HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION COMMUNICATION TECHNOLOGY

PROJECT II REPORT

SEEKWELL - An AI-Powered Web-based Platform for Early Skin Cancer Detection

Supervised by:

M.S. Nguyen Duy Hiep

Student	**ID**
Bach Nhat Minh	20225509

TABLE OF CONTENTS

Chapter 1: Project Introduction

1.1 Problem Statement

Vietnam is facing a growing public health challenge regarding skin cancer. Fsuch as climate change and a large population of outdoor workers contribute to an increasing incidence rate. A critical issue is that skin cancer is often detected at late stages, significantly reducing survival rates. This is compounded by a lack of access to specialized dermatological care, especially in rural areas. There is a clear need for a scalable, accessible, and affordable solution to facilitate early detection and connect patients with medical professionals efficiently.

1.2 Objectives and Scope

The primary objective of the **SeekWell** project is to develop a web-based platform that democratizes the early detection of skin cancer. It aims to empower community health workers (referred to as "Cadres"), patients, and doctors with modern technology.The scope of the completed project encompasses the following core functionalities:

Key Objectives:

- AI-Powered Screening: To provide an AI-driven tool that can analyze images of skin lesions and provide a preliminary risk assessment to assist healthcare workers.
- Bridging Care Gaps: To create a streamlined workflow for referring potential cases from community health workers to specialized doctors.
- Centralized Health Records: To manage patient information and skin lesion history in a secure and centralized system.
- Community and Support: To foster a community space where users can share experiences and find support.
- Efficient Consultation: To facilitate appointment booking and communication between patients and doctors.

Scope: The project encompasses the entire backend infrastructure for a multi-user platform with four distinct roles: **Patient**, **Doctor**, **Cadre**, and

Admin. The system's scope includes user authentication, profile management, AI-assisted lesion analysis, appointment scheduling, a community forum, and direct messaging capabilities.

1.3 Solution Approach

The solution is a robust web application built with a modern, decoupled architecture. A **FastAPI** backend provides a high-performance RESTful API, while a **React** frontend (to be developed) will offer a responsive and intuitive user interface.

The core of the solution is the integration of an AI service. Community Cadres can upload images of a patient's skin lesion through the app. The backend processes this image, sends it to a fine-tuned Vision Transformer (ViT) model hosted on Hugging Face Spaces for analysis, and returns a risk assessment. This data, along with the patient's case file, is then made available to doctors for review, diagnosis, and follow-up, creating an end-to-end chain of care.

1.4 Report Outline

Chapter 2 outlines the functional and non-functional requirements.

Chapter 3 describes the technology stack and design patterns.

Chapter 4 provides a deep dive into the system's architecture, database schema, backend API, and AI model development.

Chapter 5 concludes the report and suggests directions for future development.

Chapter 2: Requirements Analysis

2.1 Functional Overview

The SeekWell platform is designed to serve four main user roles with specific functionalities tailored to their needs.

- **Admin:** Manages the entire platform, including user accounts and system settings.
- **Cadre (Community Health Worker):** Acts as the frontline user, conducting initial screenings in the community.
- **Doctor:** A registered medical professional who reviews cases, provides diagnoses, and consults with patients.
- **Patient:** The end-user whose health is being monitored.

2.2 System Actors and Use Cases

Cadre Use Cases:

- Register and log into the system.
- Create and manage patient profiles under their care.
- Initiate a new skin lesion case by uploading an image through the AISkinAnalysisDashboard.
- Receive a preliminary analysis from the integrated AI service.
- View the history of lesions for their patients in the AnalysisHistory tab.
- Refer a patient's case to a registered doctor for further review.

Patient Use Cases:

- Register and log into the system.
- View their personal profile and medical history, including all lesion analyses.
- Book appointments with doctors, with an urgent pathway for high-risk AI results.
- Reset their password.

Doctor Use Cases:

- Register and log into the system.
- View a dashboard of patients referred to them.

- Review patient lesion images, AI analysis results (AnalysisResults), and history.
- Submit a formal diagnosis for a lesion.
- Manage their appointment schedule.

Admin Use Cases:

- Manage all user accounts (activate, deactivate, assign roles).
- Oversee all data within the platform.

