from fastapi.templating import Jinja2Templates
from fastapi.staticfiles import StaticFiles
from model import load_model, predict_image
from PIL import Image
import io

# Initialize App
app = FastAPI()

# Mount static files (CSS/Images)
app.mount("/static", StaticFiles(directory="static"), name="static")
templates = Jinja2Templates(directory="templates")

# Load the trained EfficientNet-B3 Model
model = load_model("Efficientnetb3_model.pth")

@app.get("/")
async def home(request: Request):
    return templates.TemplateResponse("index.html", {"request": request})

@app.post("/predict")
```

```
async def predict(request: Request, file: UploadFile = File(...)):
    # Read the image file
    image_data = await file.read()
    image = Image.open(io.BytesIO(image_data))

    # Get prediction from model
    pred_class, confidence = predict_image(model, image)

    return templates.TemplateResponse("result.html", {
        "request": request,
        "prediction": pred_class,
        "confidence": f"{confidence:.2f}%"
    })
```

Appendix A.2: Model Architecture & Inference (model.py)

This section details the custom wrapper used to load the pre-trained EfficientNet-B3 architecture.

```
import torch
from torchvision import models, transforms
import torch.nn as nn

# The 7 classes from HAM10000 dataset
CLASS_NAMES = [
    "Actinic Keratoses (akiec)", "Basal Cell Carcinoma (bcc)",
    "Benign Keratosis (bkl)", "Dermatofibroma (df)",
    "Melanoma (mel)", "Melanocytic Nevi (nv)", "Vascular Lesions (vasc)"
]
```

```
def load_model(path):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    # Load Pre-trained EfficientNet-B3
    model = models.efficientnet_b3(pretrained=False)

    # Modify the last layer for 7 classes
    num_ftrs = model.classifier[1].in_features
    model.classifier[1] = nn.Linear(num_ftrs, 7)

    # Load trained weights
    model.load_state_dict(torch.load(path, map_location=device))
    model.to(device)
    model.eval()
    return model

def predict_image(model, image):
    # Preprocessing: Resize to 224x224 and Normalize
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])

    image_tensor = transform(image).unsqueeze(0) # Add batch dimension
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    image_tensor = image_tensor.to(device)

    with torch.no_grad():
        outputs = model(image_tensor)
        probabilities = torch.nn.functional.softmax(outputs, dim=1)
        confidence, predicted_idx = torch.max(probabilities, 1)
```

```
    return CLASS_NAMES[predicted_idx.item()], confidence.item() * 100
```

Appendix A.3: Database Management (database.py)

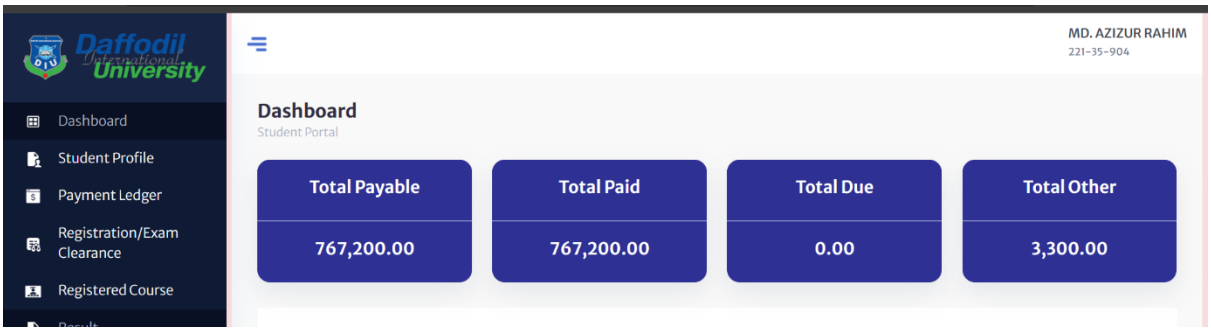
This code handles the connection to the SQLite database to store prediction history.

```
from sqlalchemy import create_engine, Column, Integer, String, Float, DateTime
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
import datetime

# Setup SQLite Database
DATABASE_URL = "sqlite:///./skin_disease_app.db"
engine = create_engine(DATABASE_URL, connect_args={"check_same_thread": False})
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

# Define the Table
class Prediction(Base):
    __tablename__ = "predictions"
    id = Column(Integer, primary_key=True, index=True)
    filsname = Column(String)
    disease_class = Column(String)
    confidence = Column(Float)
    created_at = Column(DateTime, default=datetime.datetime.utcnow)

# Create Tables
Base.metadata.create_all(bind=engine)
```



turnitin Page 2 of 39 - AI Writing Overview Submission ID trn:oid:::1:3451330376

*% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Disclaimer

Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (i.e., our AI models may produce either false positive results or false negative results), so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.

221-35-904

ORIGINALITY REPORT

7%	7%	0%	6%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Daffodil International University Student Paper	4%
2	dspace.daffodilvarsity.edu.bd:8080 Internet Source	2%
3	Submitted to NCC Education Student Paper	1%
4	ntnuopen.ntnu.no Internet Source	<1%
5	enterdimension.proboards30.com Internet Source	<1%
6	vdocument.in Internet Source	<1%
7	artifacts.opnfv.org Internet Source	<1%

Exclude quotes Off
Exclude matches Off
Exclude bibliography Off