Chapter 3: Technologies Used

3.1 Backend: FastAPI

3.2 Frontend: React with TypeScript

3.3 Database

System: PostgreSQL
ORM: SQLAlchemy

3.4 AI Development & Deployment

Core Library: PyTorch
High-Level Framework: Hugging Face Transformers
Deployment: Gradio & Hugging Face Spaces

3.5 Key Frameworks and Libraries

Pydantic: For data validation in FastAPI.
Passlib & python-jose: For authentication (password hashing, JWT).
Hugging Face transformers & datasets: For fine-tuning the AI model and data preprocessing.
scikit-learn: For computing evaluation metrics (balanced_accuracy) and class weights.
Gradio: For creating a simple web UI and API endpoint for the deployed AI model.

3.6 Deployment

Frontend Hosting: Vercel - A platform for frontend frameworks and static sites, chosen for its seamless Git integration, automatic deployments, and global CDN.

Backend & Database Hosting: Render - A unified cloud platform used for deploying the FastAPI backend and the PostgreSQL database instance, simplifying infrastructure management.

Chapter 4: Application Development and Deployment

4.1 Database Design and Schema

The database schema is defined in models.py using SQLAlchemy's declarative base.

Table: users The central table for authentication and role management.

Column	Type	Constraints	Description
id	Integer	Primary Key	Unique identifier for the user.
email	String	Unique, Indexed	The user's login email.
hashed_password	String	Not Null	The securely hashed password.
is_active	Boolean	Default: True	Whether the user account is active.
role	Enum	Not Null	User role ('patient', 'doctor', 'cadre', 'admin').
patient, doctor, cadre	Relationship	One-to-One	Links to the respective profile tables.

Profile Tables: patients, doctors, cadres These tables store role-specific information.

Table	Column	Type	Constraints	Description
patients	id	Integer	Primary Key	Unique patient ID.
	user_id	Integer	Foreign Key (users.id)	Links to the user account.
	full_name	String		Patient's full name.
doctors	id	Integer	Primary Key	Unique doctor ID.

	user_id	Integer	Foreign Key (users.id)	Links to the user account.
	full_name	String		Doctor's full name.
cadres	id	Integer	Primary Key	Unique cadre ID.
	user_id	Integer	Foreign Key (users.id)	Links to the user account.
	full_name	String		Cadre's full name.

Core Functional Tables: skin_lesions, appointments

Table	Column	Type	Constraints	Description
skin_lesions	id	Integer	Primary Key	Unique lesion record ID.
	patient_id	Integer	Foreign Key (patients.id)	The patient this lesion belongs to.
	cadre_id	Integer	Foreign Key (cadres.id)	The cadre who submitted the lesion.
	image_url	String	Not Null	URL of the stored lesion image.
	ai_prediction	JSON		Raw JSON response from the AI model.
	doctor_diagnosis	String		Final diagnosis provided by a doctor.

appointments	id	Integer	Primary Key	Unique appointment ID.
	patient_id	Integer	Foreign Key (patients.id)	The patient for the appointment.
	doctor_id	Integer	Foreign Key (doctors.id)	The doctor for the appointment.
	appointment_time	DateTime	Not Null	The scheduled date and time.
	status	String		e.g., 'scheduled', 'completed', 'cancelled'.

4.2 Backend API Implementation

The RESTful API is the core of the backend, with endpoints defined in the routers directory. All data transfer objects are validated using Pydantic schemas defined in schemas.py.

Authentication (routers/auth.py)

- **POST /token**
    - **Description:** Authenticates a user with email and password and returns a JWT access token.
    - **Request Body:** application/x-www-form-urlencoded with username (email) and password via OAuth2PasswordRequestForm.
    - **Response:** schemas.Token containing access_token and token_type.

User Management (routers/users.py, routers/password.py)

- **POST /users/**
  - **Description:** Creates a new user (patient, doctor, or cadre). The role is specified in the request body.
  - **Request Body:** schemas.UserCreate.
  - **Response:** schemas.User.
- **POST /password-recovery/{email}**
  - **Description:** Initiates a password recovery process (logic for sending email not shown, but endpoint exists).
  - **Response:** Success message.

Core Medical Workflow (routers/skin_lesions.py, routers/appointments.py)

- **POST /lesions/**
  - **Description:** Creates a new skin lesion record. This is a critical endpoint used by Cadres.
  - **Authentication:** Requires an active, authenticated user (Depends(get_current_active_user)).
  - **Request Body:** schemas.SkinLesionCreate.
  - **Process:** The endpoint calls the lesion_service to orchestrate the AI analysis before saving the record.
  - **Response:** schemas.SkinLesion.
- **GET /lesions/patient/{patient_id}**
  - **Description:** Retrieves all lesion records for a specific patient.
  - **Authentication:** Required.
  - **Response:** List[schemas.SkinLesion].
- **POST /appointments/**
  - **Description:** Creates a new appointment between a patient and a doctor.
  - **Authentication:** Required.
  - **Request Body:** schemas.AppointmentCreate.
  - **Response:** schemas.Appointment.

4.3 Core Service Implementation

The services directory contains business logic that is too complex for a simple CRUD operation.

- **services/lesion_service.py:** This service acts as an orchestrator. When the /lesions/ endpoint is called, this service is invoked. It takes the uploaded image data, calls the ai_integration service to get a prediction, and then combines this prediction with the other lesion data before passing it to crud.py to be saved in the database.
- **services/ai_integration.py:** This module is responsible for communicating with the external AI model hosted on Hugging Face Spaces. It contains a function get_ai_prediction(image) which constructs and sends an HTTP POST request to the Gradio API endpoint, passing the image data. It then parses the JSON response from the AI service and returns it to the lesion_service.

4.4 AI Model Development and Integration

A critical component of SeekWell is the AI model that provides the initial risk assessment. The development and integration followed a structured process based on the principle of **transfer learning.**

4.4.1 Literature Review and Technology Selection

Base Model: The project started with Anwarkh1/Skin_Cancer-Image_Classification, a publicly available model on the Hugging Face Hub. This model is a **Vision Transformer (ViT),** an architecture that applies the successful Transformer model from natural language processing to computer vision tasks. It has been pre-trained on the **HAM10000 ("Human Against Machine with 10000 training images") dataset,** a large, well-known collection of multi-source dermoscopic images of common pigmented skin lesions. The use of a model pre-trained on such a comprehensive medical imaging dataset provides a strong foundation of learned features relevant to skin lesion analysis.

Fine-Tuning Dataset: While the base model is powerful, it was trained on specialized dermoscopic images. To adapt the model for SeekWell's specific use case—analyzing clinical photos taken with standard smartphones—it was fine-tuned on the **PAD-UFES-20** dataset (Pacheco et al.). This benchmark dataset is uniquely suited for the project's goals, as it was created specifically to address the lack of public archives of clinical skin lesion images. It consists of 2,298 images from 1,373 patients, covering six diagnostic classes (three skin cancers and three other skin diseases). Crucially, all images were captured with smartphone cameras, making the data highly representative of the inputs expected from SeekWell users. This alignment between the fine-tuning data and the real-world application is key to improving the model's practical performance and robustness.

4.4.2 Model Fine-Tuning Process

The fine-tuning process was carefully managed to adapt the pre-trained model to the new dataset and task.

1. **Data Preprocessing:** The Hugging Face Datasets library was used to load the PAD-UFES-20 data. A ViTImageProcessor was configured to resize, normalize, and transform the images into the pixel value tensors required by the Vision Transformer model.
2. **Handling Class Imbalance:** The PAD-UFES-20 dataset, like many medical datasets, is imbalanced, with more examples of common conditions than rare ones. To prevent the model from becoming biased towards the majority classes, a weighted loss strategy was employed. Class weights were calculated using scikit-learn's compute_class_weight function with the 'balanced' setting. A custom CustomTrainer class was implemented in PyTorch, inheriting from the Hugging Face Trainer, to override the default loss computation. This custom trainer applies a CrossEntropyLoss function initialized with the calculated class weights, ensuring that misclassifications of minority class samples incur a higher penalty during training.
3. **Training:** The model was fine-tuned for 15 epochs using the AdamW optimizer, a cosine learning rate scheduler with a warmup ratio of 0.1, a learning rate of 2e-5, and a batch size of 32. The model's performance was evaluated on a held-out test set after each epoch, with the best model being saved based on the balanced_accuracy metric, which is more informative than standard accuracy for imbalanced datasets.

4.4.3 Performance and Evaluation

After 15 epochs of training, the best-performing model was evaluated on the test set. The final performance metrics are as follows:

Metric	Value
Evaluation Loss	0.919

Accuracy	69.8%
Balanced Accuracy	**67.9%**
F1-Score (Weighted)	71.1%
Precision (Weighted)	74.3%
Recall (Weighted)	69.8%

Interpretation: The **Balanced Accuracy of 67.9%** is the most critical metric here. It indicates that the model performs significantly better than random chance (which would be around 16.7% for six classes) and is not merely predicting the most common class. The weighted F1-score of 71.1% further supports that the model maintains a reasonable balance between precision (the accuracy of positive predictions) and recall (the ability to identify all relevant instances).

While these numbers may not be high enough for autonomous diagnosis, they are well-suited for the application's purpose: **a preliminary screening and triage tool.** The model is effective at identifying suspicious lesions that warrant review by a human doctor, thereby fulfilling its role of bridging the care gap and flagging high-risk cases for early intervention.

4.5 Implementation

4.5.1 Libraries and Tools Used

Backend (requirements.txt):

Purpose	**Tool Name/Version**
Web Framework	FastAPI (version 0.115.12)
ORM	SQLAlchemy (version 2.0.41)
Database Driver (PostgreSQL)	psycopg2-binary (version 2.9.10)

Data Validation	Pydantic (version 2.11.5)
ASGI Server	Uvicorn (version 0.34.2)
Password Hashing	passlib, bcrypt
JWT Handling	python-jose
Environment Variables	python-dotenv
AI SDK	google-generativeai(version 0.8.5)

Frontend (frontend/package.json):

Purpose	**Tool Name/Version**
UI Library	React (version 19.1.0)
HTTP Client	Axios (version 1.9.0)
Routing	react-router-dom (version 7.6.0)
Language	TypeScript (version 4.9.5)
Build/Dev Tool	react-scripts (version 5.0.1)
Notification Library	react-toastify (version 11.0.5)
Markdown Rendering	react-markdown (version 10.1.0)
UI Component Library	Material-UI (MUI)

AI Development & Deployment:

Purpose	Tool Name/Version
Core ML Framework	PyTorch
High-Level Toolkit	Hugging Face (transformers, datasets)
Model Evaluation	scikit-learn
Deployment	Gradio & Hugging Face Spaces

Development Tools:

- **IDE:** Visual Studio Code (assumed common choice).
- **Version Control:** Git & GitHub (repository bnmbanhmi/seekwell).
- **Training model: Kaggle**

Database Management: PostgreSQL server, pgAdmin or DataGrip.

4.6 Deployment

SeekWell has been successfully deployed, with the frontend hosted on Vercel and the backend on Render, utilizing a Google Cloud SQL (later switched to Render) for PostgreSQL instance for data persistence. Both services are configured for auto-deployment from the GitHub repository.

Application Structure:

- **Frontend:** React application located in the frontend directory.
  Live URL: seekwell.vercel.app

- **Backend:** FastAPI (Python) application located in the backend directory.

  Live URL: seekwell-backend.onrender.com
- **Database:** Google Cloud SQL for PostgreSQL, but later migrated to Render

4.6.1 Frontend Deployment (React Application on Vercel)

The React-based frontend application is deployed and hosted on Vercel, leveraging its seamless integration with GitHub for continuous deployment.

1. **Vercel Project Setup:**
   A Vercel project was created and connected to the GitHub repository: https://github.com/bnmbanhmi/seekwell. This connection enables automatic builds and deployments upon code changes to the specified branch.
2. **Project Configuration (via Vercel UI):**
   - **Framework Preset:** Vercel auto-detected the project as a "Create React App."
   - **Root Directory:** Configured to frontend, ensuring Vercel uses the correct directory for build and deployment processes.
   - **Build Command & Output Directory:** Standard Create React App settings were used (npm run build command, and frontend/build as the output directory).
3. **Environment Variables (Frontend Project on Vercel UI):**
   REACT_APP_BACKEND_URL: Set to the live backend URL (seekwell-backend.onrender.com) to enable communication between the frontend and backend services.
4. **Deployment & URL:**
   Vercel automatically deploys changes pushed to the main branch (or the designated production branch) of the connected GitHub repository.
5. **Live Frontend URL:** The deployed frontend is accessible at seekwell.vercel.app.

4.6.2 Backend Deployment (FastAPI Application on Render)

The FastAPI backend application is deployed as a web service on Render, which also supports continuous deployment from GitHub.

1. **Render Project Setup:**
   A "Web Service" was created on Render and linked to the same GitHub repository: github.com/bnmbanhmi/seekwell.
2. **Service Configuration (via Render UI):**
   - **Root Directory:** Set to backend to specify the location of the backend application code.
   - **Runtime:** Python, automatically detected by Render due to the presence of backend/requirements.txt.
   - **Build Command:** pip install -r requirements.txt (executed relative to the backend directory) to install all necessary Python dependencies.
   - **Start Command:** uvicorn app.main:app --host 0.0.0.0 --port $PORT --workers 1 to run the FastAPI application using Uvicorn, making it accessible on the port assigned by Render.
3. **Backend Dependencies (backend/requirements.txt):**
   A comprehensive requirements.txt file located at backend/requirements.txt lists all Python dependencies for the backend, critically including psycopg2-binary for PostgreSQL database connectivity.
4. **Environment Variables (Backend Service on Render UI):**
   - DATABASE_URL: The complete PostgreSQL connection string for the Google Cloud SQL database. Format: postgresql://db_user:db_password@34.67.156.97:5432/db_name.
   - SECRET_KEY, ALGORITHM, ACCESS_TOKEN_EXPIRE_MINUTES: Variables critical for JWT authentication and security.

- FRONTEND_URL: Set to the live frontend URL (seekwell.vercel.app) for CORS configuration and potentially other backend-to-frontend communications.
- MAIL_USERNAME, MAIL_PASSWORD, MAIL_FROM, etc.: Configuration for email services (if implemented).
- GEMINI_API_KEY: API key for accessing the Google Gemini service for the chatbot functionality.
- PYTHON_VERSION: Specified if a particular Python version is necessary for the application.

5. **Deployment & URL:**
   Render automatically deploys new versions of the backend when changes are pushed to the main branch of the GitHub repository.
6. **Live Backend URL:** The deployed backend API is accessible at seekwell-backend.onrender.com.

4.6.3 Database Deployment (Google Cloud SQL, later switched to Render, for PostgreSQL)

The application relies on a Google Cloud SQL (later switched to Render) for PostgreSQL instance for persistent data storage.

Instance Configuration:

- **Public IP Address:** 34.67.156.97 (used in the DATABASE_URL for the backend).
- **Authorized Networks:** Currently configured to allow access from any IP address (0.0.0.0/0) for ease of development and connection from Render. *For enhanced security in a production environment, this should be restricted to Render's outbound IP addresses or specific necessary ranges.*
- **Hardware:** 1 vCPU, 614.4 MB Memory, 10 GB HDD Storage (Auto storage increase enabled).
- **Network Configuration:** The Render backend service connects to this Cloud SQL instance using its public IP address.

4.6.4 Post-Deployment Configuration & Connection

1. **Frontend to Backend Link:** The REACT_APP_BACKEND_URL environment variable on Vercel correctly points the frontend to the live backend URL (seekwell-backend.onrender.com).
2. **Backend to Frontend Link:** The FRONTEND_URL environment variable on Render ensures the backend is aware of the frontend's live URL (seekwell.vercel.app), primarily for Cross-Origin Resource Sharing (CORS) configuration.
3. **CORS Configuration:** The FastAPI application's CORS middleware is configured to allow requests from the frontend. For development, the origins list was set to allow all origins. *In a production setting, this list should be restricted to the specific frontend URL (seekwell.vercel.app) for security.*

4.6.5 Auto-Updates from GitHub (CI/CD)

Both Vercel (monitoring the frontend directory) and Render (monitoring the backend directory) are configured for Continuous Integration/Continuous Deployment (CI/CD).

Any push to the main branch of the https://github.com/bnmbanhmi/seekwell repository that includes changes within these respective directories will automatically trigger new builds and deployments on Vercel and Render. This ensures that the live applications are always up-to-date with the latest stable code.

Chapter 5: Conclusion and Future Development

5.1 Conclusion

This project has successfully culminated in the delivery of the SeekWell platform, a modern, fully-functional web application designed to address the critical issue of late-stage skin cancer detection in Vietnam. The system effectively digitizes and streamlines the screening process by empowering a network of Patients, Doctors, and community-based Cadres with a suite of powerful tools.

The chosen client-server architecture, featuring a React frontend and a FastAPI backend, has proven highly effective. This structure ensures a clear separation of concerns, allows for independent scalability of components, and promotes a maintainable codebase. The system's robustness and reliability are further enhanced by the utilization of technologies such as SQLAlchemy for object-relational mapping, Pydantic for data validation, and JSON Web Tokens (JWT) for secure authentication.

A significant innovation of this project is the successful integration and fine-tuning of a Vision Transformer (ViT) model for preliminary skin lesion analysis. This AI-powered screening tool serves as a vital decision-support mechanism, enabling Cadres to identify high-risk cases for prioritised medical review. The application successfully implements role-based access control, guaranteeing that all user types have appropriate and secure access to system features and data.

The system has been successfully deployed and is fully operational. The frontend React application is hosted on Vercel, the backend FastAPI application and PostgreSQL database are deployed on Render, and the AI model is served via Hugging Face Spaces. This distributed deployment strategy ensures high availability and performance. Ultimately, the project has achieved its primary objectives of creating an accessible screening tool, bridging the gap between communities and specialized care, and establishing a robust platform for future public health initiatives.

5.2 Future Work

While the current system fulfills its core objectives, the platform is well-positioned for future enhancements and expansions. The future development roadmap is divided into distinct phases to ensure sustainable growth and impact.

Phase 1: Expansion and Enhancement (Near-Term)

- **Launch Mobile Application:** Develop and release native mobile applications for both iOS and Android. This is a top priority to enhance accessibility and usability for Cadres working in the field and to provide patients with on-the-go access to their health records and communication tools.
- **AI Model V2 and Continuous Learning:** Initiate a continuous learning pipeline for the AI model. By leveraging new, verified data collected through the platform, the model will be regularly re-trained to improve its accuracy, reduce bias, and potentially expand its capabilities to identify a wider range of dermatological conditions.
- **Partnership and Network Expansion:** Actively work to onboard more dermatologists, clinics, and hospitals onto the SeekWell platform. The goal is to expand the network of care providers across Vietnam, reducing wait times and improving access for patients in remote areas.

Phase 2: Deepening Integration and Service Offering (Mid-Term)

- **Telemedicine Integration:** Integrate real-time, secure video conferencing capabilities directly into the platform. This will enable remote consultations between doctors and patients, which is particularly valuable for follow-up appointments and for patients who cannot easily travel.
- **Advanced Data Analytics Dashboard:** Develop a comprehensive analytics dashboard for public health officials and partner organizations. This tool will provide aggregated, anonymized data on skin cancer prevalence, screening rates, and regional trends, enabling data-driven public health planning and resource allocation.

Phase 3: National Ecosystem Integration (Long-Term)

- **Electronic Health Record (EHR) Integration:** Explore and develop pathways for integrating SeekWell with Vietnam's national and hospital-level Electronic Health Record systems. This would ensure seamless data flow, reduce redundant data entry, and provide doctors with a more holistic view of a patient's health history.

- **Government and NGO Collaboration:** Solidify partnerships with the Ministry of Health and non-governmental organizations to integrate SeekWell into broader national health screening programs, maximizing its reach and impact on public health outcomes.

The successful completion of this project provides a strong and scalable foundation for these future enhancements, ensuring the SeekWell platform can continue to evolve and meet the growing needs of the community it serves.

References

Anwarkh1. (2024). *Skin Cancer Image Classification Model*. Hugging Face. Retrieved from https://huggingface.co/Anwarkh1/Skin_Cancer-Image_Classification

Pacheco, A. G. C., Lima, G. R., Salomão, A. S., Krohling, B., Biral, I. P., de Angelo, G. G., ... & de Barros, L. F. S. (2020). *PAD-UFES-20: a skin lesion dataset composed of patient data and clinical images collected from smartphones* (Version 1) [Data set]. Mendeley Data. https://doi.org/10.17632/zr7vgbcyr2.1